Oracle® Database

Security Guide

18c

E83683-02

May 2018

Oracle Database Security Guide, 18c

E83683-02

Primary Author: Patricia Huey

Contributing Authors: Sumit Jeloka

Contributors: Suraj Adhikari, Thomas Baby, Tammy Bednar, Todd Bottger, Sanjay Bharadwaj, Leo Cloutier, Sudha Duraiswamy, Naveen Gopal, Rishabh Gupta, Yong Hu, Srinidhi Kayoor , Peter Knaggs, Andre Kruklikov, Sanjay Kulhari, Anup A. Kumar, Bryn Llewellyn, Dah-Yoh Lim, Rahil Mir, Hari Mohankumar, Gopal Mulagund, Abhishek Munnolimath, Paul Needham, Robert Pang, Dilip Raj, Kumar Rajamani, Kathy Rich, Saikat Saha, Vipin Samar, Saravana Soundararajan, James Spiller, Srividya Tata, Kamal Tbeileh, Can Tuzla, Anand Verma, Patrick Wheeler, Peter H. Wong

Contents

20 Configuring Secure Sockets Layer Authentication

F Managing Public Key Infrastructure (PKI) Elements

List of Tables

Preface

Welcome to *Oracle Database Security Guide*. This guide describes how you can configure security for Oracle Database by using the default database features.

Audience

Oracle Database Security Guide is intended for database administrators (DBAs), security administrators, application developers, and others tasked with performing the following operations securely and efficiently.

It covers these areas:

- Designing and implementing security policies to protect the data of an organization, users, and applications from accidental, inappropriate, or unauthorized actions
- Creating and enforcing policies and practices of auditing and accountability for inappropriate or unauthorized actions
- Creating, maintaining, and terminating user accounts, passwords, roles, and privileges
- Developing applications that provide desired services securely in a variety of computational models, leveraging database and directory services to maximize both efficiency and ease of use

To use this document, you need a basic understanding of how and why a database is used, and basic familiarity with SQL.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more security-related information, see these Oracle resources:

- *Oracle Database Administrator's Guide*

- *Oracle Database 2 Day DBA*
- *Oracle Database 2 Day + Security Guide*
- *Oracle Database Concepts*
- *Oracle Database Reference*
- *Oracle Multitenant Administrator's Guide*

Many of the examples in this guide use the sample schemas of the seed PDB, which you can create when you install Oracle Database. See *Oracle Database Sample Schemas* for information about how these schemas were created and how you can use them yourself.

Oracle Technology Network (OTN)

You can download free release notes, installation documentation, updated versions of this guide, white papers, or other collateral from the Oracle Technology Network (OTN). Visit

`http://www.oracle.com/technetwork/index.html`

For security-specific information on OTN, visit

`http://www.oracle.com/technetwork/topics/security/whatsnew/index.html`

For the latest version of the Oracle documentation, including this guide, visit

`http://www.oracle.com/technetwork/documentation/index.html`

My Oracle Support

You can find information about security patches, certifications, and the support knowledge base by visiting My Oracle Support (formerly Oracle*MetaLink*) at

`https://support.oracle.com`

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Database Security Guide

This preface contains:

Changes in Oracle Database Security 18c

Oracle Database Security Guide for Oracle Database 18c has new security features.

Ability to Create Schema Only Accounts

You now can create schema only accounts, for object ownership without allowing clients to log in to the schema.

A user (or other client) cannot log in to the database schema unless the account is modified to accept an authentication method. However, this type of schema user can proxy in a single session proxy.

Integration of Active Directory Services with Oracle Database

Starting with this release, you can authenticate and authorize users directly with Microsoft Active Directory.

With centrally managed users (CMU) Oracle database users and roles can map directly to Active Directory users and groups without using Oracle Enterprise User Security (EUS) or another intermediate directory service. EUS is not being replaced or deprecated; this new feature is another simpler option if you only want to authenticate and authorize users with Active Directory. Centrally managed users is designed to be extended to work with other LDAP version 3–compliant directory services, but Microsoft Active Directory is the only service that is supported in this release.

The direct integration with directory services supports better security through simpler configuration with the enterprise identity management architecture. In the past, users may have avoided the security practice of integrating the database with directory services due to the difficulty and complexity. With the direct integration, you can improve your security posture by more easily integrating the Database to the enterprise directory service.

Ability to Encrypt Sensitive Credential Data in the Data Dictionary

Starting with this release, you can encrypt sensitive credential data that is stored in the data dictionary `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` system tables.

In previous releases, and by default in this release, the data in these tables is obfuscated. However, because of the rise of de-obfuscation algorithms that are available on the Internet, it is important to use a more secure solution to protect this type of sensitive data. You can manually encrypt this data by using the `ALTER DATABASE DICTIONARY` SQL statement.

PDB Lockdown Profile Enhancements

This release introduces several enhancements for PDB lockdown profiles.

These enhancements are as follows:

- You now can create PDB lockdown profiles in the application root, as well as in the CDB root. In previous releases, you only could create the profile in the CDB root. The ability to create a PDB lockdown profile in an application container enables you to more finely control access to the applications that are associated with the application container.
- You now can create a PDB lockdown profile that is based on another PDB lockdown profile, either a static base profile or a dynamic base profile. You can control whether subsequent changes to the base profile are reflected in the newly created profile that uses the base profile.
- Three default PDB lockown profiles have been added for this release: `PRIVATE_DBAAS`, `SAAS`, and `PUBLIC_DBAAS`. These profiles benefit Cloud environments.
- A new dynamic data dictionary view, `V$LOCKDOWN_RULES`, is available. This view enables the local user to see the lockdown rules that are applicable in the PDB.

This feature benefits environments that need enforced security and isolation in PDB provisioning.

New Authentication and Certification Parameters

This release introduces four new parameters that can be used to strengthen security on the database.

The new parameters are as follows:

- The `ADD_SSLV3_TO_DEFAULT` `sqlnet.ora` parameter controls the use of the Secure Sockets Layer version 3, which can be vulnerable to Padding Oracle On Downgraded Legacy Encryption (POODLE) attacks
- The `ADG_ACCOUNT_INFO_TRACKING` initialization parameter controls login attempts on Oracle Data Guard standby databases by enabling you to maintain a single global copy of user account information across all Data Guard primary and standby databases.

  The `ACCEPT_MD5_CERTS` `sqlnet.ora` parameter enables or disables the MD5 algorithm.
- The `ACCEPT_SHA1_CERTS` `sqlnet.ora` parameter enables or disables the SHA-1 algorithm.

Ability to Write Unified Audit Trail Records to SYSLOG or the Windows Event Viewer

Starting with this release you can write unified audit trail records to SYSLOG on UNIX or the Windows Event Viewer on Microsoft Windows.

On Microsoft Windows, you can enable or disable this behavior. On UNIX systems, you can specify the SYSLOG facility to use and the type logging category for the unified audit record, such as whether it is an alert or for an emergency. To configure this behavior, you can set the `UNIFIED_AUDIT_SYSTEMLOG` initialization parameter.

Ability to Use Oracle Data Pump to Export and Import the Unified Audit Trail

Starting with this release, you can include the unified audit trail in either full or partial export and import operations using Oracle Data Pump.

There is no change to the user interface. When you perform the export or import operation of a database, the unified audit trail is automatically included in the Data Pump dump files.

This feature benefits users who, as in previous releases, must create dump files of audit records.

Changes in Oracle Database Security 12c Release 2 (12.2)

Oracle Database Security Guide for Oracle Database 12c release 2 (12.2) has both new and deprecated security features.

New Features

The following features are new for this release:

Ability to Create Application Common Objects, Users, Roles, and Profiles

You now can create application common objects (metadata-linked and object-linked), users, roles, and profiles.

These common objects reside in the application root. This guide explains how to manage security for application common users, roles, and profiles.

See Also:

- About Common Users
- About Profiles
- Managing Common Roles and Local Roles
- Sharing Application Common Objects
- *Oracle Database Administrator's Guide*

Application Container Security Features

Application containers affect several default security features, such as privilege grants to application common users, how application contexts are used, and so on.

The following functionality is affected:

- **Application common users:** In addition to other privileges, you can grant the SYSDBA and SYSOPER administrative privileges to application common users.
- **Application contexts:** When you create an application in the application root, you can create an application context for use with this application. This application context resides in the application root.
- **Oracle Virtual Private Database:** You can create Virtual Private Database policies that protect application common objects. These policies reside in the application root.
- **Transparent Sensitive Data Protection:** You can apply Transparent Sensitive Data Protection (TSDP) policies to the current pluggable database (PDB) or to the current application PDB only.
- **Transport Layer Security:** If you are using Secure Sockets Layer (SSL) and plan to use the upgraded version of SSL, which is Transport Layer Security (TLS), then you must ensure that each PDB is able to use its own wallet with its own certifications for TLS authentication.
- **Auditing:** In a multitenant environment for application containers, traditional, unified auditing, and fine-grained auditing are all affected by the use of application containers.

Addition of SYSRAC Administrative Privilege for Oracle Real Application Clusters

Starting with this release, the Oracle Real Applications (Oracle RAC) clusterware agent that manages Oracle RAC uses the SYSRAC administrative privilege.

Administrative User Authentication Enhancements

Administrative user account authentication now has enhancements for password files, password profile parameters, Secure Sockets Layer (SSL), Kerberos, and multitenant environments.

These enhancements are as follows:

- You can create password files for external users who have been granted the SYSASM, SYSBACKUP, SYSDG, and SYSKM administrative privileges in addition to the SYSDBA and SYSOPER administrative privileges. This enhancement is available in a multitenant environment, for both local and common external administrative users, and for external users in an SSL or Kerberos authentication configuration.
- The SYSDG administrative privilege can be used in a sharding environment.
- You can use password profile parameters, such as PASSWORD_LIFE_TIME, for administrative user authentication.

Enhancements for the Management of Administrative Passwords

This release introduces better security for the management of administrative passwords, by enforcing the associated administrative user's profile password limits.

Examples of these profile limits are FAILED_LOGIN_COUNT, PASSWORD_LOCK_TIME, PASSWORD_GRACE_TIME, and PASSWORD_LIFE_TIME.

You now can create a password file that can apply to both administrative users and non-administrative users. When you create a password profile using the `PASSWORD_VERIFY_FUNCTION` clause of the `CREATE PROFILE` statement, this setting now applies to administrative users as well as non-administrative users, as long as you create the password file with the `ORAPWD` utility `FORMAT` parameter set to `12.2`.

In addition, for the `ORAPWD` utility, the restriction for the entries argument for the operating system password file has been removed.

STIG Compliance Features

To meet Security Technical Implementation Guides (STIG) compliance, Oracle Database now provides two new user security features.

These features are as follows:

- `ora12c_stig_verify_function` password complexity function
- `ora_stig_profile` user profile
- The following initialization parameters have changed to accommodate STIG requirements:
  - The default for `SEC_PROTOCOL_ERROR_FURTHER_ACTION` is now `(DROP,3)`.
  - The default for `SEC_MAX_FAILED_LOGIN_ATTEMPTS` is now `3`.
  - The default for `SQL92_SECURITY PARAMETER` is now `TRUE`.

See Also:

- ora12c_stig_verify_function Password Requirements
- ora_stig_profile User Profile
- *Oracle Database Reference* for more information about initialization parameters

Better Security for Password Versions

In this release, Oracle Database provides several enhancements for password authentication versions.

- The default for the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter is now `12` (Exclusive Mode) instead of `11`. A setting of `12` generates both `11G` and `12C` password versions. If you want to restrict the generation to the `12C` password version, which increases security for the passwords, then you can set `SQLNET.ALLOWED_LOGON_VERSION_SERVER` to `12a`.

  Be aware that you should check the versions of the clients that connect to the server to ensure that these clients use the `O5L_NP` ability. All Oracle Database release 11.2.0.3 and later clients have the `O5L_NP` ability. If you have an earlier Oracle Database client, then you must install the CPUOct2012 patch.
- The `12C` password version is now generated automatically. In previous releases, the `10G` password version was generated automatically.

See Also:

- Managing Password Case Sensitivity
- Ensuring Against Password Security Threats by Using the 12C Password Version
- *Oracle Database Upgrade Guide* for information about how password case sensitivity is affected by Oracle Database upgrades

Ability to Automatically Lock Inactive Database User Accounts

Starting with this release, you can configure user accounts to automatically lock if they have been inactive over a period of time.

The `CREATE USER` and `ALTER USER` SQL statements enable you to set a new profile parameter, `INACTIVE_ACCOUNT_TIME`, which enables you to automatically lock inactive accounts.

More Flexibility in Controlling Database Link Access

Starting with this release, you have greater control in managing database link access by users.

You can enable or disable database link access in the following areas:

- Protecting the database from man-in-the-middle attacks that access the database through database links: To control this functionality, you can set the `OUTBOUND_DBLINK_PROTOCOLS` initialization parameter.
- Controlling the ability of users to use LDAP to access data through global database links: You can set the `ALLOW_GLOBAL_DBLINKS` initialization parameter.

For more information, see Network Connection Security.

Ability to Control Definer's Rights Privileges for Database Links

The `INHERIT REMOTE PRIVILEGES` and `INHERIT ANY REMOTE PRIVILEGES` privileges control definer's rights privileges for procedures that users accesse through a database link.

These privileges are similar to the `INHERIT PRIVILEGES` and `INHERIT ANY PRIVILEGES` privileges.

PDB Lockdown Profiles to Restrict Operations on PDBs

In a multitenant environment, you now can use PDB lockdown profiles to restrict functionality available to users in a given PDB.

PDB lockdown profiles enable you to restrict the access the user has to a set of features individually or in a group. This feature is designed to enhance security for situations in which identities are shared among PDBs.

Ability to Set the Identity of the Operating System User for PDBs

For multitenant environments, the `PDB_OS_CREDENTIAL` initialization parameter enables a different operating system user to execute external procedures using the `EXTPROC` process.

In the previous release, `EXTPROC` was used to create a credential object using `DBMS_CREDENTIAL` for authenticating and invoking external procedures. This functionality now is available on PDB level by associating a `CREDENTIAL` for the entire PDB, which will be automatically used for any `EXTPROC` external procedures executed by a database user in the PDB.

This feature enhances security for the multitenant environment by enabling each PDB to have its own operating system user account, instead of relying on one user, the `oracle` operating system user, for all PDBs in the environment. The root will continue to use the oracle operating system user account.

See Configuring Operating System Users for a PDB for more information.

Updated Kerberos Utilities

Oracle Database now supports the updated version of the Kerberos tools from MIT Kerberos 1.8.

In previous releases, Oracle clients needed to be manually configured by an administrator who configures a `krb5.conf` file. In this release, Oracle Database provides a generic `krb5.conf` file, which is used by default. The `kerb5.conf` file has parameters that enable realm and KDC information to be automatically retrieved from the DNS information. The auto-discovery of the realm and KDC information reduces the amount of work that you must perform to configure a client endpoint to use Kerberos.

This enhancement provides updates to the `okinit`, `oklist`, and `okdstry` Kerberos adapter command-line utilities, and provides a new utility, `okcreate`, which enables you to automate the creation of keytabs from either the KDC or a service endpoint.

Additional Security Feature Support for Transparent Sensitive Data Protection

You now can create Transparent Sensitive Data Protection policies to use unified auditing policies, fine-grained auditing policies, and Transparent Data Encryption column encryption.

In the previous release, you could create Transparent Sensitive Data Protection policies that use the Oracle Data Redaction and Oracle Virtual Private Database security features.

In addition, you now can use a separate wallet password for each pluggable database (PDB) in a multitenant environment so that each PDB is able to use its own wallet with its own certificates for Transparent Sensitive Data Protection authentication. The ability to specify a distinct wallet password for each PDB enables you to better restrict administrative access to the wallet. This functionality provides the necessary isolation between the tenants in a multitenant environment. These tenants can be individual companies or they can be different business units in a large corporation.

Ability to Enable Unified Auditing for Groups of Users Through Roles

Starting with this release, you can enable audit policies on database roles.

This feature enables the policy to be in effect for all users who have been directly granted the role. The policy remains effective for the user as long as the user is granted the role, and as long as the role exists.

To accommodate this enhancement, the following changes have been made:

- The `AUDIT` and `NOAUDIT` statements now have a new clause, `BY USERS WITH GRANTED ROLES`.
- The `AUDIT_UNIFIED_ENABLED_POLICIES` and `DBA_XS_ENB_AUDIT_POLICIES` data dictionary views have the following new columns:
  - `ENTITY_NAME` captures the user name or role name.
  - `ENTITY_TYPE` indicates if the entity name is a `USER` or a `ROLE`.
  - `ENABLED_OPTION` displays `BY` and `EXCEPT` for policies that are enabled on users, but displays `INVALID` for policies that are enabled on roles.
- The possible output for the `AUDIT_UNIFIED_ENABLED_POLICIES.ENABLED_OPTION` column is now `BY USER`, `EXCEPT USER`, `BY GRANTED ROLE`, and `INVALID`.

See Also:

- Enabling and Applying Unified Audit Policies to Users and Roles
- *Oracle Database Reference* for information about the `AUDIT_UNIFIED_ENABLED_POLICIES` and `DBA_XS_ENB_AUDIT_POLICIES` data dictionary views

New Audit Events for Oracle Database Real Application Security

This release provides two new audit events for Oracle Real Application Security unified audit policies.

-
- `AUDIT_GRANT_PRIVILEGE` audits use of the `GRANT_SYSTEM_PRIVILEGE` privilege.
- `AUDIT_REVOKE_PRIVILEGE` audits use of the `REVOKE_SYSTEM_PRIVILEGE` privilege.

Ability to Capture Oracle Virtual Private Database Predicates in the Audit Trail

Oracle Virtual Private Database (VPD)-generated predicates now can be captured in the audit trail.

This enhancement applies to the unified audit trail, fine-grained audit trail, and if unified auditing is not enabled, the standard audit trail (`DBA_AUDIT_TRAIL`). To accommodate this enhancement, the following data dictionary views have a new column, `RLS_INFO`:

- `UNIFIED_AUDIT_TRAIL`

- DBA_AUDIT_TRAIL (used for environments that are not yet migrated to unified auditing)
- V$XML_AUDIT_TRAIL (used for environments that are not yet migrated to unified auditing)
- DBA_FGA_AUDIT_TRAIL (used for environments that are not yet migrated to unified auditing)

If multiple VPD policies are enforced on a single object, then all of the VPD policy types, policy schema, policy names, and predicates are concatenated and stored in a single column. To separate the VPD predicate information for each VPD policy into individual rows in the view, a new PL/SQL package is available, DBMS_AUDIT_UTIL.

See Also:

- Auditing of Oracle Virtual Private Database Predicates
- Fine-Grained Auditing on Tables or Views That Have Oracle VPD Policies
- *Oracle Database Reference* for information about the UNIFIED_AUDIT_TRAIL, DBA_AUDIT_TRAIL, V$XML_AUDIT_TRAIL, and DBA_FGA_AUDIT_TRAIL data dictionary views
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the DBMS_AUDIT_UTIL PL/SQL package

Enhancements for the AUDSYS Audit Schema

The AUDSYS schema, which stores unified audit records, now has better security and better read performance for unified auditing.

In previous releases, users who had the SELECT ANY TABLE system privilege were able to query objects in the AUDSYS schema. Starting with this release, users who have this privilege are no longer able to query AUDSYS schema objects. Users who have been granted the SELECT ANY DICTIONARY system privilege can access objects in the AUDSYS schema.

In addition, a new internal table is available in the AUDSYS schema. This table stores the unified audit trail records and is designed to improve the read performance of the unified audit trail. If you migrated to unified auditing in the previous release and still have audit records in the previous location, then you can transfer the unified audit records that were created in that release to this new internal table. The DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS PL/SQL procedure, also new for this release, can be used to transfer these records to the new internal relational table.

See Also:

- What Is Auditing? for more information about the AUDSYS schema
- *Oracle Database Upgrade Guide* for information about transferring unified audit records

Deprecated Features

The following features have been deprecated for this release:

Deprecated Columns from the AUDIT_UNIFIED_ENABLED_POLICIES and DBA_XS_ENB_AUDIT_POLICIES Views

The `USER_NAME` and ENABLED_OPT columns of the `AUDIT_UNIFIED_ENABLED_POLICIES` and `DBA_XS_ENB_AUDIT_POLICIES` data dictionary views are deprecated for this release.

In this release, the `USER_NAME` column continues to show the names of users on whom the policy is enabled, but for policies that are enabled on roles, the `USER_NAME` column displays `NULL`.

The `ENABLED_OPT` column displays `BY` and `EXCEPT` for policies that are enabled on users, but displays `INVALID` for policies that are enabled on roles. This column is replaced by the `ENABLED_OPTION` column.

Deprecation of the sqlnet.ora KERBEROS5_CONF_MIT Parameter

The `sqlnet.ora` `KERBEROS5_CONF_MIT` parameter has been deprecated starting with this release.

This parameter specifies whether the new Kerberos configuration format is used. In previous releases, the default for the `KERBEROS5_CONF_MIT` parameter was `FALSE`. If the value is `TRUE`, then Oracle Database uses the new Kerberos functionality that is available with this release. If the value is set to `FALSE`, then non-MIT settings are used. Because this parameter is now deprecated, only the new configuration is supported. For backward compatibility, the `okinit`, `oklist`, and `okdstry` Kerberos utilities will work with `KERBEROS5_CONF_MIT` set to `FALSE`, thereby enabling them to use the earlier versions of these utilities. Oracle recommends that you set `KERBEROS5_CONF_MIT` to `TRUE` so that you can take advantage of the new Kerberos functionality.

Deprecated Password Verification Functions

The `VERIFY_FUNCTION` and `VERIFY_FUNCTION_11G` password verify functions have been deprecated for this release.

These functions are deprecated because they enforce the weaker password restrictions from earlier releases. Instead, you should use the `ORA12C_VERIFY_FUNCTION` and `ORA12C_STRONG_VERIFY_FUNCTION` functions, which enforce stronger, more up-to-date password verification restrictions.

Deprecation of the UNIFIED_AUDIT_SGA_QUEUE_SIZE Initialization Parameter

The `UNIFIED_AUDIT_SGA_QUEUE_SIZE` initialization parameter has been deprecated.

This parameter is no longer necessary because starting with this release, Oracle Database auditing no longer depends on the system global area (SGA)-based queues to write unified audit records.

Deprecation of the CONTAINER_GUID Parameter from the DBMS_AUDIT_MGMT Package

As part of an internal redesign to improve audit performance, the `CONTAINER_GUID` parameter has been deprecated from `DBMS_AUDIT_MGMT` PL/SQL package.

This parameter is no longer necessary. This deprecation affects the following `DBMS_AUDIT_MGMT` procedures:

- `DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL`
- `DBMS_AUDIT_MGMT.CLEAR_LAST_ARCHIVE_TIMESTAMP`
- `DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP`

See Also:

- Purging Audit Trail Records
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_AUDIT_MGMT` PL/SQL package

Deprecation of Settings to Flush Audit Trail Records to Disk

You no longer need to manually flush audit records to disk because they are now automatically written to a new internal relational table.

This enhancement enables the flushing process to bypass the common logging infrastructure queues. The deprecated settings are as follows:

- `DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL` procedure
- `AUDIT_TRAIL_WRITE` mode of the `AUDIT_TRAIL_PROPERTY` parameter of the `DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY` procedure

1 Introduction to Oracle Database Security

Oracle Database provides a rich set of default security features to manage user accounts, authentication, privileges, application security, encryption, network traffic, and auditing.

About Oracle Database Security

You can use the default Oracle Database features to configure security in several areas for your Oracle Database installation.

The areas in which you can configure security are as follows:

- **User accounts.** When you create user accounts, you can secure them in a variety of ways. You can also create password profiles to better secure password policies for your site. Managing Security for Oracle Database Users, describes how to manage user accounts.
- **Authentication methods.** Oracle Database provides several ways to configure authentication for users and database administrators. For example, you can authenticate users on the database level, from the operating system, and on the network. Configuring Authentication, describes how authentication in Oracle Database works.
- **Privileges and roles.** You can use privileges and roles to restrict user access to data. The following chapters describe how to manage privileges and roles:
  - Configuring Privilege and Role Authorization
  - Managing Security for Definer's Rights and Invoker's Rights
  - Managing Fine-Grained Access in PL/SQL Packages and Types
  - Managing Security for a Multitenant Environment in Enterprise Manager
- **Application security.** The first step to creating a database application is to ensure that it is properly secure. Managing Security for Application Developers, discusses how to incorporate application security into your application security policies.
- **User session information using application context.** An application context is a name-value pair that holds the session information. You can retrieve session information about a user, such as the user name or terminal, and restrict database and application access for that user based on this information. Using Application Contexts to Retrieve User Information, describes how to use application contexts.
- **Database access on the row and column level using Virtual Private Database.** A Virtual Private Database policy dynamically imbeds a `WHERE` predicate into SQL statements the user issues. Using Oracle Virtual Private Database to Control Data Access, describes how to create and manage Virtual Private Database policies.
- **Classify and protect data in different categories.** You can find all table columns in a database that hold sensitive data (such as credit card or Social Security numbers), classify this data, and then create a policy that protects this data as a

whole for a given class. Using Transparent Sensitive Data Protection , explains how to create Transparent Sensitive Data Protection policies.

- **Network data encryption.** Manually Encrypting Data, explains how to use the `DBMS_CRYPTO` PL/SQL package to encrypt data as it travels on the network to prevent unauthorized access to that data. You can configure native Oracle Net Services data encryption and integrity for both servers and clients, which are described in Configuring Oracle Database Network Encryption and Data Integrity.
- **Thin JDBC client network configuration.** You can configure thin Java Database Connectivity (JDBC) clients to securely connect to Oracle databases. Configuring the Thin JDBC Client Network, provides detailed information.
- **Strong authentication.** You can configure your databases to use strong authentication with Oracle authentication adapters that support various third-party authentication services, including SSL with digital certificates. Oracle Database provides the following strong authentication support:
  - Centralized authentication and single sign-on.
  - Kerberos
  - Remote Authentication Dial-in User Service (RADIUS)
  - Secure Sockets Layer (SSL)

  The following chapters cover strong authentication:
  - Introduction to Strong Authentication
  - Strong Authentication Administration Tools
  - Configuring Kerberos Authentication
  - Configuring Secure Sockets Layer Authentication
  - Configuring RADIUS Authentication
  - Customizing the Use of Strong Authentication
- **Auditing database activities.** You can audit database activities in general terms, such as auditing all SQL statements, SQL privileges, schema objects, and network activity. Or, you can audit in a granular manner, such as when the IP addresses from outside the corporate network is being used. This chapter also explains how to purge the database audit trail. The following chapters describe how to configure and administer database auditing.
  - Introduction to Auditing
  - Configuring Audit Policies
  - Administering the Audit Trail

In addition, Keeping Your Oracle Database Secure, provides guidelines that you should follow when you secure your Oracle Database installation.

Additional Oracle Database Security Resources

In addition to the security resources described in this guide, Oracle Database provides several other database security products.

These products are as follows:

- **Oracle Advanced Security.** See *Oracle Database Advanced Security Guide* for information about Transparent Data Encryption and Oracle Data Redaction.
- **Oracle Label Security.** Oracle Label Security applies classification labels to data, allowing you to filter user access to data at the row level. See *Oracle Label Security Administrator's Guide* for detailed information about Oracle Label Security.
- **Oracle Database Vault.** Oracle Database Vault provides fine-grained access control to your sensitive data, including protecting data from privileged users. *Oracle Database Vault Administrator's Guide* describes how to use Oracle Database Vault.
- **Oracle Audit Vault and Database Firewall.** Oracle Audit Vault and Database Firewall collects database audit data from sources such as Oracle Database audit trail tables, database operating system audit files, and database redo logs. Using Oracle Audit Vault and Database Firewall, you can create alerts on suspicious activities, and create reports on the history of privileged user changes, schema modifications, and even data-level access. *Oracle Audit Vault and Database Firewall Administrator's Guide* explains how to administer Oracle Audit Vault and Database Firewall.
- **Oracle Enterprise User Security.** Oracle Enterprise User Security enables you to manage user security at the enterprise level. *Oracle Database Enterprise User Security Administrator's Guide* explains how to configure Oracle Enterprise User Security.
- **Oracle Enterprise Manager Data Masking and Subsetting Pack.** Data Masking and Subsetting Pack helps reduce this risk by irreversibly replacing the original sensitive data with fictitious data so that production data can be shared safely with IT developers or offshore business partners. See *Oracle Database Testing Guide* for additional information.

In addition to these products, you can find the latest information about Oracle Database security, such as new products and important information about security patches and alerts, by visiting the Security Technology Center on Oracle Technology Network at

`http://www.oracle.com/technetwork/topics/security/whatsnew/index.html`

Part I

Managing User Authentication and Authorization

Part I describes how to manage user authentication and authorization.

2

Managing Security for Oracle Database Users

You can manage the security for Oracle Database users in many ways, such as enforcing restrictions on the way that passwords are created.

About User Security

You can secure users accounts through strong passwords and by specifying special limits for the users.

Each Oracle database has a list of valid database users. To access a database, a user must run a database application, and connect to the database instance using a valid user name defined in the database.

When you create user accounts, you can specify limits to the user account. You can also set limits on the amount of various system resources available to each user as part of the security domain of that user. Oracle Database provides a set of database views that you can query to find information such as resource and session information. This chapter also describes profiles. A profile is collection of attributes that apply to a user. It enables a single point of reference for any of multiple users that share those exact attributes.

See Also:

Configuring Privilege and Role Authorization describes how you can also manage user security by assigning users privileges and roles

Creating User Accounts

A user account can have restrictions such as profiles, a default role, and tablespace restrictions.

About Common Users and Local Users

In a multitenant environment, CDB common users and application common have access to their respective containers, and local users are specific to a PDB.

About Common Users

Oracle provides two types of common users: CDB common users and application common users.

A CDB common user is a database user whose single identity and password are known in the CDB root and in every existing and future pluggable database (PDB), including any application roots. All Oracle-supplied administrative user accounts, such as `SYS` and `SYSTEM`, are CDB common users and can navigate across the system container. CDB common users can have different privileges in different PDBs. For example, the user `SYSTEM` can switch between PDBs and use the privileges that are granted to `SYSTEM` in the current PDB. However, if one of the PDBs is Oracle Database Vault-enabled, then the Database Vault restrictions, such as `SYSTEM` not being allowed to create user accounts, apply to `SYSTEM` when this user is connected to that PDB. Oracle does not recommend that you change the privileges of the Oracle-supplied CDB common users.

A CDB common user can perform all tasks that an application common user can perform, provided that appropriate privileges have been granted to that user.

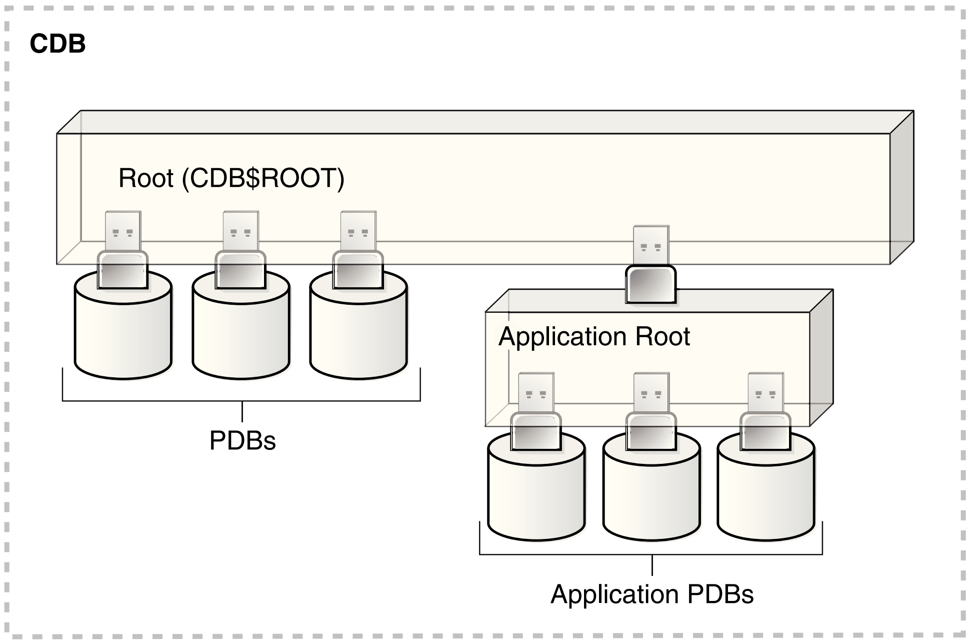
An application common user is a user account that is created in an application root, and is common only within this application container. In other words, the application common user does not have access to the entire CDB environment like CDB common users. An application common user is responsible for activities such as creating (which includes plugging), opening, closing, unplugging, and dropping application PDBs. This user can create application common objects in the application root. You can create an application common user only when you are connected to an application root. The ability for users to access the application common objects is subject to the same privileges as local and CDB common objects. For example, a local user in a PDB that is associated with an application root has access to only the objects in that PDB for which the user has privileges. In the application root itself, you can commonly grant a privilege on a CDB common object that will apply across the application container.

Both of these types of common users are responsible for managing the common objects in their respective roots. If the CDB common user or the application common user has the appropriate privileges, then this user can perform operations in PDBs as well, such as granting privileges to local users. These users can also locally grant common users different privileges in each container.

Both CDB and application common users can perform the following activities:

- Granting privileges to common users or common roles. That is, a CDB common user can grant a privilege to a common user or role, and the scope within which this privilege applies is determined by the container (CDB root, application root, or PDB) in which the statement is issued and whether the privilege is granted commonly (in the CDB root or the application root). A CDB common user connected to an application root can commonly grant a privilege on a CDB common object, and that privilege will apply across the application container.

  The following diagram illustrates the access hierarchy with CDB common users, application common users, and local users:



CDB common users are defined in the CDB root and may be able to access all PDBs within the CDB, including application roots and their application PDBs. Application common users are defined in the application root and have access to the PDBs that belong to the application container. Local users in either the CDB PDBs or the application PDBs have access only to the PDBs in which the local user resides.

- The state of a PDB can be altered by a suitably privileged user who can issue the `ALTER PLUGGABLE DATABASE` statement from the CDB root, from an application root (if a PDB is an application PDB that belongs to the application container), or from a PDB itself.

One difference between CDB common users and application common users is that only a CDB common user can run an `ALTER DATABASE` statement that specifies the recovery clauses that apply to the entire CDB.

See Also:

- About Creating Common User Accounts
- About Commonly and Locally Granted Privileges for more information about how privileges work in with PDBs
- *Oracle Database Concepts* for more conceptual information about CDB common users and application common users

How Plugging in PDBs Affects CDB Common Users

Plugging a non-CDB into a CDB as a PDB affects both Oracle-supplied administrative and user-created accounts and privileges.

This affects the passwords of these CDB common user accounts, and privileges of all accounts in the newly plugged-in database.

The following actions take place:

- The Oracle-supplied administrative accounts are merged with the existing common user accounts.
- User-created accounts are merged with the existing user-created common user accounts.
- The passwords of the existing CDB common user accounts take precedence over the passwords for the accounts from the non-CDB.
- If you had modified the privileges of a user account in its original non-CDB, then these privileges are saved, but they only apply to the PDB that was created when the PDB was plugged into the CDB, as locally granted privileges. For example, suppose you had granted the user `SYSTEM` a role called `hr_mgr` in the non-CDB `db1`. After the `db1` database has been added to a CDB, then `SYSTEM` can only use the `hr_mgr` role in the `db1` PDB, and not in any other PDBs.

The following two scenarios are possible when you plug a PDB (for example, `pdb_1`) from one CDB (`cdb_1`) to a another CDB (`cdb_2`):

- `cdb_1` has the common user `c##cdb1_user`. `cdb_2` does not have this user.

  `c##cdb1_user` remains in `PDB_1` but this account is locked. To resurrect this account, you must close `pdb_1`, recreate common user `c##cdb1_user` in the root of `cdb_2`, and then re-open `pdb_1`.
- `cdb_1` and `cdb_2` both have common user `c##common_user`.

  Both `c##common_user` accounts are merged. `c##common_user` retains its password in `cdb_2`. Any privileges assigned to it in `cdb_2` but not in `cdb_1` are retained locally in `pdb_1`.

About Local Users

In a multitenant environment, a local user is a database user that exists only in a single PDB.

Local users can have administrative privileges, but these privileges apply only in the PDB in which the local user account was created. A local user account has the following characteristics, which distinguishes it from common user accounts:

- Local user accounts cannot create common user accounts or commonly grant them privileges. A common user with the appropriate privileges can create and modify common or local user accounts and grant and revoke privileges, commonly or locally. A local user can create and modify local user accounts or locally grant privileges to common or local users in a given PDB.
- You can grant local user accounts common roles. However, the privileges associated with the common role only apply to the local user's PDB.
- The local user account must be unique only within its PDB.
- With the appropriate privileges, a local user can access objects in a common user's schema. For example, a local user can access a table within the schema of a common user if the common user has granted the local user privileges to access it.
- You can editions-enable a local user account but not a common user account.

See Also:

- About Creating Local User Accounts
- *Oracle Database Concepts* for more conceptual information about local users

Who Can Create User Accounts?

Users who has been granted the `CREATE USER` system privilege can create user accounts, including user accounts to be used as proxy users.

Because the `CREATE USER` system privilege is a powerful privilege, a database administrator or security administrator is usually the only user who has this system privilege.

If you want to create users who themselves have the privilege to create users, then include the `WITH ADMIN OPTION` clause in the `GRANT` statement. For example:

```
GRANT CREATE SESSION TO lbrown WITH ADMIN OPTION;
```

As with all user accounts to whom you grant privileges, grant these privileges to trusted users only.

In a multitenant environment, you must have the commonly granted `CREATE USER` system privilege to create common user accounts. To create local user accounts, you must have a commonly granted `CREATE USER` privilege or a locally granted `CREATE USER` privilege in the PDB in which the local user account will be created.

Note:

As a security administrator, you should create your own roles and assign only those privileges that are needed. For example, many users formerly granted the `CONNECT` privilege did not need the additional privileges `CONNECT` used to provide. Instead, only `CREATE SESSION` was actually needed. By default, the `SET CONTAINER` privilege is granted to `CONNECT` role.

Creating organization-specific roles gives an organization detailed control of the privileges it assigns, and protects it in case Oracle Database changes the roles that it defines in future releases.

Creating a New User Account That Has Minimum Database Privileges

When you create a new user account, you should enable this user to access the database.

1. Use the `CREATE USER` statement to create a new user account.

   For example:

   ```
   CREATE USER jward
   ```

```
IDENTIFIED BY password
DEFAULT TABLESPACE example
QUOTA 10M ON example
TEMPORARY TABLESPACE temp
QUOTA 5M ON system
PASSWORD EXPIRE;
```

Follow the guidelines in Minimum Requirements for Passwords to replace *password* with a password that is secure.

This example creates a local user account and specifies the user password, default tablespace, temporary tablespace where temporary segments are created, tablespace quotas, and profile.

2. At minimum, grant the user the `CREATE SESSION` privilege so that the user can access the database instance.

   ```
   GRANT CREATE SESSION TO jward;
   ```

   A newly created user cannot connect to the database until he or she has the `CREATE SESSION` privilege. If the user must access Oracle Enterprise Manager, then you should also grant the user the `SELECT ANY DICTIONARY` privilege.

Restrictions on Creating the User Name for a New Account

When you specify a name for a user account, be aware of restrictions such as naming conventions and whether the name is unique.

Uniqueness of User Names

Each user has an associated schema; within a schema, each schema object must have a unique name.

Oracle Database will prevent you from creating a user name if it is already exists. You can check existing names by querying the `USERNAME` column of the `DBA_USERS` data dictionary view.

User Names in a Multitenant Environment

Within each PDB, a user name must be unique with respect to other user names and roles in that PDB.

Note the following restrictions:

- For common user names, names for user-created common users must begin with a common user prefix. By default, for CDB common users, this prefix is `C##`. For application common users, this prefix is an empty string. This means that there are no restrictions on the name that can be assigned to an application common user other than that it cannot start with the prefix reserved for CDB common users. For example, you could name a CDB common user `c##hr_admin` and an application common user `hr_admin`.

  The `COMMON_USER_PREFIX` parameter in `CDB$ROOT` defines the common user prefix. You can change this setting, but do so only with great care.
- For local user names, the name cannot start with `C##` (or `c##`)
- A user and a role cannot have the same name.

Case Sensitivity for User Names

How you create a user name controls the case sensitivity in which the user name is stored in the database.

For example:

```
CREATE USER jward
 IDENTIFIED BY password
 DEFAULT TABLESPACE data_ts
 QUOTA 100M ON test_ts
 QUOTA 500K ON data_ts
 TEMPORARY TABLESPACE temp_ts
 PROFILE clerk
 CONTAINER = CURRENT;
```

User `jward` is stored in the database in upper-case letters. For example:

```
SELECT USERNAME FROM ALL_USERS;

USERNAME
---------
JWARD
...
```

However, if you enclose the user name in double quotation marks, then the name is stored using the case sensitivity that you used for the name. For example:

```
CREATE USER "jward" IDENTIFIED BY password;
```

So, when you query the `ALL_USERS` data dictionary view, you will find that the user account is stored using the case that you used to create it.

```
SELECT USERNAME FROM ALL_USERS;

USERNAME
---------
jward
...
```

User `JWARD` and user `jward` are both stored in the database as separate user accounts. Later on, if you must modify or drop the user that you had created using double quotation marks, then you must enclose the user name in double quotation marks.

For example:

```
DROP USER "jward";
```

Assignment of User Passwords

The `IDENTIFIED BY` clause of the `CREATE USER` statement assigns the user a password.

Ensure that you create a secure password.

In the example in Creating a New User Account That Has Minimum Database Privileges, the new local user is authenticated using the database. In this case, the connecting user must supply the correct password to the database to connect successfully.

```
CREATE USER jward
 IDENTIFIED BY password
 DEFAULT TABLESPACE data_ts
 QUOTA 100M ON test_ts
 QUOTA 500K ON data_ts
 TEMPORARY TABLESPACE temp_ts
 PROFILE clerk
 CONTAINER = CURRENT;
```

Default Tablespace for the User

A default tablespace stores objects that users create.

About Assigning a Default Tablespace for a User

Each user should have a default tablespace.

When a schema object is created in the user's schema and the DDL statement does not specify a tablespace to contain the object, the Oracle Database stores the object in the user's default tablespace.

Tablespaces enable you to separate user data from system data, such as the data that is stored in the `SYSTEM` tablespace. You use the `CREATE USER` or `ALTER USER` statement to assign a default tablespace to a user. The default setting for the default tablespaces of all users is the `SYSTEM` tablespace. If a user does not create objects, and has no privileges to do so, then this default setting is fine. However, if a user is likely to create any type of object, then you should specifically assign the user a default tablespace, such as the `USERS` tablespace. Using a tablespace other than `SYSTEM` reduces contention between data dictionary objects and user objects for the same data files. In general, do not store user data in the `SYSTEM` tablespace.

You can use the `CREATE TABLESPACE` SQL statement to create a permanent default tablespace other than `SYSTEM` at the time of database creation, to be used as the database default for permanent objects. By separating the user data from the system data, you reduce the likelihood of problems with the `SYSTEM` tablespace, which can in some circumstances cause the entire database to become nonfunctional. This default permanent tablespace is not used by system users, that is, `SYS`, `SYSTEM`, and `OUTLN`, whose default permanent tablespace is `SYSTEM`. A tablespace designated as the default permanent tablespace cannot be dropped. To accomplish this goal, you must first designate another tablespace as the default permanent tablespace. You can use the `ALTER TABLESPACE` SQL statement to alter the default permanent tablespace to another tablespace. Be aware that this will affect all users or objects created after the `ALTER` DDL statement is executed.

You can also set a user default tablespace during user creation, and change it later with the `ALTER USER` statement. Changing the user default tablespace affects only objects created after the setting is changed.

When you specify the default tablespace for a user, also specify a quota on that tablespace.

DEFAULT TABLESPACE Clause for Assigning a Default Tablespace

The `DEFAULT TABLESPACE` clause in the `CREATE USER` statement assigns a default tablespace to the user.

In the following CREATE USER statement, the default tablespace for local user jward is data_ts:

```
CREATE USER jward
 IDENTIFIED BY password
 DEFAULT TABLESPACE data_ts
 QUOTA 100M ON test_ts
 QUOTA 500K ON data_ts
 TEMPORARY TABLESPACE temp_ts
 PROFILE clerk
 CONTAINER = CURRENT;
```

Tablespace Quotas for a User

The tablespace quota defines how much space to provide for a user's tablespace.

About Assigning a Tablespace Quota for a User

You can assign each user a tablespace quota for any tablespace, except a temporary tablespace.

Assigning a quota accomplishes the following:

- Users with privileges to create certain types of objects can create those objects in the specified tablespace.
- Oracle Database limits the amount of space that can be allocated for storage of a user's objects within the specified tablespace to the amount of the quota.

By default, a user has no quota on any tablespace in the database. If the user has the privilege to create a schema object, then you must assign a quota to allow the user to create objects. At a minimum, assign users a quota for the default tablespace, and additional quotas for other tablespaces in which they can create objects. The maximum amount of space that you can assign for a tablespace is 2 TB. If you need more space, then specify UNLIMITED for the QUOTA clause.

You can assign a user either individual quotas for a specific amount of disk space in each tablespace or an unlimited amount of disk space in all tablespaces. Specific quotas prevent a user's objects from using too much space in the database.

You can assign quotas to a user tablespace when you create the user, or add or change quotas later. (You can find existing user quotas by querying the USER_TS_QUOTAS view.) If a new quota is less than the old one, then the following conditions remain true:

- If a user has already exceeded a new tablespace quota, then the objects of a user in the tablespace cannot be allocated more space until the combined space of these objects is less than the new quota.
- If a user has not exceeded a new tablespace quota, or if the space used by the objects of the user in the tablespace falls under a new tablespace quota, then the user's objects can be allocated space up to the new quota.

CREATE USER Statement for Assigning a Tablespace Quota

The QUOTA clause of the CREATE USER statement assigns the quotas for a tablespace.

The following CREATE USER statement assigns quotas for the test_ts and data_ts tablespaces:

```
CREATE USER jward
 IDENTIFIED BY password
 DEFAULT TABLESPACE data_ts
 QUOTA 500K ON data_ts
 QUOTA 100M ON test_ts
 TEMPORARY TABLESPACE temp_ts
 PROFILE clerk
 CONTAINER = CURRENT;
```

Restriction of the Quota Limits for User Objects in a Tablespace

You can restrict the quota limits for user objects in a tablespace so that the current quota is zero.

To restrict the quote limits, use the `ALTER USER` SQL statement.

After a quota of zero is assigned, the objects of the user in the tablespace remain, and the user can still create new objects, but the existing objects will not be allocated any new space. For example, you could not insert data into one of this user's existing tables. The operation will fail with an `ORA-1536 space quota exceeded for tables` error.

Grants to Users for the UNLIMITED TABLESPACE System Privilege

To permit a user to use an unlimited amount of any tablespace in the database, grant the user the `UNLIMITED TABLESPACE` system privilege.

The `UNLIMITED TABLESPACE` privilege overrides all explicit tablespace quotas for the user. If you later revoke the privilege, then you must explicitly grant quotas to individual tablespaces. You can grant this privilege only to users, not to roles.

Before granting the `UNLIMITED TABLESPACE` system privilege, consider the consequences of doing so.

Advantage:

- You can grant a user unlimited access to all tablespaces of a database with one statement.

Disadvantages:

- The privilege overrides all explicit tablespace quotas for the user.
- You cannot selectively revoke tablespace access from a user with the `UNLIMITED TABLESPACE` privilege. You can grant selective or restricted access only after revoking the privilege.

Temporary Tablespaces for the User

A temporary tablespace contains transient data that persists only for the duration of a user session.

About Assigning a Temporary Tablespace for a User

You should assign each user a temporary tablespace.

When a user executes a SQL statement that requires a temporary segment, Oracle Database stores the segment in the temporary tablespace of the user. These temporary segments are created by the system when performing sort or join operations. Temporary segments are owned by SYS, which has resource privileges in all tablespaces.

To create a temporary tablespace, you can use the CREATE TEMPORARY TABLESPACE SQL statement.

If you do not explicitly assign the user a temporary tablespace, then Oracle Database assigns the user the default temporary tablespace that was specified at database creation, or by an ALTER DATABASE statement at a later time. If there is no default temporary tablespace explicitly assigned, then the default is the SYSTEM tablespace or another permanent default established by the system administrator. Assigning a tablespace to be used specifically as a temporary tablespace eliminates file contention among temporary segments and other types of segments.

Note:

If your SYSTEM tablespace is locally managed, then users must be assigned a specific default (locally managed) temporary tablespace. They may not be allowed to default to using the SYSTEM tablespace because temporary objects cannot be placed in locally managed permanent tablespaces.

You can set the temporary tablespace for a user at user creation, and change it later using the ALTER USER statement. You can also establish tablespace groups instead of assigning individual temporary tablespaces.

See Also:

Oracle Database Administrator's Guide for more information about temporary tablespaces and using tablespace groups

TEMPORARY TABLESPACE Clause for Assigning a Temporary Tablespace

The TEMPORARY TABLESPACE clause in the CREATE USER statement assigns a user a temporary tablespace.

In the following example, the temporary tablespace of jward is temp_ts, a tablespace created explicitly to contain only temporary segments.

```
CREATE USER jward
 IDENTIFIED BY password
 DEFAULT TABLESPACE data_ts
 QUOTA 100M ON test_ts
 QUOTA 500K ON data_ts
 TEMPORARY TABLESPACE temp_ts
 PROFILE clerk
 CONTAINER = CURRENT;
```

Profiles for the User

A profile is a set of limits, defined by attributes, on database resources and password access to the database.

The profile can be applied to multiple users, enabling them to share these attributes.

You can specify a profile when you create a user. The `PROFILE` clause of the `CREATE USER` statement assigns a user a profile. If you do not specify a profile, then Oracle Database assigns the user a default profile.

For example:

```
CREATE USER jward
 IDENTIFIED BY password
 DEFAULT TABLESPACE data_ts
 QUOTA 100M ON test_ts
 QUOTA 500K ON data_ts
 TEMPORARY TABLESPACE temp_ts
 PROFILE clerk
 CONTAINER = CURRENT;
```

In a multitenant environment, different profiles can be assigned to a common user in the root and in a PDB. When the common user logs in to the PDB, a profile whose setting applies to the session depends on whether the settings are password-related or resource-related.

- Password-related profile settings are fetched from the profile that is assigned to the common user in the root. For example, suppose you assign a common profile `c##prof` (in which `FAILED_LOGIN_ATTEMPTS` is set to `1`) to common user `c##admin` in the root. In a PDB that user is assigned a local profile `local_prof` (in which `FAILED_LOGIN_ATTEMPTS` is set to `6`.) Common user `c##admin` is allowed only one failed login attempt when he or she tries to log in to the PDB where `loc_prof` is assigned to him.
- Resource-related profile settings specified in the profile assigned to a user in a PDB get used without consulting resource-related settings in a profile assigned to the common user in the root. For example, if the profile `local_prof` that is assigned to user `c##admin` in a PDB has `SESSIONS_PER_USER` set to `2`, then `c##admin` is only allowed only 2 concurrent sessions when he or she logs in to the PDB `loc_prof` is assigned to him, regardless of value of this setting in a profile assigned to him in the root.

Creation of a Common User or a Local User

The `CREATE USER` SQL statement can be used to create both common (CDB and application) users and local users.

About Creating Common User Accounts

Be aware of common user account restrictions such as where they can be created, naming conventions, and objects stored in their schemas.

To create a common user account, follow these rules:

- To create a CDB common user, you must be connected to the CDB root and have the commonly granted `CREATE USER` system privilege.
- To create an application common user, you must be connected to the application root and have the commonly granted `CREATE USER` system privilege.
- You can run the `CREATE USER ... CONTAINER = ALL` statement to create an application common user in the application root. Afterward, you must synchronize

the application so that this user can be visible in the application PDB. For example, for an application named `saas_sales_app`:

```
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app SYNC;
```

- The name that you give the common user who connects to the CDB root must begin with the prefix that is defined in the `COMMON_USER_PREFIX` parameter in the CDB root, which by default is `C##`. (You can modify this parameter, but only do so with great caution.) It must contain only ASCII or EBCDIC characters. This naming requirement does not apply to the names of existing Oracle-supplied user accounts, such as `SYS` or `SYSTEM`. To find the names of existing user accounts, query the `ALL_USERS`, `CDB_USERS`, `DBA_USERS`, and `USER_USERS` data dictionary views.
- The name that you give the common user who connects to the application root must follow the naming conventions for standard user accounts. By default, the `COMMON_USER_PREFIX` parameter in the application root is set to an empty string. In other words, you can create a user named `hr_admin` in the application root but not a user named `c##hr_admin`.
- To explicitly designate a user account as a CDB or an application common user, in the `CREATE USER` statement, specify the `CONTAINER=ALL` clause. If you are logged into the CDB or application root, and if you omit the `CONTAINER` clause from your `CREATE USER` statement, then the `CONTAINER=ALL` clause is implied.
- Do not create objects in the schemas of common users for a CDB. Instead, you can create application common objects. These are objects whose metadata, and in case of data links or extended data links, data, is shared between all application PDBs that belong to the application container. You must create the application common object in the root of an application container.
- If you specify the `DEFAULT TABLESPACE`, `TEMPORARY TABLESPACE`, `QUOTA...ON`, and `PROFILE` clauses in the `CREATE USER` statement for a CDB or an application common user account, then you must ensure that these objects—tablespaces, tablespace groups, and profiles—exist in all containers of the CDB for a CDB common user, or in the application root and all PDBs of an application container for an application common user.

CREATE USER Statement for Creating a Common User Account

The `CREATE USER` statement `CONTAINER=ALL` clause can be used to create a common user account.

You must be in the CDB root to create a CDB common user account and the application root to create an application common user account.

The following example shows how to create a CDB common user account from the CDB root by using the `CONTAINER` clause, and then granting the user the `SET CONTAINER` and `CREATE SESSION` privileges. Common users must have the `SET CONTAINER` system privilege to navigate between containers. When you create the account, there is a single common password for this common user across all containers.

```
CONNECT SYSTEM
Enter password: password
Connected.

CREATE USER c##hr_admin
IDENTIFIED BY password
DEFAULT TABLESPACE data_ts
QUOTA 100M ON test_ts
QUOTA 500K ON data_ts
TEMPORARY TABLESPACE temp_ts
CONTAINER = ALL;

GRANT SET CONTAINER, CREATE SESSION TO c##hr_admin
CONTAINER = ALL;
```

The next example shows how to create an application common user in the application root (`app_root`) by using the `CONTAINER` clause, and then granting the user the `SET CONTAINER`, and `CREATE SESSION` system privileges. Finally, to synchronize this user so that it is visible in the application PDBs, the `ALTER PLUGGABLE DATABASE APPLICATION APP$CON SYNC` statement is run.

```
CONNECT SYSTEM@app_root
Enter password: password
Connected.

CREATE USER app_admin
IDENTIFIED BY password
DEFAULT TABLESPACE data_ts
QUOTA 100M ON temp_ts
QUOTA 500K ON data_ts
TEMPORARY TABLESPACE temp_ts
CONTAINER = ALL;

GRANT SET CONTAINER, CREATE SESSION TO app_admin CONTAINER = ALL;

CONNECT SYSTEM@app_hr_pdb
Enter password: password
Connected.

ALTER PLUGGABLE DATABASE APPLICATION APP$CON SYNC;
```

About Creating Local User Accounts

Be aware of local user account restrictions such as where they can be created, naming conventions, and objects stored in their schemas.

To create a local user account, follow these rules:

- To create a local user account, you must be connected to the PDB in which you want to create the account, and have the `CREATE USER` privilege.
- The name that you give the local user must not start with a prefix that is reserved for common users, which by default is `C##` for CDB common users.
- You can include `CONTAINER=CURRENT` in the `CREATE USER` statement to specify the user as a local user. If you are connected to a PDB and omit this clause, then the `CONTAINER=CURRENT` clause is implied.
- You cannot have common users and local users with the same name. However, you can use the same name for local users in different PDBs. To find the names of existing user accounts, query the `ALL_USERS`, `CDB_USERS`, `DBA_USERS`, and `USER_USERS` data dictionary views.
- Both common and local users connected to a PDB can create local user accounts, as long as they have the appropriate privileges.

CREATE USER Statement for Creating a Local User Account

The `CREATE USER` statement `CONTAINER` clause can be used to create a local user account.

You must create the local user account in the PDB where you want this account to reside.

The following example shows how to create a local user account using the `CONTAINER` clause.

```
CONNECT SYSTEM@hrpdb
Enter password: password
Connected.

CREATE USER kmurray
 IDENTIFIED BY password
 DEFAULT TABLESPACE data_ts
 QUOTA 100M ON test_ts
 QUOTA 500K ON data_ts
 TEMPORARY TABLESPACE temp_ts
 PROFILE hr_profile
 CONTAINER = CURRENT;
```

Creating a Default Role for the User

A default role is automatically enabled for a user when the user creates a session.

You can assign a user zero or more default roles. You cannot set default roles for a user in the `CREATE USER` statement. When you first create a user, the default role setting for the user is `ALL`, which causes all roles subsequently granted to the user to be default roles.

- Use the `ALTER USER` statement to change the default roles for the user.

For example:

```
GRANT USER rdale clerk_mgr;

ALTER USER rdale DEFAULT ROLE clerk_mgr;
```

Before a role can be made the default role for a user, that user must have been already granted the role.

Altering User Accounts

The `ALTER USER` statement modifies user accounts, such their default tablespace or profile, or changing a user's password.

About Altering User Accounts

Changing user security settings affects the future user sessions, not the current session.

In most cases, you can alter user security settings with the `ALTER USER` SQL statement. Users can change their own passwords. However, to change any other option of a user security domain, you must have the `ALTER USER` system privilege. Security administrators are typically the only users that have this system privilege, as it allows a modification of *any* user security domain. This privilege includes the ability to set tablespace quotas for a user on any tablespace in the database, even if the user performing the modification does not have a quota for a specified tablespace.

In a multitenant environment, you must have the commonly granted `ALTER USER` system privilege to alter common user accounts. To alter local user accounts, you must have a commonly granted `ALTER USER` privilege or a locally granted `ALTER USER` privilege in the PDB in which the local user account resides.

ALTER USER Statement for Altering Common or Local User Accounts

The `ALTER USER` statement can alter both common and local user accounts.

You cannot change an existing common user account to be a local user account, or vice versa. In this case, you must create a new account, as either a common user account or a local user account.

The following example shows how to use the `ALTER USER` statement to restrict user `c##hr_admin`'s ability to view `V$SESSION` rows to those that pertain to sessions that are connected to `CDB$ROOT`, and to the `emp_db` and `hr_db` PDBs.

```
CONNECT SYSTEM
Enter password: password
Connected.

ALTER USER c##hr_admin
 DEFAULT TABLESPACE data_ts
 TEMPORARY TABLESPACE temp_ts
 QUOTA 100M ON data_ts
 QUOTA 0 ON test_ts
 SET CONTAINER_DATA = (emp_db, hr_db) FOR V$SESSION
 CONTAINER = CURRENT;
```

The `ALTER USER` statement here changes the security settings for the user `c##hr_admin` as follows:

- `DEFAULT TABLESPACE` and `TEMPORARY TABLESPACE` are set explicitly to `data_ts` and `temp_ts`, respectively.
- `QUOTA 100M` gives the `data_ts` tablespace 100 MB.
- `QUOTA 0` revokes the quota on the `temp_ts` tablespace.
- `SET CONTAINER_DATA` enables user `c##hr_admin` to have access to data related to the `emp_db` and `hr_db` PDBs as well as the root when he queries the `V$SESSION` view from the root.

Changing Non-SYS User Passwords

Users can change their own passwords but to change other users' passwords, they must have the correct privileges.

About Changing Non-SYS User Passwords

Users can use either the `PASSWORD` command or `ALTER USER` statement to change a password.

No special privileges (other than those to connect to the database and create a session) are required for a user to change his or her own password. Encourage users to change their passwords frequently. You can find existing users for the current database instance by querying the `ALL_USERS` view.

For better security, use the `PASSWORD` command to change the account's password. The `ALTER USER` statement displays the new password on the screen, where it can be seen by any overly curious coworkers. The `PASSWORD` command does not display the new password, so it is only known to you, not to your co-workers. In both cases, the password is encrypted on the network.

Users must have the `PASSWORD` and `ALTER USER` privilege to switch between methods of authentication. Usually, only an administrator has this privilege.

Using the PASSWORD Command or ALTER USER Statement to Change a Password

Most users can change their own passwords with the SQL*Plus `PASSWORD` command or the `ALTER USER` SQL statement.

In a multitenant environment, a CDB common user must change his or her password in the CDB root, and an application common user must change his or her password in the application root.

- Use one of the following methods to change a user’s password:
  - To use the SQL*Plus `PASSWORD` command to change a password, supply the user's name, and when prompted, enter the new password.

    For example:

    ```
    PASSWORD andy
    Changing password for andy
    New password: password
    Retype new password: password
    ```

  - To use the `ALTER USER` SQL statement change a password, include the `IDENTIFIED BY` clause.

    For example:

    ```
    ALTER USER andy IDENTIFIED BY password;
    ```

Changing the SYS User Password

To change the `SYS` user password, you must use the `ORAPWD` command line utility.

About Changing the SYS User Password

The `ORAPWD` command line utility can create a new password file that contains the `SYS` user password.

Note the following:

- Before you can change the password of the `SYS` user account, a password file must exist for this account.
- The `SYS` user account is used by most of the internal recursive SQL. Therefore, if you try to use the `ALTER USER` statement to change this password while the

database is open, then there is a chance that deadlocks will result. To prevent this problem, when you migrate a password file, set the `ORAPWD sys` option to `y`. Use the following syntax:

```
orapwd input_file=input_file_name file=file_name sys=y force=y
Enter password for SYS: new_password
```

- If you try to use `ALTER USER` to change the `SYS` user password, and if the instance initialization parameter `REMOTE_LOGIN_PASSWORDFILE` has been set to `SHARED`, then you cannot change the `SYS` password. The `ALTER USER` statement fails with an `ORA-28046: Password change for SYS disallowed` error.
- New accounts are created with created with the `SHA-2` (`SHA-512`) verifier. You can identify these accounts by querying the `PASSWORD_VERSIONS` column of the `DBA_USERS` data dictionary view. (These password versions are listed as `12C` in the view's output.) Because this verifier is too large to fit in the original password file format, the password file must be created in the extended format, by using the `format=12` argument in the `ORAPWD` command. Otherwise, if you try to use the `PASSWORD` command to change the `SYS` password, then an `ORA-28017: The password file is not in the extended format` error will be raised.
- In an Oracle Real Application Clusters (Oracle RAC) environment, store the password in the ASM disk group so that it can be shared by multiple Oracle RAC instances.

ORAPWD Utility for Changing the SYS User Password

The `ORAPWD` utility enables you to change the `SYS` user password.

You can use the `ORAPWD` utility with the `INPUT_FILE` parameter to change the `SYS` user password. To migrate the password files to a specific format, include the `FORMAT` option. By default, the format is `12.2` if you do not specify the `FORMAT` option.

Using the `ALTER USER` statement to change `SYS` password when the database is open could lead to deadlocks. Instead, use the `ORAPWD` utility to change the `SYS` user password. To set a new password for the `SYS` user using the `ORAPWD` utility, set the `SYS` option to `Y` (yes), use the `INPUT_FILE` parameter to specify the current password file name, and use the `FILE` parameter to create the password file to which the original password file is migrated. For example:

```
ORAPWD INPUT_FILE='orapworcl' FILE='orapwd' SYS=Y
Enter password for SYS: new_password
```

If you do not want to migrate the password file to a different format, then you can specify the same format as the `input_file`. For example, assuming that the input file `orapworcl` format is `12` and you want to change the `SYS` user password:

```
ORAPWD INPUT_FILE='orapworcl' FILE='orapwd' FORMAT=12 SYS=Y
Enter password for SYS: new_password
```

See Also:

Oracle Database Administrator's Guide for more information about the `ORAPWD` utility

Configuring User Resource Limits

A resource limit defines the amount of system resources that are available for a user.

About User Resource Limits

You can set limits on the amount of system resources available to each user as part of the security domain of that user.

By doing so, you can prevent the uncontrolled consumption of valuable system resources such as CPU time.

This resource limit feature is very useful in large, multiuser systems, where system resources are very expensive. Excessive consumption of these resources by one or more users can detrimentally affect the other users of the database. In single-user or small-scale multiuser database systems, the system resource feature is not as important, because user consumption of system resources is less likely to have a detrimental impact.

You manage user resource limits by using Database Resource Manager. You can set password management preferences using profiles, either set individually or using a default profile for many users. Each Oracle database can have an unlimited number of profiles. Oracle Database allows the security administrator to enable or disable the enforcement of profile resource limits universally.

Setting resource limits causes a slight performance degradation when users create sessions, because Oracle Database loads all resource limit data for each user upon each connection to the database.

See Also:

Oracle Database Administrator’s Guide for information about setting resource limits

Types of System Resources and Limits

You can limit several types of system resources, including CPU time and logical reads, at the session level, call level, or both.

Limits to the User Session Level

When a user connects to a database, a session is created. Sessions use CPU time and memory, on which you can set limits.

You can set several resource limits at the session level. If a user exceeds a session-level resource limit, then Oracle Database terminates (rolls back) the current statement and returns a message indicating that the session limit has been reached. At this point, all previous statements in the current transaction are intact, and the only operations the user can perform are `COMMIT`, `ROLLBACK`, or disconnect (in this case, the current transaction is committed). All other operations produce an error. Even after the transaction is committed or rolled back, the user cannot accomplish any more work during the current session.

Limits to Database Call Levels

Each time a user runs a SQL statement, Oracle Database performs several steps to process the statement.

During the SQL statement processing, several calls are made to the database as a part of the different execution phases. To prevent any one call from using the system excessively, Oracle Database lets you set several resource limits at the call level.

If a user exceeds a call-level resource limit, then Oracle Database halts the processing of the statement, rolls back the statement, and returns an error. However, all previous statements of the current transaction remain intact, and the user session remains connected.

Limits to CPU Time

When SQL statements and other calls are made to an Oracle database, CPU time is necessary to process the call.

Average calls require a small amount of CPU time. However, a SQL statement involving a large amount of data or a runaway query can potentially use a large amount of CPU time, reducing CPU time available for other processing.

To prevent uncontrolled use of CPU time, you can set fixed or dynamic limits on the CPU time for each call and the total amount of CPU time used for Oracle Database calls during a session. The limits are set and measured in CPU one-hundredth seconds (0.01 seconds) used by a call or a session.

Limits to Logical Reads

Input/output (I/O) is one of the most expensive operations in a database system.

SQL statements that are I/O-intensive can monopolize memory and disk use and cause other database operations to compete for these resources.

To prevent single sources of excessive I/O, you can limit the logical data block reads for each call and for each session. Logical data block reads include data block reads from both memory and disk. The limits are set and measured in number of block reads performed by a call or during a session.

Limits to Other Resources

You can control limits for user concurrent sessions and idle time.

Limits to other resources are as follows:

- **You can limit the number of concurrent sessions for each user.** Each user can create only up to a predefined number of concurrent sessions.
- **You can limit the idle time for a session.** If the time between calls in a session reaches the idle time limit, then the current transaction is rolled back, the session is terminated, and the resources of the session are returned to the system. The next call receives an error that indicates that the user is no longer connected to the instance. This limit is set as a number of elapsed minutes.

> **Note:**
>
> Shortly after a session is terminated because it has exceeded an idle time limit, the process monitor (PMON) background process cleans up after the terminated session. Until PMON completes this process, the terminated session is still counted in any session or user resource limit.

- **You can limit the elapsed connect time for each session.** If the duration of a session exceeds the elapsed time limit, then the current transaction is rolled back, the session is dropped, and the resources of the session are returned to the system. This limit is set as a number of elapsed minutes.

> **Note:**
>
> Oracle Database does not constantly monitor the elapsed idle time or elapsed connection time. Doing so reduces system performance. Instead, it checks every few minutes. Therefore, a session can exceed this limit slightly (for example, by 5 minutes) before Oracle Database enforces the limit and terminates the session.

- **You can limit the amount of private System Global Area (SGA) space (used for private SQL areas) for a session.** This limit is only important in systems that use the shared server configuration. Otherwise, private SQL areas are located in the Program Global Area (PGA). This limit is set as a number of bytes of memory in the SGA of an instance. Use the characters **K** or **M** to specify kilobytes or megabytes.

> **See Also:**
>
> *Oracle Database Administrator’s Guide* for detailed information about managing resources

Values for Resource Limits of Profiles

Before you create profiles and set resource limits, you should determine appropriate values for each resource limit.

You can base the resource limit values on the type of operations a typical user performs. For example, if one class of user does not usually perform a high number of logical data block reads, then use the `ALTER RESOURCE COST` SQL statement to set the `LOGICAL_READS_PER_SESSION` setting conservatively.

Usually, the best way to determine the appropriate resource limit values for a given user profile is to gather historical information about each type of resource usage. For example, the database or security administrator can use the `AUDIT SESSION` clause to gather information about the limits `CONNECT_TIME`, `LOGICAL_READS_PER_SESSION`.

You can gather statistics for other limits using the Monitor feature of Oracle Enterprise Manager (or SQL*Plus), specifically the Statistics monitor.

Managing Resources with Profiles

A profile is a named set of resource limits and password parameters that restrict database usage and instance resources for a user.

About Profiles

A profile is a collection of attributes that apply to a user.

The **profile** is used to enable a single point of reference for multiple users who share these attributes.

You should assign a profile to each user. Each user can have only one profile, and creating a new one supersedes an earlier assignment.

You can create and manage user profiles only if resource limits are a requirement of your database security policy. To use profiles, first categorize the related types of users in a database. Just as roles are used to manage the privileges of related users, profiles are used to manage the resource limits of related users. Determine how many profiles are needed to encompass all categories of users in a database and then determine appropriate resource limits for each profile.

User profiles in Oracle Internet Directory contain attributes pertinent to directory usage and authentication for each user. Similarly, profiles in Oracle Label Security contain attributes useful in label security user administration and operations management. Profile attributes can include restrictions on system resources. You can use Database Resource Manager to set these types of resource limits.

In a multitenant environment, profiles are useful for the administration and operations performed in the container databases (CDBs) and application containers, as well as their associated pluggable databases (PDBs). For both CDB and application containers, if you define a common profile, then the profile applies to the entire container and not outside this container. If you create a local profile, then it applies to that PDB only.

Profile resource limits are enforced only when you enable resource limitation for the associated database. Enabling this limitation can occur either before starting the database (using the `RESOURCE_LIMIT` initialization parameter) or while it is open (using the `ALTER SYSTEM` statement).

Though password parameters reside in profiles, they are unaffected by `RESOURCE_LIMIT` or `ALTER SYSTEM` and password management is always enabled. In Oracle Database, Database Resource Manager primarily handles resource allocations and restrictions.

Any authorized database user can create, assign to users, alter, and drop a profile at any time (using the `CREATE USER` or `ALTER USER` statement). Profiles can be assigned only to users and not to roles or other profiles. Profile assignments do not affect current sessions; instead, they take effect only in subsequent sessions.

To find information about current profiles, query the `DBA_PROFILES` view.

See Also:

Oracle Database Administrator's Guide for detailed information about managing resources

ora_stig_profile User Profile

The `ora_stig_profile` user profile is designed for Security Technical Implementation Guides compliance.

The `ora_stig_profile` user profile addresses STIG requirements such as the need for a password complexity function, maximum failed login attempts, reuse time, and other requirements. The definition for this profile is as follows:

```
CREATE PROFILE ora_stig_profile
  password_life_time        60
  password_grace_time       5
  password_reuse_time       365
  password_reuse_max        10
  failed_login_attempts     3
  password_lock_time        unlimited
  inactive_account_time     35
  idle_time                 15
  password_verify_function  ora12c_stig_verify_function;
```

Creating a Profile

A profile can encompass limits for a specific category, such as limits on passwords or limits on resources.

To create a profile, you must have the `CREATE PROFILE` system privilege. To find all existing profiles, you can query the `DBA_PROFILES` view.

- Use the `CREATE PROFILE` statement to create a profile.

For example, to create a profile that defines password limits:

```
CREATE PROFILE password_prof LIMIT
  FAILED_LOGIN_ATTEMPTS 6
  PASSWORD_LIFE_TIME 60
  PASSWORD_REUSE_TIME 60
  PASSWORD_REUSE_MAX 5
  PASSWORD_LOCK_TIME 1/24
  PASSWORD_GRACE_TIME 10
  PASSWORD_VERIFY_FUNCTION DEFAULT;
```

The following example shows how to create a resource limits profile.

```
CREATE PROFILE app_user LIMIT
  SESSIONS_PER_USER          UNLIMITED
  CPU_PER_SESSION            UNLIMITED
  CPU_PER_CALL               3500
  CONNECT_TIME               50
  LOGICAL_READS_PER_SESSION  DEFAULT
  LOGICAL_READS_PER_CALL     1200
  PRIVATE_SGA                20K
  COMPOSITE_LIMIT            7500000;
```

See Also:

Oracle Database SQL Language Reference for more information about the `CREATE PROFILE` SQL statement

Creating a CDB Profile or an Application Profile

The `CREATE PROFILE` or `ALTER PROFILE` statement `CONTAINER=ALL` clause can create a profile in a CDB or application root.

You cannot create local profiles in the CDB root or the application root. The profile that you create will be applied to all PDBs that are associated with the CDB root or the application root. Create the profile using the same parameters that you would in a non-multitenant environment.

- To create a profile in a CDB root or an application root, optionally include the `CONTAINER=ALL` clause in the `CREATE PROFILE` or `ALTER PROFILE` statement.

  The `CONTAINER=ALL` clause is optional because it is the default when the statement is processed.

For example:

```
CREATE PROFILE password_prof LIMIT
  FAILED_LOGIN_ATTEMPTS 6
  PASSWORD_LIFE_TIME 60
  PASSWORD_REUSE_TIME 60
  PASSWORD_REUSE_MAX 5
  PASSWORD_LOCK_TIME 1/24
  PASSWORD_GRACE_TIME 10
  PASSWORD_VERIFY_FUNCTION DEFAULT
  CONTAINER=ALL;
```

Assigning a Profile to a User

After you create a profile, you can assign it to users.

You can assign a profile to a user who has already been assigned a profile, but the most recently assigned profile takes precedence. When you assign a profile to an external user or a global user, the password parameters do not take effect for that user.

To find the profiles that are currently assigned to users, you can query the `DBA_USERS` view.

- Use the `ALTER USER` statement to assign the profile to a user.

For example:

```
ALTER USER psmith PROFILE app_user;
```

Dropping Profiles

You can drop a profile, even if it is currently assigned to a user.

When you drop a profile, the drop does not affect currently active sessions. Only sessions that were created after a profile is dropped use the modified profile assignments. To drop a profile, you must have the `DROP PROFILE` system privilege. You cannot drop the default profile.

- Use the SQL statement `DROP PROFILE` to drop a profile. To drop a profile that is currently assigned to a user, use the `CASCADE` option.

For example:

```
DROP PROFILE clerk CASCADE;
```

Any user currently assigned to a profile that is dropped is automatically is assigned to the `DEFAULT` profile. The `DEFAULT` profile cannot be dropped.

See Also:

Oracle Database SQL Language Reference for more information about the `DROP PROFILE` SQL statement

Dropping User Accounts

You can drop user accounts if the user is not in a session, and if the user has objects in the user's schema.

About Dropping User Accounts

Before you drop a user account, you must ensure that you have the appropriate privileges for doing so.

To drop a user account in any environment, you must have the `DROP USER` system privilege. In a multitenant environment, you must have the commonly granted `DROP USER` system privilege to drop common user accounts. To drop local user accounts, you must have a commonly granted `DROP USER` privilege or a locally granted `DROP USER` privilege in the PDB in which the local user account resides.

When you drop a user account, Oracle Database removes the user account and associated schema from the data dictionary. It also immediately drops all schema objects contained in the user schema, if any.

Note:

- If a user schema and associated objects must remain but the user must be denied access to the database, then revoke the `CREATE SESSION` privilege from the user.
- Do not attempt to drop the `SYS` or `SYSTEM` user. Doing so corrupts your database.

Terminating a User Session

A user who is connected to a database cannot be dropped.

You must first terminate the user session (or the user can exit the session) before you can drop the user.

1. Query the `V$SESSION` dynamic view to find the session ID of the user whose session you want to terminate.

   For example:

   ```
   SELECT SID, SERIAL#, USERNAME FROM V$SESSION;

       SID         SERIAL# USERNAME
   ------- --------------- ----------------------
       127          55234  ANDY
   ...
   ```

2. Use the `ALTER SYSTEM` SQL statement to stop the session for the user, based on the `SID` and `SERIAL#` settings of the `V$SESSION` view.

   For example:

   ```
   ALTER SYSTEM KILL SESSION '127, 55234';
   ```

See Also:

Oracle Database Administrator’s Guide for more information about terminating sessions

About Dropping a User After the User Is No Longer Connected to the Database

After a user is disconnected from the database, you can use the `DROP USER` statement to drop the user.

To drop a user and all the user schema objects (if any), you must have the `DROP USER` system privilege. Because the `DROP USER` system privilege is powerful, a security administrator is typically the only type of user that has this privilege.

If the schema of the user contains any dependent schema objects, then use the `CASCADE` option to drop the user and all associated objects and foreign keys that depend on the tables of the user successfully. If you do not specify `CASCADE` and the user schema contains dependent objects, then an error message is returned and the user is not dropped.

Dropping a User Whose Schema Contains Objects

Before you drop a user whose schema contains objects, carefully investigate the implications of dropping these schema objects.

1. Query the `DBA_OBJECTS` data dictionary view to find the objects that are owned by the user.

For example:

```
SELECT OWNER, OBJECT_NAME FROM DBA_OBJECTS WHERE OWNER LIKE 'ANDY';
```

Enter the user name in capital letters. Pay attention to any unknown cascading effects. For example, if you intend to drop a user who owns a table, then check whether any views or procedures depend on that particular table.

2. Use the `DROP USER` SQL statement with the `CASCADE` clause to drop the user and all associated objects and foreign keys that depend on the tables that the user owns.

   For example:

   ```
   DROP USER andy CASCADE;
   ```

Database User and Profile Data Dictionary Views

Oracle Database provides a set of data dictionary views that provide information about the settings that you used to create users and profiles.

Data Dictionary Views That List Information About Users and Profiles

Oracle Database provides a set of data dictionary views that contain information about database users and profiles.

Table 2-1 lists these data dictionary views. For detailed information about these views, see *Oracle Database Reference*.

Table 2-1 Data Dictionary Views That Display Information about Users and Profiles

View	Description
`ALL_OBJECTS`	Describes all objects accessible to the current user
`ALL_USERS`	Lists users visible to the current user, but does not describe them
`DBA_PROFILES`	Displays all profiles and their limits
`DBA_TS_QUOTAS`	Describes tablespace quotas for users
`DBA_OBJECTS`	Describes all objects in the database
`DBA_USERS`	Describes all users of the database
`DBA_USERS_WITH_DEFPWD`	Lists all user accounts that have default passwords
`PROXY_USERS`	Describes users who can assume the identity of other users
`RESOURCE_COST`	Lists the cost for each resource in terms of CPUs for each session, reads for each session, connection times, and SGA
`USER_PASSWORD_LIMITS`	Describes the password profile parameters that are assigned to the user
`USER_RESOURCE_LIMITS`	Displays the resource limits for the current user
`USER_TS_QUOTAS`	Describes tablespace quotas for users
`USER_OBJECTS`	Describes all objects owned by the current user
`USER_USERS`	Describes only the current user
`V$SESSION`	Lists session information for the current database session

Table 2-1 (Cont.) Data Dictionary Views That Display Information about Users and Profiles

View	Description
`V$SESSTAT`	Displays user session statistics
`V$STATNAME`	Displays decoded statistic names for the statistics shown in the `V$SESSTAT` view

The following sections present examples of using these views. These examples assume that the following statements have been run. The users are all local users.

```
CREATE PROFILE clerk LIMIT
 SESSIONS_PER_USER 1
 IDLE_TIME 30
 CONNECT_TIME 600;

CREATE USER jfee
 IDENTIFIED BY password
 DEFAULT TABLESPACE example
 TEMPORARY TABLESPACE temp
 QUOTA 500K ON example
 PROFILE clerk
    CONTAINER = CURRENT;

CREATE USER dcranney
 IDENTIFIED BY password
 DEFAULT TABLESPACE example
 TEMPORARY TABLESPACE temp
 QUOTA unlimited ON example
 CONTAINER = CURRENT;

CREATE USER userscott
 IDENTIFIED BY password
 CONTAINER = CURRENT;
```

Query to Find All Users and Associated Information

The `DBA_USERS` data dictionary view shows all users and their associated information as defined in the database.

For detailed information about the `DBA_USERS` view, see *Oracle Database Reference*.

For example:

```
col username format a11
col profile format a10
col account_status format a19
col authentication_type format a29

SELECT USERNAME, PROFILE, ACCOUNT_STATUS, AUTHENTICATION_TYPE FROM DBA_USERS;

USERNAME        PROFILE         ACCOUNT_STATUS   AUTHENTICATION_TYPE
--------------- --------------- ---------------  -------------------
SYS             DEFAULT         OPEN             PASSWORD
SYSTEM          DEFAULT         OPEN             PASSWORD
USERSCOTT       DEFAULT         OPEN             PASSWORD
```

```
JFEE            CLERK           OPEN             GLOBAL
DCRANNEY        DEFAULT         OPEN             EXTERNAL
```

Query to List All Tablespace Quotas

The `DBA_TS_QUOTAS` data dictionary view lists all tablespace quotas assigned to each user.

For detailed information about this view, see *Oracle Database Reference*.

For example:

```
SELECT * FROM DBA_TS_QUOTAS;

TABLESPACE    USERNAME    BYTES     MAX_BYTES    BLOCKS    MAX_BLOCKS
----------    ---------  --------   ----------   -------   ----------
EXAMPLE       JFEE              0       512000         0          250
EXAMPLE       DCRANNEY          0           -1         0           -1
```

When specific quotas are assigned, the exact number is indicated in the `MAX_BYTES` column. This number is always a multiple of the database block size, so if you specify a tablespace quota that is not a multiple of the database block size, then it is rounded up accordingly. Unlimited quotas are indicated by `-1`.

Query to List All Profiles and Assigned Limits

The `DBA_PROFILE` view lists all profiles in the database and associated settings for each limit in each profile.

For detailed information about this view, see *Oracle Database Reference*.

For example:

```
SELECT * FROM DBA_PROFILES
   ORDER BY PROFILE;

PROFILE             RESOURCE_NAME              RESOURCE_TYPE   LIMIT
-----------------   -----------------------    -------------   --------------
CLERK               COMPOSITE_LIMIT            KERNEL          DEFAULT
CLERK               FAILED_LOGIN_ATTEMPTS      PASSWORD        DEFAULT
CLERK               PASSWORD_LIFE_TIME         PASSWORD        DEFAULT
CLERK               PASSWORD_REUSE_TIME        PASSWORD        DEFAULT
CLERK               PASSWORD_REUSE_MAX         PASSWORD        DEFAULT
CLERK               PASSWORD_VERIFY_FUNCTION   PASSWORD        DEFAULT
CLERK               PASSWORD_LOCK_TIME         PASSWORD        DEFAULT
CLERK               PASSWORD_GRACE_TIME        PASSWORD        DEFAULT
CLERK               PRIVATE_SGA                KERNEL          DEFAULT
CLERK               CONNECT_TIME               KERNEL          600
CLERK               IDLE_TIME                  KERNEL          30
CLERK               LOGICAL_READS_PER_CALL     KERNEL          DEFAULT
CLERK               LOGICAL_READS_PER_SESSION  KERNEL          DEFAULT
CLERK               CPU_PER_CALL               KERNEL          DEFAULT
CLERK               CPU_PER_SESSION            KERNEL          DEFAULT
CLERK               SESSIONS_PER_USER          KERNEL          1
DEFAULT             COMPOSITE_LIMIT            KERNEL          UNLIMITED
DEFAULT             PRIVATE_SGA                KERNEL          UNLIMITED
DEFAULT             SESSIONS_PER_USER          KERNEL          UNLIMITED
DEFAULT             CPU_PER_CALL               KERNEL          UNLIMITED
DEFAULT             LOGICAL_READS_PER_CALL     KERNEL          UNLIMITED
DEFAULT             CONNECT_TIME               KERNEL          UNLIMITED
DEFAULT             IDLE_TIME                  KERNEL        UNLIMITED
DEFAULT             LOGICAL_READS_PER_SESSION  KERNEL        UNLIMITED
DEFAULT             CPU_PER_SESSION            KERNEL        UNLIMITED
DEFAULT             FAILED_LOGIN_ATTEMPTS      PASSWORD      10
DEFAULT             PASSWORD_LIFE_TIME         PASSWORD      180
DEFAULT             PASSWORD_REUSE_MAX         PASSWORD      UNLIMITED
DEFAULT             PASSWORD_LOCK_TIME         PASSWORD      1
DEFAULT             PASSWORD_GRACE_TIME        PASSWORD      7
DEFAULT             PASSWORD_VERIFY_FUNCTION   PASSWORD      UNLIMITED
DEFAULT             PASSWORD_REUSE_TIME        PASSWORD      UNLIMITED
32 rows selected.
```

To find the default profile values, you can run the following query:

```
SELECT * FROM DBA_PROFILES WHERE PROFILE = 'DEFAULT';

PROFILE             RESOURCE_NAME              RESOURCE_TYPE  LIMIT
-----------------   -------------------------  -------------  -------------
DEFAULT             COMPOSITE_LIMIT            KERNEL        UNLIMITED
DEFAULT             SESSIONS_PER_USER          KERNEL        UNLIMITED
DEFAULT             CPU_PER_SESSION            KERNEL        UNLIMITED
DEFAULT             CPU_PER_CALL               KERNEL        UNLIMITED
DEFAULT             LOGICAL_READS_PER_SESSION  KERNEL        UNLIMITED
DEFAULT             LOGICAL_READS_PER_CALL     KERNEL        UNLIMITED
DEFAULT             IDLE_TIME                  KERNEL        UNLIMITED
DEFAULT             CONNECT_TIME               KERNEL        UNLIMITED
DEFAULT             PRIVATE_SGA                KERNEL        UNLIMITED
DEFAULT             FAILED_LOGIN_ATTEMPTS      PASSWORD      10
DEFAULT             PASSWORD_LIFE_TIME         PASSWORD      180
DEFAULT             PASSWORD_REUSE_TIME        PASSWORD      UNLIMITED
DEFAULT             PASSWORD_REUSE_MAX         PASSWORD      UNLIMITED
DEFAULT             PASSWORD_VERIFY_FUNCTION   PASSWORD      NULL
DEFAULT             PASSWORD_LOCK_TIME         PASSWORD      1
DEFAULT             PASSWORD_GRACE_TIME        PASSWORD      7

16 rows selected.
```

Query to View Memory Use for Each User Session

The `V$SESSION` dynamic view lists the memory use for each user session.

For detailed information on this view, see *Oracle Database Reference*.

The following query lists all current sessions, showing the Oracle Database user and current User Global Area (UGA) memory use for each session:

```
SELECT USERNAME, VALUE || 'bytes' "Current UGA memory"
   FROM V$SESSION sess, V$SESSTAT stat, V$STATNAME name
WHERE sess.SID = stat.SID
   AND stat.STATISTIC# = name.STATISTIC#
   AND name.NAME = 'session uga memory';

USERNAME                       Current UGA memory
------------------------------ ---------------------------------------------
                               18636bytes
                               17464bytes
                               19180bytes
                               18364bytes
                               39384bytes
                               35292bytes
                               17696bytes
```

```
                              15868bytes
USERSCOTT                     42244bytes
SYS                           98196bytes
SYSTEM                        30648bytes

11 rows selected.
```

To see the maximum UGA memory allocated to each session since the instance started, replace `'session uga memory'` in the preceding query with `'session uga memory max'`.

3

Configuring Authentication

Authentication means to verify the identity of users or other entities that connect to the database.

About Authentication

Authentication means verifying the identity of a user, device, or other entity who wants to use data, resources, or applications.

Validating this identity establishes a trust relationship for further interactions. Authentication also enables accountability by making it possible to link access and actions to specific identities. After authentication, authorization processes can allow or limit the levels of access and action permitted to that entity.

You can authenticate both database and nondatabase users for an Oracle database. For simplicity, the same authentication method is generally used for all database users, but Oracle Database allows a single database instance to use any or all methods. Oracle Database requires special authentication procedures for database administrators, because they perform special database operations. Oracle Database also encrypts passwords during transmission to ensure the security of network authentication.

After authentication, authorization processes can allow or limit the levels of access and action permitted to that entity.

Configuring Password Protection

You can secure user passwords in a variety of ways, such as controlling the password creation requirements or using password management policies.

What Are the Oracle Database Built-in Password Protections?

Oracle Database provides a set of built-in password protections designed to protect your users' passwords.

These password protections are as follows:

- **Password encryption.** Oracle Database automatically and transparently encrypts passwords during network (client-to-server and server-to-server) connections, using Advanced Encryption Standard (AES) before sending them across the network. However, a password that is specified within a SQL statement (such as `CREATE USER` *`user_name`* `IDENTIFIED BY` *`password`*`;`) is still transmitted across the network in clear text in the network trace files. For this reason, you should have native network encryption enabled or configure Secure Sockets Layer (SSL) encryption.
- **Password complexity checking.** In a default installation, Oracle Database provides the `ora12c_verify_function` and `ora12c_strong_verify_function` password verification functions to ensure that new or changed passwords are sufficiently

complex to prevent intruders who try to break into the system by guessing passwords. You must manually enable password complexity checking. You can further customize the complexity of your users' passwords. See About Password Complexity Verification for more information.

- **Preventing passwords from being broken.** If a user tries to log in to Oracle Database multiple times using an incorrect password, Oracle Database delays each login by one second. This protection applies for attempts made from different IP addresses or multiple client connections. This feature significantly decreases the number of passwords that an intruder would be able to try within a fixed time period when attempting to log in. The failed login delay slows down each failed login attempt, increasing the overall time that is required to perform a password-guessing attack, because such attacks usually require a very large number of failed login attempts.

  For non-administrative logins, Oracle Database protects against concurrent password guessing attacks by setting an exclusive lock for the failed login delay. This prevents an intruder from attempting to sidestep the failed login delay when the intruder tries the next concurrent guess in a different database session as soon as the first guess fails and is delayed.

  By holding an exclusive lock on the account that is being attacked, Oracle Database mitigates concurrent password guessing attacks, but this can simultaneously leave the account vulnerable to denial-of-service (DoS) attacks. To remedy this problem, you should create a password profile where the `FAILED_LOGIN_ATTEMPTS` parameter is set to `UNLIMITED`, and then apply this password profile to the user account. The value `UNLIMITED` for the `FAILED_LOGIN_ATTEMPTS` parameter setting disables failed login delays and does not limit the number of failed login attempts. For these types of accounts, Oracle recommends that you use a long random password.

  The concurrent password-guessing attack protection does not apply to administrative user connections, because these kinds of connections must remain available at all times and be immune to denial-of-service attacks. Hence, Oracle recommends that you choose long passwords for any administrative privileged account.

- **Enforced case sensitivity for passwords.** Passwords are case sensitive. For example, the password `hPP5620qr` fails if it is entered as `hpp5620QR` or `hPp5620Qr`. See Managing Password Case Sensitivity for information about how case sensitivity works, and how it affects password files and database links.

- **Passwords hashed using the 12C password version.** To verify the user's password and enforce case sensitivity in password creation, Oracle Database uses the `12C` password version, which is based on a de-optimized algorithm that involves Password-Based Key Derivation Function (PBKDF2) and the SHA-512 cryptographic hash functions. See Ensuring Against Password Security Threats by Using the 12C Password Version for more information.

See Also:

Guidelines for Securing Passwords for advice about securing passwords

Minimum Requirements for Passwords

Oracle provides a set of minimum requirements for passwords.

Passwords can be at most 30 bytes long. There are a variety of ways that you can secure passwords, ranging from requiring passwords to be of a sensible length to creating custom password complexity verification scripts that enforce the password complexity policy requirements that apply at your site.

Creating a Password by Using the IDENTIFIED BY Clause

SQL statements that accept the `IDENTIFIED BY` clause also enable you to create passwords.

- To create passwords for users, use the `CREATE USER`, `ALTER USER`, `GRANT CREATE SESSION`, or `CREATE DATABASE LINK` SQL statement.

The following SQL statements create passwords with the `IDENTIFIED BY` clause.

```
CREATE USER psmith IDENTIFIED BY password;
GRANT CREATE SESSION TO psmith IDENTIFIED BY password;
ALTER USER psmith IDENTIFIED BY password;
CREATE DATABASE LINK AUTHENTICATED BY psmith IDENTIFIED BY password;
```

Using a Password Management Policy

A password management policy can create and enforce a set of restrictions that can better secure user passwords.

About Managing Passwords

Database security systems that depend on passwords require that passwords be kept secret at all times.

Because passwords are vulnerable to theft and misuse, Oracle Database uses a password management policy. Database administrators and security officers control this policy through user profiles, enabling greater control of database security.

You can use the `CREATE PROFILE` statement to create a user profile. The profile is assigned to a user with the `CREATE USER` or `ALTER USER` statement.

Finding User Accounts That Have Default Passwords

The `DBA_USERS_WITH_DEFPWD` data dictionary view can find user accounts that use default passwords.

When you create a database, most of the default accounts are locked with the passwords expired. If you have upgraded from an earlier release of Oracle Database, then you may have user accounts that have default passwords. These are default accounts that are created when you create a database, such as the `HR`, `OE`, and `SCOTT` accounts.

For greater security, you should change the passwords for these accounts. Using a default password that is commonly known can make your database vulnerable to attacks by intruders.

1. Log in to the database instance using SQL*Plus with the `SYSDBA` administrative privilege.

   For example:

```
sqlplus sys as sysdba
Enter password: password
```

2. Query the `DBA_USERS_WITH_DEFPWD` data dictionary view.

   For example, to find both the names of accounts that have default passwords and the status of the account:

   ```
   SELECT d.username, u.account_status
   FROM DBA_USERS_WITH_DEFPWD d, DBA_USERS u
   WHERE d.username = u.username
   ORDER BY 2,1;

   USERNAME  ACCOUNT_STATUS
   --------- ---------------------------
   SCOTT     EXPIRED & LOCKED
   ```

3. Change the passwords for any accounts that the `DBA_USERS_WITH_DEFPWD` view lists.

   Oracle recommends that you do **not** assign these accounts passwords that they may have had in previous releases of Oracle Database.

   For example:

   ```
   ALTER USER SCOTT ACCOUNT UNLOCK IDENTIFIED BY password;
   ```

   Follow the guidelines in Minimum Requirements for Passwords to replace *`password`* with a password that is secure.

Password Settings in the Default Profile

A profile is a collection of parameters that sets limits on database resources.

If you assign the profile to a user, then that user cannot exceed these limits. You can use profiles to configure database settings such as sessions per user, logging and tracing features, and so on. Profiles can also control user passwords. To find information about the current password settings in the profile, you can query the `DBA_PROFILES` data dictionary view.

Table 3-1 lists the password-specific parameter settings in the default profile.

Table 3-1 Password-Specific Settings in the Default Profile

Parameter	Default Setting	Description
`INACTIVE_ACCOUNT_TIME`	`UNLIMITED`	Locks the account of a database user who has not logged in to the database instance in a specified number of days. See Automatically Locking Inactive Database User Accounts for more information.

Table 3-1 (Cont.) Password-Specific Settings in the Default Profile

Parameter	Default Setting	Description
`FAILED_LOGIN_ATTEMPTS`	`10`	Sets the maximum times a user try to log in and to fail before locking the account. **Notes:** • When you set this parameter, take into consideration users who may log in using the `CONNECT THROUGH` privilege. • You can set limits on the number of times an unauthorized user (possibly an intruder) attempts to log in to Oracle Call Interface (OCI) applications by using the `SEC_MAX_FAILED_LOGIN_ATTEMPTS` initialization parameter. See Configuration of the Maximum Number of Authentication Attempts for more information about this parameter. See also Automatically Locking User Accounts After Failed Logins for more information.
`PASSWORD_GRACE_TIME`	`7`	Sets the number of days that a user has to change his or her password before it expires. See About Controlling Password Aging and Expiration for more information.
`PASSWORD_LIFE_TIME`	`180`	Sets the number of days the user can use his or her current password. See About Controlling Password Aging and Expiration for more information.
`PASSWORD_LOCK_TIME`	`1`	Sets the number of days an account will be locked after the specified number of consecutive failed login attempts. After the time passes, then the account becomes unlocked. This user's profile parameter is useful to help prevent brute force attacks on user passwords but not to increase the maintenance burden on administrators. See Automatically Locking User Accounts After Failed Logins for more information.
`PASSWORD_REUSE_MAX`	`UNLIMITED`	Sets the number of password changes required before the current password can be reused. See Controlling the User Ability to Reuse Previous Passwords for more information.
`PASSWORD_REUSE_TIME`	`UNLIMITED`	Sets the number of days before which a password cannot be reused. See Controlling the User Ability to Reuse Previous Passwords for more information.

Using the ALTER PROFILE Statement to Set Profile Limits

You can modify profile limits such as failed login attempts, password lock times, password reuse, and several other settings.

These settings are described in Table 3-1. For greater security, use the default settings that are described in this table, based on your needs.

- Use the `ALTER PROFILE` statement to modify a user's profile limits.

For example:

```
ALTER PROFILE prof LIMIT
 FAILED_LOGIN_ATTEMPTS 9
 PASSWORD_LOCK_TIME 10
 INACTIVE_ACCOUNT_TIME 21;
```

Disabling and Enabling the Default Password Security Settings

Oracle provides scripts that you can use to disable and enable the default password security settings.

If your applications use the default password security settings from Oracle Database 10*g* release 2 (10.2), then you can revert to these settings until you modify the applications to use the default password security settings from Oracle Database 11*g* or later.

1. Modify your applications to conform to the password security settings from Oracle Database 11*g* or later.
2. Update your database to use the security configuration that suits your business needs, using one of the following methods:
   - Manually update the database security configuration.
   - Run the `secconf.sql` script to apply the default password settings from Oracle Database 11*g* or later. You can customize this script to have different security settings if you like, but remember that the settings listed in the original script are Oracle-recommended settings.

If you created your database manually, then you should run the `secconf.sql` script to apply the Oracle default password settings to the database. Databases that have been created with Database Configuration Assistant (DBCA) will have these settings, but manually created databases do not.

The `secconf.sql` script is in the `$ORACLE_HOME/rdbms/admin` directory. The `secconf.sql` script affects both password and audit settings. It has no effect on other security settings.

Automatically Locking Inactive Database User Accounts

The `INACTIVE_ACCOUNT_TIME` profile parameter locks a user account that has not logged in to the database instance in a specified number of days.

Users are considered active users if they log in periodically. The `INACTIVE_ACCOUNT_TIME` timing is based on the number of days after the last time a user successfully logs in.

- To lock user accounts automatically after a specified number of days, set the `INACTIVE_ACCOUNT_TIME` profile parameter in the `CREATE PROFILE` or `ALTER PROFILE` statement.

  Note the following:

  - The default value for `INACTIVE_ACCOUNT_TIME` is `UNLIMITED`.
  - You must specify a whole number for the number of days. The minimum setting is `15` and the maximum is `24855`.
  - To set the user’s account to have an unlimited inactivity time, set the `INACTIVE_ACCOUNT_TIME` to `UNLIMITED`.
  - To set the user’s account to use the time specified by the default profile, set `INACTIVE_ACCOUNT_TIME` to `DEFAULT`.
  - You can set this parameter for all database authenticated users, including administrative users, but not for external or global authenticated users.
  - In a read-only database, the last successful login is not considered in the `INACTIVE_ACCOUNT_TIME` timing. It is not possible to lock a user account in a read-only database (except by performing consecutive failed logins equal in number to the account’s `FAILED_LOGIN_ATTEMPTS` password profile setting).
  - For a newly created user account, the timing begins at account creation time. When this user logs out and then logs again, the timing starts when the user successfully logs in.
  - In a multitenant environment, the `INACTIVE_ACCOUNT_TIME` setting applies to the last time a common user logs in to the root. A common user is considered active if this user logs in to any of the PDBs or the root.
  - For a proxy user account login, the `INACTIVE_ACCOUNT_TIME` begins the timing when the proxy user logs in successfully.

For example, to create a profile that locks an account after 60 days of being inactive:

```
CREATE PROFILE time_limit LIMIT
 INACTIVE_ACCOUNT_TIME 60;
```

Automatically Locking User Accounts After Failed Logins

Oracle Database can lock a user's account after a specified number of consecutive failed log-in attempts.

- To lock user accounts automatically after a specified time interval or to require database administrator intervention to be unlocked, set the `PASSWORD_LOCK_TIME` profile parameter in the `CREATE PROFILE` or `ALTER PROFILE` statement.

  For example, to set the time interval to 10 days:

  ```
  PASSWORD_LOCK_TIME = 10
  ```

Note the following:

- You can lock accounts manually, so that they must be unlocked explicitly by a database administrator.
- You can specify the permissible number of failed login attempts by using the `CREATE PROFILE` statement. You can also specify the amount of time an account remains locked.

- Each time the user unsuccessfully logs in, Oracle Database increases the delay exponentially with each login failure.
- If you do not specify a time interval for unlocking the account, then `PASSWORD_LOCK_TIME` assumes the value specified in a default profile. (The recommended value is 1 day.) If you specify `PASSWORD_LOCK_TIME` as `UNLIMITED`, then you must explicitly unlock the account by using an `ALTER USER` statement. For example, assuming that `PASSWORD_LOCK_TIME UNLIMITED` is specified for `johndoe`, then you use the following statement to unlock the `johndoe` account:

  ```
  ALTER USER johndoe ACCOUNT UNLOCK;
  ```
- After a user successfully logs into an account, Oracle Database resets the unsuccessful login attempt count for the user. If it is non-zero, then the count is set to zero.
- In a multitenant environment, a locked CDB common user account will be locked across all PDBs in the CDB. A locked application common user account will be locked across all PDBs that are associated with the application root.

Example: Locking an Account with the CREATE PROFILE Statement

The `CREATE PROFILE` statement can lock user accounts if a user's attempt to log in violates the `CREATE PROFILE` settings.

Example 3-1 sets the maximum number of failed login attempts for the user `johndoe` to 10 (the default), and the amount of time the account locked to 30 days. The account will unlock automatically after 30 days.

Example 3-1 Locking an Account with the CREATE PROFILE Statement

```
CREATE PROFILE prof LIMIT
 FAILED_LOGIN_ATTEMPTS 10
 PASSWORD_GRACE_TIME 3

ALTER USER johndoe PROFILE prof;
```

Explicitly Locking a User Account

When you explicitly lock a user account, the account cannot be unlocked automatically. Only a security administrator can unlock the account.

In a multitenant environment, after you have locked a CDB common user account in the CDB root, this user cannot log in to any PDB that is associated with this root, nor can this account be unlocked in a PDB. In addition, you can lock a CDB common account locally in a PDB, which will prevent the CDB common user from logging in to that PDB. Similarly, an application common user account that is locked in the application root cannot log in to any PDB associated with the application root, nor can the application common user be unlocked in an application PDB. You can explicitly lock an application common user locally in an application PDB.

- To explicitly lock a user account, use the `CREATE USER` or `ALTER USER` statement.

For example, the following statement locks the user account, `susan`:

```
ALTER USER susan ACCOUNT LOCK;
```

Controlling the User Ability to Reuse Previous Passwords

You can ensure that users do not reuse previous passwords for an amount of time or for a number of password changes.

- To ensure that users cannot reuse their passwords for a specified period of time, configure the rules for password reuse with the `CREATE PROFILE` or `ALTER PROFILE` statements.

Table 3-2 lists the `CREATE PROFILE` and `ALTER PROFILE` parameters that control ability of a user to reuse a previous password.

Table 3-2 Parameters Controlling Reuse of a Previous Password

Parameter Name	Description and Use
`PASSWORD_REUSE_TIME`	Requires either of the following: • A number specifying how many days (or a fraction of a day) between the earlier use of a password and its next use • The word `UNLIMITED`
`PASSWORD_REUSE_MAX`	Requires either of the following: • An integer to specify the number of password changes required before a password can be reused • The word `UNLIMITED`

If you do not specify a parameter, then the user can reuse passwords at any time, which is not a good security practice.

If neither parameter is `UNLIMITED`, then password reuse is allowed, but only after meeting both conditions. The user must have changed the password the specified number of times, and the specified number of days must have passed since the previous password was last used.

For example, suppose that the profile of user A had `PASSWORD_REUSE_MAX` set to `10` and `PASSWORD_REUSE_TIME` set to `30`. User A cannot reuse a password until he or she has reset the password 10 times, and until 30 days had passed since the password was last used.

If either parameter is specified as `UNLIMITED`, then the user can never reuse a password.

If you set both parameters to `UNLIMITED`, then Oracle Database ignores both, and the user can reuse any password at any time.

Note:

If you specify `DEFAULT` for either parameter, then Oracle Database uses the value defined in the `DEFAULT` profile, which sets all parameters to `UNLIMITED`. Oracle Database thus uses `UNLIMITED` for any parameter specified as `DEFAULT`, unless you change the setting for that parameter in the `DEFAULT` profile.

See Also:

- *Oracle Database SQL Language Reference* for more information about the `CREATE PROFILE` statement
- *Oracle Database SQL Language Reference* for more information about the `ALTER PROFILE` statement

About Controlling Password Aging and Expiration

You can specify a password lifetime, after which the password expires.

This means that the next time the user logs in with the current, correct password, he or she is prompted to change the password. By default, there are no complexity or password history checks, so users can still reuse any previous or weak passwords. You can control these factors by setting the `PASSWORD_REUSE_TIME`, `PASSWORD_REUSE_MAX`, and `PASSWORD_VERIFY_FUNCTION` parameters.

In addition, you can set a grace period, during which each attempt to log in to the database account receives a warning message to change the password. If the user does not change it by the end of that period, then Oracle Database expires the account.

As a database administrator, you can manually set the password state to be expired, which sets the account status to `EXPIRED`. The user must then follow the prompts to change the password before the logon can proceed.

For example, in SQL*Plus, suppose user `SCOTT` tries to log in with the correct credentials, but his password has expired. User `SCOTT` will then see the `ORA-28001: The password has expired` error and be prompted to change his password, as follows:

```
Changing password for scott
New password: new_password
Retype new password: new_password
Password changed.
```

Using the CREATE PROFILE or ALTER PROFILE Statement to Set a Password Lifetime

When you set a lifetime for a password, the user must create a new password when this lifetime ends.

- Use the `CREATE PROFILE` or `ALTER PROFILE` statement to specify a lifetime for passwords.

The following example demonstrates how to create and assign a profile to user `johndoe`, and the `PASSWORD_LIFE_TIME` clause specifies that `johndoe` can use the same password for 180 days before it expires.

```
CREATE PROFILE prof LIMIT
 FAILED_LOGIN_ATTEMPTS 4
 PASSWORD_GRACE_TIME 3
 PASSWORD_LIFE_TIME 180;
ALTER USER johndoe PROFILE prof;
```

Checking the Status of a User Account

You can check the status of any account, whether it is open, in grace, or expired.

- To check the status of a user account, query the `ACCOUNT_STATUS` column of the `DBA_USERS` data dictionary view.

For example:

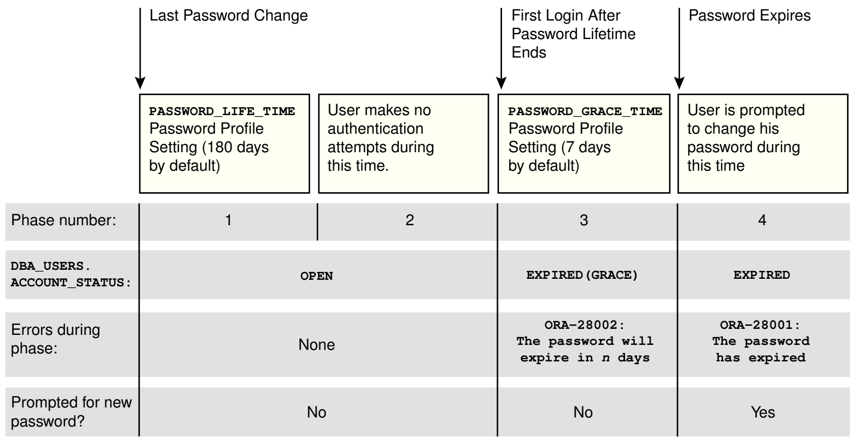
```
SELECT ACCOUNT_STATUS FROM DBA_USERS WHERE USERNAME = 'username';
```

Password Change Life Cycle

After a password is created, it follows a lifecycle and grace period in four phases.

Figure 3-1 shows the life cycle of the password lifetime and grace period.

Figure 3-1 Password Change Life Cycle


Last Password Change
First Login After Password Lifetime Ends
Password Expires
PASSWORD_LIFE_TIME Password Profile Setting (180 days by default)
User makes no authentication attempts during this time.
PASSWORD_GRACE_TIME Password Profile Setting (7 days by default)
User is prompted to change his password during this time
Phase number:
1
2
3
4
DBA_USERS. ACCOUNT_STATUS:
OPEN
EXPIRED(GRACE)
EXPIRED
Errors during phase:
None
ORA-28002: The password will expire in n days
ORA-28001: The password has expired
Prompted for new password?
No
No
Yes

In this figure:

- **Phase 1:** After the user account is created, or the password of an existing account is changed, the password lifetime period begins.
- **Phase 2:** This phase represents the period of time *after* the password lifetime ends but *before* the user logs in again with the correct password. The correct credentials are needed for Oracle Database to update the account status. Otherwise, the account status will remain unchanged. Oracle Database does not have any background process to update the account status. All changes to the account status are driven by the Oracle Database server process on behalf of authenticated users.
- **Phase 3:** When the user finally does log in, the grace period begins. Oracle Database then updates the `DBA_USERS.EXPIRY_DATE` column to a new value using the current time plus the value of the `PASSWORD_GRACE_TIME` setting from the account's password profile. At this point, the user receives an `ORA-28002` warning message about the password expiring in the near future (for example, `ORA-28002 The password will expire within 7 days` if `PASSWORD_GRACE_TIME` is set to `7` days), but the user can still log in without changing the password. The

`DBA_USERS.EXPIRY_DATE` column shows the time in the future when the user will be prompted to change their password.

- **Phase 4:** After the grace period (Phase 3) ends, the `ORA-28001: The password has expired` error appears, and the user is prompted to change the password after entering the current, correct password before the authentication can proceed. If the user has an Oracle Active Data Guard configuration, where there is a primary and a stand-by database, and the authentication attempt is made on the standby database (which is a read-only database), then the `ORA-28032: Your password has expired and the database is set to read-only` error appears. The user should log into the primary database and change the password there.

During any of these four phases, you can query the `DBA_USERS` data dictionary view to find the user's account status in the `DBA_USERS.ACCOUNT_STATUS` column.

In the following example, the profile assigned to `johndoe` includes the specification of a grace period: `PASSWORD_GRACE_TIME = 3` (the recommended value). The first time `johndoe` tries to log in to the database after 90 days (this can be *any* day after the 90th day, that is, the 91st day, 100th day, or another day), he receives a warning message that his password will expire in 3 days. If 3 days pass, and if he does not change his password, then the password expires. After this, he receives a prompt to change his password on any attempt to log in.

```
CREATE PROFILE prof LIMIT
 FAILED_LOGIN_ATTEMPTS 4
 PASSWORD_LIFE_TIME 90
 PASSWORD_GRACE_TIME 3;

ALTER USER johndoe PROFILE prof;
```

A database administrator or a user who has the `ALTER USER` system privilege can explicitly expire a password by using the `CREATE USER` and `ALTER USER` statements. The following statement creates a user with an expired password. This setting forces the user to change the password before the user can log in to the database.

```
CREATE USER jbrown
 IDENTIFIED BY password
 ...
 PASSWORD EXPIRE;
```

There is no "password unexpire" clause for the `CREATE USER` statement, but an account can be "unexpired" by changing the password on the account.

PASSWORD_LIFE_TIME Profile Parameter Low Value

Be careful if you set the `PASSWORD_LIFE_TIME` parameter of `CREATE PROFILE` or `ALTER PROFILE` to a low value (for example, 1 day).

The `PASSWORD_LIFE_TIME` limit of a profile is measured from the last time that an account's password is changed, or the account creation time if the password has never been changed. These dates are recorded in the `PTIME` (password change time) and `CTIME` (account creation time) columns of the `SYS.USER$` system table. The `PASSWORD_LIFE_TIME` limit is not measured starting from the timestamp of the last change to the `PASSWORD_LIFE_TIME` profile parameter, as may be initially thought. Therefore, any accounts affected by the changed profile whose last password change time was more than `PASSWORD_LIFE_TIME` days ago immediately expire and enter their grace period on their next connection, issuing the `ORA-28002: The password will expire within` *n* `days` warning.

As a database administrator, you can find an account's last password change time as follows:

```
ALTER SESSION SET NLS_DATE_FORMAT='DD-MON-YYYY HH24:MI:SS';
SELECT PTIME FROM SYS.USER$ WHERE NAME = 'user_name'; -- Password change time
```

To find when the account was created and the password expiration date, issue the following query:

```
SELECT CREATED, EXPIRY_DATE FROM DBA_USERS WHERE USERNAME = 'user_name';
```

If the user who is assigned this profile is currently logged in when you set the `PASSWORD_LIFE_TIME` parameter and remains logged in, then Oracle Database does not change the user's account status from `OPEN` to `EXPIRED(GRACE)` when the currently listed expiration date passes. The timing begins only when the user logs into the database. You can check the user's last login time as follows:

```
SELECT LAST_LOGIN FROM DBA_USERS WHERE USERNAME = 'user_name';
```

When making changes to a password profile, a database administrator must be aware that if some of the users who are subject to this profile are currently logged in to the Oracle database while their password profile is being updated by the administrator, then those users could potentially remain logged in to the system even beyond the expiration date of their password. You can find the currently logged in users by querying the `USERNAME` column of the `V$SESSION` view.

This is because the expiration date of a user's password is based on the timestamp of the last password change on their account plus the value of the `PASSWORD_LIFE_TIME` password profile parameter set by the administrator. It is *not* based on the timestamp of the last change to the password profile itself.

Note the following:

- If the user is not logged in when you set `PASSWORD_LIFE_TIME` to a low value, then the user's account status does not change until the user logs in.
- You can set the `PASSWORD_LIFE_TIME` parameter to `UNLIMITED`, but this only affects accounts that have not entered their grace period. After the grace period expires, the user must change the password.

Managing the Complexity of Passwords

Oracle Database provides a set of functions that you can use to manage the complexity of passwords.

About Password Complexity Verification

Complexity verification checks that each password is complex enough to protect against intruders who try to guess user passwords.

Using a complexity verification function forces users to create strong, secure passwords for database user accounts. You must ensure that the passwords for your users are complex enough to provide reasonable protection against intruders who try to break into the system by guessing passwords.

How Oracle Database Checks the Complexity of Passwords

Oracle Database provides four password verification functions to check password complexity.

These functions are in the `catpvf.sql` PL/SQL script (located in `$ORACLE_HOME/rdbms/admin`). When these functions are enabled, they can check whether users are correctly creating or modifying their passwords. When enabled, password complexity checking is not enforced for user `SYS`; it only applies to non-`SYS` users. For better security of passwords, Oracle recommends that you associate the password verification function with the default profile. About Customizing Password Complexity Verification provides an example of how to accomplish this.

Who Can Use the Password Complexity Functions?

The password complexity functions enable you to customize how users access your data.

Before you can use the password complexity verification functions in the `CREATE PROFILE` or `ALTER PROFILE` statement, you must be granted the `EXECUTE` privilege on them.

The password verification functions are located in the `SYS` schema.

verify_function_11G Function Password Requirements

The `verify_function_11G` function originated in Oracle Database Release 11*g*.

Note:

The `verify_function_11G` function has been deprecated because it enforces the weaker password restrictions from earlier releases of Oracle Database. Instead, you should use the `ORA12C_VERIFY_FUNCTION`, `ORA12C_STRONG_VERIFY_FUNCTION`, `ORA12C_STIG_VERIFY_FUNCTION` functions, which enforce stronger, more up-to-date password verification restrictions.

This function checks for the following requirements when users create or modify passwords:

- The password contains no fewer than 8 characters and includes at least one numeric and one alphabetic character.
- The password is not the same as the user name, nor is it the user name reversed or with the numbers 1–100 appended.
- The password is not the same as the server name or the server name with the numbers 1–100 appended.
- The password does not contain `oracle` (for example, `oracle` with the numbers 1–100 appended).
- The password is not too simple (for example, `welcome1`, `database1`, `account1`, `user1234`, `password1`, `oracle123`, `computer1`, `abcdefg1`, or `change_on_install`).

- The password differs from the previous password by at least 3 characters.

The following internal checks are also applied:

- The password does not exceed 30 characters.
- The password does not contain the double-quotation character ("). However, it can be surrounded by double-quotation marks.

ora12c_verify_function Password Requirements

The `ora12c_verify_function` function fulfills the *Department of Defense Database Security Technical Implementation Guide* requirements.

This function checks for the following requirements when users create or modify passwords:

- The password contains no fewer than 8 characters and includes at least one numeric and one alphabetic character.
- The password is not the same as the user name or the user name reversed.
- The password is not the same as the database name.
- The password does not contain the word `oracle` (such as `oracle123`).
- The password differs from the previous password by at least 8 characters.
- The password contains at least 1 special character.

The following internal checks are also applied:

- The password does not exceed 30 characters.
- The password does not contain the double-quotation character ("). However, it can be surrounded by double-quotation marks.

ora12c_strong_verify_function Function Password Requirements

The `ora12c_strong_verify_function` function fulfills the *Department of Defense Database Security Technical Implementation Guide* requirements.

This function checks for the following requirements when users create or modify passwords:

- The password must contain at least 2 upper case characters, 2 lower case characters, 2 numeric characters, and 2 special characters. These special characters are as follows:

  ```
  ‘ ~ ! @ # $ % ^ & * ( ) _ - + = { } [ ] \ / < > , . ; ? ' : | (space)
  ```
- The password must differ from the previous password by at least 4 characters.

The following internal checks are also applied:

- The password contains no fewer than nine characters and does not exceed 30 characters.
- The password does not contain the double-quotation character ("). It can be surrounded by double-quotation marks, however.

ora12c_stig_verify_function Password Requirements

The `ora12c_stig_verify_function` function fulfills the Security Technical Implementation Guides (STIG) requirements.

This function checks for the following requirements when users create or modify passwords:

- The password has at least 15 characters.
- The password has at least 1 lower case character and at least 1 upper case character.
- The password has at least 1 digit.
- The password has at least 1 special character.
- The password differs from the previous password by at least 8 characters.

The following internal checks are also applied:

- The password does not exceed 30 characters.
- The password does not contain the double-quotation character ("). However, it can be surrounded by double-quotation marks.

The `ora12c_stig_verify_function` function is the default handler for the `ORA_STIG_PROFILE` profile, which is available in a newly-created or upgraded Oracle database.

About Customizing Password Complexity Verification

Oracle Database enables you to customize password complexity for your site.

You can create your own password complexity verification function in the `SYS` schema, similar to the functions that are defined in `admin/catpvf.sql`. In fact, Oracle recommends that you do so to further secure your site's passwords.

Note the following:

- Do not include Data Definition Language (DDL) statements in the custom password complexity verification function. DDLs are not allowed during the execution of password complexity verification functions.
- Do not modify the `admin/catpvf.sql` script or the Oracle-supplied password complexity functions. You can create your own functions based on the contents of these files.
- If you make no modifications to the `utlpwdmg.sql` script, then it uses the `ora12c_verify_function` function as the default function.

See Also:

Guideline 1 in Guidelines for Securing Passwords for general advice on creating passwords

Enabling Password Complexity Verification

The `catpvf.sql` script can be customized to enable password complexity verification.

To enable password complexity verification, you must edit the `catpvf.sql` script to use the password verification function that you want, and then run the script to enable it.

1. Log in to SQL*Plus with administrative privileges.

   For example:

   ```
   CONNECT SYSTEM
   Enter password: password
   ```

2. Run the `catpvf.sql` script (or your modified version of this script) to create the password complexity functions in the `SYS` schema.

   ```
   @$ORACLE_HOME/rdbms/admin/catpvf.sql
   ```

3. Grant any users who must use this function the `EXECUTE` privilege on it.

   For example:

   ```
   GRANT pmsith EXECUTE ON ora12c_strong_verify_function;
   ```

4. In the default profile or the user profile, set the `PASSWORD_VERIFY_FUNCTION` setting to either the sample password complexity function in the `catpvf.sql` script, or to your customized function. Use one of the following methods:

   - Log in to SQL*Plus with administrator privileges and use the `CREATE PROFILE` or `ALTER PROFILE` statement to enable the function. Ensure that you have the `EXECUTE` privilege on the function.

     For example, to update the default profile to use the `ora12c_strong_verify_function` function:

     ```
     ALTER PROFILE default LIMIT
      PASSWORD_VERIFY_FUNCTION ora12c_strong_verify_function;
     ```

   - In Oracle Enterprise Manager Cloud Control, from the **Administration** menu, select **Security**, and then **Profiles**. Select the **Password** tab. Under **Complexity**, from the **Complexity function** list, select the name of the complexity function that you want. Click **Apply**.

After you have enabled password complexity verification, it takes effect immediately. If you must disable it, then run the following statement:

```
ALTER PROFILE DEFAULT LIMIT PASSWORD_VERIFY_FUNCTION NULL;
```

Note:

The ALTER USER statement has a REPLACE clause. With this clause, users can change their own unexpired passwords by supplying the previous password to authenticate themselves.

If the password has expired, then the user cannot log in to SQL to issue the ALTER USER command. Instead, the OCIPasswordChange() function must be used, which also requires the previous password.

A database administrator with ALTER ANY USER privilege can change any user password (force a new password) without supplying the old one.

Managing Password Case Sensitivity

You can manage the password case sensitivity for passwords from user accounts from previous releases.

SEC_CASE_SENSITIVE_LOGON Parameter and Password Case Sensitivity

The SEC_CASE_SENSITIVE_LOGON initialization parameter controls the use of case sensitivity in passwords.

Only users who have the ALTER SYSTEM privilege can set the SEC_CASE_SENSITIVE_LOGON parameter. You should ensure that this parameter is set to TRUE so that case sensitivity is enforced when a user enters a password. However, you should be aware that the SEC_CASE_SENSITIVE_LOGON parameter is deprecated, but is currently retained for backward compatibility.

When you create or modify user accounts, by default, passwords are case sensitive. Case sensitivity affects not only passwords that users enter manually, but it affects password files as well.

For greater security, Oracle recommends that you use case sensitivity in passwords. However, if you have compatibility issues with your applications, then you can use the SEC_CASE_SENSITIVE_LOGON parameter to disable password case sensitivity. Examples of application compatibility issues are applications that force passwords to uppercase before using them to authenticate to the Oracle Database server, or different application modules being inconsistent about case sensitivity when sending credentials to start a database session.

Ensure that the SEC_CASE_SENSITIVE_LOGON parameter is not set to FALSE if the SQLNET.ALLOWED_LOGON_VERSION_SERVER parameter is set to 12 or 12a. This is because the more secure password versions used for this mode only support case-sensitive password checking. For compatibility reasons, Oracle Database does not prevent the use of FALSE for SEC_CASE_SENSITIVE_LOGON when SQLNET.ALLOWED_LOGON_VERSION_SERVER is set to 12 or 12a. Setting SEC_CASE_SENSITIVE_LOGON to FALSE when SQLNET.ALLOWED_LOGON_VERSION_SERVER is set to 12 or 12a causes all accounts to become inaccessible. If SQLNET.ALLOWED_LOGON_VERSION_SERVER is set to 11 or a lower value, then Oracle recommends that you set SEC_CASE_SENSITIVE_LOGON to TRUE, because the more secure password versions used in Exclusive Mode (when SQLNET.ALLOWED_LOGON_VERSION_SERVER is 12 or 12a) in Oracle Database 12*c* do not support case insensitive password matching.

In addition to the server-side settings, you should ensure that the client software with which the users are connecting has the O5L_NP capability flag. All Oracle Database release 11.2.0.3 and later clients have the O5L_NP capability. If you have an earlier client, then you must install the CPUOct2012 patch.

Using the ALTER SYSTEM Statement to Enable Password Case Sensitivity

If password case sensitivity has been disabled, then you can enable it by setting the SEC_CASE_SENSITIVE_LOGON parameter to TRUE.

1. If you are using a password file, then ensure that it was created with the ORAPWD utility IGNORECASE parameter set to N and the FORMAT parameter set to 12.

   The IGNORECASE parameter overrides the SEC_CASE_SENSITIVE_LOGON parameter. By default, IGNORECASE is set to N, which means that passwords are treated as case sensitive.

   Note that the IGNORECASE parameter and the SEC_CASE_SENSITIVE_LOGON system parameter are deprecated. Oracle strongly recommends that you set IGNORECASE to N or omit the IGNORECASE setting entirely.

2. Enter the following ALTER SYSTEM statement:

   ```
   ALTER SYSTEM SET SEC_CASE_SENSITIVE_LOGON = TRUE;
   ```

See Also:

Oracle Database Administrator’s Guide for more information about password files

Management of Case Sensitivity for Secure Role Passwords

For better security, you should ensure that the passwords for secure roles are case sensitive.

If before upgrading to Oracle Database 12c release 2 (12.2), you created secure roles by using the IDENTIFIED BY clause of the CREATE ROLE statement, and if upon upgrading to Oracle Database 12c release 12.2, you set the SQLNET.ALLOWED_LOGON_VERSION_SERVER parameter to one of the Exclusive Modes 12 or 12a, then you must change the password for these secure roles in order for them to remain usable. Because Exclusive Mode is now the default, secure roles that were created in earlier releases (such as Oracle Database 10g, in which the 10G password version was the default) will need to have their passwords changed.

You can query the PASSWORD_REQUIRED and AUTHENTICATION_TYPE columns of the DBA_ROLES data dictionary view to find any secure roles that must have their password changed after upgrade to Oracle Database 12c, in order to become usable again.

Otherwise, the password version for these secure roles cannot be used, unless you set the SQLNET.ALLOWED_LOGON_VERSION_SERVER parameter to 8. If this parameter is set to 12 or 12a, then you must run the following SQL statement to ensure that case sensitivity is enabled. If not, then secure roles will remain unusable even after their passwords have been changed.

```
ALTER SYSTEM SET SEC_CASE_SENSITIVE_LOGON = "TRUE";
```

Management of Password Versions of Users

By default, Oracle Database uses Exclusive Mode, which does not permit case-insensitive passwords, to manage password versions.

In a default installation, the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter is set to `12` to enable Exclusive Mode. Exclusive Mode requires that the password-based authentication protocol use one of the case-sensitive password versions (`11G` or `12C`) for the account that is being authenticated. Exclusive Mode excludes the use of the `10G` password version that was used in earlier releases. After you upgrade to Oracle Database 12*c* release 2 (12.2), accounts that use the `10G` password version become inaccessible. This occurs because the server runs in Exclusive Mode by default, and Exclusive Mode cannot use the old `10G` password version to authenticate the client. The server is left with no password version with which to authenticate the client.

The user accounts from Release 10*g* use the `10G` password version. Therefore, you should find the user accounts that use the `10G` password version, and then reset the passwords for these accounts. This generates the appropriate password version based on the setting of the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter, as follows:

- `SQLNET.ALLOWED_LOGON_VERSION_SERVER=8` generates all three password versions `10G`, `11G`, and `12C`.
- `SQLNET.ALLOWED_LOGON_VERSION_SERVER=12` generates both `11G` and `12C` password versions, and removes the `10G` password version.
- `SQLNET.ALLOWED_LOGON_VERSION_SERVER=12a` generates only the `12C` password version.

If you first relax the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` setting to a more permissive value (such as `SQLNET.ALLOWED_LOGON_VERSION_SERVER=8`) and then import the user accounts from an Oracle Database release 10*g* (or earlier) release into the current database release, then because the `10G` password version (used in the older release) is not case sensitive, these users will still be able to log into the database using any case for their password. But when such a user changes their password, the new `11G` and `12C` password versions are generated automatically, and their password will automatically become case sensitive, because the default value for the instance initialization parameter `SEC_CASE_SENSITIVE_LOGON` is `TRUE`. (Be aware that `SEC_CASE_SENSITIVE_LOGON` is deprecated, but is currently retained for backward compatibility.)

The following example demonstrates the effect of setting the `SEC_CASE_SENSITIVE_LOGON` parameter to `TRUE`. In this scenario, user `rtaylor` has been imported from Oracle Database release 10*g*, and therefore this account only has the `10G` password version. On the server, the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` is set to `8` because otherwise `rtaylor` would not be able to log in. In addition, the `SEC_CASE_SENSITIVE_LOGON` parameter is set to `TRUE` to enable case sensitivity for the `11G` and `12C` password versions.

1. Check the password versions for user `rtaylor`:

   ```
   SELECT PASSWORD_VERSIONS FROM DBA_USERS WHERE USERNAME='RTAYLOR';

   PASSWORD VERSIONS
   -----------------
   10G
   ```

2. Connect as user `rtaylor`.

```
CONNECT rtaylor
Enter password: "MaresEatOats"

Connected.
```

User `rtaylor` can connect to the database because his password still uses the `10G` password version, which is case insensitive. Here, he enters his password in mixed case, though his actual password is all lower case: `mareseatoats`.

3. Check the password versions for one of the default users, `SCOTT`.

```
SELECT PASSWORD_VERSIONS FROM DBA_USERS WHERE USERNAME='SCOTT';

PASSWORD VERSIONS
-----------------
11G 12C
```

4. Try connecting as user `SCOTT` using a mixed case for the password, even though his actual password is all lowercase: `luv2walkmyk9`.

```
CONNECT SCOTT
Enter password: "LuvToWalkMyK9"

ERROR: ORA-01017: invalid username/password; logon denied
Warning: You are no longer connected to ORACLE.
```

Because user `SCOTT`'s password versions are `11G` and `12G`, the password is case sensitive. The password entered in this example is correct, but the case is incorrect.

5. Alter `rtaylor`'s password to `grumble_mumble2work`.

```
ALTER USER rtaylor IDENTIFIED BY grumble_mumble2work;

User altered.
```

6. Connect with the `SYSDBA` administrative privilege.

```
CONNECT / AS SYSDBA
```

7. Find the password versions for user `rtaylor`.

```
SELECT PASSWORD_VERSIONS FROM DBA_USERS WHERE USERNAME='RTAYLOR';

PASSWORD VERSIONS
-----------------
10G 11G 12C
```

The authentication protocol that was configured with the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` and `SEC_CASE_SENSITIVE_LOGON` settings will enforce the case sensitivity of `rtaylor`'s password, now that he has changed this password.

8. Try connecting as `rtaylor` using a mixed case for the password.

```
CONNECT rtaylor
Enter password: "Grumble_Mumble2Work"

ERROR: ORA-01017: invalid username/password; logon denied
Warning: You are no longer connected to ORACLE.
```

The password entered fails because it was not entered using the case in which the password was created.

9. Try connecting as `rtaylor` again but with the password using the correct case

   ```
   CONNECT rtaylor
   Enter password: "grumble_mumble2work"

   Connected.
   ```

   User `rtaylor` can connect.

The case sensitivity of the `rtaylor` account is a result of the server's default setting for `SEC_CASE_SENSITIVE_LOGON`, which is `TRUE`. If this setting is `FALSE`, then case-insensitive matching can be restored because the `rtaylor` account still has the `10G` password version. However, Oracle does not recommend this setting. The `SEC_CASE_SENSITIVE_LOGON` parameter is deprecated for this reason. For greater security, Oracle strongly recommends that you keep case-sensitive password authentication enabled.

Finding and Resetting User Passwords That Use the 10G Password Version

For better security, find and reset passwords for user accounts that use the `10G` password version so that they use later, more secure password versions.

Finding All Password Versions of Current Users

You can query the `DBA_USERS` data dictionary view to find a list of all the password versions that user accounts have.

For example:

```
SELECT USERNAME,PASSWORD_VERSIONS FROM DBA_USERS;

USERNAME                       PASSWORD_VERSIONS
------------------------------ -----------------
JONES                          10G 11G 12C
ADAMS                          10G 11G
CLARK                          10G 11G
PRESTON                        11G
BLAKE                          10G
```

The `PASSWORD_VERSIONS` column shows the list of password versions that exist for the account. `10G` refers to the earlier case-insensitive Oracle password version, `11G` refers to the SHA-1-based password version, and `12C` refers to the SHA-2-based SHA-512 password version.

- User `jones`: The password for this user was reset in Oracle Database 12*c* release 12.1 when the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter setting was `8`. This enabled all three password versions to be created.
- Users `adams` and `clark`: The passwords for these accounts were originally created in Oracle Database 10*g* and then reset in Oracle Database 11*g*. The Oracle Database 11*g* software was using the default `SQLNET.ALLOWED_LOGON_VERSION` setting of `8` at that time. Because case insensitivity is enabled by default, their passwords are now case sensitive, as is the password for `preston`.
- User `preston`: This account was imported from an Oracle Database 11*g* database that was running in Exclusive Mode (`SQLNET.ALLOWED_LOGON_VERSION = 12`).
- User `blake`: This account still uses the Oracle Database 10*g* password version. At this stage, user `blake` will be prevented from logging in.

Resetting User Passwords That Use the 10G Password Version

For better security, you should remove the `10G` password version from the accounts of all users. In the following procedure, to reset the passwords of users who have the `10G` password version, you must temporarily relax the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` setting, which controls the ability level required of clients before login can be allowed. This will enable these users to log in and change their passwords, and hence generate the newer password versions in addition to the `10G` password version. Afterward, you will set the database to use Exclusive Mode and ensure that the clients have the `O5L_NP` capability. Then the users can reset their passwords again, so that their password versions no longer include `10G` but only have the more secure `11G` and `12C` password versions.

1. Query the `DBA_USERS` view to find users who only use the `10G` password version.

   ```
   SELECT USERNAME FROM DBA_USERS
   WHERE ( PASSWORD_VERSIONS = '10G '
   OR PASSWORD_VERSIONS = '10G HTTP ')
   AND USERNAME <> 'ANONYMOUS';
   ```

2. Configure the database so that it does not run in Exclusive Mode, as follows:

   a. Edit the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` setting in the `sqlnet.ora` file so that it is more permissive than the default. For example:

      ```
      SQLNET.ALLOWED_LOGON_VERSION_SERVER=11
      ```

   b. Restart the database.

3. Expire the users that you found when you queried the `DBA_USERS` view to find users who only use the `10G` password version.

   You must expire the users who have only the `10G` password version, and do not have one or both of the `11G` or `12C` password versions.

   For example:

   ```
   ALTER USER username PASSWORD EXPIRE;
   ```

4. Ask the users whose passwords you expired to log in.

   When the users log in, they will be prompted to change their passwords. The database generates the missing `11G` and `12C` password versions for their account, in addition to the `10G` password version. The `10G` password version continues to be present, because the database is running in the permissive mode.

5. Ensure that the client software with which the users are connecting has the `O5L_NP` ability.

   All Oracle Database release 11.2.0.3 and later clients have the `O5L_NP` ability. If you have an earlier Oracle Database client, then you must install the CPUOct2012 patch.

6. After all clients have the `O5L_NP` capability, set the security for the server back to Exclusive Mode, as follows:

   a. Remove the `SEC_CASE_SENSITIVE_LOGON` parameter setting from the instance initialization file, or set `SEC_CASE_SENSITIVE_LOGON` to `TRUE`.

      ```
      SEC_CASE_SENSITIVE_LOGON = TRUE
      ```

   b. Remove the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter from the server `sqlnet.ora` file, or set the value of `SQLNET.ALLOWED_LOGON_VERSION_SERVER` in the server `sqlnet.ora` file back to `12`, to set it to an Exclusive Mode.

```
SQLNET.ALLOWED_LOGON_VERSION_SERVER = 12
```

   c. Restart the database.

7. Find the accounts that still have the `10G` password version.

   ```
   SELECT USERNAME FROM DBA_USERS
   WHERE PASSWORD_VERSIONS LIKE '%10G%'
   AND USERNAME <> 'ANONYMOUS';
   ```

8. Expire the accounts that still have the `10G` password version.

   ```
   ALTER USER username PASSWORD EXPIRE;
   ```

9. Ask these users to log in to their accounts.

   When the users log in, they are prompted to reset their passwords. The database then generates only the `11G` and `12C` password versions for their accounts. Because the database is running in Exclusive Mode, the `10G` password version is no longer generated.

10. Rerun the following query:

   ```
   SELECT USERNAME FROM DBA_USERS
   WHERE PASSWORD_VERSIONS LIKE '%10G%'
   AND USERNAME <> 'ANONYMOUS';
   ```

   If this query does not return any results, then it means that no user accounts have the `10G` password version. Hence, the database is running in a more secure mode than in previous releases.

How Case Sensitivity Affects Password Files

By default, password files are case sensitive. The `IGNORECASE` argument in the `ORAPWD` command line utility controls the case sensitivity of password files.

The default value for `IGNORECASE` is `N` (no), which enforces case sensitivity. For better security, set `IGNORECASE` to `N` or omit the `ignorecase` argument entirely. Note that `IGNORECASE` is deprecated.

The following example shows how to enable case sensitivity in password files.

```
orapwd file=orapw entries=100
Enter password for SYS: password
```

This command creates a case sensitive password file called `orapw`. By default, passwords are case sensitive. Afterwards, if you connect using this password, it succeeds—as long as you enter it using the *exact* case in which it was created. If you enter the same password but with a *different* case sensitivity, it will fail.

If you imported user accounts from a previous release and these accounts were created with `SYSDBA` or `SYSOPER` administrative privilege, then they will be included in the password file. The passwords for these accounts are case insensitive. The next time these users change their passwords, and assuming case sensitivity is enabled, the passwords become case sensitive. For greater security, have these users change their passwords.

See Also:

Oracle Database Administrator's Guide for more information about password files

How Case Sensitivity Affects Passwords Used in Database Link Connections

When you create a database link connection, you must define a user name and password for the connection.

When you create the database link connection, the password is case sensitive. How a user enters his or her password for connections depends on the release in which the database link was created:

- Users can connect from a pre-Oracle Database 12c database to a Oracle Database 12c database. Because case sensitivity is enabled, then the user must enter the password using the case that was used when the account was created.
- If the user connects from a Oracle Database 12c database to a pre-Oracle Database 12c database, and if the `SEC_CASE_SENSITIVE_LOGON` parameter in the pre-Release 12*c* database had been set to `FALSE`, then the password for this database link can be specified using any case.

You can find the user accounts for existing database links by querying the `V$DBLINK` view. For example:

```
SELECT DB_LINK, OWNER_ID FROM V$DBLINK;
```

See *Oracle Database Reference* for more information about the `V$DBLINK` view.

Ensuring Against Password Security Threats by Using the 12C Password Version

The `12C` password version enables users to create complex passwords that meet compliance standards.

About the 12C Version of the Password Hash

The `12C` password hash protects against password-based security threats by including support for mixed case passwords.

The cryptographic hash function used for generating the `12C` version of the password hash is based on a de-optimized algorithm involving Password-Based Key Derivation Function 2 (PBKDF2) and the SHA-512 cryptographic hash functions. The PBKDF2 algorithm introduces computational asymmetry in the challenge that faces an intruder who is trying to recover the original password when in possession of the `12C` version of the password hash. The `12C` password generation performs a SHA-512 hash of the PBKDF2 output as its last step. This two-step approach used in the `12C` password version generation allows server CPU resources to be conserved when the client has the O7L_MR capability. This is because during the password verification phase of the O5LOGON authentication, the server only needs to perform a single SHA-512 hash of a value transmitted by the O7L_MR capable client, rather than having to repeat the entire PBKDF2 calculation on the password itself.

In addition, the `12C` password version adds a salt to the password when it is hashed, which provides additional protection. The `12C` password version enables your users to create far more complex passwords. The `12C` password version's use of salt, its use of PBKDF2 de-optimization, and its support for mixed-case passwords makes it more expensive for an intruder to perform dictionary or brute force attacks on the `12C` password version in an attempt to recover the user's password. Oracle recommends that you use the `12C` version of the password hash.

The password hash values are considered to be extremely sensitive, because they are used as a "shared secret" between the server and person who is logging in. If an intruder learns this secret, then the protection of the authentication is immediately and severely compromised. Remember that administrative users who have account management privileges, administrative users who have the `SYSDBA` administrative privilege, or even users who have the `EXP_FULL_DATABASE` role can immediately access the password hash values. Therefore, this type of administrative user must be trustworthy if the integrity of the database password-based authentication is to be preserved. If you cannot trust these administrators, then it is better to deploy a directory server (such as Oracle Database Enterprise User Security) so that the password hash values remain within the Enterprise User Security directory and are never accessible to anyone except the Enterprise User Security administrator.

See Also:

Oracle Database Net Services Reference for more information about O7L_MR

Oracle Database 12C Password Version Configuration Guidelines

By default, Oracle Database generates two versions of the password hash: `11G` and `12C`.

The version of the password hash that Oracle Database uses to authenticate a given client depends on the client’s ability, and the settings for the `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` and `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameters. See the column “Ability Required of the Client” in the “SQLNET.ALLOWED_LOGON_VERSION_SERVER Settings” table in the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter description in *Oracle Database Net Services Reference* for detailed information about how the client authentication works with password versions.

The `10G` password version, which was generated in Oracle Database 10g, is not case sensitive. Both the `11G` and `12C` password versions are case sensitive.

In Oracle Database 12g release 2 (12.2), the `sqlnet.ora` parameter `SQLNET.ALLOWED_LOGON_VERSION_SERVER` defaults to `12`, which is Exclusive Mode and prevents the use of the `10G` password version, and the `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` parameter defaults to `11`. For new accounts, when the client is Oracle Database 12c, then Oracle Database uses the `12C` password version exclusively with clients that are running the Oracle Database 12c release software. For accounts that were created before Oracle Database release 12c, logins will succeed as long as the client has the O5L_NP ability, because an `11G` password version normally exists for accounts created in earlier releases such as Oracle Database release 11g. For a very old account (for example, from Oracle Database release 10g), the user’s password may need to be reset, in order to create a SHA-1 password version for the account. To configure this server to generate only the `12C` password version whenever a new account is created or an existing account password is changed, then set the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter to `12a`. However, if you want your applications to be compatible with older clients, then ensure that `SQLNET.ALLOWED_LOGON_VERSION_SERVER` is set to `12`, which is the default.

How you set the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter depends on the balance of security and interoperability with older clients that your system requires. You can control the levels of security as follows:

- **Greatest level of compatibility:** To configure the server to generate all three versions of the password hash (the `12C` password version, the `11G` password version, and the DES-based `10G` password version), whenever a new account is created or an existing account password is changed, set the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter to the value `11` or lower. (Be aware that earlier releases used the value `8` as the default.)
- **Recommended level of security:** To configure the server to generate both the `12C` password version and the `11G` password version (but *not* the `10G` password version), whenever a new account is created or an existing account password is changed, set the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter to the value `12`.
- **Highest level of security:** To configure the server to generate *only* the `12C` password version whenever a new account is created or an existing account password is changed, set the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter to the value `12a`.

During authentication, the following scenarios are possible, based on the kinds of password versions that exist for the account, and on the version of the client software being used:

- **Accounts with only the 10G version of the password hash:** If you want to force the server to generate the newer versions of the password hash for older accounts, an administrator must expire the password for any account that has only the `10G` password version (and none of the more secure password versions, `11G` or `12C`). You must generate these password versions because the database depends on using these password versions to provide stronger security. You can find these users as follows.

  ```
  SELECT USERNAME FROM DBA_USERS
  WHERE PASSWORD_VERSIONS LIKE '%10G%
  AND USERNAME <> 'ANONYMOUS';
  ```

  And then expire each account as follows:

  ```
  ALTER USER username PASSWORD EXPIRE;
  ```

  After you have expired each account, notify these users to log in, in which case they will be prompted to change their password. The version of the client determines the password version that is used. The setting of the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter determines the password versions that are generated. If the client has the O7L_MR ability (Oracle Database release 12c), then the 12C password version is used to authenticate. If the client has the O5L_NP ability but not the O7L_MR ability (such as Oracle Database release 11g clients), then the `11G` password version is used to authenticate. You should upgrade all clients to Oracle Database release 12c so that the `12C` password version can be used exclusively to authenticate. (By default, Oracle Database release 11.2.0.3 and later clients have the `O5L_NP` ability, which enables the `11G` password version to be used exclusively. If you have an earlier Oracle Database client, then you must install the CPUOct2012 patch.)

When an account password is expired and the `ALLOWED_LOGON_VERSION_SERVER` parameter is set to `12` or `12a`, then the `10G` password version is removed and only one or both of the new password versions are created, depending on how the parameter is set, as follows:

– If `ALLOWED_LOGON_VERSION_SERVER` is set to `12` (the default), then both the `11G` and `12C` versions of the password hash are generated.

– If `ALLOWED_LOGON_VERSION_SERVER` is set to `12a`, then only the `12C` version of the password hash is generated.

For more details, see the "Generated Password Version" column in the table in the "Usage Notes" section for the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter in *Oracle Database Net Services Reference*.

- **Accounts with both 10G and 11G versions of the password hash:** For users who are using a Release 10g or later client, the user logins will succeed because the `11G` version of the password hash is used. However, to use the latest version, expire these passwords, as described in the previous bulleted item for accounts.
- **Accounts with only the 11G version of the password hash:** The authentication uses the `11G` version of the password hash. To use the latest version, expire the passwords, as described in the first bulleted item.

The Oracle Database 12*c* default configuration for `SQLNET.ALLOWED_LOGON_VERSION_SERVER` is `12`, which means that it is compatible with Oracle Database 12c release 2 (12.2) authentication protocols and later products that use OCI-based drivers, including SQL*Plus, ODBC, Oracle .NET, Oracle Forms, and various third-party Oracle Database adapters. It is also compatible with JDBC type-4 (thin) versions that have had the CPUOct2012 bundle patch applied or starting with Oracle Database 11g, and Oracle Database Client interface (OCI)-based drivers starting in Oracle Database 10g release 10.2. Be aware that earlier releases of the OCI client drivers cannot authenticate to an Oracle database using password-based authentication.

Configuring Oracle Database to Use the 12C Password Version Exclusively

You should set the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter to `12a` so that only the `12C` password hash version is used.

The `12C` password version is the most restrictive and secure of the password hash versions, and for this reason, Oracle recommends that you use only this password version. By default, `SQLNET.ALLOWED_LOGON_VERSION_SERVER` is set to `12`, which enables both the `11G` and `12C` password versions to be used. (Both the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` values `12` and `12a` are considered Exclusive Mode, which prevents the use of the earlier `10G` password version.) If you have upgraded from a previous release, or if `SQLNET.ALLOWED_LOGON_VERSION_SERVER` is set to `12` or another setting that was used in previous releases, then you should reconfigure this parameter, because intruders will attempt to downgrade the authentication to use weaker password versions. Table 3-3 shows the effect of the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` setting on password version generation.
Be aware that you can use the `12C` password version exclusively only if you use Oracle Database 12c release 12.1.0.2 or later clients. Before you change the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter to `12a`, check the versions of the database clients that are connected to the server.

1. Log in to SQL*Plus as an administrative user who has the `ALTER USER` system privilege.
2. Perform the following SQL query to find the password versions of your users.

   ```
   SELECT USERNAME,PASSWORD_VERSIONS FROM DBA_USERS;
   ```

3. Expire the account of each user who does not have the `12C` password version.

   For example, assuming user `blake` is still using a `10G` password version:

   ```
   ALTER USER blake PASSWORD EXPIRE;
   ```

   The next time that these users log in, they will be forced to change their passwords, which enables the server to generate the password versions required for Exclusive Mode.
4. Remind users to log in within a reasonable period of time (such as 30 days).

   When they log in, they will be prompted to change their password, ensuring that the password versions required for authentication in Exclusive Mode are generated by the server. (For more information about how Exclusive Mode works, see the usage notes for the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter in *Oracle Database Net Services Reference*.)
5. Manually change the passwords for accounts that are used in test scripts or batch jobs so that they exactly match the passwords used by these test scripts or batch jobs, including the password's case.
6. Enable the Exclusive Mode configuration as follows:
   a. Create a back up copy of the `sqlnet.ora` parameter file.

      By default, this file is located in the `$ORACLE_HOME/network/admin` directory on UNIX operating systems and the `%ORACLE_HOME%\network\admin` directory on Microsoft Windows operating systems.

      Be aware that in a Multitenant environment, the settings in the `sqlnet.ora` file apply to all PDBs.
   b. Set the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter, using Table 3-3 for guidance.
   c. Save the `sqlnet.ora` file.

Table 3-3 shows the effect of the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` setting on password version generation.

Table 3-3 Effect of SQLNET.ALLOWED_LOGON_VERSION_SERVER on Password Version Generation

SQLNET.ALLOWED_LOGON_VERSION_SERVER Setting	8	11	12	12a
Server runs in Exclusive Mode?	No	No	Yes	Yes
Generate the `10G` password version?	Yes	Yes	No	No
Generate the `11G` password version?	Yes	Yes	Yes	No
Generate the `12C` password version?	Yes	Yes	Yes	Yes

If you only use Oracle Database 12c release 12.1.0.2 or later clients, then set `SQLNET.ALLOWED_LOGON_VERSION_SERVER` to `12a`.

The higher the setting, the more restrictive the use of password versions, as follows:

- A setting of `12a`, the most restrictive and secure setting, only permits the `12C` password version.
- A setting of `12` permits both the `11G` and `12C` password versions to be used for authentication.
- A setting of `8` permits the most password versions: `10G`, `11G`, and `12C`.

For detailed information about the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter, see *Oracle Database Net Services Reference*.

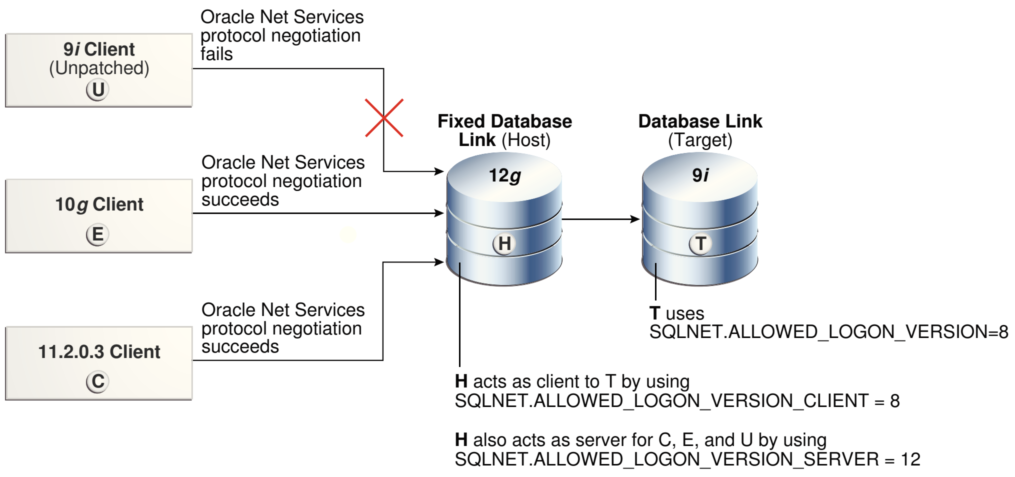
Note:

If your system hosts a fixed database link to a target database that runs an earlier release, then you can set the `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` parameter, as described in How Server and Client Logon Versions Affect Database Links.

How Server and Client Logon Versions Affect Database Links

The `SQLNET.ALLOWED_LOGON_VERSION_SERVER` and `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` parameters can accommodate connections between databases and clients of different releases.

The following diagram illustrates how connections between databases and clients of different releases work. The `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` parameter affects the "client allowed logon version" aspect of a server that hosts the database link **H**. This setting enables **H** to connect through database links to older servers, such as those running Oracle 9*i* (**T**), yet still refuse connections from older unpatched clients (**U**). When this happens, the Oracle Net Services protocol negotiation fails, which raises an `ORA-28040: No matching authentication protocol error` message in this client, which is attempting to authenticate using the Oracle 9*I* software. The Oracle Net Services protocol negotiation for Oracle Database 10*g* release 10.2 client **E** succeeds because this release incorporates the critical patch update CPUOct2012. The Oracle Net Services protocol negotiation for Release 11.2.0.3 client **C** succeeds because it uses a secure password version.


9i Client (Unpatched) U
Oracle Net Services protocol negotiation fails
10g Client E
Oracle Net Services protocol negotiation succeeds
11.2.0.3 Client C
Oracle Net Services protocol negotiation succeeds
Fixed Database Link (Host)
12g
H
Database Link (Target)
9i
T
T uses SQLNET.ALLOWED_LOGON_VERSION=8
H acts as client to T by using SQLNET.ALLOWED_LOGON_VERSION_CLIENT = 8
H also acts as server for C, E, and U by using SQLNET.ALLOWED_LOGON_VERSION_SERVER = 12

This scenario uses the following settings for the system that hosts the database link **H**:

```
SQLNET.ALLOWED_LOGON_VERSION_CLIENT=8
SQLNET.ALLOWED_LOGON_VERSION_SERVER=12
```

Note that the remote Oracle Database **T** has the following setting:

```
SQLNET.ALLOWED_LOGON_VERSION=8
```

If the release of the remote Oracle Database **T** does not meet or exceed the value defined by the `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` parameter set for the host **H**, then queries over the fixed database link would fail during authentication of the database link user, resulting in an `ORA-28040: No matching authentication protocol` error when an end-user attempts to access a table over the database link.

Note:

If you are using an older Oracle Database client (such as Oracle Database 11*g* release 11.1.0.7), then Oracle strongly recommends that you upgrade to use the critical patch update CPUOct2012.

See Also:

- *Oracle Database Net Services Reference* for more information about the `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` parameter
- `http://www.oracle.com/technetwork/topics/security/cpuoct2012-1515893.html` for more information about CPUOct2012

Configuring Oracle Database Clients to Use the 12C Password Version Exclusively

An intruder may try to provision a fake server to downgrade authentication and trick the client into using a weaker password hash version.

- To prevent the use of the `10G` password version, or both the `10G` and `11G` password versions, after you configure the server, configure the clients to run in Exclusive Mode, as follows:
  - To use the client Exclusive Mode setting to permit both the `11G` and `12C` password versions:

    ```
    SQLNET.ALLOWED_LOGON_VERSION_CLIENT = 12
    ```

  - To use the more restrictive client Exclusive Mode setting to permit only the `12C` password version (this setting permits the client to connect only to Oracle Database 12*c* release 1 (12.1.0.2) and later servers):

    ```
    SQLNET.ALLOWED_LOGON_VERSION_CLIENT = 12a
    ```

If the server and the client are both installed on the same computer, then ensure that the `TNS_ADMIN` environment variable for each points to the correct directory for its respective Oracle Net Services configuration files. If the variable is the same for both, then the server could use the client's `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` setting instead.

If you are using older Oracle Database clients (such as Oracle Database 11*g* release 11.1.0.7), then you should apply CPU Oct2012 or later to these clients. This patch provides the `O5L_NP` ability. Unless you apply this patch, users will be unable to log in.

See Also:

- *Oracle Database Net Services Reference* for more information about the `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` parameter
- The following Oracle Technology Network site for more information about CPUOct2012:

  `http://www.oracle.com/technetwork/topics/security/cpuoct2012-1515893.html`

Managing the Secure External Password Store for Password Credentials

The secure external password store is a client-side wallet that is used to store password credentials.

About the Secure External Password Store

You can store password credentials database connections by using a client-side Oracle wallet.

An Oracle wallet is a secure software container that stores authentication and signing credentials. This wallet usage can simplify large-scale deployments that rely on password credentials for connecting to databases. When this feature is configured, application code, scripts no longer need embedded user names and passwords. This reduces risk because the passwords are no longer exposed, and password management policies are more easily enforced without changing application code whenever user names or passwords change.

Note:

The external password store of the wallet is separate from the area where public key infrastructure (PKI) credentials are stored. Consequently, you cannot use Oracle Wallet Manager to manage credentials in the external password store of the wallet. Instead, use the command-line utility `mkstore` to manage these credentials.

See Also:

- Using Proxy Authentication with the Secure External Password Store
- *Oracle Database Enterprise User Security Administrator's Guide* for general information about Oracle wallets

How Does the External Password Store Work?

Users (and applications, batch jobs, and scripts) connect to databases by using a standard `CONNECT` statement that specifies a database connection string.

This string can include a user name and password, and an Oracle Net service name identifying the database on an Oracle Database network. If the password is omitted, the connection prompts the user for the password.

For example, the service name could be the URL that identifies that database, or a TNS alias you entered in the `tnsnames.ora` file in the database. Another possibility is a *`host:port:sid`* string.

The following examples are standard `CONNECT` statements that could be used for a client that is not configured to use the external password store:

```
CONNECT salesapp@sales_db.us.example.com
Enter password: password

CONNECT salesapp@orasales
Enter password: password

CONNECT salesapp@ourhost37:1527:DB17
Enter password: password
```

In these examples, `salesapp` is the user name, with the unique connection string for the database shown as specified in three different ways. You could use its URL `sales_db.us.example.com`, or its TNS alias `orasales` from the `tnsnames.ora` file, or its *`host:port:sid`* string.

However, when clients are configured to use the secure external password store, applications can connect to a database with the following `CONNECT` statement syntax, without specifying database login credentials:

```
CONNECT /@db_connect_string

CONNECT /@db_connect_string AS SYSDBA

CONNECT /@db_connect_string AS SYSOPER
```

In this specification, *`db_connect_string`* is a valid connection string to access the intended database, such as the service name, URL, or alias as shown in the earlier examples. Each user account must have its own unique connection string; you cannot create one connection string for multiple users.

In this case, the database credentials, user name and password, are securely stored in an Oracle wallet created for this purpose. The autologin feature of this wallet is turned on, so the system does not need a password to open the wallet. From the wallet, it gets the credentials to access the database for the user they represent.

> **See Also:**
>
> *Oracle Database Enterprise User Security Administrator's Guide* for information about autologin wallets

About Configuring Clients to Use the External Password Store

If your client is configured to use external authentication, such as Windows native authentication or SSL, then Oracle Database uses that authentication method.

The same credentials used for this type of authentication are typically also used to log in to the database. For clients not using such authentication methods or wanting to override them for database authentication, you can set the `SQLNET.WALLET_OVERRIDE` parameter in `sqlnet.ora` to `TRUE`. The default value for `SQLNET.WALLET_OVERRIDE` is `FALSE`, allowing standard use of authentication credentials as before.

Configuring a Client to Use the External Password Store

You can configure a client to use the secure external password store feature by using the `mkstore` command-line utility.

1. Create a wallet on the client by using the following syntax at the command line:

   ```
   mkstore -wrl wallet_location -create
   ```

   For example:

   ```
   mkstore -wrl c:\oracle\product\12.2.0\db_1\wallets -create
   Enter password: password
   ```

   `wallet_location` is the path to the directory where you want to create and store the wallet. This command creates an Oracle wallet with the autologin feature enabled at the location you specify. The autologin feature enables the client to access the wallet contents without supplying a password. See *Oracle Database Enterprise User Security Administrator's Guide* for information about autologin wallets.

   The `mkstore` utility `-create` option uses password complexity verification. See About Password Complexity Verification for more information.

2. Create database connection credentials in the wallet by using the following syntax at the command line:

   ```
   mkstore -wrl wallet_location -createCredential db_connect_string username
   Enter password: password
   ```

   For example:

   ```
   mkstore -wrl c:\oracle\product\12.2.0\db_1\wallets -createCredential orcl system
   Enter password: password
   ```

   In this specification:

   - *`wallet_location`* is the path to the directory where you created the wallet in Step 1.
   - *`db_connect_string`* is the TNS alias you use to specify the database in the `tnsnames.ora` file or any service name you use to identify the database on an

Oracle network. By default, `tnsnames.ora` is located in the `$ORACLE_HOME/network/admin` directory on UNIX systems and in *`ORACLE_HOME`*`\network\admin` on Windows.

- *`username`* is the database login credential. When prompted, enter the password for this user.

Repeat this step for each database you want accessible using the `CONNECT /@`*`db_connect_string`* syntax. The *`db_connect_string`* used in the `CONNECT /@`*`db_connect_string`* statement must be identical to the *`db_connect_string`* specified in the `-createCredential` command.

3. In the client `sqlnet.ora` file, enter the `WALLET_LOCATION` parameter and set it to the directory location of the wallet you created in Step 1.

   For example, if you created the wallet in `$ORACLE_HOME/network/admin` and your Oracle home is set to `/private/ora11`, then you need to enter the following into your client `sqlnet.ora` file:

```
WALLET_LOCATION =
  (SOURCE =
    (METHOD = FILE)
    (METHOD_DATA =
  (DIRECTORY = /private/ora11/network/admin)
  )
 )
```

4. In the client `sqlnet.ora` file, enter the `SQLNET.WALLET_OVERRIDE` parameter and set it to `TRUE` as follows:

```
SQLNET.WALLET_OVERRIDE = TRUE
```

   This setting causes all `CONNECT /@`*`db_connect_string`* statements to use the information in the wallet at the specified location to authenticate to databases.

   When external authentication is in use, an authenticated user with such a wallet can use the `CONNECT /@`*`db_connect_string`* syntax to access the previously specified databases without providing a user name and password. However, if a user fails that external authentication, then these connect statements also fail.

Note:

If an application uses SSL for encryption, then the `sqlnet.ora` parameter, `SQLNET.AUTHENTICATION_SERVICES`, specifies SSL and an SSL wallet is created. If this application wants to use secret store credentials to authenticate to databases (instead of the SSL certificate), then those credentials must be stored in the SSL wallet. After SSL authentication, if `SQLNET.WALLET_OVERRIDE = TRUE`, then the user names and passwords from the wallet are used to authenticate to databases. If `SQLNET.WALLET_OVERRIDE = FALSE`, then the SSL certificate is used.

Example: Sample SQLNET.ORA File with Wallet Parameters Set

You can set special parameters in the `sqlnet.ora` file to control how wallets are managed.

Example 3-2 shows a sample sqlnet.ora file with the WALLET_LOCATION and the SQLNET.WALLET_OVERRIDE parameters set as described in Steps 3 and 4 of Configuring a Client to Use the External Password Store.

Example 3-2 Sample SQLNET.ORA File with Wallet Parameters Set

```
WALLET_LOCATION =
   (SOURCE =
     (METHOD = FILE)
     (METHOD_DATA =
       (DIRECTORY = /private/ora11/network/admin)
     )
    )

SQLNET.WALLET_OVERRIDE = TRUE
SSL_CLIENT_AUTHENTICATION = FALSE
SSL_VERSION = 0
```

Managing External Password Store Credentials

The mkstore command-line utility manages credentials from an external password store.

Listing External Password Store Contents

You can view the contents, including specific credentials, of a client wallet external password store.

Listing the external password store contents provides information you can use to decide whether to add or delete credentials from the store.

- To list the contents of the external password store, enter the following command at the command line:

  ```
  mkstore -wrl wallet_location -listCredential
  ```

For example:

```
mkstore -wrl c:\oracle\product\12.1.0\db_1\wallets -listCredential
```

wallet_location specifies the path to the directory where the wallet, whose external password store contents you want to view, is located. This command lists all of the credential database service names (aliases) and the corresponding user name (schema) for that database. Passwords are not listed.

Adding Credentials to an External Password Store

You can store multiple credentials in one client wallet.

For example, if a client batch job connects to hr_database and a script connects to sales_database, then you can store the login credentials in the same client wallet. You cannot, however, store multiple credentials (for logging in to multiple schemas) for the same database in the same wallet. If you have multiple login credentials for the same database, then they must be stored in separate wallets.

- To add database login credentials to an existing client wallet, enter the following command at the command line:

  ```
  mkstore -wrl wallet_location -createCredential db_alias username
  ```

For example:

```
mkstore -wrl c:\oracle\product\12.1.0\db_1\wallets -createCredential orcl system
Enter password: password
```

In this specification:

- *wallet_location* is the path to the directory where the client wallet to which you want to add credentials is stored.
- *db_alias* can be the TNS alias you use to specify the database in the `tnsnames.ora` file or any service name you use to identify the database on an Oracle network.
- *username* is the database login credential for the schema to which your application connects. When prompted, enter the password for this user.

Modifying Credentials in an External Password Store

You can modify the database login credentials that are stored in the wallet if the database connection strings change.

- To modify database login credentials in a wallet, enter the following command at the command line:

  ```
  mkstore -wrl wallet_location -modifyCredential db_alias username
  ```

For example:

```
mkstore -wrl c:\oracle\product\12.2.0\db_1\wallets -modifyCredential sales_db
Enter password: password
```

In this specification:

- *wallet_location* is the path to the directory where the wallet is located.
- *db_alias* is a new or different alias you want to use to identify the database. It can be a TNS alias you use to specify the database in the `tnsnames.ora` file or any service name you use to identify the database on an Oracle network.
- *username* is the new or different database login credential. When prompted, enter the password for this user.

Deleting Credentials from an External Password Store

You can delete login credentials for a database from a wallet if the database no longer exists or to disable connections to a specific database.

- To delete database login credentials from a wallet, enter the following command at the command line:

  ```
  mkstore -wrl wallet_location -deleteCredential db_alias
  ```

For example:

```
mkstore -wrl c:\oracle\product\12.1.0\db_1\wallets -deleteCredential orcl
```

In this specification:

- *wallet_location* is the path to the directory where the wallet is located.
- *db_alias* is the TNS alias you use to specify the database in the `tnsnames.ora` file, or any service name you use to identify the database on an Oracle Database network.

Managing Passwords for Administrative Users

The passwords of administrative users have special protections, such as password files and password complexity functions.

About Managing Passwords for Administrative Users

The passwords of administrative users are stored outside of the database so that the users can be authenticated even when the database is not open.

There is no special protection with the password file. The password verifiers must be stored outside of the database so that authentication can be performed even when the database is not open. In previous releases, password complexity functions were available for non-administrative users only. Starting with Oracle Database release 12c (12.2), password complexity functions can be used for both non-administrative users and administrative users.

Setting the LOCK and EXPIRED Status of Administrative Users

Administrative users whose accounts have been locked cannot connect to the database.

- To unlock locked or expired administrative accounts, use the `ALTER USER` statement.

For example:

```
ALTER USER hr_admin ACCOUNT UNLOCK;
```

If the administrative user's password has expired, then the next time the user attempts to log in, the user will be prompted to create a new password.

Password Profile Settings for Administrative Users

There are several user profile password settings that are enforced for administrative users.

These password profile parameters are as follows:

- `FAILED_LOGIN_ATTEMPT`
- `INACTIVE_ACCOUNT_TIME`
- `PASSWORD_LOCK_TIME`
- `PASSWORD_LIFE_TIME`
- `PASSWORD_GRACE_TIME`

Last Successful Login Time for Administrative Users

The last successful login time of administrative user connections that use password file-based authentication is captured.

To find this login time, query the `LAST_LOGIN` column of the `V$PWFILE_USERS` dynamic performance view.

Management of the Password File of Administrative Users

Setting the `ORAPWD` utility `FORMAT` parameter to `12.2` enables you to manage the password profile parameters for administrative users.

The password file is particularly important for administrative users because it stores the administrative user’s credentials in an external file, not in the database itself. This enables the administrative user to log in to a database that is not open and perform tasks such as querying the data dictionary views. To create the password file, you must use the `ORAPWD` utility.

The `FORMAT` parameter setting of `12.2`, which is the default setting, enables the password file to accommodate the password profile information for the administrative user.

For example:

```
orapwd file=orapworcl input_file=orapwold format=12.2
...
```

Setting `FORMAT` to `12.2` enforces the following rules:

- The password contains no fewer than 8 characters and includes at least one numeric and one alphabetic character.
- The password is not the same as the user name or the user name reversed.
- The password is not the same as the database name.
- The password does not contain the word `oracle` (such as `oracle123`).
- The password differs from the previous password by at least 8 characters.
- The password contains at least 1 special character.

`FORMAT=12.2` also applies the following internal checks:

- The password does not exceed 30 characters.
- The password does not contain the double-quotation character ("). However, it can be surrounded by double-quotation marks.

Configuring `FORMAT=12.2` sets administrative users to use the default profile, which uses these settings:

- `PASSWORD_LIFE_TIME`: `180` days
- `PASSWORD_GRACE_TIME`: `7` days
- `FAILED_LOGIN_ATTEMPTS`: `10` attempts

You can modify these profile parameters without affecting the `FORMAT=12.2` setting.

You can find the administrative users who have been included in the password file and their administrative privileges by querying the `V$PWFILE_USERS` dynamic view.

Migration of the Password File of Administrative Users

The `ORAPWD` utility `input_file` parameter or DBUA can be used to migrate from earlier password file formats to the 12.2 format.

You can migrate from earlier password file formats to the 12.2 format by using either the `ORAPWD` utility file and `input_file` parameters, or by using Oracle Database Upgrade Assistant (DBUA).

- **The ORAPWD FILE and INPUT_FILE parameters:** To migrate using the `ORAPWD` utility, set the `FILE` parameter to a name for the new password file and the `INPUT_FILE` parameter to the name of the earlier password file.

  For example:

  ```
  orapwd file=orapworcl input_file=orapwold format=12.2
  ```

- **DBUA:** To migrate from the earlier formats of password files (`FORMAT = LEGACY` and `FORMAT = 12`), you can use the DBUA when you upgrade an earlier database to the current release. However, ensure that the database is open in read-only mode. You can check the database read-only status by querying the `OPEN_MODE` column of the `V$DATABASE` dynamic view.

See also:

See Also:

Oracle Database Administrator's Guide for more information about the `ORAPWD` parameters

How the Multitenant Option Affects Password Files for Administrative Users

In a multitenant environment, the password information for the local and common administrative users is stored in different locations.

- **For CDB administrative users:** The password information (hashes of the password) for the CDB common administrative users to whom administrative privileges were granted in the CDB root is stored in the password file.
- **For all users in a CDB to whom administrative privileges were granted outside the CDB root:** To view information about the password hash information of these users, query the `$PWFILE_USERS` dynamic view.

Password Complexity Verification Functions for Administrative Users

For better security, use password complexity verification functions for the passwords of administrative users.

Note the following:

- **Profiles:** You can specify a password complexity verification function for the `SYS` user by using the `PASSWORD_VERIFY_FUNCTION` clause of the `CREATE PROFILE` or `ALTER PROFILE` statement. Oracle recommends that you use password verification functions to better protect the passwords of administrative users.
- **ORAPWD password files:** If you created a password file using the `ORAPWD` utility, then Oracle Database enforces password complexity checking for the `SYS` user and for administrative users who have logged in using the `SYSDBA`, `SYSBACKUP`, `SYSDG`, and `SYSKM` administrative privileges.

  The password checks for the following requirements:

  - The password contains no fewer than 8 characters and includes at least one numeric character, one alphabetic character, and one special character.
  - The password is not the same as the user name or the user name reversed.
  - The password does not contain the word `oracle` (such as `oracle123`).
  - The password differs from the previous password by at least three characters.

  The following internal checks are also applied:

  - The password does not exceed 30 characters.
  - The password does not contain the double-quotation character ("). However, it can be surrounded by double-quotation marks.

Authentication of Database Administrators

You can authenticate database administrators by using strong authentication, from the operating system, or from the database using passwords.

About Authentication of Database Administrators

Database administrators perform special administrative operations, such as shutting down or starting databases.

Oracle Database provides methods to secure the authentication of database administrators who have the `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, or `SYSKM` administrative privilege.

Strong Authentication, Centralized Management for Administrators

Strong authentication methods for centrally managed databases include directory authentication, Kerberos authentication, and SSL authentication.

About Strong Authentication for Database Administrators

Strong authentication lets you centrally control `SYSDBA` and `SYSOPER` access to multiple databases.

Consider using this type of authentication for database administration for the following situations:

- You have concerns about password file vulnerability.
- Your site has very strict security requirements.
- You want to separate the identity management from your database. By using a directory server such as Oracle Internet Directory (OID), for example, you can maintain, secure, and administer that server separately.

To enable the Oracle Internet Directory server to authorize `SYSDBA` and `SYSOPER` connections, use one of the following methods described in this section, depending on your environment.

Configuring Directory Authentication for Administrative Users

Oracle Internet Directory configures directory authentication for administrative users.

1. Configure the administrative user by using the same procedures you would use to configure a typical user.
2. In Oracle Internet Directory, grant the `SYSDBA` or `SYSOPER` administrative privilege to the user for the database that this user will administer.

   Grant `SYSDBA` or `SYSOPER` only to trusted users.
3. Set the `LDAP_DIRECTORY_SYSAUTH` initialization parameter to `YES`:

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_SYSAUTH = YES;
   ```

   When set to `YES`, the `LDAP_DIRECTORY_SYSAUTH` parameter enables `SYSDBA` and `SYSOPER` users to authenticate to the database by using a strong authentication method.
4. Set the `LDAP_DIRECTORY_ACCESS` parameter to either `PASSWORD` or `SSL`. For example:

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_ACCESS = PASSWORD;
   ```

   Ensure that the `LDAP_DIRECTORY_ACCESS` initialization parameter is not set to `NONE`. Setting this parameter to `PASSWORD` or `SSL` ensures that users can be authenticated using the `SYSDBA` or `SYSOPER` administrative privileges through Oracle Internet Directory.

Afterward, this user can log in by including the net service name in the `CONNECT` statement in SQL*Plus. For example, to log on as `SYSDBA` if the net service name is `orcl`:

```
CONNECT SOMEUSER@ORCL AS SYSDBA
Enter password: password
```

If the database is configured to use a password file for remote authentication, Oracle Database checks the password file first.

See Also:

- Guidelines for Securing User Accounts and Privileges for advice on granting privileges to trusted users
- *Oracle Database Reference* for more information about `LDAP_DIRECTORY_SYSAUTH`
- *Oracle Database Reference* for more information about `LDAP_DIRECTORY_ACCESS`

Configuring Kerberos Authentication for Administrative Users

Oracle Internet Directory can be used to configure Kerberos authentication for administrative users.

1. Configure the administrative user by using the same procedures you would use to configure a typical user.

   See Configuring Kerberos Authentication , for more information.
2. Configure Oracle Internet Directory for Kerberos authentication.

See *Oracle Database Enterprise User Security Administrator's Guide* for more information.

3. In Oracle Internet Directory, grant the `SYSDBA` or `SYSOPER` administrative privilege to the user for the database that this user will administer.

   Grant `SYSDBA` or `SYSOPER` only to trusted users. See Guidelines for Securing User Accounts and Privileges for advice on this topic.

4. Set the `LDAP_DIRECTORY_SYSAUTH` initialization parameter to `YES`:

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_SYSAUTH = YES;
   ```

   When set to `YES`, the `LDAP_DIRECTORY_SYSAUTH` parameter enables `SYSDBA` and `SYSOPER` users to authenticate to the database by using strong authentication methods. See *Oracle Database Reference* for more information about `LDAP_DIRECTORY_SYSAUTH`.

5. Set the `LDAP_DIRECTORY_ACCESS` parameter to either `PASSWORD` or `SSL`. For example:

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_ACCESS = SSL;
   ```

   Ensure that the `LDAP_DIRECTORY_ACCESS` initialization parameter is not set to `NONE`. Setting this parameter to `PASSWORD` or `SSL` ensures that users can be authenticated using `SYSDBA` or `SYSOPER` through Oracle Internet Directory. See *Oracle Database Reference* for more information about `LDAP_DIRECTORY_ACCESS`.

Afterward, this user can log in by including the net service name in the `CONNECT` statement in SQL*Plus. For example, to log on as `SYSDBA` if the net service name is `orcl`:

```
CONNECT /@orcl AS SYSDBA
```

Configuring Secure Sockets Layer Authentication for Administrative Users

Both the client and server side can authenticate administrative users with Secure Sockets Layer (SSL).

1. Configure the client to use SSL:

   a. Configure the client wallet and user certificate. Update the wallet location in the `sqlnet.ora` configuration file.

      You can use Wallet Manager to configure the client wallet and user certificate. See *Oracle Database Enterprise User Security Administrator's Guide* for more information.

   b. Configure the Oracle net service name to include server DNs and use TCP/IP with SSL in `tnsnames.ora`.

   c. Configure TCP/IP with SSL in `listener.ora`.

   d. Set the client SSL cipher suites and the required SSL version, and then set SSL as an authentication service in `sqlnet.ora`.

2. Configure the server to use SSL:

   a. Enable SSL for your database listener on TCPS and provide a corresponding TNS name. You can use Net Configuration Assistant to configure the TNS name.

   b. Store the database PKI credentials in the database wallet. You can use Wallet Manager do this.

c. Set the `LDAP_DIRECTORY_ACCESS` initialization parameter to `SSL`:

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_ACCESS = SSL;
   ```

   See *Oracle Database Reference* for more information about `LDAP_DIRECTORY_ACCESS`.

3. Configure Oracle Internet Directory for SSL user authentications.

   See *Oracle Database Enterprise User Security Administrator's Guide* for information about configuring enterprise user security SSL authentication.

4. In Oracle Internet Directory, grant the `SYSDBA` or `SYSOPER` privilege to the user for the database that the user will administer.

5. On the server computer, set the `LDAP_DIRECTORY_SYSAUTH` initialization parameter to `YES`.

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_SYSAUTH = YES;
   ```

   When set to `YES`, the `LDAP_DIRECTORY_SYSAUTH` parameter enables `SYSDBA` and `SYSOPER` users to authenticate to the database by using a strong authentication method. See *Oracle Database Reference* for more information about `LDAP_DIRECTORY_SYSAUTH`.

Afterward, this user can log in by including the net service name in the `CONNECT` statement in SQL*Plus. For example, to log on as `SYSDBA` if the net service name is `orcl`:

```
CONNECT /@orcl AS SYSDBA
```

Authentication of Database Administrators by Using the Operating System

For both Windows and UNIX systems, you use `DBA`-privileged groups to authenticate for the operating system.

Operating system authentication for a database administrator typically involves establishing a group on the operating system, granting `DBA` privileges to that group, and then adding the names of persons who should have those privileges to that group. (On UNIX systems, the group is the **dba** group.)

Note:

In a multitenant environment, you can use operating system authentication for a database administrator only for the CDB root. You cannot use it for for PDBs, the application root, or application PDBs.

On Microsoft Windows systems:

- Users who connect with the `SYSDBA` administrative privilege can take advantage of the Windows native authentication. If these users work with Oracle Database using their domain accounts, then you must explicitly grant them local administrative privileges and `ORA_DBA` membership.

- Oracle recommends that you run Oracle Database services using a low privileged Microsoft Windows user account rather than a Microsoft Windows built-in account.

> **See Also:**
>
> - *Oracle Database Platform Guide for Microsoft Windows* for information about the Windows-specific operating system groups
> - *Oracle Database Platform Guide for Microsoft Windows* for information about Oracle Database services on Windows
> - *Oracle Database Platform Guide for Microsoft Windows* for information about using the Oracle Home User virtual users and groups in Windows
> - Your Oracle Database operating system-specific documentation for information about configuring operating system authentication of database administrators

Authentication of Database Administrators by Using Their Passwords

Password files are used to authenticate database administrators.

That is, Oracle Database users who have been granted the `SYSDBA`, `SYSOPER`, `SYSASM`, `SYSBACKUP`, `SYSDG`, and `SYSKM` administrative privileges are first authenticated using database-specific password files.

These privileges enable the following activities:

- The `SYSOPER` system privilege lets database administrators perform `STARTUP`, `SHUTDOWN`, `ALTER DATABASE OPEN/MOUNT`, `ALTER DATABASE BACKUP`, `ARCHIVE LOG`, and `RECOVER` operations. `SYSOPER` also includes the `RESTRICTED SESSION` privilege.
- The `SYSDBA` administrative privilege has all system privileges with `ADMIN OPTION`, including the `SYSOPER` administrative privilege, and permits `CREATE DATABASE` and time-based recovery.
- A password file containing users who have the `SYSDBA`, `SYSOPER`, `SYSASM`, `SYSBACKUP`, `SYSDG`, and `SYSKM` administrative privileges can be shared between different databases. In addition, this type of password file authentication can be used in a Secure Sockets Layer (SSL) or Kerberos configuration, and for common administrative users in a multitenant environment. You can have a shared password file that contains users in addition to the `SYS` user. To share a password file among different databases, set the `REMOTE_LOGIN_PASSWORDFILE` parameter in the `init.ora` file to `SHARED`.

  If you set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter to `EXCLUSIVE` or `SHARED` from `NONE`, then ensure that the password file is synchronized with the dictionary passwords. See *Oracle Database Administrator's Guide* for more information.
- For Automatic Storage Management (ASM) environments, you can create shared ASM password files. Remember that you must have the `SYSASM` system privilege to create an ASM password file. See *Oracle Automatic Storage Management Administrator's Guide* for more information.

- The SYSDG administrative privilege must be included in a password file for sharding administrators to perform tasks that involve file transfer and Oracle Recovery Manager (RMAN) activities.
- Password file-based authentication is enabled by default. This means that the database is ready to use a password file for authenticating users that have SYSDBA, SYSOPER, SYSASM, SYSBACKUP, SYSDG, and SYSKM administrative privileges. Password file-based authentication is activated as soon as you create a password file by using the ORAPWD utility.

  Anyone who has EXECUTE privileges and write privileges to the $ORACLE_HOME/dbs directory can run the ORAPWD utility.
- Password limits such as FAILED_LOGIN_ATTEMPTS and PASSWORD_LIFE_TIME are enforced for administrative logins, if the password file is created in the Oracle Database 12*c* release 2 (12.2) format.

Note:

- To find a list of users who are included in the password file, you can query the V$PWFILE_USERS data dictionary view.
- Connections requested AS SYSDBA or AS SYSOPER must use these phrases. Without them, the connection fails. The Oracle Database parameter O7_DICTIONARY_ACCESSIBILITY can be set to FALSE to limit sensitive data dictionary access only to authorized users. The parameter also enforces the required AS SYSDBA or AS SYSOPER syntax.

Risks of Using Password Files for Database Administrator Authentication

Be aware that using password files may pose security risks.

For this reason, consider using the authentication methods described in Strong Authentication, Centralized Management for Administrators.

Examples of password security risks are as follows:

- An intruder could steal or attack the password file.
- Many users do not change the default password.
- The password could be easily guessed.
- The password is vulnerable if it can be found in a dictionary.
- Passwords that are too short, chosen perhaps for ease of typing, are vulnerable if an intruder obtains the cryptographic hash of the password.

Note:

Oracle Database Administrator’s Guide for information about creating and maintaining password files

Database Authentication of Users

Database authentication of users entails using information within the database itself to perform the authentication.

About Database Authentication

Oracle Database can authenticate users attempting to connect to a database by using information stored in that database itself.

To configure Oracle Database to use database authentication, you must create each user with an associated password. User names can use the National Language Support (NLS) character format, but you cannot include double quotation mark characters in the password. The user must provide this user name and password when attempting to establish a connection.

Oracle Database generates a one-way hash of the user's password and stores it for use when verifying the provided login password. In order to support older clients, Oracle Database can be configured to generate the one-way hash of the user's password using a variety of different hashing algorithms. The resulting password hashes are known as password versions, which have the short names `10G`, `11G`, and `12C`. The short names `10G`, `11G`, and `12C` serve as abbreviations for the details of the one-way password hashing algorithms, which are described in more detail in the documentation for the `PASSWORD_VERSIONS` column of the `DBA_USERS` view. To find the list of password versions for any given user, query the `PASSWORD_VERSIONS` column of the `DBA_USERS` view.

By default, there are currently two versions of the one-way hashing algorithm in use in Oracle Database 12*c* release 2 (12.2): the salted SHA-1 hashing algorithm, and the salted PKBDF2 SHA-2 SHA-512 hashing algorithm. The salted SHA-1 hashing algorithm generates the hash that is used for the `11G` password version. The salted PKBDF2 SHA-2 SHA-512 hashing algorithm generates the hash that is used for the `12C` password version. This hash generation takes place for the same password; that is, both algorithms run for the same password. Oracle Database records these password versions in the `DBA_USERS` data dictionary view. When you query this view, you will see two password versions. For example:

```
SELECT USERNAME, PASSWORD_VERSIONS FROM DBA_USERS;

USERNAME  PASSWORD_VERSIONS
--------  -----------------
ADAMS     11G, 12C
SYS       11G, 12C
...
```

To specify which authentication protocol to allow during authentication of a client or of a database server acting as a client, you can explicitly set the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter in the server `sqlnet.ora` file. (The client version of this parameter is `SQLNET.ALLOWED_LOGON_VERSION_CLIENT`.) Each connection attempt is tested, and if the client or server does not meet the client ability requirements specified by its partner, authentication fails with an `ORA-28040 No matching authentication protocol` error in the “Ability Required of the Client” in the “SQLNET.ALLOWED_LOGON_VERSION_SERVER Settings” table under the description of the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter in *Oracle Database Net Services Reference*. The parameter can take the values `12a`, `12`, `11`, `10`, `9`, or `8`. The default value is `12`, which is Exclusive Mode. These values represent the version of the authentication protocol. Oracle recommends the value `12`. However, be aware that if you set `SQLNET.ALLOWED_LOGON_VERSION_SERVER` and `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` to `11`, then pre-Oracle Database Release 11.1 client applications including JDBC thin clients cannot authenticate to the Oracle database using password-based authentication.

To enhance security when using database authentication, Oracle recommends that you use password management, including account locking, password aging and expiration, password history, and password complexity verification.

See Also:

- *Oracle Database Net Services Reference* for more information about the `SQLNET.ALLOWED_LOGON_VERSION_CLIENT` parameter and the `ORA-28040 No matching authentication protocol` error
- *Oracle Database Net Services Reference* for more information about the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter and Exclusive Mode
- About Password Complexity Verification for information about password complexity verification functions
- Using a Password Management Policy for more information about password management
- Management of Password Versions of Users for more information about managing password versions

Advantages of Database Authentication

There are three advantages of using the database to authenticate users.

These advantages are as follows:

- User accounts and all authentication are controlled by the database. There is no reliance on anything outside of the database.
- Oracle Database provides strong password management features to enhance security when using database authentication.
- It is easier to administer when there are small user communities.

Creating Users Who Are Authenticated by the Database

When you create a user who is authenticated by the database, you assign this user a password.

- To create a user who is authenticated by the database, include the `IDENTIFIED BY` clause when you create the user.

For example, the following SQL statement creates a user who is identified and authenticated by Oracle Database. User `sebastian` must specify the assigned password whenever he connects to Oracle Database.

```
CREATE USER sebastian IDENTIFIED BY password;
```

Schema Only Accounts

You can create schema only accounts, that is, the schema user has no password.

About Schema Only Accounts

A schema only account cannot log in to the database but can proxy in a single session proxy.

This type of account, designed for some Oracle-provided schemas along with some customer schemas, can be created without the specification of a password or an authentication type. It cannot be authenticated unless an authentication method is assigned by using the `ALTER USER` statement. A schema only account does not contain an entry in the `DBA_USERS_WITH_DEFPWD` data dictionary view. Note the following:

- Schema only accounts can be used for both administrator and non-administrator accounts.
- Schema only accounts can be created on the database instance only, not in Oracle Automatic Storage Management (ASM) environments.
- You can grant system privileges (such as `CREATE ANY TABLE`) and administrator roles (such as `DBA`) to schema only accounts. Schema only accounts can create objects such as tables or procedures, assuming they have had to correct privileges granted to them.
- You cannot grant the `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSKM`, `SYSASM`, `SYSRAC`, and `SYSDG` administrative privileges to schema only accounts.
- You can configure schema only accounts to be used as client users in a proxy authentication in a single session proxy. This is because in a single session proxy, only the credentials of the proxy user are verified, not the credentials of the client user. Therefore, a schema only account can be a client user. However, you cannot configure schema only accounts for a two-proxy scenario, because the client credentials must be verified. Hence, the authentication for a schema only account will fail.
- Schema only accounts cannot connect through database links, either with connected user links, fixed user links, or current user links.

Creating a Schema Only Account

The `CREATE USER` SQL statement creates schema only accounts.

You can run the `CREATE USER` statement with the `NO AUTHENTICATION` clause only on a database instance. You cannot run it on an Oracle Automatic Storage Management (ASM) instance.

- Use the `CREATE USER` statement with the `NO AUTHENTICATION` clause.

  For example:

  ```
  CREATE USER psmith NO AUTHENTICATION;
  ```

Altering a Schema Only Account

The `ALTER USER` SQL statement can be used to modify schema only accounts.

1. Check if the schema user has administrative privileges.

   You can query the `V$PWFILE_USERS` to find if the schema user has administrative privileges.
2. If the schema user has administrative privileges, then use the `REVOKE` statement to revoke these privileges.
3. Use the `ALTER USER` SQL statement with the `NO AUTHENTICATION` clause to modify the schema account to have no authentication.

   For example:

   ```
   ALTER USER psmith NO AUTHENTICATION;
   ```

You can use `ALTER USER` to enable authentication for a schema only account.

Operating System Authentication of Users

Oracle Database can authenticate by using information that is maintained by the operating system.

Using the operating system to authenticate users has both advantages and disadvantages.

This functionality has the following benefits:

- Once authenticated by the operating system, users can connect to Oracle Database more conveniently, without specifying a user name or password. For example, an operating system-authenticated user can invoke SQL*Plus and omit the user name and password prompts by entering the following command at the command line:

  ```
  SQLPLUS /
  ```

  Within SQL*Plus, you enter:

  ```
  CONNECT /
  ```
- With control over user authentication centralized in the operating system, Oracle Database does not need to store or manage the cryptographic hashes (also called verifiers) of the user passwords, although it still maintains user names in the database.
- The audit trail captures the operating system user name and the database user name, where the database user name is the value of the `OS_AUTHENT_PREFIX` instance initialization parameter prefixed to the operating system user name. For example, if `OS_AUTHENT_PREFIX` is set to `OPS$` and the operating system user name is `psmith`, then the database user name will be `OPS$PSMITH`.
- You can authenticate both operating system and non-operating system users in the same system. For example:
  – **Authenticate users by the operating system.** You create the user account using the `IDENTIFIED EXTERNALLY` clause of the `CREATE USER` statement, and then you set the `OS_AUTHENT_PREFIX` initialization parameter to specify a prefix that Oracle Database uses to authenticate users attempting to connect to the server.
  – **Authenticate non-operating system users.** These are users who are assigned passwords and authenticated by the database.

  – **Authenticate Oracle Database Enterprise User Security users.** These user accounts where created using the `IDENTIFIED GLOBALLY` clause of the `CREATE USER` statement, and then authenticated by Oracle Internet Directory (OID) currently in the same database.

However, you should be aware of the following drawbacks to using the operating system to authenticate users:

- A user must have an operating system account on the computer that must be accessed. Not all users have operating system accounts, particularly non-administrative users.
- If a user has logged in using this method and steps away from the terminal, another user could easily log in because this user does not need any passwords or credentials. This could pose a serious security problem.
- When an operating system is used to authenticate database users, managing distributed database environments and database links requires special care. Operating system-authenticated database links can pose a security weakness. For this reason, Oracle recommends that you do not use them.
- In a multitenant environment, you can use operating system authentication for a database administrator only for the CDB root. You cannot use it for PDBs, the application root, or application PDBs.

See Also:

- *Oracle Database Administrator's Guide* for more information about authentication, operating systems, distributed database concepts, and distributed data management
- Operating system-specific documentation by Oracle Database for more information about authenticating by using your operating system

Network Authentication of Users

You can authenticate users over a network by using Secure Sockets Layer with third-party services.

Authentication with Secure Sockets Layer

The Secure Sockets Layer (SSL) protocol is an application layer protocol.

You can use SSL for user authentication to a database, and it is independent of global user management in Oracle Internet Directory. That is, users can use SSL to authenticate to the database without a directory server in place.

Authentication with Third-Party Services

The third-party services Kerberos, RADIUS, directory-based services, and public key infrastructure can authenticate Oracle Database over a network.

About Authentication Using Third-Party Services

You must use third-party network authentication services if you want to authenticate Oracle Database users over a network.

Prominent examples include Kerberos, PKI (public key infrastructure), the RADIUS (Remote Authentication Dial-In User Service), and directory-based services.

If network authentication services are available to you, then Oracle Database can accept authentication from the network service. If you use a network authentication service, then some special considerations arise for network roles and database links.

Authentication with Kerberos

Kerberos is a trusted third-party authentication system that relies on shared secrets.

Kerberos presumes that the third party is secure, and provides single sign-on capabilities, centralized password storage, database link authentication, and enhanced PC security. It does this through a Kerberos authentication server, or through Cybersafe Active Trust, a commercial Kerberos-based authentication server.

Authentication with RADIUS

Remote Authentication Dial-In User Service (RADIUS) is a standard lightweight protocol used for user authentication, authorization, and accounting.

RADIUS also enables users to use the RSA One-Time Password Specifications (OTPS) to authenticate to the Oracle database.

See Also:

- Configuring RADIUS Authentication for information about configuring RADIUS
- RSA documentation about OTPS

Authentication with Directory-Based Services

Using a central directory can make authentication and its administration efficient.

Directory-based services include the following:

- **Oracle Internet Directory**, which uses the Lightweight Directory Access Protocol (LDAP), uses a central repository to store and manage information about users (called enterprise users) whose accounts were created in a distributed environment. Although database users must be created (with passwords) in each database that they need to access, enterprise user information is accessible centrally in the Oracle Internet Directory. You can also integrate this directory with Microsoft Active Directory and SunOne.
- **Oracle Enterprise Security Manager** lets you store and retrieve roles from Oracle Internet Directory, which provides centralized privilege management to make administration easier and increase security levels.

Authentication with Public Key Infrastructure

Authentication systems based on public key infrastructure (PKI) issue digital certificates to user clients.

These clients can use these certificates to authenticate directly to servers in the enterprise without directly involving an authentication. Oracle Database provides a PKI for using public keys and certificates, consisting of the following components:

- **Authentication and secure session key management using SSL.** See Authentication with Secure Sockets Layer for more information.
- **Trusted certificates.** These are used to identify third-party entities that are trusted as signers of user certificates when an identity is being validated. When the user certificate is being validated, the signer is checked by using trust points or a trusted certificate chain of certificate authorities stored in the validating system. If there are several levels of trusted certificates in this chain, then a trusted certificate at a lower level is simply trusted without needing to have all its higher-level certificates reverified.
- **Oracle Wallet Manager.** An Oracle wallet is a data structure that contains the private key of a user, a user certificate, and the set of trust points of a user (trusted certificate authorities). See *Oracle Database Enterprise User Security Administrator's Guide* for information about managing Oracle wallets.

  You can use Oracle Wallet Manager to manage Oracle wallets. This is a standalone Java application used to manage and edit the security credentials in Oracle wallets. It performs the following operations:

  – Generates a public-private key pair and creates a certificate request for submission to a certificate authority, and creates wallets
  – Installs a certificate for the entity
  – Manages X.509 version 3 certificates on Oracle Database clients and servers
  – Configures trusted certificates for the entity
  – Opens a wallet to enable access to PKI-based services
- **X.509 version 3 certificates obtained from (and signed by) a trusted entity, a certificate authority.** Because the certificate authority is trusted, these certificates verify that the requesting entity's information is correct and that the public key on the certificate belongs to the identified entity. The certificate is loaded into an Oracle wallet to enable future authentication.

Configuring Operating System Users for a PDB

The `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure configures user accounts to be operating system users for a PDB.

About Configuring Operating System Users for a PDB

Instead the `oracle` operating system user, you can set a specific user account to be the operating system user for that PDB.

If you do not set a specific user to be the operating system user for the PDB, then by default the PDB uses the `oracle` operating system user. For the root, you can use the `oracle` operating system user when you must interact with the operating system.

For better security, Oracle recommends that you set a unique operating system user for each PDB in a multitenant environment. Doing so helps to ensure that operating system interactions are performed as a less powerful user than the `oracle` operating system user, and helps to protect data that belongs to one PDB from being accessed by users who are connected to other PDBs.

Configuring an Operating System User for a PDB

The `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure can set an operating system user for a PDB.

1. Log in to the database instance root as a user who has the `EXECUTE` privilege for the `DBMS_CREDENTIAL` PL/SQL package and the `ALTER SYSTEM` system privilege.

   For example:

   ```
   sqlplus c##sec_admin
   Enter password: password
   ```

2. Run the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure to create an Oracle credential for the operating system user.

   For example, to set the credential for a user named `os_admin`:

   ```
   BEGIN
    DBMS_CREDENTIAL.CREATE_CREDENTIAL (
       credential_name => 'PDB1_OS_USER',
       username        => 'os_admin',
       password        => 'password');
   END;
   /
   ```

   Follow the guidelines in Minimum Requirements for Passwords to replace *`password`* with a password that is secure.

3. Connect to the PDB for which the operating system user will be used.

   For example:

   ```
   CONNECT cc##sec_admin@hrpdb
   Enter password: password
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the show `con_name` command.

4. Set the `PDB_OS_CREDENTIAL` initialization parameter for the user whose credential was set in Step 2.

   For example:

   ```
   ALTER SYSTEM SET PDB_OS_CREDENTIAL = PDB1_OS_USER SCOPE = SPFILE;
   ```

   The `PDB_OS_CREDENTIAL` parameter is a static parameter, so you must set it using the `SCOPE = SPFILE` clause.

5. Restart the database instance.

   ```
   SHUTDOWN IMMEDIATE
   STARTUP
   ```

See also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure
- *Oracle Database Reference* for more information about the `PDB_OS_CREDENTIAL` initialization parameter

Global User Authentication and Authorization

Global user authentication and authorization enables you to centralize the management of user-related information.

About Configuring Global User Authentication and Authorization

An LDAP-based directory service centralizes the management of user-related information, including authorizations.

This enables users and administrators to be identified in the database as global users, meaning that they are authenticated by SSL and that the management of these users is handled outside of the database by the centralized directory service. Global roles are defined in a database and are known only to that database, but the directory service handles authorizations for global roles.

Note:

You can also have users authenticated by Secure Sockets Layer (SSL), whose authorizations are not managed in a directory, that is, they have local database roles only.

This centralized management enables the creation of **enterprise users** and **enterprise roles**. Enterprise users are defined and managed in the directory. They have unique identities across the enterprise and can be assigned enterprise roles that determine their access privileges across multiple databases. An enterprise role consists of one or more global roles, and might be thought of as a container for global roles.

See Also:

- Configuring Secure Sockets Layer Authentication for details about Secure Sockets Layer authentication.
- Strong Authentication, Centralized Management for Administrators if you want to centralize the management of `SYSDBA` or `SYSOPER` access

Configuration of Users Who Are Authorized by a Directory Service

You can configure either a global user or multiple enterprise users to be authorized by a directory service.

Creating a Global User Who Has a Private Schema

You can create a user account who has a private schema by providing an identifier (**distinguished name**, or **DN**) meaningful to the enterprise directory.

However, be aware that you must create this user in every database that the user must access, plus the directory.

- To create a global user who has a private schema, use the `CREATE USER ... IDENTIFIED GLOBALLY` SQL statement.

  You can include standard LDAP Data Interchange Format (LDIF) fields. For example, to create a global user (`psmith_gl` with a private schema, authenticated by SSL, and authorized by the enterprise directory service:

  ```
  CREATE USER psmith_gl IDENTIFIED GLOBALLY AS
  'CN=psmith,OU=division1,O=example,C=US';
  ```

  In this specification:

  – `CN` refers to the common name of this user, `psmith_gl`.

  – `OU` refers to the user's organizational unit, `division1`.

  – `O` refers to the user's organization, `Example`.

  – `C` refers to the country in which the organization Example is located, the `US`.

Creating Multiple Enterprise Users Who Share Schemas

Multiple enterprise users can share a single schema in the database.

These users are authorized by the enterprise directory service but do not own individual private schemas in the database. These users are not individually created in the database. They connect to a shared schema in the database.

1. Create a shared schema in the database using the following example:

   ```
   CREATE USER appschema IDENTIFIED GLOBALLY AS '';
   ```

2. In the directory, create multiple enterprise users and a mapping object.

   The mapping object tells the database how you want to map the DNs for the users to the shared schema. You can either create a full distinguished name (DN) mapping (one directory entry for each unique DN), or you can map, for each user, multiple DN components to one schema. For example:

   ```
   OU=division1,O=Example,C=US
   ```

   See Also:

   Oracle Database Enterprise User Security Administrator's Guide for an explanation of these mappings

Most users do not need their own schemas, and implementing schema-independent users separates users from databases. You create multiple users who share the same schema in a database, and as enterprise users, they can also access shared schemas in other databases.

Advantages of Global Authentication and Global Authorization

There are several advantages of global user authentication and authorization.

- Provides strong authentication using SSL, Kerberos, or Windows native authentication.
- Enables centralized management of users and privileges across the enterprise.
- Is easy to administer: You do not have to create a schema for every user in every database in the enterprise.
- Facilitates single sign-on: Users need to sign on once to only access multiple databases and services. Further, users using passwords can have a single password to access multiple databases accepting password-authenticated enterprise users.
- Because global user authentication and authorization provide password-based access, you can migrate previously defined password-authenticated database users to the directory (using the User Migration Utility) to be centrally administered. This makes global authentication and authorization available for earlier Oracle Database release clients that are still supported.
- `CURRENT_USER` database links connect as a global user. A local user can connect as a global user in the context of a stored procedure, that is, without storing the global user password in a link definition.

> **See Also:**
>
> *Oracle Database Enterprise User Security Administrator's Guide* for additional information about global authentication and authorization and enterprise users and roles

Configuring an External Service to Authenticate Users and Passwords

An external service (the operating system or the network) can administer passwords and authenticate users.

About External Authentication

With external authentication, Oracle Database maintains the user account, but an external service performs the password administration and user authentication.

This external service can be the operating system or a network service, such as Oracle Net. If you are authenticating users through a password file, then you can configure external authentication for users who have been granted the `SYSDBA`, `SYSOPER`, `SYSASM`, `SYSBACKUP`, `SYSDG`, and `SYSKM` administrative privileges.

With external authentication, your database relies on the underlying operating system or network authentication service to restrict access to database accounts. A database password is not used for this type of login. If your operating system or network service permits, then it can authenticate users before they can log in to the database.

Advantages of External Authentication

External authentication provides several advantages.

These advantages are as follows:

- More choices of authentication mechanisms are available, such as smart cards, fingerprints, Kerberos, or the operating system.
- Many network authentication services, such as Kerberos support single sign-on, enabling users to have fewer passwords to remember.
- If you are already using an external mechanism for authentication, such as one of those listed earlier, then there may be less administrative overhead to use that mechanism with the database.

Enabling External Authentication

To enable external authentication, you can set the initialization parameter `OS_AUTHENT_PREFIX`, and use this prefix in Oracle Database user names.

The `OS_AUTHENT_PREFIX` parameter defines a prefix that Oracle Database adds to the beginning of the operating system account name of every user. Oracle Database compares the prefixed user name with the Oracle Database user names in the database when a user attempts to connect.

1. Set `OS_AUTHENT_PREFIX` to a null string (an empty set of double quotation marks: `""`). Using a null string eliminates the addition of any prefix to operating system account names, so that Oracle Database user names exactly match operating system user names.

   For example:

   ```
   OS_AUTHENT_PREFIX=" "
   ```

2. Ensure that the `OS_AUTHENT_PREFIX`remains the same for the life of a database. If you change the prefix, then any database user name that includes the old prefix cannot be used to establish a connection, unless you alter the user name to have it use password authentication.

The default value of the `OS_AUTHENT_PREFIX` parameter is `OPS$` for backward compatibility with previous versions of Oracle Database. For example, assume that you set `OS_AUTHENT_PREFIX` as follows:

```
OS_AUTHENT_PREFIX=OPS$
```

If a user with an operating system account named `tsmith` is to connect to an Oracle database installation and be authenticated by the operating system, then Oracle Database checks that there is a corresponding database user `OPS$tsmith` and, if so, lets the user connect. All references to a user authenticated by the operating system must include the prefix, `OPS$`, as seen in `OPS$tsmith`.

Note:

The text of the `OS_AUTHENT_PREFIX` initialization parameter is case-sensitive on some operating systems. See your operating system-specific Oracle Database documentation for more information about this initialization parameter.

Creating a User Who Is Authenticated Externally

Externally authenticated users are authenticated by the operating system or network service.

You can create users who are authenticated externally. Oracle Database then relies on this external login authentication when it provides that specific operating system user with access to the database resources of a specific user.

- Use the `IDENTIFIED EXTERNALLY` clause of the `CREATE USER` statement to create users who are authenticated externally.

The following example creates a user who is identified by Oracle Database and authenticated by the operating system or a network service. This example assumes that the `OS_AUTHENT_PREFIX` parameter has been set to a blank space (`" "`).

```
CREATE USER psmith IDENTIFIED EXTERNALLY;
```

Authentication of User Logins By Using the Operating System

Oracle Database allows operating system-authenticated logins only over secure connections, which precludes using Oracle Net and a shared server configuration.

This type of operating system authentication is the default. This restriction prevents a remote user from impersonating another operating system user over a network connection.

Setting the `REMOTE_OS_AUTHENT` parameter to `TRUE` in the database initialization parameter file forces the database to accept the client operating system user name received over an unsecure connection and use it for account access. Because clients, in general, such as PCs, are not trusted to perform operating system authentication properly, it is very poor security practice to turn on this feature.

The default setting, `REMOTE_OS_AUTHENT = FALSE`, creates a more secure configuration that enforces proper, server-based authentication of clients connecting to an Oracle database.

Be aware that the `REMOTE_OS_AUTHENT` parameter was deprecated in Oracle Database 11*g* Release 1 (11.1), and is retained only for backward compatibility.

Any change to this parameter takes effect the next time you start the instance and mount the database. Generally, user authentication through the host operating system offers faster and more convenient connection to Oracle Database without specifying a separate database user name or password. Also, user entries correspond in the database and operating system audit trails.

Authentication of User Logins Using Network Authentication

Oracle strong authentication performs network authentication, which you can configure to use a third-party service such as Kerberos.

If you are using Oracle strong authentication as your only external authentication service, then the `REMOTE_OS_AUTHENT` parameter setting is irrelevant, because Oracle strong authentication permits only secure connections.

Multitier Authentication and Authorization

Oracle Database secures middle-tier applications by limiting privileges, preserving client identities through all tiers, and auditing actions by clients.

In applications that use a very busy middle tier, such as a transaction processing monitor, the identity of the clients connecting to the middle tier must be preserved. One advantage of using a middle tier is **connection pooling**, which allows multiple users to access a data server without each of them needing a separate connection. In such environments, you need to be able to set up and break down connections very quickly.

For these environments, you can use the Oracle Call Interface to create **lightweight sessions**, which enable database password authentication for each user. This method preserves the identity of the real user through the middle tier without the overhead of a separate database connection for each user.

You can create lightweight sessions with or without passwords. However, if a middle tier is outside of or on a firewall, then security is better when each lightweight session has its own password. For an internal application server, lightweight sessions without passwords might be appropriate.

Administration and Security in Clients, Application Servers, and Database Servers

In a multitier environment, an application server provides data for clients and serves as an interface to one or more database servers.

The application server can validate the credentials of a client, such as a Web browser, and the database server can audit operations performed by the application server. These auditable operations include actions performed by the application server on behalf of clients, such as requests that information be displayed on the client. A request to connect to the database server is an example of an application server operation not related to a specific client.

Authentication in a multitier environment is based on trust regions. Client authentication is the domain of the application server. The application server itself is authenticated by the database server. The following operations take place:

- The end user provides proof of authenticity to the application server, typically, by using a password or an X.509 certificate.
- The application server authenticates the end user and then authenticates itself to the database server.

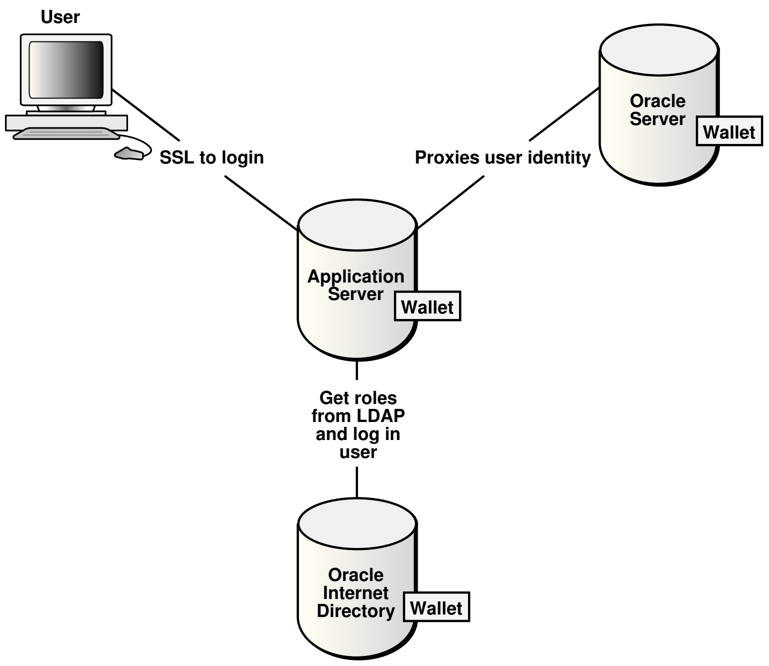
- The database server authenticates the application server, verifies that the end user exists, and verifies that the application server has the privilege to connect for the end user.

Application servers can also enable roles for an end user on whose behalf they connect. The application server can obtain these roles from a directory, which serves as an authorization repository. The application server can only request that these roles be enabled. The database verifies the following requirements:

- That the client has these roles by checking its internal role repository
- That the application server has the privilege to connect on behalf of the user and thus to use these roles as the user could

Figure 3-2 shows an example of multitier authentication.

Figure 3-2 Multitier Authentication



The following actions take place:

1. The user logs on using a password or Secure Sockets Layer. The authentication information is passed through Oracle Application Server.
2. Oracle Internet Directory authenticates the user, gets the roles associated with that user from the wallet, and then passes this information back to Oracle Application Server.
3. Oracle Application Server checks the identity of the user in Oracle Database, which contains a wallet that stores this information, and then sets the role for that user.

Security for middle-tier applications must address the following key issues:

- **Accountability.** The database server must be able to distinguish between the actions of the application and the actions an application takes on behalf of a client. It must be possible to audit both kinds of actions.

- **Least privilege.** Users and middle tiers should be given the fewest privileges necessary to perform their actions, to reduce the danger of inadvertent or malicious unauthorized activities.

Preserving User Identity in Multitiered Environments

You can use middle tier servers for proxy authentication and client identifiers to identify application users who are not known to the database.

Middle Tier Server Use for Proxy Authentication

Oracle Call Interface (OCI), JDBC/OCI, or JDBC Thin Driver supports the middle tier for proxy authentication for database users or enterprise users.

About Proxy Authentication

Oracle Database provides proxy authentication in Oracle Call Interface (OCI), JDBC/OCI, or JDBC Thin Driver for database users or enterprise users.

Enterprise users are those who are managed in Oracle Internet Directory and who access a shared schema in the database.

You can design a middle-tier server to authenticate clients in a secure fashion by using the following three forms of proxy authentication:

- The middle-tier server authenticates itself with the database server and a client, in this case an application user or another application, authenticates itself with the middle-tier server. Client identities can be maintained all the way through to the database.
- The client, in this case a database user, is not authenticated by the middle-tier server. The clients identity and database password are passed through the middle-tier server to the database server for authentication.
- The client, in this case a global user, is authenticated by the middle-tier server, and passes one of the following through the middle tier for retrieving the client's user name.
  - Distinguished name (DN)
  - Certificate

In all cases, an administrator must authorize the middle-tier server to act on behalf of the client.

See Also:

- Auditing SQL Statements and Privileges in a Multitier Environment
- *Oracle Database JDBC Developer’s Guide* for more information about proxy authentication

Advantages of Proxy Authentication

In multitier environments, proxy authentication preserves client identities and privileges through all tiers in middle-tier applications and by auditing client actions.

For example, this feature allows the identity of a user using a Web application (which acts as a proxy) to be passed through the application to the database server.

Three-tier systems provide the following benefits to organizations:

- Organizations can separate application logic from data storage, partitioning the former in application servers and the latter in databases.
- Application servers and Web servers enable users to access data stored in databases.
- Users like using a familiar, easy-to-use browser interface.
- Organizations can also lower their cost of computing by replacing many *thick clients* with numerous *thin clients* and an application server.

In addition, Oracle Database proxy authentication provides the following security benefits:

- A limited trust model, by controlling the users on whose behalf middle tiers can connect and the roles that the middle tiers can assume for the user
- Scalability, by supporting user sessions through OCI, JDBC/OCI, or JDBC Thin driver and eliminating the overhead of reauthenticating clients
- Accountability, by preserving the identity of the real user through to the database, and enabling auditing of actions taken on behalf of the real user
- Flexibility, by supporting environments in which users are known to the database, and in which users are merely application users of which the database has no awareness

Note:

Oracle Database supports this proxy authentication functionality in three tiers only. It does not support it across multiple middle tiers.

Who Can Create Proxy User Accounts?

To create proxy user accounts, users must have special privileges.

These privileges are as follows:

- The `CREATE USER` system privilege to create a database user account that will be used as a proxy user account
- The `DV_ACCTMGR` role if Oracle Database Vault is enabled, to create the proxy user account
- The ability to grant the `CREATE SESSION` system privilege to the proxy user account
- The `ALTER USER` system privilege to enable existing user accounts to connect to the database through the proxy account

Guidelines for Creating Proxy User Accounts

Oracle provides special guidelines for when you create proxy user accounts.

- For better security and to adhere to the principle of least privilege, only grant the proxy user account the `CREATE SESSION` privilege. Do not grant this user any other privileges. The proxy user account is designed to only enable another user to connect using the proxy account. Any privileges that must be exercised during the connection should belong to the connecting user, not to the proxy account.
- As with all passwords, ensure that the password you create for the proxy user is strong and not easily guessed. Remember that multiple users will be connecting as the proxy user, so it is especially important that this password be strong.
- Consider using the Oracle strong authentication network connection features, to prevent network eavesdropping.
- For further fine-tuning of the amount of control that the connecting user has, consider restricting the roles used by the connecting user when he or she is connected through the proxy account. The `ALTER USER` statement `WITH ROLE` clause enables you to configure the user to connect using specified roles, any role except a specified role, or with no roles at all. Be aware that the proxy user can only activate those roles that are included in the `WITH ROLE` clause. The proxy user session will have all the privileges that were directly granted to the client (that is, current) user.

Creating Proxy User Accounts and Authorizing Users to Connect Through Them

The `CREATE USER` and `ALTER USER` statements can be used to create a proxy user and authorize users to connect through it.

1. Use the `CREATE USER` statement to create the proxy user account.

   For example:

   ```
   CREATE USER appuser IDENTIFIED BY password;
   ```

2. Use the `GRANT CONNECT THROUGH` clause of the `ALTER USER` statement to enable an existing user to connect through the proxy user account.

   For example:

   ```
   ALTER USER preston GRANT CONNECT THROUGH appuser;
   ```

   Be aware that the user name and proxy combination must not exceed 250 characters.

   Suppose user `preston` has a large number of roles, but you only want her to use one role (for example, the `appuser_role`) when she is connected to the database through the `appuser` proxy account. You can use the following `ALTER USER` statement:

   ```
   ALTER USER preston GRANT CONNECT THROUGH appuser WITH ROLE appuser_role;
   ```

   Any other roles that user `preston` has will not be available to her as long as she is connecting as the `appuser` proxy.

After you complete these steps, user `preston` can connect using the `appuser` proxy user as follows:

```
CONNECT appuser[preston]
Enter password: appuser_password
```

See Also:

- *Oracle Database SQL Language Reference* for detailed information about the `CREATE USER` statement
- *Oracle Database SQL Language Reference* for detailed information about the `ALTER USER` statement

Proxy User Accounts and the Authorization of Users to Connect Through Them

The `CREATE USER` statement enables you to create the several types of user accounts, all of which can be used as proxy accounts.

These accounts are as follows:

- Database user accounts, which are authenticated by passwords
- External user accounts, which are authenticated by external sources, such as Secure Socket Layer (SSL) or Kerberos
- Global user accounts, which are authenticated by an enterprise directory service (Oracle Internet Directory).

Note the following:

- **The proxy user can only perform activities that the user `preston` has privileges to perform.** Remember that the proxy user itself, `appuser`, only has the minimum privileges (`CREATE SESSION`).
- **Using roles with middle-tier clients.** You can also specify roles that the middle tier is permitted to activate when connecting as the client. Operations performed on behalf of a client by a middle-tier server can be audited.
- **Finding proxy users.** To find the users who are currently authorized to connect through a middle tier, query the `PROXY_USERS` data dictionary view, for example:

  ```
  SELECT * FROM PROXY_USERS;
  ```

- **Removing proxy connections.** Use the `REVOKE CONNECT THROUGH` clause of `ALTER USER` to disallow a proxy connection. For example, to revoke user `preston` from connecting through the proxy user `appuser`, enter the following statement:

  ```
  ALTER USER preston REVOKE CONNECT THROUGH appuser;
  ```

- **Password expiration and proxy connections.** Middle-tier use of password expiration does not apply to accounts that are authenticated through a proxy. Instead, lock the account rather than expire the password.

See Also:

- *Oracle Database Enterprise User Security Administrator's Guide* for information about managing proxy users in an enterprise user environment
- Auditing SQL Statements and Privileges in a Multitier Environment for details about auditing operations done on behalf of a user by a middle tier

Using Proxy Authentication with the Secure External Password Store

Use a secure external password store if you are concerned about the password used in proxy authentication being obtained by a malicious user.

To accomplish this, you use the secure external password store with the proxy authentication to store the password credentials in a wallet.

Connecting to Oracle Database using proxy authentication and the secure external password store is ideal for situations such as running batch files. When a proxy user connects to the database and authenticates using a secure external password, the password is not exposed in the event that a malicious user tries to obtain the password.

To use proxy authentication with the secure external password store:

1. Configure the proxy authentication account, as shown in the procedure in Proxy User Accounts and the Authorization of Users to Connect Through Them.
2. Configure the secure external password store, as described in About Configuring Clients to Use the External Password Store.

Afterward, the user can connect using the proxy but without having to specify a password. For example:

```
sqlplus [preston]/@db_alias
```

When you use the secure external password store, the user logging in does not need to supply the user name and password. Only the `SERVICE_NAME` value (that is, `db_alias`) from the `tnsnames.ora` file must be specified.

How the Identity of the Real User Is Passed with Proxy Authentication

You can use Oracle Call Interface, JDBC/OCI, or Thin drivers for enterprise users or database users.

These tools enable a middle tier to set up several user sessions within a single database connection, each of which uniquely identifies a connected user (connection pooling)

These sessions reduce the network overhead of creating separate network connections from the middle tier to the database.

If you want to authenticate from clients through a middle tier to the database, then the full authentication sequence from the client to the middle tier to the database occurs as follows:

1. The client authenticates to the middle tier, using whatever form of authentication the middle tier will accept. For example, the client could authenticate to the middle tier by using a user name and password or an X.509 certificate by means of SSL.
2. The middle tier authenticates itself to the database by using whatever form of authentication the database accepts. This could be a password or an authentication mechanism supported by Oracle Database, such as a Kerberos ticket or an X.509 certificate (SSL).
3. The middle tier then creates one or more sessions for users using OCI, JDBC/OCI, or Thin driver.
   - If the user is a database user, then the session must, as a minimum, include the database user name. If the database requires it, then the session can include a password (which the database verifies against the password store in the database). The session can also include a list of database roles for the user.
   - If the user is an enterprise user, then the session may provide different information depending on how the user is authenticated.

     Example 1: If the user authenticates to the middle tier using SSL, then the middle tier can provide the DN from the X.509 certificate of the user, or the certificate itself in the session. The database uses the DN to look up the user in Oracle Internet Directory.

     Example 2: If the user is a password-authenticated enterprise user, then the middle tier must provide, as a minimum, a globally unique name for the user. The database uses this name to look up the user in Oracle Internet Directory. If the session also provides a password for the user, then the database will verify the password against Oracle Internet Directory. User roles are automatically retrieved from Oracle Internet Directory after the session is established.
   - The middle tier may optionally provide a list of database roles for the client. These roles are enabled if the proxy is authorized to use the roles on behalf of the client.
4. The database verifies that the middle tier has the privilege to create sessions on behalf of the user.

   The `OCISessionBegin` call fails if the application server cannot perform a proxy authentication on behalf of the client by the administrator, or if the application server is not allowed to activate the specified roles.

Limits to the Privileges of the Middle Tier

Least privilege is the principle that users should have the fewest privileges necessary to perform their duties and no more.

As applied to middle tier applications, this means that the middle tier should not have more privileges than it needs.

Oracle Database enables you to limit the middle tier such that it can connect only on behalf of certain database users, using only specific database roles. You can limit the privilege of the middle tier to connect on behalf of an enterprise user, stored in an LDAP directory, by granting to the middle tier the privilege to connect as the mapped database user. For instance, if the enterprise user is mapped to the `APPUSER` schema, then you must at least grant to the middle tier the ability to connect on behalf of `APPUSER`. Otherwise, attempts to create a session for the enterprise user will fail.

However, you cannot limit the ability of the middle tier to connect on behalf of enterprise users. For example, suppose that user Sarah wants to connect to the database through a middle tier, `appsrv` (which is also a database user). Sarah has multiple roles, but it is desirable to restrict the middle tier to use only the `clerk` role on her behalf.

An administrator can grant permission for `appsrv` to initiate connections on behalf of Sarah using her `clerk` role only by using the following SQL statement:

```
ALTER USER sarah GRANT CONNECT THROUGH appsrv WITH ROLE clerk;
```

By default, the middle tier cannot create connections for any client. The permission must be granted for each user.

To enable `appsrv` to use all of the roles granted to the client Sarah, you can use the following statement:

```
ALTER USER sarah GRANT CONNECT THROUGH appsrv;
```

Each time a middle tier initiates an OCI, JDBC/OCI, or Thin driver session for another database user, the database verifies that the middle tier is authorized to connect for that user by using the role specified.

Note:

Instead of using default roles, create your own roles and assign only necessary privileges to them. Creating your own roles enables you to control the privileges granted by them and protects you if Oracle Database changes or removes default roles. For example, the `CONNECT` role now has only the `CREATE SESSION` privilege, the one most directly needed when connecting to a database.

However, `CONNECT` formerly provided several additional privileges, often not needed or appropriate for most users. Extra privileges can endanger the security of your database and applications. These have now been removed from `CONNECT`.

Authorizing a Middle Tier to Proxy and Authenticate a User

You can authorize a middle-tier server to connect as a user.

- To authorize a middle-tier server to connect as a user, use the `ALTER USER` statement.

The following statement authorizes the middle-tier server `appserve` to connect as user `bill`. It uses the `WITH ROLE` clause to specify that `appserve` activate all roles associated with `bill`, except `payroll`.

```
ALTER USER bill
    GRANT CONNECT THROUGH appserve
    WITH ROLE ALL EXCEPT payroll;
```

To revoke the middle-tier server (`appserve`) authorization to connect as user `bill`, you can use the `REVOKE CONNECT THROUGH` clause. For example:

```
ALTER USER bill REVOKE CONNECT THROUGH appserve;
```

Authorizing a Middle Tier to Proxy a User Authenticated by Other Means

You can authorize a middle tier to proxy a user that has been authenticated by other means.

Currently, `PASSWORD` is the only means supported.

- Use the `AUTHENTICATION REQURED` clause of the `ALTER USER ... GRANT CONNECT THROUGH` statement to authorize a user to be proxied, but not authenticated, by a middle tier.

For example:

```
ALTER USER mary
    GRANT CONNECT THROUGH midtier
    AUTHENTICATION REQUIRED;
```

In the preceding statement, middle-tier server `midtier` is authorized to connect as user `mary`, and `midtier` must also pass the user password to the database server for authorization.

Reauthenticating a User Through the Middle Tier to the Database

You can specify that authentication is required by using the `AUTHENTICATION REQUIRED` proxy clause with the `ALTER USER` SQL statement.

In this case, the middle tier must provide user authentication credentials.

For example, suppose that user Sarah wants to connect to the database through a middle tier, `appsrv`.

- To require that `appsrv` provides authentication credentials for the user Sarah, use the following syntax:

  ```
  ALTER USER sarah GRANT CONNECT THROUGH appsrv AUTHENTICATION REQUIRED;
  ```

The `AUTHENTICATION REQUIRED` clause ensures that authentication credentials for the user must be presented when the user is authenticated through the specified proxy.

> **Note:**
>
> For backward compatibility, if you use the `AUTHENTICATED USING PASSWORD` proxy clause, then Oracle Database transforms it to `AUTHENTICATION REQUIRED`.

Using Password-Based Proxy Authentication

When you use password-based proxy authentication, Oracle Database passes the password of the client to the middle-tier server.

The middle-tier server then passes the password as an attribute to the data server for verification.

The main advantage to this type of authentication is that the client computer does not have to have Oracle software installed on it to perform database operations.

- To pass the password of the client, configure the the middle-tier server to call the `OCIAttrSet()` function as follows, passing `OCI_ATTR_PASSWORD` as the type of the attribute being set.

  ```
  OCIAttrSet(
    session_handle,    /* Pointer to a handle whose attribute gets modified. */
    OCI_HTYPE_SESSION, /* Handle type: OCI user session handle. */
    password_ptr,      /* Pointer to the value of the password attribute. */
    0,                 /* The size of the password attribute value is already
                          known by the OCI library. */
    OCI_ATTR_PASSWORD, /* The attribute type. */
    error_handle);     /* An error handle used to retrieve diagnostic
                          information in the event of an error. */
  ```

Using Proxy Authentication with Enterprise Users

How the middle-tier responds for proxy authentication depends on how the user is authenticated, either as an enterprise user or a password-authenticated user.

If the middle tier connects to the database as a client who is an enterprise user, then either the distinguished name, or the X.509 certificate containing the distinguished name is passed over instead of the database user name. If the user is a password-authenticated enterprise user, then the middle tier must provide, as a minimum, a globally unique name for the user. The database uses this name to look up the user in Oracle Internet Directory.

- To configure proxy authentication with enterprise users, configure the application server and the middle tier to use the appropriate Oracle Call Interface settings:
  - To pass over the distinguished name of the client, configure the application server to call the Oracle Call Interface method `OCIAttrSet()` with `OCI_ATTR_DISTINGUISHED_NAME` as the attribute type, as follows:

    ```
    OCIAttrSet(session_handle,
               OCI_HTYPE_SESSION,
               distinguished_name,
               0,
               OCI_ATTR_DISTINGUISHED_NAME,
               error_handle);
    ```
  - To pass over the entire certificate, configure the middle tier to call `OCIAttrSet()` with `OCI_ATTR_CERTIFICATE` as the attribute type, as follows:

    ```
    OCIAttrSet(session_handle,
               OCI_HTYPE_SESSION,
               certificate,
               certificate_length,
               OCI_ATTR_CERTIFICATE,
               error_handle);
    ```

If the type is not specified, then the database uses its default certificate type of X.509.

Note:

- `OCI_ATTR_CERTIFICATE` is Distinguished Encoding Rules (DER) encoded.
- Certificate based proxy authentication using `OCI_ATTR_CERTIFICATE` will not be supported in future Oracle Database releases. Use the `OCI_ATTR_DISTINGUISHED_NAME` or `OCI_ATTR_USERNAME` attribute instead

If you are using proxy authentication for password-authenticated enterprise users, then use the same OCI attributes as for database users authenticated by password (`OCI_ATTR_USERNAME`). Oracle Database first checks the user name against the database. If it finds no user, then the database checks the user name in the directory. This user name must be globally unique.

Using Client Identifiers to Identify Application Users Unknown to the Database

Client identifiers preserve user identity in middle tier systems; they also can be used independently of the global application context.

About Client Identifiers

Oracle Database provides the `CLIENT_IDENTIFIER` attribute of the built-in `USERENV` application context namespace for application users.

These application users are known to an application but unknown to the database. The `CLIENT_IDENTIFIER` attribute can capture any value that the application uses for identification or access control, and passes it to the database. The `CLIENT_IDENTIFIER` attribute is supported in OCI, JDBC/OCI, or Thin driver.

How Client Identifiers Work in Middle Tier Systems

Many applications use session pooling to set up several sessions to be reused by multiple application users.

Users authenticate themselves to a middle-tier application, which uses a single identity to log in to the database and maintains all the user connections. In this model, application users are users who are authenticated to the middle tier of an application, but who are not known to the database. You can use a `CLIENT_IDENTIFIER` attribute, which acts like an application user proxy for these types of applications.

In this model, the middle tier passes a client identifier to the database upon the session establishment. The client identifier could actually be anything that represents a client connecting to the middle tier, for example, a cookie or an IP address. The client identifier, representing the application user, is available in user session information and can also be accessed with an application context (by using the `USERENV` naming context). In this way, applications can set up and reuse sessions, while still being able to keep track of the *application user* in the session. Applications can reset the client identifier and thus reuse the session for a different user, enabling high performance.

Use of the CLIENT_IDENTIFIER Attribute to Preserve User Identity

The `CLIENT_IDENTIFIER` predefined attribute of the built-in application context namespace, `USERENV`, captures the application user name for use with a global application context.

You also can use the `CLIENT_IDENTIFIER` attribute independently.

When you use the `CLIENT_IDENTIFIER` attribute independently from a global application context, you can set `CLIENT_IDENTIFIER` with the `DBMS_SESSION` interface. The ability to pass a `CLIENT_IDENTIFIER` to the database is supported in Oracle Call Interface (OCI), JDBC/OCI, or Thin driver.

When you use the `CLIENT_IDENTIFIER` attribute with global application context, it provides flexibility and high performance for building applications. For example, suppose a Web-based application that provides information to business partners has three types of users: gold partner, silver partner, and bronze partner, representing different levels of information available. Instead of each user having his or her own session set up with individual application contexts, the application could set up global application contexts for gold partners, silver partners, and bronze partners. Then, use the `CLIENT_IDENTIFIER` to point the session at the correct context to retrieve the appropriate type of data. The application need only initialize the three global contexts once and use the `CLIENT_IDENTIFIER` to access the correct application context to limit data access. This provides performance benefits through session reuse and through accessing global application contexts set up once, instead of having to initialize application contexts for each session individually.

Use of the CLIENT_IDENTIFIER Independent of Global Application Context

Using the `CLIENT_IDENTIFIER` attribute is especially useful for those applications in which the users are unknown to the database.

In these situations, the application typically connects as a single database user and all actions are taken as that user.

Because all user sessions are created as the same user, this security model makes it difficult to achieve data separation for each user. These applications can use the `CLIENT_IDENTIFIER` attribute to preserve the real application user identity through to the database.

With this approach, sessions can be reused by multiple users by changing the value of the `CLIENT_IDENTIFIER` attribute, which captures the name of the real application user. This avoids the overhead of setting up a separate session and separate attributes for each user, and enables reuse of sessions by the application. When the `CLIENT_IDENTIFIER` attribute value changes, the change is added to the next OCI, JDBC/OCI, or Thin driver call for additional performance benefits.

For example, the user Daniel connects to a Web Expense application. Daniel is not a database user; he is a typical Web Expense application user. The application accesses the built-in application context namespace and sets `DANIEL` as the `CLIENT_IDENTIFIER` attribute value. Daniel completes his Web Expense form and exits the application. Then, Ajit connects to the Web Expense application. Instead of setting up a new session for Ajit, the application reuses the session that currently exists for Daniel, by changing the `CLIENT_IDENTIFIER` to `AJIT`. This avoids the overhead of setting up a new connection to the database and the overhead of setting up a global application context. The `CLIENT_IDENTIFIER` attribute can be set to any value on which the application bases access control. It does not have to be the application user name.

Setting the CLIENT_IDENTIFIER Independent of Global Application Context

You can set the `CLIENT_IDENTIFIER` setting with Oracle Call Interface to be independent of the global application context.

- To set the `CLIENT_IDENTIFIER` attribute with OCI, use the `OCI_ATTR_CLIENT_IDENTIFIER` attribute in the call to `OCIAttrSet()`. Then, on the next request to the server, the information is propagated and stored in the server sessions.

For example:

```
OCIAttrSet (session,

OCI_HTYPE_SESSION,
(dvoid *) "appuser1",
(ub4)strlen("appuser1"),
OCI_ATTR_CLIENT_IDENTIFIER,
*error_handle);
```

For applications that use JDBC, be aware that JDBC does not set the client identifier. To set the client identifier in a connection pooling environment, use Dynamic Monitoring Service (DMS) metrics. If DMS is not available, then use the `connection.setClientInfo` method. For example:

```
connection.setClientInfo("E2E_CONTEXT.CLIENT_IDENTIFIER", "appuser");
```

See Also:

- *Oracle Call Interface Programmer's Guide* about how the `OCI_ATTR_CLIENT_IDENTIFIER` user session handle attribute is used in middle-tier applications
- *Oracle Database JDBC Developer’s Guide* for more information about configuring client connections using JDBC and DMS metrics
- *Oracle Database JDBC Developer’s Guide* for more information about the `setClientInfo` method

Use of the DBMS_SESSION PL/SQL Package to Set and Clear the Client Identifier

The `DBMS_SESSION` PL/SQL package manages client identifiers on both the middle tier and the database itself.

To use the `DBMS_SESSION` package to set and clear the `CLIENT_IDENTIFIER` value on the middle tier, you must use the `SET_IDENTIFIER` and `CLEAR_IDENTIFIER` procedures.

The middle tier uses `SET_IDENTIFIER` to associate the database session with a particular user or group. Then, the `CLIENT_IDENTIFIER` is an attribute of the session and can be viewed in session information.

If you plan to use the DBMS_SESSION.SET_IDENTIFIER procedure, then be aware of the following:

- The maximum number of bytes for the client_id parameter of DBMS_SESSION.SET_IDENTIFIER is 64 bytes. If it exceeds 64, then the additional bytes are truncated.
- The DBMS_APPLICATION_INFO.SET_CLIENT_INFO procedure can overwrite the value of the client identifier. Typically, these values should be the same, so if SET_CLIENT_INFO is set, then its value can be automatically propagated to the value set by SET_IDENTIFIER if the CLIENTID_OVERWRITE event is set to ON. You can check the status of the CLIENTID_OVERWRITE event by running the SHOW PARAMETER command for the EVENT parameter.

  For example, assuming that CLIENTID_OVERWRITE is enabled:

  ```
  SHOW PARAMETER EVENT

  NAME                              TYPE                VALUE
  ------------------------------    ------------------  ------------------
  event                             string              clientid_overwrite
  ```

Enabling the CLIENTID_OVERWRITE Event System-Wide

The ALTER SYSTEM statement can enable the CLIENTID_OVERWRITE event system-wide.

1. Enter the following ALTER SYSTEM statement:

   ```
   ALTER SYSTEM SET EVENTS 'CLIENTID_OVERWRITE';
   ```

   Or, enter the following line in your init.ora file:

   ```
   event="clientid_overwrite"
   ```

2. Restart the database.

   For example:

   ```
   SHUTDOWN IMMEDIATE
   STARTUP
   ```

See Also:

- Global Application Contexts for information about using client identifiers in a global application context
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the DBMS_SESSION package

Enabling the CLIENTID_OVERWRITE Event for the Current Session

The ALTER SESSION statement can enable the CLIENTID_OVERWRITE event for the current session only.

1. Use the ALTER SESSION statement to set the CLIENTID_OVERWRITE value for the session only.

   For example:

```
ALTER SESSION SET EVENTS 'CLIENTID_OVERWRITE OFF';
```

2. If you set the client identifier by using the `DBMS_APPLICATION_INFO.SET_CLIENT_INFO` procedure, then run `DBMS_SESSION.SET_IDENTIFIER` so that the client identifier settings are the same.

   For example:

   ```
   DBMS_SESSION.SET_IDENTIFIER(session_id_p);
   ```

Disabling the CLIENTID_OVERWRITE Event

The `ALTER SYSTEM` statement can disable the `CLIENTID_OVERWRITE` event.

1. Enter the following `ALTER SYSTEM` statement:

   ```
   ALTER SYSTEM SET EVENTS 'CLIENTID_OVERWRITE OFF';
   ```

2. Restart the database.

   For example:

   ```
   SHUTDOWN IMMEDIATE
   STARTUP
   ```

User Authentication Data Dictionary Views

Oracle Database provides data dictionary views that list information about user authentication, such as roles that users have or profiles they use.

Table 3-4 lists the data dictionary views. For detailed information about these views, see *Oracle Database Reference*.

Table 3-4 Data Dictionary Views That Describe User Authentication

View	Description
`DBA_PROFILES`	Displays information about profiles, including their settings and limits
`DBA_ROLES`	Displays the kind of authentication used for a database role to log in to the database, such as `NONE` or `GLOBAL` (query the `AUTHENTICATION_TYPE` column)
`DBA_USERS`	Among other user information, displays the following: • The kind of authentication the user used to log in to the database, such as `PASSWORD` or `EXTERNAL` (`AUTHENTICATION_TYPE` column) • The list of versions of password versions (also known as hashes) that exist for the user account (`PASSWORD_VERSIONS` column)
`DBA_USERS_WITH_DEFPWD`	Displays whether the user account password is a default password
`PROXY_USERS`	Displays users who are currently authorized to connect through a middle tier
`V$DBLINK`	Displays user accounts for existing database links (`DB_LINK`, `OWNER_ID` columns); applies to the current pluggable database (PDB)

Table 3-4 (Cont.) Data Dictionary Views That Describe User Authentication

View	Description
`V$PWFILE`	Lists the names and granted administrative privileges of the administrative users who are included in the password file
`V$SESSION`	Querying the `USERNAME` column displays concurrently logged in users to the current PDB

4 Configuring Privilege and Role Authorization

Privilege and role authorization controls the permissions that users have to perform day-to-day tasks.

About Privileges and Roles

Authorization permits only certain users to access, process, or alter data; it also creates limitations on user access or actions.

The limitations placed on (or removed from) users can apply to objects such as schemas, entire tables, or table rows.

A user **privilege** is the right to run a particular type of SQL statement, or the right to access an object that belongs to another user, run a PL/SQL package, and so on. The types of privileges are defined by Oracle Database.

Roles are created by users (usually administrators) to group together privileges or other roles. They are a way to facilitate the granting of multiple privileges or roles to users.

Privileges can fall into the following general categories:

- **System privileges.** These privileges allow the grantee to perform standard administrator tasks in the database. Restrict them only to trusted users. See the following sections describe privileges:
  - Managing Administrative Privileges
  - Managing System Privileges
  - Managing Commonly and Locally Granted Privileges
- **Roles.** A **role** groups several privileges and roles, so that they can be granted to and revoked from users simultaneously. You must enable the role for a user before the user can use it. See the following sections for more information:
  - Managing Common Roles and Local Roles
  - Managing User Roles
- **Object privileges.** Each type of object has privileges associated with it. Managing Object Privileges describes how to manage privileges for different types of objects.
- **Table privileges.** These privileges enable security at the DML (data manipulation language) or DDL (data definition language) level.Table Privileges describes how to manage table privileges.
- **View privileges.** You can apply DML object privileges to views, similar to tables. See View Privileges for more information.

- **Procedure privileges.** Procedures, including standalone procedures and functions, can be granted the `EXECUTE` privilege. See Procedure Privileges for more information.
- **Type privileges.** You can grant system privileges to named types (object types, `VARRAYS`, and nested tables). See Type Privileges for more information.

See Also:

Oracle Database Vault Administrator's Guide for information about how you can create policies that analyze privilege use

Who Should Be Granted Privileges?

You grant privileges to users so they can accomplish tasks required for their jobs.

You should grant a privilege only to a user who requires that privilege to accomplish the necessary work. Excessive granting of unnecessary privileges can compromise security. For example, you never should grant `SYSDBA` or `SYSOPER` administrative privilege to users who do not perform administrative tasks.

You can grant privileges to a user in two ways:

- **You can grant privileges to users explicitly.** For example, you can explicitly grant to user `psmith` the privilege to insert records into the `employees` table.
- **You can grant privileges to a role (a named group of privileges), and then grant the role to one or more users.** For example, you can grant the privileges to select, insert, update, and delete records from the `employees` table to the role named `clerk`, which in turn you can grant to users `psmith` and `robert`.

Because roles allow for easier and better management of privileges, you should usually grant privileges to roles and not to specific users.

See Also:

- Guidelines for Securing User Accounts and Privileges for best practices to follow when granting privileges
- *Oracle Database Vault Administrator's Guide* if you are concerned about excessive privilege grants
- *Oracle Database SQL Language Reference* for the complete list of system privileges and their descriptions

How the Oracle Multitenant Option Affects Privileges

In a multitenant environment, all users, including common users, can exercise their privileges only within the current container.

However, a user connected to the root can perform certain operations that affect other pluggable databases (PDBs). These operations include `ALTER PLUGGABLE DATABASE`, CREATE USER, CREATE ROLE, and ALTER USER. The common user must possess the commonly granted privileges that enable these operations. A common user connected to the root can see metadata pertaining to PDBs by way of the container data objects (for example, multitenant container database (CDB) views and V$ views) in the root, provided that the common user has been granted privileges required to access these views and his CONTAINER_DATA attribute has been set to allow seeing data about various PDBs. The common user cannot query tables or views in a PDB.

Common users cannot exercise their privileges across other PDBs. They must first switch to the PDB that they want, and then exercise their privileges from there. To switch to a different container, the common user must have the SET CONTAINER privilege. The SET CONTAINER privilege must be granted either commonly or in the container to which the user is attempting to switch. Alternatively, the common user can start a new database session whose initial current container is the container this user wants, relying on the CREATE SESSION privilege in that PDB.

Be aware that commonly granted privileges may interfere with the security configured for individual PDBs. For example, suppose an application PDB database administrator wants to prevent any user in the PDB from modifying a particular application common object. A privilege (such as UPDATE) granted commonly to PUBLIC or to a common user or common role on the object would circumvent the PDB database administrator's intent.

Managing Administrative Privileges

Administrative privileges can be used for both general and specific database operations.

About Administrative Privileges

For better separation of duty, Oracle Database provides administrative privileges that are tailored for commonly performed specific administrative tasks.

These tasks include operations for backup and recovery, Oracle Data Guard, and encryption key management for Transparent Data Encryption (TDE).

You can find the administrative privileges that a user has by querying the V$PWFILE_USERS dynamic view, which lists users in the password file.

In previous releases, you needed to have the SYSDBA administrative privilege to perform these tasks. To support backward compatibility, you still can use the SYSDBA privilege for these tasks, but Oracle recommends that you use the administrative privileges described in this section.

The use of administrative privileges is mandatorily audited.

Grants of Administrative Privileges to Users

As with all powerful privileges, only grant administrative privileges to trusted users.

However, be aware that there is a restriction for users whose names have non-ASCII characters (for example, the umlaut in the name HÜBER). You can grant administrative privileges to these users, but if the Oracle database instance is down, the authentication using the granted privilege is not supported if the user name has non-ASCII characters. If the database instance is up, then the authentication is supported.

SYSDBA and SYSOPER Privileges for Standard Database Operations

The `SYSDBA` and `SYSOPER` administrative privileges enable you to perform standard database operations.

These database operations can include tasks such as database startups and shutdowns, creating the server parameter file (`SPFILE`), or altering the database archive log. In a multitenant environment, you can grant the `SYSDBA` and `SYSOPER` administrative privileges to application common users (but not to CDB common users).

You can find if a user has been granted an administrative privilege on a local (PDB) level, for a CDB root, or for an application root by querying the `SCOPE` column of the `V$PWFILE_USERS` dynamic view.

You cannot grant the `SYSDBA` or `SYSOPER` administrative privilege to users who have been created with no authentication.

See Also:

Oracle Database Administrator's Guide for detailed information about the `SYSDBA` and `SYSOPER` administrative privileges

SYSBACKUP Administrative Privilege for Backup and Recovery Operations

The `SYSBACKUP` administrative privilege is used to perform backup and recovery operations from either Oracle Recovery Manager (RMAN) and or through SQL*Plus.

To connect to the database as `SYSBACKUP` using a password, you must create a password file for it. See *Oracle Database Administrator's Guide* for more information about creating password files.

You cannot grant the `SYSBACKUP` administrative privilege to users who have been created with no authentication.

This privilege enables you to perform the following operations:

- `STARTUP`
- `SHUTDOWN`
- `ALTER DATABASE`
- `ALTER SYSTEM`
- `ALTER SESSION`
- `ALTER TABLESPACE`
- `CREATE CONTROLFILE`
- `CREATE ANY DIRECTORY`
- `CREATE ANY TABLE`
- `CREATE ANY CLUSTER`

- `CREATE PFILE`
- `CREATE RESTORE POINT` (including `GUARANTEED` restore points)
- `CREATE SESSION`
- `CREATE SPFILE`
- `DROP DATABASE`
- `DROP TABLESPACE`
- `DROP RESTORE POINT` (including `GUARANTEED` restore points)
- `FLASHBACK DATABASE`
- `RESUMABLE`
- `UNLIMITED TABLESPACE`
- `SELECT ANY DICTIONARY`
- `SELECT ANY TRANSACTION`
- `SELECT`
  - `X$` tables (that is, the fixed tables)
  - `V$` and `GV$` views (that is, the dynamic performance views)
  - `APPQOSSYS.WLM_CLASSIFIER_PLAN`
  - `SYSTEM.LOGSTDBY$PARAMETERS`
- `DELETE/INSERT`
  - `SYS.APPLY$_SOURCE_SCHEMA`
  - `SYSTEM.LOGSTDBY$PARAMETERS`
- `EXECUTE`
  - `SYS.DBMS_BACKUP_RESTORE`
  - `SYS.DBMS_RCVMAN`
  - `SYS.DBMS_DATAPUMP`
  - `SYS.DBMS_IR`
  - `SYS.DBMS_PIPE`
  - `SYS.SYS_ERROR`
  - `SYS.DBMS_TTS`
  - `SYS.DBMS_TDB`
  - `SYS.DBMS_PLUGTS`
  - `SYS.DBMS_PLUGTSP`
- `SELECT_CATALOG_ROLE`

In addition, the `SYSBACKUP` privilege enables you to connect to the database even if the database is not open.

See Also:

Oracle Database Backup and Recovery User's Guide for more information about backup and recovery operations

SYSDG Administrative Privilege for Oracle Data Guard Operations

You can log in as user `SYSDG` with the `SYSDG` administrative privilege to perform Data Guard operations.

You can use this privilege with either Data Guard Broker or the `DGMGRL` command-line interface. In order to connect to the database as `SYSDG` using a password, you must create a password file for it.

You cannot grant the `SYSYSDG` administrative privilege to users who have been created with no authentication.

The `SYSDG` privilege enables the following operations:

- `STARTUP`
- `SHUTDOWN`
- `ALTER DATABASE`
- `ALTER SESSION`
- `ALTER SYSTEM`
- `CREATE RESTORE POINT` (including `GUARANTEED` restore points)
- `CREATE SESSION`
- `DROP RESTORE POINT` (including `GUARANTEED` restore points)
- `FLASHBACK DATABASE`
- `SELECT ANY DICTIONARY`
- `SELECT`
  - `X$` tables (that is, the fixed tables)
  - `V$` and `GV$` views (that is, the dynamic performance views)
  - `APPQOSSYS.WLM_CLASSIFIER_PLAN`
- `DELETE`
  - `APPQOSSYS.WLM_CLASSIFIER_PLAN`
- `EXECUTE`
  - `SYS.DBMS_DRS`

In addition, the `SYSDG` privilege enables you to connect to the database even if it is not open.

See Also:

- *Oracle Database Administrator’s Guide* for more information about creating password files
- *Oracle Data Guard Concepts and Administration* for more information about Oracle Data Guard

SYSKM Administrative Privilege for Transparent Data Encryption

The `SYSKM` administrative privilege enables the `SYSKM` user to manage Transparent Data Encryption (TDE) wallet operations.

In order to connect to the database as `SYSKM` using a password, you must create a password file for it.

You cannot grant the `SYSKM` administrative privilege to users who have been created with no authentication.

The `SYSKM` administrative privilege enables the following operations:

- `ADMINISTER KEY MANAGEMENT`
- `CREATE SESSION`
- `SELECT` (only when database is open)
  - `SYS.V$ENCRYPTED_TABLESPACES`
  - `SYS.V$ENCRYPTION_WALLET`
  - `SYS.V$WALLET`
  - `SYS.V$ENCRYPTION_KEYS`
  - `SYS.V$CLIENT_SECRETS`
  - `SYS.DBA_ENCRYPTION_KEY_USAGE`

In addition, the `SYSKM` privilege enables you to connect to the database even if it is not open.

See Also:

- *Oracle Database Administrator’s Guide* for more information about creating password files
- *Oracle Database Advanced Security Guide* for more information about Transparent Data Encryption

SYSRAC Administrative Privilege for Oracle Real Application Clusters

The `SYSRAC` administrative privilege is used by the Oracle Real Application Clusters (Oracle RAC) Clusterware agent.

The SYSRAC administrative privilege provides only the minimal privileges necessary for performing day-to-day Oracle RAC operations. For example, this privilege is used for Oracle RAC utilities such as SRVCTL.

You cannot grant the SYSRAC administrative privilege to users who have been created with no authentication.

The SYSRAC administrative privilege enables the following operations:

- STARTUP
- SHUTDOWN
- ALTER DATABASE MOUNT
- ALTER DATABASE OPEN
- ALTER DATABASE OPEN READ ONLY
- ALTER DATABASE CLOSE NORMAL
- ALTER DATABASE DISMOUNT
- ALTER SESSION SET EVENTS
- ALTER SESSION SET _NOTIFY_CRS
- ALTER SESSION SET CONTAINER
- ALTER SYSTEM REGISTER
- ALTER SYSTEM SET *local_listener*|*remote_listener*|*listener_networks*

In addition to these privileges, the SYSRAC user will have access to the following views:

- V$PARAMETER
- V$DATABASE
- V$PDBS
- CDB_SERVICE$
- DBA_SERVICES
- V$ACTIVE_SERVICES
- V$SERVICES

The SYSRAC user is also granted the EXECUTE privilege for the following PL/SQL packages:

- DBMS_DRS
- DBMS_SERVICE
- DBMS_SERVICE_PRVT
- DBMS_SESSION
- DBMS_HA_ALERTS_PRVT
- Dequeue messaging SYS.SYS$SERVICE_METRICS

See Also:

Oracle Real Application Clusters Administration and Deployment Guide for more information about Oracle RAC

Managing System Privileges

To perform actions on schema objects, you must be granted the appropriate system privileges.

About System Privileges

A system privilege is the right to perform an action or to perform actions on schema objects.

For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges.

There are over 100 distinct system privileges. Each system privilege allows a user to perform a particular database operation or class of database operations. *Remember that system privileges are very powerful.* Only grant them when necessary to roles and trusted users of the database. To find the system privileges that have been granted to a user, you can query the `DBA_SYS_PRIVS` data dictionary view.

See Also:

- *Oracle Database SQL Language Reference* for a complete list of system privileges and their descriptions
- How Commonly Granted System Privileges Work

Why Is It Important to Restrict System Privileges?

System privileges are very powerful, so only grant them to trusted users. You should also secure the data dictionary and `SYS` schema objects.

About the Importance of Restricting System Privileges

System privileges are very powerful, so by default the database is configured to prevent typical (non-administrative) users from exercising the `ANY` system privileges.

For example, users are prevented from exercising `ANY` system privileges such as `UPDATE ANY TABLE` on the data dictionary.

Restricting System Privileges by Securing the Data Dictionary

The `O7_DICTIONARY_ACCESSIBILITY` initialization parameter controls restrictions on system privileges when you upgrade from Oracle Database release 7 to Oracle8*i* and later releases.

If the parameter is set to `TRUE`, then access to objects in the `SYS` schema is allowed (Oracle Database release 7 behavior). Because the `ANY` privilege applies to the data dictionary, a malicious user with `ANY` privilege could access or alter data dictionary tables.

- To secure the data dictionary, set the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter to `FALSE`, which is the default value. This feature is called the dictionary protection mechanism.

  To set the `O7_DICTIONARY_ACCESSIBILTY` initialization parameter, you can modify it in the `init`*`SID`*`.ora` file. Alternatively, you can log on to SQL*Plus as user `SYS` with the `SYSDBA` administrative privilege and then enter an `ALTER SYSTEM` statement, assuming you have started the database using a server parameter file (SPFILE).

Example 4-1 shows how to set the `O7_DICTIONARY_ACCESSIBILTY` initialization parameter to `FALSE` by issuing an `ALTER SYSTEM` statement in SQL*Plus.

Example 4-1 Setting O7_DICTIONARY_ACCESSIBILITY to FALSE

```
ALTER SYSTEM SET O7_DICTIONARY_ACCESSIBILITY=FALSE SCOPE=SPFILE;
```

When you set `O7_DICTIONARY_ACCESSIBILITY` to `FALSE`, system privileges that enable access to objects in any schema (for example, users who have `ANY` privileges, such as `CREATE ANY PROCEDURE`) do not allow access to objects in the `SYS` schema. This means that access to the objects in the `SYS` schema (data dictionary objects) is restricted to users who connect using the `SYSDBA` administrative privilege. Remember that the `SYS` user must log in with either the `SYSDBA` or `SYSOPER` privilege; otherwise, an `ORA-28009: connection as SYS should be as SYSDBA or SYSOPER` error is raised. If you set `O7_DICTIONARY_ACCESSIBILITY` to `TRUE`, then you would be able to log in to the database as user `SYS` without having to specify the `SYSDBA` or `SYSOPER` privilege.

System privileges that provide access to objects in other schemas do *not* give other users access to objects in the `SYS` schema. For example, the `SELECT ANY TABLE` privilege allows users to access views and tables in other schemas, but does not enable them to select dictionary objects (base tables of dynamic performance views, regular views, packages, and synonyms). You can, however, grant these users explicit object privileges to access objects in the `SYS` schema.

See Also:

Oracle Database Reference for more information about the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter

User Access to Objects in the SYS Schema

Users with explicit object privileges or those who connect with administrative privileges (`SYSDBA`) can access objects in the `SYS` schema.

Table 4-1 lists roles that you can grant to users who need access to objects in the SYS schema.

Table 4-1 Roles to Allow Access to SYS Schema Objects

Role	Description
SELECT_CATALOG_ROLE	Grant this role to allow users SELECT privileges on data dictionary views.
EXECUTE_CATALOG_ROLE	Grant this role to allow users EXECUTE privileges for packages and procedures in the data dictionary.

Additionally, you can grant the SELECT ANY DICTIONARY system privilege to users who require access to tables created in the SYS schema. This system privilege allows query access to any object in the SYS schema, including tables created in that schema. It must be granted individually to each user requiring the privilege. It is not included in GRANT ALL PRIVILEGES, but it can be granted through a role.

Note:

You should grant these roles and the SELECT ANY DICTIONARY system privilege with extreme care, because the integrity of your system can be compromised by their misuse.

Grants and Revokes of System Privileges

You can grant or revoke system privileges to users and roles.

If you grant system privileges to roles, then you can use the roles to exercise system privileges. For example, roles permit privileges to be made selectively available. Ensure that you follow the separation of duty guidelines described in Guidelines for Securing Roles.

Use either of the following methods to grant or revoke system privileges to or from users and roles:

- GRANT and REVOKE SQL statements
- Oracle Enterprise Manager Cloud Control

Who Can Grant or Revoke System Privileges?

Only two types of users can grant system privileges to other users or revoke those privileges from them.

These users are as follows:

- Users who were granted a specific system privilege with the ADMIN OPTION
- Users with the system privilege GRANT ANY PRIVILEGE

For this reason, only grant these privileges to trusted users.

About ANY Privileges and the PUBLIC Role

System privileges that use the ANY keyword enable you to set privileges for an entire category of objects in the database.

For example, the CREATE ANY PROCEDURE system privilege permits a user to create a procedure anywhere in the database. The behavior of an object created by users with the ANY privilege is not restricted to the schema in which it was created. For example, if user JSMITH has the CREATE ANY PROCEDURE privilege and creates a procedure in the schema JONES, then the procedure will run as JONES. However, JONES may not be aware that the procedure JSMITH created is running as him (JONES). If JONES has DBA privileges, letting JSMITH run a procedure as JONES could pose a security violation.

The PUBLIC role is a special role that every database user account automatically has when the account is created. By default, it has no privileges granted to it, but it does have numerous grants, mostly to Java objects. You cannot drop the PUBLIC role, and a manual grant or revoke of this role has no meaning, because the user account will always assume this role. Because all database user accounts assume the PUBLIC role, it does not appear in the DBA_ROLES and SESSION_ROLES data dictionary views.

You can grant privileges to the PUBLIC role, but remember that this makes the privileges available to every user in the Oracle database. For this reason, be careful about granting privileges to the PUBLIC role, particularly powerful privileges such as the ANY privileges and system privileges. For example, if JSMITH has the CREATE PUBLIC SYNONYM system privilege, he could redefine an interface that he knows everyone else uses, and then point to it with the PUBLIC SYNONYM that he created. Instead of accessing the correct interface, users would access the interface of JSMITH, which could possibly perform illegal activities such as stealing the login credentials of users.

These types of privileges are very powerful and could pose a security risk if given to the wrong person. Be careful about granting privileges using ANY or PUBLIC. As with all privileges, you should follow the principles of "least privilege" when granting these privileges to users.

To protect the data dictionary (the contents of the SYS schema) against users who have one or more of the powerful ANY system privileges, set the O7_DICTIONARY_ACCESSIBILITY initialization parameter to FALSE. You can set this parameter by using an ALTER SYSTEM statement or by modifying the init*SID*.ora file.

Managing Commonly and Locally Granted Privileges

In a multitenant environment, privileges can be granted commonly for an entire CDB or application container, or granted locally to a specific PDB.

About Commonly and Locally Granted Privileges

In a multitenant environment, both common users and local users can grant privileges to one another.

Privileges by themselves are neither common nor local. How the privileges are applied depends on whether the privilege is granted commonly or granted locally.

For commonly granted privileges:

- A privilege that is granted commonly can be used in every existing and future container.
- Only common users can grant privileges commonly, and only if the grantee is common.
- A common user can grant privileges to another common user or to a common role.
- The grantor must be connected to the root and must specify `CONTAINER=ALL` in the `GRANT` statement.
- Both system and object privileges can be commonly granted. (Object privileges become actual only with regard to the specified object.)
- When a common user connects to or switches to a given container, this user's ability to perform various activities (such as creating a table) is controlled by privileges granted commonly as well as privileges granted locally in the given container.
- Do not grant privileges to `PUBLIC` commonly.

For locally granted privileges:

- A privilege granted locally can be used only in the container in which it was granted. When the privilege is granted in the root, it applies only to the root.
- Both common users and local users can grant privileges locally.
- A common user and a local user can grant privileges to other common or local roles.
- The grantor must be connected to the container and must specify `CONTAINER=CURRENT` in the `GRANT` statement.
- Any user can grant a privilege locally to any other user or role (both common and local) or to the `PUBLIC` role.

How Commonly Granted System Privileges Work

Users can exercise system privileges only within the PDB in which they were granted.

For example, if a system privilege is locally granted to a common user `A` in a PDB `B`, user `A` can exercise that privilege only while connected to PDB `B`.

System privileges can apply in the root and in all existing and future PDBs if the following requirements are met:

- The system privilege grantor is a common user and the grantee is a common user, a common role, or the `PUBLIC` role. Do not commonly grant system privileges to the `PUBLIC` role, because this in effect makes the system privilege available to all users.
- The system privilege grantor possesses the `ADMIN OPTION` for the commonly granted privilege
- The `GRANT` statement must contain the `CONTAINER=ALL` clause.

The following example shows how to commonly grant a privilege to the common user `c##hr_admin`.

```
CONNECT SYSTEM
Enter password: password
Connected.
```

```
GRANT CREATE ANY TABLE TO c##hr_admin CONTAINER=ALL;
```

How Commonly Granted Object Privileges Work

Object privileges on common objects applies to the object as well as all associated links on this common object.

These links include all metadata links, data links (previously called object links), or extended data links that are associated with it in the root and in all PDBs belonging to the container (including future PDBs) if certain requirements are met.

These requirements are as follows:

- The object privilege grantor is a common user and the grantee is a common user, a common role, or the `PUBLIC` role.
- The object privilege grantor possesses the commonly granted `GRANT OPTION` for the privilege
- The `GRANT` statement contains the `CONTAINER=ALL` clause.

The following example shows how to grant an object privilege to the common user `c##hr_admin` so that he can select from the `DBA_PDBS` view in the CDB root or in any of the associated PDBs that he can access.

```
CONNECT SYSTEM
Enter password: password
Connected.

GRANT SELECT ON DBA_OBJECTS TO c##hr_admin
CONTAINER=ALL;
```

Granting or Revoking Privileges to Access a PDB

You can grant and revoke privileges for PDB access in a multitenant environment.

To grant a privilege in a multitenant environment:

- Include the `CONTAINER` clause in the `GRANT` or `REVOKE` statement.

Setting `CONTAINER` to `ALL` applies the privilege to all existing and future containers; setting it to `CURRENT` applies the privilege to the local container only. Omitting the `CONTAINER` clause applies the privilege to the local container. If you issue the `GRANT` statement from the root and omit the `CONTAINER` clause, then the privilege is applied locally.

Example: Granting a Privilege in a Multitenant Environment

You can use the `GRANT` statement to grant privileges in a multitenant environment.

Example 4-2 shows how to commonly grant the `CREATE TABLE` privilege to common user `c##hr_admin` so that this user can use this privilege in all existing and future containers.

Example 4-2 Granting a Privilege in a Multitenant Environment

```
CONNECT SYSTEM
Enter password: password
Connected.
```

```
GRANT CREATE TABLE TO c##hr_admin CONTAINER=ALL;
```

Enabling Common Users to View CONTAINER_DATA Object Information

Common users can view information about `CONTAINER_DATA` objects in the root or for data in specific PDBs.

Viewing Data About the Root, CDB, and PDBs While Connected to the Root

You can restrict view information for the `X$` table and the `V$`, `GV$` and `CDB_*` views when common users perform queries.

The `X$` table and these views contain information about the application root and its associated application PDBs or, if you are connected to the CDB root, the entire CDB.

Restricting this information is useful when you do not want to expose sensitive information about other PDBs. To enable this functionality, Oracle Database provides these tables and views as container data objects. You can find if a specific table or view is a container data object by querying the `TABLE_NAME`, `VIEW_NAME`, and `CONTAINER_DATA` columns of the `USER_|DBA_|ALL_VIEWS|TABLES` dictionary views.

To find information about the default (user-level) and object-specific `CONTAINER_DATA` attributes:

1. In SQL*Plus or SQL Developer, log in to the root.
2. Query the `CDB_CONTAINER_DATA` data dictionary view.

   For example:

```
COLUMN USERNAME FORMAT A15
COLUMN DEFAULT_ATTR FORMAT A7
COLUMN OWNER FORMAT A15
COLUMN OBJECT_NAME FORMAT A15
COLUMN ALL_CONTAINERS FORMAT A3
COLUMN CONTAINER_NAME FORMAT A10
COLUMN CON_ID FORMAT A6

SELECT USERNAME, DEFAULT_ATTR, OWNER, OBJECT_NAME,
       ALL_CONTAINERS, CONTAINER_NAME, CON_ID
FROM   CDB_CONTAINER_DATA
ORDER BY OBJECT_NAME;

USERNAME        DEFAULT OWNER           OBJECT_NAME     ALL CONTAINERS CON_ID
--------------- ------- --------------- --------------- --- ---------- ------
C##HR_ADMIN     N       SYS             V$SESSION       N   CDB$ROOT        1
C##HR_ADMIN     N       SYS             V$SESSION       N   SALESPDB        1
C##HR_ADMIN     Y                                       N   HRPDB           1
C##HR_ADMIN     Y                                       N   CDB$ROOT        1
DBSNMP          Y                                       Y                   1
SYSTEM          Y                                       Y                   1
```

Enabling Common Users to Query Data in Specific PDBs

You can enable common users to access data pertaining to specific PDBs by adjusting the users' `CONTAINER_DATA` attribute.

To enable common users to access data about specific PDBs:

- Issue the `ALTER USER` statement in the root.

Example 4-3 Setting the CONTAINER_DATA Attribute

This example shows how to issue the `ALTER USER` statement to enable the common user `c##hr_admin` to view information pertaining to the `CDB$ROOT`, `SALES_PDB`, and `HRPDB` containers in the `V$SESSION` view (assuming this user can query that view).

```
CONNECT SYSTEM
Enter password: password
Connected.

ALTER USER c##hr_admin
SET CONTAINER_DATA = (CDB$ROOT, SALESPDB, HRPDB)
FOR V$SESSION CONTAINER=CURRENT;
```

In this specification:

- `SET CONTAINER_DATA` lists containers, data pertaining to which can be accessed by the user.
- `FOR V$SESSION` specifies the `CONTAINER_DATA` dynamic view, which common user `c##hr_admin` will query.
- `CONTAINER = CURRENT` must be specified because when you are connected to the root, `CONTAINER=ALL` is the default for the `ALTER USER` statement, but modification of the `CONTAINER_DATA` attribute must be restricted to the root.

If you want to enable user `c##hr_admin` to view information that pertains to the `CDB$ROOT`, `SALES_PDB`, `HRPDB` containers in all `CONTAINER_DATA` objects that this user can access, then omit `FOR V$SESSION`. For example:

```
ALTER USER c##hr_admin
SET CONTAINER_DATA = (CDB$ROOT, SALESPDB, HRPDB)
CONTAINER=CURRENT;
```

Managing Common Roles and Local Roles

A common role is a role that is created in the root; a local role is created in a PDB.

About Common Roles and Local Roles

In a multitenant environment, database roles can be specific to a PDB or used throughout the entire system container or application container.

A common role is a role whose identity and (optional) password are created in the root of a container and will be known in the root and in all existing and future PDBs belonging to that container.

A local role exists in only one PDB and can only be used within this PDB. It does not have any commonly granted privileges.

Note the following:

- Common users can both create and grant common roles to other common and local users.
- You can grant a role (local or common) to a local user or role only locally.
- If you grant a common role locally, then the privileges of that common role apply only in the container where the role is granted.
- Local users cannot create common roles, but they can grant them to common and other local users.
- The `CONTAINER = ALL` clause is the default when you create a common role in the CDB root or an application root.

How Common Roles Work

Common roles are visible in the root and in every PDB of a container within which they are defined in a multitenant environment.

A privilege can be granted commonly to a common role if:

- The grantor is a common user.
- The grantor possesses the commonly granted `ADMIN OPTION` for the privilege that is being granted.
- The `GRANT` statement contains the `CONTAINER=ALL` clause.

If the common role contains locally granted privileges, then these privileges apply only within the PDB in which they were granted to the common role. A local role cannot be granted commonly.

For example, suppose the CDB common user `c##hr_mgr` has been commonly granted the `DBA` role. This means that user `c##hr_mgr` can use the privileges associated with the `DBA` role in the root and in every PDB in the multitenant environment. However, if the CDB common user `c##hr_mgr` has only been locally granted the `DBA` role for the `hr_pdb` PDB, then this user can only use the `DBA` role's privileges in the `hr_pdb` PDB.

How the PUBLIC Role Works in a Multitenant Environment

All privileges that Oracle grants to the `PUBLIC` role are granted locally.

This feature enables you to revoke privileges or roles that have been granted to the `PUBLIC` role individually in each PDB as needed. If you must grant any privileges to the `PUBLIC` role, then grant them locally. Never grant privileges to `PUBLIC` commonly.

Privileges Required to Create, Modify, or Drop a Common Role

Only common users who have the commonly granted `CREATE ROLE`, `ALTER ROLE`, and `DROP ROLE` privileges can create, alter, or drop common roles.

Common users can also create local roles, but these roles are available only in the PDB in which they were created.

Rules for Creating Common Roles

When you create a common role, you must follow special rules.

The rules are as follows:

- **Ensure that you are in the correct root.** For the creation of common roles, you must be in the correct root, either the CDB root or the application root. You cannot create common roles from a PDB. To check if you are in the correct root, run one of the following:
  - To confirm that you are in the CDB root, you can issue the `show_con_name` command. The output should be `CDB$ROOT`.
  - To confirm that you are in an application root, verify that the following query returns `YES`:

    ```
    SELECT APPLICATION_ROOT FROM V$PDBS WHERE CON_ID=SYS_CONTEXT('USERENV',
    'CON_ID');
    ```
  - **Ensure that the name that you give the common role starts with the value of the COMMON_USER_PREFIX parameter (which defaults to C##).** Note that this requirement does not apply to the names of existing Oracle-supplied roles, such as `DBA` or `RESOURCE`.
- **Optionally, set the CONTAINER clause to ALL.** As long as you are in the root, if you omit the `CONTAINER = ALL` clause, then by default the role is created as a common role for the CDB root or the application root.

Creating a Common Role

You can use the `CREATE USER` statement to create a common role.

1. Connect to the root of the CDB or the application container in which you want to create the common role.

   For example:

   ```
   CONNECT SYSTEM
   Enter password: password
   Connected.
   ```
2. Run the `CREATE ROLE` statement with the `CONTAINER` clause set to `ALL`.

   For example:

   ```
   CREATE ROLE c##sec_admin IDENTIFIED BY password CONTAINER=ALL;
   ```

Rules for Creating Local Roles

To create a local role, you must follow special rules.

These rules are as follows:

- You must be connected to the PDB in which you want to create the role, and have the `CREATE ROLE` privilege.
- The name that you give the local role must not start with the value of the `COMMON_USER_PREFIX` parameter (which defaults to `C##`).
- You can include `CONTAINER=CURRENT` in the `CREATE ROLE` statement to specify the role as a local role. If you are connected to a PDB and omit this clause, then the `CONTAINER=CURRENT` clause is implied.

- You cannot have common roles and local roles with the same name. However, you can use the same name for local roles in different PDBs. To find the names of existing roles, query the `CDB_ROLES` and `DBA_ROLES` data dictionary views.

Creating a Local Role

You can use the `CREATE ROLE` statement to create a role.

1. Connect to the PDB in which you want to create the local role.

   For example:

   ```
   CONNECT SYSTEM@hrpdb
   Enter password: password
   Connected.
   ```

2. Run the `CREATE ROLE` statement with the `CONTAINER` clause set to `CURRENT`.

   For example:

   ```
   CREATE ROLE sec_admin CONTAINER=CURRENT;
   ```

Role Grants and Revokes for Common Users and Local Users

Role grants and revokes apply only to the scope of access of the common user or the local user.

Common users can grant and revoke common roles to and from other common users. A local user can grant a common role to any user in a PDB, including common users, but this grant applies only within the PDB.

The following example shows how to grant the common user `c##sec_admin` the `AUDIT_ADMIN` common role for use in all containers.

```
CONNECT SYSTEM
Enter password: password
Connected.

GRANT AUDIT_ADMIN TO c##sec_admin CONTAINER=ALL;
```

Similarly, the next example shows how local user `aud_admin` can grant the common user `c##sec_admin` the `AUDIT_ADMIN` common role for use within the `hrpdb` PDB.

```
CONNECT aud_admin@hrpdb
Enter password: password
Connected.

GRANT AUDIT_ADMIN TO c##sec_admin CONTAINER=CURRENT;
```

This example shows how a local user `aud_admin` can revoke a role from another user in a PDB. If you omit the `CONTAINER` clause, then `CURRENT` is implied.

```
CONNECT aud_admin@hrpdb
Enter password: password
Connected.

REVOKE sec_admin FROM psmith CONTAINER=CURRENT;
```

Managing User Roles

A user role is a named collection of privileges that you can create and assign to other users.

About User Roles

User roles are useful in a variety of situations, such as restricting DDL usage.

What Are User Roles?

A user **role** is a named group of related privileges that you can grant as a group to users or other roles.

Managing and controlling privileges is easier when you use **roles**.

Within a database, each role name must be unique, different from all user names and all other role names. Unlike schema objects, roles are not contained in any schema. Therefore, a user who creates a role can be dropped with no effect on the role.

The Functionality of Roles

Roles are useful for quickly and easily granting permissions to users.

Although you can use Oracle Database-defined roles, you have more control and continuity if you create your own roles that contain only the privileges pertaining to your requirements. Oracle may change or remove the privileges in an Oracle Database-defined role.

Roles have the following functionality:

- A role can be granted system or object privileges.
- Any role can be granted to any database user.
- Each role granted to a user is, at a given time, either enabled or disabled. A user's security domain includes the privileges of all roles currently enabled for the user and excludes the privileges of any roles currently disabled for the user. Oracle Database allows database applications and users to enable and disable roles to provide selective availability of privileges.
- A role can be granted to other roles. However, a role cannot be granted to itself and cannot be granted circularly. For example, role `role1` cannot be granted to role `role2` if role `role2` has previously been granted to role `role1`.
- If a role is not password authenticated or a secure application role, then you can grant the role indirectly to the user. An indirectly granted role is a role granted to the user through another role that has already been granted to this user. For example, suppose you grant user `psmith` the `role1` role. Then you grant the `role2` and `role3` roles to the `role1` role. Roles `role2` and `role3` are now under `role1`. This means `psmith` has been indirectly granted the roles `role2` and `role3`, in addition to the direct grant of `role1`. Enabling the direct `role1` for `psmith` enables the indirect roles `role2` and `role3` for this user as well.
- Optionally, you can make a directly granted role a default role. You enable or disable the default role status of a directly granted role by using the `DEFAULT ROLE`

clause of the ALTER USER statement. Ensure that the DEFAULT ROLE clause refers only to roles that have been directly granted to the user. To find the directly granted roles for a user, query the DBA_ROLE_PRIVS data dictionary view. This view does not include the user's indirectly granted roles. To find roles that are granted to other roles, query the ROLE_ROLE_PRIVS view.

- If the role is password authenticated or a secure application role, then you cannot grant it indirectly to the user, nor can you make it a default role. You only can grant this type of role directly to the user. Typically, you enable password authenticated or secure application roles by using the SET ROLE statement.

Properties of Roles and Why They Are Advantageous

Roles have special properties that make their management very easy, such reduced privilege administration.

Table 4-2 describes the properties of roles that enable easier privilege management within a database.

Table 4-2 Properties of Roles and Their Description

Property	Description
Reduced privilege administration	Rather than granting the same set of privileges explicitly to several users, you can grant the privileges for a group of related users to a role, and then only the role must be granted to each member of the group.
Dynamic privilege management	If the privileges of a group must change, then only the privileges of the role need to be modified. The security domains of all users granted the group's role automatically reflect the changes made to the role.
Selective availability of privileges	You can selectively enable or disable the roles granted to a user. This allows specific control of a user's privileges in any given situation.
Application awareness	The data dictionary records which roles exist, so you can design applications to query the dictionary and automatically enable (or disable) selective roles when a user attempts to execute the application by way of a given user name.
Application-specific security	You can protect role use with a password. Applications can be created specifically to enable a role when supplied the correct password. Users cannot enable the role if they do not know the password.

Database administrators often create roles for a database application. You should grant a secure application role all privileges necessary to run the application. You then can grant the secure application role to other roles or users. An application can have several different roles, each granted a different set of privileges that allow for more or less data access while using the application.

The DBA can create a role with a password to prevent unauthorized use of the privileges granted to the role. Typically, an application is designed so that when it starts, it enables the proper role. As a result, an application user does not need to know the password for an application role.

Typical Uses of Roles

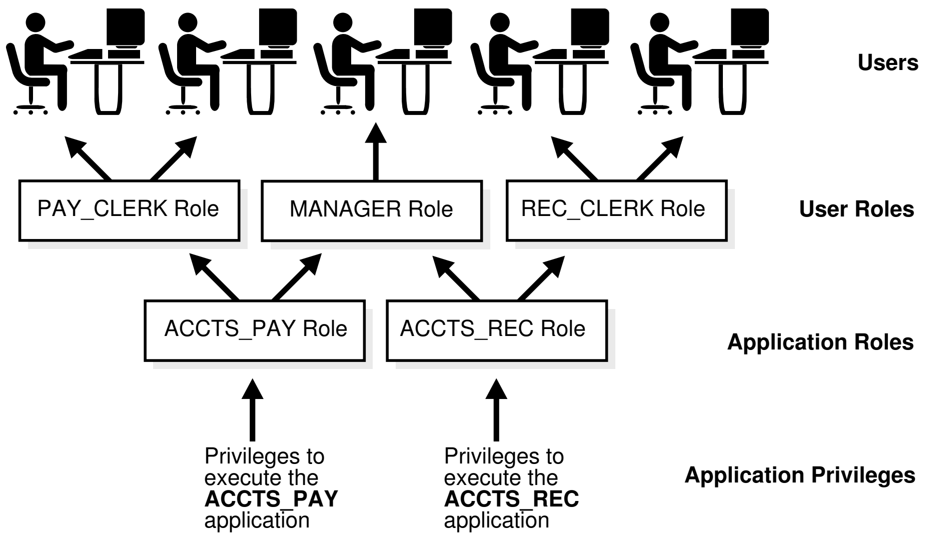
In general, you create a role to manage privileges.

Reasons are as follows:

- To manage the privileges for a database application
- To manage the privileges for a user group

Figure 4-1 describes the two uses of roles.

Figure 4-1 Common Uses for Roles



Common Uses of Application Roles

You can use application roles to control privileges to use applications.

You should grant an application role all privileges necessary to run a given database application. Then, grant the secure application role to other roles or to specific users.

An application can have several different roles, with each role assigned a different set of privileges that allow for more or less data access while using the application.

Common Uses of User Roles

You can create a user role for a group of database users with common privilege grant requirements.

You can manage user privileges by granting secure application roles and privileges to the user role and then granting the user role to appropriate users.

How Roles Affect the Scope of a User's Privileges

Each role and user has its own unique security domain.

The security domain of a role includes the privileges granted to the role plus those privileges granted to any roles that are granted to the role.

The security domain of a user includes privileges on all schema objects in the corresponding schema, the privileges granted to the user, and the privileges of roles granted to the user that are **currently enabled**. (A role can be simultaneously enabled for one user and disabled for another.) This domain also includes the privileges and roles granted to the role `PUBLIC`. The `PUBLIC` role represents all users in the database.

How Roles Work in PL/SQL Blocks

Role behavior in a PL/SQL block is determined by the type of block and by definer's rights or invoker's rights.

Roles Used in Named Blocks with Definer's Rights

All roles are disabled in any named PL/SQL block that executes with definer's rights.

Examples of named PL/SQL blocks are stored procedures, functions, and triggers.

Roles are not used for privilege checking and you cannot set roles within a definer's rights procedure.

The `SESSION_ROLES` data dictionary view shows all roles that are currently enabled and if a PL/SQL block executes with definer’s rights. If a named PL/SQL block that executes with definer's rights queries `SESSION_ROLES`, then the query does not return any rows.

See Also:

Oracle Database Reference for more information about the `SESSION_ROLES` data dictionary view

Roles Used in Named Blocks with Invoker's Rights and Anonymous PL/SQL Blocks

Named PL/SQL blocks that execute with invoker's rights and anonymous PL/SQL blocks are executed based on privileges granted through enabled roles.

Current roles are used for privilege checking within an invoker's rights PL/SQL block. You can use dynamic SQL to set a role in the session.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for an explanation of how invoker's and definer's rights can be used for name resolution and privilege checking
- *Oracle Database PL/SQL Packages and Types Reference* for information about dynamic SQL in PL/SQL

How Roles Aid or Restrict DDL Usage

A user requires one or more privileges to successfully execute a DDL statement, depending on the statement.

For example, to create a table, the user must have the `CREATE TABLE` or `CREATE ANY TABLE` system privilege.

To create a view of a table that belongs to another user, the creator must have the `CREATE VIEW` or `CREATE ANY VIEW` system privilege and either the `SELECT` *`object`* privilege for the table or the `SELECT ANY TABLE` system privilege.

Oracle Database avoids the dependencies on privileges received by way of roles by restricting the use of specific privileges in certain DDL statements. The following rules describe these privilege restrictions concerning DDL statements:

- All system privileges and object privileges that permit a user to perform a DDL operation are usable when received through a role. For example:
  - **System privileges:** `CREATE TABLE`, `CREATE VIEW`, and `CREATE PROCEDURE` privileges
  - **Object privileges:** `ALTER` and `INDEX` privileges for a table

    You cannot use the `REFERENCES` object privilege for a table to define the foreign key of a table if the privilege is received through a role.
- All system privileges and object privileges that allow a user to perform a DML operation that is required to issue a DDL statement are *not* usable when received through a role. The security domain does not contain roles when a `CREATE VIEW` statement is used. For example, a user who is granted the `SELECT ANY TABLE` system privilege or the `SELECT` *`object`* privilege for a table through a role cannot use either of these privileges to create a view on a table that belongs to another user. This is because views are definer's rights objects, so when creating them you cannot use any privileges (neither system privileges or object privileges) granted to you through a role. If the privilege is granted directly to you, then you can use the privilege. However, if the privilege is revoked at a later time, then the view definition becomes invalid ("contains errors") and must recompiled before it can be used again.

The following example further clarifies the permitted and restricted uses of privileges received through roles.

Assume that a user is:

- Granted a role that has the `CREATE VIEW` system privilege
- Directly granted a role that has the `SELECT` *`object`* privilege for the `employees` table
- Directly granted the `SELECT` *`object`* privilege for the `departments` table

Given these directly and indirectly granted privileges:

- The user can issue `SELECT` statements on both the `employees` and `departments` tables.
- Although the user has both the `CREATE VIEW` and `SELECT` privilege for the `employees` table through a role, the user cannot create a view on the `employees` table, because the `SELECT` *`object`* privilege for the `employees` table was granted through a role.

- The user can create a view on the `departments` table, because the user has the `CREATE VIEW` privilege through a role and the `SELECT` privilege for the `departments` table directly.

How Operating Systems Can Aid Roles

In some environments, you can administer database security using the operating system.

The operating system can be used to grant and revoke database roles and to manage their password authentication. This capability is not available on all operating systems.

See Also:

Your operating system-specific Oracle Database documentation for details about managing roles through the operating system

How Roles Work in a Distributed Environment

In a distributed database environment, all necessary roles must be set as the default role for a distributed (remote) session.

These roles cannot be enabled when the user connects to a remote database from within a local database session. For example, the user cannot execute a remote procedure that attempts to enable a role at the remote site.

See Also:

Oracle Database Heterogeneous Connectivity User's Guide

Predefined Roles in an Oracle Database Installation

Oracle Database provides a set of predefined roles to help in database administration.

These predefined roles, listed in Table 4-3, are automatically defined for Oracle databases when you run the standard scripts (such as `catalog.sql` and `catproc.sql`) that are part of database creation, and they are considered common roles. If you install other options or products, then other predefined roles may be created. You can find roles that are created and maintained by Oracle by querying the `ROLE` and `ORACLE_MAINTAINED` columns of the `DBA_ROLES` data dictionary view. If the output for `ORACLE_MAINTAINED` is `Y`, then you must not modify the role except by running the script that was used to create it.

Table 4-3 Oracle Database Predefined Roles

Predefined Role	Description
`ADM_PARALLEL_EXECUTE_TASK`	Provides privileges to update table data in parallel by using the `DBMS_PARALLEL_EXECUTE` PL/SQL package. **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_PARALLEL_EXECUTE` PL/SQL package.
`AQ_ADMINISTRATOR_ROLE`	Provides privileges to administer Advanced Queuing. Includes `ENQUEUE ANY QUEUE`, `DEQUEUE ANY QUEUE`, and `MANAGE ANY QUEUE`, `SELECT` privileges on Advanced Queuing tables and `EXECUTE` privileges on Advanced Queuing packages.
`AQ_USER_ROLE`	De-supported, but kept mainly for release 8.0 compatibility. Provides `EXECUTE` privileges on the `DBMS_AQ` and `DBMS_AQIN` packages.
`AUDIT_ADMIN`	Provides privileges to create unified and fine-grained audit policies, use the `AUDIT` and `NOAUDIT` SQL statements, view audit data, and manage the audit trail administration **See Also:** Who Can Perform Auditing?
`AUDIT_VIEWER`	Provides privileges to view and analyze audit data **See Also:** Who Can Perform Auditing?
`AUTHENTICATEDUSER`	Used by the XDB protocols to define any user who has logged in to the system. **See Also:** *Oracle XML DB Developer’s Guide* for more information about how this role is used for `DBUriServlet` security
`CAPTURE_ADMIN`	Provides the privileges necessary to create and manage privilege analysis policies. **See Also:** *Oracle Database Vault Administrator’s Guide* for more information
`CDB_DBA`	Provides the privileges required for administering a CDB, such as `SET CONTAINER`, `SELECT ON PDB_PLUG_IN_VIOLATIONS`, and `SELECT ON CDB_LOCAL_ADMIN_PRIVS`. If your site requires additional privileges, then you can create a role (either common or local) to cover these privileges, and then grant this role to the `CDB_DBA` role. **See Also:** *Oracle Database Administrator’s Guide* for information about administrating CDBs
`CONNECT`	Provides the `CREATE SESSION` system privilege. This role is provided for compatibility with previous releases of Oracle Database. You can determine the privileges encompassed by this role by querying the `DBA_SYS_PRIVS` data dictionary view. **Note:** Oracle recommends that you design your own roles for database security rather than relying on this role. This role may not be created automatically by future releases of Oracle Database. **See Also:** *Oracle Database Reference* for a description of the `DBA_SYS_PRIVS` view
`CSW_USR_ROLE`	Provides user privileges to manage the Catalog Services for the Web (CSW) component of Oracle Spatial. **See Also:** *Oracle Spatial and Graph Developer's Guide* for more information

Table 4-3 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
`CTXAPP`	Provides privileges to create Oracle Text indexes and index preferences, and to use PL/SQL packages. This role should be granted to Oracle Text users. **See Also:** *Oracle Text Application Developer's Guide* for more information
`CWM_USER`	Provides privileges to manage Common Warehouse Metadata (CWM), which is a repository standard used by Oracle data warehousing and decision support. **See Also:** *Oracle Database Data Warehousing Guide* for more information
`DATAPUMP_EXP_FULL_DATABASE`	Provides privileges to export data from an Oracle database using Oracle Data Pump. **Caution:** This is a very powerful role because it provides a user access to any data in any schema in the database. Use caution when granting this role to users. **See Also:** *Oracle Database Utilities* for more information
`DATAPUMP_IMP_FULL_DATABASE`	Provides privileges to import data into an Oracle database using Oracle Data Pump. **Caution:** This is a very powerful role because it provides a user access to any data in any schema in the database. Use caution when granting this role to users. **See Also:** *Oracle Database Utilities* for more information
`DBA`	Provides a large number of system privileges, including the `ANY` privileges (such as `DELETE ANY TABLE`) and the `GRANT ANY PRIVILEGE` privilege. This role is provided for compatibility with previous releases of Oracle Database. You can find the privileges that are encompassed by this role by querying the `DBA_SYS_PRIVS` data dictionary view. **Note:** Oracle recommends that you design your own roles for database security rather than relying on this role. This role may not be created automatically by future releases of Oracle Database. **See Also:** *Oracle Database Reference* for a description of the `DBA_SYS_PRIVS` view
`DBFS_ROLE`	Provides access to the DBFS (the Database Filesystem) packages and objects. **See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide*
`EJBCLIENT`	Provides privileges to connect to EJBs from a Java stored procedure.
`EM_EXPRESS_ALL`	Enables users to connect to Oracle Enterprise Manager (EM) Express and use all the functionality provided by EM Express (read and write access to all EM Express features). The `EM_EXPRESS_ALL` role includes the `EM_EXPRESS_BASIC` role. **See Also:** *Oracle Database 2 Day DBA* for more information
`EM_EXRESS_BASIC`	Enables users to connect to EM Express and to view the pages in read-only mode. The `EM_EXPRESS_BASIC` role includes the `SELECT_CATALOG_ROLE` role. **See Also:** *Oracle Database 2 Day DBA* for more information
`EXECUTE_CATALOG_ROLE`	Provides `EXECUTE` privileges on objects in the data dictionary.

Table 4-3 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
`EXP_FULL_DATABASE`	Provides the privileges required to perform full and incremental database exports using the Export utility (later replaced with Oracle Data Pump). It includes these privileges: `SELECT ANY TABLE`, `BACKUP ANY TABLE`, `EXECUTE ANY PROCEDURE`, `EXECUTE ANY TYPE`, `ADMINISTER RESOURCE MANAGER`, and `INSERT`, `DELETE`, and `UPDATE` on the tables `SYS.INCVID`, `SYS.INCFIL`, and `SYS.INCEXP`. Also includes the following roles: `EXECUTE_CATALOG_ROLE` and `SELECT_CATALOG_ROLE`. This role is provided for convenience in using the export and import utilities. **Caution:** This is a very powerful role because it provides a user access to any data in any schema in the database. Use caution when granting this role to users. **See Also:** *Oracle Database Utilities* for more information
`GATHER_SYSTEM_STATISTICS`	Provides privileges to update system statistics, which are collected using the `DBMS_STATS.GATHER_SYSTEM_STATISTICS` procedure **See Also:** *Oracle Database SQL Tuning Guide* for more information about managing optimizer statistics
`GLOBAL_AQ_USER_ROLE`	Provides privileges to establish a connection to an LDAP server, for use with Oracle Streams AQ. **See Also:** *Oracle Database Advanced Queuing User's Guide* for more information
`HS_ADMIN_EXECUTE_ROLE`	Provides the `EXECUTE` privilege for users who want to use the Heterogeneous Services (HS) PL/SQL packages. **See Also:** *Oracle Database Heterogeneous Connectivity User's Guide* for more information
`HS_ADMIN_ROLE`	Provides privileges to both use the Heterogeneous Services (HS) PL/SQL packages and query the HS-related data dictionary views. **See Also:** *Oracle Database Heterogeneous Connectivity User's Guide* for more information
`HS_ADMIN_SELECT_ROLE`	Provides privileges to query the Heterogeneous Services data dictionary views. **See Also:** *Oracle Database Heterogeneous Connectivity User's Guide*for more information
`IMP_FULL_DATABASE`	Provides the privileges required to perform full database imports using the Import utility (later replaced with Oracle Data Pump). Includes an extensive list of system privileges (use view `DBA_SYS_PRIVS` to view privileges) and the following roles: `EXECUTE_CATALOG_ROLE` and `SELECT_CATALOG_ROLE`. This role is provided for convenience in using the export and import utilities. **Caution:** This is a very powerful role because it provides a user access to any data in any schema in the database. Use caution when granting this role to users. **See Also:** *Oracle Database Utilities*for more information
`JAVADEBUGPRIV`	Provides privileges to run the Oracle Database Java applications debugger. **See Also:** *Oracle Database Java Developer’s Guide* for more information about managing security for Oracle Java applications
`JAVAIDPRIV`	Deprecated for this release.

Table 4-3 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
`JAVASYSPRIV`	Provides major permissions to use Java2, including updating Oracle JVM-protected packages. **See Also:** *Oracle Database Java Developer's Guide* for more information about managing security for Oracle Java applications
`JAVAUSERPRIV`	Provides limited permissions to use Java2. **See Also:** *Oracle Database Java Developer's Guide* for more information about managing security for Oracle Java applications
`JAVA_ADMIN`	Provides administrative permissions to update policy tables for Oracle Database Java applications. **See Also:** *Oracle Database Java Developer's Guide* for more information about managing security for Oracle Java applications
`JAVA_DEPLOY`	Provides privileges to deploy `ncomp` DLLs into the `javavm/admin` directory using the `ncomp` and `deployns` utilities. With this role, the `javavm/deploy` and `javavm/admin` directories can be accessible. **See Also:** *Oracle Database Development Guide* for more information
`JMXSERVER`	Provides privileges to start and maintain a JMX agent in a database session. **See Also:** *Oracle Database Java Developer's Guide* for more information about managing Oracle Java applications
`LBAC_DBA`	Provides permissions to use the `SA_SYSDBA` PL/SQL package. **See Also:** *Oracle Label Security Administrator's Guide* for more information
`LOGSTDBY_ADMINISTRATOR`	Provides administrative privileges to manage the SQL Apply (logical standby database) environment. **See Also:** *Oracle Data Guard Concepts and Administration* for more information
`OEM_ADVISOR`	Provides privileges to create, drop, select (read), load (write), and delete a SQL tuning set through the `DBMS_SQLTUNE` PL/SQL package, and to access to the Advisor framework using the `ADVISOR` PL/SQL package. **See Also:** *Oracle Database SQL Tuning Guide* for more information
`OEM_MONITOR`	Provides privileges needed by the Management Agent component of Oracle Enterprise Manager to monitor and manage the database. **See Also:** *Oracle Database SQL Tuning Guide* for more information
`OLAP_DBA`	Provides administrative privileges to create dimensional objects in different schemas for Oracle OLAP. **See Also:** *Oracle OLAP User's Guide* for more information
`OLAP_USER`	Provides application developers privileges to create dimensional objects in their own schemas for Oracle OLAP. **See Also:** *Oracle OLAP User's Guide* for more information
`OLAP_XS_ADMIN`	Provides privileges to administer security for Oracle OLAP. **See Also:** *Oracle OLAP User's Guide* for more information

Table 4-3 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
OPTIMIZER_PROCESSING_RATE	Provides privileges to execute the GATHER_PROCESSING_RATE, SET_PROCESSING_RATE, and DELETE_PROCESSING_RATE procedures in the DBMS_STATS package. These procedures manage the processing rate of a system for automatic degree of parallelism (Auto DOP). Auto DOP uses these processing rates to determine the optimal degree of parallelism for a SQL statement. **See Also:** *Oracle Database SQL Tuning Guide* for more information
ORDADMIN	Provides privileges to administer Oracle Multimedia DICOM. **See Also:** *Oracle Multimedia DICOM Developer's Guide*
PDB_DBA	Granted automatically to the local user that is created when you create a new PDB from the seed PDB. No privileges are provided with this role. **See Also:** *Oracle Database Administrator's Guide* for more information about creating PDBs using the seed
PROVISIONER	Provides privileges to register and update global callbacks for Oracle Database Real Application sessions and to provision principals. **See Also:** *Oracle Database Real Application Security Administrator's and Developer's Guide* for more information.
RECOVERY_CATALOG_OWNER	Provides privileges for owner of the recovery catalog. Includes: CREATE SESSION, ALTER SESSION, CREATE SYNONYM, CREATE ANY SYNONYM, DROP ANY SYNONYM, CREATE VIEW, CREATE DATABASE LINK, CREATE TABLE, CREATE CLUSTER, CREATE SEQUENCE, CREATE TRIGGER, CREATE ANY TRIGGER, QUERY REWRITE, CREATE ANY CONTEXT, EXECUTE ON DBMS_RLS, ADMINISTER DATABASE, and CREATE PROCEDURE **See Also:** *Oracle Database Backup and Recovery User's Guide* for more information.
RESOURCE	Provides the following system privileges: CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE. Be aware that RESOURCE no longer provides the UNLIMITED TABLESPACE system privilege. This role is provided for compatibility with previous releases of Oracle Database. You can determine the privileges encompassed by this role by querying the DBA_SYS_PRIVS data dictionary view. **Note:** Oracle recommends that you design your own roles for database security rather than relying on this role. This role may not be created automatically by future releases of Oracle Database. **See Also:** *Oracle Database Reference* for a description of the DBA_SYS_PRIVS view
SCHEDULER_ADMIN	Allows the grantee to execute the procedures of the DBMS_SCHEDULER package. It includes all of the job scheduler system privileges and is included in the DBA role. **See Also:** *Oracle Database Administrator's Guide* for more information about the DBMS_SCHEDULER package
SELECT_CATALOG_ROLE	Provides SELECT privilege on objects in the data dictionary.
SODA_APP	Provides privileges to use the SODA APIs, in particular, to create, drop, and list document collections.

Table 4-3 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
`SPATIAL_CSW_ADMIN`	Provides administrative privileges to manage the Catalog Services for the Web (CSW) component of Oracle Spatial. **See Also:** *Oracle Spatial and Graph Developer's Guide* for more information
`SPATIAL_WFS_ADMIN`	Provides administrative privileges to manage the Web Feature Service (WFS) component of Oracle Spatial. **See Also:** *Oracle Spatial and Graph Developer's Guide* for more information
`WFS_USR_ROLE`	Provides user privileges for the Web Feature Service (WFS) component of Oracle Spatial. **See Also:** *Oracle Spatial and Graph Developer's Guide* for more information
`WM_ADMIN_ROLE`	Provides administrative privileges for Oracle Workspace Manager. This enables users to run any `DBMS_WM` procedures on all version enabled tables, workspaces, and savepoints regardless of their owner. It also enables the user to modify the system parameters specific to Workspace Manager. **See Also:** *Oracle Database Workspace Manager Developer's Guide* for more information
`XDBADMIN`	Allows the grantee to register an XML schema globally, as opposed to registering it for use or access only by its owner. It also lets the grantee bypass access control list (ACL) checks when accessing Oracle XML DB Repository. **See Also:** *Oracle XML DB Developer’s Guide* for information about XML schemas and the XML DB Repository
`XDB_SET_INVOKER`	Allows the grantee to define invoker's rights handlers and to create or update the resource configuration for XML repository triggers. By default, Oracle Database grants this role to the `DBA` role but not to the `XDBADMIN` role. **See Also:** *Oracle XML DB Developer’s Guide* for information about Oracle Database XML repository triggers
`XDB_WEBSERVICES`	Allows the grantee to access Oracle Database Web services over HTTPS. However, it does not provide the user access to objects in the database that are public. To allow public access, you need to grant the user the `XDB_WEBSERVICES_WITH_PUBLIC` role. For a user to use these Web services, `SYS` must enable the Web service servlets. **See Also:** *Oracle XML DB Developer’s Guide* for information about Oracle Database Web services
`XDB_WEBSERVICES_OVER_HTTP`	Allows the grantee to access Oracle Database Web services over HTTP. However, it does not provide the user access to objects in the database that are public. To allow public access, you need to grant the user the `XDB_WEBSERVICES_WITH_PUBLIC` role. **See Also:** *Oracle XML DB Developer’s Guide* for information about Oracle Database Web services
`XDB_WEBSERVICES_WITH_PUBLIC`	Allows the grantee access to public objects through Oracle Database Web services. **See Also:** *Oracle XML DB Developer’s Guide* for information about Oracle Database Web services

Table 4-3 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
`XS_CACHE_ADMIN`	In Oracle Database Real Application Security, enables the grantee to manage the mid-tier cache. It is required for caching the security policy at the mid-tier level for the `checkAcl` (authorization) method of the `XSAccessController` class. Grant this role to the application connection user or the Real Application Security dispatcher. **See Also:** *Oracle Database Real Application Security Administrator's and Developer's Guide* for more information
`XS_NSATTR_ADMIN`	In Oracle Database Real Application Security, enables the grantee to manage and manipulate the namespace and attribute for a session. Grant this role to the Real Application Security session user. **See Also:** *Oracle Database Real Application Security Administrator's and Developer's Guide* for information about managing Real Application Security sessions
`XS_RESOURCE`	In Oracle Database Real Application Security, enables the grantee to manage objects in the attached schema, through the `XS_ACL` PL/SQL package. This package creates procedures to create and manage access control lists (ACLs). It contains the `ADMIN SEC POLICY` privilege. It is similar to the Oracle Database `RESOURCE` role. **See Also:** *Oracle Database Real Application Security Administrator's and Developer's Guide* for more information
`XS_SESSION_ADMIN`	In Oracle Database Real Application Security, enables the grantee to manage the life cycle of a session, including the ability to create, attach, detach, and destroy the session. Grant this role to the application connection user or Real Application Security dispatcher. **See Also:** *Oracle Database Real Application Security Administrator's and Developer's Guide* for information about managing Real Application Security sessions

Note:

Each installation should create its own roles and assign only those privileges that are needed, thus retaining detailed control of the privileges in use. This process also removes any need to adjust existing roles, privileges, or procedures whenever Oracle Database changes or removes roles that Oracle Database defines. For example, the `CONNECT` role now has only one privilege: `CREATE SESSION`.

Creating a Role

You can create a role that is authenticated with or without a password. You also can create external or global roles.

About the Creation of Roles

You can create a role by using the `CREATE ROLE` statement.

To create the role, you must have the `CREATE ROLE` system privilege. Typically, only security administrators have this system privilege. After you create a role, the role has no privileges associated with it. Your next step is to grant either privileges or other roles to the new role.

You must give each role that you create a unique name among existing user names and role names of the database. Roles are not contained in the schema of any user. In a database that uses a multi-byte character set, Oracle recommends that each role name contain at least one single-byte character. If a role name contains only multi-byte characters, then the encrypted role name and password combination is considerably less secure. See Guideline 1 in Guidelines for Securing Passwords for password guidelines.

You can use the `IDENTIFIED BY` clause to authorize the role with a password. This clause specifies how the user must be authorized before the role can be enabled for use by a specific user to which it has been granted. If you do not specify this clause, or if you specify `NOT IDENTIFIED`, then no authorization is required when the role is enabled. Roles can be specified to be authorized by the following:

- The database using a password
- An application using a specified package
- Externally by the operating system, network, or other external source
- Globally by an enterprise directory service

As an alternative to creating password-protected roles, Oracle recommends that you use secure application roles instead.

Note the following restrictions about the creation of roles:

- A role and a user cannot have the same name.
- The role name cannot start with the value of the `COMMON_USER_PREFIX` parameter (which defaults to `C##`) unless this role is a CDB common role.

Creating a Role That Is Authenticated With a Password

You can create a password authenticated role by using the `IDENTIFIED BY` clause.

However, instead of using password-protected roles, consider using secure application roles instead.

- To create a password-authenticated role, use the `CREATE ROLE` statement with the `IDENTIFIED BY` clause.

For example:

```
CREATE ROLE clerk IDENTIFIED BY password;
```

Note:

If you set the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter is set to `11` or higher, then you must recreate roles that have been created with the `IDENTIFIED BY` clause.

Creating a Role That Has No Password Authentication

You can create a role that does not require a password by omitting the `IDENTIFIED BY` clause.

- Use the `CREATE ROLE` statement with no clauses to create a role that has no password authentication.

For example:

```
CREATE ROLE salesclerk;
```

Creating a Role That Is External or Global

An external user must be authorized by an external service, such as an operating system or third-party service, before enabling the role.

A global user must be authorized to use the role by the enterprise directory service before the role is enabled at login.

- To create a role to be authorized globally, use the `CREATE ROLE` statement with the `IDENTIFIED GLOBALLY` clause.

For example:

```
CREATE ROLE clerk IDENTIFIED GLOBALLY;
```

Altering a Role

The `ALTER ROLE` statement can modify the authorization method for a role.

To alter the authorization method for a role, you must have the `ALTER ANY ROLE` system privilege or have been granted the role with `ADMIN` option.

Remember that you can only directly grant secure application roles or password-authenticated roles to a user. Be aware that if you create a common role in the root, you cannot change it to a local role.

- To alter a role, use the `ALTER ROLE` statement.

  For example, to alter the `clerk` role to specify that the user must be authorized by an external source before enabling the role:

  ```
  ALTER ROLE clerk IDENTIFIED EXTERNALLY;
  ```

Specifying the Type of Role Authorization

You can configure a role to be authorized through different sources, such the database or an external source.

Authorizing a Role by Using the Database

You can protect a role authorized by the database by assigning the role a password.

If a user is granted a role protected by a password, then you can enable or disable the role by supplying the proper password for the role in the `SET ROLE` statement. You cannot authenticate a password-authenticated role on logon, even if you add it to the list of default roles. You must explicitly enable it with the `SET ROLE` statement using the required password.

1. Use the `CREATE ROLE` statement with the `IDENTIFIED BY` clause to create the password-authenticated role.

   Creating a Role That Is Authenticated With a Password shows a `CREATE ROLE` statement that creates a role called `clerk`. When the role is enabled, the password must be supplied.

2. Use the `SET ROLE` statement to set the password-authenticated role.

   The following example shows how to set a password-authenticated role by using the `SET ROLE` statement.

   ```
   SET ROLE clerk IDENTIFIED BY password;
   ```

   In a database that uses a multibyte character set, passwords for roles must include only single-byte characters. Multibyte characters are not accepted in passwords. See Guideline 1 in Guidelines for Securing Passwords for password guidelines.

Authorizing a Role by Using an Application

An application role can be enabled only by applications that use an authorized PL/SQL package.

Application developers do not need to secure a role by embedding passwords inside applications. Instead, they can create an application role (secure application role) and specify which PL/SQL package is authorized to enable the role.

- To create a role enabled by an authorized PL/SQL package, use the `IDENTIFIED USING` *`package_name`* clause in the `CREATE ROLE` SQL statement.

For example, to indicate that the role `admin_role` is an application role and the role can only be enabled by any module defined inside the PL/SQL package `hr.admin`:

```
CREATE ROLE admin_role IDENTIFIED USING hr.admin;
```

See Also:

- Role Privileges and Secure Application Roles
- Creating Secure Application Roles to Control Access to Applications
- *Oracle Database 2 Day + Security Guide*

Authorizing a Role by Using an External Source

Oracle Database supports the use of external roles but with certain limitations.

You can define an external role locally in the database, but you cannot grant the external role to global users, to global roles, or to any other roles in the database. You can create roles that are authorized by the operating system or network clients.

- To authorize a role by using an external source, use the `CREATE ROLE` statement with the `IDENTIFIED EXTERNALLY` clause.

For example:

```
CREATE ROLE accts_rec IDENTIFIED EXTERNALLY;
```

Authorizing a Role by Using the Operating System

Oracle Database supports role authentication through the operating system but with certain limitations.

Role authentication through the operating system is useful only when the operating system is able to dynamically link operating system privileges with applications.

When a user starts an application, the operating system grants an operating system privilege to the user. The granted operating system privilege corresponds to the role associated with the application. At this point, the application can enable the application role. When the application is terminated, the previously granted operating system privilege is revoked from the operating system account of the user.

- If a role is authorized by the operating system, then configure information for each user at the operating system level. This operation is operating system dependent.

If roles are granted by the operating system, then you do not need to have the operating system authorize them also.

Authorizing a Role by Using a Network Client

Oracle Database supports role authentication by a network client but you must be aware of security risks.

If users connect to the database over Oracle Net, then by default, the operating system cannot authenticate their roles. This includes connections through a shared server configuration, as this connection requires Oracle Net. This restriction is the default because a remote user could impersonate another operating system user over a network connection. Oracle recommends that you set `REMOTE_OS_ROLES` to `FALSE`, which is the default.

- If you are not concerned with this security risk and want to use operating system role authentication for network clients, then set the initialization parameter `REMOTE_OS_ROLES` in the database initialization parameter file to `TRUE`.

The change takes effect the next time you start the instance and mount the database.

Authorizing a Global Role by an Enterprise Directory Service

A global role enables a global user to be authorized only by an enterprise directory service.

You define the global role locally in the database by granting privileges and roles to it, but you cannot grant the global role itself to any user or other role in the database. When a global user attempts to connect to the database, the enterprise directory is queried to obtain any global roles associated with the user. Global roles are one component of enterprise user security. A global role only applies to one database, but you can grant it to an enterprise role defined in the enterprise directory. An enterprise role is a directory structure that contains global roles on multiple databases and can be granted to enterprise users.

- To create a global role to be authorized by an enterprise directory service, use the `CREATE ROLE` statement with the `IDENTIFIED GLOBALLY` clause.

For example:

```
CREATE ROLE supervisor IDENTIFIED GLOBALLY;
```

See Also:

- Global User Authentication and Authorization for a general discussion of global authentication and authorization of users, and its role in enterprise user management
- *Oracle Database Enterprise User Security Administrator's Guide* for information about implementing enterprise user management

Granting and Revoking Roles

You can grant or revoke privileges to and from roles, and then grant these roles to users or to other roles.

About Granting and Revoking Roles

You can grant system or object privileges to a role, and grant any role to any database user or to another role.

However, a role cannot be granted to itself, nor can the role be granted circularly, that is, role `X` cannot be granted to role `Y` if role `Y` has previously been granted to role `X`.

To provide selective availability of privileges, Oracle Database permits applications and users to enable and disable roles. Each role granted to a user is, at any given time, either enabled or disabled. The security domain of a user includes the privileges of all roles currently enabled for the user and excludes the privileges of any roles currently disabled for the user.

A role granted to a role is called an indirectly granted role. You can explicitly enable or disable it for a user. However, whenever you enable a role that contains other roles, you implicitly enable all indirectly granted roles of the directly granted role.

You grant roles by using the `GRANT` statement, and revoke them by using the `REVOKE` statement. Privileges are granted to and revoked from roles using the same statements.

You cannot grant a secure role (that is, an `IDENTIFIED BY` role, `IDENTIFIED USING` role, or `IDENTIFIED EXTERNALLY` role) to either another secure role or to a non-secure role. You can use the `SET ROLE` statement to enable the secure role for the session.

Who Can Grant or Revoke Roles?

The `GRANT ANY ROLE` system privilege enables users to grant or revoke any role except global roles to or from other users or roles.

A global role is managed in a directory, such as Oracle Internet Directory, but its privileges are contained within a single database. By default, the `SYS` or `SYSTEM` user has the `GRANT ANY ROLE` privilege. You should grant this system privilege conservatively because it is very powerful.

Any user granted a role with the `ADMIN OPTION` can grant or revoke that role to or from other users or roles of the database. This option allows administrative powers for roles to be granted on a selective basis.

See Also:

Oracle Database Enterprise User Security Administrator's Guide for information about global roles

Granting and Revoking Roles to and from Program Units

You can grant roles to function, procedure, and PL/SQL package program units.

The role then becomes enabled during the execution of the program unit, but not during the compilation of the program unit. This enables you to temporarily escalate privileges in the PL/SQL code without granting the role directly to the user. It also increases security for applications and helps to enforce the principle of least privilege.

- Use the `GRANT` or `REVOKE` statement to grant or revoke a role to a program unit.

The following example shows how to grant the same role to the PL/SQL package `checkstats_pkg`:

```
GRANT clerk_admin TO package psmith.checkstats_pkg;
```

This example shows how to revoke the `clerk_admin` role from the PL/SQL package `checkstats_pkg`:

```
REVOKE clerk_admin FROM package psmith.checkstats_pkg;
```

The following example shows how to grant the role `clerk_admin` to the procedure `psmith.check_stats_proc`.

```
GRANT clerk_admin TO PROCEDURE psmith.checkstats_proc;
```

Dropping Roles

Dropping a role affects the security domains of users or roles who had been granted the role.

That is, the security domains of all users and roles that were granted to the dropped role are changed to reflect the absence of the dropped role privileges.

All indirectly granted roles of the dropped role are also removed from affected security domains. Dropping a role automatically removes the role from all user default role lists.

Because the existence of objects is not dependent on the privileges received through a role, tables and other objects are not dropped when a role is dropped.

To drop a role, you must have the `DROP ANY ROLE` system privilege or have been granted the role with the `ADMIN` option.

- To drop a role, use the `DROP ROLE` statement.

For example, to drop the role `CLERK`:

```
DROP ROLE clerk;
```

Restricting SQL*Plus Users from Using Database Roles

You should restrict SQL*Plus users from using database roles, which helps to safeguard the database from intruder attacks.

Potential Security Problems of Using Ad Hoc Tools

Ad hoc tools can pose problems if malicious users have access to such tools.

Prebuilt database applications explicitly control the potential actions of a user, including the enabling and disabling of user roles while using the application. By contrast, ad hoc query tools such as SQL*Plus, permit a user to submit any SQL statement (which may or may not succeed), including enabling and disabling a granted role.

Potentially, an application user can exercise the privileges attached to that application to issue destructive SQL statements against database tables by using an ad hoc tool.

For example, consider the following scenario:

- The Vacation application has a corresponding `vacation` role.
- The `vacation` role includes the privileges to issue `SELECT`, `INSERT`, `UPDATE`, and `DELETE` statements against the `emp_tab` table.
- The Vacation application controls the use of privileges obtained through the `vacation` role.

Now, consider a user who has been granted the `vacation` role. Suppose that, instead of using the Vacation application, the user executes SQL*Plus. At this point, the user is restricted only by the privileges granted to him explicitly or through roles, including the `vacation` role. Because SQL*Plus is an ad hoc query tool, the user is not restricted to a set of predefined actions, as with designed database applications. The user can query or modify data in the `emp_tab` table as he or she chooses.

How the PRODUCT_USER_PROFILE System Table Can Limit Roles

The `SYSTEM` schema `PRODUCT_USER_PROFILE` table can disable SQL and SQL*Plus commands in the SQL*Plus environment for each user.

SQL*Plus, not the Oracle Database, enforces this security. You can even restrict access to the `GRANT`, `REVOKE`, and `SET ROLE` commands to control user ability to change their database privileges.

The `PRODUCT_USER_PROFILE` table enables you to list roles that you do not want users to activate with an application. You can also explicitly disable the use of various commands, such as `SET ROLE`.

For example, you could create an entry in the `PRODUCT_USER_PROFILE` table to:

- Disallow the use of the `clerk` and `manager` roles with SQL*Plus
- Disallow the use of `SET ROLE` with SQL*Plus

Suppose user Marla connects to the database using SQL*Plus. Marla has the `clerk`, `manager`, and `analyst` roles. As a result of the preceding entry in `PRODUCT_USER_PROFILE`, Marla is only able to exercise her `analyst` role with SQL*Plus. Also, when Ginny attempts to issue a `SET ROLE` statement, she is explicitly prevented from doing so because of the entry in the `PRODUCT_USER_PROFILE` table prohibiting use of `SET ROLE`.

Be aware that the `PRODUCT_USER_PROFILE` table does not completely guarantee security, for multiple reasons. In the preceding example, while `SET ROLE` is disallowed with SQL*Plus, if Marla had other privileges granted to her directly, then she could exercise these using SQL*Plus.

See Also:

*SQL*Plus User's Guide and Reference* for more information about the `PRODUCT_USER_PROFILE` table

How Stored Procedures Can Encapsulate Business Logic

Stored procedures encapsulate privileges use with business logic so that privileges are only exercised in the context of a well-formed business transaction.

For example, an application developer can create a procedure to update the employee name and address in the `employees` table, which enforces that the data can only be updated in normal business hours.

In addition, rather than grant a human resources clerk the `UPDATE` privilege on the `employees` table, a security administrator may grant the privilege on the procedure only. Then, the human resources clerk can exercise the privilege only in the context of the procedures, and cannot update the `employees` table directly.

Role Privileges and Secure Application Roles

A secure application role can be enabled only by an authorized PL/SQL package or procedure.

The PL/SQL package itself reflects the security policies that are necessary to control access to the application.

This method of role creation restricts the enabling of this type of role to the invoking application. For example, the application can perform authentication and customized authorization, such as checking whether the user has connected through a proxy.

This type of role strengthens security because passwords are not embedded in application source code or stored in a table. This way, the actions the database performs are based on the implementation of your security policies, and these definitions are stored in one place, the database, rather than in your applications. If you need to modify the policy, you do so in one place without having to modify your applications. No matter how users connect to the database, the result is always the same, because the policy is bound to the role.

To enable the secure application role, you must execute its underlying package by invoking it directly from the application when the user logs in, before the user exercises the privileges granted by the secure application role. You cannot use a logon trigger to enable a secure application role, nor can you have this type of role be a default role.

When you enable the secure application role, Oracle Database verifies that the authorized PL/SQL package is on the calling stack, that is, it verifies that the authorized PL/SQL package is issuing the command to enable the role.

You can use secure application roles to ensure the existence of a database connection. Because a secure application role is a role implemented by a package, the package can validate that users can connect to the database through a middle tier or from a specific IP address. In this way, the secure application role prevents users from accessing data outside an application. They are forced to work within the framework of the application privileges that they have been granted.

See Also:

- Creating Secure Application Roles to Control Access to Applications
- *Oracle Database 2 Day + Security Guide*

Restricting Operations on PDBs Using PDB Lockdown Profiles

You can use PDB lockdown profiles in a multitenant environment to restrict sets of user operations in pluggable databases (PDBs).

This section contains the following topics:

About PDB Lockdown Profiles

A PDB lockdown profile is a named set of features that controls a group of operations.

In some cases, you can enable or disable operations individually. For example, a PDB lockdown profile can contain settings to disable specific clauses that come with the `ALTER SYSTEM` statement.

PDB lockdown profiles restrict user access to the functionality the features provided, similar to resource limits that are defined for users. As the name suggests, you use PDB lockdown profiles in a CDB, for an application container, or for a PDB or application PDB. You can create custom profiles to accommodate the requirements of your site. PDB profiles enable you to define custom security policies for an application. In addition, you can create a lockdown profile that is based on another profile, called a **base profile**. You can configure this profile to be dynamically updated when the base profile is modified, or configure it to be static (unchanging) when the base profile is updated. Lockdown profiles are designed for both Oracle Cloud and on-premise environments.

When identities are shared between PDBs, elevated privileges may exist. You can use lockdown profiles to prevent this elevation of privileges. Identities can be shared in the following situations:

- At the operating system level, when the database interacts with operating system resources such as files or processes
- At the network level, when the database communicates with other systems, and network identity is important

- Inside the database, as PDBs access or create common objects or they communicate across container boundaries using features such as database links

The features that use shared identifies and that benefit from PDB lockdown profiles are in the following categories:

- **Network access features.** These are operations that use the network to communicate outside the PDB. For example, the PL/SQL packages `UTL_TCP`, `UTL_HTTP`, `UTL_MAIL`, `UTL_SNMP`, `UTL_INADDR`, and `DBMS_DEBUG_JDWP` perform these kinds of operations. Currently, ACLs are used to control this kind of access to share network identity.
- **Common user or object access.** These are operations in which a local user in the PDB can proxy through common user accounts or access objects in a common schema. These kinds of operations include adding or replacing objects in a common schema, granting privileges to common objects, accessing common directory objects, granting the `INHERIT PRIVILEGES` role to a common user, and manipulating a user proxy to a common user.
- **Operating System access.** For example, you can restrict access to the `UTL_FILE` or `DBMS_FILE_TRANSFER` PL/SQL packages.
- **Connections.** For example, you can restrict common users from connecting to the PDB or you can restrict a local user who has the `SYSOPER` administrative privilege from connecting to a PDB that is open in restricted mode.

The general procedure for creating a PDB lockdown profile is to first create it in the CDB root or the application root using the `CREATE LOCKDOWN PROFILE` statement, and then use the `ALTER LOCKDOWN PROFILE` statement to add the restrictions.

To enable a PDB lockdown profile, you can use the `ALTER SYSTEM` statement to set the `PDB_LOCKDOWN` parameter. You can find information about existing PDB lockdown profiles by connecting to CDB or application root and querying the `DBA_LOCKDOWN_PROFILES` data dictionary view. A local user can find the contents of a PDB lockdown parameter by querying the `V$LOCKDOWN_RULES` dynamic data dictionary view.

PDB Lockdown Profile Inheritance

PDB lockdown profiles have inheritance behaviors between the CDB root, the application root, and their associated PDBs.

- The inheritance path between PDBs and their respective roots is as follows:
  - The `PDB_LOCKDOWN` parameter setting in a CDB PDB takes precedence over the `PDB_LOCKDOWN` parameter setting in the CDB root. Similarly, the `PDB_LOCKDOWN` setting in an application PDB takes precedence over a `PDB_LOCKDOWN` setting in the application root.
  - If a CDB PDB (or an application PDB) does not have the `PDB_LOCKDOWN` parameter set, then the PDB inherits the settings of the `PDB_LOCKDOWN` parameter in the CDB root (or the application root).
  - If the application root does not have the `PDB_LOCKDOWN` parameter set, then the application root inherits the settings of the `PDB_LOCKDOWN` parameter in the CDB root.
- If the `PDB_LOCKDOWN` parameter in a CDB PDB or an application PDB is set to a CDB lockdown profile, then the PDB ignores any lockdown profiles that are set by the `PDB_LOCKDOWN` parameter in the CDB root or the application root.

- PDB lockdown parameters can inherit rules that are stipulated in an application lockdown profile, including the disable rules that come from a CDB lockdown profile that was set in its nearest ancestor (that is, an application root or the CDB root). This applies in the case of when a `PDB_LOCKDOWN` parameter in an application PDB is set to an application lockdown profile while the `PDB_LOCKDOWN` parameter in the application root or the CDB root is set to a CDB lockdown profile.
- Sometimes a conflict arises between the rules that comprise a CDB lockdown profile and an application lockdown profile. In this case, the rules in the CDB lockdown profile take precedence. For example, the setting for an `OPTION_VALUE` clause in the CDB lockdown profile takes precedence over the setting for the `OPTION_VALUE` clause in an application lockdown profile.

Default PDB Lockdown Profiles

The default PDB lockdown profiles are `PRIVATE_DBAAS`, `PUBLIC_DBAAS`, and `SAAS`.

Detailed information about these profiles is as follows:

- `PRIVATE_DBAAS` incorporates restrictions that are suitable for private Cloud Database-as-a-Service (DBaaS) deployments. These restrictions are:
  - Must have the same database administrator for each PDB
  - Different users permitted to connect to the database
  - Different applications permitted

  `PRIVATE_DBAAS` permits users to connect to the PDBs but prevents them from using Oracle Database administrative features.
- `SAAS` incorporates restrictions that are suitable for Software-as-a-Service (SaaS) deployments. These restrictions are:
  - Must have the same database administrator for each PDB
  - Different users permitted to connect to the database
  - Must use the same application

  The `SAAS` lockdown profile is more restrictive than the `PRIVATE_DBAAS` profile. Users can be different, but the application code is the same; users are prevented from directly connecting and must connect only through the application; and users are not granted the ability to perform any administrative features.
- `PUBLIC_DBAAS` incorporates restrictions that are suitable for public Cloud Database-as-a-Service (DBaaS) deployments. The restrictions are as follows:
  - Different DBAs in each PDB
  - Different users
  - Different applications

  The `PUBLIC_DBAAS` lockdown profile is the most restrictive of the lockdown profiles.

Creating a PDB Lockdown Profile

To create a PDB lockdown profile, you must have the `CREATE LOCKDOWN PROFILE` system privilege.

After you create the lockdown profile, you can add restrictions before enabling it.

1. Connect to the CDB root or the application root as a user who has the `CREATE LOCKDOWN PROFILE` system privilege.

   For example, to connect to the CDB root:

   ```
   CONNECT c##sec_admin
   Enter password: password
   ```

2. Run the `CREATE LOCKDOWN PROFILE` statement to create the profile by using the following syntax:

   ```
   CREATE LOCKDOWN PROFILE profile_name
   [FROM static_base_profile | INCLUDING dynamic_base_profile];
   ```

   In this specification:

   - *`profile_name`* is the name that you assign the lockdown profile. You can find existing names by querying the `PROFILE_NAMES` column of the `DBA_LOCKDOWN_PROFILES` data dictionary view.
   - `FROM` *`static_base_profile`* creates a new lockdown profile by using the values from an existing profile. Any subsequent changes to the base profile will not affect the new profile.
   - `INCLUDING` *`dynamic_base_profile`* also creates a new lockdown profile by using the values from an existing base profile, except that this new lockdown profile will inherit the `DISABLE STATEMENT` rules that comprise the base profile, as well as any subsequent changes to the base profile. If rules that are explicitly added to the new profile conflict with the rules in the base profile, then the rules in the base profile take precedence. For example, an `OPTION_VALUE` clause in the base profile takes precedence over the `OPTION_VALUE` clause in the new profile.

   The following two PDB lockdown profile statements demonstrate how the inheritance works:

   ```
   CREATE LOCKDOWN PROFILE hr_prof INCLUDING PRIVATE_DBAAS;
   CREATE LOCKDOWN PROFILE hr_prof2 FROM hr_prof;
   ```

   In the first statement, `hr_prof` inherits any changes made to the `PRIVATE_DBAAS` base profile. If a new statement is enabled for `PRIVATE_DBAAS`, then it is enabled for `hr_prof`. In the second statement, in contrast, when `hr_prof` changes, then `hr_prof2` does *not* change because it is independent of its base profile.

3. Run the `ALTER LOCKDOWN PROFILE` statement to provide restrictions for the profile.

   For example:

   ```
   ALTER LOCKDOWN PROFILE hr_prof DISABLE STATEMENT  = ('ALTER SYSTEM');
   ALTER LOCKDOWN PROFILE hr_prof ENABLE STATEMENT = ('ALTER SYSTEM') clause =
   ('flush shared_pool');
   ALTER LOCKDOWN PROFILE hr_prof DISABLE FEATURE = ('XDB_PROTOCOLS');
   ```

   In the preceding example:

   - `DISABLE STATEMENT = ('ALTER SYSTEM')` disables the use of all `ALTER SYSTEM` statements for the PDB.
   - `ENABLE STATEMENT = ('ALTER SYSTEM') clause = ('flush shared_pool')` enables only the use of the `FLUSH_SHARED_POOL` clause for `ALTER SYSTEM`.
   - `DISABLE FEATURE = ('XDB_PROTOCOLS')` prohibits the use of the XDB protocols (FTP, HTTP, HTTPS) by this PDB

After you create a PDB lockdown profile, you are ready to enable it by using the `ALTER SYSTEM SET PDB_LOCKDOWN` SQL statement.

Enabling or Disabling a PDB Lockdown Profile

To enable or disable a PDB lockdown profile, use the `PDB_LOCKDOWN` initialization parameter

You can use `ALTER SYSTEM SET PDB_LOCKDOWN` to enable a lockdown profile in any of the following contexts:

- CDB (affects all PDBs)
- Application root (affects all application PDBs in the container)
- Application PDB
- PDB

> **Note:**
>
> It is not necessary to restart the instance to enable the profile. When the `ALTER SYSTEM SET PDB_LOCKDOWN` statement completes, the profile rules take effect immediately.

When you set `PDB_LOCKDOWN` in the CDB root, every PDB and application root inherits this setting unless `PDB_LOCKDOWN` is set at the container level. To disable lockdown profiles, set `PDB_LOCKDOWN` to null. If you set this parameter to null in the CDB root, then lockdown profiles are disabled for all PDBs except those that explicitly set a profile within the PDB.

A CDB common user who has been commonly granted the `SYSDBA` administrative privilege or the `ALTER SYSTEM` system privilege can set `PDB_LOCKDOWN` only to a lockdown profile that was created in the CDB root. An application common user with the application common `SYSDBA` administrative privilege or the `ALTER SYSTEM` system privilege can set `PDB_LOCKDOWN` only to a lockdown profile created in an application root.

1. Log in to the desired container as a user who has the commonly granted `ALTER SYSTEM` or commonly granted `SYSDBA` privilege.

   For example, to enable the profile for all PDBs, log in to the CDB root:

   ```
   CONNECT c##sec_admin
   Enter password: password
   ```

2. Run the `ALTER SYSTEM SET PDB_LOCKDOWN` statement.

   For example, the following statement enables the lockdown profile named `hr_prof` for all PDBs:

   ```
   ALTER SYSTEM SET PDB_LOCKDOWN = hr_prof;
   ```

   The following statement resets the `PDB_LOCKDOWN` parameter:

   ```
   ALTER SYSTEM RESET PDB_LOCKDOWN;
   ```

   This variation of the preceding statement includes the `SCOPE` clause::

```
ALTER SYSTEM RESET PDB_LOCKDOWN SCOPE = BOTH;
```

The following statement disables all lockdown profiles in the CDB except those that are explicitly set at the PDB level:

```
ALTER SYSTEM SET PDB_LOCKDOWN = '' SCOPE = BOTH;
```

To find the names of PDB lockdown profiles, query the `PROFILE_NAME` column of the `DBA_LOCKDOWN_PROFILES` data dictionary view.

3. Optionally, review information about the profiles by querying `DBA_LOCKDOWN_PROFILES`.

   For example, run the following query:

   ```
   SET LINESIZE 150
   COL PROFILE_NAME FORMAT a20
   COL RULE FORMAT a20
   COL CLAUSE FORMAT a25

   SELECT PROFILE_NAME, RULE, CLAUSE, STATUS FROM CDB_LOCKDOWN_PROFILES;
   ```

   Sample output appears below:

   ```
   PROFILE_NAME         RULE                 CLAUSE                    STATUS
   -------------------- -------------------- ------------------------- -------
   HR_PROF              XDB_PROTOCOLS                                  DISABLE
   HR_PROF              ALTER SYSTEM                                   DISABLE
   HR_PROF              ALTER SYSTEM         FLUSH SHARED_POOL         ENABLE
   HR_PROF2                                                            EMPTY
   PRIVATE_DBAAS                                                       EMPTY
   PUBLIC_DBAAS                                                        EMPTY
   SAAS                                                                EMPTY
   ```

Dropping a PDB Lockdown Profile

To drop a PDB lockdown profile, you must have the `DROP LOCKDOWN PROFILE` system privilege and be logged into the CDB or application root.

You can find the names of existing PDB lockdown profiles by querying the `DBA_LOCKDOWN_PROFILES` data dictionary view.

1. Connect to the CDB root or the application root as a user who has the `DROP LOCKDOWN PROFILE` system privilege.

   For example, to connect to the CDB root:

   ```
   CONNECT c##sec_admin
   Enter password: password
   ```

2. Run the `DROP LOCKDOWN_PROFILE` statement.

   For example:

   ```
   DROP LOCKDOWN PROFILE hr_prof2;
   ```

3. Optionally, review the current list of profiles by querying `DBA_LOCKDOWN_PROFILES`.

   For example, run the following query:

   ```
   SET LINESIZE 150
   COL PROFILE_NAME FORMAT a20
   COL RULE FORMAT a20
   COL CLAUSE FORMAT a25
   ```

```
SELECT PROFILE_NAME, RULE, CLAUSE, STATUS FROM CDB_LOCKDOWN_PROFILES;
```

Sample output appears below:

```
PROFILE_NAME         RULE                 CLAUSE                     STATUS
-------------------- -------------------- -------------------------- -------
HR_PROF              XDB_PROTOCOLS                                   DISABLE
HR_PROF              ALTER SYSTEM                                    DISABLE
HR_PROF              ALTER SYSTEM         FLUSH SHARED_POOL          ENABLE
PRIVATE_DBAAS                                                        EMPTY
PUBLIC_DBAAS                                                         EMPTY
SAAS                                                                 EMPTY
```

Managing Object Privileges

Object privileges enable you to perform actions on schema objects, such as tables or indexes.

About Object Privileges

An object privilege grants permission to perform a particular action on a specific schema object.

Different object privileges are available for different types of schema objects. The privilege to delete rows from the `departments` table is an example of an object privilege.

Some schema objects, such as clusters, indexes, triggers, and database links, do not have associated object privileges. Their use is controlled with system privileges. For example, to alter a cluster, a user must own the cluster or have the `ALTER ANY CLUSTER` system privilege.

Some examples of object privileges include the right to:

- Use an edition
- Update a table
- Select rows from another user's table
- Execute a stored procedure of another user

See Also:

- How Commonly Granted Object Privileges Work
- *Oracle Database SQL Language Reference* for a list of object privileges and the operations they authorize

Who Can Grant Object Privileges?

A user automatically has all object privileges for schema objects contained in his or her schema.

A user with the `GRANT ANY OBJECT PRIVILEGE` system privilege can grant any specified object privilege to another user with or without the `WITH GRANT OPTION` clause of the

GRANT statement. A user with the GRANT ANY OBJECT PRIVILEGE privilege can also use that privilege to revoke any object privilege that was granted either by the object owner or by some other user with the GRANT ANY OBJECT PRIVILEGE privilege.

If the grantee does not have the GRANT ANY OBJECT PRIVILEGE privilege or had been granted the privilege without the WITH GRANT OPTION clause of the GRANT statement, then this user cannot grant the privilege to other users.

The WITH GRANT OPTION can be used only with object privilege grants to users. It cannot be used for object privilege grants to roles.

See Also:

Oracle Database SQL Language Reference for information about GRANT and GRANT ANY OBJECT PRIVILEGE

Grants and Revokes of Object Privileges

You can grant privileges to or revoke privileges from objects either directly to a user or through roles.

About Granting and Revoking Object Privileges

Object privileges can be granted to and revoked from users and roles.

If you grant object privileges to roles, then you can make the privileges selectively available To grant object privileges, you can use the GRANT statement; to revoke object privileges, you can use the REVOKE statement.

How the ALL Clause Grants or Revokes All Available Object Privileges

Each type of object has different privileges associated with it, which can be controlled by the ALL clause.

You can specify ALL [PRIVILEGES] to grant or revoke all available object privileges for an object. ALL is not a privilege. Rather, it is a shortcut, or a way of granting or revoking all object privileges with one GRANT and REVOKE statement. If all object privileges are granted using the ALL shortcut, then individual privileges can still be revoked.

Similarly, you can revoke all individually granted privileges by specifying ALL. However, if you REVOKE ALL, and revoking causes integrity constraints to be deleted (because they depend on a REFERENCES privilege that you are revoking), then you must include the CASCADE CONSTRAINTS option in the REVOKE statement.

Example 4-4 revokes all privileges on the orders table in the HR schema using CASCADE CONSTRAINTS.

Example 4-4 Revoking All Object Privileges Using CASCADE CONSTRAINTS

```
REVOKE ALL
 ON ORDERS FROM HR
 CASCADE CONSTRAINTS;
```

READ and SELECT Object Privileges

The READ and SELECT privileges provide different layers of query privileges.

About Managing READ and SELECT Object Privileges

You can grant users either the READ or the SELECT object privilege.

The grant of these privileges depend on the level of access that you want to allow the user.

Follow these guidelines:

- If you want the user only to be able to query tables, views, materialized views, or synonyms, then you should grant the READ object privilege. For example:

  ```
  GRANT READ ON HR.EMPLOYEES TO psmith;
  ```

- If you want the user to be able to perform the following actions in addition to performing the query, then you should grant the user the SELECT object privilege:
  - `LOCK TABLE` *`table_name`* `IN EXCLUSIVE MODE;`
  - `SELECT ... FROM` *`table_name`* `FOR UPDATE;`

  For example:

  ```
  GRANT SELECT ON HR.EMPLOYEES TO psmith;
  ```

In either case, user psmith would use a SELECT statement to perform query.

Enabling Users to Use the READ Object Privilege to Query Any Table in the Database

The READ ANY TABLE system privilege provides the READ object privilege for querying any table in the database.

- To enable a user to have the READ object privilege for any table in the database, grant the user the READ ANY TABLE system privilege.

For example:

```
GRANT READ ANY TABLE TO psmith;
```

As with the READ object privilege, the READ ANY TABLE system privilege does not enable users to lock tables in exclusive mode nor select tables for update operations. Conversely, the SELECT ANY TABLE system privilege enables users to lock the rows of a table, or lock the entire table, through a SELECT ... FOR UPDATE statement, in addition to querying any table.

Restrictions on the READ and READ ANY TABLE Privileges

There are special restrictions on the READ and READ ANY TABLE privileges.

These privileges are as follows:

- The READ object privilege has no effect on the requirements of the SQL92_SECURITY standard. If the SQL92_SECURITY initialization parameter has been set to TRUE, then its requirement that users must be granted the SELECT object privilege in addition to

UPDATE or DELETE in order to execute the UPDATE or DELETE statements is not relaxed to require that READ is sufficient instead of SELECT.

- If Oracle Database Vault is enabled, remember that the SQL92_SECURITY initialization parameter is automatically set to TRUE. Hence, UPDATE and DELETE statements will fail if the user has only been granted the READ object privilege or the READ ANY TABLE system privilege. In this case, you must grant the user the SELECT object privilege or, if the user is a trusted user, the SELECT ANY TABLE system privilege.

Object Privilege Use with Synonyms

The CREATE SYNONYM statement create synonyms for database objects.

You can create synonyms for the following objects: tables, views, sequences, operators, procedures, stored functions, packages, materialized views, Java class schema objects, user-defined object types, or other synonyms.

If you grant users the privilege to use the synonym, then the object privileges granted on the underlying objects apply whether the user references the base object by name or by using the synonym.

For example, suppose user OE creates the following synonym for the CUSTOMERS table:

```
CREATE SYNONYM customer_syn FOR CUSTOMERS;
```

Then OE grants the READ privilege on the customer_syn synonym to user HR.

```
GRANT READ ON customer_syn TO HR;
```

User HR then tries either of the following queries:

```
SELECT COUNT(*) FROM OE.customer_syn;

SELECT COUNT(*) FROM OE.CUSTOMERS;
```

Both queries will yield the same result:

```
  COUNT(*)
----------
       319
```

Be aware that when you grant the synonym to another user, the grant applies to the underlying object that the synonym represents, not to the synonym itself. For example, if user HR queries the ALL_TAB_PRIVS data dictionary view for his privileges, he will learn the following:

```
SELECT TABLE_SCHEMA, TABLE_NAME, PRIVILEGE
FROM ALL_TAB_PRIVS
WHERE TABLE_SCHEMA = 'OE';

TABLE_SCHEMA  TABLE_NAME  PRIVILEGE
------------  ----------  ------------------
OE            CUSTOMER    READ
OE            OE          INHERIT PRIVILEGES
```

The results show that in addition to other privileges, he has the READ privilege for the underlying object of the customer_syn synonym, which is the OE.CUSTOMER table.

At this point, if user `OE` then revokes the `READ` privilege on the `customer_syn` synonym from `HR`, here are the results if `HR` checks his privileges again:

```
TABLE_SCHEMA  TABLE_NAME  PRIVILEGE
------------  ----------  ------------------
OE            OE          INHERIT PRIVILEGES
```

User `HR` no longer has the `READ` privilege for the `OE.CUSTOMER` table. If he tries to query the `OE.CUSTOMERS` table, then the following error appears:

```
SELECT COUNT(*) FROM OE.CUSTOMERS;

ERROR at line 1:
ORA-00942: table or view does not exist
```

Sharing Application Common Objects

Database objects can be configured so that their metadata links, data links, and extended data links can be shared in the application root.

See Also:

Oracle Database Administrator's Guide for information about creating application common objects: metadata-linked objects, data-linked objects, and extended data-linked objects

Metadata-Linked Application Common Objects

A metadata link enables database objects in an application pluggable database (PDB) to share metadata with objects in the application root.

Metadata links are useful for reducing disk and memory requirements because they store only one copy of an object's metadata (such as the source code for a PL/SQL package) for identically defined objects (such as Oracle-suppled PL/SQL packages). This improves the performance of upgrade operations because changes to this metadata will be made in one place, the application root.

You must configure the metadata link from the application root. You can use the `DBMS_PDB.SET_MEDATADATA_LINKED` PL/SQL procedure to change the database object to a metadata link.

The following example shows how to use the `DBMS_PDB.SET_METADATA_LINKED` procedure to change the `update_emp_rating` procedure in the `hr_mgr` schema to a metadata-linked application common object.

Example 4-5   Changing an Object to a Metadata-Linked Application Common Object

```
BEGIN
  DBMS_PDB.SET_METADATA_LINKED (
   SCHEMA_NAME => 'hr_mgr',
   OBJECT_NAME => 'update_emp_rating',
   NAMESPACE   => 1);
END;
/
```

Any common user can own metadata links. Metadata links can only be used to share the metadata of application common objects that their creator in the application root owns.

To find if an object has a metadata link, query the `SHARING` column of the `DBA_OBJECTS` data dictionary view.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_PDB.SET_METADATA_LINKED` procedure

Data-Linked Application Common Objects

Data links manage references and privileges for objects in a multitenant environment.

A data link (previously called an object link) enables references to, and privilege grants on, objects in an application root from an application pluggable database (PDB) that belong to the same application container.

If an application common user who owns an application common object wants to grant access to that object to a user in a PDB, then the application common user can accomplish this by granting the privilege on a data link that points to the common object. For example, you can create data links for objects such as tables, views, clusters, sequences, or PL/SQL packages if you want to ensure that an operation on the object (such as a query, a DML, an `EXECUTE` statement, and so on) that refers to this operation affects the same object regardless of the container in which the operation is performed.

You must configure the data link from an application root. You can use the `DBMS_PDB.SET_DATA_LINKED` PL/SQL procedure to change the data link. You should use this procedure only when you want to convert an existing object to become data linked.

The following example shows how to use the `DBMS_PDB.SET_DATA_LINKED` procedure to change the `emp_ratings` table in the `hr_mgr` schema to a data-linked application common object.

Example 4-6 Changing an Object to a Data-Linked Application Common Object

```
BEGIN
  DBMS_PDB.SET_DATA_LINKED (
   SCHEMA_NAME => 'hr_mgr',
   OBJECT_NAME => 'emp_ratings',
   NAMESPACE   => 1);
END;
/
```

Any common user can own data links.

To find if an object has an data link, query the `SHARING` column of the `DBA_OBJECTS` data dictionary view. The `NAMESPACE` column of this view provides the namespace number.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_PDB.SET_DATA_LINKED` procedure

Extended Data-Linked Application Common Objects

Extended data links can combine data from an application pluggable database (PDB) with an application root.

An extended data link enables a data link to combine data found in a table in the PDB with data from a corresponding table in the application root.

You can think of an extended data link as a hybrid of a metadata link and a data link. An extended data-link object in an application PDB inherits metadata from the extended data link object in the application root. The data for the object is stored in the application root and, optionally, in each application PDB. You can create extended data links for tables and views only. When you query the `DBA_OBJECTS` data dictionary view for an extended data link object, this view returns extended data link-related rows from both the application PDB and the application root.

You must configure the extended data link from an application root. You can use the `DBMS_PDB.SET_EXT_DATA_LINKED` PL/SQL procedure to change the database object to an extended data link.

The following example shows how to use the `DBMS_PDB.SET_EXT_DATA_LINKED` procedure to change the `emp_salaries` data dictionary view in the `hr_mgr` schema to an extended data-linked application common object.

Example 4-7 Changing an Object to an Extended Data-Linked Application Common Object

```
BEGIN
  DBMS_PDB.SET_EXT_DATA_LINKED (
   SCHEMA_NAME => 'hr_mgr',
   OBJECT_NAME => 'emp_salaries',
   NAMESPACE   => 1);
END;
/
```

Any common user can own extended data links.

To find if an object has an extended data link, query the `SHARING` column of the `DBA_OBJECTS` data dictionary view.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_PDB.SET_EXT_DATA_LINKED` procedure

Table Privileges

Object privileges for tables enable table security at the DML or DDL level of operation.

How Table Privileges Affect Data Manipulation Language Operations

You can grant privileges to use the `DELETE`, `INSERT`, `SELECT`, and `UPDATE` DML operations on tables and views.

Grant these privileges only to users and roles that need to query or manipulate data in a table.

You can restrict `INSERT` and `UPDATE` privileges for a table to specific columns of the table. With a selective `INSERT` privilege, a privileged user can insert a row with values for the selected columns. All other columns receive `NULL` or the default value of the column. With a selective `UPDATE` privilege, a user can update only specific column values of a row. You can use selective `INSERT` and `UPDATE` privileges to restrict user access to sensitive data.

For example, if you do not want data entry users to alter the `salary` column of the `employees` table, then selective `INSERT` or `UPDATE` privileges can be granted that exclude the `salary` column. Alternatively, a view that excludes the `salary` column could satisfy this need for additional security.

See Also:

Oracle Database SQL Language Reference for more information about DML operations

How Table Privileges Affect Data Definition Language Operations

The `ALTER`, `INDEX`, and `REFERENCES` privileges allow DDL operations to be performed on a table.

Because these privileges allow other users to alter or create dependencies on a table, you should grant these privileges conservatively. A user attempting to perform a DDL operation on a table may need additional system or object privileges. For example, to create a trigger on a table, the user requires both the `ALTER TABLE` object privilege for the table and the `CREATE TRIGGER` system privilege.

As with the `INSERT` and `UPDATE` privileges, you can grant the `REFERENCES` privilege on specific columns of a table. The `REFERENCES` privilege enables the grantee to use the table on which the grant is made as a parent key to any foreign keys that the grantee wishes to create in his or her own tables. This action is controlled with a special privilege because the presence of foreign keys restricts the data manipulation and table alterations that can be done to the parent key. A column-specific `REFERENCES` privilege restricts the grantee to using the named columns (which, of course, must include at least one primary or unique key of the parent table).

See Also:

Oracle Database Concepts for more information about how data integrity works with primary keys, unique keys, and integrity constraints

View Privileges

You can apply DML object privileges to views, similar to tables.

Privileges Required to Create Views

To create a view, you must have specific privileges.

Object privileges for a view allow various DML operations, which affect the base tables from which the view is derived.

These privileges to create a view are as follows:

- You must be granted one of the following system privileges, either explicitly or through a role:
  - The `CREATE VIEW` system privilege (to create a view in your schema)
  - The `CREATE ANY VIEW` system privilege (to create a view in the schema of another user)
- You must be explicitly granted one of the following privileges:
  - The `SELECT`, `INSERT`, `UPDATE`, or `DELETE` object privileges on all base objects underlying the view
  - The `SELECT ANY TABLE`, `INSERT ANY TABLE`, `UPDATE ANY TABLE`, or `DELETE ANY TABLE` system privileges
- In addition, before you can grant other users access to you view, you must have object privileges to the base objects with the `GRANT OPTION` clause or appropriate system privileges with the `ADMIN OPTION` clause. If you do not have these privileges, then you cannot to grant other users access to your view. If you try, an `ORA-01720: grant option does not exist for` *`object_name`* error is raised, with *`object_name`* referring to the view's underlying object for which you do not have the sufficient privilege.

See Also:

Oracle Database SQL Language Reference

The Use of Views to Increase Table Security

Database views can increase table security by restricting the data that users can see.

To use a view, the user must have the appropriate privileges but only for the view itself, not its underlying objects. However, if access privileges for the underlying objects of the view are removed, then the user no longer has access.

This behavior occurs because the security domain that is used when a user queries the view is that of the definer of the view. If the privileges on the underlying objects are revoked from the view's definer, then the view becomes invalid, and no one can use the view. Therefore, even if a user has been granted access to the view, the user may not be able to use the view if the definer's rights have been revoked from the view's underlying objects.

For example, suppose User A creates a view. User A has definer's rights on the underlying objects of the view. User A then grants the `SELECT` privilege on that view to User B so that User B can query the view. But if User A no longer has access to the underlying objects of that view, then User B no longer has access either.

Views add two more levels of security for tables, column-level security and value-based security, as follows:

- **A view can provide access to selected columns of base tables.** For example, you can define a view on the `employees` table to show only the `employee_id`, `last_name`, and `manager_id` columns:

  ```
  CREATE VIEW employees_manager AS
      SELECT last_name, employee_id, manager_id FROM employees;
  ```

- **A view can provide value-based security for the information in a table.** A `WHERE` clause in the definition of a view displays only selected rows of base tables. Consider the following two examples:

  ```
  CREATE VIEW lowsal AS
      SELECT * FROM employees
      WHERE salary < 10000;
  ```

  The `lowsal` view allows access to all rows of the `employees` table that have a salary value less than 10000. Notice that all columns of the `employees` table are accessible in the `lowsal` view.

  ```
  CREATE VIEW own_salary AS
      SELECT last_name, salary
      FROM employees
      WHERE last_name = USER;
  ```

  In the `own_salary` view, only the rows with an `last_name` that matches the current user of the view are accessible. The `own_salary` view uses the `user` pseudo column, whose values always refer to the current user. This view combines both column-level security and value-based security.

Procedure Privileges

The `EXECUTE` privilege enables users to run procedures and functions, either standalone or in packages.

The Use of the EXECUTE Privilege for Procedure Privileges

The EXECUTE privilege is a very powerful privilege that should be handled with caution.

The `EXECUTE` privilege is the only **object privilege** for procedures, including standalone procedures and functions, and for those within packages.

You should grant this privilege only to users who must run a procedure or compile another procedure that calls a desired procedure. You can find the privileges that a user has been granted by querying the `DBA_SYS_PRIVS` data dictionary view.

Procedure Execution and Security Domains

The `EXECUTE` object privilege for a procedure can be used to execute a procedure or compile a program unit that references the procedure.

Oracle Database performs a run-time privilege check when any PL/SQL unit is called. A user with the `EXECUTE ANY PROCEDURE` system privilege can execute any procedure in the database. Privileges to run procedures can be granted to a user through roles.

See Also:

- About Definer's Rights and Invoker's Rights
- *Oracle Database PL/SQL Packages and Types Reference* for more information about how Oracle Database checks privileges at run-time

System Privileges Required to Create or Replace a Procedure

You must have specific privileges to create or replace a procedure in your own schema or in another user’s schema.

To create or replace a procedure in your own schema, you must have the `CREATE PROCEDURE` system privilege. To create or replace a procedure in another user's schema, you must have the `CREATE ANY PROCEDURE` system privilege.

The user who owns the procedure also must have privileges for schema objects referenced in the procedure body. To create a procedure, you need to have been explicitly granted the necessary privileges (system or object) on all objects referenced by the procedure. You cannot obtain the required privileges through roles. This includes the `EXECUTE` privilege for any procedures that are called inside the procedure being created.

Note:

Triggers require that privileges on referenced objects be granted directly to the owner of the trigger. Anonymous PL/SQL blocks can use any privilege, whether the privilege is granted explicitly or through a role.

System Privileges Required to Compile a Procedure

You must have specific privileges to compile both standalone procedures and procedures that are part of a package.

To compile a standalone procedure, you should run the `ALTER PROCEDURE` statement with the `COMPILE` clause. To compile a procedure that is part of a package, you should run the `ALTER PACKAGE` statement.

The following example shows how to compile a standalone procedure.

```
ALTER PROCEDURE psmith.remove_emp COMPILE;
```

If the standalone or packaged procedure is in another user's schema, you must have the `ALTER ANY PROCEDURE` privilege to recompile it. You can recompile procedures in your own schema without any privileges.

How Procedure Privileges Affect Packages and Package Objects

The powerful `EXECUTE` privilege enables users to run any public procedures or functions within a package.

About the Effect of Procedure Privileges on Packages and Package Objects

The `EXECUTE` object privilege for a package applies to any procedure or function within this package.

A user with the`EXECUTE` object privilege for a package can execute any public procedure or function in the package, and can access or modify the value of any public package variable.

You cannot grant specific `EXECUTE` privileges for individual constructs in a package. Therefore, you may find it useful to consider two alternatives for establishing security when developing procedures, functions, and packages for a database application. The following examples describe these alternatives.

Example: Procedure Privileges Used in One Package

The `CREATE PACKAGE BODY` statement can create a package body that contains procedures to manage procedure privileges used in one package.

Example 4-8 shows four procedures created in the bodies of two packages.

Example 4-8   Procedure Privileges Used in One Packagee

```
CREATE PACKAGE BODY hire_fire AS
  PROCEDURE hire(...) IS
    BEGIN
      INSERT INTO employees . . .
    END hire;
  PROCEDURE fire(...) IS
    BEGIN
      DELETE FROM employees . . .
    END fire;
END hire_fire;

CREATE PACKAGE BODY raise_bonus AS
  PROCEDURE give_raise(...) IS
    BEGIN
      UPDATE employees SET salary = . . .
    END give_raise;
  PROCEDURE give_bonus(...) IS
    BEGIN
      UPDATE employees SET bonus = . . .
    END give_bonus;
END raise_bonus;
```

The following GRANT EXECUTE statements enable the big_bosses and little_bosses roles to run the appropriate procedures:

```
GRANT EXECUTE ON hire_fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

Example: Procedure Privileges and Package Objects

The CREATE PACKAGE BODY statement can create a package body containing procedure definitions to manage procedure privileges and package objects.

Example 4-9 shows four procedure definitions within the body of a single package. Two additional standalone procedures and a package are created specifically to provide access to the procedures defined in the main package.

Example 4-9 Procedure Privileges and Package Objects

```
CREATE PACKAGE BODY employee_changes AS
  PROCEDURE change_salary(...) IS BEGIN ... END;
  PROCEDURE change_bonus(...) IS BEGIN ... END;
  PROCEDURE insert_employee(...) IS BEGIN ... END;
  PROCEDURE delete_employee(...) IS BEGIN ... END;
END employee_changes;

CREATE PROCEDURE hire
  BEGIN
    employee_changes.insert_employee(...)
  END hire;

CREATE PROCEDURE fire
  BEGIN
    employee_changes.delete_employee(...)
  END fire;

PACKAGE raise_bonus IS
  PROCEDURE give_raise(...) AS
    BEGIN
      employee_changes.change_salary(...)
    END give_raise;

  PROCEDURE give_bonus(...)
    BEGIN
      employee_changes.change_bonus(...)
    END give_bonus;
```

Using this method, the procedures that actually do the work (the procedures in the employee_changes package) are defined in a single package and can share declared global variables, cursors, on so on. By declaring top-level procedures, hire and fire, and an additional package, raise_bonus, you can grant selective EXECUTE privileges on procedures in the main package:

```
GRANT EXECUTE ON hire, fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

Be aware that granting EXECUTE privilege for a package provides uniform access to all package objects.

Type Privileges

You can control system and object privileges for types, methods, and objects.

System Privileges for Named Types

System privileges for named types can enable users to perform actions such as creating named types in their own schemas.

Table 4-4 lists system privileges for named types (object types, `VARRAY`s, and nested tables).

Table 4-4 System Privileges for Named Types

Privilege	Enables you to ...
`CREATE TYPE`	Create named types in your own schemas
`CREATE ANY TYPE`	Create a named type in any schema
`ALTER ANY TYPE`	Alter a named type in any schema
`DROP ANY TYPE`	Drop a named type in any schema
`EXECUTE ANY TYPE`	Use and reference a named type in any schema

The `RESOURCE` role includes the `CREATE TYPE` system privilege. The `DBA` role includes all of these privileges.

Object Privileges for Named Types

The only object privilege that applies to named types is `EXECUTE`.

If the `EXECUTE` privilege exists on a named type, then a user can use the named type to:

- Define a table
- Define a column in a relational table
- Declare a variable or parameter of the named type

The `EXECUTE` privilege permits a user to invoke the methods in the type, including the type constructor. This is similar to the `EXECUTE` privilege on a stored PL/SQL procedure.

Method Execution Model for Named Types

The method execution for named types is the same as any other stored PL/SQL procedure.

Users must be granted the appropriate privileges for using the named types, such as the `EXECUTE` privilege. As with all privilege grants, only grant these privileges to trusted users. You can find the privileges that a user has been granted by querying the `DBA_SYS_PRIVS` data dictionary view.

Privileges Required to Create Types and Tables Using Types

To create a type, you must have the appropriate privileges.

These privileges are as follows:

- You must have the `CREATE TYPE` system privilege to create a type in your schema or the `CREATE ANY TYPE` system privilege to create a type in the schema of another user. These privileges can be acquired explicitly or through a role.
- The owner of the type must be explicitly granted the `EXECUTE` object privileges to access all other types referenced within the definition of the type, or have been granted the `EXECUTE ANY TYPE` system privilege. The owner cannot obtain the required privileges through roles.
- If the type owner intends to grant access to the type to other users, then the owner must receive the `EXECUTE` privileges to the referenced types with the `GRANT OPTION` or the `EXECUTE ANY TYPE` system privilege with the `ADMIN OPTION`. If not, then the type owner has insufficient privileges to grant access on the type to other users.

To create a table using types, you must meet the requirements for creating a table and the following additional requirements:

- The owner of the table must have been directly granted the `EXECUTE` object privilege to access all types referenced by the table, or has been granted the `EXECUTE ANY TYPE` system privilege. The owner cannot exercise the required privileges if these privileges were granted through roles.
- If the table owner intends to grant access to the table to other users, then the owner must have the `EXECUTE` privilege to the referenced types with the `GRANT OPTION` or the `EXECUTE ANY TYPE` system privilege with the `ADMIN OPTION`. If not, then the table owner has insufficient privileges to grant access on the table.

Example: Privileges for Creating Types and Tables Using Types

The `EXECUTE` privilege with the `GRANT OPTION` is required for users to grant the `EXECUTE` privilege on a type to other users.

Assume that three users exist with the `CONNECT` and `RESOURCE` roles:

- `user1`
- `user2`
- `user3`

The following DDL is run in the schema of `user1`:

```
CREATE TYPE type1 AS OBJECT (
  attr1 NUMBER);

CREATE TYPE type2 AS OBJECT (
  attr2 NUMBER);

GRANT EXECUTE ON type1 TO user2;
GRANT EXECUTE ON type2 TO user2 WITH GRANT OPTION;
```

The following DDL is performed in the schema of `user2`:

```
CREATE TABLE tab1 OF user1.type1;
CREATE TYPE type3 AS OBJECT (
  attr3 user1.type2);
CREATE TABLE tab2 (
  col1 user1.type2);
```

The following statements succeed because `user2` has `EXECUTE` privilege on `user1.type2` with the `GRANT OPTION`:

```
GRANT EXECUTE ON type3 TO user3;
GRANT SELECT ON tab2 TO user3;
```

However, the following grant fails because `user2` does not have `EXECUTE` privilege on `user1.type1` with the `GRANT OPTION`:

```
GRANT SELECT ON tab1 TO user3;
```

The following statements can be successfully run by `user3`:

```
CREATE TYPE type4 AS OBJECT (
  attr4 user2.type3);
CREATE TABLE tab3 OF type4;
```

Note:

The `CONNECT` role presently retains only the `CREATE SESSION` and `SET CONTAINER` privileges.

Privileges on Type Access and Object Access

Existing column-level and table-level privileges for DML statements apply to both column objects and row objects.

Table 4-5 lists the privileges for object tables.

Table 4-5 Privileges for Object Tables

Privilege	Enables you to...
`SELECT`	Access an object and its attributes from the table
`UPDATE`	Modify the attributes of the objects that make up the rows in the table
`INSERT`	Create new objects in the table
`DELETE`	Delete rows

Similar table privileges and column privileges apply to column objects. Retrieving instances does not in itself reveal type information. However, clients must access named type information to interpret the type instance images. When a client requests type information, Oracle Database checks for the `EXECUTE` privilege on the type.

Consider the following schema:

```
CREATE TYPE emp_type (
    eno NUMBER, ename CHAR(31), eaddr addr_t);
CREATE TABLE emp OF emp_t;
```

In addition, consider the following two queries:

```
SELECT VALUE(emp) FROM emp;
SELECT eno, ename FROM emp;
```

For either query, Oracle Database checks the `SELECT` privilege of the user for the `emp` table. For the first query, the user must obtain the `emp_type` type information to interpret the data. When the query accesses the `emp_type` type, Oracle Database checks the `EXECUTE` privilege of the user.

The second query, however, does not involve named types, so Oracle Database does not check type privileges.

In addition, by using the schema from the previous section, `user3` can perform the following queries:

```
SELECT tab1.col1.attr2 FROM user2.tab1 tab1;
SELECT attr4.attr3.attr2 FROM tab3;
```

Note that in both `SELECT` statements, `user3` does not have explicit privileges on the underlying types, but the statement succeeds because the type and table owners have the necessary privileges with the `GRANT OPTION`.

Oracle Database checks privileges on the following events, and returns an error if the client does not have the privilege for the action:

- Pinning an object in the object cache using its `REF` value causes Oracle Database to check for the `SELECT` privilege on the containing object table.
- Modifying an existing object or flushing an object from the object cache causes Oracle Database to check for the `UPDATE` privilege on the destination object table.
- Flushing a new object causes Oracle Database to check for the `INSERT` privilege on the destination object table.
- Deleting an object causes Oracle Database to check for the `DELETE` privilege on the destination table.
- Pinning an object of a named type causes Oracle Database to check `EXECUTE` privilege on the object.

Modifying the attributes of an object in a client third-generation language application causes Oracle Database to update the entire object. Therefore, the user needs the `UPDATE` privilege on the object table. Having the `UPDATE` privilege on only certain columns of the object table is not sufficient, even if the application only modifies attributes corresponding to those columns. Therefore, Oracle Database does not support column-level privileges for object tables.

Type Dependencies

As with stored objects, such as procedures and tables, types that are referenced by other objects are called dependencies.

There are some special issues for types on which tables depend. Because a table contains data that relies on the type definition for access, any change to the type causes all stored data to become inaccessible. Changes that can cause this are when necessary privileges required to use the type are revoked, or the type or dependent types are dropped. If these actions occur, then the table becomes invalid and cannot be accessed.

A table that is invalid because of missing privileges can automatically become valid and accessible if the required privileges are granted again. A table that is invalid because a dependent type was dropped can never be accessed again, and the only permissible action is to drop the table.

Because of the severe effects that revoking a privilege on a type or dropping a type can cause, the SQL statements `REVOKE` and `DROP TYPE` , by default, implement restricted semantics. This means that if the named type in either statement has table or type dependents, then an error is received and the statement cancels. However, if the `FORCE` clause for either statement is used, then the statement always succeeds. If there are depended-upon tables, then they are invalidated.

See Also:

Oracle Database SQL Language Reference for details about using the `REVOKE` and `DROP TYPE` SQL statements

Grants of User Privileges and Roles

The `GRANT` statement provides privileges for a user to perform specific actions, such as executing a procedure.

Granting System Privileges and Roles to Users and Roles

Before you grant system privileges and roles to users and roles, be aware of how privileges for these types of grants work.

Privileges for Grants of System Privileges and Roles to Users and Roles

You can use the `GRANT` SQL statement to grant system privileges and roles to users and roles.

The following privileges are required:

- To grant a system privilege, a user must be granted the system privilege with the `ADMIN` option or must be granted the `GRANT ANY PRIVILEGE` system privilege.
- To grant a role, a user must be granted the role with the `ADMIN` option or was granted the `GRANT ANY ROLE` system privilege.

Note:

Object privileges cannot be granted along with system privileges and roles in the same `GRANT` statement.

Example: Granting a System Privilege and a Role to a User

You can use the `GRANT` statement to grant system privileges and roles to users.

Example 4-10 grants the system privilege `CREATE SESSION` and the `accts_pay` role to the user `jward`.

Example 4-10 Granting a System Privilege and a Role to a User

```
GRANT CREATE SESSION, accts_pay TO jward;
```

Example: Granting the EXECUTE Privilege on a Directory Object

You can use the `GRANT` statement to grant the `EXECUTE` privilege on a directory object.

Example 4-10 grants the `EXECUTE` privilege on the `exec_dir` directory object to the user `jward`.

Example 4-11 Granting the EXECUTE Privilege on a Directory Object

```
GRANT EXECUTE ON DIRECTORY exec_dir TO jward;
```

Use of the ADMIN Option to Enable Grantee Users to Grant the Privilege

The `WITH ADMIN OPTION` clause can be used to expand the capabilities of a privilege grant.

These capabilities are as follows:

- The grantee can grant or revoke the system privilege or role to or from any other user or role in the database. Users cannot revoke a role from themselves.
- The grantee can grant the system privilege or role with the `ADMIN` option.
- The grantee of a role can alter or drop the role.

Example 4-12 grants the `new_dba` role with the `WITH ADMIN OPTION` clause to user `michael`.

Example 4-12 Granting the ADMIN Option

```
GRANT new_dba TO michael WITH ADMIN OPTION;
```

User `michael` is able to not only use all of the privileges implicit in the `new_dba` role, but he can also grant, revoke, and drop the `new_dba` role as deemed necessary. Because of these powerful capabilities, use caution when granting system privileges or roles with the `ADMIN` option. These privileges are usually reserved for a security administrator, and are rarely granted to other administrators or users of the system. Be aware that when a user creates a role, the role is automatically granted to the creator with the `ADMIN` option.

Creating a New User with the GRANT Statement

You can create a new user and grant this user a privilege in one `GRANT` SQL statement.

In most cases, you will want to grant the user the `CREATE SESSION` privilege.

- To create a new user with the `GRANT` statement, include the privilege and the `IDENTIFIED BY` clause.

For example, to create user `psmith` as a new user while granting `psmith` the `CREATE SESSION` system privilege:

```
GRANT CREATE SESSION TO psmith IDENTIFIED BY password;
```

If you specify a password using the `IDENTIFIED BY` clause, and the user name does not exist in the database, then a new user with that user name and password is created.

Granting Object Privileges to Users and Roles

You can grant object privileges to users and roles, and enable the grantee to grant the privilege to other users.

About Granting Object Privileges to Users and Roles

You can use the `GRANT` statement to grant object privileges to roles and users.

To grant an object privilege, you must fulfill one of the following conditions:

- You own the object specified.
- You have been granted the `GRANT ANY OBJECT PRIVILEGE` system privilege. This privilege enables you to grant and revoke privileges on behalf of the object owner.
- The `WITH GRANT OPTION` clause was specified when you were granted the object privilege.

> **Note:**
>
> System privileges and roles cannot be granted along with object privileges in the same `GRANT` statement.

The following example grants the `READ`, `INSERT`, and `DELETE` object privileges for all columns of the `emp` table to the users `jfee` and `tsmith`.

```
GRANT READ, INSERT, DELETE ON emp TO jfee, tsmith;
```

To grant all object privileges on the `salary` view to user `jfee`, use the `ALL` keyword as shown in the following example:

```
GRANT ALL ON salary TO jfee;
```

> **Note:**
>
> A grantee cannot regrant access to objects unless the original grant included the `GRANT OPTION`. Thus in the example just given, `jfee` cannot use the `GRANT` statement to grant object privileges to anyone else.

How the WITH GRANT OPTION Clause Works

The `WITH GRANT OPTION` clause with the `GRANT` statement can enable a grantee to grant object privileges to other users.

The user whose schema contains an object is automatically granted all associated object privileges with the `WITH GRANT OPTION` clause. This special privilege allows the grantee several expanded privileges:

- The grantee can grant the object privilege to any user in the database, with or without the `GRANT OPTION`, and to any role in the database.

- If both of the following conditions are true, then the grantee can create views on the table, and grant the corresponding privileges on the views to any user or role in the database:
  - The grantee receives object privileges for the table with the `GRANT OPTION`.
  - The grantee has the `CREATE VIEW` or `CREATE ANY VIEW` system privilege.

Note:

The `WITH GRANT OPTION` clause is not valid if you try to grant an object privilege to a role. Oracle Database prevents the propagation of object privileges through roles so that grantees of a role cannot propagate object privileges received by means of roles.

Grants of Object Privileges on Behalf of the Object Owner

The `GRANT ANY OBJECT PRIVILEGE` system privilege enables users to grant and revoke any object privilege on behalf of the object owner.

This privilege provides a convenient means for database and application administrators to grant access to objects in any schema without requiring that they connect to the schema. Login credentials do not need to be maintained for schema owners who have this privilege, which reduces the number of connections required during configuration.

This system privilege is part of the Oracle Database supplied `DBA` role and is thus granted (with the `ADMIN option`) to any user connecting `AS SYSDBA` (user `SYS`). As with other system privileges, the `GRANT ANY OBJECT PRIVILEGE` system privilege can only be granted by a user who possesses the `ADMIN option`.

The *recorded* grantor of access rights to an object is either the object owner or the person exercising the `GRANT ANY OBJECT PRIVILEGE` system privilege. If the grantor with `GRANT ANY OBJECT PRIVILEGE` does *not* have the object privilege with the `GRANT OPTION`, then the object owner is shown as the grantor. Otherwise, when that grantor has the object privilege with the `GRANT OPTION`, then that grantor is recorded as the grantor of the grant.

Note:

The audit record generated by the `GRANT` statement always shows the actual user who performed the grant.

For example, consider the following scenario. User `adams` possesses the `GRANT ANY OBJECT PRIVILEGE` system privilege. He does not possess any other grant privileges. He issues the following statement:

```
GRANT SELECT ON HR.EMPLOYEES TO blake WITH GRANT OPTION;
```

If you examine the `DBA_TAB_PRIVS` view, then you will see that `hr` is shown as the grantor of the privilege:

```
SELECT GRANTEE, GRANTOR, PRIVILEGE, GRANTABLE
  FROM DBA_TAB_PRIVS
  WHERE TABLE_NAME = 'EMPLOYEES' and OWNER = 'HR';

GRANTEE  GRANTOR PRIVILEGE    GRANTABLE
-------- ------- -----------  ----------
BLAKE    HR       SELECT      YES
```

Now assume that user `blake` also has the `GRANT ANY OBJECT PRIVILEGE` system. He issues the following statement:

```
GRANT SELECT ON HR.EMPLOYEES TO clark;
```

In this case, when you query the `DBA_TAB_PRIVS` view again, you see that `blake` is shown as being the grantor of the privilege:

```
GRANTEE  GRANTOR  PRIVILEGE  GRANTABLE
-------- -------- ---------  ----------
BLAKE    HR       SELECT     YES
CLARK    BLAKE    SELECT     NO
```

This occurs because `blake` already possesses the `SELECT` privilege on `HR.EMPLOYEES` with the `GRANT OPTION`.

Grants of Privileges on Columns

You can grant `INSERT`, `UPDATE`, or `REFERENCES` privileges on individual columns in a table.

Note:

Before granting a column-specific `INSERT` privilege, determine if the table contains any columns on which `NOT NULL` constraints are defined. Granting selective insert capability without including the `NOT NULL` columns prevents the user from inserting any rows into the table. To avoid this situation, ensure that each `NOT NULL` column can either be inserted into or has a non-`NULL` default value. Otherwise, the grantee will not be able to insert rows into the table and will receive an error.

The following statement grants the `INSERT` privilege on the `acct_no` column of the `accounts` table to user `psmith`:

```
GRANT INSERT (acct_no) ON accounts TO psmith;
```

In the following example, object privilege for the `ename` and `job` columns of the `emp` table are granted to the users `jfee` and `tsmith`:

```
GRANT INSERT(ename, job) ON emp TO jfee, tsmith;
```

Row-Level Access Control

You can provide access control at the row level, that is, within objects, but not with the `GRANT` statement.

To perform this kind of access control, you must use either Oracle Virtual Private Database (VPD) or Oracle Label Security (OLS).

See Also:

- Using Oracle Virtual Private Database to Control Data Access
- Policies for Column-Level Oracle Virtual Private Database
- *Oracle Label Security Administrator’s Guide*

Revokes of Privileges and Roles from a User

When you revoke system or object privileges, be aware of the cascading effects of revoking a privilege.

Revokes of System Privileges and Roles

The `REVOKE` SQL statement revokes system privileges and roles.

Any user with the `ADMIN` option for a system privilege or role can revoke the privilege or role from any other database user or role. The revoker does not have to be the user that originally granted the privilege or role. Users with `GRANT ANY ROLE` can revoke *any* role.

Example 4-13 revokes the `CREATE TABLE` system privilege and the `accts_rec` role from user `psmith`:

Example 4-13 Revoking a System Privilege and a Role from a User

```
REVOKE CREATE TABLE, accts_rec FROM psmith;
```

Be aware that the `ADMIN` option for a system privilege or role cannot be selectively revoked. Instead, revoke the privilege or role, and then grant the privilege or role again but without the `ADMIN` option.

Revokes of Object Privileges

You can revoke multiple object privileges, object privileges on behalf of an object owner, column-selective object privileges, and the `REFERENCES` object privilege.

About Revokes of Object Privileges

To revoke an object privilege, you must meet the appropriate requirements.

The requirements are either of the following conditions:

- You previously granted the object privilege to the user or role.
- You possess the `GRANT ANY OBJECT PRIVILEGE` system privilege that enables you to grant and revoke privileges on behalf of the object owner.

You can only revoke the privileges that you, the person who granted the privilege, directly authorized. You cannot revoke grants that were made by other users to whom you granted the `GRANT OPTION`. However, there is a cascading effect. If the object privileges of the user who granted the privilege are revoked, then the object privilege grants that were propagated using the `GRANT OPTION` are revoked as well.

Revokes of Multiple Object Privileges

The REVOKE statement can revoke multiple privileges on one object.

Assuming you are the original grantor of the privilege, the following statement revokes the SELECT and INSERT privileges on the emp table from users jfee and psmith:

```
REVOKE SELECT, INSERT ON emp FROM jfee, psmith;
```

The following statement revokes all object privileges for the dept table that you originally granted to the human_resource role:

```
REVOKE ALL ON dept FROM human_resources;
```

Note:

The GRANT OPTION for an object privilege cannot be selectively revoked. Instead, revoke the object privilege and then grant it again but without the GRANT OPTION. Users cannot revoke object privileges from themselves.

Revokes of Object Privileges on Behalf of the Object Owner

The GRANT ANY OBJECT PRIVILEGE system privilege can be used to revoke any object privilege where the object owner is the grantor.

This occurs when the object privilege is granted by the object owner, or on behalf of the owner by any user holding the GRANT ANY OBJECT PRIVILEGE system privilege.

In a situation where the object privilege was granted by both the owner of the object and the user executing the REVOKE statement (who has both the specific object privilege and the GRANT ANY OBJECT PRIVILEGE system privilege), Oracle Database only revokes the object privilege granted by the user issuing the REVOKE statement. This can be illustrated by continuing the example started in Grants of Object Privileges on Behalf of the Object Owner.

At this point, user blake granted the SELECT privilege on HR.EMPLOYEES to clark. Even though blake possesses the GRANT ANY OBJECT PRIVILEGE system privilege, he also holds the specific object privilege, thus this grant is attributed to him. Assume that user HR also grants the SELECT privilege on HR.EMPLOYEES to user clark. A query of the DBA_TAB_PRIVS view shows that the following grants are in effect for the HR.EMPLOYEES table:

```
GRANTEE  GRANTOR PRIVILEGE    GRANTABLE
-------- ------- -----------  ----------
BLAKE    HR       SELECT       YES
CLARK    BLAKE    SELECT       NO
CLARK    HR       SELECT       NO
```

User blake now issues the following REVOKE statement:

```
REVOKE  SELECT ON HR.EMPLOYEES FROM clark;
```

Only the object privilege for user clark granted by user blake is removed. The grant by the object owner, HR, remains.

```
GRANTEE  GRANTOR PRIVILEGE    GRANTABLE
-------- ------- -----------  ----------
BLAKE    HR       SELECT      YES
CLARK    HR       SELECT      NO
```

If `blake` issues the `REVOKE` statement again, then this time the effect is to remove the object privilege granted by `adams` (on behalf of `HR`), using the `GRANT ANY OBEJCT PRIVILEGE` system privilege.

Revokes of Column-Selective Object Privileges

`GRANT` and `REVOKE` operations for column-specific operations have different privileges and restrictions.

Although users can grant column-specific `INSERT`, `UPDATE`, and `REFERENCES` privileges for tables and views, they cannot selectively revoke column-specific privileges with a similar `REVOKE` statement. Instead, the grantor must first revoke the object privilege for all columns of a table or view, and then selectively repeat the grant of the column-specific privileges that the grantor intends to keep in effect.

For example, assume that role `human_resources` was granted the `UPDATE` privilege on the `deptno` and `dname` columns of the table `dept`. To revoke the `UPDATE` privilege on just the `deptno` column, issue the following two statements:

```
REVOKE UPDATE ON dept FROM human_resources;
GRANT UPDATE (dname) ON dept TO human_resources;
```

The `REVOKE` statement revokes the `UPDATE` privilege on all columns of the `dept` table from the role `human_resources`. The `GRANT` statement then repeats, restores, or reissues the grant of the `UPDATE` privilege on the `dname` column to the role `human_resources`.

Revokes of the REFERENCES Object Privilege

When you revoke the `REFERENCES` object privilege, it affects foreign key constraints.

If the grantee of the `REFERENCES` object privilege has used the privilege to create a foreign key constraint (that currently exists), then the grantor can revoke the privilege only by specifying the `CASCADE CONSTRAINTS` option in the `REVOKE` statement.

For example:

```
REVOKE REFERENCES ON dept FROM jward CASCADE CONSTRAINTS;
```

Any foreign key constraints currently defined that use the revoked `REFERENCES` privilege are dropped when the `CASCADE CONSTRAINTS` clause is specified.

Cascading Effects of Revoking Privileges

There are no cascading effects for revoked object privileges related to DDL operations, but there are cascading effects for object privilege revocations.

Cascading Effects When Revoking System Privileges

There are no cascading effects when you revoke a system privilege that is related to DDL operations.

This applies regardless of whether the privilege was granted with or without the `ADMIN` option.

For example, assume the following:

1. The security administrator grants the `CREATE TABLE` system privilege to user `jfee` with the `ADMIN option`.
2. User `jfee` creates a table.
3. User `jfee` grants the `CREATE TABLE` system privilege to user `tsmith`.
4. User `tsmith` creates a table.
5. The security administrator revokes the `CREATE TABLE` system privilege from user `jfee`.
6. The table created by user `jfee` continues to exist. User `tsmith` still has the table and the `CREATE TABLE` system privilege.

You can observe cascading effects when you revoke a system privilege related to a DML operation. If the `SELECT ANY TABLE` privilege is revoked from a user, then all procedures contained in the user's schema relying on this privilege can no longer be executed successfully until the privilege is reauthorized.

Cascading Effects When Revoking Object Privileges

Revoking an object privilege can have cascading effects.

Note the following:

- **Object definitions that depend on a DML object privilege can be affected if the DML object privilege is revoked.** For example, assume that the body of the `test` procedure includes a SQL statement that queries data from the `emp` table. If the `SELECT` privilege on the `emp` table is revoked from the owner of the `test` procedure, then the procedure can no longer be executed successfully.
- **When a REFERENCES privilege for a table is revoked from a user, any foreign key integrity constraints that are defined by the user and require the dropped REFERENCES privilege are automatically dropped.** For example, assume that user `jward` is granted the `REFERENCES` privilege for the `deptno` column of the `dept` table. This user now creates a foreign key on the `deptno` column in the `emp` table that references the `deptno` column of the `dept` table. If the `REFERENCES` privilege on the `deptno` column of the `dept` table is revoked, then the foreign key constraint on the `deptno` column of the `emp` table is dropped in the same operation.
- **The object privilege grants propagated using the GRANT OPTION are revoked if the object privilege of a grantor is revoked.** For example, assume that `user1` is granted the `SELECT` object privilege on the `emp` table with the `GRANT OPTION`, and grants the `SELECT` privilege on `emp` to `user2`. Subsequently, the `SELECT` privilege is revoked from `user1`. This `REVOKE` statement is also cascaded to `user2`. Any objects that depend on the revoked `SELECT` privilege of `user1` and `user2` can also be affected, as described earlier.

Object definitions that require the `ALTER` and `INDEX DDL` object privileges are not affected if the `ALTER` or `INDEX` object privilege is revoked. For example, if the `INDEX` privilege is revoked from a user that created an index on a table that belongs to another user, then the index continues to exist after the privilege is revoked.

Grants and Revokes of Privileges to and from the PUBLIC Role

You can grant and revoke privileges and roles from the role `PUBLIC`.

Because `PUBLIC` is accessible to every database user, all privileges and roles granted to `PUBLIC` are accessible to every database user. By default, `PUBLIC` does not have privileges granted to it.

Security administrators and database users should grant a privilege or role to `PUBLIC` only if every database user requires the privilege or role. This recommendation reinforces the general rule that, at any given time, each database user should have only the privileges required to accomplish the current group tasks successfully.

Revoking a privilege from the `PUBLIC` role can cause significant cascading effects. If any privilege related to a DML operation is revoked from `PUBLIC` (for example, `SELECT ANY TABLE` or `UPDATE ON emp`), then all procedures in the database, including functions and packages, must be *reauthorized* before they can be used again. Therefore, be careful when you grant and revoke DML-related privileges to or from `PUBLIC`.

See Also:

- *Oracle Database Administrator's Guide* for more information about managing object dependencies
- Guidelines for Securing Data

Grants of Roles Using the Operating System or Network

Using the operating system or network to manage roles can help centralize the role management in a large enterprise.

About Granting Roles Using the Operating System or Network

The operating system on which Oracle Database runs can be used to grant roles to users at connect time.

This feature is an alternative to a security administrator explicitly having to granting and revoking database roles to and from users using `GRANT` and `REVOKE` statements.

Roles can be administered using the operating system and passed to Oracle Database when a user creates a session. As part of this mechanism, the default roles of a user and the roles granted to a user with the `ADMIN` option can be identified. If the operating system is used to authorize users for roles, then all roles must be created in the database and privileges assigned to the role with `GRANT` statements.

Roles can also be granted through a network service.

The advantage of using the operating system to identify the database roles of a user is that privilege management for an Oracle database can be externalized. The security facilities offered by the operating system control user privileges. This option may offer advantages of centralizing security for several system activities, such as the following situation:

- MVS Oracle administrators want RACF groups to identify database user roles.
- UNIX Oracle administrators want UNIX groups to identify database user roles.
- VMS Oracle administrators want to use rights identifiers to identify database user roles.

The main disadvantage of using the operating system to identify the database roles of a user is that privilege management can only be performed at the role level. Individual privileges cannot be granted using the operating system, but they can still be granted inside the database using `GRANT` statements.

A second disadvantage of using this feature is that, by default, users cannot connect to the database through the shared server or any other network connection if the operating system is managing roles. However, you can change this default as described in Network Connections with Operating System Role Management.

In a multitenant environment, you can use operating system authentication for a database administrator only for the CDB root. You cannot use it for PDBs, the application root, or application PDBs.

Note:

The features described in this section are available only on some operating systems. See your operating system-specific Oracle Database documentation to determine if you can use these features.

Operating System Role Identification

The `OS_ROLES` initialization parameter can be used to control how the operating system identifies roles.

To have the database use the operating system to identify the database roles of each user when a session is created, you can set the initialization parameter `OS_ROLES` to `TRUE`.

If the instance is current running, you must restart the instance. When a user tries to create a session with the database, Oracle Database initializes the user security domain using the database roles identified by the operating system.

To identify database roles for a user, the operating system account for each Oracle Database user must have operating system identifiers (these may be called groups, rights identifiers, or other similar names) that indicate which database roles are to be available for the user. Role specification can also indicate which roles are the default roles of a user and which roles are available with the `ADMIN` option. No matter which operating system is used, the role specification at the operating system level follows the format:

```
ora_ID_ROLE[[_d][_a][_da]]
```

In this specification:

- `ID` has a definition that varies on different operating systems. For example, on VMS, `ID` is the instance identifier of the database; on VMS, it is the computer type; and on UNIX, it is the system `ID`.

  `ID` is case-sensitive to match your `ORACLE_SID`. `ROLE` is not case-sensitive.
- `ROLE` is the name of the database role.
- `d` is an optional character that indicates this role is to be a default role of the database user.
- `a` is an optional character that indicates this role is to be granted to the user with the `ADMIN` option. This allows the user to grant the role to other roles only. Roles cannot be granted to users if the operating system is used to manage roles.

  If either the `d` or `a` character is specified, then precede that character by an underscore (_).

For example, suppose an operating system account has the following roles identified in its profile:

```
ora_PAYROLL_ROLE1
ora_PAYROLL_ROLE2_a
ora_PAYROLL_ROLE3_d
ora_PAYROLL_ROLE4_da
```

When the corresponding user connects to the `payroll` instance of Oracle Database, `role3` and `role4` are defaults, while `role2` and `role4` are available with the `ADMIN` option.

Operating System Role Management

When you use operating system-managed roles, remember that database roles are being granted to an operating system user.

Any database user to which the operating system user is able to connect will have the authorized database roles enabled. For this reason, you should consider defining all Oracle Database users as `IDENTIFIED EXTERNALLY` if you are using `OS_ROLES = TRUE`, so that the database accounts are tied to the operating system account that was granted privileges.

Role Grants and Revokes When OS_ROLES Is Set to TRUE

Setting the `OS_ROLES` initialization parameter to `TRUE` enables the operating system to manage role grants and revokes to users.

Any previous granting of roles to users using `GRANT` statements do not apply. However, they are still listed in the data dictionary. Only the role grants to users made at the operating system level apply. Users can still grant privileges to roles and users.

Note:

If the operating system grants a role to a user with the `ADMIN` option, then the user can grant the role only to other roles.

Role Enablements and Disablements When OS_ROLES Is Set to TRUE

Setting the `OS_ROLES` initialization parameter to `TRUE` enables the `SET ROLE` statement to dynamically enable roles granted by the operating system.

This still applies, even if the role was defined to require a password or operating system authorization. However, any role not identified in the operating system account of a user cannot be specified in a `SET ROLE` statement, even if a role was granted using a `GRANT` statement when `OS_ROLES = FALSE`. (If you specify such a role, then Oracle Database ignores it.)

When `OS_ROLES` is set to `TRUE`, then the user can enable up to 148 roles. Remember that this number includes other roles that may have been granted to the role.

Network Connections with Operating System Role Management

By default, users cannot connect to the database through a shared server if the operating system manages roles.

This restriction is the default because a remote user could impersonate another operating system user over an unsecure connection.

If you are not concerned with this security risk and want to use operating system role management with the shared server, or any other network connection, then set the initialization parameter `REMOTE_OS_ROLES` to `TRUE`. The change takes effect the next time you start the instance and mount the database. The default setting of this parameter is `FALSE`.

How Grants and Revokes Work with SET ROLE and Default Role Settings

Privilege grants and the `SET ROLE` statement affect when and how grants and revokes take place.

When Grants and Revokes Take Effect

Depending on the privilege that is granted or revoked, a grant or revoke takes effect at different times.

The grants and revokes take effect as follows:

- All grants and revokes of system and object privileges to anything (users, roles, and `PUBLIC`) take immediate effect.
- All grants and revokes of roles to anything (users, other roles, `PUBLIC`) take effect only when a current user session issues a `SET ROLE` statement to reenable the role after the grant and revoke, or when a new user session is created after the grant or revoke.

You can see which roles are currently enabled by examining the `SESSION_ROLES` data dictionary view.

How the SET ROLE Statement Affects Grants and Revokes

During a user session, a user or an application can use the `SET ROLE` statement multiple times to change the roles enabled for the session.

The user must already be granted the roles that are named in the `SET ROLE` statement.

The following example enables the role `clerk`, which you have already been granted, and specifies the password.

```
SET ROLE clerk IDENTIFIED BY password;
```

Follow the guidelines in Minimum Requirements for Passwords to replace *`password`* with a password that is secure.

The following example shows how to use `SET ROLE` to disable all roles.

```
SET ROLE NONE;
```

Specifying the Default Role for a User

When a user logs on, Oracle Database enables all privileges granted explicitly to the user and all privileges in the user’s default roles.

1. Ensure that the user who you want to set the default role for has been directly granted the role with a `GRANT` statement, or that the role was created by the user with the `CREATE ROLE` privilege.
2. Use the `ALTER USER` statement with the `DEFAULT ROLE` clause to specify the default role for the user.

For example, to set the default roles `payclerk` and `pettycash` for user `jane`:

```
ALTER USER jane DEFAULT ROLE payclerk, pettycash;
```

For information about the restrictions of the `DEFAULT ROLE` clause of the `ALTER USER` statement, see *Oracle Database SQL Language Reference*.

You cannot set default roles for a user in the `CREATE USER` statement. When you first create a user, the default user role setting is `ALL`, which causes all roles subsequently granted to the user to be default roles. Use the `ALTER USER` statement to limit the default user roles.

Note:

When you create a role (other than a global role or an application role), it is granted implicitly to you, and your set of default roles is updated to include the new role. Be aware that only 148 roles can be enabled for a user session. When aggregate roles, such as the `DBA` role, are granted to a user, the roles granted to the role are included in the number of roles the user has. For example, if a role has 20 roles granted to it and you grant that role to the user, then the user now has 21 additional roles. Therefore, when you grant new roles to a user, use the `DEFAULT ROLE` clause of the `ALTER USER` statement to ensure that not too many roles are specified as that user's default roles.

The Maximum Number of Roles That a User Can Have Enabled

You can grant a user as many roles as you want, but no more than 148 roles can be enabled for a logged-in user at any given time.

Therefore, not all privileges will be available to this user during the user session. As a best practice, restrict the number of roles granted to a user to the minimum roles the user needs.

User Privilege and Role Data Dictionary Views

You can use special queries to find information about various types of privilege and role grants.

Data Dictionary Views to Find Information about Privilege and Role Grants

Oracle Database provides data dictionary views that describe privilege and role grants.

Table 4-6 lists views that you can query to access information about grants of privileges and roles.

Table 4-6 Data Dictionary Views That Display Privilege and Role Information

View	Description
`ALL_COL_PRIVS`	Describes all column object grants for which the current user or `PUBLIC` is the object owner, grantor, or grantee
`ALL_COL_PRIVS_MADE`	Lists column object grants for which the current user is object owner or grantor
`ALL_COL_PRIVS_RECD`	Describes column object grants for which the current user or `PUBLIC` is the grantee
`ALL_TAB_PRIVS`	Lists the grants on objects where the user or `PUBLIC` is the grantee
`ALL_TAB_PRIVS_MADE`	Lists the all object grants made by the current user or made on the objects owned by the current user
`ALL_TAB_PRIVS_RECD`	Lists object grants for which the user or `PUBLIC` is the grantee
`DBA_COL_PRIVS`	Describes all column object grants in the database
`DBA_CONTAINER_DATA`	In a multitenant environment, displays default (user-level) and object-specific `CONTAINER_DATA` attributes. Objects that are created with the `CONTAINER_DATA` clause include `CONTAINER_DATA` attributes.
`DBA_EPG_DAD_AUTHORIZATION`	Describes the database access descriptors (DAD) that are authorized to use a different user's privileges
`DBA_LOCKDOWN_PROFILES`	Describes information that pertains to PDB lockdown profiles
`DBA_OBJECTS`	Lists objects that have object links or metadata links. To find these objects, query the `OBJECT_NAME` and `SHARING` columns.
`DBA_TAB_PRIVS`	Lists all grants on all objects in the database

Table 4-6 (Cont.) Data Dictionary Views That Display Privilege and Role Information

View	Description
`DBA_ROLES`	Lists all roles that exist in the database, including secure application roles. Note that it does not list the `PUBLIC` role
`DBA_ROLE_PRIVS`	Lists roles directly granted to users and roles
`DBA_SYS_PRIVS`	Lists system privileges granted to users and roles
`ROLE_ROLE_PRIVS`	Lists roles granted to other roles. Information is provided only about roles to which the user has access
`ROLE_SYS_PRIVS`	Lists system privileges granted to roles. Information is provided only about roles to which the user has access
`ROLE_TAB_PRIVS`	Lists object privileges granted to roles. Information is provided only about roles to which the user has access
`SESSION_PRIVS`	Lists the privileges that are currently enabled for the user
`SESSION_ROLES`	Lists all roles that are enabled for the current user. Note that it does not list the `PUBLIC` role
`USER_COL_PRIVS`	Describes column object grants for which the current user is the object owner, grantor, or grantee
`USER_COL_PRIVS_MADE`	Describes column object grants for which the current user is the object owner
`USER_COL_PRIVS_RECD`	Describes column object grants for which the current user is the grantee
`USER_EPG_DAD_AUTHORIZATION`	Describes the database access descriptors (DAD) that are authorized to use a different user's privileges
`USER_ROLE_PRIVS`	Lists roles directly granted to the current user
`USER_TAB_PRIVS`	Lists grants on all objects where the current user is the grantee
`USER_SYS_PRIVS`	Lists system privileges granted to the current user
`USER_TAB_PRIVS_MADE`	Lists grants on all objects owned by the current user
`USER_TAB_PRIVS_RECD`	Lists object grants for which the current user is the grantee
`V$PWFILE_USERS`	Lists all users in the current PDB who have been granted administrative privileges

The following table lists views that you can query to access information about grants of privileges and roles.

This section provides some examples of using these views. For these examples, assume the following statements were issued:

```
CREATE ROLE security_admin IDENTIFIED BY password;

GRANT CREATE PROFILE, ALTER PROFILE, DROP PROFILE,
    CREATE ROLE, DROP ANY ROLE, GRANT ANY ROLE, AUDIT ANY,
    AUDIT SYSTEM, CREATE USER, BECOME USER, ALTER USER, DROP USER
    TO security_admin WITH ADMIN OPTION;

GRANT READ, DELETE ON SYS.AUD$ TO security_admin;

GRANT security_admin, CREATE SESSION TO swilliams;
GRANT security_admin TO system_administrator;

GRANT CREATE SESSION TO jward;

GRANT READ, DELETE ON emp TO jward;

GRANT INSERT (ename, job) ON emp TO swilliams, jward;
```

See Also:

Oracle Database Reference for a detailed description of these data dictionary views

Query to List All System Privilege Grants

The `DBA_SYS_PRIVS` data dictionary view returns all system privilege grants made to roles and users.

For example:

```
SELECT GRANTEE, PRIVILEGE, ADM FROM DBA_SYS_PRIVS;

GRANTEE           PRIVILEGE                         ADM
--------------    --------------------------------- ---
SECURITY_ADMIN    ALTER PROFILE                     YES
SECURITY_ADMIN    ALTER USER                        YES
SECURITY_ADMIN    AUDIT ANY                         YES
SECURITY_ADMIN    AUDIT SYSTEM                      YES
SECURITY_ADMIN    BECOME USER                       YES
SECURITY_ADMIN    CREATE PROFILE                    YES
SECURITY_ADMIN    CREATE ROLE                       YES
SECURITY_ADMIN    CREATE USER                       YES
SECURITY_ADMIN    DROP ANY ROLE                     YES
SECURITY_ADMIN    DROP PROFILE                      YES
SECURITY_ADMIN    DROP USER                         YES
SECURITY_ADMIN    GRANT ANY ROLE                    YES
SWILLIAMS         CREATE SESSION                    NO
JWARD             CREATE SESSION                    NO
```

See Also:

Oracle Database Reference for detailed information about the `DBA_SYS_PRIVS` view

Query to List All Role Grants

The `DBA_ROLE_PRIVS` query returns all the roles granted to users and other roles.

For example:

```
SELECT * FROM DBA_ROLE_PRIVS;
```

```
GRANTEE            GRANTED_ROLE                         ADM
------------------ ------------------------------------ ---
SWILLIAMS          SECURITY_ADMIN                       NO
```

See Also:

Oracle Database Reference for detailed information about the `DBA_ROLE_PRIVS` view

Query to List Object Privileges Granted to a User

The `DBA_TAB_PRIVS` and `DBA_COL_PRIVS` data dictionary views list object privileges that have bee granted to users.

The `DBA_TAB_PRIVS` data dictionary view returns all object privileges (not including column-specific privileges) granted to the specified user.

For example:

```
SELECT TABLE_NAME, PRIVILEGE, GRANTABLE FROM DBA_TAB_PRIVS
    WHERE GRANTEE = 'jward';

TABLE_NAME   PRIVILEGE    GRANTABLE
-----------  ------------ ----------
EMP          SELECT       NO
EMP          DELETE       NO
```

To list all the column-specific privileges that have been granted, you can use the following query:

```
SELECT GRANTEE, TABLE_NAME, COLUMN_NAME, PRIVILEGE
    FROM DBA_COL_PRIVS;

GRANTEE      TABLE_NAME     COLUMN_NAME      PRIVILEGE
-----------  ------------   -------------    --------------
SWILLIAMS    EMP            ENAME            INSERT
SWILLIAMS    EMP            JOB              INSERT
JWARD        EMP            NAME             INSERT
JWARD        EMP            JOB              INSERT
```

See Also:

Oracle Database Reference for detailed information about the `DBA_TAB_PRIVS` view

Query to List the Current Privilege Domain of Your Session

The `SESSION_ROLES` and `SESSION_PRIVS` data dictionary views list the current privilege domain of a database session.

The `SESSION_ROLES` view lists all roles currently enabled for the issuer.

For example:

```
SELECT * FROM SESSION_ROLES;
```

If user `swilliams` has the `security_admin` role enabled and issues the previous query, then Oracle Database returns the following information:

```
ROLE
------------------------------
SECURITY_ADMIN
```

The following query lists all system privileges currently available in the security domain of the issuer, both from explicit privilege grants and from enabled roles:

```
SELECT * FROM SESSION_PRIVS;
```

If user `swilliams` has the `security_admin` role enabled and issues the previous query, then Oracle Database returns the following results:

```
PRIVILEGE
----------------------------------------
AUDIT SYSTEM
CREATE SESSION
CREATE USER
BECOME USER
ALTER USER
DROP USER
CREATE ROLE
DROP ANY ROLE
GRANT ANY ROLE
AUDIT ANY
CREATE PROFILE
ALTER PROFILE
DROP PROFILE
```

If the `security_admin` role is disabled for user `swilliams`, then the first query would return no rows, while the second query would only return a row for the `CREATE SESSION` privilege grant.

See Also:

Oracle Database Reference for detailed information about the `SESSION_ROLES` view

Query to List Roles of the Database

The `DBA_ROLES` data dictionary view lists all roles of a database and the authentication used for each role.

For example:

```
SELECT * FROM DBA_ROLES;

ROLE                  PASSWORD
----------------      --------
CONNECT               NO
RESOURCE              NO
DBA                   NO
SECURITY_ADMIN        YES
```

See Also:

Oracle Database Reference for detailed information about the `DBA_ROLES` view

Query to List Information About the Privilege Domains of Roles

The `ROLE_ROLE_PRIVS`, `ROLE_SYS_PRIVS`, and `ROLE_TAB_PRIVS` data dictionary views list information about the privilege domains of roles.

For example:

```
SELECT GRANTED_ROLE, ADMIN_OPTION
   FROM ROLE_ROLE_PRIVS
   WHERE ROLE = 'SYSTEM_ADMIN';

GRANTED_ROLE              ADM
----------------          ----
SECURITY_ADMIN            NO
```

The following query lists all the system privileges granted to the `security_admin` role:

```
SELECT * FROM ROLE_SYS_PRIVS WHERE ROLE = 'SECURITY_ADMIN';

ROLE                    PRIVILEGE                      ADM
----------------------- ----------------------------   ---
SECURITY_ADMIN           ALTER PROFILE                 YES
SECURITY_ADMIN           ALTER USER                    YES
SECURITY_ADMIN           AUDIT ANY                     YES
SECURITY_ADMIN           AUDIT SYSTEM                  YES
SECURITY_ADMIN           BECOME USER                   YES
SECURITY_ADMIN           CREATE PROFILE                YES
SECURITY_ADMIN           CREATE ROLE                   YES
SECURITY_ADMIN           CREATE USER                   YES
SECURITY_ADMIN           DROP ANY ROLE                 YES
SECURITY_ADMIN           DROP PROFILE                  YES
SECURITY_ADMIN           DROP USER                     YES
SECURITY_ADMIN           GRANT ANY ROLE                YES
```

The following query lists all the object privileges granted to the `security_admin` role:

```
SELECT TABLE_NAME, PRIVILEGE FROM ROLE_TAB_PRIVS
    WHERE ROLE = 'SECURITY_ADMIN';

TABLE_NAME                    PRIVILEGE
---------------------------   ----------------
AUD$                          DELETE
AUD$                          SELECT
```

See Also:

Oracle Database ReferenceOracle Database Reference for detailed information about the `ROLE_ROLE_PRIVS`, `ROLE_SYS_PRIVS`, and `ROLE_TAB_PRIVS` views

5

Configuring Centrally Managed Users with Microsoft Active Directory

Oracle Database can authenticate and authorize Microsoft Active Directory users with the database directly without intermediate directories or Oracle Enterprise User Security.

Introduction to Centrally Managed Users with Microsoft Active Directory

Centrally managed users provides for a simpler integration with Microsoft Active Directory to allow centralized authentication and authorization of users.

About the Oracle Database-Microsoft Active Directory Integration

Centrally managed users provides for a simpler integration with Microsoft Active Directory to allow centralized authentication and authorization of users.

This integration enables organizations to use Active Directory to centrally manage users and roles in multiple Oracle databases with a single directory along with other Information Technology services. Users can authenticate to the Oracle Database using credentials stored in Active Directory and also be associated with Database schemas and roles using Active Directory groups. Microsoft Active Directory users can be mapped to exclusive or shared Oracle Database schemas and associated with database roles in the directory. Active Directory password policies such as password expiration time and lockout after # failed login attempts are honored by the Oracle Database when users login.

Before Oracle Database 18c release 1 (18.1), users integrated user authentication and authorization with Active Directory by configuring Oracle Enterprise User Security and installing and configuring Oracle Internet Directory (or Oracle Universal Directory). This architecture is still available and will be used going forward by users who must use trusted database links, complex enterprise roles and having a single place for auditing database access privileges and roles.

The majority of organizations don’t require these complex requirements. Instead, they can use centrally managed users (CMUs) with Active Directory. This integration is designed for organizations who prefer to use Active Directory as their centralized identity management solution. Oracle Net Naming Services continues to work as it did before with directory services.

Organizations can use Kerberos, PKI, or password authentication with CMU with Active Directory. Use of CMU with Active Directory is backwards compatible with currently supported Oracle Database clients. This means that LDAP bind operations are not used for password authentication and you will need to add an Oracle filter to Active Directory along with an extension to the Active Directory schema to store password verifiers. Organizations using Kerberos or PKI will not need to add the filter or extend Active Directory.

The Oracle Database-Active Directory integration is particularly beneficial for the following types of users:

- Users who are currently using strong authentication tools such as Kerberos or Public Key Infrastructure (PKI). These users already use a centralized identity management system
- Users who currently use Oracle Enterprise User Security, Oracle Internet Directory, Oracle Unified Directory, Oracle Virtual Directory, and need to integrate with Active Directory.

How Centrally Managed Users with Microsoft Active Directory Works

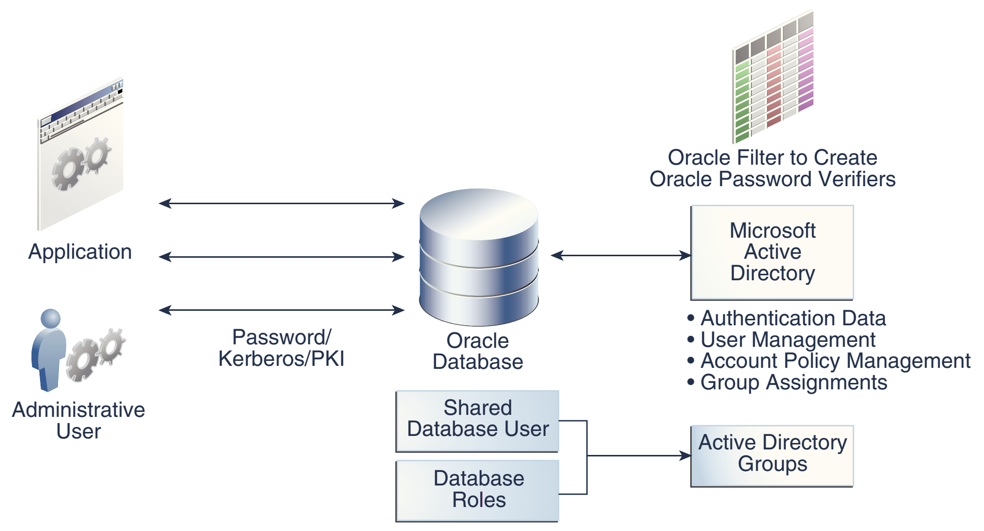
The integration works by creating a mapping between database users and roles with Microsoft Active Directory users and groups.

In order for the Oracle Database CMU with Active Directory integration to work, the Oracle database must be able to login to a service account specifically created for the database in Active Directory. The database uses this service account to query Active Directory for user and group information when a user logs into the database. This Active Directory service account must have all the privileges required to query the user and group information as well as being able to write updates related to the password policies in Active Directory (for example, failed login attempts, clear failed login attempts). Users can authenticate using passwords, Kerberos, or PKI and either be assigned to an exclusive schema or a shared schema. Mapping of an Active Directory user to a shared schema is determined by the association of the user to an Active Directory group that is mapped to the shared schema. Active Directory groups can also be mapped to database global roles. Active Directory security administrators can assign users to a shared schema and a global role mapped groups to change and update privileges and roles assigned to the Active Directory user in a database.

Centrally Managed User-Microsoft Active Directory Architecture

The CMU with Active Directory architecture enables Oracle Database users and roles to be managed in Active Directory.

The following figure illustrates the Oracle Database CMU feature. In this figure, users, either through applications as non-administrative users or administrative users, connect to the Oracle database with either password, Kerberos, or public key infrastructure (PKI) authentication. The database connection to Active Directory enables these users and roles to be mapped with Active Directory users and groups. If you plan to use password authentication, then you must install an Oracle filter in Active Directory. This filter will create Oracle password verifiers and extend the Active Directory schema to hold the Oracle password verifiers. With Oracle Database centrally managed users, an Active Directory administrator can control the authentication, user management, account policies, and group assignments of Active Directory users and groups who have been mapped to Oracle Database users and roles.



Supported Authentication Methods

The Oracle Database-Microsoft Active Directory integration supports three common authentication methods.

These authentication methods are as follows:

- Password authentication
- Kerberos authentication
- Public key infrastructure (PKI) authentication (certificate-based authentication)

Users Supported by Centrally Managed Users with Microsoft Active Directory

CMU with Active Directory supports exclusively mapped users, users mapped to shared schemas, and administrative users.

These users are as follows:

- Directory users that access an Oracle database using a shared schema.

  This type of directory user can connect to a shared schema in the database by being part of a directory group that is mapped to the shared schema (database user). Using shared schemas allows centralized Active Directory management of database users and is the recommended best practices over using exclusive schemas (described next). Even if there is only one user associated with a schema (for example, an administrator responsible for database backup), it is easier to manage adding another backup administrator or removing the existing administrator by making changes only in Active Directory instead of making changes in all associated databases as well.

  Users will be given additional privileges appropriate to their task using global roles that are mapped to groups in Active Directory. With this design, a user can change his or her tasks within an organization and have new privileges through a new group in Active Directory.

Active Directory users could accidentally (or on purpose) be a member of multiple groups in Active Directory that are mapped to different shared schemas on the same database. The user could also have an exclusive mapping to a database schema. In cases where the user has multiple possible schema mappings when they login, the following precedence rules apply:

- If an exclusive mapping exists for a user, then that mapping takes precedence over any other shared mappings.
- If multiple shared schema mappings exist for a user, then the oldest shared user mapping takes precedence.

Oracle recommends only having one possible mapping per user so unexpected schema mappings don't occur.

- Exclusively mapped global users who are regular Oracle Database users in two- and three-tier applications, or users who have direct privilege grants in the database.

  Oracle recommends that you grant privileges to these users through global roles. This type of privilege grant facilitates authorization management by centrally managing privileges and roles for a user instead of having to log in into each database to update privileges and roles for the user.

- Administrative global users, who have the following administrative privileges: `SYSDBA`, `SYSOPER`, `SYSDG`, `SYSKM`, and `SYSRAC`.

  You cannot directly grant these administrative privileges through global roles. To authorize an Active Directory user with these administrative privileges, you must map the directory user to a database user (exclusively or with a shared schema) that has the system administrative privilege already granted to the database user account.

How the Oracle Multitenant Option Affects Centrally Managed Users

Multitenant database users in a pluggable database (PDB) can connect to a central Microsoft Active Directory or, if required, connect each PDB to a different Microsoft Active Directory.

An entire multitenant database can authenticate and authorize against a single Active Directory instance. Alternatively, individual pluggable databases (PDBs) can authenticate and authorize against different Active Directory domains.

Configuring the Oracle Database-Microsoft Active Directory Integration

Before you can use Microsoft Active Directory to authenticate and authorize users, you must configure the connection from the Oracle database to Active Directory.

About Configuring the Oracle Database-Microsoft Active Directory Connection

Before you configure this connection, you must have Microsoft Active Directory installed and configured.

You must create an Oracle service directory user in Active Directory, configure the Oracle Database connection to Active Directory, and then depending on the authentication type, configure the database and Active Directory for password, Kerberos, or public key infrastructure (PKI) authentication. Before you map Database users and global roles to Active Directory users and groups, you must ensure that the Active Directory users and groups have been created. You will map the database users and global roles to Active Directory users and groups by using the `CREATE USER`, `CREATE ROLE`, `ALTER USER`, `ALTER ROLE` SQL statements with the `GLOBALLY` clause. An Active Directory system administrator must also set up new Active Directory groups with Active Directory users to meet your requirements.

Connecting to Microsoft Active Directory

You can configure a Microsoft Active Directory connection during the Oracle Database installation process or to an existing Oracle database.

Step 1: Create an Oracle Service Directory User Account on Microsoft Active Directory

The Oracle service directory user account is for the interaction between Oracle Database and the LDAP directory service.

In addition to being used for the Oracle Database-to-LDAP directory service interaction, the Oracle service directory user account can be used for Kerberos.

This account is an Active Directory account that Oracle Database uses to log in to Active Directory and query for users and group information from Active Directory, update login and success failures, and if Kerberos is configured, update Kerberos authentication. The minimum permissions required for this account are Read properties (of the Active Directory user), and if database password authentication is to be used by an Active Directory user, the Write lockoutTime (property of the Active Directory user) permission.

1. Log in to Microsoft Active Directory as a user who has privileges to create accounts and grant them privileges.
2. Create the Oracle Service Directory user account as an Active Directory user.

   Ideally, create the managed service account in the root directory. Depending on the domains that your users will use, you can also create this account in child domains. Follow these guidelines:

   - You should create this account at the lowest possible Active Directory location if the users are in multiple domains.
   - If all the Active Directory users will be in one domain, then create this account in that domain. Doing so will help performance.
   - Create this account in the Windows Active Directory root if:
     – The Active directory users will be in different domains.
     – Active Directory has multiple Windows domains, so that CMU can support these multiple domains.
3. Grant the Oracle service directory user account the Read properties (of the Active Directory user) and Write Lockout Time (property of the Active Directory user) permissions for the domains that will contain the users and groups.

Step 2: For Password Authentication, Install the Password Filter and Extend the Microsoft Active Directory Schema

You can use the Oracle `opwdintg.exe` executable on the Active Directory server to install the password filter and extend the Active Directory schema.

You do not need to perform this step if your authentication method is Kerberos or SSL. The `opwdintg.exe` executable installs the Oracle password filter, extends the AD schema, and creates the authorization Active directory groups to allow Oracle Database password authentication with Active Directory. This procedure adds an `orclCommonAttribute` attribute to the Active Directory schema for user accounts.

1. Go to the `$ORACLE_HOME/bin` directory.
2. Find the `opwdintg.exe` (Oracle Password Integration) utility.
3. Using a secure method of copying (such as `sftp`), copy `opwdintg.exe` to a temporary directory (for example, `C:\temp`) on each Windows domain controller.
4. Connect to the Windows computer as the Active Directory administrator.
5. Run the `opwdintg.exe` utility.
6. Answer the following prompts:
   - **Do you want to extend AD schema? [Yes/No]:** Enter `Yes`.
   - **Schema extension for this domain will be permanent. Continue? [Yes/No]:** Enter `Yes`.
     Note the following:
     – You can only extend the Active Directory schema from a Windows domain controller. If you try to extend the schema again, error messages appear, but you can ignore these errors.
     – This step creates the following three verifier groups. If these groups already exist, then errors will appear, but you can ignore these errors.
       * `ORA_VFR_MD5` is required when the Oracle Database WebDAV client is used.
       * `ORA_VFR_11G` enables the use of the Oracle Database 11G password verifier.
       * `ORA_VFR_12C` enables the user of the Oracle Database 12C password verifier.
   - **Found password filter installed already. Do you want to deinstall? [Yes/No]:** Enter `Yes` if you want to deinstall the password filter. Enter `No` if you do not want to deinstall the password filter.
   - **Do you want to install Oracle password filter? [Yes/No]:** Enter `Yes`.
   - **The change requires machine reboot. Do you want to reboot now? [Yes/No]:** Enter `Yes`.

Step 3: If Necessary, Install the Oracle Database Software

If you have not done so yet, then use Oracle Universal Installer (OUI) to install the Oracle database.

- Follow the instructions in the *Oracle Database Installation Guide* for your platform to install the Oracle database.

After you install the Oracle database, you can configure centrally managed users with Active Directory for this database.

Step 4: Create the dsi.ora or ldap.ora File

The `dsi.ora` file specifies connections for centrally managed users for Active Directory.

The `ldap.ora` file can also specifies the connection to the Active Directory server. But because `ldap.ora` may already be used (or may be used in the future) for other services like net naming services, Oracle recommends the use of `dsi.ora` for centrally managed users.

About Using a dsi.ora File

You use a `dsi.ora` file to specify an Active Directory service for centrally managed users.

You must manually create the `dsi.ora` file to identify the Active Directory server. The `dsi.ora` file provides Active Directory connection information for all containers if it is located in the same places where the `ldap.ora` file can be placed. A `dsi.ora` file and a wallet in the `$ORACLE_BASE/admin/db_unique_name/pdb_guid/wallet` location takes precedence over the main `dsi.ora` file for that PDB only.

Note:

If you are using `ldap.ora` for naming services, then do not make any changes to `ldap.ora` and its wallet for the CMU with Active Directory configuration. Only use `dsi.ora` to configure CMU-Active Directory.

When you create the `dsi.ora` file, Oracle Database searches for it in the following order (same locations as the Database would search for the `ldap.ora` file):

1. `$LDAP_ADMIN` environment variable setting
2. `$ORACLE_HOME/ldap/admin` directory
3. `$TNS_ADMIN` environment variable setting
4. `$ORACLE_HOME/network/admin` directory

Oracle recommends the use of directories under `$ORACLE_BASE` that are not under `$ORACLE_HOME`. Starting with Oracle Database 18c, there is an option to have read-only `$ORACLE_HOME` so `dsi.ora` should be placed in a directory outside of `$ORACLE_HOME` for future-proofing your architecture.

Oracle recommends only using dsi.ora to identify the Active Directory server for centrally managed users. If both `dsi.ora` and `ldap.ora` happen to be configured in the same database for centrally managed users for Active Directory, dsi.ora will take precedence over the `ldap.ora` file if both are located in the same directory. If they are located in different directories, then the first one that Oracle finds in the location precedence list above will be used to find the Active Directory server.

You can specify `dsi.ora` files for individual PDBs in a multitenant database. A PDB specific dsi.ora file will override the CMU-Active Directory settings for the general database for that one PDB. Different PDBs can connect to different Active Directory servers for CMU. The `dsi.ora` files for PDBs are located in the same directory as the wallet for that PDB.

Wallet locations for individual container in a multitenant database should place the wallet files in the `$ORACLE_BASE/admin/`*`db_unique_name`*`/pdb_guid/wallet` directory. To find the *`db unique`* name, connect to the CDB root and execute the following query:

```
SELECT INSTANCE_NAME FROM V$INSTANCE;
```

To find the `pdb_guid`, from the CDB root, execute the following query:

```
SELECT PDB_ID,PDB_NAME,GUID FROM DBA_PDBS;
```

You can also place `dsi.ora` in the wallet directory for the CDB root container, but that will only connect the CDB root container to the Active Directory – not the entire database.

Creating the dsi.ora File

The `dsi.ora` configuration file sets the information to find the Active Directory server for centrally managed users.

To use the `dsi.ora` configuration file:

1. Log in to the server where the Oracle database is located.
2. Go to the directory where you want to create the `dsi.ora` file.
3. Add the following parameters to the `dsi.ora` file:
   - `DSI_DIRECTORY_SERVERS`, which sets the Active Directory server host and port name, and alternate directory servers. The directory server name must be a fully qualified name. For example:

     ```
     DSI_DIRECTORY_SERVERS = (ldap-server.us.example.com:389:636,
     sparky.us.example.com:389:636)
     ```

   - `DSI_DEFAULT_ADMIN_CONTEXT`, which sets the names of the Active Server domains in which the Active Server directory users are located. For example:

     ```
     DSI_DEFAULT_ADMIN_CONTEXT = "o=Example,c=US"
     ```

   - `DSI_DIRECTORY_SERVER_TYPE`, which determines the Active Directory server access. You must set it to `AD` for Active Directory. Enter this value in upper case.

     ```
     DSI_DIRECTORY_SERVER_TYPE = AD
     ```

About Using an ldap.ora File

You can use an `ldap.ora` file to specify an Active Directory server for centrally managed services.

If you are already using an `ldap.ora` file for another purpose such as net naming services, then you must use the `dsi.ora` file to configure centrally managed users to connect with Active Directory for user authentication and authorization. Even if Active Directory is already being used for net naming services, you must create and use a `dsi.ora` file to identify the Active Directory server for centrally managed users. Even if the database currently isn't using `ldap.ora` for another service, Oracle recommends using `dsi.ora` in case `ldap.ora` will be used at a future time for net naming services..

The benefit of using `ldap.ora` is that you can use the DBCA graphical interface or the DBCA silent mode to complete the connection to the Active Directory server. When using `dsi.ora`, the steps to complete the connection to Active Directory must be done separately.

If `ldap.ora` is being used for naming services, then do not make any changes to `ldap.ora` and its wallet for the CMU with Active Directory configuration. Only use `dsi.ora` to configure CMU-Active Directory.

Typically, the `ldap.ora` file is stored in the `$ORACLE_HOME/network/admin` directory. Usually, the `ldap.ora` file cannot be in the same directory as the `WALLET_LOCATION` that is specified in the `sqlnet.ora` file, unless the `WALLET_LOCATION` is set to `$ORACLE_HOME/network/admin`.

After you create the `ldap.ora` file, Oracle Database searches for it in the following order:

1. `$LDAP_ADMIN` environment variable setting
2. `$ORACLE_HOME/ldap/admin` directory
3. `$TNS_ADMIN` environment variable setting
4. `$ORACLE_HOME/network/admin` directory

Creating the ldap.ora File

These steps assume that `ldap.ora` is not being used for net naming services and can be used to set up the connection with Active Directory for centrally managed users.

1. Log in to the server where the Oracle database is located.
2. Go to the directory where you want to create the `ldap.ora` file.
3. If the `ldap.ora` file does not exist, then create it by using a text editor.

   If the `ldap.ora` file does exist, create a back-up of this file, and then open `ldap.ora`.
4. Add the following parameters to the `ldap.ora` file:
   - `DIRECTORY_SERVERS`, which sets the directory server host and port name, and alternate directory servers. The directory server name must be a fully qualified name. For example:

     ```
     DIRECTORY_SERVERS = (ldap-server.us.example.com:389:636,
     sparky.us.example.com:389:636)
     ```
   - `DEFAULT_ADMIN_CONTEXT`, which sets the names of the Active Server domains in which the Active Server directory users are located. For example:

     ```
     DEFAULT_ADMIN_CONTEXT = "o=Example,c=US"
     ```
   - `DIRECTORY_SERVER_TYPE`, which determines the LDAP server access. You must set it to `AD` for Active Directory. Enter this value in upper case.

     ```
     DIRECTORY_SERVER_TYPE = AD
     ```

Step 5: Request an Active Directory Certificate for a Secure Connection

After you have configured the `dsi.ora` or `ldap.ora` file, you are ready to prepare Microsoft Active Directory and Oracle Database certificates for a secure connection.

- Request the Active Directory certificate from an Active Directory administrator.

Step 6: Create the Wallet for a secure connection

After you have copied the Active Directory certificate, you are ready to add it to the Oracle wallet.

1. Copy the certificate text file (for example, `AD_CA_Root_cert.txt`) from the Active Directory server to the current directory.

   If wallet location is not specified in the `sqlnet.ora` file, then the database will search the following locations in order for the wallet. The directory location may need to be created.

   a. `$ORACLE_BASE/admin/`*`db unique name`*`/wallet`

   b. `$ORACLE_HOME/admin/`*`db unique name`*`/wallet`

   Wallet locations for individual container in a multitenant database should place the wallet files in the `$ORACLE_BASE/admin/`*`db unique name`*`/pdb_guid/wallet/` directory. To find the *`db unique name`*, connect to the CDB root and execute the following query:

   ```
   SELECT INSTANCE_NAME FROM V$INSTANCE;
   ```

   To find the `pdb_guid`, from the CDB root, execute the following query:

   ```
   SELECT PDB_ID,PDB_NAME,GUID FROM DBA_PDBS;
   ```

   If `sqlnet.ora` is used to specify the wallet location, then individual PDB `dsi.ora` and wallets will not be recognized. Only the database-wide `dsi.ora` or `ldap.ora` located in one of the four default locations will be recognized. When the wallet location is specified then all containers must use the same Active Directory server.

2. Create a new wallet.

   The following command creates an `ewallet.p12` wallet in the current directory.

   ```
   orapki wallet create -wallet .
   Enter password: password
   ```

3. Create the Oracle directory service account name for performing searches in Active Directory (created in the first step).

   For example:

   ```
   mkstore -wrl . -createEntry ORACLE.SECURITY.USERNAME oracle
   ```

4. Specify the DN of the Oracle directory service account name.

   For example:

   ```
   mkstore -wrl . -createEntry ORACLE.SECURITY.DN
   cn=oracle,cn=users,dc=adintg,dc=examplecorp,dc=com
   ```

5. Create the user password credential for the Oracle directory service account name..

   For example:

   ```
   mkstore -wrl . -createEntry ORACLE.SECURITY.DN
   cn=Users,dc=test5,dc=dbsec18,dc=com
   ```

6. Add the certificate to the wallet. Use the Active Directory certificate you received from the Active Directory administrator and put it in the same directory.

For example:

```
orapki wallet add -wallet . -cert AD_CA_Root_cert.txt -trusted_cert
```

7. Verify the credentials.

   For example:

   ```
   orapki wallet display -wallet .
   ```

   The output should be similar to the following:

   ```
   Requested Certificates:
   User Certificates:
   Oracle Secret Store entries:
   ORACLE.SECURITY.DN
   ORACLE.SECURITY.PASSWORD
   ORACLE.SECURITY.USERNAME
   Trusted Certificates:
   Subject: CN=ADSVR,DC=production,DC=examplecorp,DC=com
   ```

Step 7: Configure the Microsoft Active Directory Connection

Next, you are ready to connect the database to Active Directory using the settings you have so far.

About Configuring the Microsoft Active Directory Connection

To configure the Microsoft Active Directory connection, you can set the parameters in the database or use DBCA.

DBCA creates the wallet so you do not need to create it separately. DBCA only recognizes `ldap.ora` that is configured for centrally managed users. If you are using `dsi.ora` then you must use the manual configuration methods.

> **Note:**
>
> Oracle recommends using `dsi.ora` for CMU-Active Directory.

Configuring the Access Manually Using Database System Parameters

You can configure the Active Directory services connection manually by using LDAP-specific Oracle Database system parameters.

1. Ensure that you have created the `dsi.ora` file or the `ldap.ora` file, and that you have created the wallet.
2. Log in to the database instance as a user who has the `ALTER SYSTEM` system privilege.

   In a multitenant environment, you can log in to the CDB root or to a PDB.
   For example, to log in to the CDB root:

   ```
   sqlplus c##sec_admin
   Enter password: password
   ```

3. Modify the `LDAP_DIRECTORY_ACCESS` parameter, which determines the type of LDAP directory access.

Valid values are `PASSWORD` and `NONE` (to disable the connection). `PASSWORD` requires an Active Directory server certificate and when you create the wallet, you must include the credentials for the Active Directory server account for Oracle.

For example:

```
ALTER SYSTEM SET LDAP_DIRECTORY_ACCESS = 'PASSWORD';
```

4. Set the `LDAP_DIRECTORY_SYSAUTH` parameter to `YES`, so that grants by Microsoft Active Directory administrative users to `SYSDBA`, `SYSOPER`, `SYSDG`, `SYSKM`, and `SYSRAC` are enabled.

   If you set this parameter to `NO`, then only locally authenticated users (not users authenticated through centrally managed users) can log in with these privileges.

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_SYSAUTH = YES SCOPE=SPFILE ;
   ```

5. Restart the database instance.

   - If you are in a non-multitenant environment, or if you are in a multitenant environment and in the CDB root, shut down and then restart the instance:

     ```
     SHUTDOWN IMMEDIATE
     STARTUP
     ```

   - If you are in a multitenant environment in a PDB, then close and re-open the PDB:

     ```
     ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
     ALTER PLUGGABLE DATABASE pdb_name OPEN;
     ```

After you restart the database, you can check the LDAP settings as follows:

```
show parameter ldap
```

Configuring the Access Using the Database Configuration Assistant GUI

Oracle Database Configuration Assistant (DBCA) completes the LDAP connection configuration and automatically creates the wallet and stores the Active Directory certificate for use. DBCA only works when `ldap.ora` is configured for CMU-Active Directory.

These instructions assume that you have already installed the Oracle database and that you are using an `ldap.ora` file (not `dsi.ora`) to identify the Active Directory server for the centrally managed users. If you have not installed the database yet, then you can install the software using Oracle Universal Installer (OUI) and then use DBCA to create the database at the same time as it configures the Active Directory centrally managed user connection.

1. Log in to the database server where the Oracle database is installed as a user who has administrative privileges.

2. Start `DBCA`.

   By default, the `DBCA` utility is located in the `$ORACLE_HOME/bin` directory.

   For example:

   ```
   ./dbca
   ```

3. Select the Network Configuration option (or when you get to the network configuration option when creating the database).

   The Specify Network Configuration Details window appears. If the Directory service integration area is not visible, then the `ldap.ora` file was not configured correctly. Check the `ldap.ora` configuration that you did earlier, and after you have corrected the file, rerun DBCA.

4. In the Directory service integration area, do the following:
   - In the **Service username** field, enter the name of the Oracle service directory user account, and then provide this user's password in the `Password` field.
   - In the **Service user DN** field, enter the DN for the Oracle service directory user account. The DN can be retrieved directly from the Active Directory server or from an Active Directory system administrator.
   - For **Access Type**, select the type of authentication from the list (for example, **PASSWORD**). (This setting sets the `LDAP_DIRECTORY_ACCESS` parameter.) If necessary, select the **Allow admin privileges authentication** checkbox, which allows Active Directory users to authenticate and use database schemas with administrative privileges (that is, `SYSOPER` and `SYSBACKUP`). Otherwise, only locally authenticated users can login and use administrative privileges. (This setting corresponds to the `LDAP_DIRECTORY_SYSAUTH` parameter.
   - Provide the path to the Active Directory certificate in the **Certificate file location** field. In a multitentant environment, DBCA recognizes and sets up Active Directory connections for the database instance connection. You must manually configure PDB connections if you want to connect a different Active Directory server to a PDB.
   - In the **Wallet password** and **Confirm password** fields, enter and confirm the password for the Oracle wallet that will store the Oracle service user credentials and authentication. Afterwards, DBCA automatically creates the wallet and insert the certificate.
5. Click **Next** until you reach the Finish page.
6. Click **Finish**.

Configuring the Access Using Database Configuration Assistant Silent Mode

DBCA silent mode can create a new database or alter an existing database for the Microsoft Active Directory Services-Oracle Database integration.

1. Log in to the database instance that will have the Oracle database to be used for the integration.
2. Run Database Configuration Assistant (DBCA) in silent mode.

   For example:

   ```
   ./dbca -silent -configureDatabase -sourceDB ad
   -registerWithDirService true -dirServiceUserName
   cn=example,cn=users,dc=adintg,dc=examplecorp,dc=com
   -dirServiceUser oracle -dirServicePassword password
   -ldapDirectoryAccessType PASSWORD -walletPassword password
   ```

Step 8: Verify the Oracle Wallet

The `orapki` utility can verify that the wallet for this database was created successfully.

1. Log in to the database instance that was used in the integration.
2. Go to the directory that contains the wallet.

In a non-multitenant environment, the `wallet` directory is in the `$ORACLE_BASE/admin/`*`db_unique_name`* directory.

In a multitenant environment, it is in `$ORACLE_BASE/admin/`*`db_unique_name`*`/`*`PDB_GUID`*.

3. At the command line, enter the following commands:

   `ls l wallet` (to check that the `wallet` directory contains wallet files)

   `orapki wallet display -wallet wallet` (to find the Oracle Secret Store entries)

   The output should contain the following entries:

   ```
   Requested Certificates:
   User Certificates:
   Oracle Secret Store entries:
   ORACLE.SECURITY.DN
   ORACLE.SECURITY.PASSWORD
   ORACLE.SECURITY.USERNAME
   Trusted Certificates:
   Subject: CN=ADSVR,DC=production,DC=examplecorp,DC=com
   ```

Step 9: Test the Integration

To test the integration, you must set the `ORACLE_HOME`, `ORACLE_BASE`, and `ORACLE_SID` variables and then verify the LDAP parameter settings.

1. Log in to database instance that was used for the integration.
2. Set the `ORACLE_HOME`, `ORACLE_BASE`, and `ORACLE_SID` variables.

   For example:

   ```
   export ORACLE_HOME=/app/product/18.1/dbhome_1
   export ORACLE_BASE=/app
   export ORACLE_SID=sales_db
   ```

3. Log in to the database instance as a user who has the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

4. Check the LDAP parameter settings:

   ```
   show parameter ldap
   ```

   The output should be similar to the following:

   ```
   NAME                          TYPE       VALUE
   ---------------------------   ---------  -----------------
   ldap_directory_access         string     PASSWORD
   ldap_directory_sysauth        string     yes
   ```

Configuring Authentication for Centrally Managed Users

You can configure password authentication, Kerberos authentication, or public key infrastructure (PKI) authentication.

Configuring Password Authentication for Centrally Managed Users

Configuring password authentication for centrally managed users entails the use of a password filter with Active Directory to store Oracle Database password verifiers.

About Configuring Password Authentication for Centrally Managed Users

To configure password authentication, you must deploy a password filter, set a password file, extend the Active Directory schema, and create groups for password verifiers.

For password authentication, because Oracle Database does not use `ldapbind` command to connect with Active Directory, you must install an Oracle filter and extend the Active Directory schema. The Oracle filter that you install in Active Directory creates Oracle-specific password verifiers when Active Directory users update their passwords. The Oracle filter does not generate all required Oracle password verifiers when it is first installed; the Oracle filter only generates the Oracle password verifier for a user when the user changes their Active Directory password.

To maintain backwards compatibility (if your site requires it), the Oracle filter can generate password verifiers to work with Oracle Database clients for releases 11g, 12c, and 18c. The Oracle password filter uses Active Directory groups named `ORA_VFR_MD5` (for `WebDAV`), `ORA_VFR_11G` (for release 11g) and `ORA_VFR_12C` (for releases 12c and 18c) to determine which Oracle Database password verifiers to generate. These groups must be created in Active Directory for the Oracle password verifiers to be generated for group member users. These are separate groups that dictate which specific verifiers should be generated for the Active Directory users. For example, if ten directory users need to log in to a newly created Oracle Database release 18c database that only communicated with Oracle Database release 18c and 12c clients, then an Active Directory group `ORA_VFR_12C` will have ten Active Directory users as members. The Oracle filter will only generate 12C verifiers for these ten Active Directory users when they change passwords with Active Directory (18c verifiers are the same as 12c verifiers)

Configuring Password Authentication for a Centrally Managed User

You must perform the password authentication configuration in both the Active Directory server and the Oracle database.

1. Deploy the Oracle Database password filter and extend the Active Directory schema.

   The utility tool for performing this task, `opwdintg.exe`, is located in `$ORACLE_HOME/bin`. This utility installs the password filter in Active Directory, extends the Active Directory schema to hold the Oracle password verifiers , and creates the Active Directory password filter groups. The password filter will enable the Microsoft Active Directory user accounts to be authenticated by the Oracle database when connected to clients using `WebDAV`, `11G`, and `12C` password verifiers.

   a. To deploy the `opwdintg.exe` executable, copy this file to the Active Directory server and then have the Active Directory administrator run the `opwdintg.exe` utility tool.

   b. Log in to Microsoft Active Directory as a user who has privileges to create and manage user groups.

c. Check for the following password filter user groups: `ORA_VFR_MD5`, `ORA_VFR_11G`, and `ORA_VFR_12C`. If these groups do not exist, then rerun the `opwdintg.exe` utility tool.

d. Add the Microsoft Active Directory users who will use Oracle Database to these groups, following these guidelines:

   - If either the client or the server only permits Oracle Database release 12*c* authentication, then add the user to the `ORA_VFR_12C` group. (Oracle Database release 18c uses the same verifier as Oracle Database release 12c.)
   - If both the client and the server only permit authentication lower than Oracle Database release 12*c* (that is, they have Oracle Database releases 11g, or 12.1.0.1 clients), then add the user to the `ORA_VFR_11G` group.
   - If a user must authenticate through an Oracle Database `WebDAV` client, then the user must be a member of the `ORA_VFR_MD5` group.

   This configuration enables fine-grained control over the generation of the Oracle Database password verifiers. Only the required verifiers for the required users are generated. For example, if Microsoft Active Directory user `pfitch` is added to the `ORA_VFR_12C` and `ORA_VFR_11G` groups, then both the `12C` and `11G` verifiers will be generated for `pfitch`. This ensures that when applicable, the most secure and strongest verifier is chosen, while in other cases, the `11G` verifier is chosen for the Oracle Database release 11g clients.

2. Update the database password file to version 12.2.

   a. As a user with administrative privileges, log in to the server where the database that is to be used for the Microsoft Access Directory connection resides.

   b. Go to the `$ORACLE_HOME/dbs` directory.

   c. Run the `ORAPWD` utility to set the format to 12.2.

      For example:

```
orapwd FILE='/app/oracle/product/18.1/db_1/dbs/orapwdb181' FORMAT=12.2
```

      This setting ensures that you can grant the various administrative privileges such as `SYSOPOER` and `SYSBACKUP` to the global user.

   d. Log in to the database instance as a user who has the `ALTER SYSTEM` privilege.

   e. Set the `REMOTE_LOGIN_PASSWORDFILE` to shared:

```
ALTER SYSTEM SET REMOTE_LOGIN_PASSWORDFILE = shared;
```

   f. Restart the database instance.

```
SHUTDOWN IMMEDIATE
STARTUP
```

Logging in to an Oracle Database Using Password Authentication

For password authentication, centrally managed users users have two choices of how to log in to the database.

To log in to the database that is configured to connect to Active Directory, an Active Directory user can use the following logon user name syntax if he or she is using password authentication:

```
sqlplus / nolog
connect "Windows_domain\Active_Directory_user_name"@tnsname_of_database
Password: password
```

Alternatively, the user can use their Active Directory Windows user logon name with the DNS domain name. For example:

```
connect "Active_Directory_user_name@domain.example.com"@tnsname_of_database
Password: password
```

Configuring Kerberos Authentication for Centrally Managed Users

If you plan to use Kerberos authentication, then you must configure Kerberos in the Oracle database that will be integrated with Microsoft Active Directory.

CMU-Active Directory only supports the Microsoft Active Services Kerberos server. Other non-Active Directory Kerberos servers are not supported with CMU-Active Directory.

Configuring Authentication Using PKI Certificates for Centrally Managed Users

If you plan to use PKI certificates for the authentication of centrally managed users, then you must configure Secure Sockets Layer in the Oracle database that will be integrated with Microsoft Active Directory.

Configuring Authorization for Centally Managed Users

With centrally managed users, you can manage the authorization of Oracle Database users.

Users can be added, modified, or dropped from an organization by using Active Directory without your having to add, modify, or drop the user from every database in your organization.

About Configuring Authorization for Centrally Managed Users

You can manage user authorization for a database within Active Directory.

Most Oracle Database users will be mapped to a shared database schema (user). This minimizes the work that must be done in each Oracle database when directory users are hired, change jobs within the company, or leave the company. A directory user will be assigned to an Active Directory group that is mapped to an Oracle database global user (schema). When the user logs into the database, the database will query Active Directory to find the groups the user is a member of. If your deployment is using shared schemas, then one of the groups will map to a shared database schema and the user will be assigned to that database schema. The user will have the roles and privileges that granted to the database schema. Because multiple users will be assigned to the same shared database schema, only the minimal set of roles and privileges should be granted to the shared schema. In some cases, no privileges and roles should be granted to the shared schema. Users will be assigned the appropriate set of roles and schemas through database global roles. Global roles are mapped to Active Directory groups. This way, different users can have different roles and privileges even if they are mapped to the same database shared schema. A newly hired user will be assigned to an Active Directory group mapped to a shared schema and then to one or more additional groups mapped to global roles to gain the additional roles and privileges required to complete their tasks. The combination of shared schemas and global roles allows for centralized authorization management with minimal changes to the database operationally. The database must be initially provisioned with the set of shared schemas and global roles mapped to the appropriate Active Directory groups, but then user authorization management can happen within Active Directory.

Users can also be exclusively mapped to the database global user. This requires a new user in the database that is mapped directly to the Active Directory user. New users and departing users will require updates to each database they are members of.

Users requiring administrative privileges such as `SYSOPER` and `SYSBACKUP` cannot be granted these through global roles. Administrative privileges can only be granted to a schema and not a role. But even in these cases with administrative privileges, shared schemas can be used to provide ease of user authorization management. Using a shared schema with the `SYSOPER` privilege will allow new users to be easily added to the Active Directory group mapped to the schema with `SYSOPER` without having to create a new user schema in the database. Even if only one user is assigned to the shared schema, it can still be managed centrally.

When using global roles to grant privileges and roles to the user, remember that the maximum number of enabled roles in a session is 150.

The following types of global user mappings are supported for authorization:

- Map shared global users, in which directory users are assigned to a shared database schema (user) through the mapping of a directory group to the shared schema. The directory users that are members of the group can connect to the database through this shared schema. Use of shared schemas allows for centralized management of user authorization in Active Directory.
- Exclusive global user mappings, in which a dedicated database user is exclusively mapped to a directory user. Not as common as the shared database schema, this user is created for direct database access by using either SQL*Plus or the schema user for two-tier or three-tier applications. Oracle recommends that you grant database privileges to these users through global roles, which facilitates authorization management. However, these users also can have direct privilege grants in the Oracle database, although this is not recommended. This is because two-tier and three-tier applications can use the global user as the database schema, so the global user has the full database privileges on the schema objects as the owner.

It is common for a directory user to be a member of multiple groups. However, only one of these groups should be mapped to a shared schema.

Mapping a Directory Group to a Shared Database Global User

Most users of the database will be mapped to a shared global database user (schema) through membership in a directory group.

The Active Directory group must be created before the database global user can be mapped to it. You can add Active Directory users to the group at any time before the user needs to log in to the database. On the database side, you must have the `CREATE USER` and `ALTER USER` privileges to perform these mappings. This configuration can be used for users who have the password authentication, Kerberos authentication, and public key infrastructure (PKI) authentication methods.

You can assign users who share the same database schema for an application into an Active Directory group. A shared Oracle Database global user (that is, a shared schema) is mapped to an Active Directory group. This way, any Active Directory user of this group can log in to the database through that shared global user account. Although the database global user account is shared by group members, the Active Directory user's enterprise identity (DN) is tracked and audited inside the database.

1. Log in to the database instance as a user who has been granted the `CREATE USER` system privilege.
2. Execute the `CREATE USER` statement with the `IDENTIFIED GLOBALLY` clause.

   For example, to map a directory group named `widget_sales_group` in the Sales organization unit of the `examplecorp.com` domain to a shared database global user named `widget_sales`:

   ```
   CREATE USER widget_sales IDENTIFIED GLOBALLY AS
   'cn=widget_sales_group,ou=sales,dc=examplecorp,dc=com';
   ```

   All members of the `widget_sales_group` will be assigned to the `widget_sales` shared schema when they login to the database.

Mapping a Directory Group to a Global Role

Database global roles mapped to directory groups give member users additional privileges and roles above what they have granted through their login schemas.

1. Log in to the database instance as a user who has been granted the `CREATE ROLE` system privilege.
2. Execute the `CREATE ROLE` statement with the `IDENTIFIED GLOBALLY` clause.

   For example, to map a directory user group named `widget_sales_reporting` in the Sales organization unit of the `examplecorp.com` domain to a database global role `widget_sales_role`:

   ```
   CREATE ROLE widget_sales_role IDENTIFIED GLOBALLY AS
   'cn=widget_sales_reporting,ou=sales,dc=examplecorp,dc=com';
   ```

   In a multitenant environment, to create a common role called `c##widget_sales_role`:

   ```
   CREATE ROLE c##widget_sales_role IDENTIFIED GLOBALLY AS
   'cn=widget_sales_reporting,ou=sales,dc=examplecorp,dc=com'
   CONTAINER = ALL;
   ```

Exclusively Mapping a Directory User to a Database Global User

You can map a Microsoft Directory user exclusively to an Oracle Database global user.

You perform the configuration on the Oracle Database side only, not the Active Directory side. You must have the `CREATE USER` and `ALTER USER` privileges to perform these mappings. This configuration can be used for users who have the password authentication, Kerberos authentication, and public key infrastructure (PKI) authentication methods.

1. Log in to the database instance as a user who has been granted the `CREATE USER` system privilege.

2. Execute the `CREATE USER` statement with the `IDENTIFIED GLOBALLY` clause.

   For example, to map an existing Active Directory user named Peter Fitch in the sales organization of the `examplecorp.com` domain to use a database global user named `peter_global`:

   ```
   CREATE USER peter_fitch IDENTIFIED GLOBALLY AS
   'cn=peter fitch,ou=sales,dc=examplecorp,dc=com';
   ```

Altering or Migrating a User Mapping Definition

You can update a Directory User-to-global-user mapping by using the `ALTER USER` statement.

You can update users whose accounts were created using any of the `CREATE USER` statement clauses: `IDENTIFIED BY` *`password`*, `IDENTIFIED EXTERNALLY`, or `IDENTIFIED GLOBALLY`. This is useful when migrating users to using CMU. For example, a database user that is externally authenticated to Kerberos will be identified by their user principal name (UPN). To migrate the user to use CMU with Kerberos authentication, you would need to run the `ALTER USER` statement to declare a global user and identify the user with their Active Directory domain name.

1. Log in to the database instance as a user who has been granted the `ALTER USER` system privilege.
2. Execute the `ALTER USER` statement with the `IDENTIFIED GLOBALLY` clause.

   For example:

   ```
   ALTER USER peter_fitch IDENTIFIED GLOBALLY AS
   'cn=peter fitch,ou=widget_sales,dc=examplecorp,dc=com';
   ```

Configuring Administrative Users

Administrative users can work as they have in the past, but with CMUs, they can be controlled with centralized authentication and authorization if they are using shared schemas.

Configuring Database Administrative Users with Shared Access Accounts

Using shared accounts simplifies the management of database administrators for multiple databases as they join, move, and leave the organization.

You can assign new database administrators to shared accounts in multiple databases using Active Directory groups without having to create new Oracle database accounts.

1. Ensure that the password file for the current database instance is in the 12.2 format.

   ```
   orapwd file=pwd_file FORMAT=12.2
   Enter password for SYS: password
   ```

2. In Active Directory, create an Active Directory group (for example, for a database administrator backup users group called `ad_dba_backup_users`).
3. In Oracle Database, create a global user (shared schema) (for example, `db_dba_backup_global_user`) and map this user to the Active Directory `ad_dba_backup_users` group.

4. Grant the `SYSBACKUP` administrative privilege to the global group `db_dba_backup_global_user`.

   Now, any Active Directory user that is put into the new Active Directory group will be assigned the `SYSBACKUP` administrative privilege.

At this stage, any Active Directory user who is added to the `ad_dba_backup_users` Active Directory group will be assigned to the new database shared schema with the `SYSBACKUP` administrative privilege.

Configuring Database Administrative Users Using Exclusive Mapping

Database administrators can also be mapped in exclusive schemas in databases, which requires a separate global user to be created in each database for each exclusively mapped DBA.

1. Ensure that the password file for the current database instance is in the 12.2 format.

   ```
   orapwd file=pwd_file FORMAT=12.2
   Enter password for SYS: password
   ```

2. Log in to the database instance as a user who can create users and grant administrative privileges to other users.
3. Create this user as a global user.

   For example:

   ```
   CREATE USER peter_fitch IDENTIFIED GLOBALLY AS
   'cn=peter fitch,ou=sales,dc=examplecorp,dc=com';
   ```

4. Grant this user the administrative privilege.

   For example, to grant a user the `SYSKM` administrative privilege:.

   ```
   GRANT SYSKM TO peter_fitch;
   ```

Due to the amount of work to maintain accounts and the mapping in both the database and Active Directory, a more centralized approach would be to use shared schemas for these administrative accounts as well, even if only one Active Directory user is assigned to the shared database account in some cases.

Integration of Oracle Database with Microsoft Active Directory Password Policies

As part of the Oracle Database-Microsoft Active Directory integration, Oracle Database enforces the Active Directory password policies when Active Directory users log into the Oracle database.

As part of the Oracle Database-CMU-Active Directory integration, the Oracle Database enforces the Active Directory password policies when Active Directory users try to log into the Oracle Database. Preventing users with expired passwords and locking accounts after a specified number of failed login attempts are examples of Active Directory password policies that the Oracle database will enforce.

6

Managing Security for Definer's Rights and Invoker's Rights

Invoker’s rights and definer’s rights have several security advantages when used to control access to privileges to run user-created procedures.

About Definer's Rights and Invoker's Rights

Definer's rights and invoker's rights are used to control access to the privileges necessary to run a user-created procedure, or program unit.

In a definer's rights procedure, the procedure executes with the privileges of the owner. The privileges are bound to the schema in which they were created. An invoker's rights procedure executes with the privileges of the current user, that is, the user who invokes the procedure.

For example, suppose user `bixby` creates a procedure that is designed to modify table `cust_records` and then he grants the `EXECUTE` privilege on this procedure to user `rlayton`. If `bixby` had created the procedure with definer's rights, then the procedure would look for table `cust_records` in `bixby`'s schema. Had the procedure been created with invoker's rights, then when `rlayton` runs it, the procedure would look for table `cust_records` in `rlayton`'s schema.

By default, all procedures are considered definer's rights. You can designate a procedure to be an invoker's rights procedure by using the `AUTHID CURRENT_USER` clause when you create or modify it, or you can use the `AUTHID DEFINER` clause to make it a definer's rights procedure.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more details about definer's rights and invoker's rights procedures
- *Oracle Database Vault Administrator’s Guide* for information about how you can create privilege analysis policies that capture privilege use of definer’s rights and invoker’s rights procedures

How Procedure Privileges Affect Definer's Rights

The owner of a procedure, called the *definer*, must have the necessary object privileges for objects that the procedure references.

If the procedure owner grants to another user the right to use the procedure, then the privileges of the procedure owner (on the objects the procedure references) apply to the grantee's exercise of the procedure. The privileges of the procedure's definer must be granted directly to the procedure owner, not granted through roles. These are called definer's rights.

The user of a procedure who is not its owner is called the *invoker*. Additional privileges on referenced objects are required for an invoker's rights procedure, but not for a definer's rights procedure.

A user of a definer's rights procedure requires only the privilege to execute the procedure and no privileges on the underlying objects that the procedure accesses. This is because a definer's rights procedure operates under the security domain of the user who owns the procedure, regardless of who is executing it. The owner of the procedure must have all the necessary object privileges for referenced objects. Fewer privileges need to be granted to users of a definer's rights procedure. This results in stronger control of database access.

You can use definer's rights procedures to control access to private database objects and add a level of database security. By writing a definer's rights procedure and granting only the `EXECUTE` privilege to a user, this user can be forced to access the referenced objects only through the procedure.

At run time, Oracle Database checks whether the privileges of the owner of a definer's rights procedure allow access to that procedure's referenced objects, before the procedure is executed. If a necessary privilege on a referenced object was revoked from the owner of a definer's rights procedure, then no user, including the owner, can run the procedure.

An example of when you may want to use a definer's rights procedure is as follows: Suppose that you must create an API whose procedures have unrestricted access to its tables, but you want to prevent ordinary users from selecting table data directly, and from changing it with `INSERT`, `UPDATE`, and `DELETE` statements. To accomplish this, in a separate, low-privileged schema, create the tables and the procedures that comprise the API. By default, each procedure is a definer's rights unit, so you do not need to specify `AUTHID DEFINER` when you create it. Then grant the `EXECUTE` privilege to the users who must use this API, but do not grant any privileges that allow data access. This solution gives you complete control over your API behavior and how users have access to its underlying objects.

Oracle recommends that you create your definer's rights procedures, and views that access these procedures, in their own schema. Grant this schema very low privileges, or no privileges at all. This way, when other users run these procedures or views, they will not have access to any unnecessarily high privileges from this schema.

Note:

Trigger processing follows the same patterns as definer's rights procedures. The user runs a SQL statement, which that user is privileged to run. As a result of the SQL statement, a trigger is fired. The statements within the triggered action temporarily execute under the security domain of the user that owns the trigger. For overview information about triggers, *Oracle Database Concepts*.

How Procedure Privileges Affect Invoker's Rights

An invoker’s rights procedure executes with all of the invoker's privileges.

Oracle Database enables the privileges that were granted to the invoker through any of the invoker's enabled roles to take effect, unless a definer's rights procedure calls the invoker's rights procedure directly or indirectly. A user of an invoker's rights procedure must have privileges (granted to the user either directly or through a role) on objects that the procedure accesses through external references that are resolved in the schema of the invoker. When the invoker runs an invoker's rights procedure, this user temporarily has *all* of the privileges of the invoker.

The invoker must have privileges at run time to access program references embedded in DML statements or dynamic SQL statements, because they are effectively recompiled at run time.

For all other external references, such as direct PL/SQL function calls, Oracle Database checks the privileges of the owner at compile time, but does not perform a run-time check. Therefore, the user of an invoker's rights procedure does not need privileges on external references outside DML or dynamic SQL statements. Alternatively, the developer of an invoker's rights procedure must only grant privileges on the procedure itself, not on all objects directly referenced by the invoker's rights procedure.

You can create a software bundle that consists of multiple program units, some with definer's rights and others with invoker's rights, and restrict the program entry points *(controlled step-in)*. A user who has the privilege to run an entry-point procedure can also execute internal program units indirectly, but cannot directly call the internal programs. For very precise control over query processing, you can create a PL/SQL package specification with explicit cursors.

When You Should Create Invoker's Rights Procedures

Oracle recommends that you create invoker's rights procedures in certain situations.

These situations are as follows:

- **When creating a PL/SQL procedure in a high-privileged schema.** When lower-privileged users invoke the procedure, then it can do no more than those users are allowed to do. In other words, the invoker's rights procedure runs with the privileges of the invoking user.
- **When the PL/SQL procedure contains no SQL and is available to other users.** The `DBMS_OUTPUT` PL/SQL package is an example of a PL/SQL subprogram that contains no SQL and is available to all users. The reason you should use an invoker's rights procedure in this situation is because the unit issues no SQL statements at run time, so the run-time system does not need to check their privileges. Specifying `AUTHID CURRENT_USER` makes invocations of the procedure more efficient, because when an invoker's right procedure is pushed onto, or comes from, the call stack, the values of `CURRENT_USER` and `CURRENT_SCHEMA`, and the currently enabled roles do not change.

See Also:

- Configuration of Oracle Virtual Private Database Policies
- About ANY Privileges and the PUBLIC Role
- *Oracle Database PL/SQL Packages and Types Reference* for information about how Oracle Database handles name resolution and privilege checking at runtime using invoker's and definer's rights
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the differences between invoker's rights and definer's rights units
- *Oracle Database PL/SQL Packages and Types Reference* for information about defining explicit cursors in the `CREATE PACKAGE` statement

Controlling Invoker's Rights Privileges for Procedure Calls and View Access

The `INHERIT PRIVILEGES` and `INHERIT ANY PRIVILEGES` privileges regulate the privileges used when invoker's rights procedures are run.

How the Privileges of a Schema Affect the Use of Invoker's Rights Procedures

An invoker’s rights procedure is useful in situations where a lower-privileged user must execute a procedure owned by a higher-privileged user.

When a user runs an invoker's rights procedure (or any PL/SQL program unit that has been created with the `AUTHID CURRENT_USER` clause), the procedure temporarily inherits all of the privileges of the invoking user while the procedure runs.

During that time, the procedure owner has, through the procedure, access to this invoking user's privileges. Consider the following scenario:

1. User `ebrown` creates the `check_syntax` invoker's rights procedure and then grants user `jward` the `EXECUTE` privilege on it.
2. User `ebrown`, who is a junior programmer, has only the minimum set of privileges necessary for his job. The `check_syntax` procedure resides in `ebrown`'s schema.
3. User `jward`, who is a manager, has a far more powerful set of privileges than user `ebrown`.
4. When user `jward` runs the `check_syntax` invoker's rights procedure, the procedure inherits user `jward`'s higher privileges while it runs.
5. Because user `ebrown` owns the `check_syntax` procedure, he has access to user `jward`'s privileges whenever `jward` runs the `check_syntax` procedure.

The danger in this type of situation—in which the lower privileged `ebrown`'s procedure has access to `jward`'s higher privileges whenever `jward` runs the procedure—lies in the risk that the procedure owner can misuse the higher privileges of the invoking user. For example, user `ebrown` could make use of `jward`'s higher privileges by rewriting the `check_syntax` procedure to give `ebrown` a raise or delete `ebrown`'s bad performance appraisal record. Or, `ebrown` originally could have created the procedure as a definer's rights procedure, granted its `EXECUTE` privilege to `jward`, and then later on change it to a potentially malicious invoker's rights procedure without letting `jward` know. These types of risks increase when random users, such as application users, have access to a database that uses invoker's rights procedures.

When user `jward` runs `ebrown`'s invoker's rights procedure, there is an element of trust involved. He must be assured that `ebrown` will not use the `check_syntax` procedure in a malicious way when it accesses `jward`'s privileges. The `INHERIT PRIVILEGES` and `INHERIT ANY PRIVILEGES` privileges can help user `jward` control whether user `ebrown`'s procedure can have access to his (`jward`'s) privileges. Any user can grant or revoke the `INHERIT PRIVILEGES` privilege on themselves to the user whose invoker's rights procedures they want to run. `SYS` users manage the `INHERIT ANY PRIVILEGES` privilege.

How the INHERIT [ANY] PRIVILEGES Privileges Control Privilege Access

Use the `INHERIT PRIVILEGES` and `INHERIT ANY PRIVILEGES` privileges to secure invoker's rights procedures.

The `INHERIT PRIVILEGES` and `INHERIT ANY PRIVILEGES` privileges regulate the privileges used when a user runs an invoker's rights procedure or queries a `BEQUEATH CURRENT_USER` view that references an invoker's rights procedure.

When a user runs an invoker's rights procedure, Oracle Database checks it to ensure that the procedure owner has either the `INHERIT PRIVILEGES` privilege on the invoking user, or if the owner has been granted the `INHERIT ANY PRIVILEGES` privilege. If the privilege check fails, then Oracle Database returns an `ORA-06598: insufficient INHERIT PRIVILEGES privilege` error.

The benefit of these two privileges is that they give invoking users control over who can access their privileges when they run an invoker's rights procedure or query a `BEQUEATH CURRENT_USER` view.

Grants of the INHERIT PRIVILEGES Privilege to Other Users

By default, all users are granted `INHERIT PRIVILEGES ON USER` *`newuser`* `TO PUBLIC`.

This grant takes place when the user accounts are created or when accounts that were created earlier are upgraded to the current release.

The invoking user can revoke the `INHERIT PRIVILEGE` privilege from other users on himself and then grant it only to users that he trusts.

The syntax for the `INHERIT PRIVILEGES` privilege grant is as follows:

```
GRANT INHERIT PRIVILEGES ON USER invoking_user TO procedure_owner;
```

In this specification:

- *`invoking_user`* is the user who runs the invoker's rights procedure. This user must be a database user account.

- `procedure_owner` is the user who owns the invoker's rights procedure. This value must be a database user account. As an alternative to granting the `INHERIT PRIVILEGES` privilege to the procedure's owner, you can grant the privilege to a role that is in turn granted to the procedure.

The following users or roles must have the `INHERIT PRIVILEGES` privilege granted to them by users who will run their invoker's rights procedures:

- Users or roles who own the invoker's rights procedures
- Users or roles who own `BEQUEATH CURRENT_USER` views

Example: Granting INHERIT PRIVILEGES on an Invoking User

The `GRANT` statement can grant the `INHERIT PRIVILEGES` privilege on an invoking user to a procedure owner.

Example 6-1 shows how the invoking user `jward` can grant user `ebrown` the `INHERIT PRIVILEGES` privilege.

Example 6-1 Granting INHERIT PRIVILEGES on an Invoking User to a Procedure Owner

```
GRANT INHERIT PRIVILEGES ON USER jward TO ebrown;
```

The statement enables any invoker's rights procedure that `ebrown` writes, or will write in the future, to access `jward`'s privileges when `jward` runs it.

Example: Revoking INHERIT PRIVILEGES

The `REVOKE` statement can revoke the `INHERIT PRIVILEGES` privilege from a user.

Example 6-2 shows how user `jward` can revoke the use of his privileges from `ebrown`.

Example 6-2 Revoking INHERIT PRIVILEGES

```
REVOKE INHERIT PRIVILEGES ON USER jward FROM ebrown;
```

Grants of the INHERIT ANY PRIVILEGES Privilege to Other Users

By default, user `SYS` has the `INHERIT ANY PRIVILEGES` system privilege and can grant this privilege to other database users or roles.

As with all `ANY` privileges, only grant this privilege to trusted users or roles. Once a user or role has been granted the `INHERIT ANY PRIVILEGES` privilege, then this user's invoker's rights procedures have access to the privileges of the invoking user. You can find the users who have been granted the `INHERIT ANY PRIVILEGES` privilege by querying the `DBA_SYS_PRIVS` data dictionary view.

Example: Granting INHERIT ANY PRIVILEGES to a Trusted Procedure Owner

The `GRANT` statement can grant the `INHERIT ANY PRIVILEGES` privilege to trusted procedure owners.

Example 6-3 shows how to grant the `INHERIT ANY PRIVILEGES` privilege to user `ebrown`.

Example 6-3 Granting INHERIT ANY PRIVILEGES to a Trusted Procedure Owner

```
GRANT INHERIT ANY PRIVILEGES TO ebrown;
```

Be careful about revoking the `INHERIT ANY PRIVILEGES` privilege from powerful users. For example, suppose user `SYSTEM` has created a set of invoker's rights procedures. If you revoke `INHERIT ANY PRIVILEGES` from `SYSTEM`, then other users cannot run his procedures, unless they have specifically granted him the `INHERIT PRIVILEGE` privilege.

Managing INHERIT PRIVILEGES and INHERIT ANY PRIVILEGES

By default, `PUBLIC` has the `INHERIT PRIVILEGE` privilege on new and upgraded user accounts; the `SYS` user has the `INHERIT ANY PRIVILEGES` privilege.

Oracle by default configures a set of grants of `INHERIT PRIVILEGES` that are designed to help protect against misuse of the privileges of various Oracle-defined users.

You can choose to revoke the default grant of `INHERIT PRIVILEGES ON USER` *`user_name`* `TO PUBLIC` for a customer-defined user and grant more specific grants of `INHERIT PRIVILEGES` as appropriate for that particular user. To find the users who have been granted the `INHERIT ANY PRIVILEGES` privilege, query the `DBA_SYS_PRIVS` data dictionary view.

1. Revoke the `INHERIT PRIVILEGES` privilege from `PUBLIC`.

   For example:

   ```
   REVOKE INHERIT PRIVILEGES ON invoking_user FROM PUBLIC;
   ```

   Be aware that this time, any users who run invoker's rights procedures cannot do so, due to run-time errors from failed `INHERIT PRIVILEGES` checks.
2. Selectively grant the `INHERIT PRIVILEGES` privilege to trusted users or roles.
3. Similarly, selectively grant the `INHERIT ANY PRIVILEGES` privilege only to trusted users or roles.

You can create an audit policy to audit the granting and revoking of these two privileges, but you cannot audit run-time errors that result from failed `INHERIT PRIVILEGES` privilege checks.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about SQL injection attacks
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `GRANT` statement and default privileges

Definer's Rights and Invoker's Rights in Views

The `BEQEATH` clause in the `CREATE VIEW` SQL statement can control definer's rights and invoker's rights in user-created views.

About Controlling Definer's Rights and Invoker's Rights in Views

You can configure user-defined views to accommodate invoker's rights functions that are referenced in the view.

When a user invokes an identity- or privilege-sensitive SQL function or an invoker's rights PL/SQL or Java function, then current schema, current user, and currently enabled roles within the operation's execution can be inherited from the querying user's environment, rather than being set to the owner of the view.

This configuration does not turn the view itself into an invoker's rights object. Name resolution within the view is still handled using the view owner's schema, and privilege checking for the view is done using the view owner's privileges. However, at runtime, the function referenced by view runs under the invoking user's privileges rather than those of the view owner's.

The benefit of this feature is that it enables functions such as `SYS_CONTEXT` and `USERENV`, which must return information accurate for the invoking user, to return consistent results when these functions are referenced in a view.

Using the BEQUEATH Clause in the CREATE VIEW Statement

The `BEQUEATH` controls how an invoker’s right function can be executed using the rights of the invoking user.

To enable an invoker's rights function to be executed using the rights of the user issuing SQL that references the view, in the `CREATE VIEW` statement, you can set the `BEQUEATH` clause to `CURRENT_USER`.

If you plan to issue a SQL query or DML statement against the view, then the view owner must be granted the `INHERIT PRIVILEGES` privilege on the invoking user or the view owner must have the `INHERIT ANY PRIVILEGES` privilege. If not, then when a `SELECT` query or DML statement involves a `BEQUEATH CURRENT_USER` view, the run-time system will raise error `ORA-06598: insufficient INHERIT PRIVILEGES privilege`.

- Use the use `BEQUEATH CURRENT_USER` clause to set the view's function to be executed using invoker's rights.

For example:

```
CREATE VIEW MY_OBJECTS_VIEW BEQUEATH CURRENT_USER AS
 SELECT GET_OBJS_FUNCTION;
```

If you want the function within the view to be executed using the view owner's rights, then you should either omit the `BEQUEATH` clause or set it to `DEFINER`.

For example:

```
CREATE VIEW my_objects_view BEQUEATH DEFINER AS
 SELECT OBJECT_NAME FROM USER_OBJECTS;
```

See Also:

- Controlling Invoker's Rights Privileges for Procedure Calls and View Access for more information about how the `INHERIT PRIVILEGE` privilege works
- *Oracle Database SQL Language Reference* for additional information about granting the `INHERIT PRIVILEGES` and `INHERIT ANY PRIVILEGES` privileges
- *Oracle Database Real Application Security Administrator's and Developer's Guide* for information about how to use `BEQUEATH CURRENT_USER` views with Oracle Database Real Application Security applications

Finding the User Name or User ID of the Invoking User

PL/SQL functions can be used to find the invoking user, based on whether invoker’s rights or definer’s rights are being used.

- Use the `ORA_INVOKING_USER` or `ORA_INVOKING_USERID` function to find the invoking user based on whether invoker's rights or definer's rights:
  - `ORA_INVOKING_USER`: Use this function to return the name of the user who is invoking the current statement or view. This function treats the intervening views as specified by their `BEQUEATH` clauses. If the invoking user is an Oracle Database Real Application Security-defined user, then this function returns `XS$NULL`.
  - `ORA_INVOKING_USERID`: Use this function to return the identifier (ID) of the user who is invoking the current statement or view. This function treats the intervening views as specified by their `BEQUEATH` clauses. If the invoking user is an Oracle Database Real Application Security-defined user, then this function returns an ID that is common to all Real Application Security sessions but is different from the ID of any database user.

    For example:

    ```
    CONNECT HR
    Enter password: password

    SELECT ORA_INVOKING_USER FROM DUAL;

    ORA_INVOKING_USER
    --------------------
    HR
    ```

See Also:

Oracle Database Real Application Security Administrator's and Developer's Guide for information about similar functions that are used for Oracle Database Real Application Security applications

Finding BEQUEATH DEFINER and BEQUEATH_CURRENT_USER Views

You can find out if a view is a `BEQUEATH DEFINER` or `BEQUEATH CURRENT_USER` view.

- To find if a view is `BEQUEATH DEFINER` or `BEQUEATH CURRENT_USER` view, query the `BEQUEATH` column of a `*_VIEWS` or `*_VIEWS_AE` static data dictionary view for that view.

See Also:

- *Oracle Database Reference* for more information about `*_VIEWS` static data dictionary views
- *Oracle Database Reference* for more information about `*_VIEWS_AE` static data dictionary views

For example:

```
SELECT BEQUEATH FROM USER_VIEWS WHERE VIEW_NAME = 'MY_OBJECTS';

BEQUEATH
------------
CURRENT_USER
```

Using Code Based Access Control for Definer's Rights and Invoker's Rights

Code based access control, used to attach database roles to PL/SQL functions, procedures, or packages, works well with invoker's rights and definer's procedures.

About Using Code Based Access Control for Applications

You can use code based access control (CBAC) to better manage definer’s rights program units.

Applications must often run program units in the caller's environment, while requiring elevated privileges. PL/SQL programs traditionally make use of definer's rights to temporarily elevate the privileges of the program.

However, definer's rights based program units run in the context of the definer or the owner of the program unit, as opposed to the invoker's context. Also, using definer's rights based programs often leads to the program unit getting more privileges than required.

Code based access control (CBAC) provides the solution by enabling you to attach database roles to a PL/SQL function, procedure, or package. These database roles are enabled at run time, enabling the program unit to execute with the required privileges in the calling user's environment.

See Also:

Oracle Database Vault Administrator’s Guide for information about how you can create privilege analysis policies that capture privilege use of CBAC roles

Who Can Grant Code Based Access Control Roles to a Program Unit?

Code based access control roles can be granted to a program unit if a set of conditions are met.

These conditions are as follows:

- The grantor is user `SYS` or owns the program unit.
- If the grantor owns the program unit, then the grantor must have the `GRANT ANY ROLE` system privilege, or have the `ADMIN` or `DELEGATE` option for the roles that they want to grant to program units.
- The roles to be granted are directly granted roles to the owner.
- The roles to be granted are standard database roles.

If these three conditions are not met, then error `ORA-28702: Program unit string is not owned by the grantor` is raised if the first condition is not met, and error `ORA-1924: role 'string' not granted or does not exist` is raised if the second and third conditions are not met.

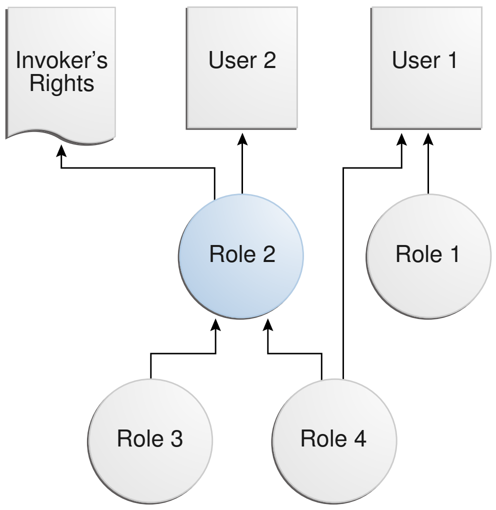
How Code Based Access Control Works with Invoker's Rights Program Units

Code based access control can run a program unit in an invoking user's context and with roles associated with this context.

Consider a scenario where there are two application users, `1` and `2`. Application user `2` creates the invoker's right program unit, grants database role `2` to the invoker's rights unit, and then grants execute privileges on the invoker's rights unit to application user `1`.

Figure 6-1 shows the database roles `1` and `2` granted to application users `1` and `2`, and an invoker's right program unit.

Figure 6-1 Roles Granted to Application Users and Invoker's Right Program Unit



The grants are as follows:

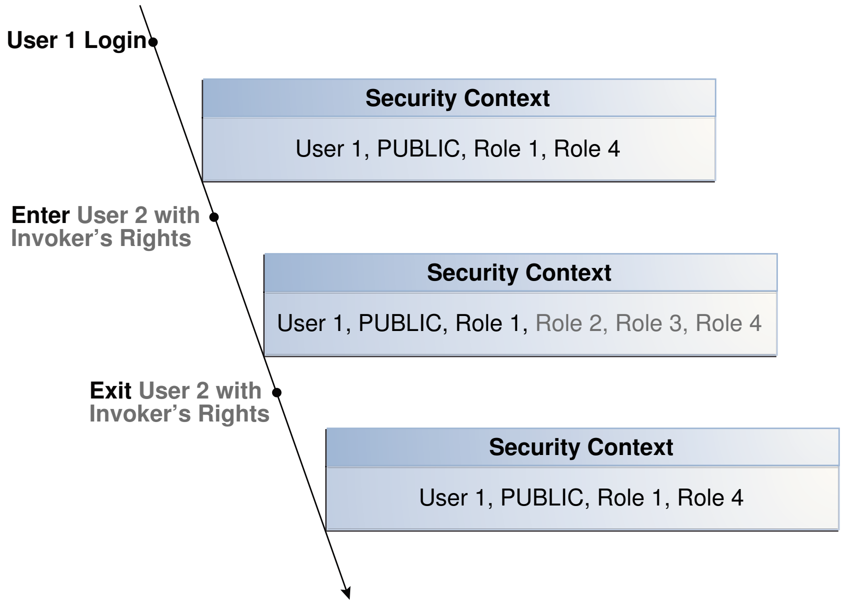
- Application user `1` is directly granted database roles `1` and `4`.
- Application user `2` is directly granted database role `2`, which includes application roles `3` and `4`.
- The invoker's right program unit is granted database role `2`.

When application user `1` logs in and executes the invoker's rights program unit, then the invoker's rights unit executes with the combined database roles of user `1` and the database roles attached to the invoker's rights unit.

Figure 6-2 shows the security context in which the invoker's rights unit is executed. When application user `1` first logs on, application user `1` has the database `PUBLIC` role (by default), and the database roles `1` and `4`, which have been granted to it. Application user `1` next executes the invoker's rights program unit created by application user `2`.

The invoker's rights unit executes in application user `1`'s context, and has the additional database role `2` attached to it. Database roles `3` and `4` are included, as they are a part of database role `2`. After the invoker's rights unit exits, then application user `1` only has the application roles that have been granted to it, `PUBLIC`, role `1`, and role `4`.

Figure 6-2 Security Context in Which Invoker's Right Program Unit IR Is Executed


User 1 Login
Security Context
User 1, PUBLIC, Role 1, Role 4
Enter User 2 with Invoker's Rights
Security Context
User 1, PUBLIC, Role 1, Role 2, Role 3, Role 4
Exit User 2 with Invoker's Rights
Security Context
User 1, PUBLIC, Role 1, Role 4

How Code Based Access Control Works with Definer's Rights Program Units

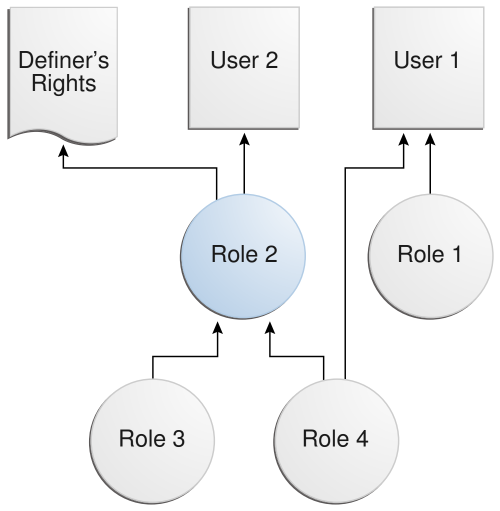
Code based access control can be used to secure definer’s rights.

Code based access control works with definer's rights program units to enable the program unit to run using the defining user's rights, with the privileges of a combined set of database roles that are associated with this user.

Consider a scenario where application user `2` creates a definer's rights program unit, grants role `2` to the definer's rights program unit, and then grants the `EXECUTE` privilege on the definer's rights program unit to application user `1`.

Figure 6-3 shows the database roles granted to application users `1` and `2`, and a definer's rights program unit.

Figure 6-3 Roles Granted to Application Users and Definer's Rights Program Unit



The grants are as follows:

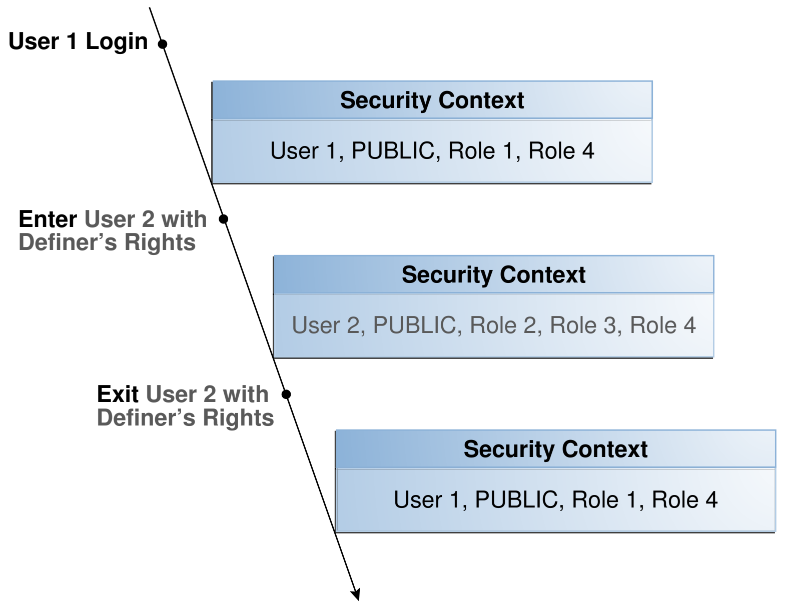
- Application user `1` is directly granted database roles `1` and `4`.
- Application user `2` is directly granted database role`2`, which includes database roles `3` and `4`.
- The definer's right program unit is granted database role `2`.

When application user `1` logs in and executes definer's right program unit, then the definer's rights unit executes with the combined database roles of application user `2` and the database roles attached to the definer's rights unit (roles `2`, `3`, and `4`).

Figure 6-4 shows the security context in which the definer's right program unit is executed. When application user `1` first logs on, application user `1` has the database `PUBLIC` role (by default), and the database roles `1` and`4`, which have been granted to it. Application user `1` next executes the definer's rights program unit created by application user `2`.

The definer's rights program unit executes in application user `2`'s context, and has the additional database role `2` attached to it. Database roles `3` and `4` are included, as they are a part of database role `2`. After the definer's rights unit exits, application user `1` only has the database roles that have been granted to it (`PUBLIC`, role `1`, and role `4`).

Figure 6-4 Security Context in Which Definer's Right Program Unit DR Is Executed


User 1 Login
Security Context
User 1, PUBLIC, Role 1, Role 4
Enter User 2 with Definer's Rights
Security Context
User 2, PUBLIC, Role 2, Role 3, Role 4
Exit User 2 with Definer's Rights
Security Context
User 1, PUBLIC, Role 1, Role 4

Grants of Database Roles to Users for Their CBAC Grants

The `DELEGATE` option in the `GRANT` statement can limit privilege grants to roles by users responsible for CBAC grants.

When you grant a database role to a user who is responsible for CBAC grants, you can include the `DELEGATE` option in the `GRANT` statement to prevent giving the grantee additional privileges on the roles.

The `DELEGATE` option enables the roles to be granted to program units, but it does not permit the granting of the role to other principals or the administration of the role itself. You also can use the `ADMIN` option for the grants, which does permit the granting of the role to other principals. Both the `ADMIN` and `DELEGATE` options are compatible; that is, you can grant both to a user, though you must do this in separate `GRANT` statements for each option. To find if a user has been granted a role with these options, query the `DELEGATE_OPTION` column or the `ADMIN_OPTION` column of either the `USER_ROLE_PRIVS` or `DBA_ROLE_PRIVS` for the user.

The syntax for using the `DELEGATE` and `ADMIN` option is as follows:

```
GRANT role_list to user_list WITH DELEGATE OPTION;

GRANT role_list to user_list WITH ADMIN OPTION;
```

For example:

```
GRANT cb_role1 to usr1 WITH DELEGATE OPTION;

GRANT cb_role1 to usr1 WITH ADMIN OPTION;

GRANT cb_role1, cb_role2 to usr1, usr2 with DELEGATE OPTION;

GRANT cb_role1, cb_role2 to usr1, usr2 with ADMIN OPTION;
```

In a multitenant environment, you can use the `DELEGATE` option for common grants such as granting common roles to common users, just as you can with the `ADMIN` option.

For example:

```
GRANT c##cb_role1 to c##usr1 WITH DELEGATE OPTION CONTAINER = ALL;
```

Be aware that CBAC grants themselves can only take place locally in a PDB.

See Also:

Oracle Database SQL Language Reference for more information about the `ADMIN` option

Grants and Revokes of Database Roles to a Program Unit

The `GRANT` and `REVOKE` statements can grant database roles to or revoke database roles from a program unit.

The following syntax to grants or revokes database roles for a PL/SQL function, procedure, or package:

```
GRANT role_list TO code_list
REVOKE {role_list | ALL} FROM code_list
```

In this specification:

```
role_list ::=  code-based_role_name[, role_list]
code_list ::=  {
      {FUNCTION  [schema.]function_name}
   |  {PROCEDURE [schema.]procedure_name}
   |  {PACKAGE   [schema.]package_name}
                 }[, code_list]
```

For example:

```
GRANT cb_role1 TO FUNCTION func1, PACKAGE pack1;

GRANT cb_role2, cb_role3 TO FUNCTION HR.func2, PACKAGE SYS.pack2;

REVOKE cb_role1 FROM FUNCTION func1, PACKAGE pack1;

REVOKE ALL FROM FUNCTION HR.func2, PACKAGE SYS.pack2;
```

Tutorial: Controlling Access to Sensitive Data Using Code Based Access Control

This tutorial demonstrates how to control access to sensitive data in the `HR` schema by using code based access control.

About This Tutorial

In this tutorial, you will create a user who must have access to specific employee information for his department.

However, the table `HR.EMPLOYEES` contains sensitive information such as employee salaries, which must not be accessible to the user. You will implement access control using code based access control. The employee data will be shown to the user through an invoker's rights procedure. Instead of granting the `SELECT` privilege directly to the user, you will grant the `SELECT` privilege to the invoker's rights procedure through a database role. In the procedure, you will hide the sensitive information, such as salaries. Because the procedure is an invoker's rights procedure, you know the caller's context inside the procedure. In this case, the caller's context is for the Finance department. The user is named `"Finance"`, so that only data for employees who work in the Finance department is accessible to the user.

Step 1: Create the User and Grant HR the CREATE ROLE Privilege

To begin, you must create the `"Finance"` user account and then grant this the `HR` user the `CREATE ROLE` privilege.

1. Log into the database instance as an administrator who has privileges to create user accounts and roles.

   For example:

   ```
   sqlplus sec_admin
   Enter password: password
   ```

2. Create the `"Finance"` user account.

   ```
   GRANT CONNECT TO "Finance" IDENTIFIED BY password;
   ```

   Ensure that you enter `"Finance"` in the case shown, enclosed by double quotation marks. Follow the guidelines in Minimum Requirements for Passwords to replace *`password`* with a password that is secure.

3. Grant the `CREATE ROLE` privilege to user `HR`.

   ```
   GRANT CREATE ROLE TO HR;
   ```

Step 2: Create the print_employees Invoker's Rights Procedure

The `print_employees` invoker's rights procedure shows employee information in the current user's department.

You must create this procedure as an invoker's rights procedure because you must know who the caller is when inside the procedure.

1. Connect as user `HR`.

   ```
   CONNECT HR
   Enter password: password
   ```

2. Create the `print_employees` procedure as follows.

   ```
   create or replace procedure print_employees
   authid current_user
   as
   ```

```
begin
  dbms_output.put_line(rpad('ID', 10) ||
                       rpad('First Name', 15)  ||
                       rpad('Last Name', 15)   ||
                       rpad('Email', 15)       ||
                       rpad('Phone Number', 20));
  for rec in (select e.employee_id, e.first_name, e.last_name,
                     e.email, e.phone_number
                from hr.employees e, hr.departments d
               where e.department_id = d.department_id
                 and d.department_name =
                     sys_context('userenv', 'current_user'))
  loop
    dbms_output.put_line(rpad(rec.employee_ID, 10)  ||
                         rpad(rec.first_name, 15)   ||
                         rpad(rec.last_name, 15)    ||
                         rpad(rec.email, 15)        ||
                         rpad(rec.phone_number, 20));
  end loop;
end;
/
```

In this example:

- `dbms_output.put_line` prints the table header.
- `for rec in (select ...` finds the employee information for the caller's department, which for this tutorial is the Finance department for user `"Finance"`. Had you created a user named `"Marketing"` (which is also listed in the `DEPARTMENT_NAME` column of the `HR.EMPLOYEES` table), then the procedure could capture information for Marketing employees.
- `loop` and `dbms_output.put_line` populate the output with the employee data from the Finance department.

Step 3: Create the hr_clerk Role and Grant Privileges for It

Next, you are ready to create the `hr_clerk` role, which must have the `EXECUTE` privilege on the `print_employees` procedure.

After you create this role, you must grant it to `"Finance"`.

1. Create the `hr_clerk` role.

   ```
   CREATE ROLE hr_clerk;
   ```

2. Grant the `EXECUTE` privilege on the print_employees procedure to the `hr_clerk` role.

   ```
   GRANT EXECUTE ON print_employees TO hr_clerk;
   ```

3. Grant the `hr_clerk` role to `"Finance"`.

   ```
   GRANT hr_clerk TO "Finance";
   ```

Step 4: Test the Code Based Access Control HR.print_employees Procedure

At this stage, you are ready to test the code based access control `HR.print_employees` procedure.

To test the code based access control `HR.print_employees` procedure, user `"Finance"` must query the `HR.EMPLOYEES` table and try to run the `HR.print_employees` procedure.

1. Connect to the database instance as user `"Finance"`.

   ```
   CONNECT "Finance"
   Enter password: password
   ```

2. Try to directly query the `HR.EMPLOYEES` table.

   ```
   SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY FROM HR.EMPLOYEES;
   ```

   The query fails because user `Finance` does not have the `SELECT` privilege for `HR.EMPLOYEES`.

   ```
   ERROR at line 1:
   ORA-00942: table or view does not exist
   ```

3. Execute the `HR.print_employees` procedure.

   ```
   EXEC HR.print_employees;
   ```

   The query fails because user `"Finance"` does not have the appropriate privileges.

   ```
   ERROR at line 1:
   ORA-00942: table or view does not exist
   ORA-06512: at "HR.PRINT_EMPLOYEES", line 13ORA-06512: at line 1
   ```

Step 5: Create the view_emp_role Role and Grant Privileges for It

Next, user `HR` must create the `view_emp_role` role and then grant privileges to it.

User `HR` grants the `SELECT` privilege `HR.EMPLOYEES` and `HR.DEPARTMENTS` to the `view_emp_role` role, and then grants `SELECT` on `HR.EMPLOYEES` and `HR.DEPARTMENTS` to the `view_emp_role` role.

1. Connect as user `HR`.

   ```
   CONNECT HR
   Enter password: password
   ```

2. Create the `view_emp_role` role.

   ```
   CREATE ROLE view_emp_role;
   ```

3. Grant the `SELECT` privilege on `HR.EMPLOYEES` and `HR.DEPARTMENTS` to the `view_emp_role` role.

   ```
   GRANT SELECT ON HR.EMPLOYEES TO view_emp_role;
   GRANT SELECT ON HR.DEPARTMENTS TO view_emp_role;
   ```

4. Grant the `view_emp_role` role to the `HR.print_employees` invoker's rights procedure.

   ```
   GRANT view_emp_role TO PROCEDURE HR.print_employees;
   ```

Step 6: Test the HR.print_employees Procedure Again

With the appropriate privileges in place, user `"Finance"` can try the `HR.print_employees` procedure again.

1. Connect as user `"Finance"`.

   ```
   CONNECT "Finance"
   Enter password: password
   ```

2. Set the server output to display.

   ```
   SET SERVEROUTPUT ON;
   ```

3. Try to directly query the `HR.EMPLOYEES` table.

   ```
   SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY FROM HR.EMPLOYEES;
   ```

   The query fails.

   ```
   ERROR at line 1:
   ORA-00942: table or view does not exist
   ```

4. Execute the `HR.print_employees` procedure to show the employee information.

   ```
   EXEC HR.print_employees;
   ```

   The call succeeds.

   ```
   ID         First Name     Last Name      Email          Phone Number
   108        Nancy          Greenberg      NGREENBE       515.124.4569
   109        Daniel         Faviet         DFAVIET        515.124.4169
   110        John           Chen           JCHEN          515.124.4269
   111        Ismael         Sciarra        ISCIARRA       515.124.4369
   112        Jose Manuel    Urman          JMURMAN        515.124.4469
   113        Luis           Popp           LPOPP          515.124.4567

   PL/SQL procedure successfully completed.
   ```

Step 7: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. Connect as a user with administrative privileges.

   For example:

   ```
   CONNECT sec_admin
   Enter password: password
   ```

2. Drop the user `"Finance"`.

   ```
   DROP USER "Finance";
   ```

3. Drop the `hr_clerk` role.

   ```
   DROP ROLE hr_clerk;
   ```

4. Connect as user `HR`.

   ```
   CONNECT HR
   Enter password: password
   ```

5. Drop the view_emp_role role and the `HR.print_employees` procedure.

   ```
   DROP ROLE view_emp_role;
   DROP PROCEDURE print_employees;
   ```

6. Connect as the administrative user.

   ```
   CONNECT sec_admin
   Enter password: password
   ```

7. Revoke the `CREATE ROLE` privilege from `HR`.

   ```
   REVOKE CREATE ROLE FROM HR;
   ```

Controlling Definer's Rights Privileges for Database Links

You can control privilege grants for definer’s rights procedures if your applications use database links and definer’s rights procedures.

About Controlling Definer's Rights Privileges for Database Links

When a definer’s rights procedure connects to a database link, operations on the database link should use the procedure owner’s credentials.

The `INHERIT REMOTE PRIVILEGES` and `INHERIT ANY REMOTE PRIVILEGES` privileges apply when a connected user database link is used with a definer’s rights procedure. These privileges allow the use of the credentials of the logged-in user for connected user database link operations with definer rights procedures.

You can perform a grant of the `INHERIT REMOTE PRIVILEGES` and `INHERIT ANY REMOTE PRIVILEGES` privileges so the users who invoke the definer’s rights procedure can use a connected user database link within a definer’s rights block. A definer’s rights procedure executes with the privileges of the procedure owner. However, a connected user database link operation must have the credentials of the logged in user. Hence, the `INHERIT REMOTE PRIVILEGES` and `INHERIT ANY REMOTE PRIVILEGES` privileges are required to be granted to enable the database link operations within the definer’s rights block.

Be aware that during an upgrade, the `INHERIT REMOTE PRIVILEGES` and `INHERIT ANY REMOTE PRIVILEGES` privileges are not granted by default to any existing users.

The `INHERIT REMOTE PRIVILEGES` and `INHERIT ANY REMOTE PRIVILEGES` privileges apply only to situations in which users are trying to connect to user database links in a definer’s rights procedure. In addition, these privileges apply to both privately created and publicly created database links. By default, database links are created as private links. In addition, by default, `INHERIT REMOTE PRIVILEGES` is not granted to `PUBLIC`.

The ways that you can perform grants of these privileges are as follows:

- `GRANT INHERIT REMOTE PRIVILEGES ON USER` *`dbuser_1`* `TO` *`dbuser_2`*: In this scenario, `dbuser_1` can explicitly grant the `INHERIT REMOTE PRIVILEGE` privilege to `dbuser_2` and use a definer’s rights procedure that user `dbuser_2` owns.
- `GRANT INHERIT REMOTE PRIVILEGES ON USER` *`dbuser_1`* `TO PUBLIC`. In this scenario, `dbuser_1` grants the `INHERIT REMOTE PRIVILEGE` privilege to public. This grant enables `dbuser_1` to use the definer’s rights procedures that any other user owns.
- `GRANT INHERIT ANY REMOTE PRIVILEGES TO` *`dbuser_2`*: In this scenario, any user can use the definer’s rights procedures that `dbuser_2` owns.

If the user does not have the `INHERIT REMOTE PRIVILEGE` privilege and tries to execute the definer’s rights privilege, then the `ORA-25433: User does not have INHERIT REMOTE PRIVILEGES` error appears.

Grants of the INHERIT REMOTE PRIVILEGES Privilege to Other Users

The `INHERIT REMOTE PRIVILEGES` privilege enables the current user to have explicit privileges over the connected user in the database.

The syntax for granting the `INHERIT REMOTE PRIVILEGES` privilege is as follows:

```
GRANT INHERIT REMOTE PRIVILEGES ON USER connected_user TO current_user;
```

In this specification:

- *`connected_user`* is the user who runs the definer's rights procedure.
- *`current_user`* is the user who owns the definer's right procedure. This value must be a database user account. As an alternative to granting the `INHERIT REMOTE PRIVILEGES` privilege to the procedure's owner, you can grant the privilege to a role that is in turn granted to the procedure.

Users or roles who own the definer's rights procedures must have the `INHERIT REMOTE PRIVILEGES` privilege granted to them by users who will run their definer's rights procedures.

Any user can grant or revoke the `INHERIT REMOTE PRIVILEGES` privilege on themselves to the user whose definer's rights procedures they want to run.

Example: Granting INHERIT REMOTE PRIVILEGES on a Connected User

You can grant the `INHERIT REMOTE PRIVILEGES` privilege on a connected user to the current user.

In this example, the connected user, `jward`, must have remote privileges on the current user, `ebrown`. This enables `jward` to execute the definer's right procedure that `ebrown` created.

Example 6-4 shows how an administrator (or user `jward`) can grant the `INHERIT REMOTE PRIVILEGES` on user `jward` to user `ebrown`. This privilege grant enables any definer's rights procedure that `ebrown` writes, or will write in the future, to access `ebrown`'s privileges when the procedure is run.

Example 6-4   Granting INHERIT REMOTE PRIVILEGES on a Connected User to the Current User

```
GRANT INHERIT REMOTE PRIVILEGES on user jward to ebrown;
```

Grants of the INHERIT ANY REMOTE PRIVILEGES Privilege to Other Users

The `INHERIT ANY REMOTE PRIVILEGES` privilege enables the grantee user to open a *`connected_user`* database link as any user.

As with all `ANY` privileges, `INHERIT ANY REMOTE PRIVILEGES` is a powerful privilege that must only be granted to trusted users. By default, user `SYS` has the `INHERIT ANY REMOTE PRIVILEGES` system privilege `WITH GRANT OPTION`. To find users who have been granted the `INHERIT ANY REMOTE PRIVILEGES` privilege, query the `DBA_SYS_PRIVS` data dictionary view.

For better security in a multitenant environment, Oracle recommends that you protect the `INHERIT ANY REMOTE PRIVILEGES` privilege with a PDB lockdown profile. A PDB lockdown profile prevents local pluggable database (PDB) users from opening a connected user database link as a common user, irrespective of the kind of `INHERIT REMOTE PRIVILEGE` the PDB user has. If the PDB is protected by a PDB lockdown profile, then grants such as `GRANT INHERIT REMOTE PRIVILEGES` and `GRANT INHERIT ANY REMOTE` privileges succeed but the effects of these grants do not apply as long as the PDB lockdown continues.

The syntax for granting the `INHERIT ANY REMOTE PRIVILEGES` privilege is as follows:

```
GRANT INHERIT ANY REMOTE PRIVILEGES TO current_user;
```

In this specification, *`current_user`* is the user who owns the define’s right procedure.

Revokes of the INHERIT [ANY] REMOTE PRIVILEGES Privilege

The methods for revoking the `INHERIT REMOTE PRIVILEGES` and `INHERIT ANY REMOTE PRIVILEGES` privileges differ.

The `INHERIT REMOTE PRIVILEGES` privilege can be revoked by a user from another user. The `INHERIT ANY REMOTE PRIVILEGES` privilege must be revoked by a user with administrative privileges.

The revocation syntax is as follows

```
REVOKE INHERIT REMOTE PRIVILEGES ON USER connected_user FROM current_user;
```

In this specification:

- *`connected_user`* is the user who runs the definer's rights procedure.
- *`current_user`* is the user who owns the definer’s rights procedure.

If you want to revoke the `INHERIT REMOTE PRIVILEGES` or `INHERIT ANY REMOTE PRIVILEGES` privilege from a user, use the standard revocation syntax, as follows:

```
REVOKE INHERIT REMOTE PRIVILEGES FROM connected_user;
REVOKE INHERIT ANY REMOTE PRIVILEGES FROM current_user;
```

See Also:

Oracle Database SQL Language Reference for more information about the `REVOKE` SQL statement

Example: Revoking the INHERIT REMOTE PRIVILEGES Privilege

The `REVOKE` SQL statement can revoke the `INHERIT REMOTE PRIVILEGES` privilege.

After you revoke the `INHERIT REMOTE PRIVILEGES` privilege, if user `jward` executes a definer's rights procedure that `jward` owns, then any operation on a connected user database link inside the definer’s rights procedure fails because `jward` has explicitly denied `ebrown` the privilege to open a connected user database link using `jward`’credentials.

Example 6-5 shows how to revoke the `INHERIT REMOTE PRIVILEGES` procedure on the connecting user, `jward`, from the procedure owner, `ebrown`.

Example 6-5 Revoking the INHERIT REMOTE PRIVILEGES Privilege

```
REVOKE INHERIT REMOTE PRIVILEGES ON USER jward FROM ebrown;
```

Example: Revoking the INHERIT REMOTE PRIVILEGES Privilege from PUBLIC

The `REVOKE` SQL statement can revoke the `INHERIT REMOTE PRIVILEGES` from `PUBLIC`, as well as from individual procedure owners.

Example 6-6 shows how to revoke this privilege from `PUBLIC`.

Example 6-6 Revoking the INHERIT REMOTE PRIVILEGES Privilege from PUBLIC

```
REVOKE INHERIT REMOTE PRIVILEGES FROM PUBLIC;
```

Tutorial: Using a Database Link in a Definer's Rights Procedure

This tutorial demonstrates how the `INHERIT REMOTE PRIVILEGES` privilege works in a definer's rights procedure that uses a database link.

About This Tutorial

In this tutorial, you test the privilege grant and revoke of the `INHERIT REMOTE PRIVILEGES` privilege.

To accomplish this, you must create two users, one who creates a definer’s rights procedure that refers to a database link, and a second user to execute this definer’s rights procedure. Both users create identical look-up tables in their schemas. The definer’s rights procedure must enable the second user to query the lookup table that belongs to the definer’s rights users.

Step 1: Create User Accounts

You must create a user who creates a definer's rights procedure that has a database link, and a second user who executes this procedure.

1. Connect as a user who has privileges to create users and perform privilege grants.

   For example:

   ```
   sqlplus sec_admin
   Enter password: password
   ```

2. Create the user accounts as follows:

   ```
   GRANT CONNECT, RESOURCE, UNLIMITED TABLESPACE TO dbuser1 IDENTIFIED BY password;
   GRANT CONNECT, RESOURCE, UNLIMITED TABLESPACE TO dbuser2 IDENTIFIED BY password;
   ```

   Follow the guidelines in Minimum Requirements for Passwords to replace *`password`* with a password that is secure.

Step 2: As User dbuser2, Create a Table to Store User IDs

The user IDs in this table are the IDs that the database link uses.

1. Connect as user `dbuser2` to instance `inst1`.

```
connect dbuser2@inst1
Enter password: password
```

2. Create the following table:

```
CREATE TABLE dbusertab(ID NUMBER(2));
```

3. Populate this table with the ID value `10`.

```
INSERT INTO dbusertab VALUES(10);
```

Step 3: As User dbuser1, Create a Database Link and Definer's Rights Procedure

User `dbuser1` is ready to create a database link and then a definer's rights procedure that references the database link.

1. Connect as user `dbuser1` to instance `inst1`.

```
connect dbuser1@inst1
Enter password: password
```

2. Create a database link, which will be used in the definer's rights procedure.

```
CREATE DATABASE LINK dblink USING 'inst1';
```

3. Create a `dbusertab` table and then populate it with the ID `20`.

```
CREATE TABLE DBUSERTAB(ID NUMBER(2));
INSERT INTO dbusertab VALUES(20);
```

4. Create a definer's rights procedure that contains a reference to the database lnk

```
CREATE OR REPLACE PROCEDURE test_remote_db_link
AS
v_id varchar(50);
BEGIN
    SELECT ID INTO v_id FROM dbusertab@dblink;
    DBMS_OUTPUT.PUT_LINE('v_id : ' || v_id);
END ;
/
```

5. Test the definer’s rights procedure.

```
SET SERVEROUTPUT ON
EXEC test_remote_db_link;
```

The output should be as follows, indicating that user `dbuser1` has executed the procedure on his own version of the table `dbusertab`:

```
v_id : 20
```

6. Grant the user `dbuser2` the `EXECUTE` privilege on the `test_remote_db_link` procedure.

```
GRANT EXECUTE ON test_remote_db_link TO dbuser2;
```

Step 4: Test the Definer's Rights Procedure

User `dbuser2` must grant `INHERIT REMOTE PRIVILEGES` to `dbuser1` before the definer’s rights procedure can be tested.

1. Connect as user `dbuser2` to instance `inst1`.

```
connect dbuser2@inst1
Enter password: password
```

2. Grant the `INHERIT REMOTE PRIVILEGE` privilege on user `dbuser2` to `dbuser1`.

   ```
   GRANT INHERIT REMOTE PRIVILEGES ON user dbuser2 TO dbuser1;
   ```

3. Relog back in, because the grant does not take effect until you start a new session.

   ```
   connect dbuser2@inst1
   Enter password: password
   ```

4. Execute the `test_remote_db_link` definer’s rights procedure:

   ```
   SET SERVEROUTPUT ON
   EXEC dbuser1.test_remote_db_link;
   ```

   The output shows the following, which indicates that user `dbuser1` is able to use the database link to connect to the schema of `dbuser2` and access the values in the `dbusertab` table in `dbuser2`’s schema.

   ```
   v_id : 10
   ```

5. Revoke the `INHERIT REMOTE PRIVILEGE` privilege on `dbuser2` from `dbuser1`.

   ```
   REVOKE INHERIT REMOTE PRIVILEGES ON USER dbuser2 FROM dbuser1;
   ```

6. Try executing the `test_remote_db_link` definer’s rights procedure again.

   ```
   EXEC dbuser1.test_remote_db_link;
   ```

   The `ORA-25433: User DBUSER1 does not have INHERIT REMOTE PRIVILEGES on connected user DBUSER2` error should appear.

Step 5: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. Connect as a user who has privileges to drop user accounts and database links

   For example:

   ```
   connect sec_admin
   Enter password: password
   ```

2. Drop the user accounts.

   ```
   DROP USER dbuser1 CASCADE;
   DROP USER dbuser2 CASCADE;
   ```

3. Drop the `dblink` database link.

   ```
   DROP PUBLIC DATABASE LINK dblink;
   ```

7

Managing Fine-Grained Access in PL/SQL Packages and Types

Oracle Database provides PL/SQL packages and types for fine-grained access to control access to external network services and wallets.

About Managing Fine-Grained Access in PL/SQL Packages and Types

You can configure user access to external network services and wallets through a set of PL/SQL packages and one type.

These packages are the `UTL_TCP`, `UTL_SMTP`, `UTL_MAIL`, `UTL_HTTP`, and `UTL_INADDR` ,and the `DBMS_LDAP` PL/SQL packages, and the `HttpUriType` type.

The following scenarios are possible:

- **Configuring fine-grained access control for users and roles that need to access external network services from the database.** This way, specific groups of users can connect to one or more host computers, based on privileges that you grant them. Typically, you use this feature to control access to applications that run on specific host addresses.
- **Configuring fine-grained access control to Oracle wallets to make HTTP requests that require password or client-certificate authentication.** This feature enables you to grant privileges to users who are using passwords and client certificates stored in Oracle wallets to access external protected HTTP resources through the `UTL_HTTP` package. For example, you can configure applications to use the credentials stored in the wallets instead of hard-coding the credentials in the applications.

About Fine-Grained Access Control to External Network Services

Oracle Application Security access control lists (ACL) can implement fine-grained access control to external network services.

This guide explains how to configure the access control for database users and roles by using the `DBMS_NETWORK_ACL_ADMIN` PL/SQL package.

This feature enhances security for network connections because it restricts the external network hosts that a database user can connect to using the PL/SQL network utility packages `UTL_TCP`, `UTL_SMTP`, `UTL_MAIL`, `UTL_HTTP`, and `UTL_INADDR`; the `DBMS_LDAP` and `DBMS_DEBUG_JDWP` PL/SQL packages; and the `HttpUriType` type. Otherwise, an intruder who gained access to the database could maliciously attack the network, because, by default, the PL/SQL utility packages are created with the `EXECUTE` privilege granted to `PUBLIC` users. These PL/SQL network utility packages, and the

DBMS_NETWORK_ACL_ADMIN and DBMS_NETWORK_ACL_UTILITY packages, support both IP Version 4 (IPv4) and IP Version 6 (IPv6) addresses. This guide explains how to manage access control to both versions. For detailed information about how the IPv4 and IPv6 notation works with Oracle Database, see *Oracle Database Net Services Administrator's Guide*.

See Also:

Tutorial: Adding an Email Alert to a Fine-Grained Audit Policy for an example of configuring access control to external network services for email alerts

About Access Control to Oracle Wallets

When accessing remote Web server-protected Web pages, users can authenticate themselves with passwords and client certificates stored in an Oracle wallet.

The Oracle wallet provides secure storage of user passwords and client certificates.

To configure access control to a wallet, you must have the following components:

- **An Oracle wallet.** You can create the wallet using the Oracle Database mkstore utility or Oracle Wallet Manager. The HTTP request will use the external password store or the client certificate in the wallet to authenticate the user
- **An access control list to grant privileges to the user to use the wallet.** To configure the access control list, you use the DBMS_NETWORK_ACL_ADMIN PL/SQL package.

The use of Oracle wallets is beneficial because it provides secure storage of passwords and client certificates necessary to access protected Web pages.

Upgraded Applications That Depend on Packages That Use External Network Services

Upgraded applications may have ORA-24247 network access errors.

If you have upgraded from a release before Oracle Database 11*g* Release 1 (11.1), and your applications depend on PL/SQL network utility packages (UTL_TCP, UTL_SMTP, UTL_MAIL, UTL_HTTP, UTL_INADDR, and DBMS_LDAP) or the HttpUriType type, then the ORA-24247 error may occur when you try to run the application.

The error message is as follows:

```
ORA-24247: network access denied by access control list (ACL)
```

Use the procedures in this chapter to reconfigure the network access for the application.

See Also:

Oracle Database Upgrade Guide for compatibility issues for applications that depend on the PL/SQL network utility packages

Configuring Access Control for External Network Services

The `DBMS_NETWORK_ACL` packages configures access control for external network services.

Syntax for Configuring Access Control for External Network Services

You can use the `DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE` procedure to grant the access control privileges to a user.

This procedure appends an access control entry (ACE) with the specified privilege to the ACL for the given host, and creates the ACL if it does not exist yet. The resultant configuration resides in the `SYS` schema, not the schema of the user who created it.

The syntax is as follows:

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE (
  host         => 'host_name',
  lower_port   => null|port_number,
  upper_port   => null|port_number,
  ace          => ace_definition);
END;
```

In this specification:

- `host`: Enter the name of the host. It can be the host name or an IP address of the host. You can use a wildcard to specify a domain or an IP subnet. (See Precedence Order for a Host Computer in Multiple Access Control List Assignments for the precedence order when you use wildcards in domain names.) The host or domain name is case insensitive. Examples are as follows:

  ```
  host     => 'www.example.com',
  ```

  ```
  host     => '*example.com',
  ```

- `lower_port`: (Optional) For TCP connections, enter the lower boundary of the port range. Use this setting for the `connect` privilege only. Omit it for the `resolve` privilege. The default is `null`, which means that there is no port restriction (that is, the ACL applies to all ports). The range of port numbers is between 1 and 65535.

  For example:

  ```
  lower_port => 80,
  ```

- `upper_port`: (Optional) For TCP connections, enter the upper boundary of the port range. Use this setting for `connect` privileges only. Omit it for the `resolve` privilege. The default is `null`, which means that there is no port restriction (that is, the ACL applies to all ports). The range of port numbers is between 1 and 65535

  For example:

```
upper_port => 3999);
```

If you enter a value for the `lower_port` and leave the `upper_port` at `null` (or just omit it), then Oracle Database assumes the `upper_port` setting is the same as the `lower_port`. For example, if you set `lower_port` to `80` and omit `upper_port`, the `upper_port` setting is assumed to be `80`.

The `resolve` privilege in the access control list has no effect when a port range is specified in the access control list assignment.

- `ace`: Define the ACE by using the `XS$ACE_TYPE` constant, in the following format:

```
ace    => xs$ace_type(privilege_list => xs$name_list('privilege'),
                      principal_name => 'user_or_role',
                      principal_type => xs$ace_type_user));
```

In this specification:

- `privilege_list`: Enter one or more of the following privileges, which are case insensitive. Enclose each privilege with single quotation marks and separate each with a comma (for example, `'http', 'http_proxy'`).

  For tighter access control, grant only the `http`, `http_proxy`, or `smtp` privilege instead of the `connect` privilege if the user uses the `UTL_HTTP`, `HttpUriType`, `UTL_SMTP`, or `UTL_MAIL` only.

  - `http`: Makes an HTTP request to a host through the `UTL_HTTP` package and the `HttpUriType` type

  - `http_proxy`: Makes an HTTP request through a proxy through the `UTL_HTTP` package and the `HttpUriType` type. You must include `http_proxy` in conjunction to the `http` privilege if the user makes the HTTP request through a proxy.

  - `smtp`: Sends SMTP to a host through the `UTL_SMTP` and `UTL_MAIL` packages

  - `resolve`: Resolves a network host name or IP address through the `UTL_INADDR` package

  - `connect`: Grants the user permission to connect to a network service at a host through the `UTL_TCP`, `UTL_SMTP`, `UTL_MAIL`, `UTL_HTTP`, and `DBMS_LDAP` packages, or the `HttpUriType` type

  - `jdwp`: Used for Java Debug Wire Protocol debugging operations for Java or PL/SQL stored procedures. See Configuring Network Access for Java Debug Wire Protocol Operations for more information.

- `principal_name`: Enter a database user name or role. This value is case insensitive, unless you enter it in double quotation marks (for example, `'"ACCT_MGR'"`).

- `principal_type`: Enter `XS_ACL.PTYPE_DB` for a database user or role. You must specify `PTYPE_DB` because the `principal_type` value defaults to `PTYPE_XS`, which is used to specify an Oracle Database Real Application Security application user.

See Also:

Oracle Database Real Application Security Administrator's and Developer's Guide for information about additional `XS$ACE_TYPE` parameters that you can include for the `ace` parameter setting: `granted`, `inverted`, `start_date`, and `end_date`

Example: Configuring Access Control for External Network Services

The `DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE` procedure can configure access control for external network services.

Example 7-1 shows how to grant the `http` and `smtp` privileges to the `acct_mgr` database role for an ACL created for the host `www.example.com`.

Example 7-1 Granting Privileges to a Database Role External Network Services

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
  host       => 'www.example.com',
  ace        =>  xs$ace_type(privilege_list => xs$name_list('http', 'smtp'),
                             principal_name => 'acct_mgr',
                             principal_type => xs_acl.ptype_db));
END;
/
```

Revoking Access Control Privileges for External Network Services

You can remove access control privileges for external network services.

- To revoke access control privileges for external network services, run the `DBMS_NETWORK_ACL_ADMIN.REMOVE_HOST_ACE` procedure.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_NETWORK_ACL_ADMIN.REMOVE_HOST_ACE` procedure

Example: Revoking External Network Services Privileges

The `DBMS_NETWORK_ACL_ADMIN.REMOVE_HOST_ACE` procedure can be used to revoke external network privileges.

Example 7-2 shows how to revoke external network privileges.

Example 7-2 Revoking External Network Services Privileges

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.REMOVE_HOST_ACE (
  host       => 'www.example.com',
  lower_port => 80,
  upper_port => upper_port => 3999,
  ace        => xs$ace_type(privilege_list => xs$name_list('http', 'smtp'),
                            principal_name => 'acct_mgr',
                            principal_type => xs_acl.ptype_db),
  remove_empty_acl => TRUE);
END;
/
```

In this specification, the `TRUE` setting for `remove_empty_acl` removes the ACL when it becomes empty when the ACE is removed.

Configuring Access Control to an Oracle Wallet

Fine-grained access control for Oracle wallets provide user access to network services that require passwords or certificates.

About Configuring Access Control to an Oracle Wallet

You can configure access control to grant access to passwords and client certificates.

These passwords and client certificates are stored in an Oracle wallet. The access control that you configure enables users to authenticate themselves to an external network service when using the PL/SQL network utility packages.

This enables the user to gain access to the network service that requires password or certificate identification.

Step 1: Create an Oracle Wallet

An Oracle wallet can use both standard and PKCS11 wallet types, as well as being an auto-login wallet.

1. To create the wallet, use either the `mkstore` command-line utility or the Oracle Wallet Manager user interface.

   To store passwords in the wallet, you must use the `mkstore` utility.
2. Ensure that you have exported the wallet to a file.
3. Make a note of the directory in which you created the wallet. You will need this directory path when you complete the procedures in this section.

Step 2: Configure Access Control Privileges for the Oracle Wallet

After you have created the wallet, you are ready to configure access control privileges for the wallet.

- Use the `DBMS_NETWORK_ACL_ADMIN.APPEND_WALLET_ACE` procedure to configure the wallet access control privileges.

The syntax for the `DBMS_NETWORK_ACL_ADMIN.APPEND_WALLET_ACE` procedure is as follows:

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.APPEND_WALLET_ACE (
  wallet_path => 'directory_path_to_wallet',
  ace         => xs$ace_type(privilege_list => xs$name_list('privilege'),
                             principal_name => 'user_or_role',
                             principal_type => xs$ace_type_user));
END;
```

In this specification:

- `wallet_path`: Enter the path to the directory that contains the wallet that you created in Step 1: Create an Oracle Wallet. When you specify the wallet path, you must use an absolute path and include `file:` before this directory path. Do not use environment variables, such as `$ORACLE_HOME`, nor insert a space after `file:` and before the path name. For example:

  ```
  wallet_path   => 'file:/oracle/wallets/hr_wallet',
  ```

- `ace`: Define the ACL by using the `XS$ACE_TYPE` constant. For example:

  ```
    ace         =>  xs$ace_type(privilege_list => xs$name_list(privilege),
                               principal_name => 'hr_clerk',
                               principal_type => xs_acl.ptype_db);
  ```

  In this specification, *`privilege`* must be one of the following when you enter wallet privileges using `xs$ace_type` (note the use of underscores in these privilege names):

  – `use_client_certificates`

  – `use_passwords`

  For detailed information about these parameters, see the `ace` parameter description in Syntax for Configuring Access Control for External Network Services. Be aware that for wallets, you must specify either the `use_client_certificates` or `use_passwords` privileges.

See Also:

Oracle Database Real Application Security Administrator's and Developer's Guide for information about additional `XS$ACE_TYPE` parameters that you can include for the `ace` parameter setting: `granted`, `inverted`, `start_date`, and `end_date`

Step 3: Make the HTTP Request with the Passwords and Client Certificates

The `UTL_HTTP` package can create an HTTP request object to hold wallet information, which can authenticate using a client certificate or a password.

Making the HTTPS Request with the Passwords and Client Certificates

The `UTL_HTTP` package makes Hypertext Transfer Protocol (HTTP) callouts from SQL and PL/SQL.

- Use the `UTL_HTTP` PL/SQL package to create a request context object that is used privately with the HTTP request and its response.

For example:

```
DECLARE
 req_context UTL_HTTP.REQUEST_CONTEXT_KEY;
 req         UTL_HTTP.REQ;
BEGIN
```

```
 req_context := UTL_HTTP.CREATE_REQUEST_CONTEXT (
              wallet_path          => 'file:path_to_directory_containing_wallet',
              wallet_password      => 'wallet_password'|NULL);
 req := UTL_HTTP.BEGIN_REQUEST(
              url                  => 'URL_to_application',
              request_context      => 'request_context'|NULL);
 ...
END;
```

In this specification:

- `req_context`: Use the `UTL_HTTP.CREATE_REQUEST_CONTEXT_KEY` data type to create the request context object. This object stores a randomly-generated numeric key that Oracle Database uses to identify the request context. The `UTL_HTTP.CREATE_REQUEST_CONTEXT` function creates the request context itself.
- `req`: Use the `UTL_HTTP.REQ` data type to create the object that will be used to begin the HTTP request. You will refer to this object later on, when you set the user name and password from the wallet to access a password-protected Web page.
- `wallet_path`: Enter the path to the directory that contains the wallet. Ensure that this path is the same path you specified when you created access control list in Step 2: Configure Access Control Privileges for the Oracle Wallet in the previous section. You must include `file:` before the directory path. Do not use environment variables, such as `$ORACLE_HOME`.

  For example:

  ```
  wallet_path          => 'file:/oracle/wallets/hr_wallet',
  ```
- `wallet_password`: Enter the password used to open the wallet. The default is `NULL`, which is used for auto-login wallets. For example:

  ```
  wallet_password      => 'wallet_password');
  ```
- `url`: Enter the URL to the application that uses the wallet.

  For example:

  ```
  url                  => 'www.hr_access.example.com',
  ```
- `request_context`: Enter the name of the request context object that you created earlier in this section. This object prevents the wallet from being shared with other applications in the same database session.

  For example:

  ```
  request_context      => req_context);
  ```

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `UTL_HTTP` package

Using a Request Context to Hold the Wallet When Sharing the Session with Other Applications

You should use a request context to hold the wallet when other applications share the database session.

If your application has exclusive use of the database session, you can hold the wallet in the database session by using the `UTL_HTTP.SET_WALLET` procedure.

- Use the `UTL_HTTP.SET_WALLET` procedure to configure the request to hold the wallet.

For example:

```
DECLARE
 req         UTL_HTTP.REQ;
BEGIN
 UTL_HTTP.SET_WALLET(
             path          => 'file:path_to_directory_containing_wallet',
             password      => 'wallet_password'|NULL);
 req := UTL_HTTP.BEGIN_REQUEST(
             url           => 'URL_to_application');
 ...
END;
```

If the protected URL being requested requires the user name and password to authenticate, then you can use the `SET_AUTHENTICATION_FROM_WALLET` procedure to set the user name and password from the wallet to authenticate.

Use of Only a Client Certificate to Authenticate

Only a client certificate can authenticate users, as long as the user has been granted the appropriate privilege in the ACL wallet.

If the protected URL being requested requires only the client certificate to authenticate, then the `BEGIN_REQUEST` function sends the necessary client certificate from the wallet. assuming the user has been granted the `use_client_certificates` privilege in the ACL assigned to the wallet.

The authentication should succeed at the remote Web server and the user can proceed to retrieve the HTTP response by using the `GET_RESPONSE` function.

Use of a Password to Authenticate

If the protected URL being requested requires username and password authentication, then set the username and password from the wallet to authenticate.

For example:

```
DECLARE
 req_context UTL_HTTP.REQUEST_CONTEXT_KEY;
 req         UTL_HTTP.REQ;
BEGIN
...
 UTL_HTTP.SET_AUTHENTICATION_FROM_WALLET(
  r              => HTTP_REQUEST,
  alias          => 'alias_to_retrieve_credentials_stored_in_wallet',
  scheme         => 'AWS|Basic',
  for_proxy       => TRUE|FALSE);
END;
```

In this specification:

- `r`: Enter the HTTP request defined in the `UTL_HTTP.BEGIN_REQUEST` procedure that you created above, in the previous section. For example:

  ```
  r               => req,
  ```

- `alias`: Enter the alias used to identify and retrieve the user name and password credential stored in the Oracle wallet. For example, assuming the alias used to identify this user name and password credential is `hr_access`.

  ```
  alias           => 'hr_access',
  ```

- `scheme`: Enter one of the following:
  - `AWS`: Specifies the Amazon Simple Storage Service (S3) scheme. Use this scheme only if you are configuring access to the Amazon.com Web site. (Contact Amazon for more information about this setting.)
  - `Basic`: Specifies HTTP basic authentication. The default is `Basic`.

  For example:

  ```
  scheme          => 'Basic',
  ```

- `for_proxy`: Specify whether the HTTP authentication information is for access to the HTTP proxy server instead of the Web server. The default is `FALSE`.

  For example:

  ```
  for_proxy       => TRUE);
  ```

The use of the user name and password in the wallet requires the `use_passwords` privilege to be granted to the user in the ACL assigned to the wallet.

Revoking Access Control Privileges for Oracle Wallets

You can revoke access control privileges for an Oracle wallet.

- To revoke privileges from access control entries (ACE) in the access control list (ACL) of a wallet, run the `DBMS_NETWORK_ACL_ADMIN.REMOVE_WALLET_ACE` procedure.

For example:

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.REMOVE_WALLET_ACE (
  wallet_path   => 'file:/oracle/wallets/hr_wallet',
  ace           =>  xs$ace_type(privilege_list => xs$name_list(privilege),
                            principal_name => 'hr_clerk',
                            principal_type => xs_acl.ptype_db),
  remove_empty_acl  => TRUE);
END;
/
```

In this example, the `TRUE` setting for `remove_empty_acl` removes the ACL when it becomes empty when the wallet ACE is removed.

Examples of Configuring Access Control for External Network Services

You can configure access control for a variety of situations, such as for a single role and network connection.

Example: Configuring Access Control for a Single Role and Network Connection

The `DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE` procedure can configure access control for a single role and network connection.

Example 7-3 shows how you would configure access control for a single role (`acct_mgr`) and grant this role the `http` privilege for access to the `www.us.example.com` host. The privilege expires January 1, 2013.

Example 7-3   Configuring Access Control for a Single Role and Network Connection

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
  host       => 'www.us.example.com',
  lower_port => 80,
  ace        =>  xs$ace_type(privilege_list => xs$name_list('http'),
                          principal_name => 'acct_mgr',
                          principal_type => xs_acl.ptype_db,
                          end_date => TIMESTAMP '2013-01-01 00:00:00.00 -08:00');
END;
/
```

Example: Configuring Access Control for a User and Role

The `DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE` can configure access control to deny or grant privileges for a user and a role.

Afterwards, you can query the `DBA_HOST_ACES` data dictionary view to find information about the privilege grants.

Example 7-4 grants to a database role (`acct_mgr`) but denies a particular user (`psmith`) even if he has the role. The order is important because ACEs are evaluated in the given order. In this case, the deny ACE (`granted => false`) must be appended first or else the user cannot be denied.

Example 7-4   Configuring Access Control Using a Grant and a Deny for User and Role

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
  host       => 'www.us.example.com',
  lower_port => 80,
  upper_port => 80,
  ace        =>  xs$ace_type(privilege_list => xs$name_list('http'),
                            principal_name => 'psmith',
                            principal_type => xs_acl.ptype_db,
                            granted        => false));
 DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
  host       => 'www.us.example.com',
  lower_port => 80,
  upper_port => 80,
  ace        =>  xs$ace_type(privilege_list => xs$name_list('http'),
                             principal_name => 'acct_mgr',
                             principal_type => xs_acl.ptype_db,
                             granted        => true));
END;
```

Example: Using the DBA_HOST_ACES View to Show Granted Privileges

The `DBA_HOST_ACE` data dictionary view shows privileges that have been granted to users.

Example 7-5 shows how the `DBA_HOST_ACES` data dictionary view displays the privilege granted in the previous access control list.

Example 7-5   Using the DBA_HOST_ACES View to Show Granted Privileges

```
SELECT PRINCIPAL, PRIVILEGE, GRANT_TYPE FROM DBA_HOST_ACE WHERE PRIVILEGE = 'HTTP';

PRINCIPAL    PRIVILEGE  GRANT_TYPE
------------ ---------- --------------------
PSMITH       HTTP       FALSE
ACCT_MGR     HTTP       TRUE
```

Example: Configuring ACL Access Using Passwords in a Non-Shared Wallet

The `DBMS_NETWORK_ACL_ADMIN` and `UTL_HTTP` PL/SQL packages can configure ACL access using passwords in a non-shared wallet.

Example 7-6 configures wallet access for two Human Resources department roles, `hr_clerk` and `hr_manager`. These roles use the `use_passwords` privilege to access passwords stored in the wallet. In this example, the wallet will not be shared with other applications within the same database session.

Example 7-6   Configuring ACL Access Using Passwords in a Non-Shared Wallet

```
/* 1. At a command prompt, create the wallet. The following example uses the
      user name hr_access as the alias to identify the user name and password
      stored in the wallet. You must use this alias name when you call the
      SET_AUTHENTICATION_FROM_WALLET procedure later on. */
$ mkstore -wrl $ORACLE_HOME/wallets/hr_wallet -create
Enter password: password
Enter password again: password
$ mkstore -wrl $ORACLE_HOME/wallets/hr_wallet -createCredential hr_access hr_usr
Your secret/Password is missing in the command line
Enter your secret/Password: password
Re-enter your secret/Password: password
Enter wallet password: password

/* 2. In SQL*Plus, create an access control list to grant privileges for the
      wallet. The following example grants the use_passwords privilege to the
      hr_clerk role.*/
BEGIN
 DBMS_NETWORK_ACL_ADMIN.APPEND_WALLET_ACE (
  wallet_path => 'file:/oracle/wallets/hr_wallet',
  ace         => xs$ace_type(privilege_list => xs$name_list('use_passwords'),
                             principal_name => 'hr_clerk',
                             principal_type => xs_acl.ptype_db));
END;
/

/* 3. Create a request context and request object, and then set the authentication
      for the wallet. */
DECLARE
  req_context  UTL_HTTP.REQUEST_CONTEXT_KEY;
  req          UTL_HTTP.REQ;

BEGIN
 req_context := UTL_HTTP.CREATE_REQUEST_CONTEXT(
     wallet_path          => 'file:/oracle/wallets/hr_wallet',
     wallet_password      => NULL,
     enable_cookies       => TRUE,
     max_cookies          => 300,
     max_cookies_per_site => 20);
  req := UTL_HTTP.BEGIN_REQUEST(
     url                  => 'www.hr_access.example.com',
     request_context      => req_context);
  UTL_HTTP.SET_AUTHENTICATION_FROM_WALLET(
     r                    => req,
     alias                => 'hr_access'),
     scheme               => 'Basic',
     for_proxy            => FALSE);
END;
/
```

Example: Configuring ACL Access for a Wallet in a Shared Database Session

The DBMS_NETWORK_ACL_ADMIN and UTL_HTTP PL/SQL packages can configure ACL access for a wallet in a shared database session.

Example 7-7 configures the wallet to be used for a shared database session; that is, all applications within the current database session will have access to this wallet.

Example 7-7 Configuring ACL Access for a Wallet in a Shared Database Session

```
/* Follow these steps:
   1. Use Oracle Wallet Manager to create the wallet and add the client
      certificate.

   2. In SQL*Plus, configure access control to grant privileges for the wallet.
      The following example grants the use_client_certificates privilege
      to the hr_clerk and hr_mgr roles. */
BEGIN
 DBMS_NETWORK_ACL_ADMIN.APPEND_WALLET_ACE (
  wallet_path => 'file:/oracle/wallets/hr_wallet',
  ace         => xs$ace_type(privilege_list => xs$name_list('use-
client_certificates'),
                             principal_name => 'hr_clerk',
                             principal_type => xs_acl.ptype_db));

 DBMS_NETWORK_ACL_ADMIN.APPEND_WALLET_ACE (
  wallet_path => 'file:/oracle/wallets/hr_wallet',
  ace         => xs$ace_type(privilege_list =>
xs$name_list('use_client_certificates'),
                             principal_name => 'hr_mgr',
                             principal_type => xs_acl.ptype_db));
END;
/
COMMIT;

/* 3. Create a request object to handle the HTTP authentication for the wallet.*/
DECLARE
  req  UTL_HTTP.req;
BEGIN
  UTL_HTTP.SET_WALLET(
   path            => 'file:/oracle/wallets/hr_wallet',
   password        => NULL);
 req := UTL_HTTP.BEGIN_REQUEST(
   url             => 'www.hr_access.example.com',
   method          => 'POST',
   http_version    => NULL,
   request_context => NULL);
END;
/
```

Specifying a Group of Network Host Computers

You can use wildcards to specify a group of network host computers.

- To assign an access control list to a group of network host computers, use the asterisk (*) wildcard character.

For example, enter `*.example.com` for host computers that belong to a domain or `192.0.2.*` for IPv4 addresses that belong to an IP subnet. The asterisk wildcard must be at the beginning, before a period (.) in a domain, or at the end, after a period (.), in an IP subnet. For example, `*.example.com` is valid, but `*example.com` and `*.example.*` are not. Be aware that the use of wildcard characters affects the order of precedence for multiple access control lists that are assigned to the same host computer. You cannot use wildcard characters for IPv6 addresses.

The Classless Inter-Domain Routing (CIDR) notation defines how IPv4 and IPv6 addresses are categorized for routing IP packets on the internet. The `DBMS_NETWORK_ACL_ADMIN` package supports CIDR notation for both IPv4 and IPv6 addresses. This package considers an IPv4-mapped IPv6 address or subnet equivalent to the IPv4-native address or subnet it represents. For example, `::ffff:192.0.2.1` is equivalent to `192.0.2.1`, and `::ffff:192.0.2.1/120` is equivalent to `192.0.2.*`.

Precedence Order for a Host Computer in Multiple Access Control List Assignments

The access control list assigned to a domain has a lower precedence than those assigned to the subdomains.

For multiple access control lists that are assigned to the host computer and its domains, the access control list that is assigned to the host computer takes precedence over those assigned to the domains.

The access control list assigned to a domain has a lower precedence than those assigned to the subdomains.For example, Oracle Database first selects the access control list assigned to the host `server.us.example.com`, ahead of other access control lists assigned to its domains. If additional access control lists were assigned to the sub domains, their order of precedence is as follows:

1. `server.us.example.com`
2. `*.us.example.com`
3. `*.example.com`
4. `*.com`
5. `*`

Similarly, for multiple access control lists that are assigned to the IP address (both IPv4 and IPv6) and the subnets it belongs to, the access control list that is assigned to the IP address takes precedence over those assigned to the subnets. The access control list assigned to a subnet has a lower precedence than those assigned to the smaller subnets it contains.

For example, Oracle Database first selects the access control list assigned to the IP address `192.0.2.3`, ahead of other access control lists assigned to the subnets it belongs to. If additional access control lists were assigned to the subnets, their order of precedence is as follows:

1. `192.0.2.3` (or `::ffff:192.0.2.3`)
2. `192.0.2.3/31` (or `::ffff:192.0.2.3/127`)
3. `192.0.2.3/30` (or `::ffff:192.0.2.3/126`)
4. `192.0.2.3/29` (or `::ffff:192.0.2.3/125`)
5. ...
6. `192.0.2.3/24` (or `::ffff:192.0.2.3/120` or `192.0.2.*`)
7. `...`
8. `192.0.2.3/16` (or `::ffff:192.0.2.3/112` or `192.0.*`)
9. ...
10. `192.0.2.3/8` (or `::ffff:192.0.2.3/104` or `192.*`)
11. ...
12. `::ffff:192.0.2.3/95`
13. `::ffff:192.0.2.3/94`
14. ...
15. `*`

Precedence Order for a Host in Access Control List Assignments with Port Ranges

The precedence order for a host in an access control list is determined by the use of port ranges.

When an access control list is assigned to a host computer, a domain, or an IP subnet with a port range, it takes precedence over the access control list assigned to the same host, domain, or IP subnet without a port range.

For example, suppose you have TCP connections to any port between port 80 and 99 at `server.us.example.com`. Oracle Database first selects the access control list assigned to port 80 through 99 at `server.us.example.com`, ahead of the other access control list assigned to `server.us.example.com` that is without a port range.

Checking Privilege Assignments That Affect User Access to Network Hosts

Both administrators and users can check network connection and domain privileges.

About Privilege Assignments that Affect User Access to Network Hosts

Oracle provides DBA-specific data dictionary views to find information about privilege assignments.

Database administrators can use the `DBA_HOST_ACES` data dictionary view to query network privileges that have been granted to or denied from database users and roles in the access control lists, and whether those privileges take effect during certain times only

Using the information provided by the view, you may need to combine the data to determine if a user is granted the privilege at the current time, the roles the user has, the order of the access control entries, and so on.

Users without database administrator privileges do not have the privilege to access the access control lists or to invoke those `DBMS_NETWORK_ACL_ADMIN` functions. However, they can query the `USER_HOST_ACES` data dictionary view to check their privileges instead.

Database administrators and users can use the following `DBMS_NETWORK_ACL_UTILITY` functions to determine if two hosts, domains, or subnets are equivalent, or if a host, domain, or subnet is equal to or contained in another host, domain, or subnet:

- `EQUALS_HOST`: Returns a value to indicate if two hosts, domains, or subnets are equivalent
- `CONTAINS_HOST`: Returns a value to indicate if a host, domain, or subnet is equal to or contained in another host, domain, or subnet, and the relative order of precedence of the containing domain or subnet for its ACL assignments

If you do not use IPv6 addresses, database administrators and users can use the following `DBMS_NETWORK_ACL_UTILITY` functions to generate the list of domains or IPv4 subnet a host belongs to and to sort the access control lists by their order of precedence according to their host assignments:

- `DOMAINS`: Returns a list of the domains or IP subnets whose access control lists may affect permissions to a specified network host, subdomain, or IP subnet
- `DOMAIN_LEVEL`: Returns the domain level of a given host

How to Check User Network Connection and Domain Privileges

A database administrator can query the `DBA_HOST_ACES` data dictionary view to find the privileges that have been granted for specific users or roles.

The `DBA_HOST_ACES` view shows the access control lists that determine the access to the network connection or domain, and then determines if each access control list grants (`GRANTED`), denies (`DENIED`), or does not apply (`NULL`) to the access privilege of the user. Only the database administrator can query this view.

Example: Administrator Checking User Network Access Control Permissions

The `DBA_HOST_ACES` data dictionary view can check the network access control permissions for users.

Example 7-8 shows how a database administrator can check the privileges for user `preston` to connect to `www.us.example.com`.

In this example, user `preston` was granted privileges for all the network host connections found for `www.us.example.com`. However, suppose `preston` had been granted access to a host connection on port 80, but then denied access to the host connections on ports 3000–3999. In this case, you must configure access control for the host connection on port 80, and a separate access control configuration for the host connection on ports 3000–3999.

Example 7-8 Administrator Checking User Network Access Control Permissions

```
SELECT HOST, LOWER_PORT, UPPER_PORT,
       ACE_ORDER, PRINCIPAL, PRINCIPAL_TYPE,
       GRANT_TYPE, INVERTED_PRINCIPAL, PRIVILEGE,
       START_DATE, END_DATE
  FROM (SELECT ACES.*,
DBMS_NETWORK_ACL_UTILITY.CONTAINS_HOST('www.us.example.com', HOST) PRECEDENCE
          FROM DBA_HOST_ACES ACES)
 WHERE PRECEDENCE IS NOT NULL
 ORDER BY PRECEDENCE DESC,
          LOWER_PORT NULLS LAST,
          UPPER_PORT NULLS LAST,
          ACE_ORDER;
HOST               LOWER_PORT UPPER_PORT ACE_ORDER PRINCIPAL PRINCIPAL_TYPE   GRANT_TYPE
INVERTED_PRINCIPAL PRIVILEGE START_DATE END_DATE
------------------ ---------- ---------- --------- --------- ---------------- ----------
------------------ --------- ---------- --------
www.us.example.com        80        80          1 PRESTON DATABASE USER      GRANT
NO                 HTTP
www.us.example.com        80        80          2 SEBASTIAN DATABASE USER    GRANT
NO                 HTTP
*.us.example.com                                1 ACCT_MGR DATABASE USER     GRANT
NO                 CONNECT
*                                               1 HR_DBA DATABASE USER       GRANT
NO                 CONNECT
```

```
*                                                  1 HR_DBA DATABASE USER       GRANT
NO                 RESOLVE
```

How Users Can Check Their Network Connection and Domain Privileges

Users can query the `USER_HOST_ACES` data dictionary view to check their network and domain permissions.

The `USER_HOST_ACES` view is `PUBLIC`, so all users can query it.

This view hides the access control lists from the user. It evaluates the permission status for the user (`GRANTED` or `DENIED`) and filters out the `NULL` case because the user does not need to know when the access control lists do not apply to him or her. In other words, Oracle Database only shows the user on the network hosts that explicitly grant or deny access to him or her. Therefore, the output does not display the `*.example.com` and `*` that appear in the output from the database administrator-specific `DBA_HOST_ACES` view.

Example: User Checking Network Access Control Permissions

The USER_HOST_ACES data dictionary view shows network access control permissions for a host computer.

Example 7-9 shows how user `preston` can check her privileges to connect to `www.us.example.com`.

Example 7-9 User Checking Network Access Control Permissions

```
SELECT HOST, LOWER_PORT, UPPER_PORT, PRIVILEGE, STATUS
  FROM (SELECT ACES.*,
DBMS_NETWORK_ACL_UTILITY.CONTAINS_HOST('www.us.example.com', HOST) PRECEDENCE
          FROM USER_HOST_ACES ACES)
 WHERE PRECEDENCE IS NOT NULL
 ORDER BY PRECEDENCE DESC,
          LOWER_PORT NULLS LAST,
          UPPER_PORT NULLS LAST;

HOST               LOWER_PORT UPPER_PORT PRIVILEGE STATUS
------------------ ---------- ---------- --------- -------
www.us.example.com         80         80  HTTP      GRANTED
```

Configuring Network Access for Java Debug Wire Protocol Operations

Before you can debug Java PL/SQL procedures, you must be granted the `jdwp` ACL privilege.

If you want to debug Java PL/SQL procedures in the database through a Java Debug Wire Protocol (JDWP)-based debugger, such as SQL Developer, JDeveloper, or Oracle Developer Tools For Visual Studio (ODT), then you must be granted the `jdwp` ACL privilege to connect your database session to the debugger at a particular host.

The `jdwp` privilege is needed in conjunction with the `DEBUG CONNECT SESSION` system privilege.

If you have not been granted the `jdwp` ACL privilege, then when you try to debug your Java and PL/SQL stored procedures from a remote host, the following errors may appear:

```
ORA-24247: network access denied by access control list (ACL)
ORA-06512: at "SYS.DBMS_DEBUG_JDWP", line line_number
```

- To configure network access for JDWP operations, use the `DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE` procedure.

The following example illustrates how to configure network access for JDWP operations.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host         => 'host',
    lower_port   => null|port_number,
    upper_port   => null|port_number,
    ace => xs$ace_type(privilege_list => xs$name_list('jdwp'),
                       principal_name => 'username',
                       principal_type => xs_acl.ptype_db));
END;
/
```

In this specification:

- *`host`* can be a host name, domain name, IP address, or subnet.
- *`port_number`* enables you to specify a range of ports. If you want to use any port, then omit the `lower_port` and `upper_port` values.
- *`username`* is case-insensitive unless it is quoted (for example, `principal_name => '"PSMITH"'`).

See Also:

- *Oracle Database Java Developer's Guide* for more information about debugging server applications with JDWP
- *Oracle SQL Developer User's Guide* for information about remote debugging in SQL Developer

Data Dictionary Views for Access Control Lists Configured for User Access

Oracle Database provides data data dictionary views that you can use to find information about existing access control lists.

Table 7-1 lists these views.

Table 7-1 Data Dictionary Views That Display Information about Access Control Lists

View	Description
`DBA_HOST_ACES`	Shows the network privileges defined for the network hosts. The `SELECT` privilege on this view is granted to the `SELECT_CATALOG_ROLE` role only.
`DBA_WALLET_ACES`	Lists the wallet path, ACE order, start and end times, grant type, privilege, and information about principals
`DBA_WALLET_ACLS`	Shows the access control list assignments to the wallets. The `SELECT` privilege on this view is granted to the `SELECT_CATALOG_ROLE` role only.
`DBA_HOST_ACLS`	Shows the access control list assignments to the network hosts. The `SELECT` privilege on this view is granted to the `SELECT_CATALOG_ROLE` role only.
`USER_HOST_ACES`	Shows the status of the network privileges for the current user to access network hosts. The `SELECT` privilege on the view is granted to `PUBLIC`.
`USER_WALLET_ACES`	Shows the status of the wallet privileges for the current user to access contents in the wallets. The `SELECT` privilege on the view is granted to `PUBLIC`.

See Also:

Oracle Database Reference for more information about these views

8 Managing Security for a Multitenant Environment in Enterprise Manager

You can manage common and local users and roles for a multitenant environment by using Oracle Enterprise Manager.

This section contains the following topics:

About Managing Security for a Multitenant Environment in Enterprise Manager

Oracle Enterprise Manager Cloud Control supports the management of multitenant environment security.

In a multitenant environment, you can use Oracle Enterprise Manager Cloud Control to create, manage, and monitor common users and roles for both the root and the associated pluggable databases (PDBs).

Enterprise Manager enables you to switch easily between the root and a designated PDB.

Logging into a Multitenant Environment in Enterprise Manager

In a multitenant environment, you can log in to a CDB or a PDB, and switch from a PDB to a different PDB or to the root.

This section contains the following topics:

Logging into a CDB or a PDB

Different variations of the Enterprise Manager Database login page appear automatically based on the feature that you requested while logging in.

To log into a multitenant environment as a CDB administrator (an Enterprise Manager user who has the `CONNECT` privilege on the CDB target) to use a CDB-scoped feature:

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

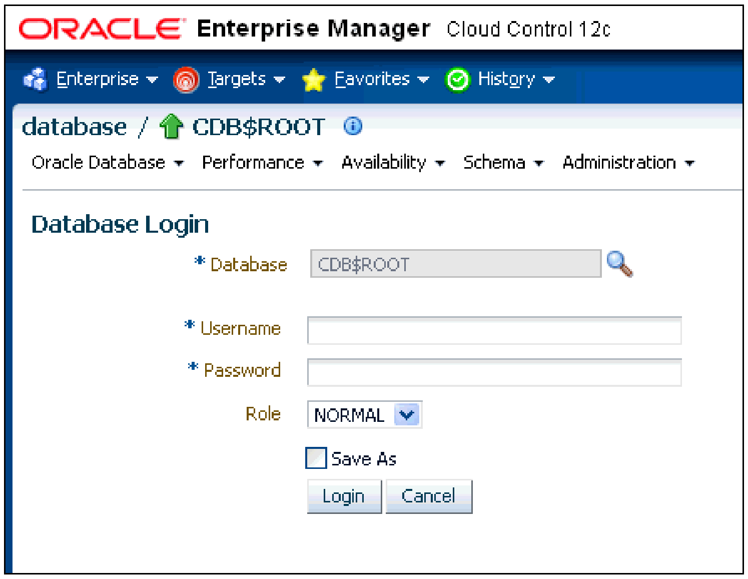
   https://*host*:*port*/em

2. Navigate to the Databases page.
3. Select the database that you want to access.

   The database home page appears.

4. Select the menu item for the action that you want to perform, such as selecting **Administration**, then **Security**, and then **Users** to authenticate a user.

   The Database Login page appears. The following example shows the Database Login page for the CDB (because the database name is shown as `CDB$ROOT`). Because of this name, this page is colloquially referred to as the database login page for the `root` of the multitenant environment. The **Database** field refers to the current database; had you selected a PDB, then the name of the PDB would appear in this field.



Database Login

* Database	CDB$ROOT
* Username	
* Password	
Role	NORMAL
	☐ Save As

5. Log in using the appropriate credentials.

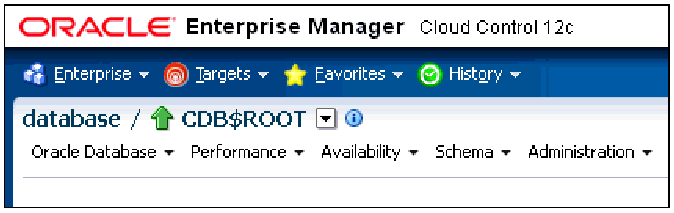
   Remember that only common users can log into the root, and that the names of common users begin with `C##` or `c##`. Both common and local users can log into a PDB, depending on their privileges.

Switching to a Different PDB or to the Root

From Oracle Enterprise Manager, you can switch from one PDB to a different PDB, or to the root.

1. At the top left side of the page, find the **database** link.

   In the **database** link, the current container name appears. The following example shows that the current database is the CDB itself (`CDB$ROOT`), colloquially known as the root.



2. Select the menu icon to the right of the container, and from this menu, select the database that you want to access.

If the menu item does not appear, then navigate to a page where it does appear, such as the Database home page.

3. When you decide which activity you want to perform (such as creating users), log in with the appropriate privileges.

   If you attempt to perform an activity without first having authenticated with the appropriate privileges, then you will be prompted to log in with the appropriate privilege.

Managing Common and Local Users in Enterprise Manager

In a multitenant environment, Oracle Enterprise Manager enables you to create, edit, and drop common and local users.

This section contains the following topics:

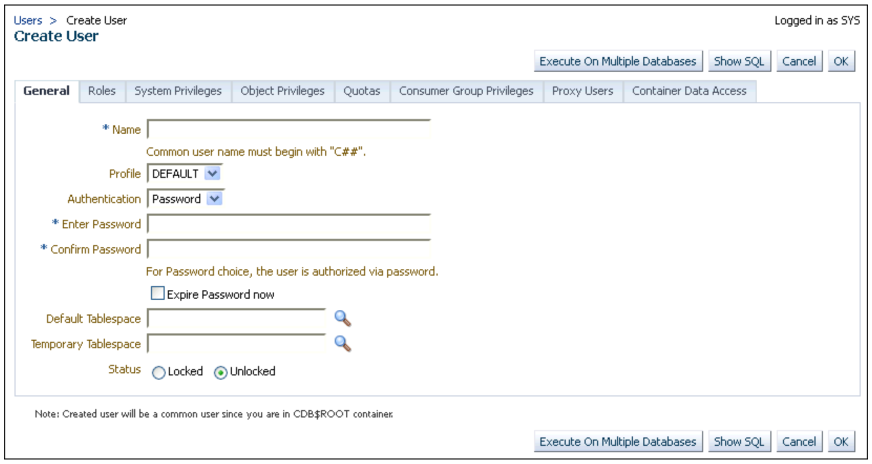
Creating a Common User Account in Enterprise Manager

A common user is a user that exists in the root and can access PDBs in the CDB.

1. From the Enterprise Manager database home page, log in to the root as a common user who has the common `CREATE USER` and `SET CONTAINER` privileges.
2. From the **Administration** menu, select **Security**, then **Users**.

   If prompted, enter your login information. Afterward, the Users page appears.
3. Click **Create**.

   The Create User page appears.



* Name
Common user name must begin with "C##".
Profile: DEFAULT
Authentication: Password
* Enter Password
* Confirm Password
For Password choice, the user is authorized via password.
☐ Expire Password now
Default Tablespace
Temporary Tablespace
Status: Locked [✓ Unlocked]

Note: Created user will be a common user since you are in CDB$ROOT container.

4. Select the options to create a common user and grant this user privileges.

   Ensure that you preface the user name with `C##` or `c##`.
5. Click **OK** or **Apply**.

   The common user is created in the root and will appear in the Users page for any associated PDBs.

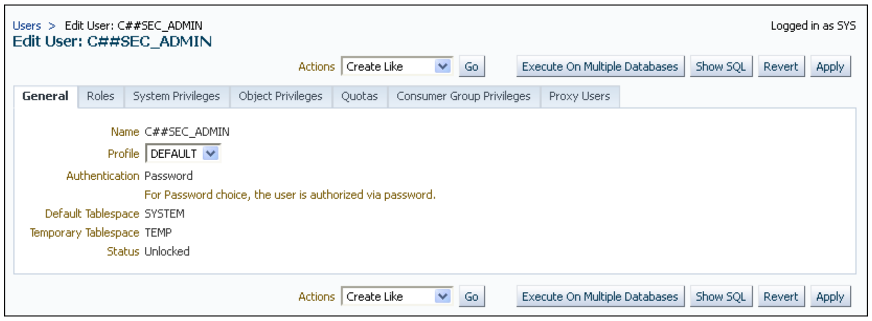
Editing a Common User Account in Enterprise Manager

You can edit a common user account from the root.

1. From the Enterprise Manager database home page, log in to the root as a common user who has the common `CREATE USER` and `SET CONTAINER` privileges.
   - If you are logging into the root, then ensure that you are a common user who has the common `CREATE USER` and `SET CONTAINER` privileges.
   - If you are logging into a PDB, ensure that you have the `CREATE USER` privilege for that PDB.
2. From the **Administration** menu, select **Security**, then **Users**.

   If prompted, enter your login information. Afterward, the Users page appears. In the root, only common users are listed. In the PDB, both common and local users are listed.
3. Select the common user to be edited and then click **Edit**.

   The Edit User page appears. For a common user in the root, you can modify all settings for the common user. For a common user in a PDB, you cannot change the user password, default tablespace, and temporary tablespace. The settings that you make apply only to the current PDB. The following screen shows how a common user Edit User page appears in a PDB.



Edit User: C##SEC_ADMIN

Name	C##SEC_ADMIN
Profile	DEFAULT
Authentication	Password
	For Password choice, the user is authorized via password.
Default Tablespace	SYSTEM
Temporary Tablespace	TEMP
Status	Unlocked

4. Modify the common user as necessary.
5. Click **Apply**.

Dropping a Common User Account in Enterprise Manager

You can drop a common user from the CDB root.

1. From the Enterprise Manager database home page, log in to the root as a common user who has the common `CREATE USER` and `SET CONTAINER` privileges.

   You cannot drop common users from PDBs.
2. From the **Administration** menu, select **Security**, then **Users**.

   If prompted, enter your login information. Afterward, the Users page appears, listing only common users.
3. Select the common user that you want to drop and then click **Delete**.
4. Confirm that you want to delete the common user.

Creating a Local User Account in Enterprise Manager

A local user is a user that exists only in a specific PDB and does not have access to any other PDBs in the multitenant environment.

1. From the Enterprise Manager database home page, log in to the root as a local or common user who has the local `CREATE USER` privilege.
2. From the **Administration** menu, select **Security**, then **Users**.

   If prompted, enter your login information. Afterward, the Users page appears, showing only local users for the current PDB.
3. Click **Create**.

   The Create User page appears.
4. Select the options that create a local user and grant this user privileges.

   Ensure that you do not preface the user name with `C##` or `c##`.
5. Click **OK**.

   The local user is created in the current PDB.

Editing a Local User Account in Enterprise Manager

You can edit a local user from the PDB in which the local user resides.

1. From the Enterprise Manager database home page, log in to the PDB as a local or common user who has the local `CREATE USER` privilege.
2. From the **Administration** menu, select **Security**, then **Users**.

   If prompted, enter your login information. Afterward, the Users page appears, showing only local users for the current PDB and common users.
3. Select the local user to be edited and then click **Edit**.

   The Edit User page appears.
4. Modify the local user as necessary.
5. Click **Apply**.

Dropping a Local User Account in Enterprise Manager

You can drop a local user from the PDB in which the local user resides.

1. From the Enterprise Manager database home page, log in to the PDB as a local or common user who has the local `CREATE USER` privilege.
2. From the **Administration** menu, select **Security**, then **Users**.

   If prompted, enter your login information. Afterward, the Users page appears, showing only local users for the current PDB and common users. (You cannot drop common users from a PDB.)
3. Select the local user you want to drop and then click **Delete**.

   Enterprise Manager prompts you to confirm deletion of the user.
4. Confirm that you want to delete the local user.

Managing Common and Local Roles and Privileges in Enterprise Manager

In a multitenant environment, you can use Oracle Enterprise Manager to create, edit, drop, and revoke common and local roles.

This section contains the following topics:

Creating a Common Role in Enterprise Manager

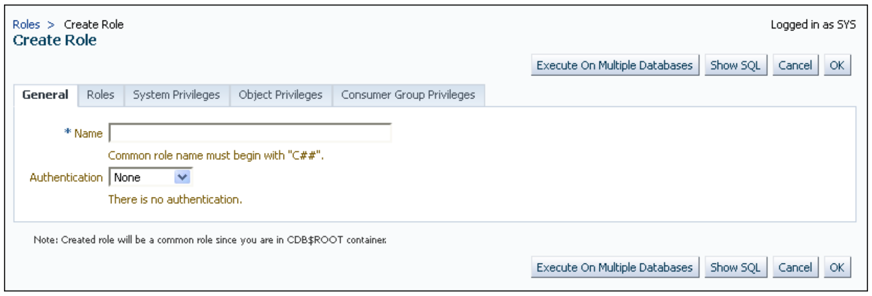
Common roles can be used to assign common privileges to common users.

These roles are valid across all containers of the multitenant environment.

1. From the Enterprise Manager database home page, log in to the root as a common user who has the common `CREATE ROLE` and `SET CONTAINER` privileges.
2. From the **Administration** menu, select **Security**, then **Roles**.

   If prompted, enter your login information. Afterward, the Create Role page appears.
3. Click **Create**.

   The Create Role page appears.



* Name

Common role name must begin with "C##".

Authentication: None

There is no authentication.

Note: Created role will be a common role since you are in CDB$ROOT container.

4. Select the options that create a common role and grant this role privileges.

   Ensure that you preface the role name with `C##` or `c##`.
5. Click **OK**.

   The common role is created in the root.

Editing a Common Role in Enterprise Manager

You can edit a common role from the root.

1. From the Enterprise Manager database home page, log in to the root or the PDB. If you are logging into the root, then ensure that you are a common user who has the common `CREATE ROLE` and `SET CONTAINER` privileges. If you are logging into a PDB, ensure that you have the `CREATE ROLE` privilege for that PDB.
2. From the **Administration** menu, select **Security**, then **Roles**.

If prompted, enter your login information. Afterward, the Roles page appears. In the root, only common roles are shown. In the PDB, both common and local roles are shown.

3. Select the common role to be edited and then click **Edit**.

   The Edit Role page appears. For a common user in the root, you can modify all settings for the common user.

   For a common role in a PDB, you can only change the role's authentication and grant this user different roles, system privileges, object privileges, and consumer group privileges. These settings apply only to the current PDB.

4. Modify the common user as necessary.
5. Click **Apply**.

Dropping a Common Role in Enterprise Manager

You can drop a common role from the root.

1. From the Enterprise Manager database home page, log in to the root as a common user who has the common `CREATE ROLE` and `SET CONTAINER` privileges.

   You cannot drop common roles from PDBs.

2. From the **Administration** menu, select **Security**, then **Roles**.

   If prompted, enter your login information. Afterward, the Roles page appears, showing only common roles.

3. Select the common role that you want to drop and then click **Delete**.
4. Confirm that you want to delete the common role.

Revoking Common Privilege Grants in Enterprise Manager

You can revoke common privilege grants from the root.

1. From the Enterprise Manager database home page, log in to the root as a common user who has the common `CREATE USER`, `CREATE ROLE`, and `SET CONTAINER` privileges.
2. From the **Administration** menu, select **Security**, then **Users**.

   The Users page lists the common users.

3. Select the user whose privileges you want to revoke and then click **Edit**.

   The Edit User page appears.

4. Select **Roles** or the appropriate **Privileges** tab.

   Enterprise Manager displays a list of roles and privileges assigned to this user.

5. Select **Edit List** and then remove the roles or privileges as necessary.
6. Click the **OK** button.

Creating a Local Role in Enterprise Manager

A common role can be used to assign a local set of privileges to local users later.

These roles will be valid across PDB containers for whom they are defined.

1. From the Enterprise Manager database home page, log in to the PDB as a user who has the local `CREATE ROLE` privilege.
2. From the **Administration** menu, select **Security**, then **Roles**.

   The Roles page appears.
3. Click **Create**.

   If prompted, enter your login information. Afterward, the Create Role page appears.
4. Select the options that create a local role and grant this role privileges.

   Ensure that you do not preface the role name with `C##` or `c##`.
5. Click **OK**.

   The local role is created in the current PDB.

Editing a Local Role in Enterprise Manager

You can edit a local role in the PDB in which the local role resides.

1. From the Enterprise Manager database home page, log in to the PDB as a user who has the local `CREATE ROLE` privilege.
2. From the **Administration** menu, select **Security**, then **Roles**.

   If prompted, enter your login information. Afterward, the Roles page appears, showing only local roles for the current PDB and common roles.
3. Select the local role to be edited and then click **Edit**.

   The Edit User page appears.
4. Modify the local user as necessary.
5. Click **Apply**.

Dropping a Local Role in Enterprise Manager

You can drop local role from the PDB in which the local role resides.

1. From the Enterprise Manager database home page, log in to the PDB as a user who has the local `CREATE ROLE` privilege.
2. From the **Administration** menu, select **Security**, then **Role**.

   If prompted, enter your login information. Afterward, the Roles page appears, showing only local roles for the current PDB and common roles. (You cannot drop common roles from a PDB.)
3. Select the local role you want to drop and then click **Delete**.

   Enterprise Manager prompts you to confirm deletion of the role.
4. Confirm that you want to delete the local role.

Revoking Local Privilege Grants in Enterprise Manager

You can revoke local privileges in the PDB in which the privileges are used.

1. From the Enterprise Manager database home page, log in to the PDB as a common or local user who has the `CREATE USER` and `CREATE ROLE` privileges.
2. From the **Administration** menu, select **Security**, then **Users**.

   If prompted, enter your login information. Afterward, the Users page appears. In a PDB, both common and local users are listed.
3. Select the user whose privileges you want to revoke and then click **Edit**.

   The Edit User page appears.
4. Select **Roles** or the appropriate **Privileges** tab.

   Enterprise Manager displays a list of roles and privileges assigned to this user.
5. Select **Edit List** and then remove the privileges as necessary.
6. Click the **OK** button.

Part II
Application Development Security

Part II describes how to manage application development security.

9

Managing Security for Application Developers

A security policy for application developers should encompass areas such as password management and securing external procedures and application privileges.

About Application Security Policies

An application security policy is a list of application security requirements and rules that regulate user access to database objects.

Creating an application security policy is the first step to create a secure database application. You should draft security policies for each database application. For example, each database application should have one or more database roles that provide different levels of security when executing the application. You then can grant the database roles to other roles or directly to specific users.

Applications that can potentially allow unrestricted SQL statement processing (through tools such as SQL*Plus or SQL Developer) also need security policies that prevent malicious access to confidential or important schema objects. In particular, you must ensure that your applications handle passwords in a secure manner.

Considerations for Using Application-Based Security

An application security implementation should consider both application and database users and whether to enforce security in the application or in the database.

Are Application Users Also Database Users?

Where possible, build applications in which application users are database users, so that you can use the intrinsic security mechanisms of the database.

For many commercial packaged applications, application users are not database users. For these applications, multiple users authenticate themselves to the application, and the application then connects to the database as a single, highly-privileged user. This is called the *One Big Application User* model.

Applications built in this way generally cannot use many of the intrinsic security features of the database, because the identity of the user is not known to the database. However, you can use client identifiers to perform some types of tracking. For example, the `OCI_ATTR_CLIENT_IDENTIFIER` attribute of the Oracle Call Interface method `OCIAttrSet` can be used to enable auditing and monitoring of client connections. Client identifiers can be used to gather audit trail data on individual Web users, apply rules that restrict data access by Web users, and monitor and trace applications that each Web user users.

Table 9-1 describes how the One Big Application User model affects various Oracle Database security features:

Table 9-1 Features Affected by the One Big Application User Model

Oracle Database Feature	Limitations of One Big Application User Model
Auditing	A basic principle of security is accountability through auditing. If One Big Application User performs all actions in the database, then database auditing cannot hold individual users accountable for their actions. The application must implement its own auditing mechanisms to capture individual user actions.
Oracle strong authentication	Strong forms of authentication (such as client authentication over SSL, tokens, and so on) cannot be used if the client authenticating to the database is the application, rather than an individual user.
Roles	Roles are assigned to database users. Enterprise roles are assigned to enterprise users who, though not created in the database, are known to the database. If application users are not database users, then the usefulness of roles is diminished. Applications must then craft their own mechanisms to distinguish between the privileges which various application users need to access data within the application.
Enterprise user management	The Enterprise user management feature enables an Oracle database to use the Oracle Identity Management Infrastructure by securely storing and managing user information and authorizations in an LDAP-based directory such as Oracle Internet Directory. While enterprise users do not need to be created in the database, they do need to be known to the database. The One Big Application User model cannot take advantage of Oracle Identity Management.

Is Security Better Enforced in the Application or in the Database?

Oracle recommends that applications use the security enforcement mechanisms of the database as much as possible.

Applications, whose users are also database users, can either build security into the application, or rely on intrinsic database security mechanisms such as granular privileges, virtual private databases (fine-grained access control with application context), roles, stored procedures, and auditing (including fine-grained auditing).

When security is enforced in the database itself, rather than in the application, it cannot be bypassed. The main shortcoming of application-based security is that security is bypassed if the user bypasses the application to access data. For example, a user who has SQL*Plus access to the database can execute queries without going through the Human Resources application. The user, therefore, bypasses all of the security measures in the application.

Applications that use the One Big Application User model must build security enforcement into the application rather than use database security mechanisms. Because it is the application, and not the database, that recognizes users; the application itself must enforce security measures for each user.

This approach means that each application that accesses data must re-implement security. Security becomes expensive, because organizations must implement the same security policies in multiple applications, and each new application requires an expensive reimplementation.

Securing Passwords in Application Design

Oracle provides strategies for securely invoking password services, such as from scripts, and for applying these strategies to other sensitive data.

General Guidelines for Securing Passwords in Applications

Guidelines for securing passwords in applications cover areas such as platform-specific security threats.

Platform-Specific Security Threats

You should be aware of potential security threats, which may not be obvious.

These security threats are as follows:

- **On UNIX and Linux platforms, command parameters are available for viewing by all operating system users on the same host computer.** As a result, passwords entered on the command line could be exposed to other users. However, do not assume that non-UNIX and Linux platforms are safe from this threat.
- **On some UNIX platforms, such as HP Tru64 and IBM AIX, environment variables for all processes are available for viewing by all operating system users.** However, do not assume that non-UNIX and Linux platforms are safe from this threat.
- **On Microsoft Windows, the command recall feature (the Up arrow) remembers user input across command invocations.** For example, if you use the `CONNECT SYSTEM/`*`password`* notation in SQL*Plus, exit, and then press the Up arrow to repeat the `CONNECT` command, the command recall feature reveals the connect string and displays the password. In addition, do not assume that non-Microsoft Windows platforms are safe from this threat.

Guidelines for Designing Applications to Handle Password Input

Oracle provides guidelines for designing applications to handle password input.

- **Design applications to interactively prompt for passwords.** For command-line utilities, do not force users to expose passwords at a command prompt.

  Check the APIs for the programming language you use to design applications for the best way to handle passwords from users. For an example of Java code that handles this functionality, see Example: Java Code for Reading Passwords.
- **Protect your database against code injection attacks.** Code injection attacks most commonly occur in the client application tool that sends SQL to the database (for example, SQL*Plus, or any Oracle Call Interface (OCI) or JDBC application. This includes database drivers that are built using these tools. A SQL injection attack causes SQL statements to behave in a manner that is not intended by the PL/SQL application. The injection attack takes place before the statement is sent to the database. For example, an intruder can bypass password authentication by setting a `WHERE` clause to `TRUE`.

To address the problem of SQL injection attacks, use bind variable arguments or create validation checks. If you cannot use bind variables, then consider using the `DBMS_ASSERT` PL/SQL package to validate the properties of input values. *Oracle Database PL/SQL Packages and Types Reference* describes the `DBMS_ASSERT` package in detail. You also should review any grants to roles such as `PUBLIC`.

Note that client applications users may not always associate SQL injection with PL/SQL, because the injection could occur before the statement is sent to the database.

See *Oracle Database PL/SQL Language Reference* for more information about preventing SQL injection.

- **If possible, design your applications to defer authentication.** For example:
  - Use certificates for logins.
  - Authenticate users by using facilities provided by the operating system. For example, applications on Microsoft Windows can use domain authentication.
- **Mask or encrypt passwords.** If you must store passwords, then mask or encrypt them. For example, you can mask passwords in log files and encrypt passwords in recovery files.
- **Authenticate each connection.** For example, if schema A exists in database 1, then do not assume that schema A in database 2 is the same user. Similarly, the local operating system user `psmith` is not necessarily the same person as remote user `psmith`.
- **Do not store clear text passwords in files or repositories.** Storing passwords in files increases the risk of an intruder accessing them.
- **Use a single master password.** For example:
  - You can grant a single database user proxy authentication to act as other database users. In this case, only a single database password is needed. See Proxy User Accounts and the Authorization of Users to Connect Through Them for more information.
  - You can create a password wallet, which can be opened by the master password. The wallet then contains the other passwords. See *Oracle Database Enterprise User Security Administrator's Guide* for more information about Wallet Manager.

Guidelines for Configuring Password Formats and Behavior

Oracle Database provides guidelines for configuring password formats and behavior.

- **Limit the lifetime for passwords.** You can set a password lifetime, after which the password expires and must be changed before the user can log in to the account. See About Controlling Password Aging and Expiration for parameters you can use to control the lifetime of a password.
- **Limit the ability of users to reuse old passwords.** See Controlling the User Ability to Reuse Previous Passwords for more information.
- **Force users to create strong, secure passwords.** See Guidelines for Securing Passwords for advice on creating strong passwords. About Password Complexity Verification explains how you can customize password requirements.
- **Enable case sensitivity in passwords.** See Managing Password Case Sensitivity for more information.

Guidelines for Handling Passwords in SQL Scripts

Oracle provides guidelines for handling passwords in SQL scripts.

- **Do not invoke SQL*Plus with a password on the command line, either in programs or scripts.** If a password is required but omitted, SQL*Plus prompts the user for it and then automatically disables the echo feature so that the password is not displayed.

  The following examples are secure because passwords are not exposed on the command line. Oracle Database also automatically encrypts these passwords over the network.

  ```
  $ sqlplus system
  Enter password: password

  SQL> CONNECT SYSTEM
  Enter password: password
  ```

  The following example exposes the password to other operating system users:

  ```
  sqlplus system/password
  ```

  The next example poses two security risks. First, it exposes the password to other users who may be watching over your shoulder. Second, on some platforms, such as Microsoft Windows, it makes the password vulnerable to a command line recall attack.

  ```
  $ sqlplus /nolog
  SQL> CONNECT SYSTEM/password
  ```

- **For SQL scripts that require passwords or secret keys, for example, to create an account or to log in as an account, do not use positional parameters, such as substitution variables &1, &2, and so on.** Instead, design the script to prompt the user for the value. You should also disable the echo feature, which displays output from a script or if you are using spool mode. To disable the echo feature, use the following setting:

  ```
  SET ECHO OFF
  ```

  A good practice is to ensure that the script makes the purpose of the value clear. For example, it should be clear whether or not the value will establish a new value, such as an account or a certificate, or if the value will authenticate, such as logging in to an existing account.

  The following example is secure because it prevents users from invoking the script in a manner that poses security risks: It does not echo the password; it does not record the password in a spool file.

  ```
  SET VERIFY OFF
   ACCEPT user CHAR PROMPT ‘Enter user to connect to: ‘
   ACCEPT password CHAR PROMPT ‘Enter the password for that user: ' HIDE
   CONNECT &user/&password
  ```

  In this example:

  – `SET VERIFY OFF` prevents the password from being displayed. (`SET VERIFY` lists each line of the script before and after substitution.) Combining the `SET VERIFY OFF` command with the `HIDE` command is a useful technique for hiding passwords and other sensitive input data.

  – `ACCEPT password CHAR PROMPT` includes the `HIDE` option for the `ACCEPT password` prompt, which prevents the input password from being echoed.

  The next example, which uses positional parameters, poses security risks because a user may invoke the script by passing the password on the command line. If the user does not enter a password and instead is prompted, the danger lies in that whatever the user types is echoed to the screen and to a spool file if spooling is enabled.

  ```
  CONNECT &1/&2
  ```

- **Control the log in times for batch scripts.** For batch scripts that require passwords, configure the account so that the script can only log in during the time in which it is supposed to run. For example, suppose you have a batch script that runs for an hour each evening starting at 8 p.m. Set the account so that the script can only log in during this time. If an intruder manages to gain access, then he or she has less of a chance of exploiting any compromised accounts.
- **Be careful when using DML or DDL SQL statements that prompt for passwords.** In this case, sensitive information is passed in clear text over the network. You can remedy this problem by using Oracle strong authentication.

  The following example of altering a password is secure because the password is not exposed:

  ```
  password psmith
  Changing password for psmith
  New password: password
  Retype new password: password
  ```

  This example poses a security risk because the password is exposed both at the command line and on the network:

  ```
  ALTER USER psmith IDENTIFIED BY password
  ```

Use of an External Password Store to Secure Passwords

You can store password credentials for connecting to a database by using a client-side Oracle wallet.

An Oracle wallet is a secure software container that stores the authentication and signing credentials needed for a user to log in.

See Also:

- Managing the Secure External Password Store for Password Credentials for more information about the secure external password store
- *Oracle Database Enterprise User Security Administrator's Guide* for information about using Oracle Wallet Manager to configure Oracle wallets

Securing Passwords Using the ORAPWD Utility

SYSDBA or SYSOPER users can use password files to connect to an application over a network.

- To create the password file, use the ORAPWD utility.

See Also:

Oracle Database Administrator’s Guide for more information about creating and maintaining a password file

Example: Java Code for Reading Passwords

You can create Java packages that can be used to read passwords.

Example 9-1 demonstrates how to create a Java package that can be used to read passwords.

Example 9-1 Java Code for Reading Passwords

```
// Change the following line to a name for your version of this package
package passwords.sysman.emSDK.util.signing;

import java.io.IOException;
import java.io.PrintStream;
import java.io.PushbackInputStream;
import java.util.Arrays;

/**
 * The static readPassword method in this class issues a password prompt
 * on the console output and returns the char array password
 * entered by the user on the console input.
 */
public final class ReadPassword {
  //----------------------------------
  /**
   * Test driver for readPassword method.
   * @param args the command line args
   */
  public static void main(String[] args) {
    char[] pass = ReadPassword.readPassword("Enter password: ");
    System.out.println("The password just entered is \""
      + new String(pass) + "\"");
    System.out.println("The password length is " + pass.length);
  }
   * Issues a password prompt on the console output and returns
   * the char array password entered by the user on the console input.
   * The password is not displayed on the console (chars are not echoed).
   * As soon as the returned char array is not needed,
   * it should be erased for security reasons (Arrays.fill(charArr, ' '));
   * A password should never be stored as a java String.
   *
   * Note that Java 6 has a Console class with a readPassword method,
   * but there is no equivalent in Java 5 or Java 1.4.
 * The readPassword method here is based on Sun's suggestions at
 * http://java.sun.com/developer/technicalArticles/Security/pwordmask.
 *
 * @param prompt the password prompt to issue
 * @return new char array containing the password
 * @throws RuntimeException if some error occurs
 */
public static final char[] readPassword(String prompt)
throws RuntimeException {
  try {
    StreamMasker masker = new StreamMasker(System.out, prompt);
    Thread threadMasking = new Thread(masker);
    int firstByte = -1;
    PushbackInputStream inStream = null;
    try {
      threadMasking.start();
      inStream = new PushbackInputStream(System.in);
      firstByte = inStream.read();
    } finally {
      masker.stopMasking();
    }
    try {
      threadMasking.join();
    } catch (InterruptedException e) {
      throw new RuntimeException("Interrupt occurred when reading password");
    }
    if (firstByte == -1) {
      throw new RuntimeException("Console input ended unexpectedly");
    }
    if (System.out.checkError()) {
      throw new RuntimeException("Console password prompt output error");
    }
    inStream.unread(firstByte);
    return readLineSecure(inStream);
  }
  catch (IOException e) {
    throw new RuntimeException("I/O error occurred when reading password");
  }
}
//----------------------------------
/**
 * Reads one line from an input stream into a char array in a secure way
 * suitable for reading a password.
 * The char array will never contain a '\n' or '\r'.
 *
 * @param inStream the pushback input stream
 * @return line as a char array, not including end-of-line-chars;
 *  never null, but may be zero length array
 * @throws RuntimeException if some error occurs
 */
private static final char[] readLineSecure(PushbackInputStream inStream)
throws RuntimeException {
  if (inStream == null) {
    throw new RuntimeException("readLineSecure inStream is null");
  }
  try {
    char[] buffer = null;
    try {
      buffer = new char[128];
      int offset = 0;
      // EOL is '\n' (unix), '\r\n' (windows), '\r' (mac)
      loop:
      while (true) {
        int c = inStream.read();
        switch (c) {
        case -1:
        case '\n':
          break loop;
        case '\r':
          int c2 = inStream.read();
          if ((c2 != '\n') && (c2 != -1))
            inStream.unread(c2);
          break loop;
        default:
          buffer = checkBuffer(buffer, offset);
          buffer[offset++] = (char) c;
          break;
        }
      }
      char[] result = new char[offset];
      System.arraycopy(buffer, 0, result, 0, offset);
      return result;
    }
    finally {
      if (buffer != null)
        Arrays.fill(buffer, ' ');
    }
  }
  catch (IOException e) {
    throw new RuntimeException("I/O error occurred when reading password");
  }
}
//----------------------------------
/**
 * This is a helper method for readLineSecure.
 *
 * @param buffer the current char buffer
 * @param offset the current position in the buffer
 * @return the current buffer if it is not yet full;
 *  otherwise return a larger buffer initialized with a copy
 *  of the current buffer and then erase the current buffer
 * @throws RuntimeException if some error occurs
 */
private static final char[] checkBuffer(char[] buffer, int offset)
throws RuntimeException
{
  if (buffer == null)
    throw new RuntimeException("checkBuffer buffer is null");
  if (offset < 0)
    throw new RuntimeException("checkBuffer offset is negative");
  if (offset < buffer.length)
    return buffer;
  else {
    try {
      char[] bufferNew = new char[offset + 128];
      System.arraycopy(buffer, 0, bufferNew, 0, buffer.length);
      return bufferNew;
    } finally {
      Arrays.fill(buffer, ' ');
    }
  }
}
```

```
//---------------------------------
/**
 * This private class prints a one line prompt
 * and erases reply chars echoed to the console.
 */
private static final class StreamMasker
extends Thread {
  private static final String BLANKS = StreamMasker.repeatChars(' ', 10);
  private String m_promptOverwrite;
  private String m_setCursorToStart;
  private PrintStream m_out;
  private volatile boolean m_doMasking;
  //---------------------------------
  /**
   * Constructor.
   * @throws RuntimeException if some error occurs
   */
  public StreamMasker(PrintStream outPrint, String prompt)
  throws RuntimeException {
    if (outPrint == null)
      throw new RuntimeException("StreamMasker outPrint is null");
    if (prompt == null)
      throw new RuntimeException("StreamMasker prompt is null");
    if (prompt.indexOf('\r') != -1)
      throw new RuntimeException("StreamMasker prompt contains a CR");
    if (prompt.indexOf('\n') != -1)
      throw new RuntimeException("StreamMasker prompt contains a NL");
    m_out = outPrint;
    m_setCursorToStart = StreamMasker.repeatChars('\010',
      prompt.length() + BLANKS.length());
    m_promptOverwrite = m_setCursorToStart + prompt + BLANKS
      + m_setCursorToStart + prompt;
  }
  //---------------------------------
  /**
   * Begin masking until asked to stop.
   * @throws RuntimeException if some error occurs
   */
  public void run()
  throws RuntimeException {
    int priorityOriginal = Thread.currentThread().getPriority();
    Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
    try {
      m_doMasking = true;
      while (m_doMasking) {
        m_out.print(m_promptOverwrite);
        if (m_out.checkError())
          throw new RuntimeException("Console output error writing prompt");
        try {
          Thread.currentThread().sleep(1);
        } catch (InterruptedException ie) {
          Thread.currentThread().interrupt();
          return;
        }
      }
      m_out.print(m_setCursorToStart);
    } finally {
      Thread.currentThread().setPriority(priorityOriginal);
    }
  }
  //---------------------------------
```

```
      /**
       * Instructs the thread to stop masking.
       */
      public void stopMasking() {
        m_doMasking = false;
      }
      //---------------------------------
      /**
       * Returns a repeated char string.
       *
       * @param c the char to repeat
       * @param length the number of times to repeat the char
       * @throws RuntimeException if some error occurs
       */
      private static String repeatChars(char c, int length)
      throws RuntimeException {
        if (length < 0)
          throw new RuntimeException("repeatChars length is negative");
        StringBuffer sb = new StringBuffer(length);
        for (int i = 0; i < length; i++)
          sb.append(c);
        return sb.toString();
      }
    }
  }
```

Securing External Procedures

An external procedure is stored in a `.dll` or an `.so` file, separately from the database, and can be through a credential authentication.

About Securing External Procedures

For safety reasons, Oracle external procedures run in a process that is physically separate from the database.

In most cases, you configure this process to execute as a user other than the Oracle software account. When your application invokes this external procedure—such as when a library of `.dll` or `.so` files must be accessed—then Oracle Database creates an operating system process called `extproc`. By default, the `extproc` process communicates directly through your server process. In other words, if you do not use a credential, then Oracle Database creates an `extproc` process for you in the default Oracle Database server configuration, and runs `extproc` as the `oracle` software account. Alternatively, it can communicate through the Oracle Database listener.

See Also:

Guideline for Securing External Procedures

General Process for Configuring extproc for a Credential Authentication

For better security, you can configure the `extproc` process to be authenticated through a credential.

The general process is as follows:

1. You create a credential.

   The credential is in an encrypted container. Both public and private synonyms can refer to this credential. Configuring Authentication for External Procedures describes how to create this credential and configure your database to use it.
2. You make your initial connection to the database, which you are running in either a dedicated server or a shared server process.
3. Your application makes a call to an external procedure.

   If this is the first call, then Oracle Database creates an `extproc` process. Note that if you want to use a credential for `extproc`, then you cannot use the Oracle listener to spawn the `extproc` process.
4. The `extproc` process impersonates (that is, it runs on behalf of your supplied credential), loads the requisite `.dll`, `.so`, `.sl`, or `.a` file, and then sends your data between SQL and C.

extproc Process Authentication and Impersonation Expected Behaviors

The `extproc` process has a set of behaviors for authentication and impersonation.

Table 9-2 describes the expected behaviors of an `extproc` process based on possible authentication and impersonation scenarios.

Table 9-2 Expected Behaviors for extproc Process Authentication and Impersonation Settings

ENFORCE_CREDENTIAL Environment Variable Setting	PL/SQL Library with Credential?	GLOBAL_EXTPROC_CREDENTIAL Credential Existence[1]	Expected Behavior
`FALSE`	No	No	Uses the pre-release 12c authentication, which authenticates by operating system privilege of the owners of the Oracle listener or Oracle server process.
`FALSE`	No	Yes	Authenticates and impersonates with the Oracle instance-wide supplied `GLOBAL_EXTPROC_CREDENTIAL` ([2])
`FALSE`	Yes	No	Authenticates and impersonates with the credential defined in the PL/SQL library

Table 9-2 (Cont.) Expected Behaviors for extproc Process Authentication and Impersonation Settings

ENFORCE_CREDENTIAL Environment Variable Setting	PL/SQL Library with Credential?	GLOBAL_EXTPROC_CREDENTIAL Credential Existence[1]	Expected Behavior
`FALSE`	Yes	Yes	Authenticates and impersonates ([3])
`TRUE`	No	No	Returns error `ORA-28575: unable to open RPC connection to external procedure agent`
`TRUE`	No	Yes	Authenticates and impersonates with the Oracle system-wide supplied `GLOBAL_EXTPROC_CREDENTIAL` (footnote 1)
`TRUE`	Yes	No	Authenticates and impersonates with the credential defined in the PL/SQL library
`TRUE`	Yes	Yes	Authenticates and impersonates (footnote 2)

1 `GLOBAL_EXTPROC_CREDENTIAL` is a reserved credential name for the default credential if the credential is not explicitly specified and the `ENFORCE_CREDENTIAL` environment variable is set to `TRUE`. Therefore, Oracle strongly recommends that you create a credential by the that name if `ENFORCE_CREDENTIAL` is set to `TRUE`.

2 If only the `GLOBAL_EXTPROC_CREDENTIAL` credential is in use, then the `EXECUTE` privilege on this global credential is automatically granted to all users implicitly.

3 If both the PL/SQL library and the `GLOBAL_EXTPROC_CREDENTIAL` settings have credentials defined, then the credential of the PL/SQL library takes precedence.

Configuring Authentication for External Procedures

To configure a credential for `extproc` processes, you can use the `DBMS_CREDENTIAL` PL/SQL package.

1. Log in to SQL*Plus as a user who has been granted the `CREATE CREDENTIAL` or `CREATE ANY CREDENTIAL` privilege.

   ```
   sqlplus psmith
   Enter password: password
   Connected.
   ```

   In a multitenant environment, you must connect to the appropriate pluggable database (PDB). For example:

   ```
   sqlplus psmith@hpdb
   Enter password: password
   Connected.
   ```

   In addition, ensure that you also have the `CREATE LIBRARY` or `CREATE ANY LIBRARY` privilege, and the `EXECUTE` object privilege on the library that contains the external calls.

2. Using the `DBMS_CREDENTIAL` PL/SQL package, create a new credential.

For example:

```
BEGIN
  DBMS_CREDENTIAL.CREATE_CREDENTIAL (
    credential_name   => 'smith_credential',
    user_name         => 'tjones',
    password          => 'password')
END;
/
```

In this example:

- `credential_name`: Enter the name of the credential. Optionally, prefix it with the name of a schema (for example, `psmith.smith_credential`). If the `ENFORCE_CREDENTIAL` environment variable is set to `TRUE`, then you should create a credential with credential_name `GLOBAL_EXTPROC_CREDENTIAL`.
- `user_name`: Enter a valid operating system user name to be to used to run as the user.
- `password`: Enter the password for the `user_name` user.

3. Associate the credential with a PL/SQL library.

   For example:

   ```
   CREATE OR REPLACE LIBRARY ps_lib
    AS 'smith_lib.so' IN DLL_LOC
    CREDENTIAL smith_credential;
   ```

   In this example, `DLL_LOC` is a directory object that points to the `$ORACLE_HOME/bin` directory. Oracle does not recommend using absolute paths to the DLL.

   When the PL/SQL library is loaded by an external procedure call through the `extproc` process, `extproc` now can authenticate and impersonate on behalf of the defined `smith_credential` credential.

4. Register the external procedure by creating a PL/SQL procedure or function that tells PL/SQL how to call the external procedure and what arguments to pass to it.

   For example, to create a function that registers an external procedure that was written in C, only use the `AS LANGUAGE C`, `LIBRARY`, and `NAME` clauses of the `CREATE FUNCTION` statement, as follows:

   ```
   CREATE OR REPLACE FUNCTION getInt (x VARCHAR2, y BINARY_INTEGER)
   RETURN BINARY_INTEGER
   AS LANGUAGE C
   LIBRARY ps_lib
   NAME "get_int_vals"
   PARAMETERS (x STRING, y int);
   ```

See Also:

- Guideline for Securing External Procedures
- *Oracle Database PL/SQL Packages and Types Reference*, for information about the `DBMS_CREDENTIAL` package
- *Oracle Call Interface Programmer's Guide* for information about the `extproc` agent
- *Oracle Database Net Services Administrator's Guide* for detailed information about the `extproc.ora` file

External Procedures for Legacy Applications

For maximum security, set the `ENFORCE_CREDENTIAL` environment variable to `TRUE`.

However, if you must accommodate backward compatibility, then set `ENFORCE_CREDENTIAL` to `FALSE`. `FALSE` enables the `extproc` process to authenticate, impersonate, and perform user-defined callout functions on behalf of the supplied credential when either of the following occurs:

- The credential is defined with a PL/SQL library.
- The credential is not defined but the `GLOBAL_EXTPROC_CREDENTIAL` credential exists.

If neither of these credential definitions is in place, then setting the `ENFORCE_CREDENTIAL` parameter to `FALSE` sets the `extproc` process to be authenticated by the operating system privilege of the owners of the Oracle listener or Oracle server process.

For legacy applications that run on top of `extproc` processes, ideally you should change the legacy application code to associate all alias libraries with credentials. If you cannot do this, then Oracle Database uses the `GLOBAL_EXTPROC_CREDENTIAL` credential to determine how authentication will be handled. If the `GLOBAL_EXTPROC_CREDENTIAL` credential is not defined, then the `extproc` process is authenticated by the operating system privilege of the owners of the Oracle listener or Oracle server process.

Managing Application Privileges

Most database applications involve different privileges on different schema objects.

Keeping track of the privileges that are required for each application can be complex. In addition, authorizing users to run an application can involve many `GRANT` operations.

- To simplify application privilege management, create a role for each application and grant that role all the privileges a user must run the application.

  In fact, an application can have several roles, each granted a specific subset of privileges that allow greater or lesser capabilities while running the application.

For example, suppose every administrative assistant uses the Vacation application to record the vacation taken by members of the department. To best manage this application, you should:

1. Create a `VACATION` role.

2. Grant all privileges required by the Vacation application to the `VACATION` role.
3. Grant the `VACATION` role to all administrative assistants. Better yet, create a role that defines the privileges the administrative assistants have, and then grant the `VACATION` role to that role.

See Also:

- Configuring Privilege and Role Authorization, for a complete discussion of creating, enabling, and disabling roles, and granting and revoking privileges
- User Privilege and Role Data Dictionary Views for more information about the security uses of the `ROLE_TAB_PRIVS`, `ROLE_SYS_PRIVS`, and `DBA_ROLE_PRIVS` data dictionary views

Advantages of Using Roles to Manage Application Privileges

Grouping application privileges in a role aids privilege management.

Consider the following administrative options:

- You can grant the role, rather than many individual privileges, to those users who run the application. Then, as employees change jobs, you need to grant or revoke only one role, rather than many privileges.
- You can change the privileges associated with an application by modifying only the privileges granted to the role, rather than the privileges held by all users of the application.
- You can determine the privileges that are necessary to run a particular application by querying the `ROLE_TAB_PRIVS` and `ROLE_SYS_PRIVS` data dictionary views.
- You can determine which users have privileges on which applications by querying the `DBA_ROLE_PRIVS` data dictionary view.

Creating Secure Application Roles to Control Access to Applications

A secure application role is only enabled through its associated PL/SQL package or procedure.

See Also:

- *Oracle Database 2 Day + Security Guide* for a tutorial on creating a secure application role
- *Oracle Database Vault Administrator's Guide* for information about create a privilege analysis policy that tracks the privileges used by secure application roles

Step 1: Create the Secure Application Role

The `CREATE ROLE` statement with the `IDENTIFIED USING` clause creates a secure application role.

You must have the `CREATE ROLE` system privilege to execute this statement.

For example, to create a secure application role called `hr_admin` that is associated with the `sec_mgr.hr_admin` package:

1. Create the security application role as follows:

   ```
   CREATE ROLE hr_admin IDENTIFIED USING sec_mgr.hr_admin_role_check;
   ```

   This statement indicates the following:

   - The role `hr_admin` to be created is a secure application role.
   - The role can only be enabled by modules defined inside the PL/SQL procedure `sec_mgr.hr_admin_role_check`. At this stage, this procedure does not need to exist; Step 2: Create a PL/SQL Package to Define the Access Policy for the Application explains how to create the package or procedure.

2. Grant the security application role the privileges you would normally associate with this role.

   For example, to grant the `hr_admin` role `SELECT`, `INSERT`, `UPDATE`, and `DELETE` privileges on the `HR.EMPLOYEES` table, you enter the following statement:

   ```
   GRANT SELECT, INSERT, UPDATE, DELETE ON HR.EMPLOYEES TO hr_admin;
   ```

   Do not grant the role directly to the user. The PL/SQL procedure or package does that for you, assuming the user passes its security policies.

Step 2: Create a PL/SQL Package to Define the Access Policy for the Application

You can create a PL/SQL package that defines the access policy for your application.

About Creating a PL/SQL Package to Define the Access Policy for an Application

To enable or disable the secure application role, you must create the security policies of the role within a PL/SQL package.

You also can create an individual procedure to do this, but a package lets you group a set of procedures together. This lets you group a set of policies that, used together, present a solid security strategy to protect your applications. For users (or potential intruders) who fail the security policies, you can add auditing checks to the package to record the failure. Typically, you create this package in the schema of the security administrator.

The package or procedure must accomplish the following:

- **It must use invoker's rights to enable the role.**To create the package using invoker's rights, you must set the `AUTHID` property to `CURRENT_USER`. You cannot create the package by using definer's rights.

  For more information about invoker's rights and definer's rights, see *Oracle Database PL/SQL Language Reference*.
- **It must include one or more security checks to validate the user.** One way to validate users is to use the `SYS_CONTEXT` SQL function. See *Oracle Database SQL Language Reference* for more information about `SYS_CONTEXT`. To find session information for a user, you can use `SYS_CONTEXT` with an application context. See Using Application Contexts to Retrieve User Information, for details.
- **It must issue a SET ROLE SQL statement or DBMS_SESSION.SET_ROLE procedure when the user passes the security checks.** Because you create the package using invoker's rights, you must set the role by issuing the `SET ROLE` SQL statement or the `DBMS_SESSION.SET_ROLE` procedure. (However, you cannot use the `SET ROLE ALL` statement for this type of role enablement.) The PL/SQL embedded SQL syntax does not support the `SET ROLE` statement, but you can invoke `SET ROLE` by using dynamic SQL (for example, with `EXECUTE IMMEDIATE`).

  For more information about `EXECUTE IMMEDIATE`, see *Oracle Database PL/SQL Language Reference*.

Because of the way that you must create this package or procedure, you cannot use a logon trigger to enable or disable a secure application role. Instead, invoke the package directly from the application when the user logs in, before the user must use the privileges granted by the secure application role.

Creating a PL/SQL Package or Procedure to Define the Access Policy for an Application

The PL/SQL package or procedure that you create must use invoker’s rights to define the access policy.

For example, suppose you wanted to restrict anyone using the `hr_admin` role to employees who are on site (that is, using certain terminals) and between the hours of 8 a.m. and 5 p.m. As the system or security administrator, you can create a procedure that defines the access policy for the application.

1. Create the procedure as follows:

```
CREATE OR REPLACE PROCEDURE hr_admin_role_check
 AUTHID CURRENT_USER
 AS
 BEGIN
  IF (SYS_CONTEXT ('userenv','ip_address')
    BETWEEN '192.0.2.10' and '192.0.2.20'
     AND
    TO_CHAR (SYSDATE, 'HH24') BETWEEN 8 AND 17)
  THEN
    EXECUTE IMMEDIATE 'SET ROLE hr_admin';
  END IF;
 END;
/
```

In this example:

- `AUTHID CURRENT_USER` sets the AUTHID property to CURRENT_USER so that invoker's rights can be used.
- `IF (SYS_CONTEXT ('userenv','ip_address')` validates the user by using the SYS_CONTEXT SQL function to retrieve the user session information.
- `BETWEEN ... TO_CHAR` creates a test to grant or deny access. The test restricts access to users who are on site (that is, using certain terminals) and working between the hours of 8:00 a.m. and 5:00 p.m. If the user passes this check, the `hr_admin` role is granted.
- `THEN... EXECUTE` grants the role to the user by issuing the `SET ROLE` statement using the `EXECUTE IMMEDIATE` command, assuming the user passes the test.

2. Grant `EXECUTE` permissions for the `hr_admin_role_check` procedure to any user who was assigned it.

   For example:

   ```
   GRANT EXECUTE ON hr_admin_role_check TO psmith;
   ```

Testing the Secure Application Role

As a user who has been granted the secure application role, try performing an action that requires the privileges the role grants.

When you log in as a user who has been granted the secure application role, the role is then enabled.

1. Log in to the database session as the user.

   For example:

   ```
   CONNECT PSMITH@hrpdb
   Enter password: password
   ```

2. Perform an action that requires the privileges the secure application role grants.

   For example, if the role grants the `EXECUTE` privilege for a procedure called `sec_admin.hr_admin_role_check`:

   ```
   EXECUTE sec_admin.hr_admin_role_check;
   ```

Association of Privileges with User Database Roles

Ensure that users have only the privileges associated with the current database role.

Why Users Should Only Have the Privileges of the Current Database Role

A single user can use many applications and associated roles.

However, you should ensure that the user has only the privileges associated with the current database role.

Consider the following scenario:

- The `ORDER` role (for an application called Order) contains the `UPDATE` privilege for the `INVENTORY` table.
- The `INVENTORY` role (for an application called Inventory) contains the `SELECT` privilege for the `INVENTORY` table.
- Several order entry clerks were granted both the `ORDER` and `INVENTORY` roles.

In this scenario, an order entry clerk who was granted both roles can use the privileges of the `ORDER` role when running the `INVENTORY` application to update the `INVENTORY` table. The problem is that updating the `INVENTORY` table is not an authorized action for the `INVENTORY` application. It is an authorized action for the `ORDER` application. To avoid this problem, use the `SET ROLE` statement as explained in the following section.

Use of the SET ROLE Statement to Automatically Enable or Disable Roles

You can use a `SET ROLE` statement at the beginning of each application to automatically enable its associated role and to disable all others.

This way, each application dynamically enables particular privileges for a user only when required. The `SET ROLE` statement simplifies privilege management. You control what information users can access and when they can access it. The `SET ROLE` statement also keeps users operating in a well-defined privilege domain. If a user obtains privileges only from roles, then the user cannot combine these privileges to perform unauthorized operations.

Protecting Database Objects by Using Schemas

A schema is a security domain that can contain database objects. Privileges granted to users and roles control access to these database objects.

Protecting Database Objects in a Unique Schema

Think of most schemas as user names: the accounts that enable users to connect to a database and access the database objects.

However, a *unique schema* does not allow connections to the database, but is used to contain a related set of objects. Schemas of this sort are created as typical users, and yet are not granted the `CREATE SESSION` system privilege (either explicitly or through a role).

- To protect the objects, temporarily grant the `CREATE SESSION` and `RESOURCE` privilege to a unique schema if you want to use the `CREATE SCHEMA` statement to create multiple tables and views in a single transaction.

For example, a given schema might own the schema objects for a specific application. If application users have the privileges to do so, then they can connect to the database using typical database user names and use the application and the corresponding objects. However, no user can connect to the database using the schema set up for the application. This configuration prevents access to the associated objects through the schema, and provides another layer of protection for schema objects. In this case, the application could issue an `ALTER SESSION SET CURRENT_SCHEMA` statement to connect the user to the correct application schema.

Protection of Database Objects in a Shared Schema

For many applications, users only need access to an application schema; they do not need their own accounts or schemas in the database.

For example, users John, Firuzeh, and Jane are all users of the Payroll application, and they need access to the `payroll` schema on the `finance` database. None of them need to create their own objects in the database. They need to only access the `payroll` objects. To address this issue, Oracle Database provides the enterprise users, which are schema-independent users.

Enterprise users, users managed in a directory service, do not need to be created as database users because they use a shared database schema. To reduce administration costs, you can create an enterprise user once in the directory, and point the user at a shared schema that many other enterprise users can also access.

See Also:

Oracle Database Enterprise User Security Administrator's Guide for more information about managing enterprise users

Object Privileges in an Application

When you design an application, consider the types of users and the level access they need.

What Application Developers Must Know About Object Privileges

Object privileges enable end users to perform actions on objects such as tables, views, sequences, procedures, functions, or packages.

Table 9-3 summarizes the object privileges available for each type of object.

Table 9-3 How Privileges Relate to Schema Objects

Object Privilege	Applies to Table?	Applies to View?	Applies to Sequence?	Applies to Procedure?[1]
`ALTER`	Yes	No	Yes	No
`DELETE`	Yes	Yes	No	No
`EXECUTE`	No	No	No	Yes
`INDEX`	Yes[2]	No	No	No
`INSERT`	Yes	Yes	No	No
`REFERENCES`	YesYes(5)	No	No	No
`SELECT`	Yes	Yes[3]	Yes	No

Table 9-3 (Cont.) How Privileges Relate to Schema Objects

Object Privilege	Applies to Table?	Applies to View?	Applies to Sequence?	Applies to Procedure?[1]
`UPDATE`	Yes	Yes	No	No

1 Standalone stored procedures, functions, and public package constructs
2 Privilege that cannot be granted to a role
3 Can also be granted for snapshots

SQL Statements Permitted by Object Privileges

As you implement and test your application, you should create each necessary role.

Test the usage scenario for each role to ensure that the users of your application will have proper access to the database. After completing your tests, coordinate with the administrator of the application to ensure that each user is assigned the proper roles.

Table 9-4 lists the SQL statements permitted by the object privileges shown in Table 9-3.

Table 9-4 SQL Statements Permitted by Database Object Privileges

Object Privilege	SQL Statements Permitted
`ALTER`	`ALTER` object (table or sequence) `CREATE TRIGGER ON` object (tables only)
`DELETE`	`DELETE FROM` object (table, view, or synonym)
`EXECUTE`	`EXECUTE` object (procedure or function) References to public package variables
`INDEX`	`CREATE INDEX ON` object (table, view, or synonym)
`INSERT`	`INSERT INTO` object (table, view, or synonym)
`REFERENCES`	`CREATE` or `ALTER TABLE` statement defining a `FOREIGN KEY` integrity constraint on object (tables only)
`SELECT`	`SELECT`...`FROM` object (table, view, synonym, or snapshot) SQL statements using a sequence

Parameters for Enhanced Security of Database Communication

Parameters can be used to manage security, such as handling bad packets from protocol errors or configuring the maximum number of authentication errors.

Bad Packets Received on the Database from Protocol Errors

The `SEC_PROTOCOL_ERROR_TRACE_ACTION` initialization parameter controls how trace files are managed when protocol errors are generated.

Networking communication utilities such as Oracle Call Interface (OCI) or Two-Task Common (TTC) can generate a large disk file containing the stack trace and heap dump when the server receives a bad packet, out-of-sequence packet, or a private or an unused remote procedure call.

Typically, this disk file can grow quite large. An intruder can potentially cripple a system by repeatedly sending bad packets to the server, which can result in disk flooding and Denial of Service (DOS) attacks. An unauthenticated client can also mount this type of attack.

You can prevent these attacks by setting the `SEC_PROTOCOL_ERROR_TRACE_ACTION` initialization parameter to one of the following values:

- `None`: Configures the server to ignore the bad packets and does not generate any trace files or log messages. Use this setting if the server availability is overwhelmingly more important than knowing that bad packets are being received.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = None
  ```

- `Trace` (default setting): Creates the trace files, but it is useful for debugging purposes, for example, when a network client is sending bad packets as a result of a bug.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = Trace
  ```

- `Log`: Writes a short, one-line message to the server trace file. This choice balances some level of auditing with system availability.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = Log
  ```

- `Alert`: Sends an alert message to a database administrator or monitoring console.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = Alert
  ```

Controlling Server Execution After Receiving a Bad Packet

The `SEC_PROTOCOL_ERROR_FURTHER_ACTION` initialization parameter controls server execution after the server receives a bad packet.

After Oracle Database detects a client or server protocol error, it must continue execution. However, this could subject the server to further bad packets, which could lead to disk flooding or denial-of-service attacks.

- To control the further execution of a server process when it is receiving bad packets from a potentially malicious client, set the `SEC_PROTOCOL_ERROR_FURTHER_ACTION` initialization parameter to one of the following values:

  – `Continue`: Continues the server execution. However, be aware that the server may be subject to further attacks.

    For example:

    ```
    SEC_PROTOCOL_ERROR_FURTHER_ACTION = Continue
    ```

- `(Delay,`*`m`*`)`: Delays the client *m* seconds before the server can accept the next request from the same client connection. This setting prevents malicious clients from excessively using server resources while legitimate clients experience a degradation in performance but can continue to function. When you enter this setting, enclose it in parentheses.

  For example:

  ```
  SEC_PROTOCOL_ERROR_FURTHER_ACTION = (Delay,3)
  ```

  If you are setting `SEC_PROTOCOL_ERROR_FURTHER_ACTION` by using the `ALTER SYSTEM` or `ALTER SESSION` SQL statement, then you must enclose the `Delay` setting in either single or double quotation marks.

  ```
  ALTER SYSTEM SEC_PROTOCOL_ERROR_FURTHER_ACTION = '(Delay,3)';
  ```

- `(Drop,`*`n`*`)`: Forcefully terminates the client connection after *n* bad packets. This setting enables the server to protect itself at the expense of the client, for example, loss of a transaction. However, the client can still reconnect, and attempt the same operation again. Enclose this setting in parentheses. The default value of `SEC_PROTOCOL_ERROR_FURTHER_ACTION` is `(Drop,3)`.

  For example:

  ```
  SEC_PROTOCOL_ERROR_FURTHER_ACTION = (Drop,10)
  ```

  Similar to the `Delay` setting, you must enclose the `Drop` setting in single or double quotation marks if you are using `ALTER SYSTEM` or `ALTER SESSION` to change this setting.

Configuration of the Maximum Number of Authentication Attempts

The `SEC_MAX_FAILED_LOGIN_ATTEMPTS` initialization parameter sets the number of authentication attempts before the database will drop a failed connection.

As part of connection creation, the listener starts the server process and attaches it to the client. Using this physical connection, the client is this able to authenticate the connection. After a server process starts, client authenticates with this server process. An intruder could start a server process, and then issue an unlimited number of authenticated requests with different user names and passwords in an attempt to gain access to the database.

You can limit the number of failed login attempts for application connections by setting the `SEC_MAX_FAILED_LOGIN_ATTEMPTS` initialization parameter to restrict the number of authentication attempts on a connection. After the specified number of authentication attempts fail, the database process drops the connection and the server process is terminated. By default, `SEC_MAX_FAILED_LOGIN_ATTEMPTS` is set to `3`.

Remember that the `SEC_MAX_FAILED_LOGIN_ATTEMPTS` initialization parameter is designed to prevent potential intruders from attacking your applications, as well as valid users who have forgotten their passwords. The `sqlnet.ora` `INBOUND_CONNECT_TIMEOUT` parameter and the `FAILED_LOGIN_ATTEMPTS` profile parameter also restrict failed logins, but the difference is that these two parameters only apply to valid user accounts.

For example, to limit the maximum attempts to 5, set `SEC_MAX_FAILED_LOGIN_ATTEMPTS` as follows in the `init`*`sid`*`.ora` initialization parameter file:

```
SEC_MAX_FAILED_LOGIN_ATTEMPTS = 5
```

Configuring the Display of the Database Version Banner

The `SEC_RETURN_SERVER_RELEASE_BANNER` initialization parameter can be used to prevent the display of detailed product information during authentication.

Detailed product version information should not be accessible before a client connection (including an Oracle Call Interface client) is authenticated. An intruder could use the database version to find information about security vulnerabilities that may be present in the database software.

- To restrict the display of the database version banner to unauthenticated clients, set the `SEC_RETURN_SERVER_RELEASE_BANNER` initialization parameter in the `init`*`sid`*`.ora` initialization parameter file to either `TRUE` or `FALSE`.

  By default, `SEC_RETURN_SERVER_RELEASE_BANNER` is set to `FALSE`.

For example, if you set it to `TRUE`, then Oracle Database displays the full correct database version. For example, for Release 12.2.0.0:

```
Oracle Database 12c Enterprise Edition Release 12.2.0.0 - Production
```

If a release number uses point release notation (for example, Oracle Database Release 12.2.0.1), then the banner displays as follows:

```
Oracle Database 12c Enterprise Edition Release 12.2.0.1 - Production
```

However, if in that same release, you set it to `NO`, then Oracle Database restricts the banner to display the following fixed text starting with Release 12.2, which instead of 12.2.0.1 is 12.2.0.0.0:

```
Oracle Database 12c Release 12.2.0.0.0 - Production
```

Configuring Banners for Unauthorized Access and Auditing User Actions

The `SEC_USER_UNAUTHORIZED_ACCESS_BANNER` and `SEC_USER_AUDIT_ACTION_BANNER` initialization parameters control the display of banners for unauthorized access and for auditing users.

You should create and configure banners to warn users against unauthorized access and possible auditing of user actions. The notices are available to the client application when it logs into the database.

- To configure these banners to display, set the following `sqlnet.ora` parameters on the database server side to point to a text file that contains the banner information:
  - `SEC_USER_UNAUTHORIZED_ACCESS_BANNER`. For example:

    ```
    SEC_USER_UNAUTHORIZED_ACCESS_BANNER = /opt/Oracle/12c/dbs/unauthaccess.txt
    ```
  - `SEC_USER_AUDIT_ACTION_BANNER`. For example:

    ```
    SEC_USER_AUDIT_ACTION_BANNER = /opt/Oracle/12c/dbs/auditactions.txt
    ```

By default, these parameters are not set. In addition, be aware that there is a 512-byte limitation for the number of characters used for the banner text.

After you set these parameters, the Oracle Call Interface application must use the appropriate OCI APIs to retrieve these banners and present them to the end user.

Part III
Controlling Access to Data

Part III describes how to control access to data.

10

Using Application Contexts to Retrieve User Information

An application context stores user identification that can enable or prevent a user from accessing data in the database.

About Application Contexts

An application context provides many benefits in controlling the access that a user has to data.

What Is an Application Context?

An **application context** is a set of name-value pairs that Oracle Database stores in memory.

The context has a label called a **namespace** (for example, `empno_ctx` for an application context that retrieves employee IDs). This context enables Oracle Database to find information about both database and nondatabase users during authentication.

Inside the context are the name-value pairs (an associative array): the name points to a location in memory that holds the value. An application can use the application context to access session information about a user, such as the user ID or other user-specific information, or a client ID, and then securely pass this data to the database.

You can then use this information to either permit or prevent the user from accessing data through the application. You can use application contexts to authenticate both database and non-database users.

Components of the Application Context

An application context has two components, comprising a name-value pair.

These components are as follows:

- **Name.** Refers to the name of the attribute set that is associated with the value. For example, if the `empno_ctx` application context retrieves an employee ID from the `HR.EMPLOYEES` table, it could have a name such as `employee_id`.
- **Value.** Refers to a value set by the attribute. For example, for the `empno_ctx` application context, if you wanted to retrieve an employee ID from the `HR.EMPLOYEES` table, you could create a value called `emp_id` that sets the value for this ID.

Think of an application context as a global variable that holds information that is accessed during a database session. To set the values for a secure application context, you must create a PL/SQL package procedure that uses the `DBMS_SESSION.SET_CONTEXT` procedure. In fact, this is the only way that you can set application context values if the context is not marked `INITIALIZED EXTERNALLY` or `INITIALIZED GLOBALLY`. You can assign the values to the application context attributes at run time, not when you create the application context. Because the **trusted** procedure, and not the user, assigns the values, it is a called secure application context. For client-session based application contexts, another way to set the application context is to use Oracle Call Interface (OCI) calls.

Where Are the Application Context Values Stored?

Oracle Database stores the application context values in a secure data cache.

This cache is available in the User Global Area (UGA) or the System (sometimes called "Shared") Global Area (SGA). This way, the application context values are retrieved during the session. Because the application context stores the values in this data cache, it increases performance for your applications. You can use an application context by itself, with Oracle Virtual Private Databases policies, or with other fine-grained access control policies.

Benefits of Using Application Contexts

Most applications contain the kind of information that can be used for application contexts.

For example, in an order entry application that uses a table containing the columns `ORDER_NUMBER` and `CUSTOMER_NUMBER`, you can use the values in these columns as security attributes to restrict access by a customer to his or her own orders, based on the ID of that customer.

Application contexts are useful for the following purposes:

- Enforcing fine-grained access control (for example, in Oracle Virtual Private Database polices)
- Preserving user identity across multitier environments
- Enforcing stronger security for your applications, because the application context is controlled by a trusted procedure, not the user
- Increasing performance by serving as a secure data cache for attributes needed by an application for fine-grained auditing or for use in PL/SQL conditional statements or loops

  This cache saves the repeated overhead of querying the database each time these attributes are needed. Because the application context stores session data in cache rather than forcing your applications to retrieve this data repeatedly from a table, it greatly improves the performance of your applications.
- Serving as a holding area for name-value pairs that an application can define, modify, and access

How Editions Affects Application Context Values

Oracle Database sets the application context in all editions that are affected by the application context package.

The values the application context sets are visible in all editions the application context affects. To find all editions in your database, and whether they are usable, you can query the `ALL_EDITIONS` data dictionary view.

See Also:

Oracle Database Development Guide for detailed information about editions

Application Contexts in a Multitenant Environment

Where you create an application in a multitenant environment determines where you must create the application context.

If an application is installed in the application root or CDB root, then it becomes accessible across the application container or system container and associated application PDBs. You will need to create a common application context in this root.

When you create a common application context for use with an application container, note the following:

- You can create application contexts in a multitenant environment by setting the `CONTAINER` clause in the `CREATE CONTEXT` SQL statement. For example, to create a common application context in the application root, you must execute `CREATE CONTEXT` with `CONTAINER` set to `ALL`. To create the application context in a PDB, set `CONTAINER` to `CURRENT`.
- You cannot use the same name for a local application context for a common application context. You can find the names of existing application contexts by running the following query:

  ```
  SELECT OBJECT_NAME FROM DBA_OBJECTS WHERE OBJECT_TYPE ='CONTEXT';
  ```
- The PL/SQL package that you create to manage a common application context must be a common PL/SQL package. That is, it must exist in the application root or CDB root. If you create the application context for a specific PDB, then you must store the associated PL/SQL package in that PDB.
- The name-value pairs that you set under a common session application context from an application container or a system container for a common application context are not accessible from other application containers or system containers when a common user accesses a different container.
- The name-value pairs that you set under a common global application context from an application container or a system container, are accessible only within the same container in the same user session.
- An application can retrieve the value of an application context whether it resides in the application root, the CDB root, or a PDB.
- During a plug-in operation of a PDB into a CDB or an application container, if the name of the common application context conflicts with a PDB's local application context, then the PDB must open in restricted mode. A database administrator would then need to correct the conflict before opening the PDB in normal mode.
- During an unplug operation, a common application context retains its common semantics, so that later on, if the PDB is plugged into another CDB where a common application context with the same name exists, it would continue to behave like a common object. If a PDB is plugged into an application container or a system container, where the same common application context does not exist, then it behaves like a local object.

To find if an application context is a local application context or an application common application context, query the SCOPE column of the DBA_CONTEXT or ALL_CONTEXT data dictionary view.

Types of Application Contexts

There are three general categories of application contexts.

These categories are as follows:

- **Database session-based application contexts.** This type retrieves data that is stored in the database user session (that is, the UGA) cache. There are three categories of database session-based application contexts:
  - **Initialized locally.** Initializes the application context locally, to the session of the user.
  - **Initialized externally.** Initializes the application context from an Oracle Call Interface (OCI) application, a job queue process, or a connected user database link.
  - **Initialized globally.** Uses attributes and values from a centralized location, such as an LDAP directory.

  Using Database Session-Based Application Contexts describes this type of application context.
- **Global application contexts.** This type retrieves data that is stored in the System Global Area (SGA) so that it can be used for applications that use a sessionless model, such as middle-tier applications in a three-tiered architecture. A global application context is useful if the session context must be shared across sessions, for example, through connection pool implementations.

  Global Application Contexts describes this type.
- **Client session-based application contexts.** This type uses Oracle Call Interface functions on the client side to set the user session data, and then to perform the necessary security checks to restrict user access.

  Using Client Session-Based Application Contexts describes this type.

Table 10-1 summarizes the different types of application contexts.

Table 10-1 Types of Application Contexts

Application Context Type	Stored in UGA	Stored in SGA	Supports Connected User Database Links	Supports Centralized Storage of Users' Application Context	Supports Sessionless Multitier Applications
Database session-based application context initialized locally	Yes	No	No	No	No
Database session-based application context initialized externally	Yes	No	Yes	No	No
Database session-based application context initialized globally	Yes	No	No	Yes	No

Table 10-1 (Cont.) Types of Application Contexts

Application Context Type	Stored in UGA	Stored in SGA	Supports Connected User Database Links	Supports Centralized Storage of Users' Application Context	Supports Sessionless Multitier Applications
Global application context	No	Yes	No	No	Yes
Client session-based application context	Yes	No	Yes	No	Yes

Using Database Session-Based Application Contexts

A database session-based application context enables you to retrieve session-based information about a user.

About Database Session-Based Application Contexts

A database session-based application context retrieves session information for database users.

This type of application context uses a PL/SQL procedure within Oracle Database to retrieve, set, and secure the data it manages.

The database session-based application context is managed entirely within Oracle Database. Oracle Database sets the values, and then when the user exits the session, automatically clears the application context values stored in cache. If the user connection ends abnormally, for example, during a power failure, then the PMON background process cleans up the application context data.You do not need to explicitly clear the application context from cache.

The advantage of having Oracle Database manage the application context is that you can centralize the application context management. Any application that accesses this database will need to use this application context to permit or prevent user access to that application. This provides benefits both in improved performance and stronger security.

Note:

If your users are application users, that is, users who are not in your database, consider using a global application context instead.

Components of a Database Session-Based Application Context

A database session-based application context retrieves and sets data for the context and then sets this context when a user logs in.

You must use three components to create and use a database session-based application context: the application context, a procedure to retrieve the data and set the context, and a way to set the context when the user logs in.

- **The application context.** You use the `CREATE CONTEXT` SQL statement to create an application context. This statement names the application context (namespace) and associates it with a PL/SQL procedure that is designed to retrieve session data and set the application context.
- **A PL/SQL procedure to perform the data retrieval and set the context.** About the Package That Manages the Database Session-Based Application Context describes the tasks this procedure must perform. Ideally, create this procedure within a package, so that you can include other procedures if you want (for example, to perform error checking tasks).
- **A way to set the application context when the user logs on.** Users who log on to applications that use the application context must run a PL/SQL package that sets the application context. You can achieve this with either a logon trigger that fires each time the user logs on, or you can embed this functionality in your applications.

Tutorial: Creating and Using a Database Session-Based Application Context shows how to create and use a database session-based application context that is initialized locally.

In addition, you can initialize session-based application contexts either externally or globally. Either method stores the context information in the user session.

- **External initialization.** This type can come from an OCI interface, a job queue process, or a connected user database link. See Initializing Database Session-Based Application Contexts Externally for detailed information.
- **Global initialization.** This type uses attributes and values from a centralized location, such as an LDAP directory. Initializing Database Session-Based Application Contexts Globally provides more information.

Creating Database Session-Based Application Contexts

A database session-based application context is a named object that stores the user's session information.

About Creating Database Session-Based Application Contexts

A database user session (UGA) stores session-based application context, using a user-created namespace.

Each application context must have a unique attribute and belong to a namespace. That is, context names must be unique within the database, not just within a schema.

You must have the `CREATE ANY CONTEXT` system privilege to create an application context, and the `DROP ANY CONTEXT` privilege to use the `DROP CONTEXT` statement if you want to drop the application context.

The ownership of the application context is as follows: Even though a user who has been granted the `CREATE ANY CONTEXT` and `DROP ANY CONTEXT` privileges can create and drop the application context, it is owned by the `SYS` schema. Oracle Database associates the context with the schema account that created it, but if you drop this user, the context still exists in the `SYS` schema. As user `SYS`, you can drop the application context.

You can find the names of existing application contexts by running the following query:

```
SELECT OBJECT_NAME FROM DBA_OBJECTS WHERE OBJECT_TYPE ='CONTEXT';
```

Creating a Database Session-Based Application Context

The CREATE CONTEXT SQL statement can be used to create a database session-based application context.

When you create a database session-based application context, you must create a namespace for the application context and then associate it with a PL/SQL package that manages the name-value pair that holds the session information of the user. At the time that you create the context, the PL/SQL package does not need to exist, but it must exist at run time.

- To create a database session-based application context, use the CREATE CONTEXT SQL statement.

  For example:

  ```
  CREATE CONTEXT empno_ctx USING set_empno_ctx_pkg CONTAINER = CURRENT;
  ```

In this example:

- empno_ctx is the context namespace.
- set_empno_ctx_pkg is the package (which does not need to exist when you create the context) that sets attributes for the empno_ctx namespace. Step 3: Create a Package to Retrieve Session Data and Set the Application Context shows an example of how to create a package that can be used with this application context.
- CONTAINER creates the application context in the current PDB. To create the application context in the application or CDB root, you must set CONTAINER to ALL.

Notice that when you create the context, you do not set its name-value attributes in the CREATE CONTEXT statement. Instead, you set these in the PL/SQL package that you associate with the application context. The reason you must do this is to prevent a malicious user from changing the context attributes without proper attribute validation. Ensure that this package is in the same container as the application context. For example, if you created the application context in a PDB, then the PL/SQL package must reside in that PDB.

Note:

You cannot create a context called CLIENTCONTEXT. This word is reserved for use with client session-based application contexts. See Using Client Session-Based Application Contexts for more information about this type of application context.

Database Session-Based Application Contexts for Multiple Applications

For each application, you can create an application context that has its own attributes.

Suppose, for example, you have three applications: General Ledger, Order Entry, and Human Resources.

You can specify different attributes for each application:

- For the order entry application context, you could specify the attribute CUSTOMER_NUMBER.

- For the general ledger application context, you could specify the attributes `SET_OF_BOOKS` and `TITLE`.
- For the human resources application context, you could specify the attributes `ORGANIZATION_ID`, `POSITION`, and `COUNTRY`.

The data the attributes access is stored in the tables behind the applications. For example, the order entry application uses a table called `OE.CUSTOMERS`, which contains the `CUSTOMER_NUMBER` column, which provides data for the `CUSTOMER_NUMBER` attribute. In each case, you can adapt the application context to your precise security needs.

Creating a Package to Set a Database Session-Based Application Context

A PL/SQL package can be used to retrieve the session information and set the name-value attributes of the application context.

About the Package That Manages the Database Session-Based Application Context

This defines procedures that manage the session data represented by the application context.

This package is usually created in the security administrator schema. The package must perform the following tasks:

- **Retrieve session information.** To retrieve the user session information, you can use the `SYS_CONTEXT` SQL function. The `SYS_CONTEXT` function returns the value of the parameter associated with the context namespace. You can use this function in both SQL and PL/SQL statements. Typically, you will use the built-in `USERENV` namespace to retrieve the session information of a user. You also can use the `SYS_SESSION_ROLES` namespace to indicate whether the specified role is currently enabled for the session.
- **Set the name-value attributes of the application context you created with CREATE CONTEXT.** You can use the `DBMS_SESSION.SET_CONTEXT` procedure to set the name-value attributes of the application context. The name-value attributes can hold information such as the user ID, IP address, authentication mode, the name of the application, and so on. The values of the attributes you set remain either until you reset them, or until the user ends the session. Note the following:
  - If the value of the parameter in the namespace already has been set, then `SET_CONTEXT` overwrites this value.
  - Be aware that any changes in the context value are reflected immediately and subsequent calls to access the value through the `SYS_CONTEXT` function will return the most recent value.
- **Be executed by users.** After you create the package, the user will need to execute the package when he or she logs on. You can create a logon trigger to execute the package automatically when the user logs on, or you can embed this functionality in your applications. Remember that the application context session values are cleared automatically when the user ends the session, so you do not need to manually remove the session data.

It is important to remember that the procedure is a trusted procedure: It is designed to prevent the user from setting his or her own application context attribute values. The user runs the procedure, but the procedure sets the application context values, not the user.

See Also:

- *Oracle Database SQL Language Reference* for detailed information about the `SYS_CONTEXT` function
- Tutorial: Creating and Using a Database Session-Based Application Context shows how to create a database session-based application context

Using the SYS_CONTEXT Function to Retrieve Session Information

You can retrieve session information for the application context by using the `SYS_CONTEXT` function.

The `SYS_CONTEXT` function provides a default namespace, `USERENV`, which describes the current session of the user logged on. `SYS_CONTEXT` enables you to retrieve different types of session-based information about a user, such as the user host computer ID, host IP address, operating system user name, and so on. Remember that you only use `USERENV` to *retrieve* session data, not *set* it. The predefined attributes are listed in the description for the PL/SQL function in the *Oracle Database SQL Language Reference*.

- To use retrieve session information, set the namespace, parameter, and optionally, the length values of the `SYS_CONTEXT` function.

  For example:

  ```
  SYS_CONTEXT ('USERENV','HOST')
  ```

The syntax for the PL/SQL function `SYS_CONTEXT` is as follows:

```
SYS_CONTEXT ('namespace','parameter'[,length])
```

In this specification:

- *`namespace`* is the name of the application context. You can specify either a string or an expression that evaluates to a string. The `SYS_CONTEXT` function returns the value of parameter associated with the context namespace at the current instant. If the value of the parameter in the namespace already has been set, then `SET_CONTEXT` overwrites this value.
- *`parameter`* is a parameter within the *`namespace`* application context. This value can be a string or an expression.
- *`length`* is the default maximum size of the return type, which is 256 bytes, but you can override the length by specifying a value up to 4000 bytes. Enter a value that is a `NUMBER` data type, or a value that can be can be implicitly converted to `NUMBER`. The data type of the `SYS_CONTEXT` return type is a `VARCHAR2`. This setting is optional.

Note:

The USERENV application context namespace replaces the USERENV function provided in earlier Oracle Database releases.

Checking the SYS_CONTEXT Settings

You can check the SYS_CONTEXT settings, which are stored in the DUAL table.

The DUAL table is a small table in the data dictionary that Oracle Database and user-written programs can reference to guarantee a known result. This table has one column called DUMMY and one row that contains the value X.

- To check the SYS_CONTEXT settings, issue a SELECT SQL statement on the DUAL table.

For example, to find the host computer on which you are logged, assuming that you are logged on to the SHOBEEN_PC host computer under EMP_USERS:

```
SELECT SYS_CONTEXT ('USERENV', 'HOST') FROM DUAL;

SYS_CONTEXT(USERENV,HOST)
-------------------------
EMP_USERS\SHOBEEEN_PC
```

Dynamic SQL with SYS_CONTEXT

During a session in which you expect a change in policy between executions of a given query, the query must use dynamic SQL.

You must use dynamic SQL because static SQL and dynamic SQL parse statements differently:

- Static SQL statements are parsed at compile time. They are not parsed again at execution time for performance reasons.
- Dynamic SQL statements are parsed every time they are executed.

Consider a situation in which Policy A is in force when you compile a SQL statement, and then you switch to Policy B and run the statement. With static SQL, Policy A remains in force. Oracle Database parses the statement at compile time, but does not parse it again upon execution. With dynamic SQL, Oracle Database parses the statement upon execution, then the switch to Policy B takes effect.

For example, consider the following policy:

```
EMPLOYEE_NAME = SYS_CONTEXT ('USERENV', 'SESSION_USER')
```

The policy EMPLOYEE_NAME matches the database user name. It is represented in the form of a SQL predicate in Oracle Virtual Private Database: the predicate is considered a policy. If the predicate changes, then the statement must be parsed again to produce the correct result.

SYS_CONTEXT in a Parallel Query

If you use SYS_CONTEXT inside a SQL function that is embedded in a parallel query, then the function includes the application context.

Consider a user-defined function within a SQL statement, which sets the user ID to 5:

```
CREATE FUNCTION set_id
 RETURN NUMBER IS
BEGIN
 IF SYS_CONTEXT ('hr', 'id') = 5
   THEN RETURN 1; ELSE RETURN 2;
 END IF;
END;
```

Now consider the following statement:

```
SELECT * FROM emp WHERE set_id( ) = 1;
```

When this statement is run as a parallel query, the user session, which contains the application context information, is propagated to the parallel execution servers (query child processes).

SYS_CONTEXT with Database Links

The `SYS_CONTEXT` function is compatible with the use of database links.

When SQL statements within a user session involve database links, Oracle Database runs the `SYS_CONTEXT` function at the host computer of the database link, and then captures the context information in the host computer.

If remote PL/SQL procedure calls are run on a database link, then Oracle Database runs any `SYS_CONTEXT` function inside such a procedure at the destination database of the link.

In this case, only externally initialized application contexts are available at the database link destination site. For security reasons, Oracle Database propagates only the externally initialized application context information to the destination site from the initiating database link site.

DBMS_SESSION.SET_CONTEXT for Setting Session Information

After `SYS_CONTEXT` retrieves the session data of a user, you can set the application context values from the user session.

To set the context values, you can use the `DBMS_SESSION.SET_CONTEXT` procedure. You must have the `EXECUTE` privilege for the `DBMS_SESSION` PL/SQL package.

The syntax for `DBMS_SESSION.SET_CONTEXT` is as follows:

```
DBMS_SESSION.SET_CONTEXT (
   namespace VARCHAR2,
   attribute VARCHAR2,
   value     VARCHAR2,
   username  VARCHAR2,
   client_id VARCHAR2);
```

In this specification:

- `namespace` is the namespace of the application context to be set, limited to 30 bytes. For example, if you were using a namespace called `custno_ctx`, you would specify it as follows:

  ```
  namespace => 'custno_ctx',
  ```

- `attribute` is the attribute of the application context to be set, limited to 30 bytes. For example, to create the `ctx_attrib` attribute for the `custno_ctx` namespace:

  ```
  attribute => 'ctx_attrib',
  ```

- `value` is the value of the application context to be set, limited to 4000 bytes. Typically, this is the value retrieved by the `SYS_CONTEXT` function and stored in a variable. For example:

  ```
  value => ctx_value,
  ```

- `username` is the database user name attribute of the application context. The default is `NULL`, which permits any user to access the session. For database session-based application contexts, omit this setting so that it uses the `NULL` default. This setting is optional.

  The `username` and `client_id` parameters are used for globally accessed application contexts. See DBMS_SESSION.SET_CONTEXT username and client_id Parameters for more information.

- `client_id` is the application-specific `client_id` attribute of the application context (64-byte maximum). The default is `NULL`, which means that no client ID is specified. For database session-based application contexts, omit this setting so that it uses the `NULL` default.

See Also:

- Tutorial: Creating and Using a Database Session-Based Application Context for how to create a package that retrieves the user session information and then sets the application context based on this information
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DBMS_SESSION.SET_CONTEXT` procedure
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DBMS_SESSION` package

Example: Simple Procedure to Create an Application Context Value

You can use the `DBMS_SESSION.SET_CONTEXT` statement in a procedure to set an application context value.

Example 10-1 shows how to create a simple procedure that creates an attribute for the `empno_ctx` application context.

Example 10-1   Simple Procedure to Create an Application Context Value

```
CREATE OR REPLACE PROCEDURE set_empno_ctx_proc(
  emp_value IN VARCHAR2)
 IS
 BEGIN
  DBMS_SESSION.SET_CONTEXT('empno_ctx', 'empno_attrib', emp_value);
 END;
/
```

In this example:

- `emp_value IN VARCHAR2` takes `emp_value` as the input parameter. This parameter specifies the value associated with the application context attribute `empno_attrib`. The limit is 4000 bytes.
- `DBMS_SESSION.SET_CONTEXT('empno_ctx', 'empno_attrib', emp_value)` sets the value of the application context by using the `DBMS_SESSION.SET_CONTEXT` procedure as follows:
  - `'empno_ctx'` refers to the application context namespace. Enclose its name in single quotation marks.
  - `'empno_attrib'` creates the attribute associated with the application context namespace.
  - `emp_value` specifies the value for the `empno_attrib` attribute. Here, it refers to the `emp_value` parameter.

At this stage, you can run the `set_empno_ctx_proc` procedure to set the application context:

```
EXECUTE set_empno_ctx_proc ('42783');
```

(In a real world scenario, you would set the application context values in the procedure itself, so that it becomes a trusted procedure. This example is only used to show how data can be set for demonstration purposes.)

To check the application context setting, run the following `SELECT` statement:

```
SELECT SYS_CONTEXT ('empno_ctx', 'empno_attrib') empno_attrib FROM DUAL;

EMPNO_ATTRIB
--------------
42783
```

You can also query the `SESSION_CONTEXT` data dictionary view to find all the application context settings in the current session of the database instance. For example:

```
SELECT * FROM SESSION_CONTEXT;

NAMESPACE                ATTRIBUTE          VALUE
--------------------------------------------------
EMPNO_CTX                EMP_ID             42783
```

Logon Triggers to Run a Database Session Application Context Package

Users must run database session application context package after when they log in to the database instance.

You can create a logon trigger that handles this automatically. You do not need to grant the user `EXECUTE` permissions to run the package.

Note the following:

- **If the PL/SQL package procedure called by the logon trigger has any unhandled exceptions or raises any exceptions (because, for example, a security check failed), then the logon trigger fails.** When the logon trigger fails, the logon fails, that is, the user is denied permission to log in to the database.

- **Logon triggers may affect performance.** In addition, test the logon trigger on a sample schema user first before creating it for the database. That way, if there is an error, you can easily correct it.
- **Be aware of situations in which if you have a changing set of books, or if positions change constantly.** In these cases, the new attribute values may not be picked up right away, and you must force a cursor reparse to pick them up.

Note:

A logon trigger can be used because the user context (information such as `EMPNO`, `GROUP`, `MANAGER`) should be set before the user accesses any data.

Example: Creating a Simple Logon Trigger

The `CREATE TRIGGER` statement can create a simple logon trigger.

Example 10-2 shows a simple logon trigger that executes a PL/SQL procedure.

Example 10-2 Creating a Simple Logon Trigger

```
CREATE OR REPLACE TRIGGER set_empno_ctx_trig AFTER LOGON ON DATABASE
 BEGIN
  sec_mgr.set_empno_ctx_proc;
 END;
```

Example: Creating a Logon Trigger for a Production Environment

The `CREATE TRIGGER` statement can create a logon trigger for a production environment.

Example 10-3 shows how to create a logon trigger that uses a `WHEN OTHERS` exception. Otherwise, if there is an error in the PL/SQL logic that creates an unhandled exception, then all connections to the database are blocked.

This example shows a `WHEN OTHERS` exception that writes errors to a table in the security administrator's schema. In a production environment, this is safer than sending the output to the user session, where it could be vulnerable to security attacks.

Example 10-3 Creating a Logon Trigger for a Production Environment

```
CREATE OR REPLACE TRIGGER set_empno_ctx_trig AFTER LOGON ON DATABASE
 BEGIN
  sec_mgr.set_empno_ctx_proc;
 EXCEPTION
  WHEN OTHERS THEN
        v_code := SQLCODE;
        v_errm := SUBSTR(SQLERRM, 1 , 64);
       -- Invoke another procedure,
       -- declared with PRAGMA AUTONOMOUS_TRANSACTION,
       -- to insert information about errors.
  INSERT INTO sec_mgr.errors VALUES (v_code, v_errm, SYSTIMESTAMP);
 END;
/
```

Example: Creating a Logon Trigger for a Development Environment

The `CREATE TRIGGER` statement can create a logon trigger for a development environment.

Example 10-4 shows how to create the same logon trigger for a development environment, in which you may want to output errors the user session for debugging purposes.

Example 10-4 Creating a Logon Trigger for a Development Environment

```
CREATE TRIGGER set_empno_ctx_trig
 AFTER LOGON ON DATABASE
 BEGIN
  sysadmin_ctx.set_empno_ctx_pkg.set_empno;
 EXCEPTION
  WHEN OTHERS THEN
   RAISE_APPLICATION_ERROR(
    -20000, 'Trigger sysadmin_ctx.set_empno_ctx_trig violation. Login denied.');
 END;
/
```

Tutorial: Creating and Using a Database Session-Based Application Context

This tutorial demonstrates how to create an application context that checks the ID of users who try to log in to the database.

Step 1: Create User Accounts and Ensure the User SCOTT Is Active

To begin this tutorial, you must create the necessary database accounts and endure that the `SCOTT` user account is active.

1. Log on as user `SYS` and connect using the `SYSDBA` administrative privilege.

   ```
   sqlplus sys as sysdba
   Enter password: password
   ```

2. In a multitenant environment, connect to the appropriate PDB.

   For example:

   ```
   CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

3. Create the local user account `sysadmin_ctx`, who will administer the database session-based application context.

   ```
   CREATE USER sysadmin_ctx IDENTIFIED BY password;
   GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE, CREATE TRIGGER,
   ADMINISTER DATABASE TRIGGER TO sysadmin_ctx;
   GRANT READ ON HR.EMPLOYEES TO sysadmin_ctx;
   GRANT EXECUTE ON DBMS_SESSION TO sysadmin_ctx;
   ```

Follow the guidelines in Minimum Requirements for Passwords to replace *password* with a password that is secure.

4. Create the following user account for Lisa Ozer, who is listed as having `lozer` for her email account in the `HR.EMPLOYEES` table.

   ```
   GRANT CREATE SESSION TO LOZER IDENTIFIED BY password;
   ```

   Replace *password* with a password that is secure.

5. The sample user `SCOTT` will also be used in this tutorial, so query the `DBA_USERS` data dictionary view to ensure that the account status for `SCOTT` is `OPEN`.

   ```
   SELECT USERNAME, ACCOUNT_STATUS FROM DBA_USERS WHERE USERNAME = 'SCOTT';
   ```

   If the `DBA_USERS` view lists user `SCOTT` as locked and expired, then enter the following statement to unlock the `SCOTT` account and create a new password for him:

   ```
   ALTER USER SCOTT ACCOUNT UNLOCK IDENTIFIED BY password;
   ```

   Enter a password that is secure. For greater security, do **not** give the `SCOTT` account the same password from previous releases of Oracle Database. See Minimum Requirements for Passwords for the minimum requirements for creating passwords.

Step 2: Create the Database Session-Based Application Context

As the `sysadmin_ctx` user, you are ready to create the database session-based application context.

1. Log on to SQL*Plus as `sysadmin_ctx`.

   ```
   CONNECT sysadmin_ctx -- Or, CONNECT sysadmin_ctx@hrpdb
   Enter password: password
   ```

2. Create the application context using the following statement:

   ```
   CREATE CONTEXT empno_ctx USING set_empno_ctx_pkg;
   ```

   Remember that even though user `sysadmin_ctx` has created this application context, the `SYS` schema owns the context.

Step 3: Create a Package to Retrieve Session Data and Set the Application Context

Next, you must create a PL/SQL package that retrieves the session data and then sets the application context.

- To create the package, use the `CREATE OR REPLACE PACKAGE` statement.

Example 10-5 shows how to create the package you need to retrieve the session data and set the application context. Before creating the package, ensure that you are still logged on as user `sysadmin_ctx`. (You can copy and paste this text by positioning the cursor at the start of `CREATE OR REPLACE` in the first line.)

Example 10-5 Package to Retrieve Session Data and Set a Database Session Context

```
CREATE OR REPLACE PACKAGE set_empno_ctx_pkg IS
   PROCEDURE set_empno;
 END;
 /
 CREATE OR REPLACE PACKAGE BODY set_empno_ctx_pkg IS
   PROCEDURE set_empno
   IS
    emp_id HR.EMPLOYEES.EMPLOYEE_ID%TYPE;
   BEGIN
    SELECT EMPLOYEE_ID INTO emp_id FROM HR.EMPLOYEES
       WHERE email = SYS_CONTEXT('USERENV', 'SESSION_USER');
    DBMS_SESSION.SET_CONTEXT('empno_ctx', 'employee_id', emp_id);
   EXCEPTION
    WHEN NO_DATA_FOUND THEN NULL;
  END;
 END;
/
```

This package creates a procedure called `set_empno` that performs the following actions:

- `emp_id HR.EMPLOYEES.EMPLOYEE_ID%TYPE` declares a variable, `emp_id`, to store the employee ID for the user who logs on. It uses the same data type as the `EMPLOYEE_ID` column in `HR.EMPLOYEES`.
- `SELECT EMPLOYEE_ID INTO emp_id FROM HR.EMPLOYEES` performs a `SELECT` statement to copy the employee ID that is stored in the `employee_id` column data from the `HR.EMPLOYEES` table into the `emp_id` variable.
- `WHERE email = SYS_CONTEXT('USERENV', 'SESSION_USER')` uses a `WHERE` clause to find all employee IDs that match the email account for the session user. The `SYS_CONTEXT` function uses the predefined `USERENV` context to retrieve the user session ID, which is the same as the `email` column data. For example, the user ID and email address for Lisa Ozer are both the same: `lozer`.
- `DBMS_SESSION.SET_CONTEXT('empno_ctx', 'employee_id', emp_id)` uses the `DBMS_SESSION.SET_CONTEXT` procedure to set the application context:
  - `'empno_ctx'`: Calls the application context `empno_ctx`. Enclose `empno_ctx` in single quotes.
  - `'employee_id'`: Creates the attribute value of the `empno_ctx` application context name-value pair, by naming it `employee_id`. Enclose `employee_id` in single quotes.
  - `emp_id`: Sets the value for the `employee_id` attribute to the value stored in the `emp_id` variable.

  To summarize, the `set_empno_ctx_pkg.set_empno` procedure says, "Get the session ID of the user and then match it with the employee ID and email address of any user listed in the `HR.EMPLOYEES` table."
- `EXCEPTION ... WHEN_NO_DATA_FOUND` adds a `WHEN NO_DATA_FOUND` system exception to catch any `no data found` errors that may result from the `SELECT` statement. Without this exception, the package and logon trigger will work fine and set the application context as needed, but then any non-system administrator users other than the users listed in the `HR.EMPLOYEES` table will not be able to log in to the database. Other users should be able to log in to the database, assuming they are valid

database users. Once the application context information is set, then you can use this session information as a way to control user access to a particular application.

Step 4: Create a Logon Trigger for the Package

The logon trigger will execute when the user logs in.

- As user `sysadmin_ctx`, create a logon trigger for `set_empno_ctx_pkg.set_empno` package procedure.

```
CREATE TRIGGER set_empno_ctx_trig AFTER LOGON ON DATABASE
 BEGIN
  sysadmin_ctx.set_empno_ctx_pkg.set_empno;
 END;
/
```

Step 5: Test the Application Context

Now that the components are all in place, you are ready to test the application context.

1. Log on as user `lozer`.

   ```
   CONNECT lozer -- Or, CONNECT lozer@hrpdb
   Enter password: password
   ```

   When user `lozer` logs on, the `empno_ctx` application context collects her employee ID. You can check it as follows:

   ```
   SELECT SYS_CONTEXT('empno_ctx', 'employee_id') emp_id FROM DUAL;
   ```

   The following output should appear:

   ```
   EMP_ID
   --------------------------------------------------------
   168
   ```

2. Log on as user `SCOTT`.

   ```
   CONNECT SCOTT -- Or, CONNECT SCOTT@hrpdb
   Enter password: password
   ```

   User `SCOTT` is not listed as an employee in the `HR.EMPLOYEES` table, so the `empno_ctx` application context cannot collect an employee ID for him.

   ```
   SELECT SYS_CONTEXT('empno_ctx', 'employee_id') emp_id FROM DUAL;
   ```

   The following output should appear:

   ```
   EMP_ID
   --------------------------------------------------------
   ```

From here, the application can use the user session information to determine how much access the user can have in the database. You can use Oracle Virtual Private Database to accomplish this. .

Step 6: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. Connect as `SYS` with the `SYSDBA` administrative privilege.

```
CONNECT SYS AS SYSDBA -- Or, CONNECT SYS@hrpdb AS SYSDBA
Enter password: password
```

2. Drop the users `sysadmin_ctx` and `lozer`:

   ```
   DROP USER sysadmin_ctx CASCADE;
   DROP USER lozer;
   ```

3. Drop the application context.

   ```
   DROP CONTEXT empno_ctx;
   ```

   Remember that even though `sysadmin_ctx` created the application context, it is owned by the `SYS` schema.

4. If you want, lock and expire `SCOTT`, unless other users want to use this account:

   ```
   ALTER USER SCOTT PASSWORD EXPIRE ACCOUNT LOCK;
   ```

Initializing Database Session-Based Application Contexts Externally

Initializing database session-based application contexts externally increases performance because the application context is stored in the user global area (UGA).

About Initializing Database Session-Based Application Contexts Externally

You must use a special type of namespace to initialize session-based application context externally.

This namespace must accept the initialization of attribute values from external resources and then stores them in the local user session.

Initializing an application context externally enhances performance because it is stored in the UGA and enables the automatic propagation of attributes from one session to another. Connected user database links are supported only by application contexts initialized from OCI-based external sources.

Default Values from Users

Oracle Database enables you to capture and use default values from users for your applications.

Sometimes you need the default values from users. Initially, these default values may be hints or preferences, and then after validation, they become trusted contexts. Similarly, it may be more convenient for clients to initialize some default values, and then rely on a login event trigger or applications to validate the values.

For job queues, the job submission routine records the context being set at the time the job is submitted, and restores it when executing the batched job. To maintain the integrity of the context, job queues cannot bypass the designated PL/SQL package to set the context. Rather, the externally initialized application context accepts initialization of context values from the job queue process.

Automatic propagation of context to a remote session may create security problems. Developers or administrators can effectively handle the context that takes default values from resources other than the designated PL/SQL procedure by using logon triggers to reset the context when users log in.

Values from Other External Resources

An application context can accept the initialization of attributes and values through external resources.

Examples include an Oracle Call Interface (OCI) interface, a job queue process, or a database link.

Externally initialized application contexts provide the following features:

- For remote sessions, automatic propagation of context values that are in the externally initialized application context namespace
- For job queues, restoration of context values that are in the externally initialized application context namespace
- For OCI interfaces, a mechanism to initialize context values that are in the externally initialized application context namespace

Although any client program that is using Oracle Call Interface can initialize this type of namespace, you can use login event triggers to verify the values. It is up to the application to interpret and trust the values of the attributes.

Example: Creating an Externalized Database Session-based Application Context

The `CREATE CONTEXT` SQL statement can create an externalized database session-based application context.

Example 10-6 shows how to create a database session-based application context that obtains values from an external source.

Example 10-6 Creating an Externalized Database Session-based Application Context

```
CREATE CONTEXT ext_ctx USING ext_ctx_pkg INITIALIZED EXTERNALLY;
```

Initialization of Application Context Values from a Middle-Tier Server

Middle-tier servers can initialize application context values on behalf of database users.

In this process, context attributes are propagated for the remote session at initialization time, and the remote database accepts the values if the namespace is externally initialized.

For example, a three-tier application creating lightweight user sessions through OCI or JDBC/OCI can access the `PROXY_USER` attribute in `USERENV`. This attribute enables you to determine if the user session was created by a middle-tier application. You could allow a user to access data only for connections where the user is proxied. If users connect directly to the database, then they would not be able to access any data.

You can use the `PROXY_USER` attribute from the `USERENV` namespace within Oracle Virtual Private Database to ensure that users only access data through a particular middle-tier application. For a different approach, you can develop a secure application role to enforce your policy that users access the database only through a specific proxy.

See Also:

- Preserving User Identity in Multitiered Environments for information about proxy authentication and about using the `USERENV` attribute `CLIENT_IDENTIFIER` to preserve user identity across multiple tiers
- Middle Tier Server Use for Proxy Authentication for information about using a secure application role to enforce a policy through a specific proxy
- *Oracle Call Interface Programmer's Guide*

Initializing Database Session-Based Application Contexts Globally

When a database session-based application is stored in a centralized location, it can be used globally from an LDAP directory.

About Initializing Database Session-Based Application Contexts Globally

You can use a centralized location to store the database session-based application context of the user.

A centralized location enables applications to set up a user context during initialization based upon user identity.

In particular, this feature supports Oracle Label Security labels and privileges. Initializing an application context globally makes it easier to manage contexts for large numbers of users and databases.

For example, many organizations want to manage user information centrally, in an LDAP-based directory. Enterprise User Security supports centralized user and authorization management in Oracle Internet Directory. However, there may be additional attributes an application must retrieve from Lightweight Directory Access Protocol (LDAP) to use for Oracle Virtual Private Database enforcement, such as the user title, organization, or physical location. Initializing an application context globally enables you to retrieve these types of attributes.

Database Session-Based Application Contexts with LDAP

An application context that is initialized globally uses LDAP, a standard, extensible, and efficient directory access protocol.

The LDAP directory stores a list of users to which this application is assigned. Oracle Database uses a directory service, typically Oracle Internet Directory, to authenticate and authorize enterprise users.

Note:

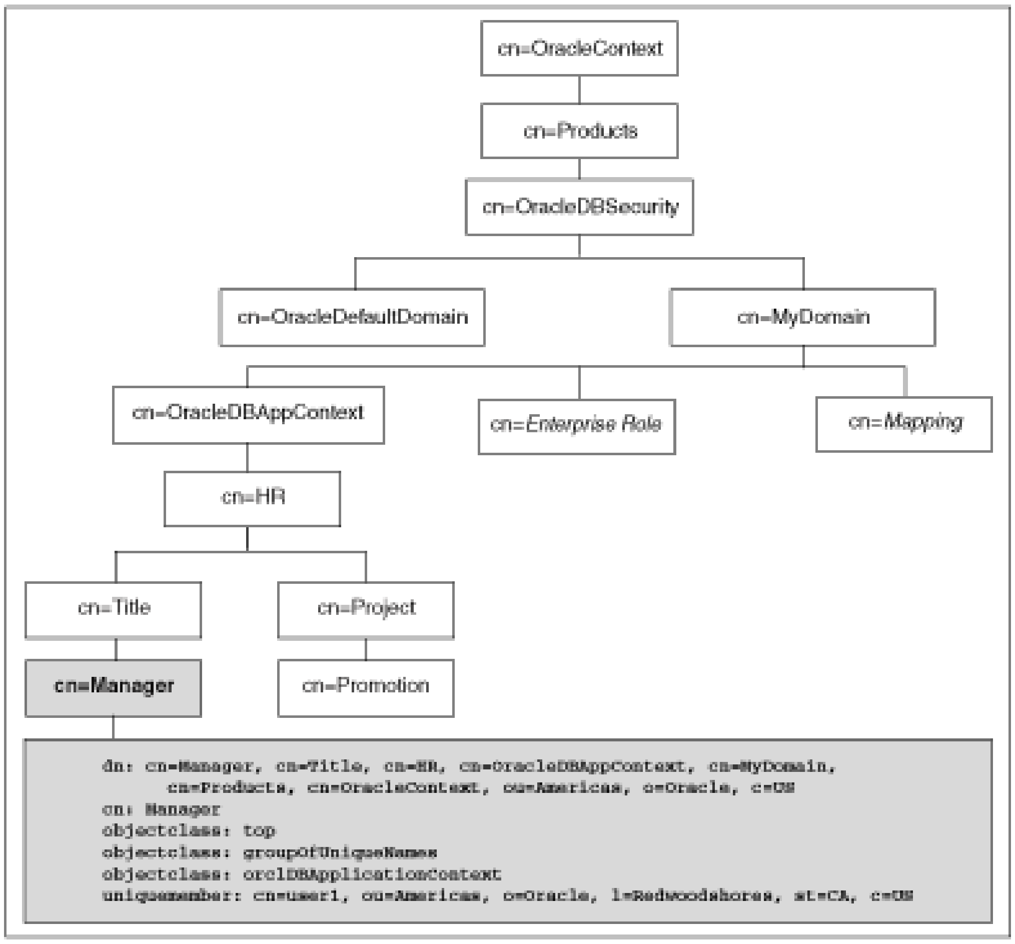
You can use third-party directories such as Microsoft Active Directory and Sun Microsystems SunONE as the directory service.

The `orclDBApplicationContext` LDAP object (a subclass of `groupOfUniqueNames`) stores the application context values in the directory. The location of the application context object is described in Figure 10-1, which is based on the Human Resources example.

The LDAP object `inetOrgPerson` enables multiple entries to exist for some attributes. However, be aware that when these entries are loaded into the database and accessed with the `SYS_LDAP_USER_DEFAULT` context namespace, only the first of these entries is returned. For example, the `inetOrgPerson` object for a user allows multiple entries for `telephoneNumber` (thus allowing a user to have multiple telephone numbers stored). When you use the `SYS_LDAP_USER_DEFAULT` context namespace, only the first telephone number is retrieved.

On the LDAP side, an internal C function is required to retrieve the `orclDBApplicationContext` value, which returns a list of application context values to the database. In this example, `HR` is the namespace; Title and Project are the attributes; and Manager and Promotion are the values.

Figure 10-1 Location of Application Context in LDAP Directory Information Tree



```
dn: cn=Manager, cn=Title, cn=HR, cn=OracleDBAppContext, cn=MyDomain,
      cn=Products, cn=OracleContext, ou=Americas, o=Oracle, c=US
cn: Manager
objectclass: top
objectclass: groupOfUniqueNames
objectclass: orclDBApplicationContext
uniquemember: cn=user1, ou=Americas, o=Oracle, l=Redwoodshores, st=CA, c=US
```

How Globally Initialized Database Session-Based Application Contexts Work

To use a globally initialized secure application, you must first configure Enterprise User Security.

Then, you configure the application context values for the user in the database and the directory.

When a global user (enterprise user) connects to the database, Enterprise User Security verifies the identity of the user connecting to the database. After authentication, the global user roles and application context are retrieved from the directory. When the user logs on to the database, the global roles and initial application context are already set.

See Also:

Oracle Database Enterprise User Security Administrator's Guide for information about configuring Enterprise User Security

Initializing a Database Session-Based Application Context Globally

You can configure and store the initial application context for a user, such as the department name and title, in the LDAP directory.

The values are retrieved during user login so that the context is set properly. In addition, any information related to the user is retrieved and stored in the `SYS_USER_DEFAULTS` application context namespace.

1. Create the application context in the database.

   ```
   CREATE CONTEXT hr USING hrapps.hr_manage_pkg INITIALIZED GLOBALLY;
   ```

2. Create and add new entries in the LDAP directory.

   An example of the entries added to the LDAP directory follows. These entries create an attribute named `Title` with the attribute value `Manager` for the application (namespace) `HR`, and assign user names `user1` and `user2`. In the following, `cn=example` refers to the name of the domain.

   ```
   dn:
   cn=OracleDBAppContext,cn=example,cn=OracleDBSecurity,cn=Products,cn=OracleContext
   ,ou=Americas,o=oracle,c=US
   changetype: add
   cn: OracleDBAppContext
   objectclass: top
   objectclass: orclContainer

   dn:
   cn=hr,cn=OracleDBAppContext,cn=example,cn=OracleDBSecurity,cn=Products,cn=OracleC
   ontext,ou=Americas,o=oracle,c=US
   changetype: add
   cn: hr
   objectclass: top
   objectclass: orclContainer

   dn: cn=Title,cn=hr,
   cn=OracleDBAppContext,cn=example,cn=OracleDBSecurity,cn=Products,cn=OracleContext
   ,ou=Americas,o=oracle,c=US
   changetype: add
   cn: Title
   objectclass: top
   objectclass: orclContainer
   ```

```
dn: cn=Manager,cn=Title,cn=hr,
cn=OracleDBAppContext,cn=example,cn=OracleDBSecurity,cn=Products,cn=OracleContext
,ou=Americas,o=oracle,c=US
cn: Manager
objectclass: top
objectclass: groupofuniquenames
objectclass: orclDBApplicationContext
uniquemember: CN=user1,OU=Americas,O=Oracle,L=Redwoodshores,ST=CA,C=US
uniquemember: CN=user2,OU=Americas,O=Oracle,L=Redwoodshores,ST=CA,C=US
```

3. If an LDAP `inetOrgPerson` object entry exists for the user, then the connection retrieves the attributes from `inetOrgPerson`, and assigns them to the namespace `SYS_LDAP_USER_DEFAULT`.

   The following is an example of an `inetOrgPerson` entry:

```
dn: cn=user1,ou=Americas,O=oracle,L=redwoodshores,ST=CA,C=US
changetype: add
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: user1
sn: One
givenName: User
initials: UO
title: manager, product development
uid: uone
mail: uone@us.example.com
telephoneNumber: +1 650 555 0105
employeeNumber: 00001
employeeType: full time
```

4. Connect to the database.

   When `user1` connects to a database that belongs to the `example` domain, `user1` will have his `Title` set to `Manager`. Any information related to `user1` will be retrieved from the LDAP directory. The value can be obtained using the following syntax:

```
SYS_CONTEXT('namespace','attribute name')
```

   For example:

```
DECLARE
 tmpstr1 VARCHAR2(30);
 tmpstr2 VARCHAR2(30);
BEGIN
 tmpstr1 = SYS_CONTEXT('HR','TITLE);
 tmpstr2 = SYS_CONTEXT('SYS_LDAP_USER_DEFAULT','telephoneNumber');
 DBMS_OUTPUT.PUT_LINE('Title is ' || tmpstr1);
 DBMS_OUTPUT.PUT_LINE('Telephone Number is ' || tmpstr2);
END;
```

   The output of this example is:

```
Title is Manager
Telephone Number is +1 650 555 0105
```

Externalized Database Session-Based Application Contexts

Many applications store attributes used for fine-grained access control within a database metadata table.

For example, an `employees` table could include cost center, title, signing authority, and other information useful for fine-grained access control. Organizations also centralize user information for user management and access control in LDAP-based directories, such as Oracle Internet Directory. Application context attributes can be stored in Oracle Internet Directory, and assigned to one or more enterprise users. They can also be retrieved automatically upon login for an enterprise user, and then used to initialize an application context.

See Also:

- Initializing Database Session-Based Application Contexts Externally for information about initializing local application context through external resources such as an OCI interface, a job queue process, or a database link
- Initializing Database Session-Based Application Contexts Globally for information about initializing local application context through a centralized resource, such as Oracle Internet Directory
- *Oracle Database Enterprise User Security Administrator's Guide* for information about enterprise users

Global Application Contexts

You can use a global application context to access application values across database sessions, including an Oracle Real Application Clusters environment.

About Global Application Contexts

A global application context enables application context values to be accessible across database sessions, including Oracle RAC instances.

Oracle Database stores the global application context information in the System (sometimes called "Shared") Global Area (SGA) so that it can be used for applications that use a sessionless model, such as middle-tier applications in a three-tiered architecture.

These applications cannot use a session-based application context because users authenticate to the application, and then it typically connects to the database as a single identity. Oracle Database initializes the global application context once, rather than for each user session. This improves performance, because connections are reused from a connection pool.

You can clear a global application context value by running the `ALTER SYSTEM FLUSH GLOBAL_CONTEXT` SQL statement.

Uses for Global Application Contexts

There are three general uses for global application contexts.

These uses are as follows:

- **You must share application values globally for all database users.** For example, you may need to disable access to an application based on a specific situation. In this case, the values the application context sets are not user-specific, nor are they based on the private data of a user. The application context defines a situation, for example, to indicate the version of application module that is running.
- **You have database users who must move from one application to another.** In this case, the second application the user is moving to has different access requirements from the first application.
- **You must authenticate nondatabase users, that is, users who are not known to the database.** This type of user, who does not have a database account, typically connects through a Web application by using a connection pool. These types of applications connect users to the database as single user, using the One Big Application User authentication model. To authenticate this type of user, you use the client session ID of the user.

Components of a Global Application Context

A global application context uses a package to manage its attributes and middle-tier application to manage the client session ID.

- **The global application context.** You use the `CREATE CONTEXT` SQL statement to create the global application context, and include the `ACCESSED GLOBALLY` clause in the statement. This statement names the application context and associates it with a PL/SQL procedure that is designed to set the application data context data. The global application context is created and stored in the database schema of the security administrator who creates it.
- **A PL/SQL package to set the attributes.** The package must contain a procedure that uses the `DBMS_SESSION.SET_CONTEXT` procedure to set the global application context. The `SET_CONTEXT` procedure provides parameters that enable you to create a global application context that fits any of the three user situations described in this section. You create, store, and run the PL/SQL package on the database server. Typically, it belongs in the schema of the security administrator who created it.
- **A middle-tier application to get and set the client session ID.** For nondatabase users, which require a client session ID to be authenticated, you can use the Oracle Call Interface (OCI) calls in the middle-tier application to retrieve and set their session data. You can also use the `DBMS_SESSION.SET_IDENTIFIER` procedure to set the client session ID. An advantage of creating a client session ID to store the nondatabase user's name is that you can query the `CLIENT_ID` column of `DBA_AUDIT_TRAIL`, `DBA_FGA_AUDIT_TRAIL`, and `DBA_COMMON_AUDIT_TRAIL` data dictionary views to audit this user's activity.

Note:

Be aware that the DBMS_APPLICATION_INFO.SET_CLIENT_INFO setting can overwrite the value.

Global Application Contexts in an Oracle Real Application Clusters Environment

In an Oracle RAC environment, whenever a global application context is loaded or changed, it is visible only to the existing active instances.

Be aware that setting a global application context value in an Oracle RAC environment has performance overhead of propagating the context value consistently to all Oracle RAC instances.

If you flush the global application context (using the ALTER SYSTEM FLUSH GLOBAL_CONTEXT SQL statement) in one Oracle RAC instance, then all the global application context is flushed in all other Oracle RAC instances as well.

Creating Global Application Contexts

The CREATE CONTEXT SQL statement creates the global application context, which is then located in the SYS schema.

Ownership of the Global Application Context

A global application context is owned by the SYS schema.

The ownership of the global application context is as follows: Even though a user who has been granted the CREATE ANY CONTEXT and DROP ANY CONTEXT privileges can create and drop the global application context, it is owned by the SYS schema.

Oracle Database associates the context with the schema account that created it, but if you drop this user, the context still exists in the SYS schema. As user SYS, you can drop the application context.

Creating a Global Application Context

As with local application contexts, the global application context is created and stored in the security administrator’s database schema.

You must have the CREATE ANY CONTEXT system privilege before you can create a global application context, and the DROP ANY CONTEXT privilege before you can drop the context with the DROP CONTEXT statement.

- To create a global application context, use the CREATE CONTEXT SQL statement to create the application context and include the ACCESSED GLOBALLY clause in the statement.

For example:

```
CREATE OR REPLACE CONTEXT global_hr_ctx USING hr_ctx_pkg ACCESSED GLOBALLY CONTAINER
= ALL;
```

PL/SQL Package to Manage a Global Application Context

The DBMS_SESSION PL/SQL package to manages global application contexts.

About the Package That Manages the Global Application Context

The package that is associated with a global application context uses the DBMS_SESSION package to set and clear the global application context values.

You must have the EXECUTE privilege for the DBMS_SESSION package before you use its procedures. Typically, you create and store this package in the database schema of a security administrator. The SYS schema owns the DBMS_SESSION package.

Unlike PL/SQL packages used to set a local application context, you do not include a SYS_CONTEXT function to get the user session data. You do not need to include this function because the owner of the session, recorded in the USERENV context, is the same for every user who is connecting.

You can run the procedures within the PL/SQL package for a global application context at any time. You do not need to create logon and logoff triggers to execute the package procedures associated with the global application context. A common practice is to run the package procedures from within the database application. Additionally, for nondatabase users, you use middle-tier applications to get and set client session IDs.

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the DBMS_SESSION package

How Editions Affects the Results of a Global Application Context PL/SQL Package

Global application context packages, Oracle Virtual Private Database packages, and fine-grained audit policies can be used across multiple editions.

Follow these guidelines:

- **If you want to have the PL/SQL package results be the same across all editions.** To do so, create the package in the schema of a user who has not been editions enabled. To find users who are not editions enabled, you can query the DBA_USERS and USER_USERS data dictionary views. Remember that SYS, SYSTEM, and other default Oracle Database administrative accounts that are listed in the DBA_REGISTRY data dictionary view are not and cannot be editions enabled.
- **If you want to have the PL/SQL package results depend on the current state of the edition in which the package is run.** Here, the results may be different across all editions to which the package applies. In this case, create the package in the schema of a user who has been editions enabled. If the schema is editions enabled, then it is likely that there will be different actual copies of the package in different editions, where each copy has different behavior. This is useful for the following types of scenarios:

- The package must use a new application context.
- The package must encode input values using a different scheme.
- The package must apply different validation rules for users logging in to the database.

For PL/SQL packages that set a global application context, use a single getter function to wrap the primitive `SYS_CONTEXT` calls that will read the key-value application context pairs. You can put this getter function in the same package as the application context setter procedure. This approach lets you tag the value for the application context key to reflect a relevant concept. For example, the tag can be the edition in which the setter function is actual. Or, it can be the current edition of the session that set the context, which you can find by using `SYS_CONTEXT('USERENV', 'CURRENT_EDITION_NAME')`. This tag can be any specific notion to which the setter function applies.

See Also:

Oracle Database Development Guide for detailed information about editions

DBMS_SESSION.SET_CONTEXT username and client_id Parameters

The `DBMS_SESSION.SYS_CONTEXT` procedure provides the `client_id` and `username` parameters, to be used for global application contexts.

Table 10-2 explains how the combination of these settings controls the type of global application context you can create.

Table 10-2 Setting the DBMS_SESSION.SET_CONTEXT username and client_id Parameters

Combination Settings	Result

`username` set to `NULL`
`client_id` set to `NULL`
	This combination enables all users to access the application context. See Sharing Global Application Context Values for All Database Users for more information.
	These settings are also used for database session-based application contexts. See Using Database Session-Based Application Contexts for more information.

`username` set to a value `client_id` set to `NULL`	This combination enables an application context to be accessed by multiple sessions, as long as the `username` setting is the same throughout. Ensure that the user name specified is a valid database user. See Global Contexts for Database Users Who Move Between Applications for more information.
`username` set to `NULL` `client_id` set to a value	This combination enables an application to be accessed by multiple user sessions, as long as the `client_id` parameter is set to the same value throughout. This enables sessions of all users to see the application context values.

Table 10-2 (Cont.) Setting the DBMS_SESSION.SET_CONTEXT username and client_id Parameters

Combination Settings	Result
`username` set to a value `client_id` set to a value	This combination enables the following two scenarios: • **Lightweight users.** If the user does not have a database account, the username specified is a connection pool owner. The `client_id` setting is then associated with the nondatabase user who is logging in. • **Database users.** If the user is a database user, this combination can be used for stateless Web sessions. Setting the `username` parameter in the `SET_CONTEXT` procedure to `USER` calls the Oracle Database-supplied `USER` function. The `USER` function specifies the session owner from the application context retrieval process and ensures that only the user who set the application context can access the context. See *Oracle Database SQL Language Reference* for more information about the `USER` function. See Global Application Context for Nondatabase Users for more information.

Sharing Global Application Context Values for All Database Users

You can share global application values for all database users to give them access to data in the database.

- To share global application values for all database users, set the `namespace`, `attribute`, and `value` parameters in the `SET_CONTEXT` procedure.

Example: Package to Manage Global Application Values for All Database Users

The `CREATE PACKAGE` statement can manage global application values for all database users.

Example 10-7 shows how to create a package that sets and clears a global application context for all database users.

Example 10-7 Package to Manage Global Application Values for All Database Users

```
CREATE OR REPLACE PACKAGE hr_ctx_pkg
   AS
    PROCEDURE set_hr_ctx(sec_level IN VARCHAR2);
    PROCEDURE clear_hr_context;
   END;
  /
  CREATE OR REPLACE PACKAGE BODY hr_ctx_pkg
   AS
    PROCEDURE set_hr_ctx(sec_level IN VARCHAR2)
    AS
    BEGIN
     DBMS_SESSION.SET_CONTEXT(
      namespace  => 'global_hr_ctx',
      attribute  => 'job_role',
      value      => sec_level);
     END set_hr_ctx;
  PROCEDURE clear_hr_context
    AS
    BEGIN
     DBMS_SESSION.CLEAR_CONTEXT('global_hr_ctx', 'job_role');
    END clear_context;
  END;
 /
```

In this example:

- `DBMS_SESSION.SET_CONTEXT ... END set_hr_ctx` uses the `DBMS_SESSION.SET_CONTEXT` procedure to set values for the `namespace`, `attribute`, and `value` parameters. The `sec_level` value is specified when the database application runs the `hr_ctx_pkg.set_hr_ctx` procedure.

  The `username` and `client_id` values are not set, hence, they are `NULL`. This enables all users (database users) to have access to the values, which is appropriate for server-wide settings.
- `namespace => 'global_hr_ctx'` sets the `namespace` to `global_hr_ctx`, in the `SET_CONTEXT` procedure.
- `attribute => 'job_role'` creates the `job_role` attribute.
- `value => sec_level` sets the value for the `job_role` attribute to `sec_level`.
- `PROCEDURE clear_hr_context` creates the `clear_hr_context` procedure to clear the context values. See Clearing Session Data When the Session Closes for more information.

Typically, you execute this procedure within a database application. For example, if all users logging in are clerks, and you want to use "clerk" as a security level, you would embed a call within a database application similar to the following:

```
BEGIN
 hr_ctx_pkg.set_hr_ctx('clerk');
END;
/
```

If the procedure successfully completes, then you can check the application context values as follows:

```
SELECT SYS_CONTEXT('global_hr_ctx', 'job_role') job_role FROM DUAL;

JOB_ROLE
-----------
clerk
```

You can clear the global application context values for all database users by running the following procedure:

```
BEGIN
 hr_ctx_pkg.clear_hr_context;
END;
/
```

To check that the global context value is really cleared, the following `SELECT` statement should return no values:

```
SELECT SYS_CONTEXT('global_hr_ctx', 'job_role') job_role FROM DUAL;
```

```
JOB_ROLE
-----------
```

If Oracle Database returns error messages saying that you have insufficient privileges, then ensure that you have correctly created the global application context. You should also query the `DBA_CONTEXT` database view to ensure that your settings are correct, for example, that you are calling the procedure from the schema in which you created it.

If `NULL` is returned, then you may have inadvertently set a client identifier. To clear the client identifier, run the following procedure:

```
EXEC DBMS_SESSION.CLEAR_IDENTIFIER;
```

Global Contexts for Database Users Who Move Between Applications

A global application context can be used for database users who move between application, even when the applications have different access requirements.

To do so, you must include the `username` parameter in the `DBMS_SESSION.SET_CONTEXT` procedure.

This parameter specifies that the same schema be used for all sessions.

You can use the following `DBMS_SESSION.SET_CONTEXT` parameters:

- `namespace`
- `attribute`
- `value`
- `username`

Oracle Database matches the `username` value so that the other application can recognize the application context. This enables the user to move between applications.

By omitting the `client_id` setting, its value is `NULL`, the default. This means that values can be seen by multiple sessions if the `username` setting is the same for a database user who maintains the same context in different applications. For example, you can have a suite of applications that control user access with Oracle Virtual Private Database policies, with each user restricted to a job role.

Example 10-8 demonstrates how to set the `username` parameter so that a specific user can move between applications. The use of the `username` parameter is indicated in **bold** typeface.

Example 10-8 Package for Global Application Context Values for Moving Between Applications

```
CREATE OR REPLACE PACKAGE hr_ctx_pkg
  AS
    PROCEDURE set_hr_ctx(sec_level IN VARCHAR2, user_name IN VARCHAR2);
    PROCEDURE clear_hr_context;
   END;
  /
  CREATE OR REPLACE PACKAGE BODY hr_ctx_pkg
   AS
    PROCEDURE set_hr_ctx(sec_level IN VARCHAR2, user_name IN VARCHAR2)
    AS
     BEGIN
      DBMS_SESSION.SET_CONTEXT(
       namespace  => 'global_hr_ctx',
       attribute  => 'job_role',
       value      => sec_level,
       username   => user_name);
      END set_hr_ctx;

   PROCEDURE clear_hr_context
    AS
     BEGIN
      DBMS_SESSION.CLEAR_CONTEXT('global_hr_ctx');
     END clear_context;
  END;
 /
```

Typically, you execute this procedure within a database application by embedding a call similar to the following example. Ensure that the value for the `user_name` parameter (`scott` in this case) is a valid database user name.

```
BEGIN
 hr_ctx_pkg.set_hr_ctx('clerk', 'scott');
END;
```

A secure way to manage this type of global application context is within your applications, embed code to grant a secure application role to the user. This code should include `EXECUTE` permissions on the trusted PL/SQL package that sets the application context. In other words, the application, not the user, will set the context for the user.

Global Application Context for Nondatabase Users

When a nondatabase user starts a client session, the application server generates a client session ID.

A nondatabase user is a user who is not known to the database, such as a Web application user.

Once this ID is set on the application server, it must be passed to the database server side. You can do this by using the `DBMS_SESSION.SET_IDENTIFIER` procedure to set the client session ID.

To set the context, you can set the `client_id` parameter in the `DBMS_SESSION.SET_CONTEXT` procedure, in a PL/SQL procedure on the server side. This enables you to manage the application context globally, yet each client sees only his or her assigned application context.

The `client_id` value is the key here to getting and setting the correct attributes for the global application context. Remember that the client identifier is controlled by the middle-tier application, and once set, it remains open until it is cleared.

A typical way to manage this type of application context is to place the `session_id` value (`client_identifier`) in a cookie, and send it to the end user's HTML page so that is returned on the next request. A lookup table in the application should also keep client identifiers so that they are prevented from being reused for other users and to implement an end-user session time out.

For nondatabase users, configure the following `SET_CONTEXT` parameters:

- `namespace`
- `attribute`

- `value`
- `username`
- `client_id`

Example: Package to Manage Global Application Context Values for Nondatabase Users

The `CREATE PACKAGE` statement can manage global application context values for nondatabase users.

Example 10-9 shows how to create a package that manages this type of global application context.

Example 10-9 Package to Manage Global Application Context Values for Nondatabase Users

```
CREATE OR REPLACE PACKAGE hr_ctx_pkg
  AS
   PROCEDURE set_session_id(session_id_p IN NUMBER);
   PROCEDURE set_hr_ctx(sec_level_attr IN VARCHAR2,
      sec_level_val IN VARCHAR2);
   PROCEDURE clear_hr_session(session_id_p IN NUMBER);
   PROCEDURE clear_hr_context;
  END;
/
  CREATE OR REPLACE PACKAGE BODY hr_ctx_pkg
   AS
    session_id_global NUMBER;
  PROCEDURE set_session_id(session_id_p IN NUMBER)
   AS
   BEGIN
    session_id_global := session_id_p;
    DBMS_SESSION.SET_IDENTIFIER(session_id_p);
  END set_session_id;

  PROCEDURE set_hr_ctx(sec_level_attr IN VARCHAR2,
     sec_level_val IN VARCHAR2)
   AS
   BEGIN
    DBMS_SESSION.SET_CONTEXT(
     namespace  => 'global_hr_ctx',
     attribute  => sec_level_attr,
     value      => sec_level_val,
     username   => USER,
     client_id  => session_id_global);
   END set_hr_ctx;

  PROCEDURE clear_hr_session(session_id_p IN NUMBER)
   AS
   BEGIN
      DBMS_SESSION.SET_IDENTIFIER(session_id_p);
      DBMS_SESSION.CLEAR_IDENTIFIER;
   END clear_hr_session;

  PROCEDURE clear_hr_context
  AS
  BEGIN
   DBMS_SESSION.CLEAR_CONTEXT('global_hr_ctx', session_id_global);
  END clear_hr_context;
 END;
 /
```

In this example:

- `session_id_global NUMBER` creates the `session_id_global` variable, which will hold the client session ID. The `session_id_global` variable is referenced throughout the package definition, including the procedure that creates the global application context attributes and assigns them values. This means that the global application context values will always be associated with this particular session ID.
- `PROCEDURE set_session_id ... END set_session_id` creates the `set_session_id` procedure, which writes the client session ID to the `session_id_global` variable.
- `PROCEDURE set_hr_ctx ... END set_hr_ctx` creates the `set_hr_ctx` procedure, which creates global application context attributes and enables you to assign values to these attributes. Within this procedure:
  - `username => USER` specifies the `username` value. This example sets it by calling the Oracle Database-supplied `USER` function, which adds the session owner from the context retrieval process. The `USER` function ensures that only the user who set the application context can access the context. See *Oracle Database SQL Language Reference* for more information about the `USER` function.

    If you had specified `NULL` (the default for the `username` parameter), then any user can access the context.

    Setting both the `username` and `client_id` values enables two scenarios. For lightweight users, set the `username` parameter to a connection pool owner (for example, `APPS_USER`), and then set `client_id` to the client session ID. If you want to use a stateless Web session, set the `user_name` parameter to the same database user who has logged in, and ensure that this user keeps the same client session ID. See DBMS_SESSION.SET_CONTEXT username and client_id Parameters for an explanation of how different `username` and `client_id` settings work.
  - `client_id => session_id_global` specifies `client_id` value. This example sets it to the `session_id_global` variable. This associates the context settings defined here with a specific client session ID, that is, the one that is set when you run the `set_session_id` procedure. If you specify the `client_id` parameter default, `NULL`, then the global application context settings could be used by any session.
- `PROCEDURE clear_hr_session ... END clear_hr_session` creates the `clear_hr_session` procedure to clear the client session identifier. The `AS` clause sets it to ensure that you are clearing the correct session ID, that is, the one stored in variable `session_id_p` defined in the `CREATE OR REPLACE PACKAGE BODY hr_ctx_pkg` procedure.
- `PROCEDURE clear_hr_context ... END clear_hr_context` creates the `clear_hr_context` procedure, so that you can clear the context settings for the current user session, which were defined by the `global_hr_ctx` variable. See Clearing Session Data When the Session Closes for more information.

Clearing Session Data When the Session Closes

The application context exists within memory, so when the user exits a session, you must clear the `client_identifier` context value.

This releases memory and prevents other users from accidentally using any left over values.

- To clear session data when a user exits a session, use either of the following methods in the server-side PL/SQL package:
  - **Clearing the client identifier when a user exits a session.** Use the `DBMS_SESSION.CLEAR_IDENTIFIER` procedure. For example:

    ```
    DBMS_SESSION.CLEAR_IDENTIFIER;
    ```

  - **Continuing the session but still clearing the context.** If you want the session to continue, but you still need to clear the context, use the `DBMS_SESSION.CLEAR_CONTEXT` or the `DBMS_SESSION.CLEAR_ALL_CONTEXT` procedure. For example:

    ```
    DBMS_SESSION.CLEAR_CONTEXT('my_ctx', 'my_attribute');
    ```

    The `CLEAR_CONTEXT` procedure clears the context for the current user. To clear the context values for all users, for example, when you need to shut down the application server, use the `CLEAR_ALL_CONTEXT` procedure.

    Global application context values are available until they are cleared, so you should use `CLEAR_CONTEXT` or `CLEAR_ALL_CONTEXT` to ensure that other sessions do not have access to these values. Be aware that any changes in the context value are reflected immediately and subsequent calls to access the value through the `SYS_CONTEXT` function will return the most recent value.

Embedding Calls in Middle-Tier Applications to Manage the Client Session ID

You can embed calls in middle-tier applications to manage client session IDs.

About Managing Client Session IDs Using a Middle-Tier Application

The application server generates the client session ID.

From a middle-tier application, you can get, set, and clear the client session IDs. To do so, you can embed either Oracle Call Interface (OCI) calls or `DBMS_SESSION` PL/SQL package procedures into the middle-tier application code.

The application authenticates the user, sets the client identifier, and sets it in the current session. The PL/SQL package `SET_CONTEXT` sets the `client_identifier` value in the application context.

Step 1: Retrieve the Client Session ID Using a Middle-Tier Application

When a user starts a client session, the application server generates a client session ID.

You can retrieve this ID for use in authenticating the user's access.

- To retrieve this client ID, use the `OCIStmtExecute` call with any of the following statements:
  - `SELECT SYS_CONTEXT('userenv', 'client_identifier') FROM DUAL;`
  - `SELECT CLIENT_IDENTIFIER from V$SESSION;`
  - `SELECT value FROM session_context WHERE attribute='CLIENT_IDENTIFIER';`

For example, to use the `OCIStmtExecute` call to retrieve a client session ID value:

```
oratext    clientid[31];
 OCIDefine  *defnp1 = (OCIDefine *) 0;
 OCIStmt    *statementhndle;
 oratext    *selcid = (oratext *)"SELECT SYS_CONTEXT('userenv',
            'client_identifier') FROM  DUAL";

OCIStmtPrepare(statementhndle, errhp, selcid,
  (ub4) strlen((char *) selcid), (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);

OCIDefineByPos(statementhndle, &defnp1, errhp, 1, (dvoid *)clientid, 31,
  SQLT_STR, (dvoid *) 0, (ub2 *) 0, (ub2 *) 0, OCI_DEFAULT);

OCIStmtExecute(servhndle, statementhndle, errhp, (ub4) 1, (ub4) 0,
 (CONST OCISnapshot *) NULL, (OCISnapshot *) NULL, OCI_DEFAULT);

  printf("CLIENT_IDENTIFIER = %s \n", clientid);
```

In this example:

- `oratext`, `OCIDefine`, `OCIStmt`, and `oratext` create variables to store the client session ID, reference call for `OCIDefine`, the statement handle, and the `SELECT` statement to use.
- `OCIStmtPrepar` prepares the statement `selcid` for execution.
- `OCIDefineByPos` defines the output variable `clientid` for client session ID.
- `OCIStmtExecute` executes the statement in the `selcid` variable.
- `printf` prints the formatted output for the retrieved client session ID.

Step 2: Set the Client Session ID Using a Middle-Tier Application

Next, you are ready to set the client session ID using a middle-tier application.

About Setting the Client Session ID Using a Middle-Tier Application

After you use the `OCIStmtExecute` call to retrieve the client session ID, you are ready to set this ID.

The `DBMS_SESSION.SET_CONTEXT` procedure in the server-side PL/SQL package then sets this session ID and optionally, overwrites the application context values.

You must ensure that the middle-tier application code checks that the client session ID value (for example, the value written to `user_id` in the previous examples) matches the `client_id` setting defined in the server-side `DBMS_SESSION.SET_CONTEXT` procedure. The sequence of calls on the application server side should be as follows:

1. Get the current client session ID. The session should already have this ID, but it is safer to ensure that it truly has the correct value.

2. Clear the current client session ID. This prepares the application to service a request from a different end user.
3. Set the new client session ID or the client session ID that has been assigned to the end user. This ensures that the session is using a different set of global application context values.

Setting the Client Session ID Using a Middle-Tier Application

Oracle Call Interface or the `DBMS_SESSION` PL/SQL package can set the client session ID using a middle-tier application.

- Use either of the following methods to set the client session ID on the application server side:
  - **Oracle Call Interface.** Set the `OCI_ATTR_CLIENT_IDENTIFIER` attribute in an `OCIAttrSet` OCI call. This attribute sets the client identifier in the session handle to track the end user identity.

    The following example shows how to use `OCIAttrSet` with the `ATTR_CLIENT_IDENTIFIER` parameter. The `user_id` setting refers to a variable that stores the ID of the user who is logging on.

    ```
    OCIAttrSet((void *)session_handle, (ub4) OCI_HTYPE_SESSION,
               (void *) user_id, (ub4)strlen(user_id),
               OCI_ATTR_CLIENT_IDENTIFIER, error_handle);
    ```
  - **DBMS_SESSION package.** Use the `DBMS_SESSION.SET_IDENTIFIER` procedure to set the client identifier for the global application context. For example, assuming you are storing the ID of the user logging on in a variable called `user_id`, you would enter the following line into the middle-tier application code:

    ```
    DBMS_SESSION.SET_IDENTIFIER(user_id);
    ```

Note:

When the application generates a session ID for use as a `CLIENT_IDENTIFIER`, then the session ID must be suitably random and protected over the network by encryption. If the session ID is not random, then a malicious user could guess the session ID and access the data of another user. If the session ID is not encrypted over the network, then a malicious user could retrieve the session ID and access the connection.

You can encrypt the session ID by using network data encryption. See Configuring Oracle Database Network Encryption and Data Integrity for more information. To learn more about encrypting data over a network, see *Oracle Database 2 Day + Security Guide*.

Checking the Value of the Client Identifier

For both `OCIAttrSet` and `DBMS_SESSION.SET_IDENTIFIER`, you can check the value of the client identifier.

- To check the value of the client identifier, use one of the of the following approaches:

  - To check it using the `SYS_CONTEXT` function:

    ```
    SELECT SYS_CONTEXT('userenv', 'client_identifier') FROM DUAL;
    ```

  - To check it by querying the `V$SESSION` view:

    ```
    SELECT CLIENT_IDENTIFIER from V$SESSION;
    ```

Step 3: Clear the Session Data Using a Middle-Tier Application

The application context exists entirely within memory.

When the user exits a session, you must clear the context for the `client_identifier` value. This releases memory and prevents other users from accidentally using any left over values

- To clear session data when a user exits a session, use either of the following methods in the middle-tier application code:
  - **Clearing the client identifier when a user exits a session.** Use the `DBMS_SESSION.CLEAR_IDENTIFIER` procedure. For example:

    ```
    DBMS_SESSION.CLEAR_IDENTIFIER;
    ```

  - **Continuing the session but still clearing the context.** If you want the session to continue, but you still need to clear the context, use the `DBMS_SESSION.CLEAR_CONTEXT` or the `DBMS_SESSION.CLEAR_ALL_CONTEXT` procedure. For example:

    ```
    DBMS_SESSION.CLEAR_CONTEXT(namespace, client_identifier, attribute);
    ```

    The `CLEAR_CONTEXT` procedure clears the context for the current user. To clear the context values for all users, for example, when you need to shut down the application server, use the `CLEAR_ALL_CONTEXT` procedure.

    Global application context values are available until they are cleared, so you should use `CLEAR_CONTEXT` or `CLEAR_ALL_CONTEXT` to ensure that other sessions do not have access to these values.

Tutorial: Creating a Global Application Context That Uses a Client Session ID

This tutorial demonstrates how you can create a global application context that uses a client session ID.

About This Tutorial

This tutorial shows how to create a global application context that uses a client session ID for a lightweight user application.

It demonstrates how to control nondatabase user access by using a connection pool. If you are using a multitenant environment, then this tutorial applies to the current PDB only.

Step 1: Create User Accounts

A security administrator will manage the application context and its package, and a user account will own the connection pool.

1. Log on to SQL*Plus as `SYS` with the `SYSDBA` administrative privilege.

   ```
   sqlplus sys as sysdba
   Enter password: password
   ```

2. In a multitenant environment, connect to the appropriate PDB.

   For example:

   ```
   CONNECT SYS@my_pdb AS SYSDBA
   Enter password: password
   ```

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.

3. Create the local user account `sysadmin_ctx`, who will administer the global application context.

   ```
   CREATE USER sysadmin_ctx IDENTIFIED BY password CONTAINER = CURRENT;
   GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE TO sysadmin_ctx;

   GRANT EXECUTE ON DBMS_SESSION TO sysadmin_ctx;
   ```

   Follow the guidelines in Minimum Requirements for Passwords to replace *`password`* with a password that is secure.

4. Create the local database account `apps_user`, who will own the connection pool.

   ```
   CREATE USER apps_user IDENTIFIED BY password CONTAINER = CURRENT;
   GRANT CREATE SESSION TO apps_user;
   ```

   Replace *`password`* with a password that is secure.

Step 2: Create the Global Application Context

Next, you are ready to create the global application context.

1. Log on as the security administrator `sysadmin_ctx`.

   ```
   CONNECT sysadmin_ctx -- Or, CONNECT sysadmin_ctx@hrpdb
   Enter password: password
   ```

2. Create the `cust_ctx` global application context.

   ```
   CREATE CONTEXT global_cust_ctx USING cust_ctx_pkg ACCESSED GLOBALLY;
   ```

   The `cust_ctx` context is created and associated with the schema of the security administrator `sysadmin_ctx`. However, the `SYS` schema owns the application context.

Step 3: Create a Package for the Global Application Context

The PL/SQL package will manage the global application context that you created.

1. As `sysadmin_ctx`, create the following PL/SQL package:

   ```
   CREATE OR REPLACE PACKAGE cust_ctx_pkg
     AS
      PROCEDURE set_session_id(session_id_p IN NUMBER);
      PROCEDURE set_cust_ctx(sec_level_attr IN VARCHAR2,
        sec_level_val IN VARCHAR2);
      PROCEDURE clear_hr_session(session_id_p IN NUMBER);
      PROCEDURE clear_hr_context;
   ```

```
  END;
 /
CREATE OR REPLACE PACKAGE BODY cust_ctx_pkg
  AS
  session_id_global NUMBER;

 PROCEDURE set_session_id(session_id_p IN NUMBER)
  AS
  BEGIN
   session_id_global := session_id_p;
   DBMS_SESSION.SET_IDENTIFIER(session_id_p);
 END set_session_id;

 PROCEDURE set_cust_ctx(sec_level_attr IN VARCHAR2, sec_level_val IN VARCHAR2)
  AS
  BEGIN
   DBMS_SESSION.SET_CONTEXT(
    namespace  => 'global_cust_ctx',
    attribute  => sec_level_attr,
    value      => sec_level_val,
    username   => USER, -- Retrieves the session user, in this case, apps_user
    client_id  => session_id_global);
  END set_cust_ctx;

  PROCEDURE clear_hr_session(session_id_p IN NUMBER)
   AS
   BEGIN
     DBMS_SESSION.SET_IDENTIFIER(session_id_p);
     DBMS_SESSION.CLEAR_IDENTIFIER;
   END clear_hr_session;

 PROCEDURE clear_hr_context
  AS
  BEGIN
   DBMS_SESSION.CLEAR_CONTEXT('global_cust_ctx', session_id_global);
  END clear_hr_context;
 END;
/
```

For a detailed explanation of how this type of package works, see Example 10-9.

2. Grant `EXECUTE` privileges on the `cust_ctx_pkg` package to the connection pool owner, `apps_user`.

```
GRANT EXECUTE ON cust_ctx_pkg TO apps_user;
```

Step 4: Test the Newly Created Global Application Context

At this stage, you are ready to explore how this global application context and session ID settings work.

1. Log on to SQL*Plus as the connection pool owner, user `apps_user`.

```
CONNECT apps_user -- Or, CONNECT apps_user@hrpdb
Enter password: password
```

2. When the connection pool user logs on, the application sets the client session identifier as follows:

```
BEGIN
 sysadmin_ctx.cust_ctx_pkg.set_session_id(34256);
```

```
END;
/
```

3. Test the value of the client session identifier.
   a. Set the session ID:

      ```
      EXEC sysadmin_ctx.cust_ctx_pkg.set_session_id(34256);
      ```

   b. Check the session ID:

      ```
      SELECT SYS_CONTEXT('userenv', 'client_identifier') FROM DUAL;
      ```

      The following output should appear:

      ```
      SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER')
      --------------------------------------------------
      34256
      ```

4. Set the global application context as follows:

   ```
   EXEC sysadmin_ctx.cust_ctx_pkg.set_cust_ctx('Category', 'Gold Partner');
   EXEC sysadmin_ctx.cust_ctx_pkg.set_cust_ctx('Benefit Level', 'Highest');
   ```

   (In a real-world scenario, the middle-tier application would set the global application context values, similar to how the client session identifier was set in Step 2.)

5. Enter the following `SELECT SYS_CONTEXT` statement to check that the settings were successful:

   ```
   col category format a13
   col benefit_level format a14

   SELECT SYS_CONTEXT('global_cust_ctx', 'Category') category,
   SYS_CONTEXT('global_cust_ctx', 'Benefit Level') benefit_level FROM DUAL;
   ```

   The following output should appear:

   ```
   CATEGORY       BENEFIT_LEVEL
   -------------  --------------
   Gold Partner   Highest
   ```

What `apps_user` has done here, within the client session 34256, is set a global application context on behalf of a nondatabase user. This context sets the `Category` and `Benefit Level DBMS_SESSION.SET_CONTEXT attributes` to be `Gold Partner` and `Highest`, respectively. The context exists only for user `apps_user` with client ID 34256. When a nondatabase user logs in, behind the scenes, he or she is really logging on as the connection pool user `apps_user`. Hence, the `Gold Partner` and `Highest` context values are available to the nondatabase user.

Suppose the user had been a database user and could log in without using the intended application. (For example, the user logs in using SQL*Plus.) Because the user has not logged in through the connection pool user `apps_user`, the global application context appears empty to our errant user. This is because the context was created and set under the `apps_user` session. If the user runs the `SELECT SYS_CONTEXT` statement, then the following output appears:

```
CATEGORY       BENEFIT_LEVEL
-------------  --------------
```

Step 5: Modify the Session ID and Test the Global Application Context Again

Next, clear and then modify the session ID and test the global application context again.

1. As user `apps_user`, clear the session ID.

   ```
   EXEC sysadmin_ctx.cust_ctx_pkg.clear_hr_session(34256);
   ```

2. Check the global application context settings again.

   ```
   SELECT SYS_CONTEXT('global_cust_ctx', 'Category') category,
   SYS_CONTEXT('global_cust_ctx', 'Benefit Level') benefit_level FROM DUAL;

   CATEGORY       BENEFIT_LEVEL
   -------------  --------------
   ```

   Because `apps_user` has cleared the session ID, the global application context settings are no longer available.

3. Restore the session ID to 34256, and then check the context values.

   ```
   EXEC sysadmin_ctx.cust_ctx_pkg.set_session_id(34256);

   SELECT SYS_CONTEXT('global_cust_ctx', 'Category') category,
   SYS_CONTEXT('global_cust_ctx', 'Benefit Level') benefit_level FROM DUAL;
   ```

   The following output should appear:

   ```
   CATEGORY       BENEFIT_LEVEL
   -------------  --------------
   Gold Partner   Highest
   ```

   As you can see, resetting the session ID to 34256 brings the application context values back again. To summarize, the global application context must be set only *once* for this user, but the client session ID must be set *each time* the user logs on.

4. Now try clearing and then checking the global application context values.

   ```
   EXEC sysadmin_ctx.cust_ctx_pkg.clear_hr_context;

   SELECT SYS_CONTEXT('global_cust_ctx', 'Category') category,
   SYS_CONTEXT('global_cust_ctx', 'Benefit Level') benefit_level FROM DUAL;
   ```

   The following output should appear:

   ```
   CATEGORY       BENEFIT_LEVEL
   -------------  --------------
   ```

   At this stage, the client session ID, 34256 is still in place, but the application context settings no longer exist. This enables you to continue the session for this user but without using the previously set application context values.

Step 6: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. Connect as `SYS` with the `SYSDBA` administrative privilege.

   ```
   CONNECT SYS AS SYSDBA -- Or, CONNECT SYS@mypdb AS SYSDBA
   Enter password: password
   ```

2. Drop the global application context.

   ```
   DROP CONTEXT global_cust_ctx;
   ```

   Remember that even though `sysadmin_ctx` created the global application context, it is owned by the `SYS` schema.

3. Drop the two sample users.

   ```
   DROP USER sysadmin_ctx CASCADE;
   DROP USER apps_user;
   ```

Global Application Context Processes

A simple global application context uses a database user account create the user session; a global application context is for lightweight users.

Simple Global Application Context Process

In a simple global application context process, the application uses a database user to create a user session.

The value for the context attribute of a simple global application context process can be retrieved from a `SELECT` statement.

Consider the application server, `AppSvr`, which has assigned the client identifier `12345` to client `SCOTT`. The `AppSvr` application uses the `SCOTT` user to create a session. (In other words, it is not a connection pool.) The value assigned to the context attribute can come from anywhere, for example, from running a `SELECT` statement on a table that holds the responsibility codes for users. When the application context is populated, it is stored in memory. As a result, any action that needs the responsibility code can access it quickly with a `SYS_CONTEXT` call, without the overhead of accessing a table. The only advantage of a global context over a local context in this case is if `SCOTT` were changing applications frequently and used the same context in each application.

The following steps show how the global application context process sets the client identifier for `SCOTT`:

1. The administrator creates a global context namespace by using the following statement:

   ```
   CREATE OR REPLACE CONTEXT hr_ctx USING hr.init ACCESSED GLOBALLY;
   ```

2. The administrator creates a PL/SQL package for the `hr_ctx` application context to indicate that, for this client identifier, there is an application context called `responsibility` with a value of `13` in the `HR` namespace.:

   ```
   CREATE OR REPLACE PROCEDURE hr.init
    AS
    BEGIN
     DBMS_SESSION.SET_CONTEXT(
       namespace => 'hr_ctx',
       attribute => 'responsibility',
       value     => '13',
       username  => 'SCOTT',
       client_id => '12345' );
    END;
   /
   ```

This PL/SQL procedure is stored in the HR database schema, but typically it is stored in the schema of the security administrator.

3. The AppSvr application issues the following command to indicate the connecting client identity each time scott uses AppSvr to connect to the database:

```
EXEC DBMS_SESSION.SET_IDENTIFIER('12345');
```

4. When there is a SYS_CONTEXT('hr_ctx','responsibility') call within the database session, the database matches the client identifier, 12345, to the global context, and then returns the value 13.

5. When exiting this database session, AppSvr clears the client identifier by issuing the following procedure:

```
EXEC DBMS_SESSION.CLEAR_IDENTIFIER( );
```

6. To release the memory used by the application context, AppSvr issues the following procedure:

```
DBMS_SESSION.CLEAR_CONTEXT('hr_ctx', '12345');
```

   CLEAR_CONTEXT is needed when the user session is no longer active, either on an explicit logout, timeout, or other conditions determined by the AppSvr application.

Note:

After a client identifier in a session is cleared, it becomes a NULL value. This implies that subsequent SYS_CONTEXT calls only retrieve application contexts with NULL client identifiers, until the client identifier is set again using the SET_IDENTIFIER interface.

Global Application Context Process for Lightweight Users

You can set a global application contexts for lightweight users.

You can configure this access so that when other users log in, they cannot access the global application context.

The following steps show the global application context process for a lightweight user application. The lightweight user, robert, is not known to the database through the application.

1. The administrator creates the global context namespace by using the following statement:

```
CREATE CONTEXT hr_ctx USING hr.init ACCESSED GLOBALLY;
```

2. The HR application server, AppSvr, starts and then establishes multiple connections to the HR database as the appsmgr user.
3. User robert logs in to the HR application server.
4. AppSvr authenticates robert to the application.
5. AppSvr assigns a temporary session ID (or uses the application user ID), 12345, for this connection.

6. The session ID is returned to the Web browser used by `robert` as part of a cookie or is maintained by `AppSvr`.
7. `AppSvr` initializes the application context for this client by calling the `hr.init` package, which issues the following statements:

   ```
   DBMS_SESSION.SET_CONTEXT( 'hr_ctx', 'id', 'robert', 'APPSMGR', 12345 );
   DBMS_SESSION.SET_CONTEXT( 'hr_ctx', 'dept', 'sales', 'APPSMGR', 12345 );
   ```
8. `AppSvr` assigns a database connection to this session and initializes the session by issuing the following statement:

   ```
   DBMS_SESSION.SET_IDENTIFIER( 12345 );
   ```
9. All `SYS_CONTEXT` calls within this database session return application context values that belong only to the client session.

   For example, `SYS_CONTEXT('hr','id')` returns the value `robert`.
10. When finished with the session, `AppSvr` issues the following statement to clean up the client identity:

    ```
    DBMS_SESSION.CLEAR_IDENTIFIER ( );
    ```

Even if another user logged in to the database, this user cannot access the global context set by `AppSvr`, because `AppSvr` specified that only the application with user `APPSMGR` logged in can see it. If `AppSvr` used the following, then any user session with client ID set to `12345` can see the global context:

```
DBMS_SESSION.SET_CONTEXT( 'hr_ctx', 'id', 'robert', NULL , 12345 );
DBMS_SESSION.SET_CONTEXT( 'hr_ctx', 'dept', 'sales', NULL , 12345 );
```

Setting `USERNAME` to `NULL` enables different users to share the same context.

Note:

Be aware of the security implication of different settings of the global context. `NULL` in the user name means that any user can access the global context. A `NULL` client ID in the global context means that a session with an uninitialized client ID can access the global context. To ensure that only the user who has logged on can access the session, specify `USER` instead of `NULL`.

You can query the client identifier set in the session as follows:

```
SELECT SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER') FROM DUAL;
```

The following output should appear:

```
SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER')
-------------------------------------------------
12345
```

A security administrator can see which sessions have the client identifier set by querying the `V$SESSION` view for the `CLIENT_IDENTIFIER` and `USERNAME`, for example:

```
COL client_identifier format a18
SELECT CLIENT_IDENTIFIER, USERNAME from V$SESSION;
```

The following output should appear:

```
CLIENT_IDENTIFIER   USERNAME
------------------  --------
12345               APPSMGR
```

To check the amount of global context area (in bytes) being used, use the following query:

```
SELECT SYS_CONTEXT('USERENV','GLOBAL_CONTEXT_MEMORY') FROM DUAL;
```

The following output should appear:

```
SYS_CONTEXT('USERENV','GLOBAL_CONTEXT_MEMORY')
----------------------------------------------
584
```

See Also:

For more information about using the `CLIENT_IDENTIFIER` predefined attribute of the `USERENV` application context:

- Use of the CLIENT_IDENTIFIER Attribute to Preserve User Identity
- *Oracle Database SQL Language Reference*
- *Oracle Call Interface Programmer's Guide*

Using Client Session-Based Application Contexts

A client session-based application context is stored in the User Global Area (UGA).

About Client Session-Based Application Contexts

Oracle Call Interface (OCI) functions can set and clear the User Global Area (UGA) user session information.

The advantage of this type of application context in a session-based application context is that an individual application can check for specific nondatabase user session data, rather than having the database perform this task. Another advantage is that the calls to set the application context value are included in the next call to the server, which improves performance.

However, be aware that application context security is compromised with a client session-based application context: any application user can set the client application context, and no check is performed in the database.

You configure the client session-based application context for the client application only. You do not configure any settings on the database server to which the client connects. Any application context settings in the database server do not affect the client session-based application context.

To configure a client session-based application context, use the `OCIAppCtxSet` OCI function. A client session-based application context uses the `CLIENTCONTEXT` namespace, updatable by any OCI client or by the existing `DBMS_SESSION` package for application context. Oracle Database performs no privilege or package security checks for this type.

The CLIENTCONTEXT namespace enables a single application transaction to both change the user context information and use the same user session handle to service the new user request. You can set or clear individual values for attributes in the CLIENTCONTEXT namespace, or clear all their values.

- An OCI client uses the OCIAppCtx function to set variable length data for the namespace, called OCISessionHandle. The OCI network single, round-trip transport sends all the information to the server in one round-trip. On the server side, you can query the application context information by using the SYS_CONTEXT SQL function on the namespace. For example:
- A JDBC client uses the oracle.jdbc.internal.OracleConnection function to achieve the same purposes.

Any user can set, clear, or collect the information in the CLIENTCONTEXT namespace, because it is not protected by package-based security.

See Also:

Oracle Call Interface Programmer's Guide for more information about client application contexts

Setting a Value in the CLIENTCONTEXT Namespace

Oracle Call Interface (OCI) can set the CLIENTCONTEXT namespace.

- To set a value in the CLIENTCONTEXT namespace, use the OCIAppCTXSet command, in the following syntax:

```
err = OCIAppCtxSet((void *) session_handle,(dvoid *)"CLIENTCONTEXT",(ub4) 13,
                   (dvoid *)attribute_name, length_of_attribute_name
                   (dvoid *)attribute_value, length_of_attribute_value, errhp,
                   OCI_DEFAULT);
```

In this specification:

- *session_handle* represents the OCISessionHandle namespace.
- *attribute_name* is the name of the attribute. For example, responsibility, with a length of 14.
- *attribute_value* is the value of the attribute. For example, manager, with a length of 7.

See Also:

Oracle Call Interface Programmer's Guide for details about the OCIAppCtx function

Retrieving the CLIENTCONTEXT Namespace

You can use Oracle Call Interface to retrieve the `CLIEINTCONTEXT` namespace.

- To retrieve the `CLIENTCONTEXT` namespace, use the `OCIStmtExecute` call with either of the following statements:
  - `SELECT SYS_CONTEXT('CLIENTCONTEXT', 'attribute-1') FROM DUAL;`
  - `SELECT VALUE FROM SESSION_CONTEXT WHERE NAMESPACE='CLIENTCONTEXT' AND ATTRIBUTE='attribute-1';`

The *`attribute-1`* value can be any attribute value that has already been set in the `CLIENTCONTEXT` namespace. Oracle Database only retrieves the set attribute; otherwise, it returns `NULL`. Typically, you set the attribute by using the `OCIAppCtxSet` call. In addition, you can embed a `DBMS_SESSION.SET_CONTEXT` call in the OCI code to set the attribute value.

Example: Retrieving a Client Session ID Value for Client Session-Based Contexts

The OCI `OCIStmtExecute` call can retrieve client session ID values for client session-based contexts.

Example 10-10 shows how to use the `OCIStmtExecute` call to retrieve a client session ID value.

Example 10-10 Retrieving a Client Session ID Value for Client Session-Based Contexts

```
oratext    clientid[31];
OCIDefine  *defnp1 = (OCIDefine *) 0;
OCIStmt    *statementhndle;
oratext    *selcid = (oratext *)"SELECT SYS_CONTEXT('CLIENTCONTEXT',
            attribute) FROM  DUAL";

OCIStmtPrepare(statementhndle, errhp, selcid, (ub4) strlen((char *) selcid),
  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);

OCIDefineByPos(statementhndle, &defnp1, errhp, 1, (dvoid *)clientid, 31,
  SQLT_STR, (dvoid *) 0, (ub2 *) 0, (ub2 *) 0, OCI_DEFAULT);

OCIStmtExecute(servhndle, statementhndle, errhp, (ub4) 1, (ub4) 0,
  (CONST OCISnapshot *) NULL, (OCISnapshot *) NULL, OCI_DEFAULT);

 printf("CLIENT_IDENTIFIER = %s \n", clientid);
```

In this example:

- `oratext`, `OCIDefine`, `OCIStmt`, and `oratext` create variables to store the client session ID, reference call for `OCIDefine`, the statement handle, and the `SELECT` statement to use.
- `OCIStmtPrepare` prepares the statement `selcid` for execution.
- `OCIDefineByPos` defines the output variable `clientid` for client session ID.
- `OCIStmtExecute` executes the statement in the `selcid` variable.

- `printf` prints the formatted output for the retrieved client session ID.

Clearing a Setting in the CLIENTCONTEXT Namespace

You can use Oracle Call Interface to clear the `CLIENTCONTEXT` namespace.

- To clear a setting in `CLIENTCONTEXT`, set the value to `NULL` or to an empty string by using one of the following commands:
  - The following command sets the empty string to zero:

    ```
    (void) OCIAppCtxSet((void *) session_handle, (dvoid *)"CLIENTCONTEXT", 13,
                        (dvoid *)attribute_name, length_of_attribute_name,
                        (dvoid *)0, 0,errhp
                        OCI_DEFAULT);
    ```
  - This following command sets the empty string to a blank value:

    ```
    (void) OCIAppCtxSet((void *) session_handle, (dvoid *)"CLIENTCONTEXT", 13
                        (dvoid *)attribute_name, length_of_attribute_name,
                        (dvoid *)"", 0,errhp,
                        OCI_DEFAULT);
    ```

Clearing All Settings in the CLIENTCONTEXT Namespace

You can use Oracle Call Interface (OCI) to clear the `CLIENTCONTEXT` namespace.

- To clear the namespace, use the `OCIAppCtxClearAll` command in the following form:

  ```
  err = OCIAppCtxClearAll((void *) session_handle,
                         (dvoid *)"CLIENTCONTEXT", 13,
                          errhp,                         OCI_DEFAULT);
  ```

Application Context Data Dictionary Views

Oracle Database provides data dictionary views that provide information about application contexts.

Table 10-3 lists these data dictionary views.

Table 10-3 Data Dictionary Views That Display Information about Application Contexts

View	Description
`ALL_CONTEXT`	Describes all context namespaces in the current session for which attributes and values were specified using the `DBMS_SESSION.SET_CONTEXT` procedure. It lists the namespace and its associated schema and PL/SQL package.
`ALL_POLICY_CONTEXTS`	Describes the driving contexts defined for the synonyms, tables, and views accessible to the current user. (A driving context is a context used in a Virtual Private Database policy.)
`DBA_CONTEXT`	Provides all context namespace information in the database. Its columns are the same as those in the `ALL_CONTEXT` view, except that it includes the `TYPE` column. The `TYPE` column describes how the application context is accessed or initialized.

Table 10-3 (Cont.) Data Dictionary Views That Display Information about Application Contexts

View	Description
`DBA_OBJECTS`	Provides the names of existing application contexts. Query the `OBJECT_TYPE` column of the `DBA_OBJECTS` view, as follows: `SELECT OBJECT_NAME FROM DBA_OBJECTS WHERE OBJECT_TYPE ='CONTEXT';`
`DBA_POLICY_CONTEXTS`	Describes all driving contexts in the database that were added by the `DBMS_RLS.ADD_POLICY_CONTEXT` procedure. Its columns are the same as those in `ALL_POLICY_CONTEXTS`.
`SESSION_CONTEXT`	Describes the context attributes and their values set for the current session.
`USER_POLICY_CONTEXTS`	Describes the driving contexts defined for the synonyms, tables, and views owned by the current user. Its columns (except for `OBJECT_OWNER`) are the same as those in `ALL_POLICY_CONTEXTS`.
`V$CONTEXT`	Lists set attributes in the current PDB session. Users do not have access to this view unless you grant the user the `SELECT` privilege on it.
`V$SESSION`	Lists detailed information about each current PDB session. Users do not have access to this view unless you grant the user the `SELECT` privilege on it.

Tip:

In addition to these views, check the database trace file if you find errors when running applications that use application contexts. The `USER_DUMP_DEST` initialization parameter sets the directory location of the trace files. You can find the value of this parameter by issuing `SHOW PARAMETER USER_DUMP_DEST` in SQL*Plus.

See Also:

- *Oracle Database SQL Tuning Guide* for more information about trace files
- *Oracle Database Reference* for detailed information about these views

11

Using Oracle Virtual Private Database to Control Data Access

Oracle Virtual Private Database (VPD) enables you to filter users who access data.

About Oracle Virtual Private Database

Oracle Virtual Private Database (VPD) provides important benefits for filtering user access to data.

What Is Oracle Virtual Private Database?

Oracle Virtual Private Database (VPD) creates security policies to control database access at the row and column level.

Note:

Oracle Database release 12c introduced Real Application Security (RAS) to supersede VPD. Oracle recommends that you use RAS for new projects that require row and column level access controls for their applications.

Essentially, Oracle Virtual Private Database adds a dynamic `WHERE` clause to a SQL statement that is issued against the table, view, or synonym to which an Oracle Virtual Private Database security policy was applied.

Oracle Virtual Private Database enforces security, to a fine level of granularity, directly on database tables, views, or synonyms. Because you attach security policies directly to these database objects, and the policies are automatically applied whenever a user accesses data, there is no way to bypass security.

When a user directly or indirectly accesses a table, view, or synonym that is protected with an Oracle Virtual Private Database policy, Oracle Database dynamically modifies the SQL statement of the user. This modification creates a `WHERE` condition (called a predicate) returned by a function implementing the security policy. Oracle Database modifies the statement dynamically, transparently to the user, using any condition that can be expressed in or returned by a function. You can apply Oracle Virtual Private Database policies to `SELECT`, `INSERT`, `UPDATE`, `INDEX`, and `DELETE` statements.

For example, suppose a user performs the following query:

```
SELECT * FROM OE.ORDERS;
```

The Oracle Virtual Private Database policy dynamically appends the statement with a `WHERE` clause. For example:

```
SELECT * FROM OE.ORDERS
 WHERE SALES_REP_ID = 159;
```

In this example, the user can only view orders by Sales Representative 159.

If you want to filter the user based on the session information of that user, such as the ID of the user, then you can create the `WHERE` clause to use an application context. For example:

```
SELECT * FROM OE.ORDERS
 WHERE SALES_REP_ID = SYS_CONTEXT('USERENV','SESSION_USER');
```

Note:

Oracle Virtual Private Database does not support filtering for DDLs, such as `TRUNCATE` or `ALTER TABLE` statements.

Benefits of Using Oracle Virtual Private Database Policies

Oracle Virtual Private Database policies provide the important benefits.

Security Policies Based on Database Objects Rather Than Applications

Oracle Virtual Private Database provides benefits in security, simplicity, and flexibility.

Attaching Oracle Virtual Private Database security policies to database tables, views, or synonyms, rather than implementing access controls in all your applications, provides the following benefits:

- **Security.** Associating a policy with a database table, view, or synonym can solve a potentially serious application security problem. Suppose a user is authorized to use an application, and then drawing on the privileges associated with that application, wrongfully modifies the database by using an ad hoc query tool, such as SQL*Plus. By attaching security policies directly to tables, views, or synonyms, fine-grained access control ensures that the same security is in force, no matter how a user accesses the data.
- **Simplicity.** You add the security policy to a table, view, or synonym only once, rather than repeatedly adding it to each of your table-based, view-based, or synonym-based applications.
- **Flexibility.** You can have one security policy for `SELECT` statements, another for `INSERT` statements, and still others for `UPDATE` and `DELETE` statements. For example, you might want to enable Human Resources clerks to have `SELECT` privileges for all employee records in their division, but to update only salaries for those employees in their division whose last names begin with `A` through `F`. Furthermore, you can create multiple policies for each table, view, or synonym.

Control Over How Oracle Database Evaluates Policy Functions

Running policy functions multiple times can affect performance.

You can control the performance of policy functions by configuring how Oracle Database caches the Oracle Virtual Private Database predicates.

The following options are available:

- Evaluate the policy once for each query (static policies).

- Evaluate the policy only when an application context within the policy function changes (context-sensitive policies).
- Evaluate the policy each time it is run (dynamic policies).

Who Can Create Oracle Virtual Private Database Policies?

The `DBMS_RLS` PL/SQL package enables you to create VPD policies.

Users who have been granted the `EXECUTE` privilege on the `DBMS_RLS` PL/SQL package can create Oracle Virtual Private Database policies. As with all privileges, only grant this privilege to trusted users. You can find the privileges that a user has been granted by querying the `DBA_SYS_PRIVS` data dictionary view.

Privileges to Run Oracle Virtual Private Database Policy Functions

You should be aware of the correct privileges for running Oracle Virtual Private Database (VPD) policy functions.

For greater security, the Oracle Virtual Private Database policy function runs as if it had been declared with definer's rights.

Do not declare it as invoker's rights because this can confuse yourself and other users who maintain the code.

See Also:

Oracle Database PL/SQL Language Reference for detailed information about definer's rights

Oracle Virtual Private Database Use with an Application Context

You can use application contexts with Oracle Virtual Private Database policies.

When you create an application context, it securely caches user information. Only the designated application package can set the cached environment. It cannot be changed by the user or outside the package. In addition, because the data is cached, performance is increased.

For example, suppose you want to base access to the `ORDERS_TAB` table on the customer ID number. Rather than querying the customer ID number for a logged-in user each time you need it, you could store the number in the application context. Then, the customer number is available in the session when you need it.

Application contexts are especially helpful if your security policy is based on multiple security attributes. For example, if a policy function bases a `WHERE` predicate on four attributes (such as employee number, cost center, position, spending limit), then multiple subqueries must execute to retrieve this information. Instead, if this data is available through an application context, then performance is much faster.

You can use an application context to return the correct security policy, enforced through a predicate. For example, consider an order entry application that enforces the following rules: customers only see their own orders, and clerks see all orders for all customers. These are two different policies. You could define an application context with a `position` attribute, and this attribute could be accessed within the policy function to return the correct predicate, depending on the value of the attribute. Thus, you can enable a user in the `clerk` position to retrieve all orders, but a user in the `customer` position can see only those records associated with that particular user.

To design a fine-grained access control policy that returns a specific predicate for an attribute, you need to access the application context within the function that implements the policy. For example, suppose you want to limit customers to seeing only their own records. The user performs the following query:

```
SELECT * FROM orders_tab
```

Fine-grained access control dynamically modifies this query to include the following `WHERE` predicate:

```
SELECT * FROM orders_tab
  WHERE custno = SYS_CONTEXT ('order_entry', 'cust_num');
```

Continuing with the preceding example, suppose you have 50,000 customers, and you do not want to have a different predicate returned for each customer. Customers all share the same `WHERE` predicate, which prescribes that they can only see their own orders. It is merely their customer numbers that are different.

Using application context, you can return one `WHERE` predicate within a policy function that applies to 50,000 customers. As a result, there is one shared cursor that executes differently for each customer, because the customer number is evaluated at execution time. This value is different for every customer. Use of application context in this case provides optimum performance, and at row-level security.

The `SYS_CONTEXT` function works much like a bind variable; only the `SYS_CONTEXT` arguments are constants.

Oracle Virtual Private Database in a Multitenant Environment

You can create Virtual Private Database policies in an application root for use throughout any associated application PDBs.

The CDB restriction applies to shared context sensitive policies and views related to Virtual Private Database policies as well. You cannot create a Virtual Private Database policy for an entire multitenant environment.

With regard to application containers, you can create Virtual Private Database policies to protect application common objects by applying the common policy to all PDBs that belong to the application root. In other words, when you install an application in the application root, all the common Virtual Private Database policies that protect the common objects will be applied to and immediately enforced for all PDBs in the application container.

Note the following:

- You can only create the common Virtual Private Database policy and its associated PL/SQL function in the application root and only attach it to application common objects. If the function is not in the same location as the policy, then an error is raised at runtime.
- A Virtual Private Database policy that is applied to common objects is considered a common policy that will be automatically enforced in PDBs that belong to the

application container when it accesses the application common objects from application PDBs.

- Application common Virtual Private Database policies can only protect application common objects.
- A Virtual Private Database policy that is applied to application common objects in the application root and is applied to all application PDBs is considered a common Virtual Private Database policy. A policy that is applied to a local database table and enforced in one PDB is considered a local Virtual Private Database policy.

  For example, if policy `VPD_P1` is applied to the application common table `T1` in the application root, then it is a considered to be a common policy. It will be enforced in each application PDB. If a policy named `VPD_P1` is applied to a local table called `T1` in `PDB1`, then it is considered a local policy, which means that it affects only `PDB1`. If a policy called `VPD_P1` is applied to a local table `T1` in the application root, then it is still considered a local policy because it affects only the application root. This concept applies to other operations, such as enabling, disabling, and removing Virtual Private Database policies.
- Application common Virtual Private Database policies only protect application common objects, while local Virtual Private Database policies only protect local objects.
- If you are using application contexts, then ensure common database session-based application contexts and common global application context objects are used in the common Virtual Private Database configuration.
- Application container Virtual Private Database policies are stored in the application root. PDBs store only local policies. If you plug a PDB into the application container, then the common policies are not converted to local policies. Instead, Oracle Database loads them from the application root and enforces them in the local PDB when the policies access common objects in the local PDB.

Components of an Oracle Virtual Private Database Policy

A VPD policy uses a function to generate the dynamic `WHERE` clause, and a policy to attach the function to objects to protect.

Function to Generate the Dynamic WHERE Clause

The Oracle Virtual Private Database (VPD) function defines the restrictions that you want to enforce.

To generate the Oracle Virtual Private Database (VPD) dynamic `WHERE` clause (predicate), you must create a function (not a procedure) that defines these restrictions.

Usually, the security administrator creates this function in his or her own schema. For more complex behavior, such as including calls to other functions or adding checks to track failed logon attempts, create these functions within a package.

The function must have the following behavior:

- **It must take as arguments a schema name and an object (table, view, or synonym) name as inputs.** Define input parameters to hold this information, but do not specify the schema and object name themselves within the function. The policy that you create with the `DBMS_RLS` package (described in Policies to Attach the Function to the Objects You Want to Protect) provides the names of the schema, and object to which the policy will apply. You must create the parameter for the schema first, followed by the parameter for the object.

- **It must provide a return value for the WHERE clause predicate that will be generated.** The return value for the `WHERE` clause is always a `VARCHAR2` data type.
- **It must generate a valid WHERE clause.** This code can be as basic as the example in Tutorial: Creating a Simple Oracle Virtual Private Database Policy, in that its `WHERE` clause is the same for all users who log on.

  But in most cases, you may want to design the `WHERE` clause to be different for each user, each group of users, or each application that accesses the objects you want to protect. For example, if a manager logs in, the `WHERE` clause can be specific to the rights of that particular manager. You can do this by incorporating an application context, which accesses user session information, into the `WHERE` clause generation code. Tutorial: Implementing a Session-Based Application Context Policy demonstrates how to create an Oracle Virtual Private Database policy that uses an application context.

  You can create Oracle Virtual Private Database functions that do not use an application context, but an application context creates a much stronger Oracle Virtual Private Database policy, by securely basing user access on the session attributes of that user, such as the user ID. Using Application Contexts to Retrieve User Information, discusses different types of application contexts in detail.

  In addition, you can embed C or Java calls to access operating system information or to return `WHERE` clauses from an operating system file or other source.
- **It must not select from a table within the associated policy function.** Although you can define a policy against a table, you cannot select that table from within the policy that was defined against the table.
- **It must be a pure function.** The VPD function must rely only on the application context and the arguments that are passed to the function to generate the `WHERE` clause. This function must not depend on the package variables.

Note:

If you plan to run the function across different editions, you can control the results of the function: whether the results are uniform across all editions, or specific to the edition in which the function is run. See How Editions Affects the Results of a Global Application Context PL/SQL Package for more information.

Policies to Attach the Function to the Objects You Want to Protect

The Oracle Virtual Private Database policy associates the VPD function with a table, view, or synonym.

You create the policy by using the `DBMS_RLS` package. If you are not `SYS`, then you must be granted `EXECUTE` privileges to use the `DBMS_RLS` package. This package contains procedures that enable you to manage the policy and set fine-grained access control. For example, to attach the policy to a table, you use the `DBMS_RLS.ADD_POLICY` procedure. Within this setting, you set fine-grained access control, such as setting the policy to go into effect when a user issues a `SELECT` or `UPDATE` statement on the table or view.

The combination of creating the function and then applying it to a table or view is referred to as creating the Oracle Virtual Private Database policy.

Configuration of Oracle Virtual Private Database Policies

The `DBMS_RLS` PL/SQL package can configure Oracle Virtual Private Database (VPD) policies.

About Oracle Virtual Private Database Policies

The Oracle Virtual Private Database policy associates the VPD function with a database table, view, or synonym.

This function defines the actions of the Oracle Virtual Private Database `WHERE` clause. You must then associate this function with the database table to which the Oracle Virtual Private Database (VPD) action applies.

You can do this by configuring an Oracle Virtual Private Database policy. The policy itself is a mechanism for managing the Virtual Private Database function. The policy also enables you to add fine-grained access control, such as specifying the types of SQL statements or particular table columns the policy affects. When a user tries to access the data in this database object, the policy goes into effect automatically.

Table 11-1 lists the procedures in the `DBMS_RLS` package.

Table 11-1 DBMS_RLS Procedures

Procedure	Description
For Handling Individual Policies	-
`DBMS_RLS.ADD_POLICY`	Adds a policy to a table, view, or synonym
`DBMS_RLS.ENABLE_POLICY`	Enables (or disables) a policy you previously added to a table, view, or synonym
`DBMS_RLS.ALTER_POLICY`	Alters an existing policy to associate or disassociate attributes with the policy
`DBMS_RLS.REFRESH_POLICY`	Invalidates cursors associated with nonstatic policies
`DBMS_RLS.DROP_POLICY`	To drop a policy from a table, view, or synonym
For Handling Grouped Policies	-
`DBMS_RLS.CREATE_POLICY_GROUP`	Creates a policy group
`DBMS_RLS.ALTER_GROUPED_POLICY`	Alters a policy group
`DBMS_RLS.DELETE_POLICY_GROUP`	Drops a policy group
`DBMS_RLS.ADD_GROUPED_POLICY`	Adds a policy to the specified policy group
`DBMS_RLS.ENABLE_GROUPED_POLICY`	Enables a policy within a group

Table 11-1 (Cont.) DBMS_RLS Procedures

Procedure	Description
`DBMS_RLS.REFRESH_GROUPED_POLICY`	Parses again the SQL statements associated with a refreshed policy
`DBMS_RLS.DISABLE_GROUPED_POLICY`	Disables a policy within a group
`DBMS_RLS.DROP_GROUPED_POLICY`	Drops a policy that is a member of the specified group
For Handling Application Contexts	-
`DBMS_RLS.ADD_POLICY_CONTEXT`	Adds the context for the active application
`DBMS_RLS.DROP_POLICY_CONTEXT`	Drops the context for the application

Attaching a Policy to a Database Table, View, or Synonym

The `DBMS_RLS` PL/SQL package can attach a policy to a table, view, or synonym.

- To attach a policy to a database table, view, or synonym, use the `DBMS_RLS.ADD_POLICY` procedure.

You must specify the table, view, or synonym to which you are adding a policy, and a name for the policy. You can also specify other information, such as the types of statements the policy controls (`SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE INDEX`, or `ALTER INDEX`).

Follow these guidelines:

- If a view has been created as an extended data-linked object, then Oracle recommends that you apply the same VPD policy on this type of view as you would on the underlying objects of the view.
- Determine if the base object to which you want to add the VPD policy has dependent objects. If it does have dependent objects, then these objects will become invalid when the VPD policy is added to the base object, and these objects will be recompiled automatically when they are used.

  Alternatively, you can proactively recompile them yourself by using an `ALTER ... COMPILE` statement. Be aware that invalidating dependent objects (by adding a VPD policy on their base object) and causing them to need to be recompiled can decrease performance in the overall system. Oracle recommends that you only add a VPD policy to an object that has dependent objects during off-peak hours or during a scheduled downtime.
- Be aware that the maximum number of policies that can be created for a single object is 255.

Example: Attaching a Simple Oracle Virtual Private Database Policy to a Table

The `DBMS_RLS.ADD_POLICY` procedure can attach an Oracle Virtual Private Database (VPD) policy to a table, view, or synomym.

Example 11-1 shows how to use DBMS_RLS.ADD_POLICY to attach an Oracle Virtual Private Database policy called secure_update to the HR.EMPLOYEES table. The function attached to the policy is check_updates.

Example 11-1 Attaching a Simple Oracle Virtual Private Database Policy to a Table

```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'check_updates',
...
```

If the function was created inside a package, include the package name. For example:

```
 policy_function => 'pkg.check_updates',
...
```

Although you can define a policy against a table, you cannot select that table from within the policy that was defined against the table.

Enforcing Policies on Specific SQL Statement Types

You can enforce Oracle Virtual Private Database policies for SELECT, INSERT, UPDATE, INDEX, and DELETE statements.

- To specify a SQL statement type for the policy, use the statement_types parameter in the DBMS_RLS.ADD_POLICY procedure. If you want to specify more than one, separate each with a comma. Enclose the list in a pair of single quotation marks.

If you do not specify a statement type, then by default, Oracle Database specifies SELECT, INSERT, UPDATE, and DELETE, but not INDEX. You can enter any combination of these statement types.

When you specify the statement_types parameter, be aware of the following functionality:

- **The application code affected by the Virtual Private Database policy can include the MERGE INTO statement.** However, in the Virtual Private Database policy, you must ensure that the statement_types parameter includes all three of the INSERT, UPDATE, and DELETE statements for the policy to succeed. Alternatively, you can omit the statement_types parameter.
- **Be aware that a user who has privileges to maintain an index can see all the row data, even if the user does not have full table access under a regular query such as SELECT.** For example, a user can create a function-based index that contains a user-defined function with column values as its arguments. During index creation, Oracle Database passes column values of every row into the user function, making the row data available to the user who creates the index. You can enforce Oracle Virtual Private Database policies on index maintenance operations by specifying INDEX with the statement_types parameter.

Example: Specifying SQL Statement Types with DBMS_RLS.ADD_POLICY

The `DBMS_RLS.ADD_POLICY` procedure `statement_types` parameter can specify the `SELECT` and `INDEX` statements for a policy.

Example 11-2 shows an how this works.

Example 11-2 Specifying SQL Statement Types with DBMS_RLS.ADD_POLICY

```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'check_updates',
  statement_types => 'SELECT,INDEX');
END;
/
```

Control of the Display of Column Data with Policies

You can create policies that enforce row-level security when a security-relevant column is referenced in a query.

Policies for Column-Level Oracle Virtual Private Database

Column-level policies enforce row-level security when a query references a security-relevant column.

You can apply a column-level Oracle Virtual Private Database policy to tables and views, but not to synonyms. To apply the policy to a column, specify the security-relevant column by using the `SEC_RELEVANT_COLS` parameter of the `DBMS_RLS.ADD_POLICY` procedure. This parameter applies the security policy whenever the column is referenced, explicitly or implicitly, in a query.

For example, users who are not in a Human Resources department typically are allowed to view only their own Social Security numbers. A sales clerk initiates the following query:

```
SELECT fname, lname, ssn FROM emp;
```

The function implementing the security policy returns the predicate `ssn='my_ssn'`. Oracle Database rewrites the query and executes the following:

```
SELECT fname, lname, ssn FROM emp
 WHERE ssn = 'my_ssn';
```

Example: Creating a Column-Level Oracle Virtual Private Database Policy

The `CREATE FUNCTION` statement and the `DBMS_RLS.ADD_POLICY` procedure can configure a column-level Oracle Virtual Private Database policy.

Example 11-3 shows an Oracle Virtual Private Database policy in which sales department users cannot see the salaries of people outside the department (department number 30) of the sales department users. The relevant columns for this policy are `sal` and `comm`. First, the Oracle Virtual Private Database policy function is created, and then it is added by using the `DBMS_RLS` PL/SQL package.

Example 11-3   Creating a Column-Level Oracle Virtual Private Database Policy

```
CREATE OR REPLACE FUNCTION hide_sal_comm (
 v_schema IN VARCHAR2,
 v_objname IN VARCHAR2)

RETURN VARCHAR2 AS
con VARCHAR2 (200);

BEGIN
 con := 'deptno=30';
 RETURN (con);
END hide_sal_comm;
```

Then you configure the policy with the `DBMS_RLS.ADD_POLICY` procedure as follows:

```
BEGIN
 DBMS_RLS.ADD_POLICY (
  object_schema     => 'scott',
  object_name       => 'emp',
  policy_name       => 'hide_sal_policy',
  policy_function   => 'hide_sal_comm',
  sec_relevant_cols => 'sal,comm');
END;
```

Display of Only the Column Rows Relevant to the Query

Be default, column-level Oracle Virtual Private Database restricts the number of rows a query returns that references columns containing sensitive information.

You specify these security-relevant columns by using the `SEC_RELEVANT_COLUMNS` parameter of the `DBMS_RLS.ADD_POLICY` procedure, as shown in Example 11-3.

For example, consider sales department users with the `SELECT` privilege on the `emp` table, which is protected with the column-level Oracle Virtual Private Database policy created in Example 11-3. The user (for example, user `SCOTT`) runs the following query:

```
SELECT ENAME, d.dname, JOB, SAL, COMM
 FROM emp e, dept d
 WHERE d.deptno = e.deptno;
```

The database returns the following rows:

```
ENAME      DNAME          JOB              SAL       COMM
---------- -------------- --------- ---------- ----------
ALLEN      SALES          SALESMAN        1600        300
WARD       SALES          SALESMAN        1250        500
MARTIN     SALES          SALESMAN        1250       1400
BLAKE      SALES          MANAGER         2850
TURNER     SALES          SALESMAN        1500          0
JAMES      SALES          CLERK            950

6 rows selected.
```

The only rows that are displayed are those that the user has privileges to access all columns in the row.

Column Masking to Display Sensitive Columns as NULL Values

If a query references a sensitive column, then by default column-level Oracle Virtual Private Database restricts the number of rows returned.

With column-masking behavior, all rows display, even those that reference sensitive columns. However, the sensitive columns display as `NULL` values. To enable column-masking, set the `SEC_RELEVANT_COLS_opt` parameter of the `DBMS_RLS.ADD_POLICY` procedure.

For example, consider the results of the sales clerk query, described in the previous example. If column-masking is used, then instead of seeing only the row containing the details and Social Security number of the sales clerk, the clerk would see all rows from the `emp` table, but the `ssn` column values would be returned as `NULL`. Note that this behavior is fundamentally different from all other types of Oracle Virtual Private Database policies, which return only a subset of rows.

In contrast to the default action of column-level Oracle Virtual Private Database, column-masking displays all rows, but returns sensitive column values as `NULL`. To include column-masking in your policy, set the `SEC_RELEVANT_COLS_OPT` parameter of the `DBMS_RLS.ADD_POLICY` procedure to `DBMS_RLS.ALL_ROWS`.

The following considerations apply to column masking:

- Column-masking applies only to `SELECT` statements.
- Column-masking conditions generated by the policy function must be simple Boolean expressions, unlike regular Oracle Virtual Private Database predicates.
- For applications that perform calculations, or do not expect `NULL` values, use standard column-level Oracle Virtual Private Database, specifying `SEC_RELEVANT_COLS` rather than the `SEC_RELEVANT_COLS_OPT` column-masking option.
- Do not include columns of the object data type (including the `XMLtype`) in the `sec_relevant_cols` setting. This column type is not supported for the `sec_relevant_cols` setting.
- Column-masking used with `UPDATE AS SELECT` updates only the columns that users are allowed to see.
- For some queries, column-masking may prevent some rows from displaying. For example:

  ```
  SELECT * FROM emp
   WHERE sal = 10;
  ```

  Because the column-masking option was set, this query may not return rows if the `salary` column returns a `NULL` value.

Example: Adding Column Masking to an Oracle Virtual Private Database Policy

The `DBMS_RLS.ADD_POLICY` procedure can configure column-level Oracle Virtual Private Database column masking.

Example 11-4 shows column-level Oracle Virtual Private Database column masking. It uses the same VPD policy as Example: Creating a Column-Level Oracle Virtual Private Database Policy, but with `sec_relevant_cols_opt` specified as `DBMS_RLS.ALL_ROWS`.

Example 11-4 Adding Column Masking to an Oracle Virtual Private Database Policy

```
BEGIN
 DBMS_RLS.ADD_POLICY(
   object_schema         => 'scott',
   object_name           => 'emp',
   policy_name           => 'hide_sal_policy',
   policy_function       => 'hide_sal_comm',
   sec_relevant_cols     =>' sal,comm',
   sec_relevant_cols_opt => dbms_rls.ALL_ROWS);
END;
```

Assume that a sales department user with `SELECT` privilege on the `emp` table (such as user `SCOTT`) runs the following query:

```
SELECT ENAME, d.dname, job, sal, comm
 FROM emp e, dept d
 WHERE d.deptno = e.deptno;
```

The database returns all rows specified in the query, but with certain values masked because of the Oracle Virtual Private Database policy:

```
ENAME      DNAME          JOB              SAL       COMM
---------- -------------- --------- ---------- ----------
CLARK      ACCOUNTING     MANAGER
KING       ACCOUNTING     PRESIDENT
MILLER     ACCOUNTING     CLERK
JONES      RESEARCH       MANAGER
FORD       RESEARCH       ANALYST
ADAMS      RESEARCH       CLERK
SMITH      RESEARCH       CLERK
SCOTT      RESEARCH       ANALYST
WARD       SALES          SALESMAN        1250        500
TURNER     SALES          SALESMAN        1500          0
ALLEN      SALES          SALESMAN        1600        300
JAMES      SALES          CLERK            950
BLAKE      SALES          MANAGER         2850
MARTIN     SALES          SALESMAN        1250       1400

14 rows selected.
```

The column-masking returned all rows requested by the sales user query, but made the `sal` and `comm` columns `NULL` for employees outside the sales department.

Oracle Virtual Private Database Policy Groups

An Oracle Virtual Private Database policy group is a named collection of VPD policies that can be applied to an application.

About Oracle Virtual Private Database Policy Groups

You can group multiple security policies together, and apply them to an application.

A policy group is a set of security policies that belong to an application. You can designate an application context (known as a *driving context* or *policy context*) to indicate the policy group in effect. Then, when a user accesses the table, view, or synonym column, Oracle Database looks up the driving context to determine the policy group in effect. It enforces all the associated policies that belong to the policy group.

Policy groups are useful for situations where multiple applications with multiple security policies share the same table, view, or synonym. This enables you to identify those policies that should be in effect when the table, view, or synonym is accessed.

For example, in a hosting environment, Company A can host the `BENEFIT` table for Company B and Company C. The table is accessed by two different applications, Human Resources and Finance, with two different security policies. The Human Resources application authorizes users based on ranking in the company, and the Finance application authorizes users based on department. Integrating these two policies into the `BENEFIT` table requires joint development of policies between the two companies, which is not a feasible option. By defining an application context to drive the enforcement of a particular set of policies to the base objects, each application can implement a private set of security policies.

To do this, you organize security policies into groups. By referring to the application context, Oracle Database determines which group of policies should be in effect at run time. The server enforces all the policies that belong to that policy group.

Creation of a New Oracle Virtual Private Database Policy Group

The `DBMS_RLS.ADD_GROUPED_POLICY` procedure adds a VPD policy to a VPD policy group.

To specify which policies will be effective, you can add a driving context using the `DBMS_RLS.ADD_POLICY_CONTEXT` procedure. If the driving context returns an unknown policy group, then an error is returned.

If the driving context is not defined, then Oracle Database runs all policies. Likewise, if the driving context is `NULL`, then policies from all policy groups are enforced. An application accessing the data cannot bypass the security setup module (which sets up application context) to avoid any applicable policies.

You can apply multiple driving contexts to the same table, view, or synonym, and each of them will be processed individually. This enables you to configure multiple active sets of policies to be enforced.

Consider, for example, a hosting company that hosts Benefits and Financial applications, which share some database objects. Both applications are striped for hosting using a `SUBSCRIBER` policy in the `SYS_DEFAULT` policy group. Data access is partitioned first by subscriber ID, then by whether the user is accessing the Benefits or Financial applications (determined by a driving context). Suppose that Company A, which uses the hosting services, wants to apply a custom policy that relates only to its own data access. You could add an additional driving context (such as `COMPANY A SPECIAL`) to ensure that the additional, special policy group is applied for data access for Company A only. You would not apply this under the `SUBSCRIBER` policy, because the policy relates only to Company A, and it is more efficient to segregate the basic hosting policy from other policies.

Default Policy Group with the SYS_DEFAULT Policy Group

Within a group of security policies, you can designate one security policy to be the default security policy.

This is useful in situations where you partition security policies by application, so that they will be always be in effect. Default security policies enable developers to base security enforcement under all conditions, while partitioning security policies by application (using security groups) enables layering of additional, application-specific security on top of default security policies. To implement default security policies, you add the policy to the SYS_DEFAULT policy group.

Policies defined in this group for a particular table, view, or synonym are run with the policy group specified by the driving context. As described earlier, a driving context is an application context that indicates the policy group in effect. The SYS_DEFAULT policy group may or may not contain policies. You cannot to drop the SYS_DEFAULT policy group. If you do, then Oracle Database displays an error.

If, to the SYS_DEFAULT policy group, you add policies associated with two or more objects, then each object will have a separate SYS_DEFAULT policy group associated with it. For example, the emp table in the scott schema has one SYS_DEFAULT policy group, and the dept table in the scott schema has a different SYS_DEFAULT policy group associated with it. Think of them as being organized in the tree structure as follows:

```
SYS_DEFAULT
  - policy1 (scott/emp)
  - policy3 (scott/emp)
SYS_DEFAULT
  - policy2 (scott/dept)
```

You can create policy groups with identical names. When you select a particular policy group, its associated schema and object name are displayed in the property sheet on the right side of the screen.

Multiple Policies for Each Table, View, or Synonym

You can establish several policies for the same table, view, or synonym.

Suppose, for example, you have a base application for Order Entry, and each division of your company has its own rules for data access. You can add a division-specific policy function to a table without having to rewrite the policy function of the base application.

All policies applied to a table are enforced with AND syntax. If you have three policies applied to the CUSTOMERS table, then each policy is applied to the table. You can use policy groups and an application context to partition fine-grained access control enforcement so that different policies apply, depending upon which application is accessing data. This eliminates the requirement for development groups to collaborate on policies, and simplifies application development. You can also have a default policy group that is always applicable (for example, to enforce data separated by subscriber in a hosting environment).

Validation of the Application Used to Connect to the Database

The package implementing the driving context must correctly validate the application that is being used to connect to the database.

Although Oracle Database checks the call stack to ensure that the package implementing the driving context sets context attributes, inadequate validation can still occur within the package. For example, in applications where database users or enterprise users are known to the database, the user needs the EXECUTE privilege on the package that sets the driving context. Consider a user who knows that the BENEFITS application enables more liberal access than the HR application. The setctx procedure (which sets the correct policy group within the driving context) does not perform any validation to determine which application is actually connecting. That is, the procedure does not check either the IP address of the incoming connection (for a three-tier system) or the `proxy_user` attribute of the user session.

This user could pass to the driving context package an argument setting the context to the more liberal `BENEFITS` policy group, and then access the `HR` application instead. Because the `setctx` does no further validation of the application, this user bypasses the more restrictive HR security policy.

By contrast, if you implement proxy authentication with Oracle Virtual Private Database, then you can determine the identity of the middle tier (and the application) that is connecting to the database on behalf of a user. The correct policy will be applied for each application to mediate data access.

For example, a developer using the proxy authentication feature could determine that the application (the middle tier) connecting to the database is `HRAPPSERVER`. The package that implements the driving context can thus verify whether the `proxy_user` in the user session is `HRAPPSERVER`. If so, then it can set the driving context to use the `HR` policy group. If `proxy_user` is not `HRAPPSERVER`, then it can deny access.

In this case, the following query is executed:

```
SELECT * FROM apps.benefit;
```

Oracle Database picks up policies from the default policy group (`SYS_DEFAULT`) and active namespace `HR`. The query is internally rewritten as follows:

```
SELECT * FROM apps.benefit
 WHERE company = SYS_CONTEXT('ID','MY_COMPANY')
 AND SYS_CONTEXT('ID','TITLE') = 'MANAGER';
```

Optimizing Performance by Using Oracle Virtual Private Database Policy Types

You can optimize performance by using the Oracle Virtual Private Database (VPD) the dynamic, static, or shared policy types.

About Oracle Virtual Private Database Policy Types

Specifying a policy type for your policies can optimize performance each the Oracle Virtual Private Database policy runs.

Policy types control how Oracle Database caches Oracle Virtual Private Database policy predicates. Consider setting a policy type for your policies, because the execution of policy functions can use a significant amount of system resources. Minimizing the number of times that a policy function can run optimizes database performance.

You can choose from five policy types: `DYNAMIC`, `STATIC`, `SHARED_STATIC`, `CONTEXT_SENSITIVE`, and `SHARED_CONTEXT_SENSITIVE`. These enable you to precisely specify how often a policy predicate should change. To specify the policy type, set the `policy_type` parameter of the `DBMS_RLS.ADD POLICY` procedure.

Dynamic Policy Type to Automatically Rerun Policy Functions

The `DYNAMIC` policy type runs the policy function each time a user accesses the Virtual Private Database-protected database objects.

If you do not specify a policy type in the DBMS_RLS.ADD_POLICY procedure, then, by default, your policy will be dynamic. You can specifically configure a policy to be dynamic by setting the policy_type parameter of the DBMS_RLS.ADD_POLICY procedure to DYNAMIC.

This policy type does not optimize database performance as the static and context sensitive policy types do. However, Oracle recommends that before you set policies as either static or context-sensitive, you should first test them as DYNAMIC policy types, which run every time. Testing policy functions as DYNAMIC policies first enables you to observe how the policy function affects each query, because nothing is cached. This ensures that the functions work properly before you enable them as static or context-sensitive policy types to optimize performance.

You can use the DBMS_UTILITY.GET_TIME function to measure the start and end times for a statement to execute. For example:

```
-- 1. Get the start time:
SELECT DBMS_UTILITY.GET_TIME FROM DUAL;

  GET_TIME
----------
   2312721

-- 2. Run the statement:
SELECT COUNT(*) FROM HR.EMPLOYEES;

  COUNT(*)
----------
       107

-- 3. Get the end time:
SELECT DBMS_UTILITY.GET_TIME FROM DUAL;

  GET_TIME
----------
   2314319
```

Example: Creating a DYNAMIC Policy with DBMS_RLS.ADD_POLICY

The DBMS_RLS.ADD_POLICY procedure can create a dynamic Oracle Virtual Private Database policy.

Example 11-5 shows how to create the DYNAMIC policy type.

Example 11-5 Creating a DYNAMIC Policy with DBMS_RLS.ADD_POLICY

```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.DYNAMIC);
END;
/
```

Static Policy to Prevent Policy Functions from Rerunning for Each Query

The static policy type enforces the same predicate for all users in the instance.

Oracle Database stores static policy predicates in SGA, so policy functions do not rerun for each query. This results in faster performance.

You can enable static policies by setting the `policy_type` parameter of the `DBMS_RLS.ADD_POLICY` procedure to either `STATIC` or `SHARED_STATIC`, depending on whether or not you want the policy to be shared across multiple objects.

Each execution of the same cursor could produce a different row set for the same predicate, because the predicate may filter the data differently based on attributes such as `SYS_CONTEXT` or `SYSDATE`.

For example, suppose you enable a policy as either a `STATIC` or `SHARED_STATIC` policy type, which appends the following predicate to all queries made against policy protected database objects:

```
WHERE dept = SYS_CONTEXT ('hr_app','deptno')
```

Although the predicate does not change for each query, it applies to the query based on session attributes of the `SYS_CONTEXT`. In the case of the preceding example, the predicate returns only those rows where the department number matches the `deptno` attribute of the `SYS_CONTEXT`, which is the department number of the user who is querying the policy-protected database object.

Note:

When using shared static policies, ensure that the policy predicate does not contain attributes that are specific to a particular database object, such as a column name.

Example: Creating a Static Policy with DBMS_RLS.ADD_POLICY

The `DBMS_RLS.ADD_POLICY` procedure can create a static Oracle Virtual Private Database (VPD) policy.

Example 11-6 shows how to create the `STATIC` policy type.

Example 11-6 Creating a Static Policy with DBMS_RLS.ADD_POLICY

```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => DBMS_RLS.STATIC);
END;
/
```

Example: Shared Static Policy to Share a Policy with Multiple Objects

The `DBMS_RLS.ADD_POLICY` procedure can create a shared static Oracle Virtual Private Database policy to share the policy with multiple objects.

If, for example, you wanted to apply the policy in Example 11-6 to a second table in the HR schema that may contain financial data that you want to side, you could use the SHARED_STATIC setting for both tables.

Example 11-7 shows how to set the SHARED_STATIC policy type for two tables that share the same policy.

Example 11-7   Creating a Shared Static Policy to Share a Policy with Multiple Objects

```
-- 1. Create a policy for the first table, employees:
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.SHARED_STATIC);
END;
/
-- 2. Create a policy for the second table, fin_data:
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'fin_data',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.SHARED_STATIC);
END;
/
```

When to Use Static and Shared Static Policies

Static policies are ideal when every query requires the same predicate and fast performance is essential, such as hosting environments.

For these situations when the policy function appends the same predicate to every query, rerunning the policy function each time adds unnecessary overhead to the system. For example, consider a data warehouse that contains market research data for customer organizations that are competitors. The warehouse must enforce the policy that each organization can see only their own market research, which is expressed by the following predicate:

```
WHERE subscriber_id = SYS_CONTEXT('customer', 'cust_num')
```

Using SYS_CONTEXT for the application context enables the database to dynamically change the rows that are returned. You do not need to rerun the function, so the predicate can be cached in the SGA, thus conserving system resources and improving performance.

Context-Sensitive Policy for Application Context Attributes That Change

Context-sensitive policies are useful when different predicates must be applied depending on which executes the query.

For example, consider the case where managers should have the predicate WHERE group set to managers, and employees should have the predicate WHERE empno_ctx set to emp_id. A context-sensitive policy will enable you to present only the information that the managers must see when the managers log in, and only the information that the employees must see when they log in. The policy uses application contexts to determine which predicate to use.

In contrast to static policies, context-sensitive policies do not always cache the predicate. With context-sensitive policies, the database assumes that the predicate will change after statement parse time. But if there is no change in the local application context, then Oracle Database does not rerun the policy function within the user session. If there is a change in any attribute of any application context during the user session, then by default the database re-executes the policy function to ensure that it captures all changes to the predicate since the initial parsing. This results in unnecessary re-executions of the policy function if none of the associated attributes have changed. You can restrict the evaluation to a specific application context by including both the `namespace` and `attribute` parameters.

If you plan to use the `namespace` and `attribute` parameters in your policy, then follow these guidelines:

- Ensure that you specify both `namespace` and `attribute` parameters, not just one.
- Ensure that your policy has the `policy_type` argument set to `DBMS_RLS.CONTEXT_SENSITIVE` or `SHARED_CONTEXT_SENSITIVE`. You cannot use the `namespace` and `attribute` parameters in static or dynamic policies.

If there are no attributes associated with the Virtual Private Database policy function, then Oracle Database evaluates the context-sensitive function for any application context changes.

Shared context-sensitive policies operate in the same way as regular context-sensitive policies, except they can be shared across multiple database objects. For this policy type, all objects can share the policy function from the UGA, where the predicate is cached until the local session context changes.

Example: Creating a Context-Sensitive Policy with DBMS_RLS.ADD_POLICY

The `DBMS_RLS.ADD_POLICY` procedure can create an Oracle Virtual Private Database context-sensitive policy.

Example 11-8shows how to create a `CONTEXT_SENSITIVE` policy in which the policy is evaluated only for changes to the `empno_ctx` namespace and `emp_id` attribute.

Example 11-8 Creating a Context-Sensitive Policy with DBMS_RLS.ADD_POLICY

```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.CONTEXT_SENSITIVE,
  namespace       => 'empno_ctx',
  attribute       => 'emp_id');
END;
/
```

Example: Refreshing Cached Statements for a VPD Context-Sensitive Policy

The `DBMS_RLS.REFRESH_POLICY` statement can refresh cached statements for Oracle Virtual Private Database context-sensitive policies.

Example 11-9 shows you can manually refresh all the cached statements that are associated with a Virtual Private Database context-sensitive policy by running the `DBMS_RLS.REFRESH_POLICY` procedure.

Example 11-9 Refreshing Cached Statements for a VPD Context-Sensitive Policy

```
BEGIN
 DBMS_RLS.REFRESH_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update');
END;
/
```

Example: Altering an Existing Context-Sensitive Policy

The `DBMS_RLS.ALTER_POLICY` procedure can modify an Oracle Virtual Private Database policy.

Example 11-10 shows how you can use the `DBMS_RLS.ALTER_POLICY` statement to alter an existing context-sensitive policy so that the `order_update_pol` policy function is executed only if the relevant context attributes change.

Example 11-10 Altering an Existing Context-Sensitive Policy

```
BEGIN
 DBMS_RLS.ALTER_POLICY(
  object_schema   => 'oe',
  object_name     => 'orders',
  policy_name     => 'order_update_pol',
  alter_option    =>  DBMS_RLS.ADD_ATTRIBUTE_ASSOCIATION,
  namespace       => 'empno_ctx',
  attribute       => 'emp_role');
END;
/
```

Example: Using a Shared Context Sensitive Policy to Share a Policy with Multiple Objects

The `DBMS_RLS.ADD_POLICY` procedure can create a shared context-sensitive Oracle Virtual Private Database to share a policy that has multiple objects.

Example 11-11 shows how to create two shared context sensitive policies that share a policy with multiple tables, and how to restrict the evaluation only for changes to the `empno_ctx` namespace and `emp_id` attribute.

Example 11-11 Shared Context-Sensitive Policy with DBMS_RLS.ADD_POLICY

```
-- 1. Create a policy for the first table, employees:
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.SHARED_CONTEXT_SENSITIVE,
  namespace       => 'empno_ctx',
  attribute       => 'emp_id');
```

```
END;
/
--2. Create a policy for the second table, fin_data:
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'fin_data',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.SHARED_CONTEXT_SENSITIVE,
  namespace       => 'empno_ctx',
  attribute       => 'emp_id');
END;
/
```

Note the following:

- When using shared context-sensitive policies, ensure that the policy predicate does not contain attributes that are specific to a particular database object, such as a column name.
- To manually refresh all the cached statements that are associated with a Virtual Private Database shared context-sensitive policy, run the `DBMS_RLS.REFRESH_GROUPED_POLICY` procedure.

When to Use Context-Sensitive and Shared Context-Sensitive Policies

Use context-sensitive policies when a predicate does not need to change for a user session, but the policy must enforce multiple predicates for different users or groups.

For example, consider a `sales_history` table with a single policy. This policy states that analysts can see only their own products and regional employees can see only their own region. In this case, the database must rerun the policy function each time the type of user changes. The performance gain is realized when a user can log in and issue several DML statements against the protected object without causing the server to rerun the policy function.

Note:

For session pooling where multiple clients share a database session, the middle tier must reset the context during client switches.

Summary of the Five Oracle Virtual Private Database Policy Types

Oracle Virtual Private Database provides five policy types, based on user needs such as hosting environments.

Table 11-2 summarizes the types of policy types available.

Table 11-2 DBMS_RLS.ADD_POLICY Policy Types

Policy Types	When the Policy Function Executes	Usage Example	Shared Across Multiple Objects?
`DYNAMIC`	Policy function re-executes every time a policy-protected database object is accessed.	Applications where policy predicates must be generated for each query, such as time-dependent policies where users are denied access to database objects at certain times during the day	No
`STATIC`	Once, then the predicate is cached in the SGA[1]	View replacement	No
`SHARED_STATIC`	Same as `STATIC`	Hosting environments, such as data warehouses where the same predicate must be applied to multiple database objects	Yes
`CONTEXT_SENSITIVE`	• At statement parse time • At statement execution time when the local application context changed since the last use of the cursor	Three-tier, session pooling applications where policies enforce two or more predicates for different users or groups	No
`SHARED_CONTEXT_SENSITIVE`	First time the object is reference in a database session. Predicates are cached in the private session memory UGA so policy functions can be shared among objects.	Same as `CONTEXT_SENSITIVE`, but multiple objects can share the policy function from the session UGA	Yes

1 Each execution of the same cursor could produce a different row set for the same predicate because the predicate may filter the data differently based on attributes such as `SYS_CONTEXT` or `SYSDATE`.

Tutorials: Creating Oracle Virtual Private Database Policies

These tutorials show how to create a simple and a database session-based Oracle Virtual Private policy, and how to create policy groups.

Tutorial: Creating a Simple Oracle Virtual Private Database Policy

This tutorial shows how to create a simple Oracle Virtual Private Database policy using the `OE` user account.

About This Tutorial

This tutorial shows how to create a VPD policy that limits access to orders created by Sales Representative 159 in the `OE.ORDERS` table.

In essence, the policy translates the following statement:

```
SELECT * FROM OE.ORDERS;
```

To the following statement:

```
SELECT * FROM OE.ORDERS WHERE SALES_REP_ID = 159;
```

> **Note:**
>
> If you are using a multitenant environment, then this tutorial applies to the current PDB only.

Step 1: Ensure That the OE User Account Is Active

First, you must ensure that `OE` user account is active.

1. Log on to SQL*Plus as user `SYS` with the `SYSDBA` administrative privilege.

   ```
   sqlplus sys as sysdba
   Enter password: password
   ```

2. In a multitenant environment, connect to the appropriate PDB.

   For example:

   ```
   CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

3. Query the `DBA_USERS` data dictionary view to find the account status of `OE`.

   ```
   SELECT USERNAME, ACCOUNT_STATUS FROM DBA_USERS WHERE USERNAME = 'OE';
   ```

   The status should be `OPEN`. If the `DBA_USERS` view lists user `OE` as locked and expired, then enter the following statement to unlock the `OE` account and create a new password:

   ```
   ALTER USER OE ACCOUNT UNLOCK IDENTIFIED BY password;
   ```

   Follow the guidelines in Minimum Requirements for Passwords to replace *`password`* with a password that is secure. For greater security, do not reuse the same password that was used in previous releases of Oracle Database.

Step 2: Create a Policy Function

Next, you are ready to create a policy function.

As user `SYS`, create the following function, which will append the `WHERE SALES_REP_ID = 159` clause to any `SELECT` statement on the `OE.ORDERS` table. (You can copy and paste this text by positioning the cursor at the start of `CREATE OR REPLACE` in the first line.)

```
CREATE OR REPLACE FUNCTION auth_orders(
  schema_var IN VARCHAR2,
  table_var  IN VARCHAR2
 )
 RETURN VARCHAR2
 IS
  return_val VARCHAR2 (400);
 BEGIN
  return_val := 'SALES_REP_ID = 159';
  RETURN return_val;
 END auth_orders;
/
```

In this example:

- `schema_var` and `table_var` create input parameters to specify to store the schema name, `OE`, and table name, `ORDERS`. First, define the parameter for the schema, and then define the parameter for the object, in this case, a table. Always create them in this order. The Virtual Private Database policy you create will need these parameters to specify the `OE.ORDERS` table.
- `RETURN VARCHAR2` returns the string that will be used for the `WHERE` predicate clause. Remember that return value is always a `VARCHAR2` data type.
- `IS ... RETURN return_val` encompasses the creation of the `WHERE SALES_REP_ID = 159` predicate.

Step 3: Create the Oracle Virtual Private Database Policy

After you create the policy function, you are ready to associate it with a VPD policy.

- Create the following policy by using the `ADD_POLICY` procedure in the `DBMS_RLS` package.

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema    => 'oe',
    object_name      => 'orders',
    policy_name      => 'orders_policy',
    function_schema  => 'sys',
    policy_function  => 'auth_orders',
    statement_types  => 'select'
   );
 END;
/
```

  In this example:

  - `object_schema => 'oe'` specifies the schema that you want to protect, that is, `OE`.
  - `object_name => 'orders'` specifies the object within the schema to protect, that is, the `ORDERS` table.
  - `policy_name => 'orders_policy'` names this policy `orders_policy`.
  - `function_schema => 'sys'` specifies the schema in which the `auth_orders` function was created. In this example, `auth_orders` was created in the `SYS` schema. But typically, it should be created in the schema of a security administrator.
  - `policy_function => 'auth_orders'` specifies a function to enforce the policy. Here, you specify the `auth_orders` function that you created in Step 2: Create a Policy Function.
  - `statement_types => 'select'` specifies the operations to which the policy applies. In this example, the policy applies to all `SELECT` statements that the user may perform.

Step 4: Test the Policy

After you create the Oracle Virtual Private Database policy, it goes into effect immediately.

The next time a user, including the owner of the schema, performs a `SELECT` on `OE.ORDERS`, only the orders by Sales Representative 159 will be accessed.

1. Connect as user `OE`.

   ```
   CONNECT oe -- Or, CONNECT OE@hrpdb
   Enter password: password
   ```

2. Enter the following `SELECT` statement:

   ```
   SELECT COUNT(*) FROM ORDERS;
   ```

   The following output should appear:

   ```
    COUNT(*)
   ---------
           7
   ```

   The policy is in effect for user `OE`: As you can see, only 7 of the 105 rows in the orders table are returned.

   But users with administrative privileges still have access to all the rows in the table.

3. Connect as user `SYS` with the `SYSDBA` administrative privilege.

   ```
   CONNECT SYS AS SYSDBA -- Or, CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password
   ```

4. Enter the following `SELECT` statement:

   ```
   SELECT COUNT(*) FROM OE.ORDERS;
   ```

   The following output should appear:

   ```
    COUNT(*)
   ---------
         105
   ```

Step 5: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. As user `SYS`, remove the function and policy as follows:

   ```
   DROP FUNCTION auth_orders;
   EXEC DBMS_RLS.DROP_POLICY('OE','ORDERS','ORDERS_POLICY');
   ```

2. If you need to lock and expire the `OE` account, then enter the following statement:

   ```
   ALTER USER OE ACCOUNT LOCK PASSWORD EXPIRE;
   ```

Tutorial: Implementing a Session-Based Application Context Policy

This tutorial demonstrates how to create an Oracle Virtual Private Database policy that uses a database session-based application context.

About This Tutorial

This tutorial shows how to use a database session-based application context to implement a policy in which customers see only their own orders.

If you are using a multitenant environment, then this tutorial applies to the current PDB only.

In this tutorial, you create the following layers of security:

1. When a user logs on, a database session-based application context checks whether the user is a customer. If a user is not a customer, the user still can log on, but this user cannot access the orders entry table you will create for this example.
2. If the user is a customer, he or she can log on. After the customer has logged on, an Oracle Virtual Private Database policy restricts this user to see only his or her orders.
3. As a further restriction, the Oracle Virtual Private Database policy prevents users from adding, modifying, or removing orders.

Step 1: Create User Accounts and Sample Tables

First, create user accounts and the sample tables.

1. Start SQL*Plus and log on as a user who has administrative privileges.

   ```
   sqlplus sys as sysdba
   Enter password: password
   ```

2. In a multitenant environment, connect to the appropriate PDB.

   For example:

   ```
   CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password
   ```

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.

3. Create the following administrative user, who will administer the Oracle Virtual Private Database policy.

   The following SQL statements create this user and then grant the user the necessary privileges for completing this tutorial.

```
CREATE USER sysadmin_vpd IDENTIFIED BY password CONTAINER = CURRENT;
GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE, CREATE TRIGGER, ADMINISTER DATABASE
TRIGGER TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_SESSION TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_RLS TO sysadmin_vpd;
```

   Follow the guidelines in Minimum Requirements for Passwords to replace *password* with a password that is secure.

4. Create the following local users:

   ```
   CREATE USER tbrooke IDENTIFIED BY password CONTAINER = CURRENT;
   CREATE USER owoods IDENTIFIED BY password CONTAINER = CURRENT;
   ```

```
GRANT CREATE SESSION TO tbrooke, owoods;
```

Replace *password* with a password that is secure.

5. Check the account status of the sample user SCOTT, who you will use for this tutorial:

   ```
   SELECT USERNAME, ACCOUNT_STATUS FROM DBA_USERS WHERE USERNAME = 'SCOTT';
   ```

   The status should be OPEN. If the DBA_USERS view lists user SCOTT as locked and expired, then enter the following statement to unlock the SCOTT account and create a new password for him:

   ```
   ALTER USER SCOTT ACCOUNT UNLOCK IDENTIFIED BY password;
   ```

   Follow the guidelines in Minimum Requirements for Passwords to replace *password* with a password that is secure. For greater security, do not reuse the same password that was used in previous releases of Oracle Database.

6. Connect as user SCOTT.

   ```
   CONNECT SCOTT -- Or, CONNECT SCOTT@hrpdb
   Enter password: password
   ```

7. Create and populate the customers table.

   ```
   CREATE TABLE customers (
    cust_no    NUMBER(4),
    cust_email VARCHAR2(20),
    cust_name  VARCHAR2(20));

   INSERT INTO customers VALUES (1234, 'TBROOKE', 'Thadeus Brooke');
   INSERT INTO customers VALUES (5678, 'OWOODS', 'Oberon Woods');
   ```

   When you enter the user email IDs, enter them in upper-case letters. Later on, when you create the application context PL/SQL package, the SESSION_USER parameter of the SYS_CONTEXT function expects the user names to be in upper case. Otherwise, you will be unable to set the application context for the user.

8. User sysadmin_vpd will need SELECT privileges for the customers table, so as user SCOTT, grant him this privilege.

   ```
   GRANT READ ON customers TO sysadmin_vpd;
   ```

9. Create and populate the orders_tab table.

   ```
   CREATE TABLE orders_tab (
     cust_no  NUMBER(4),
     order_no NUMBER(4));

   INSERT INTO orders_tab VALUES (1234, 9876);
   INSERT INTO orders_tab VALUES (5678, 5432);
   INSERT INTO orders_tab VALUES (5678, 4592);
   ```

10. Users tbrooke and owoods need to query the orders_tab table, so grant them the READ object privilege.

```
GRANT READ ON orders_tab TO tbrooke, owoods;
```

At this stage, the two sample customers, tbrooke and owoods, have a record of purchases in the orders_tab order entry table, and if they tried right now, they can see all the orders in this table.

Step 2: Create a Database Session-Based Application Context

Next, you are ready to create the database session-based application context.

1. Connect as user `sysadmin_vpd`.

   ```
   CONNECT sysadmin_vpd -- Or, CONNECT sysadmin_vpd@hrpdb
   Enter password: password
   ```

2. Enter the following statement:

   ```
   CREATE OR REPLACE CONTEXT orders_ctx USING orders_ctx_pkg;
   ```

   This statement creates the `orders_ctx` application context. Remember that even though user `sysadmin_vpd` has created this context and it is associated with the `sysadmin_vpd` schema, the `SYS` schema owns the application context.

Step 3: Create a PL/SQL Package to Set the Application Context

After you create the application context, you are ready to create a package to set the context.

- As user `sysadmin_vpd`, create the following PL/SQL package, which will set the database session-based application context when the customers `tbrooke` and `owoods` log onto their accounts.

  ```
  CREATE OR REPLACE PACKAGE orders_ctx_pkg IS
    PROCEDURE set_custnum;
   END;
  /
  CREATE OR REPLACE PACKAGE BODY orders_ctx_pkg IS
    PROCEDURE set_custnum
    AS
      custnum NUMBER;
    BEGIN
       SELECT cust_no INTO custnum FROM SCOTT.CUSTOMERS
          WHERE cust_email = SYS_CONTEXT('USERENV', 'SESSION_USER');
       DBMS_SESSION.SET_CONTEXT('orders_ctx', 'cust_no', custnum);
    EXCEPTION
     WHEN NO_DATA_FOUND THEN NULL;
    END set_custnum;
  END;
  /
  ```

  In this example:

  – `custnum NUMBER` creates the `custnum` variable, which will hold the customer ID.

  – `SELECT cust_no INTO custnum` performs a `SELECT` statement to copy the customer ID that is stored in the `cust_no` column data from the `scott.customers` table into the `custnum` variable.

  – `WHERE cust_email = SYS_CONTEXT('USERENV', 'SESSION_USER')` uses a `WHERE` clause to find all the customer IDs that match the user name of the user who is logging on.

  – `DBMS_SESSION.SET_CONTEXT('orders_ctx', 'cust_no', custnum)` sets the `orders_ctx` application context values by creating the `cust_no` attribute and then setting it to the value stored in the `custnum` variable.

   – `EXCEPTION ... WHEN` adds a `WHEN NO_DATA_FOUND` system exception to catch any `no data found` errors that may result from the `SELECT` statement in the `SELECT cust_no INTO custnum ...` statement.

To summarize, the `sysadmin_vpd.set_cust_num` procedure identifies whether or not the session user is a registered customer by attempting to select the user's customer ID into the `custnum` variable. If the user is a registered customer, then Oracle Database sets an application context value for this user. As you will see in Step 6: Create a PL/SQL Policy Function to Limit User Access to Their Orders, the policy function uses the context value to control the access a user has to data in the `orders_tab` table.

Step 4: Create a Logon Trigger to Run the Application Context PL/SQL Package

The logon trigger runs the PL/SQL package procedure so that the next time a user logs on, the application context is set.

- As user `sysadmin_vpd`, create the following logon trigger:

```
CREATE TRIGGER set_custno_ctx_trig AFTER LOGON ON DATABASE
 BEGIN
  sysadmin_vpd.orders_ctx_pkg.set_custnum;
 END;
/
```

Step 5: Test the Logon Trigger

The logon trigger sets the application context for the user when the trigger runs the `sysadmin_vpd.orders_ctx_pkg.set_custnum` procedure.

1. Connect as user `tbrooke`.

```
CONNECT tbrooke -- For a CDB, connect to the PDB, e.g., @hrpdb
Enter password: password
```

2. Execute the following query:

```
SELECT SYS_CONTEXT('orders_ctx', 'cust_no') custnum FROM DUAL;
```

   The following output should appear:

```
EMP_ID
-------------------------------------------------------------------
1234
```

Step 6: Create a PL/SQL Policy Function to Limit User Access to Their Orders

The next step is to create a PL/SQL function to control the display of the user's query.

When the user who has logged in performs a `SELECT * FROM scott.orders_tab` query, the function should cause the output to display only the orders of that user.

1. Connect as user `sysadmin_vpd`.

```
CONNECT sysadmin_vpd -- Or, CONNECT sysadmin_vpd@hrpdb
Enter password: password
```

2. Create the following function:

```
CREATE OR REPLACE FUNCTION get_user_orders(
  schema_p   IN VARCHAR2,
  table_p    IN VARCHAR2)
 RETURN VARCHAR2
 AS
  orders_pred VARCHAR2 (400);
 BEGIN
  orders_pred := 'cust_no = SYS_CONTEXT(''orders_ctx'', ''cust_no'')';
 RETURN orders_pred;
END;
/
```

This function creates and returns a `WHERE` predicate that translates to "where the orders displayed belong to the user who has logged in." It then appends this `WHERE` predicate to any queries this user may run against the `scott.orders_tab` table. Next, you are ready to create an Oracle Virtual Private Database policy that applies this function to the `orders_tab` table.

Step 7: Create the New Security Policy

Finally, you are ready to create the VPD security policy.

- As user `sysadmin_vpd`, use the `DBMS_RLS.ADD_POLICY` procedure to create the policy as follows:

```
BEGIN
 DBMS_RLS.ADD_POLICY (
  object_schema    => 'scott',
  object_name      => 'orders_tab',
  policy_name      => 'orders_policy',
  function_schema  => 'sysadmin_vpd',
  policy_function  => 'get_user_orders',
  statement_types  => 'select',
  policy_type      => DBMS_RLS.CONTEXT_SENSITIVE,
  namespace        => 'orders_ctx',
  attribute        => 'cust_no');
END;
/
```

This statement creates a policy named `orders_policy` and applies it to the `orders_tab` table, which customers will query for their orders, in the `SCOTT` schema. The `get_user_orders` function implements the policy, which is stored in the `sysadmin_vpd` schema. The policy further restricts users to issuing `SELECT` statements only. The `namespace` and `attribute` parameters specify the application context that you created earlier.

Step 8: Test the New Policy

Now that you have created all the components, you are ready to test the policy.

1. Connect as user `tbrooke`.

```
CONNECT tbrooke -- Or, CONNECT tbrooke@hrpdb
Enter password: password
```

   User `tbrooke` can log on because he has passed the requirements you defined in the application context.

2. As user `tbrooke`, access your purchases.

```
SELECT * FROM scott.orders_tab;
```

The following output should appear:

```
   CUST_NO    ORDER_NO
----------  ----------
      1234        9876
```

User `tbrooke` has passed the second test. He can access his own orders in the `scott.orders_tab` table.

3. Connect as user `owoods`, and then access your purchases.

   ```
   CONNECT owoods -- For a CDB, connect to the PDB, e.g., @hrpdb
   Enter password: password

   SELECT * FROM scott.orders_tab
   ```

   The following output should appear:

   ```
      CUST_NO    ORDER_NO
   ----------  ----------
         5678        5432
         5678        4592
   ```

   As with user `tbrooke`, user `owoods` can log on and see a listing of his own orders.

Note the following:

- You can create several predicates based on the position of a user. For example, a sales representative would be able to see records only for his customers, and an order entry clerk would be able to see any customer order. You could expand the `custnum_sec` function to return different predicates based on the user position context value.
- The use of an application context in a fine-grained access control package effectively gives you a bind variable in a parsed statement. For example:

  ```
  SELECT * FROM scott.orders_tab
  WHERE cust_no = SYS_CONTEXT('order_entry', 'cust_num');
  ```

  This is fully parsed and optimized, but the evaluation of the `cust_num` attribute value of the user for the `order_entry` context takes place at run-time. This means that you get the benefit of an optimized statement that executes differently for each user who issues the statement.

  Note:

  You can improve the performance of the function in this tutorial by indexing `cust_no`.

- You can set context attributes based on data from a database table or tables, or from a directory server using Lightweight Directory Access Protocol (LDAP).

Note:

Oracle Database PL/SQL Language Reference for more information about triggers

Compare this tutorial, which uses an application context within the dynamically generated predicate, with About Oracle Virtual Private Database Policies, which uses a subquery in the predicate.

Step 9: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. Connect as user `SCOTT`.

   ```
   CONNECT SCOTT -- Or, CONNECT SCOTT@hrpdb
   Enter password: password
   ```

2. Remove the `orders_tab` and `customers` tables.

   ```
   DROP TABLE orders_tab;
   DROP TABLE customers;
   ```

3. Connect as user `SYS`, connecting with `AS SYSDBA`.

   ```
   CONNECT SYS AS SYSDBA -- Or, CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password
   ```

4. Run the following statements to drop the components for this tutorial:

   ```
   DROP CONTEXT orders_ctx;
   DROP USER sysadmin_vpd CASCADE;
   DROP USER tbrooke;
   DROP USER owoods;
   ```

Tutorial: Implementing an Oracle Virtual Private Database Policy Group

This tutorial demonstrates how to create an Oracle Virtual Private Database policy group.

About This Tutorial

This tutorial shows how you can use Oracle Virtual Private Database (VPD) to create a policy group.

Oracle Virtual Private Database Policy Groups describes how you can group a set of policies for use in an application. When a nondatabase user logs onto the application, Oracle Database grants the user access based on the policies defined within the appropriate policy group.

For column-level access control, every column or set of hidden columns is controlled by one policy. In this tutorial, you must hide two sets of columns. So, you must create two policies, one for each set of columns that you want to hide. You only want one policy for each user; the driving application context separates the policies for you.

Note:

If you are using a multitenant environment, then this tutorial applies to the current PDB only.

Step 1: Create User Accounts and Other Components for This Tutorial

First, you must create user accounts and tables for this tutorial, and grant the appropriate privileges.

1. Log on as user `SYS` with the `SYSDBA` administrative privilege.

   ```
   sqlplus sys as sysdba
   Enter password: password
   ```

2. In a multitenant environment, connect to the appropriate PDB.

   For example:

   ```
   CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

3. Create the following local users:

   ```
   CREATE USER apps_user IDENTIFIED BY password CONTAINER = CURRENT;
   GRANT CREATE SESSION TO apps_user;
   CREATE USER sysadmin_pg  IDENTIFIED BY password CONTAINER = CURRENT;
   GRANT CREATE SESSION, CREATE PROCEDURE, CREATE ANY CONTEXT TO sysadmin_pg;
   ```

   Follow the guidelines in Minimum Requirements for Passwords to replace *`password`* with a password that is secure.

4. Grant the following additional privilege to user `sysadmin_pg`:

   ```
   GRANT EXECUTE ON DBMS_RLS TO sysadmin_pg;
   ```

5. Log on as user `OE`.

   ```
   CONNECT OE -- Or, CONNECT OE@hrpdb
   Enter password: password
   ```

   If the `OE` account is locked and expired, then reconnect as user `SYS` with the `SYSDBA` administrative privilege and enter the following statement to unlock the account and give it s new password:

   ```
   ALTER USER OE ACCOUNT UNLOCK IDENTIFIED BY password;
   ```

   Replace *`password`* with a password that is secure. For greater security, do not reuse the same password that was used in previous releases of Oracle Database.

6. Create the `product_code_names` table:

   ```
   CREATE TABLE product_code_names(
   group_a     varchar2(32),
   year_a      varchar2(32),
   group_b     varchar2(32),
   year_b      varchar2(32));
   ```

7. Insert some values into the `product_code_names` table:

   ```
   INSERT INTO product_code_names values('Biffo','2008','Beffo','2004');
   INSERT INTO product_code_names values('Hortensia','2008','Bunko','2008');
   INSERT INTO product_code_names values('Boppo','2006','Hortensia','2003');

   COMMIT;
   ```

8. Grant the `apps_user` user `SELECT` privileges on the `product_code_names` table.

   ```
   GRANT SELECT ON product_code_names TO apps_user;
   ```

Step 2: Create the Two Policy Groups

Next, you must create a policy group for each of the two nondatabase users, `provider_a` and `provider_b`.

1. Connect as user `sysadmin_pg`.

   ```
   CONNECT sysadmin_pg -- Or, CONNECT sysadmin_pg@hrpdb
   Enter password: password
   ```

2. Create the `provider_a_group` policy group, to be used by user `provider_a`:

   ```
   BEGIN
    DBMS_RLS.CREATE_POLICY_GROUP(
    object_schema   => 'oe',
    object_name     => 'product_code_names',
    policy_group    => 'provider_a_group');
   END;
   /
   ```

3. Create the `provider_b_group` policy group, to be used by user `provider_b`:

   ```
   BEGIN
    DBMS_RLS.CREATE_POLICY_GROUP(
    object_schema   => 'oe',
    object_name     => 'product_code_names',
    policy_group    => 'provider_b_group');
   END;
   /
   ```

Step 3: Create PL/SQL Functions to Control the Policy Groups

A policy group must have a function that defines how the application can control data access for users.

The function that you will create for this policy group applies to users `provider_a` and `provider_b`.

1. Create the `vpd_function_provider_a` function, which restricts the data accessed by user `provider_a`.

   ```
   CREATE OR REPLACE FUNCTION vpd_function_provider_a
    (schema in varchar2, tab in varchar2) return varchar2 as
     predicate  varchar2(8) default NULL;
     BEGIN
      IF LOWER(SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER')) = 'provider_a'
       THEN predicate := '1=2';
      ELSE NULL;
     END IF;
     RETURN predicate;
   END;
   /
   ```

   This function checks that the user logging in is really user `provider_a`. If this is true, then only the data in the `product_code_names` table columns `group_a` and `year_a` will be visible to `provider_a`. Data in columns `group_b` and `year_b` will not appear for `provider_a`. This works as follows: Setting `predicate := '1=2'` hides the relevant columns. In Step 5: Add the PL/SQL Functions to the Policy Groups, you specify these columns in the `SEC_RELEVANT_COLS` parameter.

2. Create the `vpd_function_provider_b`, function, which restricts the data accessed by user `provider_a`.

```
CREATE OR REPLACE FUNCTION vpd_function_provider_b
 (schema in varchar2, tab in varchar2) return varchar2 as
  predicate  varchar2(8) default NULL;
  BEGIN
   IF LOWER(SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER')) = 'provider_b'
    THEN predicate := '1=2';
   ELSE NULL;
  END IF;
  RETURN predicate;
END;
/
```

Similar to the `vpd_function_provider_a` function, this function checks that the user logging in is really user `provider_b`. If this is true, then only the data in the columns `group_b` and `year_b` will be visible to `provider_b`, with data in the `group_a` and `year_a` not appearing for `provider_b`. Similar to the `vpd_function_provider_a` function, `predicate := '1=2'` hides the relevant columns specified Step 5: Add the PL/SQL Functions to the Policy Groups in the `SEC_RELEVANT_COLS` parameter.

Step 4: Create the Driving Application Context

The application context determines which policy the nondatabase user who is the logging on should use.

1. As user `sysadmin_pg`, create the driving application context as follows:

```
CREATE OR REPLACE CONTEXT provider_ctx USING provider_package;
```

2. Create the PL/SQL `provider_package` package for the application context.

```
CREATE OR REPLACE PACKAGE provider_package IS
 PROCEDURE set_provider_context (policy_group varchar2 default NULL);
END;
/
CREATE OR REPLACE PACKAGE BODY provider_package AS
 PROCEDURE set_provider_context (policy_group varchar2 default NULL) IS
 BEGIN
  CASE LOWER(SYS_CONTEXT('USERENV', 'CLIENT_IDENTIFIER'))
   WHEN 'provider_a' THEN
    DBMS_SESSION.SET_CONTEXT('provider_ctx','policy_group','PROVIDER_A_GROUP');
   WHEN 'provider_b' THEN
    DBMS_SESSION.SET_CONTEXT('provider_ctx','policy_group','PROVIDER_B_GROUP');
  END CASE;
 END set_provider_context;
END;
/
```

3. Associate the `provider_ctx` application context with the `product_code_names` table, and then provide a name.

```
BEGIN
 DBMS_RLS.ADD_POLICY_CONTEXT(
 object_schema  =>'oe',
 object_name    =>'product_code_names',
 namespace      =>'provider_ctx',
 attribute      =>'policy_group');
```

```
END;
/
```

4. Grant the `apps_user` account the `EXECUTE` privilege for the `provider_package` package.

   ```
   GRANT EXECUTE ON provider_package TO apps_user;
   ```

Step 5: Add the PL/SQL Functions to the Policy Groups

Now that you have created the necessary functions, you are ready to associate them with their appropriate policy groups.

1. Add the `vpd_function_provider_a` function to the `provider_a_group` policy group.

   ```
   BEGIN
    DBMS_RLS.ADD_GROUPED_POLICY(
    object_schema         => 'oe',
    object_name           => 'product_code_names',
    policy_group          => 'provider_a_group',
    policy_name           => 'filter_provider_a',
    function_schema       => 'sysadmin_pg',
    policy_function       => 'vpd_function_provider_a',
    statement_types       => 'select',
    policy_type           => DBMS_RLS.CONTEXT_SENSITIVE,
    sec_relevant_cols     => 'group_b,year_b',
    sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS,
    namespace             => 'provider_ctx',
    attribute             => 'provider_group');
   END;
   /
   ```

   The `group_b` and `year_b` columns specified in the `sec_relevant_cols` parameter are hidden from user `provider_a`.

2. Add the `vpd_function_provider_b` function to the `provider_b_group` policy group.

   ```
   BEGIN
    DBMS_RLS.ADD_GROUPED_POLICY(
    object_schema         => 'oe',
    object_name           => 'product_code_names',
    policy_group          => 'provider_b_group',
    policy_name           => 'filter_provider_b',
    function_schema       => 'sysadmin_pg',
    policy_function       => 'vpd_function_provider_b',
    statement_types       => 'select',
    policy_type           => DBMS_RLS.CONTEXT_SENSITIVE,
    sec_relevant_cols     => 'group_a,year_a',
    sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS,
    namespace             => 'provider_ctx',
    attribute             => 'provider_group');
   END;
   /
   ```

   The `group_a` and `year_a` columns specified in the `sec_relevant_cols` parameter are hidden from user `provider_b`.

Step 6: Test the Policy Groups

Now you are ready to test the two policy groups.

1. Connect as user `apps_user` and then enter the following statements to ensure that the output you will create later on is nicely formatted.

   ```
   CONNECT apps_user -- Or, CONNECT apps_user@hrpdb
   Enter password: password

   col group_a format a16
   col group_b format a16;
   col year_a format a16;
   col year_b format a16;
   ```

2. Set the session identifier to `provider_a`.

   ```
   EXEC DBMS_SESSION.SET_IDENTIFIER('provider_a');
   ```

   Here, the application sets the identifier. Setting the identifier to `provider_a` sets the `apps_user` user to a user who should only see the products available to products in the `provider_a_group` policy group.

3. Run the `provider_package` to set the policy group based on the context.

   ```
   EXEC sysadmin_pg.provider_package.set_provider_context;
   ```

   At this stage, you can check the application context was set, as follows:

   ```
   SELECT SYS_CONTEXT('USERENV', 'CLIENT_IDENTIFIER') AS END_USER FROM DUAL;
   ```

   The following output should appear:

   ```
   END_USER
   -----------------
   provider_a
   ```

4. Enter the following `SELECT` statement:

   ```
   SELECT * FROM oe.product_code_names;
   ```

   The following output should appear:

   ```
   GROUP_A          YEAR_A           GROUP_B          YEAR_B
   ---------------- ---------------- ---------------- ----------------
   Biffo            2008
   Hortensia        2008
   Boppo            2006
   ```

5. Set the client identifier to `provider_b` and then enter the following statements:

   ```
   EXEC DBMS_SESSION.SET_IDENTIFIER('provider_b');
   EXEC sysadmin_pg.provider_package.set_provider_context;
   SELECT * FROM oe.product_code_names;
   ```

   The following output should appear:

   ```
   GROUP_A          YEAR_A           GROUP_B          YEAR_B
   ---------------- ---------------- ---------------- ----------------
                                     Beffo            2004
                                     Bunko            2008
                                     Hortensia        2003
   ```

Step 7: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. Connect as user `OE`.

```
CONNECT OE -- Or, CONNECT OE@hrpdb
Enter password: password
```

2. Drop the `product_code_names` table.

   ```
   DROP TABLE product_code_names;
   ```

3. Connect as user `SYS` with the `SYSDBA` administrative privilege.

   ```
   CONNECT SYS AS SYSDBA -- Or, CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password
   ```

4. Drop the application context and users for this tutorial.

   ```
   DROP CONTEXT provider_ctx;
   DROP USER sysadmin_pg cascade;
   DROP USER apps_user;
   ```

How Oracle Virtual Private Database Works with Other Oracle Features

You should be aware of the impact of using Oracle Virtual Private Database with other Oracle features.

Oracle Virtual Private Database Policies with Editions

You should be aware of how to use Oracle VPD with editions.

If you are preparing an application for edition-based redefinition, and you cover each table that the application uses with an editioning view, then you must move the Virtual Private Database polices that protect these tables to the editioning view.

When an editioned object has a Virtual Private Database policy, then it applies in all editions in which the object is visible. When an editioned object is actualized, any VPD policies that are attached to it are newly attached to the new actual occurrence. When you newly apply a VPD policy to an inherited editioned object, this action will actualize it.

See Also:

Oracle Database Development Guide for detailed information about editions

SELECT FOR UPDATE Statement in User Queries on VPD-Protected Tables

As a general rule, users should not include the `FOR UPDATE` clause when querying Virtual Private Database-protected tables.

The Virtual Private Database technology depends on rewriting the user's query against an inline view that includes the VPD predicate generated by the VPD policy function. Because of this, the same limitations on views also apply to VPD-protected tables. If a user's query against a VPD-protected table includes the `FOR UPDATE` clause in a `SELECT` statement, in most cases, the query may not work. However, the user's query may work in some situations if the inline view generated by VPD is sufficiently simple.

See Also:

Oracle Database SQL Language Reference for more information about the restrictions of the `FOR UPDATE` clause in the `SELECT` statement

Oracle Virtual Private Database Policies and Outer or ANSI Joins

Oracle Virtual Private Database rewrites SQL by using dynamic views.

For SQL that contains outer join or ANSI operations, some views may not merge and some indexes may not be used. This problem is a known optimization limitation. To remedy this problem, rewrite the SQL to not use outer joins or ANSI operations.

Oracle Virtual Private Database Security Policies and Applications

An Oracle Virtual Private Database security policy is applied within the database itself, rather than within an application.

Hence, a user trying to access data by using a different application cannot bypass the Oracle Virtual Private Database security policy. Another advantage of creating the security policy in the database is that you maintain it in one central place, rather than maintaining individual security policies in multiple applications. Oracle Virtual Private Database provides stronger security than application-based security, at a lower cost of ownership.

You may want to enforce different security policies depending on the application that is accessing data. Consider a situation in which two applications, Order Entry and Inventory, both access the `orders` table. You may want to have the Inventory application use a policy that limits access based on type of product. At the same time, you may want to have the Order Entry application use a policy that limits access based on customer number.

In this case, you must partition the use of fine-grained access by application. Otherwise, both policies would be automatically concatenated together, which may not be the result that you want. You can specify two or more policy groups, and a driving application context that determines which policy group is in effect for a given transaction. You can also designate default policies that always apply to data access. In a hosted application, for example, data access should be limited by subscriber ID.

Automatic Reparsing for Fine-Grained Access Control Policies Functions

Queries against objects enabled with fine-grained access control run the policy function so that the most current predicate is used for each policy.

For example, in the case of a time-based policy function, in which queries are only allowed between 8:00 a.m. and 5:00 p.m., a cursor execution parsed at noon runs the policy function at that time, ensuring that the policy is consulted again for the query. Even if the curser was parsed at 9 a.m., when it runs later on (for example, at noon), then the Virtual Private Database policy function runs again to ensure that the execution of the cursor is still permitted at the current time (noon). This ensures that the security check it must perform is the most recent.

Automatic re-execution of the Virtual Private Database policy function does not occur when you set the `DBMS_RLS.ADD_POLICY` setting `STATIC_POLICY` to `TRUE` while adding the policy. This setting causes the policy function to return the same predicate.

Oracle Virtual Private Database Policies and Flashback Queries

Operations on the database use the most recently committed data available.

The flashback query feature enables you to query the database at some point in the past.

To write an application that uses flashback query, you can use the `AS OF` clause in SQL queries to specify either a time or a system change number (SCN), and then query against the committed data from the specified time. You can also use the `DBMS_FLASHBACK` PL/SQL package, which requires more code, but enables you to perform multiple operations, all of which refer to the same point in time.

However, if you use flashback query against a database object that is protected with Oracle Virtual Private Database policies, then the current policies are applied to the old data. Applying the current Oracle Virtual Private Database policies to flashback query data is more secure because it reflects the most current business policy.

See Also:

- *Oracle Database Development Guide* for more information about the flashback query feature and how to write applications that use it
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_FLASHBACK` PL/SQL package

Oracle Virtual Private Database and Oracle Label Security

You can use Oracle Virtual Private Database with Oracle Label Security, but when you do, you should be aware of security exceptions.

Using Oracle Virtual Private Database to Enforce Oracle Label Security Policies

Oracle Virtual Private Database policies provide column or row-level access control based on Oracle Label Security user authorizations.

In general, you must perform the following steps:

1. When you create the Oracle Label Security policy, do not apply the policy to the table that you want to protect. (The Virtual Private Database policy that you create handles this for you.) In the `SA_SYSDBA.CREATE_POLICY` procedure, set the `default_options` parameter to `NO_CONTROL`.

2. Create the Oracle Label Security label components and authorize users as you normally would.
3. When you create the Oracle Virtual Private Database policy, do the following:
   - In the PL/SQL function you create for the policy, use the Oracle Label Security `DOMINATES` function to compare the authorization of the user with the label that you created in Step 2. The `DOMINATES` function determines if the user authorization is equal to, or if it is more sensitive than, the label used in the comparison. If the user authorization passes, then the user is granted access to the column. Otherwise, the user is denied access.
   - In the Virtual Private Database policy definition, apply this function to the table that you want to protect. In the `DBMS_RLS.ADD_POLICY` procedure, use the sensitive column (`SEC_RELEVANT_COLS` parameter) and column masking (`SEC_RELEVANT_COLS_OPT` parameter) functionality to show or hide columns based on Oracle Label Security user authorizations.

For an example of how to accomplish this, visit the following Oracle Technology Network site:

`http://www.oracle.com/technetwork/database/focus-areas/security/ols-cs1-099558.html`

See Also:

Oracle Label Security Administrator's Guide for more information about the dominance functions

Oracle Virtual Private Database and Oracle Label Security Exceptions

Be aware of the security exceptions when you use Oracle Virtual Private Database and Oracle Label Security.

These security exceptions are as follows:

- **When you are exporting data, Oracle Virtual Private Database and Oracle Label Security policies are not enforced during a direct path export operation.** In a direct path export operation, Oracle Database reads data from disk into the buffer cache and transfers rows directly to the Export client.
- **You cannot apply Oracle Virtual Private Database policies and Oracle Label Security policies to objects in the SYS schema.** The `SYS` user and users making a DBA-privileged connection to the database (for example, `CONNECT/AS SYSDBA`) do not have Oracle Virtual Private Database or Oracle Label Security policies applied to their actions. The database user `SYS` is thus always exempt from Oracle Virtual Private Database or Oracle Label Security enforcement, regardless of the export mode, application, or utility used to extract data from the database.

  However, you can audit `SYSDBA` actions by enabling auditing upon installation and specifying that this audit trail be stored in a secure location in the operating system. You can also closely monitor the `SYS` user by using Oracle Database Vault.
- **Database users who were granted the EXEMPT ACCESS POLICY privilege, either directly or through a database role, are exempt from Oracle Virtual**

Private Database enforcements. The system privilege EXEMPT ACCESS POLICY allows a user to be exempted from all fine-grained access control policies on any SELECT or DML operation (INSERT, UPDATE, and DELETE). This provides ease of use for administrative activities, such as installation and import and export of the database, through a non-SYS schema.

However, the following policy enforcement options remain in effect even when EXEMPT ACCESS POLICY is granted:

– INSERT_CONTROL, UPDATE_CONTROL, DELETE_CONTROL, WRITE_CONTROL, LABEL_UPDATE, and LABEL_DEFAULT

– If the Oracle Label Security policy specifies the ALL_CONTROL option, then all enforcement controls are applied except READ_CONTROL and CHECK_CONTROL.

Because EXEMPT ACCESS POLICY negates the effect of fine-grained access control, you should only grant this privilege to users who have legitimate reasons for bypassing fine-grained access control enforcement. Do not grant this privilege using the WITH ADMIN OPTION. If you do, users could pass the EXEMPT ACCESS POLICY privilege to other users, and thus propagate the ability to bypass fine-grained access control.

Note:

- The EXEMPT ACCESS POLICY privilege does not affect the enforcement of object privileges such as SELECT, INSERT, UPDATE, and DELETE. These privileges are enforced even if a user was granted the EXEMPT ACCESS POLICY privilege.
- The SYS_CONTEXT values that Oracle Virtual Private Database uses are not propagated to secondary databases for failover.

See Also:

Oracle Database Utilities for more information about direct path export operations

Export of Data Using the EXPDP Utility access_method Parameter

Be aware if you try to export data from objects that have VPD policies defined on them.

If you try to use the Oracle Data Pump Export (EXPDP) utility with the access_method parameter set to direct_path to export data from a schema that contains an object that has a Virtual Private Database policy defined on it, then an ORA-31696 error message may appear and the export operation will fail.

The error message is as follows:

```
ORA-31696: unable to export/import TABLE_DATA:"schema.table" using client specified
DIRECT_PATH method
```

This problem only occurs when you perform a schema-level export as a user who has not been granted the `EXP_FULL_DATABASE` role. It does not occur during a full database export, which requires the `EXP_FULL_DATABASE` role. The `EXP_FULL_DATABASE` role includes the `EXEMPT ACCESS POLICY` system privilege, which bypasses Virtual Private Database policies.

To find the underlying problem, try the `EXPDP` invocation again, but do not set the `access_method` parameter to `direct_path`. Instead, use either `automatic` or `external_table`. The underlying problem could be a permissions problem, for example:

```
ORA-39181: Only partial table data may be exported due to fine grain access control
on "schema_name"."object_name"
```

See Also:

Oracle Database Utilities for more information about using Data Pump Export

User Models and Oracle Virtual Private Database

You can use Oracle Virtual Private Database in several types of user models.

These user models are as follows:

- **Application users who are also database users.** Oracle Database enables applications to enforce fine-grained access control for each user, regardless of whether that user is a database user or an application user unknown to the database. When application users are also database users, Oracle Virtual Private Database enforcement works as follows: users connect to the database, and then the application sets up application contexts for each session. (You can use the default `USERENV` application context namespace, which provides many parameters for retrieve different types of user session data.) As each session is initiated under a different user name, it can enforce different fine-grained access control conditions for each user.
- **Proxy authentication using OCI or JDBC/OCI.** Proxy authentication permits different fine-grained access control for each user, because each session (OCI or JDBC/OCI) is a distinct database session with its own application context.
- **Proxy authentication integrated with Enterprise User Security.** If you have integrated proxy authentication by using Enterprise User Security, you can retrieve user roles and other attributes from Oracle Internet Directory to enforce Oracle Virtual Private Database policies. (In addition, globally initialized application context can also be retrieved from the directory.)
- **Users connecting as One Big Application User.** Applications connecting to the database as a single user on behalf of all users can have fine-grained access control for each user. The user for that single session is often called *One Big Application User*. Within the context of that session, however, an application developer can create a global application context attribute to represent the individual application user (for example, `REALUSER`). Although all database sessions and audit records are created for One Big Application User, the attributes for each session can vary, depending on who the end user is. This model works best for applications with a limited number of users and no reuse of sessions. The scope of roles and database auditing is diminished because each session is created as the same database user.

- **Web-based applications.** Web-based applications typically have hundreds of users. Even when there are persistent connections to the database, supporting data retrieval for many user requests, these connections are not specific to particular Web-based users. Instead, Web-based applications typically set up and reuse connections, to provide scalability, rather than having different sessions for each user. For example, when Web users Jane and Ajit connect to a middle tier application, it may establish a single database session that it uses on behalf of both users. Typically, neither Jane nor Ajit is known to the database. The application is responsible for switching the user name on the connection, so that, at any given time, it is either Jane or Ajit using the session.

  Oracle Virtual Private Database helps with connection pooling by allowing multiple connections to access more than one global application context. This ability makes it unnecessary to establish a separate application context for each distinct user session.

Table 11-3 summarizes how Oracle Virtual Private Database applies to user models.

Table 11-3 Oracle Virtual Private Database in Different User Models

User Model Scenario	Individual Database Connection	Separate Application Context per User	Single Database Connection	Application Must Switch User Name
Application users are also database users	Yes	Yes	No	No
Proxy authentication using OCI or JDBC/OCI	Yes	Yes	No	No
Proxy authentication integrated with Enterprise User Security[1]	No	No	Yes	Yes
One Big Application User	No	No[2]	No	Yes[2]
Web-based applications	No	No	Yes	Yes

1 User roles and other attributes, including globally initialized application context, can be retrieved from Oracle Internet Directory to enforce Oracle Virtual Private Database.

2 Application developers can create a global application context attribute representing individual application users (for example, `REALUSER`), which can then be used for controlling each session attributes, or for auditing.

Oracle Virtual Private Database Data Dictionary Views

Oracle Database provides data dictionary views that list information about Oracle Virtual Private Database policies.

Table 11-4 lists Virtual Private Database-specific views

Table 11-4 Data Dictionary Views That Display Information about VPD Policies

View	Description
`ALL_POLICIES`	Describes all Oracle Virtual Private Database security policies for objects accessible to the current user.
`ALL_POLICY_ATTRIBUTES`	Describes all the application context namespaces, attributes, and Virtual Private Database policy associations where the logged in user is the owner of the VPD policy or the VPD policy belongs to `PUBLIC`.

Table 11-4 (Cont.) Data Dictionary Views That Display Information about VPD Policies

View	Description
`ALL_POLICY_CONTEXTS`	Describes the driving contexts defined for the synonyms, tables, and views accessible to the current user. A driving context is an application context used in an Oracle Virtual Private Database policy.
`ALL_POLICY_GROUPS`	Describes the Oracle Virtual Private Database policy groups defined for the synonyms, tables, and views accessible to the current user
`ALL_SEC_RELEVANT_COLS`	Describes the security relevant columns of the security policies for the tables and views accessible to the current user
`DBA_POLICIES`	Describes all Oracle Virtual Private Database security policies in the database.
`DBA_POLICY_ATTRIBUTES`	Describes all the application context namespaces, attributes, and Virtual Private Database policy associations for context-sensitive and shared context-sensitive Virtual Private Database policies
`DBA_POLICY_GROUPS`	Describes all policy groups in the database.
`DBA_POLICY_CONTEXTS`	Describes all driving contexts in the database. Its columns are the same as those in `ALL_POLICY_CONTEXTS`.
`DBA_SEC_RELEVANT_COLS`	Describes the security relevant columns of all security policies in the database
`UNIFIED_AUDIT_TRAIL`	Captures the VPD predicates in the `RLS_INFO` column, for unified auditing and fine-grained auditing
`USER_POLICIES`	Describes all Oracle Virtual Private Database security policies associated with objects owned by the current user. This view does not display the `OBJECT_OWNER` column.
`USER_POLICY_ATTRIBUTES`	Describes all the application context namespaces, attributes, and Virtual Private Database policy associations where the owner of the Virtual Private Database policy is the current user
`USER_POLICY_CONTEXTS`	Describes the driving contexts defined for the synonyms, tables, and views owned by the current user. Its columns (except for `OBJECT_OWNER`) are the same as those in `ALL_POLICY_CONTEXTS`.
`USER_SEC_RELEVANT_COLS`	Describes the security relevant columns of the security policies for the tables and views owned by the current user. Its columns (except for `OBJECT_OWNER`) are the same as those in `ALL_SEC_RELEVANT_COLS`.
`USER_POLICY_GROUPS`	Describes the policy groups defined for the synonyms, tables, and views owned by the current user. This view does not display the `OBJECT_OWNER` column.
`V$VPD_POLICY`	For the current PDB, displays all the fine-grained security policies and predicates associated with the cursors currently in the library cache. This view is useful for finding the policies that were applied to a SQL statement.

Tip:

In addition to these views, check the database trace file if you find errors in application that use Virtual Private Database policies. The `USER_DUMP_DEST` initialization parameter specifies the current location of the trace files. You can find the value of this parameter by issuing `SHOW PARAMETER USER_DUMP_DEST` in SQL*Plus.

See Also:

- *Oracle Database Reference* for more information about these views
- *Oracle Database SQL Tuning Guide* for more information about trace files

12

Using Transparent Sensitive Data Protection

Transparent sensitive data protection enables you to find all table columns in a database that hold sensitive data.

About Transparent Sensitive Data Protection

Transparent sensitive data protection is a way to find and classify table columns that hold sensitive information.

This feature enables you to quickly find the table columns in a database that hold sensitive data, classify this data, and then create a policy that protects this data as a whole for a given class. Examples of this type of sensitive data are credit card numbers or Social Security numbers.

The TSDP policy then protects the sensitive data in these table columns by using either Oracle Data Redaction or Oracle Virtual Private Database settings. The TSDP policy applies at the column level of the table that you want to protect, targeting a specific column data type, such as all `NUMBER` data types of columns that contain credit card information. You can create a uniform TSDP policy for all of the data that you classify, and then modify this policy as necessary, as compliance regulations change. Optionally, you can export the TSDP policies for use in other databases.

The benefits of TSDP policies are enormous: You easily can create and apply TSDP policies throughout a large organization with numerous databases. This helps auditors greatly by enabling them to estimate the protection for the data that the TSDP policies target. TSDP is particularly useful for government environments, in which you may have a lot of data with similar security restrictions and you must apply a policy to all of this data consistently. The policy could be to redact it, encrypt it, control access to it, audit access to it, and mask it in the audit trail. Without TSDP, you would have to configure every redaction policy, column-level encryption configuration, and Virtual Private Database policy column by column.

General Steps for Using Transparent Sensitive Data Protection

To use TSDP with Oracle Data Redaction, you must follow a set of general steps.

1. Create a sensitive type to classify the types of columns that you want to protect.

   For example, you can create a sensitive type for classify all Social Security numbers or credit card numbers. To create the sensitive type, either use the `DBMS_TSDP_MANAGE.ADD_SENSITIVE_TYPE` PL/SQL procedure or use an Enterprise Manager Cloud Control Application Data Model. To add multiple sensitive types in one operation from an Application Data Model, you can use the `DBMS_TSDP_MANAGE.IMPORT_SENSITIVE_TYPES` procedure.

2. Identify a list of sensitive columns that are associated with the sensitive types.

   To determine and generate this list, you can use either of the following methods:

   - The `DBMS_TSDP_MANAGE.ADD_SENSITIVE_COLUMN` procedure individually identifies sensitive columns.
   - An Oracle Enterprise Manager Cloud Control Application Data Model enables you to identify a group of sensitive columns. It then prepares this list of sensitive columns in XML format, which you then import into your database.

3. If you used an Application Data Model for Step 2, then import the list of sensitive columns from the Application Data Model into your database by using the `DBMS_TSDP_MANAGE.IMPORT_DISCOVERY_RESULT` procedure.
4. Create the TSDP policy by using the `DBMS_TSDP_PROTECT.ADD_POLICY` procedure within an anonymous block that defines the Data Redaction or Virtual Private Database settings that you want to use.
5. Associate the TSDP policy with one or more sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.
6. Enable the TSDP policy protections by using the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE`, `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_COLUMN`, or the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_TYPE` procedure.
7. Optionally, export the TSDP policy to other databases by using Oracle Data Pump to perform a full database export. (You cannot individually export TSDP policies.)

Use Cases for Transparent Sensitive Data Protection Policies

Transparent sensitive data protection has several benefits.

These benefits are as follows:

- **You configure the sensitive data protection once, and then deploy this protection as necessary.** You can configure transparent sensitive data protection policies to designate how a class of data (for example, credit card columns) must be protected without actually having to specify the target data. In other words, when you create the transparent sensitive data protection policy, you do not need to include references to the actual target columns that you want to protect. The transparent sensitive data protection policy finds these target columns based on a list of sensitive columns in the database and the policy's associations with the specified sensitive types. This can be useful when you add more sensitive data to your databases after you have created the transparent sensitive data protection policies. After you create the policy, you can enable protection for the sensitive data in a single step (for example, enable protection based on the entire source database). The sensitive type of the new data and the sensitive type and policy associations determine how the sensitive data is protected. In this way, as new sensitive data is added, you do not need to configure its protection, as long as it meets the current transparent sensitive data protection policy's requirements.
- **You can manage protection of multiple sensitive columns.** You can enable or disable protection for multiple sensitive columns based on a suitable attribute (such as the source database of the identification, the sensitive type itself, or a specific schema, table, or column). This granularity provides a high level of control

over data security. The design of this feature enables you to manage data security based on specific compliance needs for large data sets that fall under the purview of these compliance regulations. You can configure data security based on a specific category rather than for each individual column. For example, you can configure protection for credit card numbers or Social Security numbers, but you do not need to configure protection for each and every column in the database that contains this data.

- **You can protect the sensitive columns identified using the Oracle Enterprise Manager Cloud Control Application Data Modeling (ADM) feature.** You can use the Cloud Control ADM feature to create sensitive types and discover a list of sensitive columns. Then you can import this list of sensitive columns and their corresponding sensitive types into your database. From there, you can create and manage transparent sensitive data protection policies using this information.

Privileges Required for Using Transparent Sensitive Data Protection

To use transparent sensitive data protection, you must have the `EXECUTE` privilege for several PL/SQL packages.

These privileges are as follows:

- `DBMS_TSDP_MANAGE`, which enables you to import and manage sensitive columns and sensitive types into your database. The procedures in this package execute with invoker's rights. Typically, an application database administrator will be granted privileges for this package.
- `DBMS_TSDP_PROTECT`, which you use to create the TSDP policy. The procedures in this package execute with invoker's rights. Typically, a security database administrator will be granted privileges for this package.
- `DBMS_REDACT`, if you plan to create Data Redaction policies. Typically, a security database administrator will be granted privileges for this package.
- `DBMS_RLS`, if you plan to incorporate Oracle Virtual Private Database functionality into your TSDP policies. Typically, a security database administrator will be granted privileges for this package.

For better separation of duty, these packages are designed so that either an application database administrator has control over one area of the TSDP policy creation (as in the case of the `DBMS_TSDP_MANAGE` package) or a security database administrator (for the `DBMS_TSDP_PROTECT`, `DBMS_REDACT`, and `DBMS_RLS` packages).

How a Multitenant Environment Affects Transparent Sensitive Data Protection

In a multitenant environment, you can apply Transparent Sensitive Data Protection policies to the current PDB or current application PDB only.

If you are using Enterprise Manager Cloud Control Application Data Model, then you can find sensitive columns that belong to both local and common application objects (that is, common objects that are visible and accessible in the current PDB) inside the PDB. This enables you to use a TSDP policy to protect both local objects to the PDB and common objects that are accessible from the PDB.

In an application root:

- For application containers in general:
  - When you create scripts for application install, upgrade, patch, or uninstall operations, you can include SQL statements within the `ALTER PLUGGABLE DATABASE` *`app_name`* `BEGIN INSTALL` and `ALTER PLUGGABLE DATABASE` *`app_name`* `END INSTALL` blocks to perform various operations. If you include TSDP statements within these blocks, then the TSDP statements will fail. You can, however, include TSDP statements outside these blocks in the script.
- In the application root:
  - You can perform TSDP operations in both application common objects and application root local objects.
  - A TSDP policy that is defined in the application root container behaves as if it is a local policy to the application root. That is, the policy is effective only in the application root container.

In an application PDB:

- The security policies that protect an application PDB apply to TSDP operations that are performed on local application objects.
- The security policies that protect an application PDB apply to TSDP operations that are performed on application common objects that are accessed from the PDB. However, access to the application common object outside the application PDB is not governed by the security policy that protects the application PDB.

You can find a listing of TSDP policies and the security features that are associated with them by querying the `DBA_TSDP_POLICY_FEATURE` data dictionary views. To find all PDBs, query the `DBA_PDBS` view.

Creating Transparent Sensitive Data Protection Policies

You must create a sensitive type, find the sensitive columns to be protected, and then import these columns from ADM into your database.

Step 1: Create a Sensitive Type

The sensitive type is a class of data that you designate as sensitive.

For example, you can create a `credit_card_type` sensitive type for all credit card numbers.

- To create a sensitive type, either create it from an Enterprise Manager Cloud Control Application Data Model or use the `DBMS_TSDP_MANAGE.ADD_SENSITIVE_TYPE` PL/SQL procedure.

  To drop a sensitive type, you can use the `DBMS_TSDP_MANAGE.DROP_SENSITIVE_TYPE` procedure.

For example, to create the sensitive type `credit_card_num_type`:

```
BEGIN
 DBMS_TSDP_MANAGE.ADD_SENSITIVE_TYPE (
  sensitive_type  => 'credit_card_num_type',
  user_comment    => 'Type for credit card columns using a number data type');
```

```
END;
/
```

In this example:

- `sensitive_type`: Create a name that describes the sensitive type that you want to capture. This value is case sensitive, so when you reference it later on, ensure that you use the case in which you created it. You can find existing sensitive types by querying the `DBA_SENSITIVE_COLUMN_TYPES` data dictionary view.
- `user_comment`: Optionally, enter a description for the sensitive type.

See Also:

- *Oracle Database Testing Guide* for detailed information about Application Data Models
- *Oracle Database PL/SQL Packages and Types Reference* information about the `DBMS_TSDP_MANAGE.ADD_SENSITIVE_TYPE` PL/SQL procedure

Step 2: Identify the Sensitive Columns to Protect

After you define a sensitive column, you are ready to identify the columns to protect.

To identify the columns to protect, based on the sensitive type that you defined, you either can use an Enterprise Manager Cloud Control Application Data Model to identify these columns, or you can use the `DBMS_TSDP_MANAGE.ADD_SENSITIVE_COLUMN` procedure.

To remove the column from the list of sensitive columns for the database, you can use the `DBMS_TSDP_MANAGE.DROP_SENSITIVE_COLUMN` procedure.

1. Find the sensitive type that you want to use.

   For example:

   ```
   SELECT NAME FROM DBA_SENSITIVE_COLUMN_TYPES;

   NAME
   ---------------------
   credit_card_num_type
   ```

2. Execute the `DBMS_TSDP_MANAGE.ADD_SENSITIVE_COLUMN` procedure to associate the sensitive type with a table column. Ensure that you enter the `sensitive_type` parameter using the case in which you used to create the sensitive type.

   For example:

   ```
   BEGIN
    DBMS_TSDP_MANAGE.ADD_SENSITIVE_COLUMN(
    schema_name        => 'OE',
    table_name         => 'CUST_CC',
    column_name        => 'CREDIT_CARD',
    sensitive_type     => 'credit_card_num_type',
    user_comment       => 'Sensitive column addition of credit_card_num_type');
   END;
   /
   ```

Step 3: Import the Sensitive Columns List from ADM into Your Database

Next, you are ready to import the sensitive columns list from ADM into your database.

- If you had used an Application Data Model to create the list of sensitive columns, then import this list into your database by running the `DBMS_TSDP_MANAGE.IMPORT_DISCOVERY_RESULT` procedure.

  If you had used the `DBMS_TSDP_MANAGE.ADD_SENSITIVE_COLUMN` procedure to identify these columns, then you can bypass this step.

For example, to import the Cloud Control Application Data Model into the current database:

```
BEGIN
 DBMS_TSDP_MANAGE.IMPORT_DISCOVERY_RESULT (
 discovery_result      => xml_adm_result,
 discovery_source      => 'ADM_Demo');
END;
/
```

In this example:

- `discovery_result` refers to the list of sensitive columns and their associated sensitive types. This list is in XML format.
- `discover_source` refers to the name of the Application Data Model that contains the list of sensitive columns referred by the `discovery_result` setting. You can find a list of the Application Data Models from the Data Discovery and Modeling page in Enterprise Manager Cloud Control. (To access this page, from the **Enterprise** menu, select **Quality Management**, and then **Data Discovery and Modeling**. You can find a list of the sensitive columns and their associated types in the **Sensitive Columns** tab.)

Step 4: Create the Transparent Sensitive Data Protection Policy

After you have created the list of sensitive columns and imported this list into your database, you can create the transparent sensitive data protection policy.

About Creating the Transparent Sensitive Data Protection Policy

The `DBMS_TSDP_PROTECT.ADD_POLICY` procedure creates the transparent sensitive data protection policy.

After you have identified the sensitive columns, and if you had used an Application Data Model to create the list of sensitive columns, and imported this list into your database, you are ready to create the transparent sensitive data protection policy. To create the transparent sensitive data protection policy, you must configure it for the Virtual Private Database or Oracle Data Redaction settings that you want to use, and then apply these settings to a transparent sensitive data protection policy defined by `DBMS_TSDP_PROTECT.ADD_POLICY`.

You can create the policy by defining an anonymous block that has the following components:

- If you are using Oracle Data Redaction for your policy, a specification of the type of Data Redaction that you want to use, such as partial Data Redaction
- If you are using Oracle Virtual Private Database for your policy, a specification of the VPD settings that you want to use
- Conditions to test when the policy is enabled. For example, the data type of the column which should be satisfied before the policy can be enabled.
- A named transparent sensitive data protection policy to tie these components together, by using the `DBMS_TSDP_PROTECT.ADD_POLICY` procedure

After you create the sensitive type, it resides in the `SYS` schema.

Creating the Transparent Sensitive Data Protection Policy

You can create a transparent sensitive data protection policy that uses a partial number data type-based Data Redaction policy.

Example 12-1 shows how to create this type of policy.

- To create the policy, use the `DBMS_TSDP_PROTECT.ADD_POLICY` procedure, as shown in Example 12-1.

Example 12-1 Creating a Transparent Sensitive Data Protection Policy

```
DECLARE
  redact_feature_options DBMS_TSDP_PROTECT.FEATURE_OPTIONS;
  policy_conditions DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
BEGIN
  redact_feature_options ('expression') :=
   'SYS_CONTEXT(''USERENV'',''SESSION_USER'') =''APPUSER''';
  redact_feature_options ('function_type') := 'DBMS_REDACT.PARTIAL';
  redact_feature_options ('function_parameters') := '0,1,6';
  policy_conditions(DBMS_TSDP_PROTECT.DATATYPE) := 'NUMBER';
  policy_conditions(DBMS_TSDP_PROTECT.LENGTH) := '16';
 DBMS_TSDP_PROTECT.ADD_POLICY ('redact_partial_cc',
  DBMS_TSDP_PROTECT.REDACT,redact_feature_options,
   policy_conditions);
END;
/
```

In this example:

- `redact_feature_options DBMS_TSDP_PROTECT.FEATURE_OPTIONS` creates the variable `redact_feature_options`, which uses the `FEATURE_OPTIONS` data type. See Setting the Oracle Data Redaction or Virtual Private Database Feature Options for more information.
- `policy_conditions DBMS_TSDP_PROTECT.POLICY_CONDITIONS` creates the variable `policy_conditions`, which uses the `POLICY_CONDITIONS` data type. See Setting Conditions for the Transparent Sensitive Data Protection Policy for more information.
- `redact_feature_options` lines (3) write the Data Redaction policy settings to the `redact_feature_option` variable. This example applies the Data Redaction policy to the user `APPUSER` and defines the policy as a partial data redaction for number data types. See *Oracle Database Advanced Security Guide* for information about how the `function_parameters` parameter works for this case.

- `policy_conditions` lines (2) write the TSDP policy conditions to the `policy_conditions` variable (that is, the data type and length) for the protected `NUMBER` data type column.
- `DBMS_TSDP_PROTECT.ADD_POLICY` executes the `DBMS_TSDP_PROTECT.ADD_POLICY` procedure, which creates the `redact_partial_cc` TSDP policy. See Specifying the DBMS_TSDP_PROTECT.ADD_POLICY Procedure for more information.

If you want to see an example of a similar policy for VPD, see Step 4: Create and Enable a Transparent Sensitive Data Protection Policy.

Setting the Oracle Data Redaction or Virtual Private Database Feature Options

The TSDP feature options describe the Oracle Data Redaction or Virtual Private Database settings to use for the transparent sensitive data protection policy.

- For Data Redaction, define the feature options by using the name `redact_feature_options` variable and for the type, you must use the type `DBMS_TSDP_PROTECT.FEATURE_OPTIONS`, which is an associative array of the data type `VARCHAR2(TSDP_PARAM_MAX)`. Initialize these options with the parameter-value pairs that correspond with the `DBMS_REDACT.ADD_POLICY` parameters.

For example, to specify a TSDP policy that uses partial Data Redaction, Example 12-1 shows the following parameter-value pair:

```
redact_feature_options ('function_type')       := 'DBMS_REDACT.PARTIAL';
```

For a partial Data Redaction policy that uses a number data type for the protected column, Example 12-1 specifies the following additional parameter-value pairs:

```
redact_feature_options ('expression')          := 'expression';
redact_feature_options ('function_parameters') := 'values';
```

Similarly, for Virtual Private Database, you use the `vpd_feature_options` variable to define the VPD feature options. For example:

```
vpd_feature_options ('statement_types')   := 'SELECT, INSERT, UPDATE, DELETE';
```

See Also:

- *Oracle Database Advanced Security Guide* for more information about Data Redaction policy creation parameters
- DBMS_RLS.ADD_POLICY Parameters That Are Used for TSDP Policies for more information about available VPD parameters

Setting Conditions for the Transparent Sensitive Data Protection Policy

Optionally, you can specify conditions for the transparent sensitive data protection policy.

However, if you choose to omit conditions, you still must include the following line in the `DECLARE` variables. (In this case, the default value for `policy_conditions` is an empty associative array.)

```
policy_conditions SYS.DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
```

- To define the conditions, use the name `policy_conditions` for the variable and for the type, use type `DBMS_TSDP_PROTECT.POLICY_CONDITIONS`, which is an associative array of the data type `VARCHAR2(TSDP_PARAM_MAX)`. Ensure that no two conditions are satisfied by a single target sensitive column. The target column's properties should satisfy all the condition properties for the corresponding `DBMS_TSDP_PROTECT.FEATURE_OPTIONS` settings to be applied on the column

Example 12-1 shows the policy conditions as follows:

```
policy_conditions(DBMS_TSDP_PROTECT.DATATYPE) := 'NUMBER';
policy_conditions(DBMS_TSDP_PROTECT.LENGTH)   := '16';
```

Optionally, you can specify one or more of the following keys for the `POLICY_CONDITIONS` settings:

- `DBMS_TSDP_PROTECT.DATATYPE` enables you to specify a data type.
- `DBMS_TSDP_PROTECT.LENGTH` enables you to specify a data type length for the `DBMS_TSDP_PROTECT.DATATYPE` key.
- `DBMS_TSDP_PROTECT.PARENT_SCHEMA` enables you to restrict the policy to a specific schema. If you omit this setting, then the policy applies to all schemas in the database.
- `DBMS_TSDP_PROTECT.PARENT_TABLE` enables you to restrict the policy to a table specified by the `DBMS_TSDP_PROTECT.PARENT_SCHEMA` key. If you omit this setting, then the policy applies to all tables within the specified schema.

Specifying the DBMS_TSDP_PROTECT.ADD_POLICY Procedure

The `DBMS_TSDP_PROTECT.ADD_POLICY` procedure names the TSDP policy and executes the `FEATURE_OPTIONS` and `POLICY_CONDITIONS` settings.

In the policy, the `redact_feature_options` and the `policy_conditions` settings work together: When the policy is enabled (using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION*` procedures) on the target object, then the `redact_feature_options` settings apply only if the corresponding `policy_condition` settings are satisfied. You must enter the following parameters:

- To specify a procedure that names the transparent sensitive data protection policy and executes the necessary settings, include the following parameters:
  – `policy_name` creates a name for the TSDP policy. The name that you enter is stored in the database using the case sensitivity that you used when you created it. For example, if you had entered `redact_partial_cc`, then the database stores it as `redact_partial_cc`, not `redact_partial_cc`.
  – `security_feature` refers to the security feature the TSDP policy will use. Enter `DBMS_TSDP_PROTECT.REDACT` to specify Oracle Data Redaction.
  – `policy_enable_options` refers to the variable that you defined for the `DBMS_TSDP_PROTECT.FEATURE_OPTIONS` type.
  – `policy_apply_condition` refers to the variable that you defined for the `DBMS_TSDP_PROTECT.POLICY_CONDITIONS` type.

Example 12-1 shows the policy set as follows:

```
DBMS_TSDP_PROTECT.ADD_POLICY('redact_partial_cc', DBMS_TSDP_PROTECT.REDACT,
redact_feature_options, policy_conditions);
```

Step 5: Associate the Policy with a Sensitive Type

The `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure associates a TSDP policy with a sensitive type.

1. Find the sensitive type that you want to use.

   For example, to find a list of all sensitive types:

   ```
   SELECT NAME FROM DBA_SENSITIVE_COLUMN_TYPES ORDER BY NAME;

   NAME
   --------------------
   credit_card_num_type
   ```

2. Run the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure to associate the policy with a sensitive column type.

   For example:

   ```
   BEGIN
    DBMS_TSDP_PROTECT.ASSOCIATE_POLICY(
    policy_name        => 'redact_partial_cc',
    sensitive_type     => 'credit_card_num_type',
    associate          => true);
   END;
   /
   ```

   The following query shows that the `credit_card_num_type` is now associated with the `redact_partial_cc` policy.

   ```
   SELECT POLICY_NAME, SENSITIVE_TYPE FROM DBA_TSDP_POLICY_TYPE ORDER BY
   SENSITIVE_TYPE;

   POLICY_NAME       SENSITIVE_TYPE
   ----------------- --------------------
   redact_partial_cc  credit_card_num_type
   ```

Step 6: Enable the Transparent Sensitive Data Protection Policy

You can enable the TSDP policy for the current database in a protected source, a specific table column, or a specific column type.

Enabling Protection for the Current Database in a Protected Source

You can enable transparent sensitive data protection for the current database in a protected source.

If you must disable the protection, then you can run the `DBMS_TSDP_PROTECT.DISABLE_PROTECTION_SOURCE` procedure.

- Run the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE` procedure to enable this type of protection.

For example, to enable transparent sensitive data protection policies for the `orders_db` database.

```
BEGIN
 DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE(
  discovery_source       => 'orders_db');
```

```
END;
/
```

Enabling Protection for a Specific Table Column

You can enable transparent sensitive data protection for a specific column in a table.

Remember that you can enable only one policy per table. If you must disable the protection, then you can run the `DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN` procedure.

- Run the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_COLUMN` procedure to enable this type of protection.

For example, to enable the transparent sensitive data protection policy `redact_partial_cc` for a specific table column:

```
BEGIN
 DBMS_TSDP_PROTECT.ENABLE_PROTECTION_COLUMN(
  schema_name          => 'OE',
  table_name           => 'CUST_CC',
  column_name          => 'CREDIT_CARD',
  policy               => 'redact_partial_cc');
END;
/
```

If an `ORA-45622: warnings generated during policy enforcement` error appears, then check the configuration of the policy. In this example, the `redact_partial_cc` policy is enabled on a column if this column is of the `NUMBER` data type and has a length of `16`. Even though the `OE.CUST_CC.CREDIT_CARD` column is associated with the `redact_partial_cc` policy, the policy is not enabled if this column fails to satisfy the conditions (data type and length).

Enabling Protection for a Specific Column Type

You can enable transparent sensitive data protection for a specific column type, such as all columns that use the `VARCHAR2` data type.

If you must disable the protection, then you can run the `DBMS_TSDP_PROTECT.DISABLE_PROTECTION_TYPE` procedure.

- Run the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_TYPE` procedure to enable this type of protection.

For example, to enable transparent sensitive data protection for all columns that use the `credit_card_num_type` sensitive type:

```
BEGIN
 DBMS_TSDP_PROTECT.ENABLE_PROTECTION_TYPE(
  sensitive_type          => 'credit_card_num_type');
END;
/
```

Step 7: Optionally, Export the Policy to Other Databases

You can export or import the policy to or from another database.

- To export or import the TSDP policy to or from another database, use Oracle Data Pump to perform a full export or import of the database that contains the policy.

Remember that the export and import operations apply to the entire database, not just the transparent sensitive data protection policy.

See Also:

- *Oracle Database Utilities* for information about using Oracle Data Pump
- *Oracle Database Vault Administrator's Guide* for information about using Oracle Data Pump in an Oracle Database Vault environment

Altering Transparent Sensitive Data Protection Policies

The `DBMS_TSDP_PROTECT.ALTER_POLICY` procedure can alter a TSDP policy.

When you alter a transparent data protection policy, you must define how the Data Redaction settings must change, and then apply these changes to the transparent sensitive data protection policy itself.

You can find a list of existing policies and their protection definitions by querying the `DBA_TSDP_POLICY_FEATURE` data dictionary view.

- To alter a transparent sensitive data protection policy, use the `DBMS_TSDP_PROTECT.ALTER_POLICY` procedure.

For example, to alter an existing transparent sensitive data protection policy:

```
DECLARE
  redact_feature_options SYS.DBMS_TSDP_PROTECT.FEATURE_OPTIONS;
  policy_conditions SYS.DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
  BEGIN
  redact_feature_options ('expression') :=
   'SYS_CONTEXT(''USERENV'',''SESSION_ USER'') =''APPUSER''';
  redact_feature_options ('function_type') := 'DBMS_REDACT.PARTIAL';
  redact_feature_options ('function_parameters') := '9,1,6';
  policy_conditions(DBMS_TSDP_PROTECT.DATATYPE) := 'NUMBER';
  policy_conditions(DBMS_TSDP_PROTECT.LENGTH) := '22';
 DBMS_TSDP_PROTECT.ALTER_POLICY ('redact_partial_cc',
  redact_feature_options, policy_conditions);
END;
/
```

In this example:

- `redact_feature_options SYS.DBMS_TSDP_PROTECT.FEATURE_OPTIONS` creates the variable `redact_feature_options`, which uses the `FEATURE_OPTIONS` data type.
- `policy_conditions SYS.DBMS_TSDP_PROTECT.POLICY_CONDITIONS` creates the variable `policy_conditions`, which uses the `POLICY_CONDITIONS` data type.
- `redact_feature_options ... redact_feature_options`writes the Data Redaction policy settings to the `redact_feature_option` variable. This example applies the Data Redaction policy to the user `APPUSER`, defines the policy as a partial data redaction for number data types. See *Oracle Database Advanced Security Guide* for information about how the `function_parameters` parameter works for this case.

- `policy_conditions ... policy_conditions` writes the TSDP policy conditions to the `policy_conditions` variable (that is, the data type and length) for the protected `NUMBER` data type column.
- `DBMS_TSDP_PROTECT.ALTER_POLICY ...` executes the `DBMS_TSDP_PROTECT.ALTER_POLICY` procedure, which alters the `redact_partial_cc` TSDP policy to use the definitions set in the `redact_feature_options` and `policy_conditions` variables.

Disabling Transparent Sensitive Data Protection Policies

The `DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN` procedure disables one or all TSDP policies.

1. Query the `DBA_TSDP_POLICY_PROTECTION` data dictionary view to find the protected columns and their associated transparent sensitive data protection policies.

   For example:

   ```
   SELECT COLUMN_NAME, TSDP_POLICY FROM DBA_TSDP_POLICY_PROTECTION WHERE TABLE_NAME
   = 'CUST_CC';

   COLUMN_NAME   TSDP_POLICY
   ------------  ------------------
   CREDIT_CARD   redact_partial_cc
   ```

2. Run the `DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN` procedure.

   For example, to disable the `redact_partial_cc` policy on the `CREDIT_CARD` column of the `CUST_CC` table:

   ```
   BEGIN
    DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN(
     schema_name          => 'OE',
     table_name           => 'CUST_CC',
     column_name          => 'CREDIT_CARD',
     policy               => 'redact_partial_cc');
   END;
   /
   ```

   You can use the `%` wildcard in this procedure to specify multiple items. For example, to disable protection for any columns that begin with `CREDIT`, you could enter the following:

   ```
   BEGIN
    DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN(
     schema_name          => 'OE',
     table_name           => 'CUST_CC',
     column_name          => 'CREDIT%',
     policy               => 'redact_partial_cc');
   END;
   /
   ```

   To disable all transparent sensitive data protection policies for a table, you can omit the `policy` parameter. For example:

   ```
   BEGIN
    DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN(
     schema_name          => 'OE',
     table_name           => 'CUST_CC',
     column_name          => '%');
   ```

```
END;
/
```

Dropping Transparent Sensitive Data Protection Policies

You can drop an entire TSDP policy or a condition-enable-options combination from the policy.

If the policy only has one condition-enable-options combination, then Oracle Database drops the entire policy. You do not need to disable a policy before dropping it, but you do need to drop its associated sensitive column first, then its sensitive type.

1. Query the `POLICY_NAME` column of the `DBA_TSDP_POLICY_FEATURE` data dictionary view to find the policy that you want to drop.

   ```
   SELECT POLICY_NAME FROM DBA_TSDP_POLICY_FEATURE;

   POLICY_NAME
   -----------------
   redact_partial_cc
   ```

   Remember that you must be granted the `SELECT_CATALOG_ROLE` role to query the transparent sensitive data protection data dictionary views.

2. Find the sensitive column that is associated with this policy.

   For example:

   ```
   SELECT COLUMN_NAME FROM DBA_TSDP_POLICY_PROTECTION WHERE TSDP_POLICY =
   'redact_partial_cc';

   COLUMN_NAME
   -----------------
   CREDIT_CARD
   ```

3. Drop this sensitive column.

   For example:

   ```
   BEGIN
    DBMS_TSDP_MANAGE.DROP_SENSITIVE_COLUMN (
      schema_name        => 'OE',
      table_name         => 'CUST_CC',
      column_name        => 'CREDIT_CARD');
   END;
   /
   ```

4. Find the sensitive type that is associated with this policy.

   For example:

   ```
   SELECT SENSITIVE_TYPE FROM DBA_TSDP_POLICY_TYPE WHERE POLICY_NAME =
   'redact_partial_cc';

   SENSITIVE_TYPE
   --------------------
   credit_card_num_type
   ```

5. Drop this sensitive type.

   For example:

   ```
   BEGIN
    DBMS_TSDP_MANAGE.DROP_SENSITIVE_TYPE (   sensitive_type     =>
   ```

```
'credit_card_num_type');END;
/
```

6. Run the `DBMS_TSDP_PROTECT.DROP_POLICY` procedure to drop the policy.

   For example, to completely drop the policy:

```
BEGIN
 DBMS_TSDP_PROTECT.DROP_POLICY(
   policy_name     => 'redact_partial_cc');
END;
/
```

   To drop the default condition-enable options combination from the policy:

```
DECLARE
   policy_conditions DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
BEGIN
   DBMS_TSDP_PROTECT.DROP_POLICY ('redact_partial_cc', policy_conditions);
END;
/
```

   To drop the default condition-enable options combination from the policy based on a specific condition:

```
DECLARE
   policy_conditions DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
BEGIN
   policy_conditions (DBMS_TSDP_PROTECT.DATATYPE) := 'NUMBER';
   DBMS_TSDP_PROTECT.DROP_POLICY ('redact_partial_cc', policy_conditions);
END;
/
```

Using the Predefined REDACT_AUDIT Policy to Mask Bind Values

The predefined `REDACT_AUDIT` policy masks bind values, which can appear in trace files when an event is set.

About the REDACT_AUDIT Policy

The predefined `REDACT_AUDIT` transparent sensitive data protection policy masks bind values.

The bind values of the bind variables that are used in SQL statements can appear in audit records when auditing is configured. Similarly, bind values can appear in trace files when the appropriate event is set. Bind values can also appear when you query the `V$SQL_BIND_DATA` dynamic view.

The `REDACT_AUDIT` transparent sensitive data protection policy displays the data as an asterisk (`*`) in audit records, trace files, and in `V$SQL_BIND_DATA` view queries. By default the `REDACT_AUDIT` policy is associated with every sensitive type in the database. When you identify a column as sensitive, by default, the `REDACT_AUDIT` policy is enabled for it.

You can disable and enable the `REDACT_AUDIT` policy, but you cannot alter or drop it.

Variables Associated with Sensitive Columns

Bind variables affect the use of sensitive columns with conditions, `SELECT` items, and `INSERT` or `UPDATE` operations.

About Variables Associated with Sensitive Columns

You can associate variables with sensitive columns in TSDP policies.

A bind variable can be considered to be sensitive or "associated" with a sensitive column if the bind variable occurs in the same comparison condition as a sensitive column, if it occurs in a `SELECT` statement alongside a sensitive column, or if it occurs in an `INSERT` or `UPDATE` operation that involves a sensitive column.

Bind Variables and Sensitive Columns in the Expressions of Conditions

You can include sensitive columns in SQL queries that have `WHERE` clauses.

A SQL query that contains a `WHERE` clause can include sensitive columns and bind variables for use with comparison operators such as `=`, `IS`, `IS NOT`, `LIKE`, `BETWEEN`, and `IN`, as well as in subqueries.

In the following comparison query, the bind value in `VAR1` is masked because `VAR1` and the sensitive column `SALARY` appear in the expression that is compared using the comparison condition `>`.

```
SELECT EMPLOYEE_ID FROM HR.EMPLOYEES WHERE SALARY > :VAR1;
```

In the next query, the bind values in `VAR1` and `VAR2` are masked because `VAR1`, `VAR2`, and the sensitive column `SALARY` appear in the expression that uses the comparison equality condition `=`.

```
SELECT EMPLOYEE_ID FROM HR.EMPLOYEES WHERE SALARY + :VAR1 = TO_NUMBER(:VAR2,
'9G999D99');
```

For floating point conditions, the sensitive column and the bind variable appear in the expression that is evaluated. In the following example, the bind value in `VAR1` is masked because `VAR1` and the sensitive column `SALARY` appear in the expression for the `IS NOT NAN` condition.

```
SELECT COUNT( ) FROM HR.EMPLOYEES WHERE (SALARY * :VAR1) IS NOT NAN;
```

In pattern matching conditions, the sensitive column and the bind variable appear as arguments. In the following example, the bind value in `VAR1` is masked because `VAR1` and the sensitive column `LAST_NAME` are the arguments for the `LIKE` condition.

```
SELECT LAST_NAME FROM HR.EMPLOYEES WHERE LAST_NAME LIKE :VAR1;
```

For `BETWEEN` conditions, the sensitive column and the bind variable appear in the expressions that are arguments. In the following example, bind values in `VAR1` and `VAR2` are masked because `VAR1`, `VAR2`, and `SALARY` appear in expressions that are arguments to the `BETWEEN` condition.

```
SELECT EMPLOYEE_ID FROM HR.EMPLOYEES WHERE SALARY BETWEEN :VAR1 AND :VAR2;
```

In the next example, the sensitive column and the bind variable are the arguments of the `IN` condition. Here, the bind values in `VAR1` and `VAR2` are masked because `VAR1`, `VAR2`, and the sensitive column `SALARY` appear as arguments to the `IN` condition.

```
SELECT COUNT( ) FROM HR.EMPLOYEES WHERE SALARY IN ( :VAR1, :VAR2);
```

When a condition has a nested subquery as an argument, the bind variables and sensitive columns that appear in the nested subquery are not considered to be associated with the condition. In the following query, the sensitive column `SALARY` and the subquery are expressions for the greater-than condition `>`.

```
SELECT EMPLOYEE_ID FROM HR.EMPLOYEES WHERE SALARY > (SELECT SALARY FROM HR.EMPLOYEES
WHERE MANAGER_ID = :VAR1);
```

However, variable `VAR1` is associated with column `MANAGER_ID` as variable `VAR1` and `MANAGER_ID` appears in expressions being compared using the condition `=`. Because `MANAGER_ID` is not a sensitive column, variable `VAR1` is not considered sensitive. The variable `VAR1` is not considered to be associated with the sensitive column `SALARY`.

In the case of the logical conditions, model conditions, multiset conditions, XML conditions, compound conditions, `IS OF` type conditions, and `EXISTS` conditions, there can be no cases where a bind variable and a sensitive column are associated with each other. This is due to the structure or the nature of these conditions.

A Bind Variable and a Sensitive Column Appearing in the Same SELECT Item

If a column in a `SELECT` item is sensitive, then all the binds in the `SELECT` item are considered sensitive.

For example, assume that `HR.EMPLOYEES.SALARY` and `HR.EMPLOYEES.COMMISSION_PCT` are sensitive columns. In the following query, the bind variable `VAR1` is considered sensitive because it appears in the same `SELECT` item as the sensitive column `SALARY`, so its bind value is masked.

```
SELECT (SALARY * :VAR1) AS BONUS AS FROM HR.EMPLOYEES WHERE EMPLOYEE_ID = :VAR2;
```

In the next example, the bind variable `VAR1` is considered sensitive because it appears in the same `SELECT` item as `SALARY`. `VAR2` is considered sensitive because it appears in the same `SELECT` item as the sensitive column `COMMISSION_PCT`.

```
SELECT (SALARY * :VAR1), (COMMISSION_PCT * :VAR2), (EMPNO + :VAR3) AS BONUS AS FROM
PAYROLL.ACCOUNT;
```

Bind Variables in Expressions Assigned to Sensitive Columns in INSERT or UPDATE Operations

You can assign multiple bind variables to different columns in one `INSERT` or `UPDATE` statement.

Consider the following `INSERT` statement:

```
INSERT INTO PAYROLL.ACCOUNT (ACCOUNT_NUM, SALARY) VALUES (:VAR1 * :VAR2 , :VAR3);
```

In this `INSERT` statement, the following takes place:

- The bind variables `VAR1` and `VAR2` appear in the expression (`:VAR1 * :VAR2`), which is assigned to the sensitive column `ACCOUNT_NUM`.

- The bind variable VAR3 is assigned to sensitive column SALARY.

Consider the following UPDATE statement:

```
UPDATE PAYROLL.ACCOUNT SET ACCOUNT_NUM = :VAR1, SALARY = :VAR2;
```

In this UPDATE statement, the following takes place:

- The bind variable VAR1 is assigned to sensitive column ACCOUNT_NUM.
- The bind variable VAR2 is assigned to sensitive column SALARY.

How Bind Variables on Sensitive Columns Behave with Views

A bind variable that appears in a query on a view is considered sensitive if the view column references a sensitive column.

For example, suppose you identify the SALARY column in the HR.EMPLOYEES table as sensitive. Then you create the view EMPLOYEES_VIEW as follows:

```
CREATE OR REPLACE VIEW HR.EMPLOYEES_VIEW AS SELECT * FROM HR.EMPLOYEES;
```

When a user references the SALARY column from this view in a SQL statement, any bind variable that has been associated with the SALARY column is considered sensitive and its bind value then masked.

```
SELECT EMPLOYEE_ID FROM HR.EMPLOYEES_VIEW WHERE SALARY = :VAR1;
```

In this case, the bind variable VAR1 is masked because it is associated with the HR.EMPLOYEES_VIEW.SALARY column, which references the sensitive column HR.EMPLOYEES.SALARY.

Disabling the REDACT_AUDIT Policy

By default, the REDACT_AUDIT policy is enabled for all sensitive columns.

You can disable it for a specific sensitive column or all sensitive columns, and when needed, re-enable it. Remember that you cannot alter or delete the REDACT_AUDIT policy.

- To disable the REDACT_AUDIT policy, use the DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN procedure.

For example, to disable the REDACT_AUDIT policy for the SALARY column of HR.EMPLOYEES:

```
BEGIN
 DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN(
  schema_name          => 'HR',
  table_name           => 'EMPLOYEES',
  column_name          => 'SALARY',
  policy               => 'REDACT_AUDIT');
END;
/
```

The following example shows how to disable the REDACT_AUDIT policy for all sensitive columns in the current database.

```
BEGIN
 DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN(
  policy               => 'REDACT_AUDIT');
```

```
END;
/
```

Enabling the REDACT_AUDIT Policy

You can enable the `REDACT_AUDIT` policy for a specific sensitive column or for all columns in the database.

- To enable the `REDACT_AUDIT` policy, use the `DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN` procedure.

For example, to re-enable the `REDACT_AUDIT` policy for the `SALARY` column of `HR.EMPLOYEES`:

```
BEGIN
 DBMS_TSDP_PROTECT.ENABLE_PROTECTION_COLUMN(
  schema_name          => 'HR',
  table_name           => 'EMPLOYEES',
  column_name          => 'SALARY',
  policy               => 'REDACT_AUDIT');
END;
/
```

The following example shows how to enable the `REDACT_AUDIT` policy for all sensitive columns in the current database.

```
BEGIN
 DBMS_TSDP_PROTECT.ENSABLE_PROTECTION_COLUMN(
  policy               => 'REDACT_AUDIT');
END;
/
```

Transparent Sensitive Data Protection Policies with Data Redaction

Oracle Data Redaction features work with transparent sensitive data protection policies.

The Data Redaction function types, function parameters, and expressions can be used in the TSDP policy definition. For example, you can set the enable the TSDP policy to use `FULL` or `PARTIAL` data redaction. This chapter uses Data Redaction for examples of managing TSDP policies.

See Also:

- Creating Transparent Sensitive Data Protection Policies for an example of how to create TSDP policies that use Data Redaction function types
- *Oracle Database Advanced Security Guide* for details about Oracle Data Redaction

Using Transparent Sensitive Data Protection Policies with Oracle VPD Policies

You can combine protections from TSDP and Oracle Virtual Private Database into one policy.

About Using TSDP Policies with Oracle Virtual Private Database Policies

To incorporate Oracle Virtual Private Database protection with transparent sensitive data protection policies, you must use the `DBMS_TSDP_PROTECT` and `DBMS_RLS` packages.

This feature works as follows:

1. You create a VPD policy function with a suitable predicate. Later on, when you create the TSDP policy, you will refer to this VPD policy function by using the `policy_function` setting of the `DBMS_RLS.ADD_POLICY` procedure for the `feature_options` parameter of the `DBMS_TSDP_PROTECT.ADD_POLICY` procedure.
2. You create a TSDP policy with the necessary VPD settings similar to the VPD policy function.

   The TSDP policy uses parameter settings from the `DBMS_RLS.ADD_POLICY` procedure to provide VPD protection. Table 12-1 lists these parameters. Be aware that parameters from the `DBMS_RLS.ADD_GROUPED_POLICY` policy are not supported.
3. You associate the TSDP policy with the necessary sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.
4. You then enable TSDP protection by using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_*` procedures.
5. You enable the TSDP policy. At this point, Oracle Database creates an internal VPD policy that uses the function that you created in Step 1.

   The name of the internal policy begins with `ORA$VPD` followed by an identifier (for example, `ORA$VPD_6J6L3RSJSN2VAN0XF`). You can find this policy by querying the `POLICY_NAME` column of the `DBA_POLICIES` data dictionary view.
6. When users query the table, the output for the column is based on both the VPD protections and the TSDP policy that are now in place.
7. These protections remain in place until you disable the TSDP policy for this column. At that point, Oracle Database automatically drops the internal VPD policy, because it is no longer needed. If you reenable the TSDP policy, then the internal VPD policy is recreated.

DBMS_RLS.ADD_POLICY Parameters That Are Used for TSDP Policies

Oracle Database provides a set of parameters for fine-tuning the behavior of TSDP policies.

Table 12-1 describes the DBMS_RLS.ADD_POLICY parameters that are permissible in the FEATURE_OPTIONS parameter when you use the DBMS_TSDP_PROTECT.ADD_POLICY or DBMS_TSDP_PROTECT.ALTER_POLICY procedure.

Table 12-1 DBMS_RLS.ADD_POLICY Parameters Used for TSDP Policies

Parameter	Description	Default
function_schema	Schema of the policy function (current default schema, if NULL). If no function_schema is specified, then the current user's schema is assumed.	NULL
policy_function	Name of a function that generates a predicate for the policy. If the function is defined within a package, then you must include the name of the package (for example, my_package.my_function).	NULL
statement_types	Statement types to which the policy applies. It can be any combination of INDEX, SELECT, INSERT, UPDATE, or DELETE. The default is to apply to most of these types except INDEX.	NULL
update_check	Optional argument for INSERT or UPDATE statement types. Setting update_check to TRUE sets Oracle Database to check the policy against the value after an INSERT or UPDATE operation. The check applies only to the security relevant columns that are included in the policy definition. In other words, the INSERT or UPDATE operation will fail only if the security relevant column that is defined in the policy is added or updated in the INSERT or UPDATE statement.	FALSE
static_policy	If you set this value to TRUE, then Oracle Database assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for SYS or the privileged user who has the EXEMPT ACCESS POLICY privilege.	FALSE
policy_type	Default is NULL, which means policy_type is decided by the value of the static_policy parameter. Specifying any of these policy types overrides the value of static_policy.	NULL
long_predicate	Default is FALSE, which means the policy function can return a predicate with a length of up to 4000 bytes. TRUE means the predicate text string length can be up to 32K bytes. Policies existing before the availability of the long_predicate parameter retain a 32K limit.	FALSE

Table 12-1 (Cont.) DBMS_RLS.ADD_POLICY Parameters Used for TSDP Policies

Parameter	Description	Default
`sec_relevant_cols_opt`	If you specify this parameter, then transparent sensitive data protection inputs the sensitive column on which the protection is enabled to the `sec_relevant_cols` parameter of the `DBMS_RLS.ADD_POLICY` procedure. Allowed values are for `sec_relevant_cols_opt` are as follows: • `NULL` enables the filtering defined with `sec_relevant_cols` to take effect. • `DBMS_RLS.ALL_ROWS` displays all rows, but with sensitive column values, which are filtered by the `sec_relevant_cols` parameter, they display as `NULL`.	`NULL`

Tutorial: Creating a TSDP Policy That Uses Virtual Private Database Protection

This tutorial demonstrates how to incorporate Oracle Virtual Private Database protection with a transparent sensitive data protection policy.

Step 1: Create the hr_appuser User Account

First, you must create a sample user account and then grant this user the appropriate privileges.

1. Log into the database instance as user `SYS` with the `SYSDBA` administrative privilege.

   ```
   sqlplus sys as sysdba
   Enter password: password
   ```

2. If you are using a multitenant environment, then connect to the appropriate pluggable database (PDB).

   For example:

   ```
   CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

3. Create the following user accounts:

   ```
   GRANT CREATE SESSION TO hr_appuser IDENTIFIED BY password;
   GRANT CREATE SESSION TO tsdp_admin IDENTIFIED BY password;
   ```

   Follow the guidelines in Minimum Requirements for Passwords to replace *`password`* with a password that is secure.

4. Grant user `tsdp_admin` the following privileges:

   ```
   GRANT CREATE PROCEDURE TO tsdp_admin;
   GRANT EXECUTE ON DBMS_TSDP_MANAGE TO tsdp_admin;
   ```

```
GRANT EXECUTE ON DBMS_TSDP_PROTECT TO tsdp_admin;
GRANT EXECUTE ON DBMS_RLS to tsdp_admin;
```

5. Connect as user `SCOTT`.

   ```
   CONNECT SCOTT -- Or, CONNECT SCOTT@hrpdb
   Enter password: password
   ```

6. Grant the `hr_appuser` the `READ` object privilege for the `EMP` table.

   ```
   GRANT READ ON EMP TO hr_appuser;
   ```

Step 2: Identify the Sensitive Columns

As the sample user `tsdp_admin`, you are ready to identify sensitive columns to protect.

1. Connect as user `tsdp_admin`.

   ```
   CONNECT tsdp_admin -- Or, CONNECT tsdb_admin@hrpdb
   Enter password: password
   ```

2. Create the `salary_type` sensitive type:

   ```
   BEGIN
    DBMS_TSDP_MANAGE.ADD_SENSITIVE_TYPE (
     sensitive_type  => 'salary_type',
     user_comment    => 'Type for SCOTT.EMP column');
   END;
   /
   ```

3. Associate the `salary_type` sensitive type with the `SCOTT.EMP` table.

   ```
   BEGIN
    DBMS_TSDP_MANAGE.ADD_SENSITIVE_COLUMN (
    schema_name        => 'SCOTT',
    table_name         => 'EMP',
    column_name        => 'SAL',
    sensitive_type     => 'salary_type',
    user_comment       => 'Sensitive column addition of SALARY_TYPE');
   END;
   /
   ```

Step 3: Create an Oracle Virtual Private Database Function

TSDP will associate the Oracle VPD policy function with the VPD policy that is automatically created when the TSDP policy is enabled.

- To create the VPD policy function, use the `CREATE OR REPLACE FUNCTION` procedure, as follows:

  ```
  CREATE OR REPLACE FUNCTION vpd_function (
    v_schema IN VARCHAR2,
    v_objname IN VARCHAR2)
   RETURN VARCHAR2 AS
  BEGIN
   RETURN 'SYS_CONTEXT(''USERENV'',''SESSION_USER'') = ''HR_APPUSER''';
  END vpd_function;
  /
  ```

Step 4: Create and Enable a Transparent Sensitive Data Protection Policy

After you have created the VPD policy function, you can associate it with a transparent sensitive data protection policy.

1. Create the Transparent Sensitive Data Protection policy.

   ```
   DECLARE
    vpd_feature_options DBMS_TSDP_PROTECT.FEATURE_OPTIONS;
    policy_conditions DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
   BEGIN
    vpd_feature_options ('policy_function') := 'vpd_function';
    vpd_feature_options ('sec_relevant_cols_opt') := 'DBMS_RLS.ALL_ROWS';
    dbms_tsdp_protect.add_policy('tsdp_vpd', DBMS_TSDP_PROTECT.VPD,
   vpd_feature_options, policy_conditions);
   END;
   /
   ```

   In this example, the `vpd_feature_options` parameter refers to the `sec_relevant_cols_opt` parameter from the `DBMS_RLS.ADD_POLICY` procedure. When the TSDP policy is enabled, the VPD policy that is automatically created will have its `sec_relevant_cols` parameter (of `DBMS_RLS.ADD_POLICY`) set to the name of the sensitive column on which TSDP enables the VPD policy. If you had not used the `sec_relevant_cols_opt` parameter, then TSDP would not have used the `DBMS_RLS.ADD_POLICY sec_relevant_cols_opt` parameter.

2. Associate the `tsdp_vpd1` TSDP policy with the `salary_type` sensitive type.

   ```
   BEGIN
    DBMS_TSDP_PROTECT.ASSOCIATE_POLICY(
    policy_name       => 'tsdp_vpd',
    sensitive_type    => 'salary_type',
    associate         => TRUE);
   END;
   /
   ```

3. Enable protection to enforce the Virtual Private Database policy on all columns identified as `SALARY_TYPE`:

   ```
   BEGIN
    DBMS_TSDP_PROTECT.ENABLE_PROTECTION_TYPE(
     sensitive_type          => 'salary_type');
   END;
   /
   ```

Step 5: Test the Transparent Sensitive Data Protection Policy

Now, you are ready to test the transparent sensitive data protection policy.

1. Connect as user `hr_appuser`.

   ```
   CONNECT hr_appuser -- Or, CONNECT hr_appuser@hrpdb
   Enter password: password
   ```

2. Query the `SCOTT.EMP` table as follows:

   ```
   SELECT SAL, COMM, EMPNO FROM SCOTT.EMP;
   ```

   The following output appears:

```
    SAL   COMM   EMPNO
--------- ------ ----------
    800          7369
   1600    300   7499
   1250    500   7521
   2975          7566
   1250   1400   7654
   2850          7698
   2450          7782
   3000          7788
   5000          7839
   1500     0    7844
   1100          7876
    950          7900
   3000          7902
   1300          7934
14 rows selected.
```

The `vpd_function` function enables user `hr_appuser` to see the salaries in the `SAL` column of the `EMP` table.

3. Connect as user `SCOTT` and then perform the same query.

   ```
   CONNECT SCOTT -- Or, CONNECT SCOTT@hrpdb
   Enter password: password

   SELECT SAL, COMM, EMPNO FROM SCOTT.EMP;
   ```

   The following output appears:

   ```
       SAL   COMM   EMPNO
   --------- ------ ----------
                    7369
               300   7499
               500   7521
                    7566
              1400   7654
                    7698
                    7782
                    7788
                    7839
                0    7844
                    7876
                    7900
                    7902
                    7934
   14 rows selected.
   ```

   Even though `SCOTT` owns the `EMP` table, the `vpd_function` function prevents him from seeing the salaries in the `SAL` column of this table

Step 6: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. Connect as user `tsdp_admin`.

   ```
   CONNECT tsdp_admin -- Or, CONNECT tsdp_admin@hrpdb
   Enter password: password
   ```

2. Execute the following statements in the order shown.

```
BEGIN
 DBMS_TSDP_MANAGE.DROP_SENSITIVE_COLUMN (
   schema_name        => 'SCOTT',
   table_name         => 'EMP',
   column_name        => 'SAL');
END;
/

BEGIN
 DBMS_TSDP_MANAGE.DROP_SENSITIVE_TYPE(
 sensitive_type     => 'salary_type');
END;
/

BEGIN
 DBMS_TSDP_PROTECT.DROP_POLICY(
   policy_name     => 'tsdp_vpd');
END;
/
```

3. Connect as user `SYSTEM`.

   ```
   CONNECT SYSTEM -- Or, CONNECT SYSTEM@hrpdb
   Enter password: password
   ```

4. Drop the `tsdp_admin` and `hr_appuser` accounts.

   ```
   DROP USER tsdp_admin CASCADE;
   DROP USER hr_appuser
   ```

Using Transparent Sensitive Data Protection Policies with Unified Auditing

The transparent sensitive data protection and unified auditing procedures can combine the protections of these two features.

About Using TSDP Policies with Unified Audit Policies

You can configure transparent sensitive data protection policies to audit object actions using unified auditing.

The `DBMS_TSDP_PROTECT.ADD_POLICY` and `DBMS_TSDP_PROTECT.ALTER_POLICY` procedures enable you to specify settings from the `CREATE AUDIT POLICY`, `ALTER AUDIT POLICY`, `AUDIT POLICY`, and `COMMENT` SQL statements. The TSDP policy enables the creation of action audit-options for object-specific options in the policy, such as `INSERT` or `DELETE` operations. System-wide audit options are not supported. Therefore, the audited object type is always `TABLE`. Only standard actions (such as `INSERT`) are permitted. Component actions, such as creating policies for Oracle Label Security or other Oracle Database features, are not supported.

This feature works as follows:

1. You create a TSDP policy with the necessary unified audit settings.

   The TSDP policy uses parameter settings from the `CREATE AUDIT POLICY`, `AUDIT POLICY`, and `COMMENT` statements. Unified Audit Policy Settings That Are Used with TSDP Policies lists these settings.

2. You associate the TSDP policy with the necessary sensitive types by using the DBMS_TSDP_PROTECT.ASSOCIATE_POLICY procedure.
3. You then enable TSDP protection by using any of the DBMS_TSDP_PROTECT.ENABLE_PROTECTION_* procedures.
4. You enable the TSDP policy. As part of the TSDP policy enablement process, Oracle Database internally creates a unified audit policy and then enables it on the list of target users and roles that you specified in the DBMS_TSDP_PROTECT.ADD_POLICY procedure from Step 1.

   The name of the internal policy begins with ORA$UNIFIED_AUDIT_ followed by a random alpha-numeric string (for example, ORA$UNIFIED_AUDIT_6J6L3RSJSN2VAN0XF). You can find this policy by querying the POLICY_NAME column of the AUDIT_UNIFIED_POLICIES data dictionary view. To find the names of the users and roles on which this internally created TSDP unified audit policy is enforced, query the AUDIT_UNIFIED_ENABLED_POLICIES view.
5. When users try to perform an action on the table that is being protected by the TSDP policy, then based on the TSDP unified audit policy configuration, a unified audit record is written to the unified audit trail for this object access. You can then query the UNIFIED_AUDIT_TRAIL view to see the unified audit record that was created because of the TSDP unified audit policy enforcement.
6. These protections remain in place until you disable the TSDP policy for this column. At that point, Oracle Database automatically disables and then drops the internal policy, because it is no longer necessary. (A unified audit policy must be disabled before it can be dropped.) If you re-enable the TSDP policy, then the internal policy is recreated.

Unified Audit Policy Settings That Are Used with TSDP Policies

Audit policy settings can be used in the POLICY_ENABLE_OPTIONS parameter for the DBMS_TSDP_PROTECT.ADD_POLICY or DBMS_TSDP_PROTECT.ALTER_POLICY procedure.

These audit policy settings are from the AUDIT, CREATE AUDIT POLICY, and ALTER AUDIT POLICY statements.

The following table describes these settings.

Table 12-2 Unified Audit Policy Settings Used for TSDP Policies

Parameter	Description	Default
ACTION_AUDIT_OPTIONS	A string containing a comma-separated list of SQL actions. Valid actions are: ALTER, AUDIT, COMMENT, DELETE, FLASHBACK, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT, UPDATE To configure the policy to audit all of these actions, specify the keyword ALL.	ALL

Table 12-2 (Cont.) Unified Audit Policy Settings Used for TSDP Policies

Parameter	Description	Default

`AUDIT_CONDITION`	`SYS_CONTEXT (`*`namespace, attribute`*`)` *`operation value-list`*
	In this syntax, *`operation`* can be any of the following operators: `IN`,| `NOT IN`, =, <, >, or <>
	If the audit condition contains a single quotation mark, then specify two single quotation marks instead of one, and enclose the `SYS_CONTEXT` in single quotations. For example:
	`'SYS_CONTEXT(''USERENV'', ''CLIENT_IDENTIFIER'') = ''myclient'''`
		`NULL`

`EVALUATE_PER`	Can be one of the following: • `STATEMENT` • `SESSION` • `INSTANCE`	`STATEMENT`
`ENTITY_NAME`	A string that contains a comma-separated list of users or roles. If you omit this parameter, then the audit policy is enabled for all users.	`NULL` (that is, all database users)
`ENABLE_OPTION`	Applies only if the `ENTITY_NAME` parameter is used. It specifies if the `ENTITY_NAME` is a `BY` user list, an `EXCEPT` user list, or a `BY USERS WITH GRANTED ROLES` role list. Valid settings are: • `BY` • `EXCEPT` • `BY USERS WITH GRANTED ROLES`	`BY`
`UNIFIED_AUDIT_POLICY_COMMENT`	A string that describes the unified audit policy that will be created	`NULL`

Using Transparent Sensitive Data Protection Policies with Fine-Grained Auditing

The transparent sensitive data protection and fine-grained auditing procedures can combine the protections of these two features.

About Using TSDP Policies with Fine-Grained Auditing

You can configure a Transparent Sensitive Data Protection policy for fine-grained auditing.

The `DBMS_TSDP_PROTECT.ADD_POLICY` and `DBMS_TSDP_PROTECT.ALTER_POLICY` procedures enable you to specify settings from the `DBMS_FGA.ADD_POLICY` procedure.

This feature works as follows:

1. You create a TSDP policy with the necessary fine-grained audit settings.

   The TSDP policy uses parameter settings from the `DBMS_FGA.ADD_POLICY` procedure. Fine-Grained Auditing Parameters That Are Used with TSDP Policies lists these settings.

2. You associate the TSDP policy with the necessary sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.

3. You then enable TSDP protection by using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_*` procedures.

4. You enable the TSDP policy. As part of the TSDP policy enablement process, Oracle Database internally creates a fine-grained audit policy that you specified in the `DBMS_TSDP_PROTECT.ADD_POLICY` procedure from Step 1.

   The name of the internal policy begins with `ORA$FGA_` followed by a random alpha-numeric string (for example, `ORA$FGA_6J6L3RSJSN2VAN0XF`). You can find this policy by querying the `POLICY_NAME` column of the `DBA_POLICIES` data dictionary view.

5. When users try to perform an action on the table that is being protected by the TSDP policies, then based on the policy configuration, a fine-grained audit record is generated in the `DBA_FGA_AUDIT_TRAIL` data dictionary view for this object access.

6. These protections remain in place until you disable the TSDP policy for this column. At that point, Oracle Database automatically drops the internal policy, because it is no longer needed. If you reenable the TSDP policy, then the internal policy is recreated.

Fine-Grained Auditing Parameters That Are Used with TSDP Policies

`DBMS_FGA.ADD_POLICY` settings can be used in the `POLICY_ENABLE_OPTIONS` parameter for the `DBMS_TSDP_PROTECT.ADD_POLICY` or `DBMS_TSDP_PROTECT.ALTER_POLICY` procedure.

The following table describes these settings.

Table 12-3 Fine-Grained Audit Policy Settings Used for TSDP Policies

Parameter	Description	Default
`audit_condition`	Specifies a Boolean value to indicate a monitoring condition, using the following syntax: *`operator value`* For example: `< 1000`	`NULL`
`handler_schema`	Schema that contains the event handler. The default, `NULL`, enables the current schema to be used.	`NULL`
`handler_module`	Function name of the event handler. Include the package name if necessary. This function is invoked only after the first row that matches the audit condition in the query is processed. If the procedure fails with an exception, then the user's SQL statement fails as well.	`NULL`
`statement_types`	You can specify one of the following statement types: `INSERT`, `UPDATE`, `SELECT`, or `DELETE`.	`SELECT`

Table 12-3 (Cont.) Fine-Grained Audit Policy Settings Used for TSDP Policies

Parameter	Description	Default
`audit_trail`	If you have not yet migrated the database to full unified auditing, then use this setting to set the destination of the audit records: `DB` for the database or XML for XML records. This setting also specifies whether to populate the `LSQLTEXT` and `LSQLBIND` columns in the `FGA_LOG$` system table. If full unified auditing is enabled, then Oracle Database ignores this parameter and writes the audit records to the unified audit trail.	`NULL`
`object_schema`	The schema that corresponds to the sensitive column	Schema that contains the sensitive column
`object_name`	The table that contains the sensitive column	The object (table or view) that contains the sensitive column
`policy_name`	A system-generated name for the internal fine-grained audit policy	Internal fine-grained audit policy system-generated name
`audit_column`	The sensitive column	The sensitive column
`audit_column_opts`	Determines whether to audit all or specific columns	`DBMS_FGA.ANY_COLUMN`
`enable`	Enable status for the TSDP policy; can be either `TRUE` or `FALSE`	`TRUE`
`policy_owner`	User who invokes the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_*` procedure	Current user

Using Transparent Sensitive Data Protection Policies with TDE Column Encryption

The TSDP procedures and Transparent Data Encryption column encryption statements can combine the protections of these two features.

About Using TSDP Policies with TDE Column Encryption

A TSDP policy can enable the encryption of columns that use Transparent Data Encryption.

The `DBMS_TSDP_PROTECT.ADD_POLICY` and `DBMS_TSDP_PROTECT.ALTER_POLICY` procedures enable you to specify the `ENCRYPT` clause settings from the `CREATE TABLE` or `ALTER TABLE` statement.

This feature works as follows:

1. You can create a TSDP policy by using the `DBMS_TSDP_PROTECT.ADD_POLICY` procedure. In the `ADD_POLICY` procedure, you can configure the policy for column encryption by setting the `SECURITY_FEATURE` parameter to `DBMS_TSDP_PROTECT.COLUMN_ENCRYPTION`. This setting enables encryption on the sensitive column when the TSDP policy is enabled on the object.
2. You create a TSDP policy with the necessary table encryption settings.

   The TSDP policy uses parameter settings from the `CREATE TABLE` or `ALTER TABLE` SQL statement. TDE Column Encryption ENCRYPT Clause Settings Used with TSDP Policies lists these settings.
3. You associate the TSDP policy with the necessary sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.
4. You then enable TSDP protection by using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_*` procedures.
5. You enable the TSDP policy. At this point, Oracle Database creates an internal TSDP policy that uses the encrypted table settings that you created earlier in this procedure.

   The name of the internal policy begins with `ORA$TDECE_` followed by a random alpha-numeric string (for example, `ORA#TDECE_6J6L3RSJSN2VAN0XF`). You can find this policy by querying the `TSDP_POLICY` column of `DBA_TSDP_POLICY_PROTECTION` view.
6. When users try to perform an action on the table that is being protected by the policies, the output for the column is based on both the TDE column protections and the TSDP policy that are now in place. You can check if the column has been encrypted after you enabled the TSDP policy by querying the `ENCRYPTION_ALG` column of the `DBA_ENCRYPTED_COLUMNS` view.
7. These protections remain in place until you disable the TSDP policy for this column. At that point, Oracle Database internally issues an `ALTER TABLE` statement on the table that contains the sensitive column, so that the sensitive column is decrypted. If you reenable the TSDP policy, then TSDP internally executes the `ALTER TABLE` statement with the `ENCRYPT` clause for the column.

Note:

It is possible to create two policies on the same column with each policy specifying a different encryption algorithm. In this case, the stronger of the two algorithms is enforced on the sensitive column.

TDE Column Encryption ENCRYPT Clause Settings Used with TSDP Policies

The `CREATE TABLE` and `ALTER TABLE` statement `ENCRYPT` clause settings can be used in the `POLICY_ENABLE_OPTIONS` parameter for the `DBMS_TSDP_PROTECT.ADD_POLICY` or `DBMS_TSDP_PROTECT.ALTER_POLICY` procedure.

The following table describes these settings.

Table 12-4 TDE Column Encryption ENCRYPT Settings Used for TSDP Policies

Parameter	Description	Default
`encrypt_algorithm`	Available values • `3DES168` • `AES128` • `AES192` • `AES256` • `ARIA128` • `ARIA192` • `ARIA256` • `SEED128` • `GOST256`	`AES192`
`salt`	Available values: • `SALT` • `NO SALT`	`SALT`
`integrity_algorithm`	Available values: • `SHA-1` • `NOMAC`	`SHA-1`

Transparent Sensitive Data Protection Data Dictionary Views

Oracle Database provides data dictionary views that list information about transparent sensitive data protection policies.

Table 12-5 describes these views. Before you can use these views, you must be granted the `SELECT_CATALOG_ROLE` role.

Table 12-5 Transparent Sensitive Data Protection Views

View	Description
`DBA_DISCOVERY_SOURCE`	Describes discovery import information with regard to transparent sensitive data protection policies
`DBA_SENSITIVE_COLUMN_TYPES`	Describes the sensitive column types that have been defined for the current database
`DBA_SENSITIVE_DATA`	Describes the sensitive columns in the database
`DBA_TSDP_IMPORT_ERRORS`	Shows information regarding the errors encountered during import of discovery result. It shows information with regard to the error code, schema name, table name, column name, and sensitive type.
`DBA_TSDP_POLICY_CONDITION`	Describes the transparent sensitive data protection policy and condition mapping. This view also lists the property-value pairs for the condition.
`DBA_TSDP_POLICY_FEATURE`	Shows the transparent sensitive data protection policy security feature mapping. (At this time, only Oracle Data Redaction and Oracle Virtual Private Database are supported.)

Table 12-5 (Cont.) Transparent Sensitive Data Protection Views

View	Description
`DBA_TSDP_POLICY_PARAMETER`	Describes the parameters of transparent sensitive data protection policies
`DBA_TSDP_POLICY_PROTECTION`	Shows the list of columns that have been protected through transparent sensitive data protection
`DBA_TSDP_POLICY_TYPE`	Shows the policy to sensitive column type mapping

See Also:

Oracle Database Reference for more information about these views

13

Encryption of Sensitive Credential Data in the Data Dictionary

You can encrypt sensitive credential information, such as passwords that are stored in the data dictionary.

About Encrypting Sensitive Credential Data in the Data Dictionary

The data dictionary `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` system tables store sensitive credential data, such as user passwords.

The `SYS.LINK$` table stores information about database links. `SYS.SCHEDULER$_CREDENTIAL` stores information about Oracle Scheduler events. By default, the sensitive credential data stored in these tables is obfuscated.

You can manually encrypt the data that is stored in the `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` tables by using the `ALTER DATABASE DICTIONARY` statement. Though this feature makes use of Transparent Data Encryption (TDE), you do not need to have an Advanced Security Option license to perform the encryption, but you must have the `SYSKM` administrative privilege. TDE performs the encryption by using the AES256 (Advanced Encryption Standard) algorithm. The encryption follows the same behavior as other data that is encrypted using TDE.

As a best security practice, Oracle recommends that you encrypt this sensitive credential data. To check the status the data dictionary credentials, you can query the `DICTIONARY_CREDENTIALS_ENCRYPT` data dictionary view.

How the Multitenant Option Affects the Encryption of Sensitive Data

In a multitenant environment, you can encrypt sensitive data dictionary information from the application root, as well as within individual pluggable databases (PDBs).

When you encrypt, rekey, or decrypt sensitive credential data in the `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` system tables, you must synchronize the affected PDBs after you complete the process. The instructions for doing so are in the procedures that cover these topics.

Encrypting Sensitive Credential Data in System Tables

The `ALTER DATABASE DICTIONARY` statement can encrypt sensitive credential data in the `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` system tables.

The database must have an open keystore and an encryption key before you run the `ALTER DATABASE DICTIONARY` statement with the `ENCRYPT CREDENTIALS` clause to encrypt

SYS.LINK$ and SYS.SCHEDULER$_CREDENTIAL. The credential data encryption process de-obfuscates the obfuscated passwords and then encrypts them. The encryption applies to any future password changes that users may make after you complete this process.

1. Connect to the database instance as a user who as been granted the SYSKM administrative privilege.

   For example:

   ```
   CONNECT hr_admin AS SYSKM
   Enter password: password
   ```

   In a multitenant environment, connect to either the application root or to a pluggable database (PDB).

2. If necessary, create and open a keystore and then set an encryption key.

   You can query the V$ENCRYPTION_WALLET dynamic view to find the status of a keystore.

   Use the ADMINISTER KEY MANAGEMENT statement to perform these three tasks. For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED
   BY password;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "password";
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "password" WITH
   BACKUP;
   ```

   In a multitenant environment, include the CONTAINER = ALL clause if you are in the application root. This applies the keystore operation for PDBs that are in united mode. For PDBs that are in isolated mode, run the statement from within the PDB.

3. Run the ALTER DATABASE DICTIONARY statement to encrypt the data.

   For example:

   ```
   ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS;
   ```

   In an application root, to apply the encryption to the associated PDBs, include the CONTAINER = ALL clause.

   ```
   ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS CONTAINER = ALL;
   ```

4. If you performed the encryption from the application root, then synchronize the associated PDBs.

   ```
   ALTER PLUGGABLE DATABASE APPLICATION APP$CDB$SYSTEM SYNC;
   ```

Rekeying Sensitive Credential Data in the SYS.LINK$ System Table

You can use the ALTER DATABASE DICTIONARY statement to rekey sensitive credential data in the data dictionary SYS.LINK$ and SYS.SCHEDULER$_CREDENTIAL system tables.

To rekey this sensitive credential data, you must run the ALTER DATABASE DICTIONARY statement with the REKEY CREDENTIALS clause. The rekey operation, which uses column encryption, does not affect other TDE master encryption keys.

1. Connect to the database instance as a user who as been granted the SYSKM administrative privilege.

For example:

```
CONNECT hr_admin AS SYSKM
Enter password: password
```

In a multitenant environment, connect to either the application root or to a pluggable database (PDB).

2. If necessary, create and open a keystore and then set an encryption key.

   You can query the `V$ENCRYPTION_WALLET` dynamic view to find the status of a keystore.

   Use the `ADMINISTER KEY MANAGEMENT` statement to perform these three tasks. For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED
   BY password;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "password";
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "password" WITH
   BACKUP;
   ```

   In a multitenant environment, include the `CONTAINER = ALL` clause if you are in the application root.

3. Run the `ALTER DATABASE DICTIONARY` statement to rekey the data.

   For example:

   ```
   ALTER DATABASE DICTIONARY REKEY CREDENTIALS;
   ```

   In an application root, to apply the encryption to the associated PDBs, include the `CONTAINER = ALL` clause.

   ```
   ALTER DATABASE DICTIONARY REKEY CREDENTIALS CONTAINER = ALL;
   ```

4. If you performed the rekey operation from the application root, then synchronize the associated PDBs.

   ```
   ALTER PLUGGABLE DATABASE APPLICATION APP$CDB$SYSTEM SYNC;
   ```

Deleting Sensitive Credential Data in System Tables

The `ALTER DATABASE DICTIONARY` statement can invalidate existing credentials in `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` and obfuscate future credential entries to those tables.

To delete this credential data, you must run the `ALTER DATABASE DICTIONARY` statement with the `DELETE CREDENTIALS` clause. This statement is mainly used in cases where you must recover the database link from the loss of a Transparent Data Encryption (TDE) keystore.

1. Connect to the database instance as a user who as been granted the `SYSKM` administrative privilege.

   For example:

   ```
   CONNECT hr_admin AS SYSKM
   Enter password: password
   ```

   In a multitenant environment, connect to either the application root or to a pluggable database (PDB).

2. If necessary, create and open a keystore and then set an encryption key.

   You can query the `V$ENCRYPTION_WALLET` dynamic view to find the status of a keystore.

   Use the `ADMINISTER KEY MANAGEMENT` statement to perform these three tasks. For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED
   BY password;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "password";
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "password" WITH
   BACKUP;
   ```

   In a multitenant environment, include the `CONTAINER = ALL` clause if you are in the application root.

3. Run the `ALTER DATABASE DICTIONARY` statement to delete the password credential.

   For example:

   ```
   ALTER DATABASE DICTIONARY DELETE CREDENTIALS;
   ```

   In an application root, to delete the `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` password credentials in the associated PDBs, include the `CONTAINER = ALL` clause.

   ```
   ALTER DATABASE DICTIONARY DELETE CREDENTIALS CONTAINER = ALL;
   ```

4. If you performed the credential deletion from the application root, then synchronize the associated PDBs.

   ```
   ALTER PLUGGABLE DATABASE APPLICATION APP$CDB$SYSTEM SYNC;
   ```

Restoring the Functioning of Database Links After a Lost Keystore

Database links can be adversely affected if the TDE keystore and its master encryption key is inadvertently lost.

When a TDE keystore and master encryption key are lost, existing database links that are authenticated with encrypted passwords become unusable.

1. Connect to the database instance as a user who as been granted the `SYSKM` administrative privilege and who has the `ALTER DATABASE LINK` system privilege.

   For example:

   ```
   CONNECT hr_admin AS SYSKM
   Enter password: password
   ```

   In a multitenant environment, connect to either the application root or to a pluggable database (PDB).

2. Delete the encrypted credentials from the `SYS.LINK$` system table.

   ```
   ALTER DATABASE DICTIONARY DELETE CREDENTIALS KEY;
   ```

   If you are performing the deletion from the application root, then include the `CONTAINER = ALL` clause.

   ```
   ALTER DATABASE DICTIONARY DELETE CREDENTIALS CONTAINER = ALL;
   ```

3. Create and open a keystore and then set an encryption key.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED
   BY password;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "password";
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "password" WITH
   BACKUP;
   ```

   In a multitenant environment, include the `CONTAINER = ALL` clause if you are in the application root.

4. Encrypt the password credentials in `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL`.

   ```
   ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS;
   ```

   If you are performing the encryption from the application root, then include the `CONTAINER = ALL` clause.

   ```
   ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS CONTAINER = ALL;
   ```

5. Using the password of the user who is associated with the database link, reset the database link passwords that were affected by the `ALTER DATABASE DICTIONARY DELETE CREDENTIALS KEY` statement.

   For example:

   ```
   ALTER DATABASE LINK database_link_name CONNECT TO user_id IDENTIFIED BY password;
   ```

   To find existing database links and their owners, query the `DBA_DB_LINKS` data dictionary view.

6. If you performed the credential deletion from the application root, then synchronize the associated PDBs.

   ```
   ALTER PLUGGABLE DATABASE APPLICATION APP$CDB$SYSTEM SYNC;
   ```

Data Dictionary Views for Encrypted Data Dictionary Credentials

Oracle Database provides a set of data dictionary views that provide information about the encryption of sensitive credential data in the data dictionary.

Table 13-1 lists the data dictionary views. For detailed information about these views, see *Oracle Database Reference*.

Table 13-1 Data Dictionary Views for Encrypted Data Dictionary Credentials

View	Description
`ALL_DB_LINKS`	Describes database links that are accessible to the current user. A value of `YES` in the `VALID` column indicates that the database link is usable.

Table 13-1 (Cont.) Data Dictionary Views for Encrypted Data Dictionary Credentials

View	Description
`DBA_DB_LINKS`	Describes describes all database links in the database. A value of `YES` in the `VALID` column indicates that the database link is usable. (This view is available to administrative users only, such as `SYS` or users who have been granted the `DBA` role.)
`DICTIONARY_CREDENTIALS_ENCRYPT`	Describes the status of dictionary credentials. The `ENFORCEMENT` column lists `ENABLED` if the credentials are encrypted and `DISABLED` if the credentials are not encrypted.
`USER_DB_LINKS`	Describes the database links hat are owned by the current user. A value of `YES` in the `VALID` column indicates that the database link is usable.

14

Manually Encrypting Data

You can use the DBMS_CRYPTO PL/SQL package to manually encrypt data.

Security Problems That Encryption Does Not Solve

While there are many good reasons to encrypt data, there are many reasons not to encrypt data.

Principle 1: Encryption Does Not Solve Access Control Problems

When you encrypt data, you should be aware that encryption must not interfere with how you configure access control.

Most organizations must limit data access to users who need to see this data. For example, a human resources system may limit employees to viewing only their own employment records, while allowing managers of employees to see the employment records of subordinates. Human resource specialists may also need to see employee records for multiple employees.

Typically, you can use access control mechanisms to address security policies that limit data access to those with a need to see it. Oracle Database has provided strong, independently evaluated access control mechanisms for many years. It enables access control enforcement to a fine level of granularity through Virtual Private Database.

Because human resource records are considered sensitive information, it is tempting to think that all information should be encrypted for better security. However, encryption cannot enforce granular access control, and it may hinder data access. For example, an employee, his manager, and a human resources clerk may all need to access an employee record. If all employee data is encrypted, then all three must be able to access the data in unencrypted form. Therefore, the employee, the manager and the human resources clerk would have to share the same encryption key to decrypt the data. Encryption would, therefore, not provide any additional security in the sense of better access control, and the encryption might hinder the proper or efficient functioning of the application. An additional issue is that it is difficult to securely transmit and share encryption keys among multiple users of a system.

A basic principle behind encrypting stored data is that it must not interfere with access control. For example, a user who has the SELECT privilege on emp should not be limited by the encryption mechanism from seeing all the data he is otherwise allowed to see. Similarly, there is little benefit to encrypting part of a table with one key and part of a table with another key if users need to see all encrypted data in the table. In this case, encryption adds to the overhead of decrypting the data before users can read it. If access controls are implemented well, then encryption adds little additional security within the database itself. A user who has privileges to access data within the database has no more nor any less privileges as a result of encryption. Therefore, you should never use encryption to solve access control problems.

Principle 2: Encryption Does Not Protect Against a Malicious Administrator

You can protect your databases against malicious database administrators by using other Oracle features, such as Oracle Database Vault.

Some organizations, concerned that a malicious user might gain elevated (database administrator) privileges by guessing a password, like the idea of encrypting stored data to protect against this threat.

However, the correct solution to this problem is to protect the database administrator account, and to change default passwords for other privileged accounts. The easiest way to break into a database is by using a default password for a privileged account that an administrator allowed to remain unchanged. One example is `SYS/CHANGE_ON_INSTALL`.

While there are many destructive things a malicious user can do to a database after gaining the `DBA` privilege, encryption will not protect against many of them. Examples include corrupting or deleting data, exporting user data to the file system to email the data back to himself to run a password cracker on it, and so on.

Some organizations are concerned that database administrators, typically having all privileges, are able to see all data in the database. These organizations feel that the database administrators should administer the database, but should not be able to see the data that the database contains. Some organizations are also concerned about concentrating so much privilege in one person, and would prefer to partition the DBA function, or enforce two-person access rules.

It is tempting to think that encrypting all data (or significant amounts of data) will solve these problems, but there are better ways to protect against these threats. For example, Oracle Database supports limited partitioning of `DBA` privileges. Oracle Database provides native support for `SYSDBA` and `SYSOPER` users. `SYSDBA` has all privileges, but `SYSOPER` has a limited privilege set (such as startup and shutdown of the database).

Furthermore, you can create smaller roles encompassing several system privileges. A `jr_dba` role might not include all system privileges, but only those appropriate to a junior database administrator (such as `CREATE TABLE`, `CREATE USER`, and so on).

Oracle Database also enables auditing the actions taken by `SYS` (or `SYS`-privileged users) and storing that audit trail in a secure operating system location. Using this model, a separate auditor who has root privileges on the operating system can audit all actions by `SYS`, enabling the auditor to hold all database administrators accountable for their actions.

You can also fine-tune the access and control that database administrators have by using Oracle Database Vault.

The database administrator function is a trusted position. Even organizations with the most sensitive data, such as intelligence agencies, do not typically partition the database administrator function. Instead, they manage their database administrators strongly, because it is a position of trust. Periodic auditing can help to uncover inappropriate activities.

Encryption of stored data must not interfere with the administration of the database, because otherwise, larger security issues can result. For example, if by encrypting data you corrupt the data, then you create a security problem, the data itself cannot be interpreted, and it may not be recoverable.

You can use encryption to limit the ability of a database administrator or other privileged user to see data in the database. However, it is not a substitute for managing the database administrator privileges properly, or for controlling the use of powerful system privileges. If untrustworthy users have significant privileges, then they can pose multiple threats to an organization, some of them far more significant than viewing unencrypted credit card numbers.

See Also:

Oracle Database Vault Administrator's Guide for more information about using Oracle Database Vault to fine-tune the access and control that database administrators have

Principle 3: Encrypting Everything Does Not Make Data Secure

A common error is to think that if encrypting some data strengthens security, then encrypting everything makes all data secure.

As the discussion of the previous two principles illustrates, encryption does not address access control issues well, and it is important that encryption not interfere with normal access controls. Furthermore, encrypting an entire production database means that all data must be decrypted to be read, updated, or deleted. Encryption is inherently a performance-intensive operation; encrypting all data will significantly affect performance.

Availability is a key aspect of security. If encrypting data makes data unavailable, or adversely affects availability by reducing performance, then encrypting everything will create a new security problem. Availability is also adversely affected by the database being inaccessible when encryption keys are changed, as good security practices require on a regular basis. When the keys are to be changed, the database is inaccessible while data is decrypted and reencrypted with a new key or keys.

There may be advantages to encrypting data stored off-line. For example, an organization may store backups for a period of 6 months to a year off-line, in a remote location. Of course, the first line of protection is to secure the facility storing the data, by establishing physical access controls. Encrypting this data before it is stored may provide additional benefits. Because it is not being accessed on-line, performance need not be a consideration. While an Oracle database does not provide this capability, there are vendors who provide encryption services. Before embarking on large-scale encryption of backup data, organizations considering this approach should thoroughly test the process. It is essential to verify that data encrypted before off-line storage can be decrypted and re-imported successfully.

Data Encryption Challenges

In cases where encryption can provide additional security, there are some associated technical challenges.

Encrypted Indexed Data

Special difficulties arise when encrypted data is indexed.

For example, suppose a company uses a national identity number, such as the U.S. Social Security number (SSN), as the employee number for its employees. The company considers employee numbers to be sensitive data, and, therefore, wants to encrypt data in the `employee_number` column of the `employees` table. Because `employee_number` contains unique values, the database designers want to have an index on it for better performance.

However, if `DBMS_CRYPTO` (or another mechanism) is used to encrypt data in a column, then an index on that column will also contain encrypted values. Although an index can be used for equality checking (for example, `SELECT * FROM emp WHERE employee_number = '987654321'`), if the index on that column contains encrypted values, then the index is essentially unusable for any other purpose. You should not encrypt indexed data.

Oracle recommends that you do not use national identity numbers as unique IDs. Instead, use the `CREATE SEQUENCE` statement to generate unique identity numbers. Reasons to avoid using national identity numbers are as follows:

- There are privacy issues associated with overuse of national identity numbers (for example, identity theft).
- Sometimes national identity numbers can have duplicates, as with U.S. Social Security numbers.

Generated Encryption Keys

Encrypted data is only as secure as the key used for encrypting it.

An encryption key must be securely generated using secure cryptographic key generation. Oracle Database provides support for secure random number generation, with the `RANDOMBYTES` function of `DBMS_CRYPTO`. (This function replaces the capabilities provided by the `GetKey` procedure of the earlier `DBMS_OBFUSCATION_TOOLKIT`, which has been deprecated.) `DBMS_CRYPTO` calls the secure random number generator (RNG) previously certified by RSA Security.

Note:

Do not use the `DBMS_RANDOM` package. The `DBMS_RANDOM` package generates pseudo-random numbers, which, as Randomness Recommendations for Security (RFC-1750) states that using pseudo-random processes to generate secret quantities can result in pseudo-security.

Be sure to provide the correct number of bytes when you encrypt a key value. For example, you must provide a 16-byte key for the `ENCRYPT_AES128` encryption algorithm.

Transmitted Encryption Keys

If the encryption key is to be passed by the application to the database, then you must encrypt it.

Otherwise, an intruder could get access to the key as it is being transmitted. Network data encryption protects all data in transit from modification or interception, including cryptographic keys.

Storing Encryption Keys

You can store encryption keys in the database or on an operating system.

About Storing Encryption Keys

Storing encryption keys is one of the most important, yet difficult, aspects of encryption.

To recover data encrypted with a symmetric key, the key must be accessible to an authorized application or user seeking to decrypt the data. At the same time, the key must be inaccessible to someone who is maliciously trying to access encrypted data that he is not supposed to see.

Storage of Encryption Keys in the Database

Storing encryption keys in the database does not always prevent a database administrator from accessing encrypted data.

An all-privileged database administrator could still access tables containing encryption keys. However, it can often provide good security against the casual curious user or against someone compromising the database file on the operating system.

As a trivial example, suppose you create a table (`EMP`) that contains employee data. You want to encrypt the employee Social Security number (SSN) stored in one of the columns. You could encrypt employee SSN using a key that is stored in a separate column. However, anyone with `SELECT` access on the entire table could retrieve the encryption key and decrypt the matching SSN.

While this encryption scheme seems easily defeated, with a little more effort you can create a solution that is much harder to break. For example, you could encrypt the SSN using a technique that performs some additional data transformation on the `employee_number` before using it to encrypt the SSN. This technique might be as simple as using an `XOR` operation on the `employee_number` and the birth date of the employee to determine the validity of the values.

As additional protection, PL/SQL source code performing encryption can be wrapped, (using the `WRAP` utility) which obfuscates (scrambles) the code. The `WRAP` utility processes an input SQL file and obfuscates the PL/SQL units in it. For example, the following command uses the `keymanage.sql` file as the input:

```
wrap iname=/mydir/keymanage.sql
```

A developer can subsequently have a function in the package call the `DBMS_CRYPTO` package calls with the key contained in the wrapped package.

Oracle Database enables you to obfuscate dynamically generated PL/SQL code. The DBMS_DDL package contains two subprograms that allow you to obfuscate dynamically generated PL/SQL program units. For example, the following block uses the DBMS_DDL.CREATE_WRAPPED procedure to wrap dynamically generated PL/SQL code.

```
BEGIN
......
SYS.DBMS_DDL.CREATE_WRAPPED(function_returning_PLSQL_code());
......
END;
```

While wrapping is not unbreakable, it makes it harder for an intruder to get access to the encryption key. Even in cases where a different key is supplied for each encrypted data value, you should not embed the key value within a package. Instead, wrap the package that performs the key management (that is, data transformation or padding).

See Also:

Oracle Database PL/SQL Packages and Types Reference for additional information about the WRAP command line utility and the DBMS_DDL subprograms for dynamic wrapping

An alternative to wrapping the data is to have a separate table in which to store the encryption key and to envelope the call to the keys table with a procedure. The key table can be joined to the data table using a primary key to foreign key relationship. For example, employee_number is the primary key in the employees table that stores employee information and the encrypted SSN. The employee_number column is a foreign key to the ssn_keys table that stores the encryption keys for the employee SSN. The key stored in the ssn_keys table can also be transformed before use (by using an XOR operation), so the key itself is not stored unencrypted. If you wrap the procedure, then that can hide the way in which the keys are transformed before use.

The strengths of this approach are:

- Users who have direct table access cannot see the sensitive data unencrypted, nor can they retrieve the keys to decrypt the data.
- Access to decrypted data can be controlled through a procedure that selects the encrypted data, retrieves the decryption key from the key table, and transforms it before it can be used to decrypt the data.
- The data transformation algorithm is hidden from casual snooping by wrapping the procedure, which obfuscates the procedure code.
- SELECT access to both the data table and the keys table does not guarantee that the user with this access can decrypt the data, because the key is transformed before use.

The weakness to this approach is that a user who has SELECT access to both the key table and the data table, and who can derive the key transformation algorithm, can break the encryption scheme.

The preceding approach is not infallible, but it is adequate to protect against easy retrieval of sensitive information stored in clear text.

Storage of Encryption Keys in the Operating System

When you store encryption keys in an operating system flat file, you can make callouts from PL/SQL to retrieve these encryption keys.

However, if you store keys in the operating system and make callouts to it, then your data is only as secure as the protection on the operating system.

If your primary security concern is that the database can be broken into from the operating system, then storing the keys in the operating system makes it easier for an intruder to retrieve encrypted data than storing the keys in the database itself.

Users Managing Their Own Encryption Keys

Having the user supply the key assumes the user will be responsible with the key.

Considering that 40 percent of help desk calls are from users who have forgotten their passwords, you can see the risks of having users manage encryption keys. In all likelihood, users will either forget an encryption key, or write the key down, which then creates a security weakness. If a user forgets an encryption key or leaves the company, then your data is not recoverable.

If you do decide to have user-supplied or user-managed keys, then you need to ensure you are using network encryption so that the key is not passed from the client to the server in the clear. You also must develop key archive mechanisms, which is also a difficult security problem. Key archives and backdoors create the security weaknesses that encryption is attempting to solve.

Manual Encryption with Transparent Database Encryption and Tablespace Encryption

Transparent database encryption and tablespace encryption provide secure encryption with automatic key management for the encrypted tables and tablespaces.

If the application requires protection of sensitive column data stored on the media, then these two types of encryption are a simple and fast way of achieving this.

See Also:

Oracle Database Advanced Security Guide for more information about Transparent Data Encryption

Importance of Changing Encryption Keys

Prudent security practice dictates that you periodically change encryption keys.

For stored data, this requires periodically unencrypting the data, and then reencrypting it with another well-chosen key.

You would most likely change the encryption key while the data is not being accessed, which creates another challenge. This is especially true for a Web-based application encrypting credit card numbers, because you do not want to shut down the entire application while you switch encryption keys.

Encryption of Binary Large Objects

Certain data types require more work to encrypt.

For example, Oracle Database supports storage of binary large objects (BLOBs), which stores very large objects (for example, multiple gigabytes) in the database. A BLOB can be either stored internally as a column, or stored in an external file.

Data Encryption Storage with the DBMS_CRYPTO Package

The `DBMS_CRYPTO` package provides several ways to address security issues.

While encryption is not the ideal solution for addressing several security threats, it is clear that selectively encrypting sensitive data before storage in the database does improve security. Examples of such data could include:

- Credit card numbers
- National identity numbers

Oracle Database provides the PL/SQL package `DBMS_CRYPTO` to encrypt and decrypt stored data. This package supports several industry-standard encryption and hashing algorithms, including the Advanced Encryption Standard (AES) encryption algorithm. AES was approved by the National Institute of Standards and Technology (NIST) to replace the Data Encryption Standard (DES).

The `DBMS_CRYPTO` package enables encryption and decryption for common Oracle Database data types, including `RAW` and large objects (LOBs), such as images and sound. Specifically, it supports BLOBs and CLOBs. In addition, it provides Globalization Support for encrypting data across different database character sets.

The following cryptographic algorithms are supported:

- Advanced Encryption Standard (AES)
- SHA-2 Cryptographic Hash settings:
  - `HASH_SH256`
  - `HASH_SH384`
  - `HASH_SH512`
- SHA-2 Message Authentication Code (MAC)

Block cipher modifiers are also provided with `DBMS_CRYPTO`. You can choose from several padding options, including Public Key Cryptographic Standard (PKCS) #5, and from four block cipher chaining modes, including Cipher Block Chaining (CBC). Padding must be done in multiples of eight bytes.

Note:

- DES is no longer recommended by the National Institute of Standards and Technology (NIST).
- Usage of SHA-1 is more secure than MD5.
- Usage of SHA-2 is more secure than SHA-1.
- Keyed MD5 is not vulnerable.

Table 14-1 summarizes the `DBMS_CRYPTO` package features.

Table 14-1 DBMS_CRYPTO Package Feature Summary

Feature	DBMS_CRYPTO Supported Functionality
Cryptographic algorithms	AES
Padding forms	PKCS5, zeroes
Block cipher chaining modes	CBC, CFB, ECB, OFB
Cryptographic hash algorithms	SHA-1, SHA-2, MD4, MD5, HASH_SH256, HASH_SH384, HASH_SH512
Keyed hash (MAC) algorithms	HMAC_MD5, HMAC_SH1, HMAC_SH256, HMAC_SH384, HMAC_SH512
Cryptographic pseudo-random number generator	`RAW`, `NUMBER`, `BINARY_INTEGER`
Database types	`RAW`, `CLOB`, `BLOB`

`DBMS_CRYPTO` supports a range of algorithms that accommodate both new and existing systems. Although 3DES_2KEY and MD4 are provided for backward compatibility, you achieve better security using 3DES, AES, or SHA-1. Therefore, 3DES_2KEY is not recommended.

The `DBMS_CRYPTO` package includes cryptographic checksum capabilities (MD5), which are useful for comparisons, and the ability to generate a secure random number (the `RANDOMBYTES` function). Secure random number generation is an important part of cryptography; predictable keys are easily guessed keys; and easily guessed keys may lead to easy decryption of data. Most cryptanalysis is done by finding weak keys or poorly stored keys, rather than through brute force analysis (cycling through all possible keys).

Note:

Do not use `DBMS_RANDOM`, because it is unsuitable for cryptographic key generation.

Key management is programmatic. That is, the application (or caller of the function) must supply the encryption key. This means that the application developer must find a way of storing and retrieving keys securely. The relative strengths and weaknesses of various key management techniques are discussed in the sections that follow. The DES algorithm itself has an effective key length of 56-bits.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DBMS_CRYPTO` package
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `UTL_RAW` package

Using Ciphertexts Encrypted in OFB Mode in Oracle Database Release 11g

In Oracle Database Release 11g, ciphertexts configured to use output feedback (OFB) used electronic codebook (ECB) mode instead.

In Oracle Database Release 11*g*, if you set the `DBMS_CRYPTO.CHAIN_OFB` block cipher chaining modifier to configure ciphertext encryption to use output feedback (OFB) mode, then due to Oracle Bug 13001552, the result is that the configuration used electronic codebook (ECB) mode erroneously. This bug has been fixed in Oracle Database Release 12c. Therefore, after an upgrade from Oracle Database release 11g to Release 12c, the ciphertexts that were encrypted using OFB mode in release 11g will no longer decrypt properly in the corrected OFB mode in Oracle Database Release 12c or later.
To remedy this problem:

1. Log in to the database as a user who has the `EXECUTE` privilege for the `DBMS_CRYPTO` PL/SQL package.
2. Decrypt the cyphertexts using the `DBMS_CRYPTO.CHAIN_ECB` block cipher chaining modifier.

The following example, `dbmscrypto11.sql`, shows the wrong behavior in Oracle Database Release 11*g*:

```
dbmscrypto11.sql:
set serveroutput on

declare
  l_mod_ofb pls_integer;
  l_mod_ecb pls_integer;
  v_key raw(32);
  v_iv  raw(16);
  v_test_in raw(16);
  v_ciphertext raw(16);
  v_test_out_ECB raw(16);
  v_test_out_OFB raw(16);
begin
  l_mod_ofb := dbms_crypto.ENCRYPT_AES256
       + dbms_crypto.CHAIN_OFB
       + DBMS_CRYPTO.PAD_NONE ;
  l_mod_ecb := dbms_crypto.ENCRYPT_AES256
       + dbms_crypto.CHAIN_ECB
       + DBMS_CRYPTO.PAD_NONE ;

  v_key := hextoraw
  ('603deb1015ca71be2b73aef0857d77811f352c073b6108d72d9810a30914dff4');
  v_iv :=   hextoraw('000102030405060708090A0B0C0D0E0F');
  v_test_in := hextoraw('6bc1bee22e409f96e93d7e117393172a');
  v_ciphertext := dbms_crypto.encrypt(src => v_test_in,
                                  TYP => l_mod_ofb,
                                  key => v_key,
                                  iv => v_iv);
  v_test_out_ECB := dbms_crypto.decrypt(src => v_ciphertext,
                                  TYP => l_mod_ecb,
                                  key => v_key,
                                  iv => v_iv);
  v_test_out_OFB := dbms_crypto.decrypt(src => v_ciphertext,
                                  TYP => l_mod_ofb,
                                  key => v_key,
                                  iv => v_iv);
  dbms_output.put_line
  ('Input plaintext                     : '||rawtohex(v_test_in));
  dbms_output.put_line
  ('11g: Ciphertext (encrypt in OFB mode): '||rawtohex(v_ciphertext));
  dbms_output.put_line
  ('11g: Output of decrypt in ECB mode   : '||rawtohex(v_test_out_ECB));
  dbms_output.put_line
  ('11g: Output of decrypt in OFB mode   : '||rawtohex(v_test_out_OFB));
end;
/
```

The resulting output is as follows:

```
SQL> @dbmscrypto11.sql

Input plaintext                      : 6BC1BEE22E409F96E93D7E117393172A
11g: Ciphertext (encrypt in OFB mode): F3EED1BDB5D2A03C064B5A7E3DB181F8
11g: Output of decrypt in ECB mode   : 6BC1BEE22E409F96E93D7E117393172A
11g: Output of decrypt in OFB mode   : 6BC1BEE22E409F96E93D7E117393172A
```

This output illustrates that in Oracle Database release 11*g*, OFB mode is wrongly ECB mode, and therefore decrypting in either OFB or ECB mode results in the correct plaintext.

The next example, `dbmscrypto12from11.sql`, shows that, after an upgrade from Oracle Database release 11*g* to release 12*c*, ECB mode and not OFB mode has to be used in order to properly decrypt a ciphertext encrypted in OFB mode in Release 11*g*.

```
dbmscrypto12from11.sql:
set serveroutput on

declare
  l_mod_ofb pls_integer;
  l_mod_ecb pls_integer;
  v_key raw(32);
  v_iv  raw(16);
  v_test_in raw(16);
  v_ciphertext11 raw(16);
  v_test_out_ECB raw(16);
  v_test_out_OFB raw(16);
begin
  l_mod_ofb := dbms_crypto.ENCRYPT_AES256
       + dbms_crypto.CHAIN_OFB
```

```
        + DBMS_CRYPTO.PAD_NONE ;
  l_mod_ecb := dbms_crypto.ENCRYPT_AES256
        + dbms_crypto.CHAIN_ECB
        + DBMS_CRYPTO.PAD_NONE ;

  v_key := hextoraw
  ('603deb1015ca71be2b73aef0857d77811f352c073b6108d72d9810a30914dff4');
  v_iv :=   hextoraw('000102030405060708090A0B0C0D0E0F');
  v_test_in := hextoraw('6bc1bee22e409f96e93d7e117393172a');
  v_ciphertext11 := hextoraw('F3EED1BDB5D2A03C064B5A7E3DB181F8');

  v_test_out_ECB := dbms_crypto.decrypt(src => v_ciphertext11,
                                TYP => l_mod_ecb,
                                key => v_key,
                                iv => v_iv);
  v_test_out_OFB := dbms_crypto.decrypt(src => v_ciphertext11,
                                TYP => l_mod_ofb,
                                key => v_key,
                                iv => v_iv);
  dbms_output.put_line
  ('Input plaintext (to 11g)             : '||rawtohex(v_test_in));
  dbms_output.put_line
  ('11g: Ciphertext (encrypt in OFB mode): '||rawtohex(v_ciphertext11));
  dbms_output.put_line
  ('12c: Output of decrypt in ECB mode   : '||rawtohex(v_test_out_ECB));
  dbms_output.put_line
  ('12c: Output of decrypt in OFB mode   : '||rawtohex(v_test_out_OFB));
end;
/
```

The resulting output is as follows:

```
SQL> @dbmscrypto12from11.sql
Input plaintext (to 11g)             : 6BC1BEE22E409F96E93D7E117393172A
11g: Ciphertext (encrypt in OFB mode): F3EED1BDB5D2A03C064B5A7E3DB181F8
12c: Output of decrypt in ECB mode   : 6BC1BEE22E409F96E93D7E117393172A
12c: Output of decrypt in OFB mode   : 4451EBE041EB29E191BBA0E9D67FAEB2
```

If you are preparing to upgrade from Oracle Database Release 11*g* to Release 12*c*, then edit any scripts that you may have in which OFB mode is specified so that the decrypt operations use ECB mode. This way, the scripts will work in both release 11g and release 12c and later, ensuring business continuity.

Examples of Using the Data Encryption API

Examples of using the data encryption API include using the `DBMS_CRYPTO.SQL` procedure, encrypting AES 256-bit data, and encrypting BLOB data.

Example: Data Encryption Procedure

The `DBMS_CRYPTO.SQL` PL/SQL program can be used to encrypt data.

This example code performs the following actions:

- Encrypts a string (`VARCHAR2` type) using DES after first converting it into the `RAW` data type.

  This step is necessary because encrypt and decrypt functions and procedures in `DBMS_CRYPTO` package work on the `RAW` data type only.

- Shows how to create a 160-bit hash using SHA-1 algorithm.
- Demonstrates how MAC, a key-dependent one-way hash, can be computed using the MD5 algorithm.

The `DBMS_CRYPTO.SQL` procedure follows:

```
DECLARE
    input_string     VARCHAR2(16) := 'tigertigertigert';
    raw_input        RAW(128) :=
UTL_RAW.CAST_TO_RAW(CONVERT(input_string,'AL32UTF8','US7ASCII'));
    key_string       VARCHAR2(8)  := 'scottsco';
    raw_key          RAW(128) :=
UTL_RAW.CAST_TO_RAW(CONVERT(key_string,'AL32UTF8','US7ASCII'));
    encrypted_raw    RAW(2048);
    encrypted_string VARCHAR2(2048);
    decrypted_raw    RAW(2048);
    decrypted_string VARCHAR2(2048);
-- Begin testing Encryption:
BEGIN
    dbms_output.put_line('> Input String                     : ' ||
    CONVERT(UTL_RAW.CAST_TO_VARCHAR2(raw_input),'US7ASCII','AL32UTF8'));
    dbms_output.put_line('> ========= BEGIN TEST Encrypt =========');
    encrypted_raw := dbms_crypto.Encrypt(
        src => raw_input,
        typ => DBMS_CRYPTO.DES_CBC_PKCS5,
        key => raw_key);
        dbms_output.put_line('> Encrypted hex value              : ' ||
        rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_raw)));
decrypted_raw := dbms_crypto.Decrypt(
        src => encrypted_raw,
        typ => DBMS_CRYPTO.DES_CBC_PKCS5,
        key => raw_key);
    decrypted_string :=
    CONVERT(UTL_RAW.CAST_TO_VARCHAR2(decrypted_raw),'US7ASCII','AL32UTF8');
dbms_output.put_line('> Decrypted string output          : ' ||
        decrypted_string);
if input_string = decrypted_string THEN
    dbms_output.put_line('> String DES Encyption and Decryption successful');
END if;
dbms_output.put_line('');
dbms_output.put_line('> ========= BEGIN TEST Hash =========');
    encrypted_raw := dbms_crypto.Hash(
        src => raw_input,
        typ => DBMS_CRYPTO.HASH_SH1);
dbms_output.put_line('> Hash value of input string       : ' ||
        rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_raw)));
dbms_output.put_line('> ========= BEGIN TEST Mac =========');
    encrypted_raw := dbms_crypto.Mac(
        src => raw_input,
        typ => DBMS_CRYPTO.HMAC_MD5,
        key => raw_key);
dbms_output.put_line('> Message Authentication Code      : ' ||
        rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_raw)));
dbms_output.put_line('');
dbms_output.put_line('> End of DBMS_CRYPTO tests  ');
END;
/
```

Example: AES 256-Bit Data Encryption and Decryption Procedures

You can use a PL/SQL block to encrypt and decrypt a predefined variable.

For the following example, the predefined variable is named `input_string` and it uses the AES 256-bit algorithm with Cipher Block Chaining and PKCS #5 padding:

```
declare
   input_string       VARCHAR2 (200) := 'Secret Message';
   output_string      VARCHAR2 (200);
   encrypted_raw      RAW (2000);             -- stores encrypted binary text
   decrypted_raw      RAW (2000);             -- stores decrypted binary text
   num_key_bytes      NUMBER := 256/8;        -- key length 256 bits (32 bytes)
   key_bytes_raw      RAW (32);               -- stores 256-bit encryption key
   encryption_type    PLS_INTEGER :=          -- total encryption type
                            DBMS_CRYPTO.ENCRYPT_AES256
                          + DBMS_CRYPTO.CHAIN_CBC
                          + DBMS_CRYPTO.PAD_PKCS5;
begin
   DBMS_OUTPUT.PUT_LINE ('Original string: ' || input_string);
   key_bytes_raw := DBMS_CRYPTO.RANDOMBYTES (num_key_bytes);
   encrypted_raw := DBMS_CRYPTO.ENCRYPT
      (
         src => UTL_I18N.STRING_TO_RAW (input_string, 'AL32UTF8'),
         typ => encryption_type,
         key => key_bytes_raw
      );
    -- The encrypted value in the encrypted_raw variable can be used here:
   decrypted_raw := DBMS_CRYPTO.DECRYPT
      (
         src => encrypted_raw,
         typ => encryption_type,
         key => key_bytes_raw
      );
   output_string := UTL_I18N.RAW_TO_CHAR (decrypted_raw, 'AL32UTF8');
   DBMS_OUTPUT.PUT_LINE ('Decrypted string: ' || output_string);
end;
```

Example: Encryption and Decryption Procedures for BLOB Data

You can encrypt BLOB data.

The following sample PL/SQL program (`blob_test.sql`) shows how to encrypt and decrypt BLOB data. This example code does the following, and prints out its progress (or problems) at each step:

- Creates a table for the BLOB column
- Inserts the raw values into that table
- Encrypts the raw data
- Decrypts the encrypted data

The `blob_test.sql` procedure follows:

```
-- 1. Create a table for BLOB column:
create table table_lob (id number, loc blob);

-- 2. Insert 3 empty lobs for src/enc/dec:
```

```
insert into table_lob values (1, EMPTY_BLOB());
insert into table_lob values (2, EMPTY_BLOB());
insert into table_lob values (3, EMPTY_BLOB());

set echo on
set serveroutput on

declare
    srcdata    RAW(1000);
    srcblob    BLOB;
    encrypblob BLOB;
    encrypraw  RAW(1000);
    encrawlen  BINARY_INTEGER;
    decrypblob BLOB;
    decrypraw  RAW(1000);
    decrawlen  BINARY_INTEGER;

    leng       INTEGER;

begin

    -- RAW input data 16 bytes
    srcdata := hextoraw('6D6D6D6D6D6D6D6D6D6D6D6D6D6D6D6D');

    dbms_output.put_line('---');
    dbms_output.put_line('input is ' || srcdata);
    dbms_output.put_line('---');

    -- select empty lob locators for src/enc/dec
    select loc into srcblob from table_lob where id = 1;
    select loc into encrypblob from table_lob where id = 2;
    select loc into decrypblob from table_lob where id = 3;

    dbms_output.put_line('Created Empty LOBS');
    dbms_output.put_line('---');

    leng := DBMS_LOB.GETLENGTH(srcblob);
    IF leng IS NULL THEN
        dbms_output.put_line('Source BLOB Len NULL ');
    ELSE
        dbms_output.put_line('Source BLOB Len ' || leng);
    END IF;

    leng := DBMS_LOB.GETLENGTH(encrypblob);
    IF leng IS NULL THEN
        dbms_output.put_line('Encrypt BLOB Len NULL ');
    ELSE
        dbms_output.put_line('Encrypt BLOB Len ' || leng);
    END IF;

    leng := DBMS_LOB.GETLENGTH(decrypblob);
    IF leng IS NULL THEN
        dbms_output.put_line('Decrypt  BLOB Len NULL ');
    ELSE
        dbms_output.put_line('Decrypt BLOB Len ' || leng);
    END IF;

    -- 3. Write source raw data into blob:
    DBMS_LOB.OPEN (srcblob, DBMS_LOB.lob_readwrite);
    DBMS_LOB.WRITEAPPEND (srcblob, 16, srcdata);
    DBMS_LOB.CLOSE (srcblob);
    dbms_output.put_line('Source raw data written to source blob');
    dbms_output.put_line('---');

    leng := DBMS_LOB.GETLENGTH(srcblob);
    IF leng IS NULL THEN
        dbms_output.put_line('source BLOB Len NULL ');
    ELSE
        dbms_output.put_line('Source BLOB Len ' || leng);
    END IF;

    /*
    * Procedure Encrypt
    * Arguments: srcblob -> Source BLOB
    *            encrypblob -> Output BLOB for encrypted data
    *            DBMS_CRYPTO.AES_CBC_PKCS5 -> Algo : AES
    *                                         Chaining : CBC
    *                                         Padding : PKCS5
    *            256 bit key for AES passed as RAW
    *                -> 
    hextoraw('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F')
    *            IV (Initialization Vector) for AES algo passed as RAW
    *                -> hextoraw('00000000000000000000000000000000')
    */

    DBMS_CRYPTO.Encrypt(encrypblob,
                srcblob,
                DBMS_CRYPTO.AES_CBC_PKCS5,
                hextoraw
('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F'),
                hextoraw('00000000000000000000000000000000'));


    dbms_output.put_line('Encryption Done');
    dbms_output.put_line('---');

    leng := DBMS_LOB.GETLENGTH(encrypblob);
    IF leng IS NULL THEN
        dbms_output.put_line('Encrypt BLOB Len NULL');
    ELSE
        dbms_output.put_line('Encrypt BLOB Len ' || leng);
    END IF;

    -- 4. Read encrypblob to a raw:
    encrawlen := 999;

    DBMS_LOB.OPEN (encrypblob, DBMS_LOB.lob_readwrite);
    DBMS_LOB.READ (encrypblob, encrawlen, 1, encrypraw);
    DBMS_LOB.CLOSE (encrypblob);

    dbms_output.put_line('Read encrypt blob to a raw');
    dbms_output.put_line('---');

    dbms_output.put_line('Encrypted data is (256 bit key) ' || encrypraw);
    dbms_output.put_line('---');

    /*
    * Procedure Decrypt
    * Arguments: encrypblob -> Encrypted BLOB to decrypt
    *            decrypblob -> Output BLOB for decrypted data in RAW
    *            DBMS_CRYPTO.AES_CBC_PKCS5 -> Algo : AES
    *                                          Chaining : CBC
    *                                          Padding : PKCS5
    *            256 bit key for AES passed as RAW (same as used during Encrypt)
    *                ->
    hextoraw('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F')
    *            IV (Initialization Vector) for AES algo passed as RAW (same as
                 used during Encrypt)
    *                -> hextoraw('00000000000000000000000000000000')
    */

    DBMS_CRYPTO.Decrypt(decrypblob,
                encrypblob,
                DBMS_CRYPTO.AES_CBC_PKCS5,
                hextoraw
           ('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F'),
                hextoraw('00000000000000000000000000000000'));

    leng := DBMS_LOB.GETLENGTH(decrypblob);
    IF leng IS NULL THEN
        dbms_output.put_line('Decrypt BLOB Len NULL');
    ELSE
        dbms_output.put_line('Decrypt BLOB Len ' || leng);
    END IF;

    -- Read decrypblob to a raw
    decrawlen := 999;

    DBMS_LOB.OPEN (decrypblob, DBMS_LOB.lob_readwrite);
    DBMS_LOB.READ (decrypblob, decrawlen, 1, decrypraw);
    DBMS_LOB.CLOSE (decrypblob);

    dbms_output.put_line('Decrypted data is (256 bit key) ' || decrypraw);
    dbms_output.put_line('---');

    DBMS_LOB.OPEN (srcblob, DBMS_LOB.lob_readwrite);
    DBMS_LOB.TRIM (srcblob, 0);
    DBMS_LOB.CLOSE (srcblob);

    DBMS_LOB.OPEN (encrypblob, DBMS_LOB.lob_readwrite);
    DBMS_LOB.TRIM (encrypblob, 0);
    DBMS_LOB.CLOSE (encrypblob);

    DBMS_LOB.OPEN (decrypblob, DBMS_LOB.lob_readwrite);
    DBMS_LOB.TRIM (decrypblob, 0);
    DBMS_LOB.CLOSE (decrypblob);

end;
/

truncate table table_lob;
drop table table_lob;
```

Data Dictionary Views for Encrypted Data

Oracle Database provides data dictionary views to find information about encrypted data.

Table 14-2 lists these data dictionary views.

Table 14-2 Data Dictionary Views That Display Information about Encrypted Data

View	Description
`ALL_ENCRYPTED_COLUMNS`	Describes encryption algorithm information for all encrypted columns in all tables accessible to the user
`DBA_ENCRYPTED_COLUMNS`	Describes encryption algorithm information for all encrypted columns in the database
`USER_ENCRYPTED_COLUMNS`	Describes encryption algorithm information for all encrypted columns in all tables in the schema of the user
`V$ENCRYPTED_TABLESPACES`	Displays information about the current pluggable database (PDB) tablespaces that are encrypted
`V$ENCRYPTION_WALLET`	Displays information on the status of the wallet and the wallet location for Transparent Data Encryption; applies to the current PDB only
`V$RMAN_ENCRYPTION_ALGORITHMS`	Displays supported encryption algorithms for the current PDB

See Also:

Oracle Database Reference for detailed information about these views

Part IV
Securing Data on the Network

Part IV describes how to secure data on the network.

15

Configuring Oracle Database Network Encryption and Data Integrity

You can configure native Oracle Net Services data encryption and data integrity for both servers and clients.

About Oracle Database Network Encryption and Data Integrity

Oracle Database enables you to encrypt data that is sent over a network.

About Oracle Data Network Encryption and Integrity

Oracle Database provides data network encryption and integrity to ensure that data is secure as it travels across the network.

The purpose of a secure cryptosystem is to convert plaintext data into unintelligible ciphertext based on a key, in such a way that it is very hard (computationally infeasible) to convert ciphertext back into its corresponding plaintext without knowledge of the correct key.

In a symmetric cryptosystem, the same key is used both for encryption and decryption of the same data. Oracle Database provides the Advanced Encryption Standard (AES) symmetric cryptosystem for protecting the confidentiality of Oracle Net Services traffic.

Advanced Encryption Standard

Oracle Database supports the Federal Information Processing Standard (FIPS) encryption algorithm, Advanced Encryption Standard (AES).

AES can be used by all U.S. government organizations and businesses to protect sensitive data over a network. This encryption algorithm defines three standard key lengths, which are 128-bit, 192-bit, and 256-bit. All versions operate in outer Cipher Block Chaining (CBC) mode.

ARIA

Oracle Database supports the Academia, Research Institute, and Agency (ARIA) algorithm.

This algorithm acknowledges the cooperative efforts of Korean researchers in designing the algorithm.

ARIA defines three standard key lengths, which are 128-bit, 192-bit, and 256-bit. All versions operate in outer cipher Cipher Block Chaining (CBC) mode.

GOST

Oracle Database supports the GOsudarstvennyy STandart (GOST) algorithm.

The GOST algorithm was created by the Euro-Asian Council for Standardization, Metrology and Certification (EACS).

GOST defines a key size of 256-bits. In Oracle Database, outer Cipher Block Chaining (CBC) mode is used.

SEED

Oracle Database supports the Korea Information Security Agency (KISA) encryption algorithm, SEED.

SEED defines a key size of 128-bits. There are extensions to the standard that defines additional key sizes of 192- and 256-bits, but Oracle Database does not support these extensions. In the Oracle Database, SEED operates in outer Cipher Block Chaining (CBC) mode.

Triple-DES Support

Oracle Database supports Triple-DES encryption (3DES), which encrypts message data with three passes of the DES algorithm.

3DES provides a high degree of message security, but with a performance penalty. The magnitude of the performance penalty depends on the speed of the processor performing the encryption. 3DES typically takes three times as long to encrypt a data block when compared to the standard DES algorithm.

3DES is available in two-key and three-key versions, with effective key lengths of 112-bits and 168-bits, respectively. Both versions operate in outer Cipher Block Chaining (CBC) mode.

The DES40 algorithm, available with Oracle Database and Secure Network Services, is a variant of DES in which the secret key is preprocessed to provide 40 effective key bits. It was designed to provide DES-based encryption to customers outside the U.S. and Canada at a time when the U.S. export laws were more restrictive. Currently DES40, DES, and 3DES are all available for export. DES40 is still supported to provide backward-compatibility for international customers.

Oracle Database Network Encryption Data Integrity

Encrypting network data provides data privacy so that unauthorized parties cannot view plaintext data as it passes over the network.

Oracle Database also provides protection against two forms of active attacks.

Table 15-1 provides information about these attacks.

Table 15-1 Two Forms of Network Attacks

Type of Attack	Explanation
Data modification attack	An unauthorized party intercepting data in transit, altering it, and retransmitting it is a data modification attack. For example, intercepting a $100 bank deposit, changing the amount to $10,000, and retransmitting the higher amount is a data modification attack.
Replay attack	Repetitively retransmitting an entire set of valid data is a replay attack, such as intercepting a $100 bank withdrawal and retransmitting it ten times, thereby receiving $1,000.

Data Integrity Algorithms Support

A keyed, sequenced implementation of the Message Digest 5 (MD5) algorithm or the Secure Hash Algorithm (SHA-1 and SHA-2) protect against these attacks.

Both of these hash algorithms create a checksum that changes if the data is altered in any way. This protection operates independently from the encryption process so you can enable data integrity with or without enabling encryption.

Diffie-Hellman Based Key Negotiation

You can use the Diffie-Hellman key negotiation algorithm to secure data in a multiuser environment.

Secure key distribution is difficult in a multiuser environment. Oracle Database uses the well known to perform secure key distribution for both encryption and data integrity.

When encryption is used to protect the security of encrypted data, keys must be changed frequently to minimize the effects of a compromised key. Accordingly, the Oracle Database key management function changes the session key with every session.

You can use Authentication Key Fold-in to defeat a possible third-party attack (historically called the *man-in-the-middle attack*) on the Diffie-Hellman key negotiation algorithm key negotiation. It strengthens the session key significantly by combining a shared secret, known only to the client and the server, with the original session key negotiated by Diffie-Hellman.

The client and the server begin communicating using the session key generated by Diffie-Hellman. When the client authenticates to the server, they establish a shared secret that is only known to both parties. Oracle Database combines the shared secret and the Diffie-Hellman session key to generate a stronger session key designed to defeat a man-in-the-middle attack.

Note:

The authentication key fold-in function is an imbedded feature of Oracle Database and requires no configuration by the system or network administrator.

Configuration of Data Encryption and Integrity

Oracle Database native Oracle Net Services encryption and integrity presumes the prior installation of Oracle Net Services.

About Activating Encryption and Integrity

In any network connection, both the client and server can support multiple encryption algorithms and integrity algorithms.

When a connection is made, the server selects which algorithm to use, if any, from those algorithms specified in the `sqlnet.ora` files.The server searches for a match between the algorithms available on both the client and the server, and picks the first algorithm in its own list that also appears in the client list. If one side of the connection does not specify an algorithm list, all the algorithms installed on that side are acceptable. The connection fails with error message `ORA-12650` if either side specifies an algorithm that is not installed.

Encryption and integrity parameters are defined by modifying a `sqlnet.ora` file on the clients and the servers on the network.

You can choose to configure any or all of the available encryption algorithms, and either or both of the available integrity algorithms. Only one encryption algorithm and one integrity algorithm are used for each connect session.

Note:

Oracle Database selects the first encryption algorithm and the first integrity algorithm enabled on the client and the server. Oracle recommends that you select algorithms and key lengths in the order in which you prefer negotiation, choosing the strongest key length first.

See Also:

- Table 15-3 for a listing of valid encryption algorithms
- *Oracle Database Advanced Security Guide* for a listing of available integrity algorithms
- Data Encryption and Integrity Parameters

About Negotiating Encryption and Integrity

The `sqlnet.ora` file on systems using data encryption and integrity must contain some or all the `REJECTED`, `ACCEPTED`, `REQUESTED`, and `REQUIRED` parameters.

About the Values for Negotiating Encryption and Integrity

Oracle Net Manager can be used to specify four possible values for the encryption and integrity configuration parameters.

The following four values are listed in the order of increasing security, and they must be used in the profile file (`sqlnet.ora`) for the client and server of the systems that are using encryption and integrity.

The value `REJECTED` provides the *minimum* amount of security between client and server communications, and the value `REQUIRED` provides the *maximum* amount of network security:

- `REJECTED`
- `ACCEPTED`
- `REQUESTED`
- `REQUIRED`

The default value for each of the parameters is `ACCEPTED`.

Oracle Database servers and clients are set to `ACCEPT` encrypted connections out of the box. This means that you can enable the desired encryption and integrity settings for a connection pair by configuring just one side of the connection, server-side or client-side.

So, for example, if there are many Oracle clients connecting to an Oracle database, you can configure the required encryption and integrity settings for all these connections by making the appropriate sqlnet.ora changes at the server end. You do not need to implement configuration changes for each client separately.

Table 15-2 shows whether the security service is enabled, based on a combination of client and server configuration parameters. If either the server or client has specified `REQUIRED`, the lack of a common algorithm *causes the connection to fail.* Otherwise, if the service is enabled, lack of a common service algorithm results in the service being *disabled*.

Table 15-2 Encryption and Data Integrity Negotiations

Client Setting	Server Setting	Encryption and Data Negotiation
`REJECTED`	`REJECTED`	`OFF`
`ACCEPTED`	`REJECTED`	`OFF`
`REQUESTED`	`REJECTED`	`OFF`
`REQUIRED`	`REJECTED`	Connection fails
`REJECTED`	`ACCEPTED`	`OFF`
`ACCEPTED`	`ACCEPTED`	`OFF`[1]
`REQUESTED`	`ACCEPTED`	`ON`

Table 15-2 (Cont.) Encryption and Data Integrity Negotiations

Client Setting	Server Setting	Encryption and Data Negotiation
`REQUIRED`	`ACCEPTED`	`ON`
`REJECTED`	`REQUESTED`	`OFF`
`ACCEPTED`	`REQUESTED`	`ON`
`REQUESTED`	`REQUESTED`	`ON`
`REQUIRED`	`REQUESTED`	`ON`
`REJECTED`	`REQUIRED`	Connection fails
`ACCEPTED`	`REQUIRED`	`ON`
`REQUESTED`	`REQUIRED`	`ON`
`REQUIRED`	`REQUIRED`	`ON`

1 This value defaults to `OFF`. Cryptography and data integrity are not enabled until the user changes this parameter by using Oracle Net Manager or by modifying the `sqlnet.ora` file.

REJECTED Configuration Parameter

The `REJECTED` value disables the security service, even if the other side requires this service.

In this scenario, this side of the connection specifies that the security service is not permitted. If the other side is set to `REQUIRED`, the connection *terminates* with error message `ORA-12650`. If the other side is set to `REQUESTED`, `ACCEPTED`, or `REJECTED`, the connection continues without error and without the security service enabled.

ACCEPTED Configuration Parameter

The `ACCEPTED` value enables the security service if the other side requires or requests the service.

In this scenario, this side of the connection does not require the security service, but it is enabled if the other side is set to `REQUIRED` or `REQUESTED`. If the other side is set to `REQUIRED` or `REQUESTED`, and an encryption or integrity algorithm match is found, the connection continues without error and with the security service enabled. If the other side is set to `REQUIRED` and no algorithm match is found, the connection terminates with error message `ORA-12650`.

If the other side is set to `REQUESTED` and no algorithm match is found, or if the other side is set to `ACCEPTED` or `REJECTED`, the connection continues without error and without the security service enabled.

REQUESTED Configuration Parameter

The `REQUESTED` value enables the security service if the other side permits this service.

In this scenario, this side of the connection specifies that the security service is desired but not required. The security service is enabled if the other side specifies `ACCEPTED`, `REQUESTED`, or `REQUIRED`. There must be a matching algorithm available on the other side, otherwise the service is not enabled. If the other side specifies `REQUIRED` and there is no matching algorithm, *the connection fails.*

REQUIRED Configuration Parameter

The `REQUIRED` value enables the security service or preclude the connection.

In this scenario, this side of the connection specifies that the security service *must be enabled*. The connection *fails* if the other side specifies `REJECTED` or if there is no compatible algorithm on the other side.

Configuring Encryption and Integrity Parameters Using Oracle Net Manager

You can set up or change encryption and integrity parameter settings using Oracle Net Manager.

Configuring Encryption on the Client and the Server

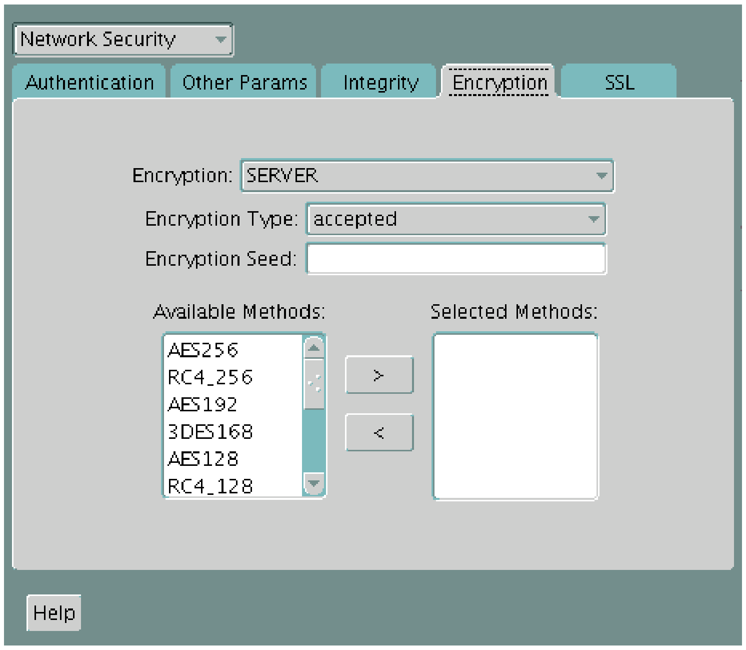
Use Oracle Net Manager to configure encryption on the client and on the server.

1. Start Oracle Net Manager.
   - (UNIX) From `$ORACLE_HOME/bin`, enter the following command at the command line:

     ```
     netmgr
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **Encryption** tab.



Encryption: SERVER
Encryption Type: accepted
Encryption Seed:

Available Methods:
AES256
RC4_256
AES192
3DES168
AES128
RC4_128

Selected Methods:

5. Select **CLIENT** or **SERVER** option from the **Encryption** box.
6. From the Encryption Type list, select one of the following:
   - **REQUESTED**
   - **REQUIRED**
   - **ACCEPTED**
   - **REJECTED**
7. (Optional) In the **Encryption Seed** field, enter between 10 and 70 random characters. The encryption seed for the client should not be the same as that for the server.
8. Select an encryption algorithm in the **Available Methods** list. Move it to the **Selected Methods** list by choosing the right arrow (**>**). Repeat for each additional method you want to use.
9. Select **File**, **Save Network Configuration**. The `sqlnet.ora` file is updated.
10. Repeat this procedure to configure encryption on the other system. The `sqlnet.ora` file on the two systems should contain the following entries:
    - On the server:

      ```
      SQLNET.ENCRYPTION_SERVER = [accepted | rejected | requested | required]
      SQLNET.ENCRYPTION_TYPES_SERVER = (valid_encryption_algorithm
      [,valid_encryption_algorithm])
      ```

    - On the client:

      ```
      SQLNET.ENCRYPTION_CLIENT = [accepted | rejected | requested | required]
      SQLNET.ENCRYPTION_TYPES_CLIENT = (valid_encryption_algorithm
      [,valid_encryption_algorithm])
      ```

Table 15-3 lists valid encryption algorithms and their associated legal values.

Table 15-3 Valid Encryption Algorithms

Algorithm Name	Legal Value
AES 256-bit key	AES256
AES 192-bit key	AES192
AES 128-bit key	AES128

Configuring Integrity on the Client and the Server

You can use Oracle Net Manager to configure network integrity on both the client and the server.

1. Start Oracle Net Manager.
   - (UNIX) From *$ORACLE_HOME*`/bin`, enter the following command at the command line:

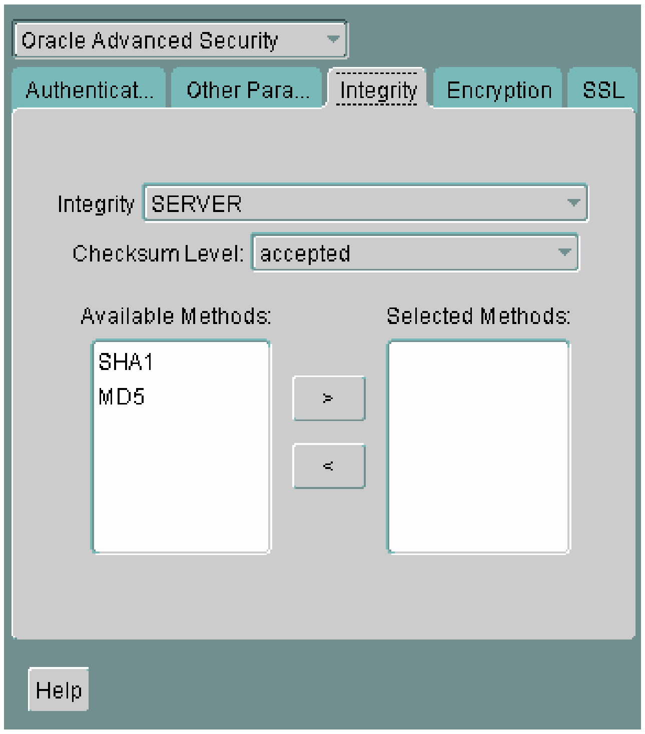
     ```
     netmgr
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.

3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.

4. Select the **Integrity** tab.



Oracle Advanced Security

Integrity: SERVER

Checksum Level: accepted

Available Methods: SHA1, MD5

Selected Methods:

5. Depending upon which system you are configuring, select the **Server** or **Client** from the **Integrity** box.

6. From the **Checksum Level** list, select one of the following checksum level values:
   - **REQUESTED**
   - **REQUIRED**
   - **ACCEPTED**
   - **REJECTED**

7. Select an integrity algorithm in the **Available Methods** list. Move it to the **Selected Methods** list by choosing the right arrow (**>**). Repeat for each additional method you want to use.

8. Select **File**, **Save Network Configuration**.

   The `sqlnet.ora` file is updated.

9. Repeat this procedure to configure integrity on the other system.

   The `sqlnet.ora` file on the two systems should contain the following entries:

   - On the server:

     ```
     SQLNET.CRYPTO_CHECKSUM_SERVER = [accepted | rejected | requested | required]
     SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER = (valid_crypto_checksum_algorithm
     [,valid_crypto_checksum_algorithm])
     ```

   - On the client:

```
SQLNET.CRYPTO_CHECKSUM_CLIENT = [accepted | rejected | requested | required]
SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT = (valid_crypto_checksum_algorithm
[,valid_crypto_checksum_algorithm])
```

See Also:

Oracle Database Advanced Security Guide for a list of the supported encryption and integrity algorithms

16

Configuring the Thin JDBC Client Network

Oracle Database network encryption and strong authentication enables thin Java Database Connectivity (JDBC) clients to securely connect to Oracle databases.

About the Java Implementation

Oracle Database provides a Java implementation of network encryption and strong authentication.

The Java implementation of Oracle Database network encryption and strong authentication provides network authentication, encryption and integrity protection for Thin JDBC clients that must communicate with Oracle Databases that have Oracle Database network encryption and strong authentication configured.

See Also:

Oracle Database JDBC Developer's Guide for information about JDBC, including examples

Java Database Connectivity Support

JDBC, an industry-standard Java interface, is a Java standard for connecting to a relational database from a Java program.

Sun Microsystems defined the JDBC standard and Oracle implements and extends the standard with its own JDBC drivers.

Oracle JDBC drivers are used to create Java Database Connectivity (JDBC) applications to communicate with Oracle databases. Oracle implements two types of JDBC drivers: Thick JDBC drivers built on top of the C-based Oracle Net client, as well as a Thin (Pure Java) JDBC driver to support downloadable applets. Oracle extensions to JDBC include the following features:

- Data access and manipulation
- LOB access and manipulation
- Oracle object type mapping
- Object reference access and manipulation
- Array access and manipulation
- Application performance enhancement

Thin JDBC Features

The Thin JDBC driver provides security features such as strong authentication, data encryption, and data integrity checking.

Because the Thin JDBC driver is designed to be used with downloadable applets used over the Internet, Oracle designed a 100 percent Java implementation of Oracle Database network encryption and strong authentication, encryption, and integrity algorithms, for use with thin clients.

Oracle Database provides the following features for Thin JDBC:

- Strong Authentication
- Data encryption
- Data integrity checking
- Secure connections from Thin JDBC clients to the Oracle RDBMS
- Ability for developers to build applets that transmit data over a secure communication channel
- Secure connections from middle tier servers with Java Server Pages (JSP) to the Oracle RDBMS
- Secure connections from Oracle Database 12*c* release 1 (12.1) to older versions of Oracle databases

The Oracle JDBC Thin driver supports the Oracle Database SSL implementation and third party authentication methods such as RADIUS and Kerberos. Thin JDBC support for authentication methods like RADIUS, Kerberos, and SSL were introduced in Oracle Database 11*g* release 1 (11.1).

The Oracle Database network encryption and strong authentication Java implementation provides Java versions of the following encryption algorithms:

- `AES256`: AES 256-bit key
- `AES192`: AES 192-bit key
- `AES128`: AES 128-bit key

Note:

In the preceding list of algorithms, CBC refers to the Cipher Block Chaining mode.

Thin JDBC support for the Advanced Encryption Standard (AES) has been newly introduced in Oracle Database 12*c* Release 1 (12.1).

In addition, this implementation provides data integrity checking for Thin JDBC using Secure Hash Algorithm (`SHA1`) and Message Digest 5 (`MD5`). Thin JDBC support for `SHA1` was introduced in Oracle Database 11*g* release 1 (11.1).

See Also:

Oracle Database JDBC Developer's Guide for details on configuring authentication, encryption, and integrity for thin JDBC clients.

Implementation Overview

On the server side, the negotiation of algorithms and the generation of keys function exactly the same as Oracle Database native encryption.

This feature enables backward and forward compatibility of clients and servers.

On the client side, the algorithm negotiation and key generation occur in exactly the same manner as OCI clients. The client and server negotiate encryption algorithms, generate random numbers, use Diffie-Hellman to exchange session keys, and use the Oracle Password Protocol, in the same manner as the traditional Oracle Net clients. Thin JDBC contains a complete implementation of an Oracle Net client in pure Java.

Obfuscation of the Java Cryptography Code

The obfuscation of the Java cryptography code protects Java classes and methods that contain encryption and decryption capabilities with obfuscation software.

Java byte code obfuscator is a process frequently used to protect intellectual property written in the form of Java programs. It mixes up Java symbols found in the code. The process leaves the original program structure intact, letting the program run correctly while changing the names of the classes, methods, and variables in order to hide the intended behavior. Although it is possible to decompile and read non-obfuscated Java code, obfuscated Java code is sufficiently difficult to decompile to satisfy U.S. government export controls.

Configuration Parameters for the Thin JDBC Network Implementation

The Thin JDBC network implementation for the client provides parameters to control encryption, integrity, and the authentication service.

About the Thin JDBC Network Implementation Configuration Parameters

The JDBC network implementation configuration parameters control network settings such as the level of security used between client and server connections.

A properties class object containing several configuration parameters is passed to the Oracle Database network encryption and strong authentication interface.

All JDBC connection properties including the ones pertaining to Oracle Database are defined as constants in the `oracle.jdbc.OracleConnection` interface. The following list enumerates some of these connection properties:

See Also:

Oracle Database JDBC Developer's Guide for detailed information on configuration parameters and configuration examples

Client Encryption Level Parameter

The `CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL` parameter defines the level of security that the client uses to negotiate with the server.

Table 16-1 describes the attributes of this parameter.

Table 16-1 CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	`REJECTED`; `ACCEPTED`; `REQUESTED`; `REQUIRED`
Default Value	`ACCEPTED`
Syntax	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL,`*`level`*`);` where `prop` is an object of the `Properties` class
Example	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL,"REQUIRED");` where `prop` is an object of the `Properties` class

Client Encryption Selected List Parameter

The `CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES` parameter defines the encryption algorithm to be used.

Table 16-2 describes attributes of this parameter.

Table 16-2 CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	AES256 (AES 256-bit key), AES192 (AES 192-bit key), AES128 (AES 128-bit key),
Syntax	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES,`*`algorithm`*`);` where `prop` is an object of the `Properties` class

Table 16-2 (Cont.) CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES Attributes

Attribute	Description
Example	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES, "( AES256, AES192 )");` where `prop` is an object of the `Properties` class

Client Integrity Level Parameter

The `CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL` parameter defines the level of security to negotiate with the server for data integrity.

Table 16-3 describes the attributes of this parameter.

Table 16-3 CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	`REJECTED`; `ACCEPTED`; `REQUESTED`; `REQUIRED`
Default Value	`ACCEPTED`
Syntax	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL,`*`level`*`);` where `prop` is an object of the `Properties` class
Example	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL,"REQUIRED");` where `prop` is an object of the `Properties` class

Client Integrity Selected List Parameter

The `CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES` parameter defines the data integrity algorithm to be used.

Table 16-4 describes this parameter's attributes.

Table 16-4 CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	SHA1
Syntax	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES, `*`algorithm`*`);` where `prop` is an object of the `Properties` class

Table 16-4 (Cont.) CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES Attributes

Attribute	Description
Example	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES,"( MD5, SHA1 )");` where `prop` is an object of the `Properties` class

Client Authentication Service Parameter

The `CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES` parameter determines the authentication service to be used.

Table 16-5 describes this parameter's attributes.

Table 16-5 CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	`RADIUS, KERBEROS, SSL`
Syntax	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES,`*`authentication`*`);` where `prop` is an object of the `Properties` class
Example	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES,"( RADIUS, KERBEROS, SSL)");` where `prop` is an object of the `Properties` class

AnoServices Constants

The `oracle.net.ano.AnoServices` interface includes the names of the encryption, authentication, and checksum algorithms that the JDBC Thin driver supports.

The following constants are in the `oracle.net.ano.AnoServices` interface:

```
// ---- SUPPORTED ENCRYPTION ALG -----
public static final String ENCRYPTION_RC4_40 = "RC4_40";
public static final String ENCRYPTION_RC4_56 = "RC4_56";
public static final String ENCRYPTION_RC4_128 = "RC4_128";
public static final String ENCRYPTION_RC4_256 = "RC4_256";
public static final String ENCRYPTION_DES40C = "DES40C";
public static final String ENCRYPTION_DES56C = "DES56C";
public static final String ENCRYPTION_3DES112 = "3DES112";
public static final String ENCRYPTION_3DES168 = "3DES168";
public static final String ENCRYPTION_AES128 = "AES128";
public static final String ENCRYPTION_AES192 = "AES192";
public static final String ENCRYPTION_AES256 = "AES256";
// ---- SUPPORTED INTEGRITY ALG ----
public static final String CHECKSUM_MD5 = "MD5";
public static final String CHECKSUM_SHA1 = "SHA1";
// ---- SUPPORTED AUTHENTICATION ADAPTORS ----
public static final String AUTHENTICATION_RADIUS = "RADIUS";
public static final String AUTHENTICATION_KERBEROS = "KERBEROS";
```

You can use these constants to set the encryption, integrity, and authentication parameters. Example 16-1 illustrates one such scenario.

Example 16-1 Using AnoServices Constants in JDBC Client Code

```
import java.sql.*;
import java.util.Properties;import oracle.jdbc.*;
import oracle.net.ano.AnoServices;
/**
 * JDBC thin driver demo: new security features in 11gR1.
 *
 * This program attempts to connect to the database using the JDBC thin
 * driver and requires the connection to be encrypted with either AES256 or AES192
 * and the data integrity to be verified with SHA1.
 *
 * In order to activate encryption and checksumming in the database you need to
 * modify the sqlnet.ora file. For example:
 *
 *   SQLNET.ENCRYPTION_TYPES_SERVER = (AES256,AES192,AES128)
 *   SQLNET.ENCRYPTION_SERVER = accepted
 *   SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER= (SHA1)
 *   SQLNET.CRYPTO_CHECKSUM_SERVER = accepted
 *
 * This output of this program is:
 *   Connection created! Encryption algorithm is: AES256, data integrity algorithm
 *   is: SHA1
 *
 */
public class DemoAESAndSHA1
{
  static final String USERNAME= "hr";
  static final String PASSWORD= "hr";
  static final String URL =
"jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=somehost.us.example.com)
(PORT=5561))"
+"(CONNECT_DATA=(SERVICE_NAME=itydemo.regress.rdbms.dev.us.example.com)))";

  public static final void main(String[] argv)
  {
    DemoAESAndSHA1 demo = new DemoAESAndSHA1();
    try
    {
      demo.run();
    }catch(SQLException ex)
    {
      ex.printStackTrace();
    }
  }
  void run() throws SQLException
  {
    OracleDriver dr = new OracleDriver();
    Properties prop = new Properties();
    // We require the connection to be encrypted with either AES256 or AES192.
    // If the database doesn't accept such a security level, then the connection
    // attempt will fail.
    prop.setProperty(
OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL,AnoServices.ANO_REQUIR
ED);
    prop.setProperty(
      OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES,      "( " +
AnoServices.ENCRYPTION_AES256 + "," +AnoServices.ENCRYPTION_AES192 + ")");
    // We also require the use of the SHA1 algorithm for data integrity checking.
    prop.setProperty(

OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL,AnoServices.ANO_REQUIRED
);
    prop.setProperty(
      OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES,      "( " +
AnoServices.CHECKSUM_SHA1 + " )");

    prop.setProperty("user",DemoAESAndSHA1.USERNAME);
    prop.setProperty("password",DemoAESAndSHA1.PASSWORD);
    OracleConnection oraConn =
 (OracleConnection)dr.connect(DemoAESAndSHA1.URL,prop);

    System.out.println("Connection created! Encryption algorithm is:
"+oraConn.getEncryptionAlgorithmName()    +", data integrity algorithm is:
"+oraConn.getDataIntegrityAlgorithmName());

    oraConn.close();
  }

}
```

Part V
Managing Strong Authentication

Part V describes how to manage strong authentication.

17 Introduction to Strong Authentication

Strong authentication supports tools such as Secure Sockets Layer (SSL) to verify the identities of users who log in to the database.

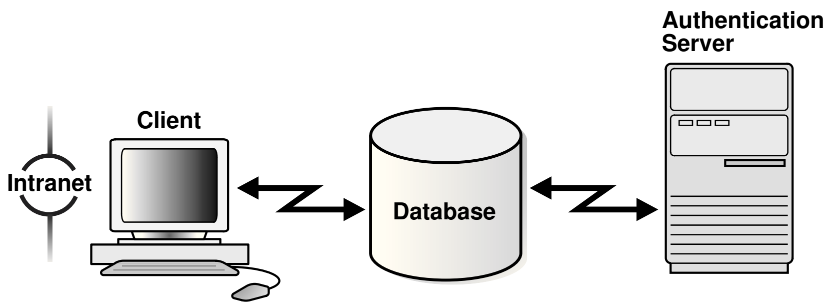
What Is Strong Authentication?

You use authentication to prove the identities of users who are attempting to log into the database.

Authenticating user identity is imperative in distributed environments, without which there can be little confidence in network security. Passwords are the most common means of authentication. Oracle Database enables strong authentication with Oracle authentication adapters that support various third-party authentication services, including SSL with digital certificates.

Figure 17-1 shows user authentication with an Oracle database instance configured to use a third-party authentication server. Having a central facility to authenticate all members of the network (clients to servers, servers to servers, users to both clients and servers) is one effective way to address the threat of network nodes falsifying their identities.

Figure 17-1 Strong Authentication with Oracle Authentication Adapters



Centralized Authentication and Single Sign-On

Single sign-on enables users to access multiple accounts and applications with a single password.

Centralized authentication also provides the benefit of single sign-on (SSO) for users.

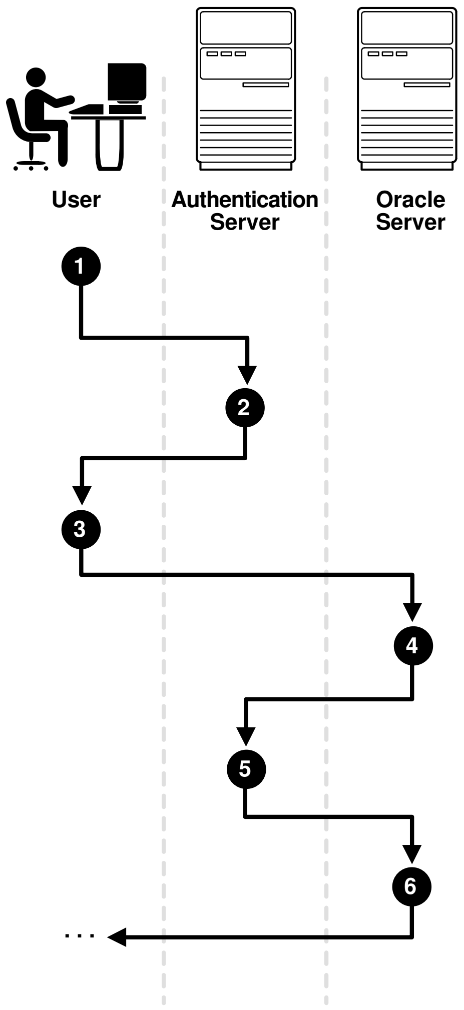
In single sign-on, a user only needs to login once and can then automatically connect to any other service without having to giving user name and password again. Single sign-on eliminates the need for the user to remember and administer multiple passwords, reducing the time spent logging into multiple services.

How Centralized Network Authentication Works

A centralized network authentication system works with an Oracle server, an authentication server, and users who connect to the Oracle server.

Figure 17-2 shows how a centralized network authentication service typically operates.

Figure 17-2 How a Network Authentication Service Authenticates a User



The following steps describe how centralized Network Authentication Process works.

1. A user (client) requests authentication services and provides identifying information, such as a token or password.
2. The authentication server validates the user's identity and passes a ticket or credentials back to the client, which may include an expiration time.
3. The client passes these credentials to the Oracle server concurrent with a service request, such as connection to a database.
4. The server sends the credentials back to the authentication server for authentication.

5. The authentication server checks the credentials and notifies the Oracle server.
6. If the credentials were accepted by the authentication server, then the Oracle server authenticates the user. If the authentication server rejected the credentials, then authentication fails, and the service request is denied.

Supported Strong Authentication Methods

Oracle Database supports industry-standard authentication methods.

About Kerberos

Oracle Database support for Kerberos provides the benefits of single sign-on and centralized authentication of Oracle users.

Kerberos is a trusted third-party authentication system that relies on shared secrets. It presumes that the third party is secure, and provides single sign-on capabilities, centralized password storage, database link authentication, and enhanced PC security. It does this through a Kerberos authentication server. Refer to Configuring Kerberos Authentication for information about configuring and using this adapter.

Note:

Oracle authentication for Kerberos provides database link authentication (also called proxy authentication). Kerberos is also an authentication method that is supported with Enterprise User Security.

About Remote Authentication Dial-In User Service (RADIUS)

RADIUS is a client/server security protocol that is most widely known for enabling remote authentication and access.

Oracle Database uses this standard in a client/server network environment to enable use of any authentication method that supports the RADIUS protocol. RADIUS can be used with a variety of authentication mechanisms, including token cards and smart cards.

- **Smart Cards.** A RADIUS-compliant smart card is a credit card-like hardware device which has memory and a processor. It is read by a smart card reader located at the client workstation.
- **Token Cards.** Token cards (Secure ID or RADIUS-compliant) can improve ease of use through several different mechanisms. Some token cards dynamically display one-time passwords that are synchronized with an authentication service. The server can verify the password provided by the token card at any given time by contacting the authentication service. Other token cards have a keypad and operate on a challenge-response basis. In this case, the server offers a challenge (a number) that the user enters into a token card. The token card provides a response (another number cryptographically derived from the challenge) that the user enters and sends to the server.

  You can use SecurID tokens through the RADIUS adapter.

About Secure Sockets Layer

Secure Sockets Layer (SSL) is an industry standard protocol for securing network connections.

SSL provides authentication, data encryption, and data integrity.

The SSL protocol is the foundation of a public key infrastructure (PKI). For authentication, SSL uses digital certificates that comply with the X.509v3 standard and a public and private key pair.

You can use the Oracle Database SSL can be used to secure communications between any client and any server. You can configure SSL to provide authentication for the server only, the client only, or both client and server. You can also configure SSL features in combination with other authentication methods supported by Oracle Database (database user names and passwords, RADIUS, and Kerberos).

To support your PKI implementation, Oracle Database includes the following features in addition to SSL:

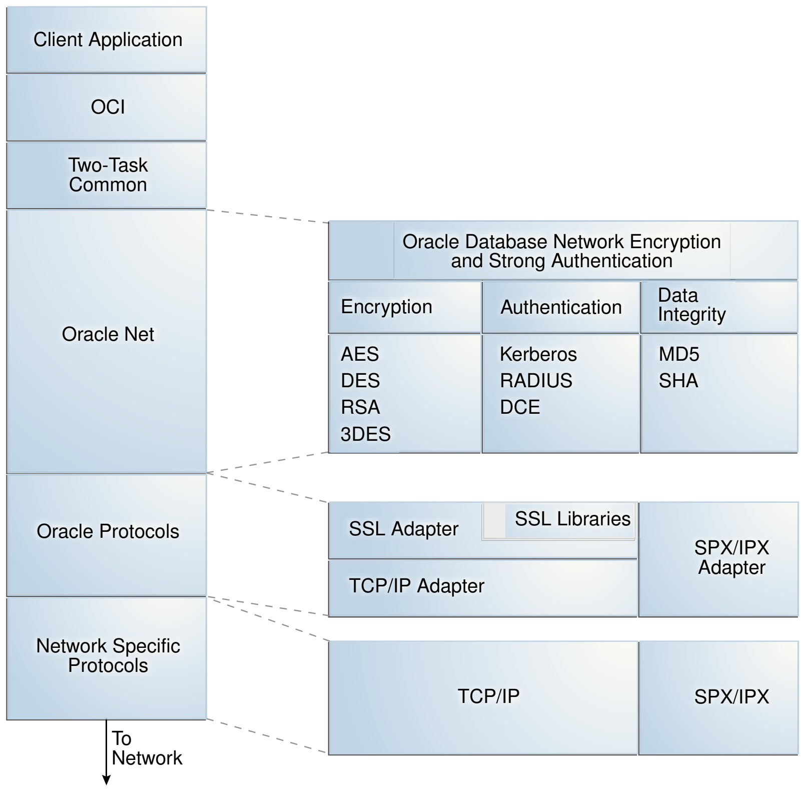
- Oracle wallets, where you can store PKI credentials
- Oracle Wallet Manager, which you can use to manage your Oracle wallets
- Certificate validation with certificate revocation lists (CRLs)
- Hardware security module support

Oracle Database Network Encryption/Strong Authentication Architecture

The Oracle Database network encryption and strong authentication architecture complements an Oracle database server or client installations.

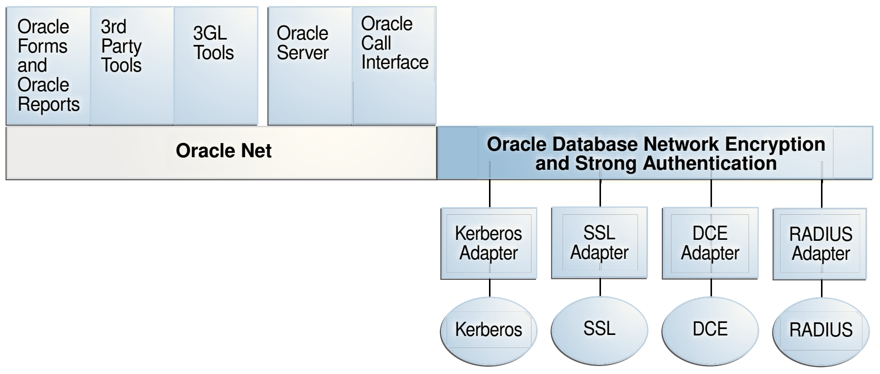
Figure 17-3 shows the this architecture within an Oracle networking environment.

Figure 17-3 Oracle Network Encryption and Strong Authentication Architecture



Oracle Database supports authentication through adapters that are similar to the existing Oracle protocol adapters. As shown in Figure 17-4, authentication adapters integrate the Oracle Net interface, and allow existing applications to take advantage of new authentication systems transparently, without any changes to the application.

Figure 17-4 Oracle Net Services with Authentication Adapters



See Also:

Oracle Database Net Services Administrator's Guide for more information about stack communications in an Oracle networking environment

System Requirements for Strong Authentication

Kerberos, RADIUS, and Secure Sockets Layer (SSL) have a set of system requirements for strong authentication.

Table 17-1 lists the SSL system requirements for strong authentication.

Table 17-1 Authentication Methods and System Requirements

Authentication Method	System Requirements
Kerberos	• MIT Kerberos Version 5, release 1.8 or above. • The Kerberos authentication server must be installed on a physically secure system.
RADIUS	• A RADIUS server that is compliant with the standards in the Internet Engineering Task Force (IETF) RFC #2138, *Remote Authentication Dial In User Service (RADIUS)* and RFC #2139 *RADIUS Accounting*. • To enable challenge-response authentication, you must run RADIUS on an operating system that supports the Java Native Interface as specified in release 1.1 of the Java Development Kit from JavaSoft.
SSL	• A wallet that is compatible with the Oracle Wallet Manager 10*g release*. Wallets created in earlier releases of the Oracle Wallet Manager are not forward compatible.

Oracle Network Encryption and Strong Authentication Restrictions

Oracle applications support Oracle network encryption and strong authentication.

However, because Oracle network encryption and strong authentication requires Oracle Net Services to transmit data securely, these external authentication features are not supported by some parts of Oracle Financial, Human Resource, and Manufacturing Applications when they are running on Microsoft Windows.

The portions of these products that use Oracle Display Manager (ODM) do not take advantage of Oracle network encryption and strong authentication, because ODM does not use Oracle Net Services.

18 Strong Authentication Administration Tools

You can use a set of strong authentication administration tools for network encryption and public key infrastructure credentials.

About the Configuration and Administration Tools

The configuration and administration tools manage the encryption, integrity (checksumming), and strong authentication methods for Oracle Net Services.

Strong authentication method configuration can include third-party software, as is the case for Kerberos or RADIUS, or it may entail configuring and managing a public key infrastructure for using digital certificates with Secure Sockets Layer (SSL).

Network Encryption and Strong Authentication Configuration Tools

Oracle Net Services can encrypt data using standard encryption algorithms, and for strong authentication methods, such as Kerberos, RADIUS, and SSL.

About Oracle Net Manager

Oracle Net Manager configures Oracle Net Services for an Oracle home on a local client or server host.

Although you can use Oracle Net Manager, a graphical user interface tool, to configure Oracle Net Services, such as naming, listeners, and general network settings, it also enables you to configure the following features, which use the Oracle Net protocol:

- Strong authentication (Kerberos, RADIUS, and Secure Sockets Layer)
- Network encryption (RC4, DES, Triple-DES, and AES)
- Checksumming for data integrity (MD5, SHA-1, SHA-2)

Kerberos Adapter Command-Line Utilities

The Kerberos adapter provides command-line utilities that obtain, cache, display, and remove Kerberos credentials.

The following table briefly describes these utilities.

Table 18-1 Kerberos Adapter Command-Line Utilities

Utility Name	Description
`okinit`	Obtains Kerberos tickets from the Key Distribution Center (KDC) and caches them in the user's credential cache

Table 18-1 (Cont.) Kerberos Adapter Command-Line Utilities

Utility Name	Description
`oklist`	Displays a list of Kerberos tickets in the specified credential cache
`okdstry`	Removes Kerberos credentials from the specified credential cache
`okcreate`	Automates the creation of keytabs from either the KDC or a service endpoint

Note:

The Cybersafe adapter is not supported beginning with this release. You should use Oracle's Kerberos adapter in its place. Kerberos authentication with the Cybersafe KDC (Trust Broker) continues to be supported when using the Kerberos adapter.

Public Key Infrastructure Credentials Management Tools

The security provided by a public key infrastructure (PKI) depends on how effectively you store, manage, and validate your PKI credentials.

About Oracle Wallet Manager

Wallet owners and security administrators use Oracle Wallet Manager to manage and edit the security credentials in their Oracle wallets.

A wallet is a password-protected container that is used to store authentication and signing credentials, including private keys, certificates, and trusted certificates needed by SSL. You can use Oracle Wallet Manager to perform the following tasks:

- Create public and private key pairs
- Store and manage user credentials
- Generate certificate requests
- Store and manage certificate authority certificates (root key certificate and certificate chain)
- Upload and download wallets to and from an LDAP directory
- Create wallets to store hardware security module credentials

Note:

In previous releases of Oracle Database, you could use Oracle Wallet Manager to configure wallets for Transparent Data Encryption. In this release, you can use the `ADMINISTER KEY MANAGEMENT SQL` statement instead. For more information, see *Oracle Database Advanced Security Guide*.

About the orapki Utility

The `orapki` utility manages certificate revocation lists (CRLs), creates and manages Oracle wallets, and creates signed certificates.

The basic syntax for this command-line utility is as follows:

```
orapki module command -option_1 argument ... -option_n argument
```

For example, the following command lists all CRLs in the CRL subtree in an instance of Oracle Internet Directory that is installed on `machine1.us.example.com` and that uses port 389:

```
orapki crl list -ldap machine1.us.example.com:389
```

Note:

The use of `orapki` to configure Transparent Data Encryption has been deprecated. Instead, use the `ADMINISTER KEY MANAGEMENT` SQL statement.

See Also:

- Certificate Revocation List Management for information about how to use `orapki` to manage CRLs in the directory
- Managing Public Key Infrastructure (PKI) Elements for reference information on all available `orapki` commands
- *Oracle Database Advanced Security Guide* for information about using the `ADMINISTER KEY MANAGEMENT` SQL statement

Duties of Strong Authentication Administrators

Most of the tasks of a security administrator involve ensuring that the connections to and from Oracle databases are secure.

The following table describes the primary tasks of security administrators who are responsible for strong authentication, the tools used to perform the tasks, and links to where the tasks are documented.

Table 18-2 Common Security Administrator/DBA Configuration and Administrative Tasks

Task	Tools Used	See Also
Configure encrypted Oracle Net connections between database servers and clients	Oracle Net Manager	Configuring Encryption on the Client and the Server
Configure checksumming on Oracle Net connections between database servers and clients	Oracle Net Manager	Configuring Integrity on the Client and the Server
Configure database clients to accept RADIUS authentication	Oracle Net Manager	Step 1A: Configure RADIUS on the Oracle Client
Configure a database to accept RADIUS authentication	Oracle Net Manager	Step 1B: Configure RADIUS on the Oracle Database Server
Create a RADIUS user and grant them access to a database session	SQL*Plus	Step 2: Create a User and Grant Access
Configure Kerberos authentication on a database client and server	Oracle Net Manager	Step 6: Configure Kerberos Authentication
Create a Kerberos database user	• `kadmin.local` • Oracle Net Manager	• Step 7: Create a Kerberos User • Step 8: Create an Externally Authenticated Oracle User
Manage Kerberos credentials in the credential cache	• `okinit` • `oklist` • `okdstry` • `okcreate`	• okinit Utility Options for Obtaining the Initial Ticket • oklist Utility Options for Displaying Credentials • okdstry Utility Options for Removing Credentials from the Cache File
Create a wallet for a database client or server	Oracle Wallet Manager	*Oracle Database Enterprise User Security Administrator's Guide*
Request a user certificate from a certificate authority (CA) for SSL authentication	Oracle Wallet Manager	• *Oracle Database Enterprise User Security Administrator's Guide* to add a certificate request • *Oracle Database Enterprise User Security Administrator's Guide* to import a user certificate into an Oracle wallet
Import a user certificate and its associated trusted certificate (CA certificate) into a wallet	Oracle Wallet Manager	• *Oracle Database Enterprise User Security Administrator's Guide* to import a trusted certificate • *Oracle Database Enterprise User Security Administrator's Guide* to import a user certificate into an Oracle wallet
Configuring SSL connections for a database client	Oracle Net Manager	Step 2: Configure Secure Sockets Layer on the Client
Configuring SSL connections for a database server	Oracle Net Manager	Step 1: Configure Secure Sockets Layer on the Server
Enabling certificate validation with a certificate revocation list (CRL)	Oracle Net Manager	Configuring Certificate Validation with Certificate Revocation Lists

19 Configuring Kerberos Authentication

Kerberos is a trusted third-party authentication system that relies on shared secrets and presumes that the third party is secure.

Enabling Kerberos Authentication

To enable Kerberos authentication for Oracle Database, you must first install it, and then follow a set of configuration steps.

> **See Also:**
>
> *Oracle Database Enterprise User Security Administrator's Guide* for information on migrating Kerberos users to Kerberos-authenticated enterprise users

Step 1: Install Kerberos

You should install Kerberos Version 5.

The source distribution for notes about building and installing Kerberos provide details. After you install Kerberos, if you are using IBM AIX on POWER systems (64-bit), you should ensure that Kerboros 5 is the preferred authentication method.

1. Install Kerberos on the system that functions as the authentication server.

   > **Note:**
   >
   > After upgrading from a 32-bit version of Oracle Database, the first use of the Kerberos authentication adapter causes an error message: `ORA-01637: Packet receive failed.`
   >
   > **Workaround:** After upgrading to the 64-bit version of the database and before using Kerberos external authentication method, check for a file named `/usr/tmp/oracle_service_name.RC` on your computer, and remove it.

2. For IBM AIX on POWER systems (64-bit), check the authentication method.

   For example:

   ```
   /usr/bin/lsauthent
   ```

   Output similar to the following may appear:

   ```
   Standard Aix
   ```

3. Configure Kerberos 5 as the preferred method.

For example:

```
/usr/bin/chauthent -k5 -std
```

This command sets Kerberos 5 as the preferred authentication method (`k5`) and Standard AIX as the second (`std`).

4. To ensure that Kerberos 5 is now the preferred method, check the new configuration.

```
/usr/bin/lsauthent

Kerberos 5
Standard Aix
```

Step 2: Configure a Service Principal for an Oracle Database Server

You must create a service principal for Oracle Database before the server can validate the identity of clients that authenticate themselves using Kerberos.

1. Decide on a name for the service principal, using the following format:

```
kservice/kinstance@REALM
```

Each of the fields in the service principal specify the following values:

Service Principal Field	Description
`kservice`	A case-sensitive string that represents the Oracle service. This can be the same as the database service name.
`kinstance`	Typically the fully qualified DNS name of the system on which Oracle Database is running.
`REALM`	The name of the Kerberos realm with which the service principal is registered. `REALM` must always be uppercase and is typically the DNS domain name.

The utility names in this section are executable programs. However, the Kerberos user name `krbuser` and the realm `EXAMPLE.COM` are examples only.

For example, suppose `kservice` is `oracle`, the fully qualified name of the system on which Oracle Database is running is `dbserver.example.com` and the realm is `EXAMPLE.COM`. The principal name then is:

```
oracle/dbserver.example.com@EXAMPLE.COM
```

2. Run `kadmin.local` to create the service principal. On UNIX, run this command as the root user, by using the following syntax:

```
# cd /kerberos-install-directory/sbin
# ./kadmin.local
```

For example, to add a principal named `oracle/dbserver.example.com@EXAMPLE.COM` to the list of server principals known by Kerberos, you can enter the following:

```
kadmin.local:addprinc -randkey oracle/dbserver.example.com@EXAMPLE.COM
```

Step 3: Extract a Service Key Table from Kerberos

Next, you are ready to extract the service key table from Kerberos and copy it to the Oracle database server/Kerberos client system.

For example, to extract a service key table for `dbserver.example.com`:

1. Enter the following to extract the service key table:

   ```
   kadmin.local:  ktadd -k /tmp/keytab oracle/dbserver.example.com
   Entry for principal oracle/dbserver.example.com with kvno 2,
   encryption type AES-256 CTS mode with 96-bit SHA-1 HMAC added to keytab WRFILE:
   WRFILE:/tmp/keytab

   kadmin.local:  exit
   ```

2. To check the service key table, enter the following command:

   ```
   oklist -k -t /tmp/keytab
   ```

3. After the service key table has been extracted, verify that the new entries are in the table in addition to the old ones.

   If they are not, or you need to add more, use `kadmin.local` to append to them.

   If you do not enter a realm when using `ktadd`, it uses the default realm of the Kerberos server. `kadmin.local` is connected to the Kerberos server running on the `localhost`.

4. If the Kerberos service key table is on the same system as the Kerberos client, you can move it. If the service key table is on a different system from the Kerberos client, you must transfer the file with a program such as FTP. If using FTP, transfer the file in binary mode.

   The following example shows how to move the service key table on a UNIX platform:

   ```
   # mv /tmp/keytab /etc/v5srvtab
   ```

   The default name of the service file is `/etc/v5srvtab`.

5. Verify that the owner of the Oracle database server executable can read the service key table (`/etc/v5srvtab` in the previous example).

   To do so, set the file owner to the Oracle user, or make the file readable by the group to which Oracle belongs.

   Do not make the file readable to all users. This can cause a security breach.

Step 4: Install an Oracle Database Server and an Oracle Client

After you extract a service key table from Kerberos, you are ready to install the Oracle Database server and an Oracle client.

- See the Oracle Database operating system-specific installation documentation for instructions on installing the Oracle database server and client software.

Step 5: Configure Oracle Net Services and Oracle Database

After you install the Oracle Database server and client, you can configure Oracle Net Services on the server and client.

- See the following documentation for information on configuring Oracle Net Services on the Oracle database server and client.
  - Oracle Database operating system-specific installation documentation
  - *Oracle Database Net Services Administrator's Guide*

Step 6: Configure Kerberos Authentication

You must set the required parameters in the Oracle database server and client `sqlnet.ora` files.

> **Note:**
>
> Be aware that in a multitenant environment, the settings in the `sqlnet.ora` file apply to all pluggable databases (PDBs). However, this does not mean that all PDBs must authenticate with one KDC if using Kerberos; the settings in the `sqlnet.ora` file and Kerberos configuration files can support multiple KDCs.

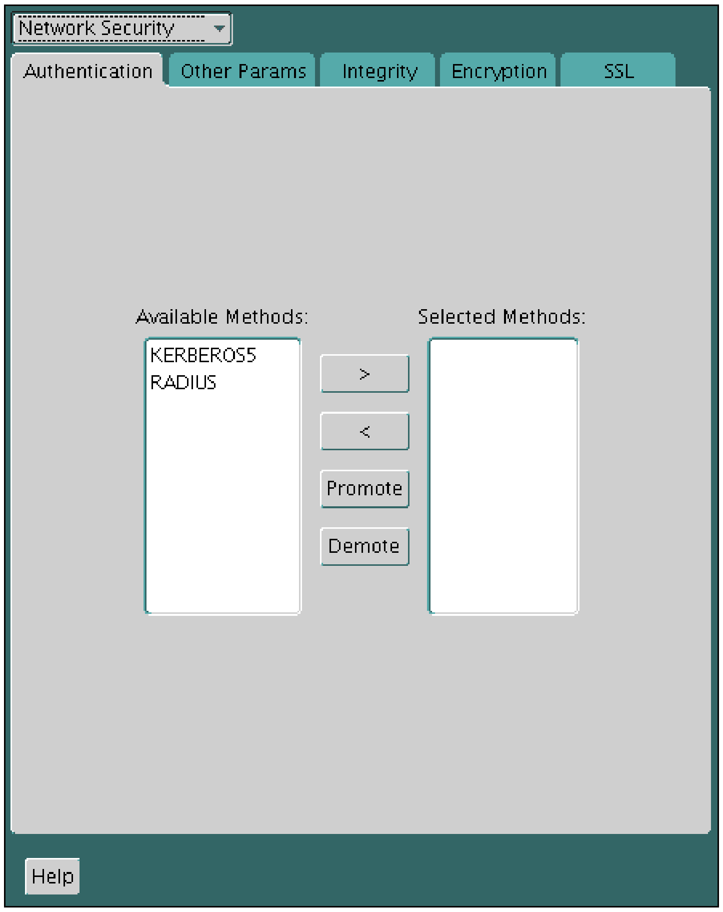
Step 6A: Configure Kerberos on the Client and on the Database Server

First, you must configure Kerberos authentication service parameters on the client and on the database server.

1. Start Oracle Net Manager.
   - (UNIX) From *$ORACLE_HOME*/bin, enter the following command at the command line:

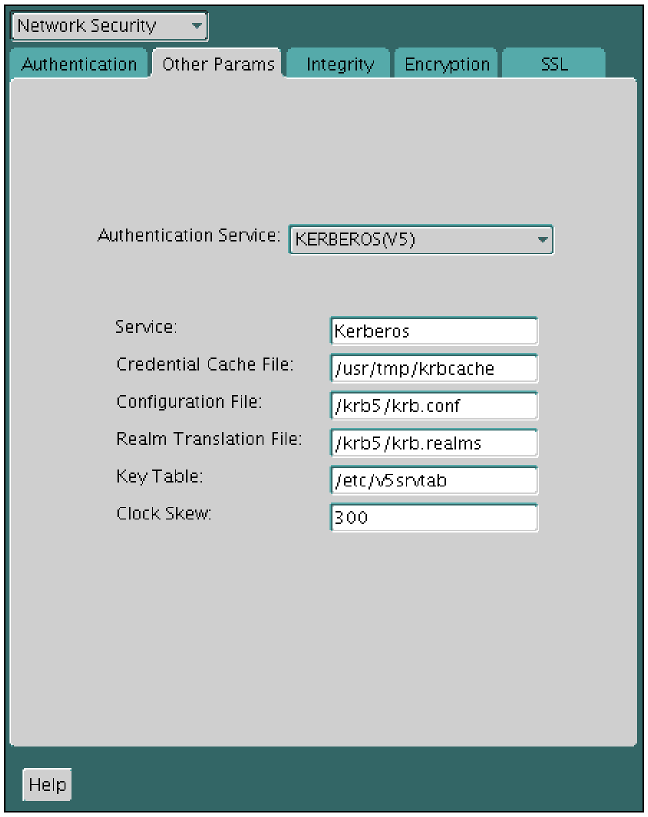
     ```
     netmgr
     ```
   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **Authentication** tab.



5. From the Available Methods list, select **KERBEROS5**.
6. Move **KERBEROS5** to the Selected Methods list by clicking the right arrow (**>**).
7. Arrange the selected methods in order of use.

   To do so, select a method in the Selected Methods list, then click **Promote** or **Demote** to position it in the list. For example, if you want `KERBEROS5` to be the first service used, move it to the top of the list.
8. Select the **Other Params** tab.
9. From the Authentication Service list, select **KERBEROS(V5)**.
10. Type **Kerberos** into the **Service** field.



Authentication Service: KERBEROS(V5)

Service: Kerberos
Credential Cache File: /usr/tmp/krbcache
Configuration File: /krb5/krb.conf
Realm Translation File: /krb5/krb.realms
Key Table: /etc/v5srvtab
Clock Skew: 300

This field defines the name of the service Oracle Database uses to obtain a Kerberos service ticket. When you provide the value for this field, the other fields are enabled.

11. Optionally enter values for the following fields:
    - **Credential Cache File**
    - **Configuration File**
    - **Realm Translation File**
    - **Key Table**
    - **Clock Skew**

    See the Oracle Net Manager online Help, and Step 6C: Set sqlnet.ora Parameters (Optional), for more information about the fields and the parameters they configure.
12. From the **File** menu, select **Save Network Configuration**.

    The `sqlnet.ora` file is updated with the following entries in addition to any optional choices that you may have made in the previous step:

```
SQLNET.AUTHENTICATION_SERVICES=(KERBEROS5)
SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=kservice
```

Step 6B: Set the Initialization Parameters

Next, you are ready to set the `OS_AUTHENT_PREFIX` initialization parameter.

1. Locate the `init.ora` file.

   By default, the `init.ora` file is located in the *`ORACLE_HOME`*`/dbs` directory (or the same location of the data files) on Linux and UNIX systems, and in the *`ORACLE_HOME`*`\database` directory on Windows.

2. In the `init.ora` file, set the value of `OS_AUTHENT_PREFIX` to null in the `init.ora` initialization parameter file.

   For example:

   ```
   OS_AUTHENT_PREFIX=""
   ```

   Set this value to null because Kerberos user names can be long, and Oracle user names are limited to 30 characters. Setting this parameter to null overrides the default value of `OPS$`.

> **Note:**
>
> You can create external database users that have Kerberos user names of more than 30 characters. See Step 8: Create an Externally Authenticated Oracle User for more information.

Step 6C: Set sqlnet.ora Parameters (Optional)

You can set optional `sqlnet.ora` parameters, in addition to the required parameters, for better security.

- Optionally, set the parameters listed in the following table on both the client and the Oracle database server.

Table 19-1 Kerberos-Specific sqlnet.ora Parameters

Parameter	Description

`SQLNET.KERBEROS5_CC_NAME=`*`pathname_to_credentials_cache_file`*`|`*`OS_MEMORY`*	Specifies the complete path name to the Kerberos credentials cache (CC) file. The default value is operating system-dependent. For UNIX, it is `/tmp/krb5cc_`*`userid`*.
	Using the `OS_MEMORY` option indicates that an OS-managed memory credential cache is used for the credential cache file. The only value currently supported for this is `OSMSFT` (for Microsoft Windows).
	You can use the following formats to specify a value for `SQLNET.KERBEROS5_CC_NAME`:
	• `SQLNET.KERBEROS5_CC_NAME=`*`complete_path_to_cc_file`*
	For example:
	`SQLNET.KERBEROS5_CC_NAME=/tmp/kcache`
	`SQLNET.KERBEROS5_CC_NAME=D:\tmp\kcache`
	• `SQLNET.KERBEROS5_CC_NAME=FILE:`*`complete_path_to_cc_file`*
	For example:
	`SQLNET.KERBEROS5_CC_NAME=FILE:/tmp/kcache`
	• `SQLNET.KERBEROS5_CC_NAME=OSMSFT:`
	Use this value if you are running Windows and using a Microsoft KDC.
	You can also set this parameter by using the `KRB5CCNAME` environment variable, but the value set in the `sqlnet.ora` file takes precedence over the value set in `KRB5CCNAME`.
	For example:
	`SQLNET.KERBEROS5_CC_NAME=/usr/tmp/krbcache`

`SQLNET.KERBEROS5_CLOCKSKEW=`*`number_of_seconds_accepted_as_network_delay`*	This parameter specifies how many seconds can pass before a Kerberos credential is considered out-of-date. It is used when a credential is actually received by either a client or a database server. An Oracle database server also uses it to decide if a credential needs to be stored to protect against a replay attack. The default is 300 seconds. For example: `SQLNET.KERBEROS5_CLOCKSKEW=1200`

`SQLNET.KERBEROS5_CONF=`*`pathname_to_Kerberos_configuration_file`*`|`
`AUTO_DISCOVER`
	This parameter specifies the complete path name to the `Kerberos` configuration file. The configuration file contains the realm for the default KDC (key distribution center) and maps realms to KDC hosts. The default is operating system-dependent. For UNIX, it is `/krb5/krb.conf`.
	Using the `AUTO_DISCOVER` option in place of the configuration file enables Kerberos clients to auto-discover the KDC.
	For example:
	`SQLNET.KERBEROS5_CONF=/krb/krb.conf`
	`SQLNET.KERBEROS5_CONF=AUTO_DISCOVER`

Table 19-1 (Cont.) Kerberos-Specific sqlnet.ora Parameters

Parameter	Description
`SQLNET.KERBEROS5_CONF_LOCATION=`*`path_to_Kerberos_configuration_directory`*	This parameter indicates that the Kerberos configuration file is created by the system, and does not need to be specified by the client. The configuration file uses DNS lookup to obtain the realm for the default KDC, and maps realms to KDC hosts. For example: `SQLNET.KERBEROS5_CONF_LOCATION=/krb`
`SQLNET.KERBEROS5_KEYTAB=`*`path_to_Kerberos_principal/key_table`*	This parameter specifies the complete path name to the Kerberos principal/secret key mapping file. It is used by the Oracle database server to extract its key and decrypt the incoming authentication information from the client. The default is operating system-dependent. For UNIX, it is `/etc/v5srvtab`. For example: `SQLNET.KERBEROS5_KEYTAB=/etc/v5srvtab`
`SQLNET.KERBEROS5_REALMS=`*`path_to_Kerberos_realm_translation_file`*	This parameter specifies the complete path name to the Kerberos realm translation file. The translation file provides a mapping from a host name or domain name to a realm. The default is operating system-dependent. For UNIX, it is `/etc/krb.realms`. For example: `SQLNET.KERBEROS5_REALMS=/krb5/krb.realms`

Step 7: Create a Kerberos User

You must create the Kerberos user on the Kerberos authentication server where the administration tools are installed.

The realm must already exist.

Note:

The utility names in this section are executable programs. However, the Kerberos user name `krbuser` and realm `EXAMPLE.COM` are examples only. They can vary among systems.

- Run `/krb5/admin/kadmin.local` as root to create a new Kerberos user, such as `krbuser`.

  For example, to create a Kerberos user is UNIX-specific:

  ```
  # /krb5/admin/kadmin.local
  kadmin.local: addprinc krbuser
  Enter password for principal: "krbuser@example.com": (password does not display)
  Re-enter password for principal: "krbuser@example.com": (password does not
  display)
  kadmin.local: exit
  ```

Step 8: Create an Externally Authenticated Oracle User

Next, you are ready to create an externally authenticated Oracle user.

1. Log in to SQL*Plus as a user who has the `CREATE USER` privilege.

   ```
   sqlplus sec_admin - Or, CONNECT sec_admin@hrpdb
   Enter password: password
   ```

2. Ensure that the `OS_AUTHENT_PREFIX` is set to null (`""`).
3. Create an Oracle Database user account that corresponds to the Kerberos user. Enter the Oracle user name in uppercase and enclose it in double quotation marks.

   For example:

   ```
   CREATE USER "KRBUSER@EXAMPLE.COM" IDENTIFIED EXTERNALLY;
   GRANT CREATE SESSION TO "KRBUSER@EXAMPLE.COM";
   ```

   If the user's Kerberos principal name is longer than 30 characters, and up to 1024 characters, then create the user by using the following syntax:

   ```
   CREATE USER db_user_name IDENTIFIED EXTERNALLY AS 'kerberos_principal_name';
   ```

   For example:

   ```
   CREATE USER KRBUSER IDENTIFIED EXTERNALLY AS 'KerberosUser@example.com';
   ```

Note:

The database administrator should ensure that two database users are not identified externally by the same Kerberos principal name.

Step 9: Get an Initial Ticket for the Kerberos/Oracle User

Before you can connect to the database, you must ask the Key Distribution Center (KDC) for an initial ticket.

- To request an initial ticket, run the following command on the client:

  ```
  % okinit username
  ```

  If you want to enable credentials that can be used across database links, then include the `-f` option and provide the Kerberos password when prompted.

  ```
  % okinit -f
  Password for krbuser@EXAMPLE.COM:(password does not display)
  ```

Utilities for the Kerberos Authentication Adapter

The Oracle Kerberos authentication adapter utilities are designed for an Oracle client with Oracle Kerberos authentication support installed.

okinit Utility Options for Obtaining the Initial Ticket

The `okinit` utility obtains and caches Kerberos tickets.

This utility is typically used to obtain the ticket-granting ticket, using a password entered by the user to decrypt the credential from the key distribution center (KDC). The ticket-granting ticket is then stored in the user's credential cache.

The following table lists the options available with `okinit`. To use the functionality that is described in this table, you must set the `sqlnet.ora` `SQLNET.KERBEROS5_CONF_MIT` parameter to `TRUE`. (Note that `SQLNET.KERBEROS5_CONF_MIT` is deprecated, but is retained for backward compatibility for `okinit`.)

Table 19-2 Options for the okinit Utility

Option	Description
`-f` | `-F`	Requests forwardable or non-forwardable tickets. This option is necessary to follow database links.
`-l` *`lifetime`*	Specifies the lifetime of the ticket-granting ticket and all subsequent tickets. By default, the ticket-granting ticket is good for eight (8) hours, but shorter or longer-lived credentials may be desired. The KDC can ignore this option or put site-configured limits on what can be specified. The lifetime value is a string that consists of a number qualified by `w` (weeks), `d` (days), `h` (hours), `m` (minutes), or `s` (seconds), as in the following example: `okinit -l 2wld6h20m30s` The example requests a ticket-granting ticket that has a lifetime of 2 weeks, 1 day, 6 hours, 20 minutes, and 30 seconds.
`-s` *`start_time`*	Specifies the duration of the delay before the ticket can become valid. Tickets are issued with the invalid flag set.
`-r` *`renewable_life`*	Requests renewable tickets with a total lifetime of *`renewable_life`*
`-p` | `-P`	Requests proxiable or non-proxiable tickets
`-a`	Requests tickets that are restricted to the local address of the host
`-A`	Requests tickets not restricted by address
`-E`	Treats the principal name as an enterprise name
`-v`	Requests that the ticket-granting ticket in the cache be passed to the KDC for validation. If the ticket is within the requested time range, then the cache is replaced with the validated ticket.
`-R`	Requests renewal of the ticket-granting ticket
`-k` [`-t` *`keytab_file`*]	Requests a ticket, which is obtained from a key in the local host’s keytab
`-n`	Requests anonymous processing
`-C`	Requests canonicalization of the principal name, and enables the KDC to reply with a different client principal from the one that was requested
`-c` *`cache_name`*	Specifies the name of a cache as a cache location. For UNIX, the default is `/tmp/krb5cc_`*uid*. You can also specify the alternate credential cache by using the `SQLNET.KERBEROS5_CC_NAME` parameter in the `sqlnet.ora` file.

Table 19-2 (Cont.) Options for the okinit Utility

Option	Description
`-I` *`input_cache`*	Specifies the name of a credential cache that already contains a ticket. When it obtains that ticket, if the information about how the ticket was obtained is stored in cache, then the same information will be used to affect how new credentials are obtained.
`-T` *`armor_cache`*	If supported by the KDC, this cache is used to armor the request, preventing offline dictionary attacks and enabling the use of additional pre-authentication mechanisms.
`-X` *`attribute`*`[=`*`value`*	Specifies a pre-authentication attribute and value. Specifies one of the following values: • `X509_user_identity=`*`value`* specifies where to find the user's X509 identity information • `X509_anchors=`*`value`* specifies where to find trusted X509 anchor information • `flag_RSA_PROTOCOL[=yes]` specifies the use of RSA rather than the default Diffie-Hellman protocol
`-e`	Specifies a number representing the Kerberos encryption type to use. This option can be used to request a particular Kerberos encryption type key for the session. If you specify more than one encryption type, then the KDC chooses the common and strongest encryption type from the list. The following values are allowed: • 1 for `DES-CBC-CRC` • 3 for `DES-CBC-MD5` • 16 for `DES3-CBC-SHA1` • 18 for `AES256-CTS` • 23 for `RC4-HMAC` The following example requests for the `AES256-CTS` encryption type: `okinit -e 1 -e 18 krbuser@REALM` Note that you can repeat the option to request multiple encryption types.
`-?`	List command line options.

oklist Utility Options for Displaying Credentials

The `oklist` utility displays the list of tickets held.

The following table lists the available `oklist` options. To use the functionality that is described in this table, you must set the `sqlnet.ora` `SQLNET.KERBEROS5_CONF_MIT` parameter to `TRUE`. (Note that `SQLNET.KERBEROS5_CONF_MIT` is deprecated, but is retained for backward compatibility for `oklist`.)

Table 19-3 Options for the oklist Utility

Option	Description
`-f`	Show flags with credentials. Relevant flags are: • `I`, credential is a ticket-granting ticket • `F`, credential is forwardable • `f`, credential is forwarded.
`-c`	Specify an alternative credential cache. In UNIX, the default is `/tmp/krb5cc_uid`. The alternate credential cache can also be specified by using the `SQLNET.KERBEROS5_CC_NAME` parameter in the `sqlnet.ora` file.
`-k`	List the entries in the service table (default `/etc/v5srvtab`) on UNIX. The alternate service table can also be specified by using the `SQLNET.KERBEROS5_KEYTAB` parameter in the `sqlnet.ora` file.
`-e`	Displays the encryption types of the session key and the ticket for each credential in the credential cache, or each key in the keytab file.
`-l`	If a cache collection is available, displays a table summarizing the caches present in the collection.
`-A`	If a cache collection is available, displays the contents of all of the caches in the collection
`-s`	Runs utility without producing output. Utility will exit with status 1 if the cache cannot be read or is expired, else with status 0
`-a`	Displays a list of addresses in the credential
`-n`	Shows numeric addresses instead of reverse-resolving addresses
`-C`	Lists configuration data that has been stored in the credentials cache when `klist` encounters it. By default, configuration data is not listed.
`-t`	Displays the time entry timestamps for each keytab entry in the keytab file
`-K`	Displays the value of the encryption key in each keytab entry in the keytab file
`-V`	Displays the Kerberos version number and exit.

The show flag option (`-f`) displays additional information, as shown in the following example:

```
% oklist -f
04-Aug-2015 21:57:51   28-Aug-2015 05:58:14
krbtgt/EXAMPLE.COM@EXAMPLE.COM
Flags: FI
```

okdstry Utility Options for Removing Credentials from the Cache File

The `okdstry` (`okdestroy`) utility removes credentials from the cache file.

The following table lists the available `okdstry` options. To use the functionality that is described in this table, you must set the `sqlnet.ora` `SQLNET.KERBEROS5_CONF_MIT` parameter to `TRUE`. (Note that `SQLNET.KERBEROS5_CONF_MIT` is deprecated, but is retained for backward compatibility for `okdstry`.)

Table 19-4   Options for the okdstry Utility

Option	Description
`–A`	Destroys all caches in the collection, if a cache collection is available
`–q`	Runs quietly. Normally `okdstry` beeps if it fails to destroy the user's tickets. This flag suppresses this behavior.
`–c` *`cache_name`*	Uses *`cache_name`* as the credentials (ticket) cache name and location. For UNIX, the default is `/tmp/krb5cc_`*`uid`*. You can also specify the alternate credential cache by using the `SQLNET.KERBEROS5_CC_NAME` parameter in the `sqlnet.ora` file.

okcreate Utility Options for Automatic Keytab Creation

The `okcreate` utility automates the creation of keytabs from either the KDC or a service endpoint.

The following table lists the available `okcreate` options.

Table 19-5   okcreate Utility Options for Automatic Keytab Creation

Option	Description
`-name` *`service_name`*	Specifies the service name of the kerberized service for which to get a keytab.The default is `oracle`.
`–hosts` *`path-to_hosts_list`*	Specifies either a comma-separated list of hosts for which to get the keytab, or the path to a text file that contains a list of the hosts. The default is `none`.
`–out` *`path_to_output`*	Specifies the output path to store the resulting keytabs. The default is the current directory. Ensure that this directory is readable only by the root user. Never send keytabs over the network in clear text.
`–k`	For use if the operation is performed on the KDC. Do not use this option if you are using `–s`.
`–s`	For use if the operation is performed on a Kerberized service. Do not use this option if you are using `–k`.
`-u` *`KDC_username`*	Specifies the user name for the KDC. Only use this setting on a Kerberized service endpoint. If you specify the `–s` and omit this setting, then okcreate prompts for the *`KDCuser@KDCmachine`*.
`-r`	Specifies the Kerberos realm
`–p`	Specifies the Kerberos principal
`-q`	Specifies the Kerberos query
`–d`	Specifies the KDC database name
`–e`	Specifies the salt list to be used for any new keys that are created
`–m`	Specifies to prompt for the KDC master password

Connecting to an Oracle Database Server Authenticated by Kerberos

After Kerberos is configured, you can connect to an Oracle database server without using a user name or password.

- Use the following syntax to connect to the database without using a user name or password:

  ```
  $ sqlplus /@net_service_name
  ```

In this specification, *net_service_name* is an Oracle Net Services service name. For example:

```
$ sqlplus /@oracle_dbname
```

See Also:

Oracle Database Heterogeneous Connectivity User's Guide for information about external authentication

Configuring Interoperability with a Windows 2008 Domain Controller KDC

You can configure Oracle Database to interoperate with a Microsoft Windows 2008 domain controller key distribution center (KDC).

About Configuring Interoperability with a Windows 2008 Domain Controller KDC

Oracle Database complies with MIT Kerberos.

This enables Oracle Database to interoperate with tickets that are issued by a Kerberos Key Distribution Center (KDC) on a Windows 2008 domain controller. This process enables Kerberos authentication with an Oracle database.

Step 1: Configure Oracle Kerberos Client for Windows 2008 Domain Controller

You can configure the Oracle Kerberos client to interoperate with a Microsoft Windows 2008 Domain Controller KDC.

Step 1A: Create the Client Kerberos Configuration Files

You must configure a set of client Kerberos configuration files that refer to the Windows 2008 domain controller as the Kerberos KDC.

- Create the `krb.conf` and `krb5.realms` files. Oracle Database provides a default `krb5.conf` file, which you must modify for your site.

  The `krb5.conf` file is located in the location indicated by the `SQLNET.KERBEROS_CONF` parameter.

For example, assuming that the Windows 2008 domain controller is running on a node named `sales3854.us.example.com`:

- **`krb.conf`** file

  For example:

  ```
  SALES3854.US.EXAMPLE.COM
  SALES3854.US.EXAMPLE.COM
  sales3854.us.example.com admin server
  ```

- **`krb5.conf`** file

  For example:

  ```
  [libdefaults]
  default_realm=SALES.US.EXAMPLE.COM
  [realms]
  SALES.US.EXAMPLE.COM= { kdc=sales3854.us.example.com:88 }
  [domain_realm]
  .us.example.com=SALES.US.EXAMPLE.COM
  ```

- **`krb5.realms`** file

  For example:

  ```
  us.example.com SALES.US.EXAMPLE.COM
  ```

Step 1B: Specify the Oracle Configuration Parameters in the sqlnet.ora File

The `sqlnet.ora` file has Kerbose 5–specific parameters.

Configuring an Oracle client to interoperate with a Windows 2008 domain controller KDC uses the same `sqlnet.ora` file parameters that are used for Kerberos on the client and on the database server. These parameters are described in Step 6A: Configure Kerberos on the Client and on the Database Server.

- Set the following parameters in the `sqlnet.ora` file on the client:

  ```
  SQLNET.KERBEROS5_CONF=pathname_to_Kerberos_configuration_file
  SQLNET.KERBEROS5_CONF_MIT=TRUE
  SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=Kerberos_service_name
  SQLNET.AUTHENTICATION_SERVICES=(BEQ,KERBEROS5)
  ```

Note:

- The `SQLNET.KERBEROS5_CONF_MIT` parameter has been deprecated, but is retained for backward compatibility for the `okint`, `oklist`, and `okdstry` utilities.
- Ensure that the `SQLNET.KERBEROS5_CONF_MIT` parameter is set to `TRUE` because the Windows 2008 operating system is designed to interoperate only with security services that are based on MIT Kerberos version 5.

Step 1C: Specify the Listening Port Number

The Windows 2008 domain controller KDC listens on UDP/TCP port 88.

- Ensure that the system file entry for `kerberos5` is set to UDP/TCP port 88.

  For the UNIX environment, ensure that the first `kerberos5` entry in the `/etc/services` file is set to 88.

Step 2: Configure a Windows 2008 Domain Controller KDC for the Oracle Client

Next, you are ready to configure a Microsoft Windows 2008 Domain Controller KDC to interoperate with an Oracle Client.

See Also:

Microsoft documentation for information about how to create users in Active Directory.

Step 2A: Create the User Account

You must create a user account for the Microsoft Windows 2008 Domain Controller KDC.

- On the Windows 2008 domain controller, create a new user account for the Oracle client in Microsoft Active Directory.

Step 2B: Create the Oracle Database Principal User Account and Keytab

After you create the user account, you are ready to create the Oracle Database principal user account.

After you create this account on the Windows 2008 domain controller, you must use the `okcreate` utility to register it with the principal keytab. You can run this utilty on the same KDC to create all the service keytabs rather than creating them individually, or you can run `okcreate` from a service endpoint that connects to the KDC, run the ncessary commands, and then copy the resulting keytab back to the service endpoint.

1. Create a new user account for the Oracle database in Microsoft Active Directory.

For example, if the Oracle database runs on the host `sales3854.us.example.com`, then use Active Directory to create a user with the user name `sales3854.us.example.com` and the password `oracle`.

Do not create a user as `host/hostname.dns.com`, such as `oracle/sales3854.us.example.com`, in Active Directory. Microsoft's KDC does not support multipart names like an MIT KDC does. An MIT KDC allows multipart names to be used for service principals because it treats all principals as user names. However, Microsoft's KDC does not.

2. Run the `okcreate` command to create a keytab that will use this user account. The syntax is as follows:

```
okcreate (-s [-u KDCuser@KDCmachine] | -k)
  [-name service_name] [-hosts path_to_host_list]
  [-out path_to_output] [-r realm] [-p principal]
  [-q query] [-d dbname] [-e enc:salt...] [-m]
  [-x db_args]
```

   For example:

```
okcreate -s -u kdcuser1@kdcmachine1 -name oracle
  -hosts sales3854.us.example.com
  -out /OSsecured/keytablocation
```

3. Copy the extracted `keytab` file to the host computer where the Oracle database is installed.

   For example, the `keytab` that was created in the previous step can be copied to `/krb5/v5svrtab`.

See Also:

- Kerberos Authentication in Windows 2000 Server for detailed information about Windows 2008 interoperability with Kerberos 5
- okcreate Utility Options for Automatic Keytab Creation

Step 3: Configure Oracle Database for a Windows 2008 Domain Controller KDC

You must configure the Oracle database for the domain controller on the host computer where the Oracle database is installed.

Step 3A: Set Configuration Parameters in the sqlnet.ora File

You must first set configuration parameters for the database.

- Specify values for the following parameters in the `sqlnet.ora` file for the database server:

```
SQLNET.KERBEROS5_CONF=pathname_to_Kerberos_configuration_file
SQLNET.KERBEROS5_KEYTAB=pathname_to_Kerberos_principal/key_table
SQLNET.KERBEROS5_CONF_MIT=TRUE
SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=Kerberos_service_name
SQLNET.AUTHENTICATION_SERVICES=(BEQ,KERBEROS5)
```

Note:

- The `SQLNET.KERBEROS5_CONF_MIT` parameter has been deprecated, but is retained for backward compatibility for the `okint`, `oklist`, and `okdstry` utilities.
- Ensure that the `SQLNET.KERBEROS5_CONF_MIT` parameter is set to `TRUE` because the Windows 2008 operating system is designed to interoperate only with security services that are based on MIT Kerberos version 5.
- Be aware that in a multitenant environment, the settings in the `sqlnet.ora` file apply to all PDBs. However, this does not mean that all PDBs must authenticate with one KDC if using Kerberos; the settings in the `sqlnet.ora` file and Kerberos configuration files can support multiple KDCs.

Step 3B: Create an Externally Authenticated Oracle User

After you set the configuration parameters, you are ready to create an externally authenticated Oracle user.

- Follow the procedure under Step 8: Create an Externally Authenticated Oracle User to create an externally authenticated Oracle user.

  Ensure that you create the username in all uppercase characters (for example, `ORAKRB@SALES.US.EXAMPLE.COM`).

See Also:

Step 6: Configure Kerberos Authentication for information about using Oracle Net Manager to set the `sqlnet.ora` file parameters.

Step 4: Obtain an Initial Ticket for the Kerberos/Oracle User

Before a client can connect to the database, the client must request an initial ticket.

- To request an initial ticket, follow the task information for Step 9: Get an Initial Ticket for the Kerberos/Oracle User.

Note:

The user does not need to explicitly request for an initial ticket, using the `okinit` command, when using the Windows native cache.

If the Oracle client is running on Microsoft Windows 2008 or later, the Kerberos ticket is automatically retrieved when the user logs in to Windows.

See Also:

Microsoft documentation for details about the `Kerbtray.exe` utility, which can be used to display Kerberos ticket information for a system

Configuring Kerberos Authentication Fallback Behavior

You can configure fallback behavior (password-based authentication) in case the Kerberos authentication fails.

After you have configured Kerberos authentication for Oracle clients to use Kerberos authentication to authenticate to an Oracle database, there are cases where you may want to fall back to password-based authentication. An example would be if you have fixed user database links in the Oracle database.

- To enable Kerberos authentication to fall back to password-based authentication, set the `SQLNET.FALLBACK_AUTHENTICATION` parameter to `TRUE` in the `sqlnet.ora` files on both the client and server.

  The default of this parameter is `FALSE`. This means that by default, the connection fails when Kerberos authentication fails.

See Also:

Oracle Database Net Services Reference for more information about the `SQLNET.FALLBACK_AUTHENTICATION` parameter

Troubleshooting the Oracle Kerberos Authentication Configuration

Oracle provides guidance for common Kerberos configuration problems.

Common problems are as follows:

- If you cannot get your ticket-granting ticket using `okinit`:
  - Ensure that the default realm is correct by examining the `krb.conf` file.
  - Ensure that the KDC is running on the host specified for the realm.
  - Ensure that the KDC has an entry for the user principal and that the passwords match.
  - Ensure that the `krb.conf` and `krb.realms` files are readable by Oracle.
  - Ensure that the `TNS_ADMIN` environment variable is pointing to the directory containing the `sqlnet.ora` configuration file.
- If you have an initial ticket but still cannot connect:
  - After trying to connect, check for a service ticket.
  - Check that the `sqlnet.ora` file on the database server side has a service name that corresponds to a service known by Kerberos.

  - Check that the clocks on all systems involved are set to times that are within a few minutes of each other or change the `SQLNET.KERBEROS5_CLOCKSKEW` parameter in the `sqlnet.ora` file.

- If you have a service ticket and you still cannot connect:
  - Check the clocks on the client and database server.
  - Check that the `v5srvtab` file exists in the correct location and is readable by Oracle. Remember to set the `sqlnet.ora` parameters.
  - Check that the `v5srvtab` file has been generated for the service named in the `sqlnet.ora` file on the database server side.
- If everything seems to work fine, but then you issue another query and it fails:
  - Check that the initial ticket is forwardable. You must have obtained the initial ticket by running the `okinit` utility.
  - Check the expiration date on the credentials. If the credentials have expired, then close the connection and run `okinit` to get a new initial ticket.

20 Configuring Secure Sockets Layer Authentication

You can configure Oracle Database to use Secure Sockets Layer authentication.

Secure Sockets Layer and Transport Layer Security

Netscape Communications Corporation designed Secure Sockets Layer (SSL) to secure network connections.

The Difference Between Secure Sockets Layer and Transport Layer Security

Transport Layer Security (TLS) is an incremental version of Secure Sockets Layer (SSL) version 3.0.

Although SSL was primarily developed by Netscape Communications Corporation, the Internet Engineering Task Force (IETF) took over development of it, and renamed it Transport Layer Security (TLS).

Note:

To simplify discussion, this chapter uses the term SSL where either SSL or TLS may be appropriate because SSL is the most widely recognized term. However, where distinctions occur between how you use or configure these protocols, this chapter specifies what is appropriate for either SSL or TLS.

See Also:

The TLS Protocol Version 1.0 [RFC 2246] at the IETF Web site

Using Transport Layer Security in a Multitenant Environment

Transport Layer Security (TLS) can be used in a multitenant environment for application containers.

If you want to use Transport Layer Security (TLS) in a multitenant environment for an application container, then you must ensure that each PDB is able to use its own wallet with its own certificates for TLS authentication.

- Because there is no individual `sqlnet.ora` file for each PDB, place the wallet in a subdirectory of the `wallet` directory where the name of the subdirectory is the GUID of the PDB that uses the wallet.

For example, suppose the `WALLET_LOCATION` parameter in `sqlnet.ora` is set as follows:

```
(SOURCE=(METHOD=FILE)(METHOD_DATA=
   (DIRECTORY=/home/oracle/wallet)))
```

Place each PDB's wallet in the `/home/oracle/wallet` directory. You can find the existing PDBs and their GUIDs by querying the `DBA_PDBS` data dictionary view.

If the `WALLET_LOCATION` parameter is not specified, then you must place the PDB wallet in a subdirectory of the default wallet path where the name of the subdirectory is the GUID of the PDB. For example:

```
$ORACLE_BASE/admin/db_unique_name/wallet/PDB_GUID
```

Or if the `ORACLE_BASE` environment variable is not set, then you can use the Oracle home:

```
$ORACLE_HOME/admin/db_unique_name/wallet/PDB_GUID
```

These default locations correspond to the default that is used by Oracle Enterprise User Security to locate wallets for authentication to LDAP.

How Oracle Database Uses Secure Sockets Layer for Authentication

Secure Sockets Layer works with the core Oracle Database features such as encryption and data access controls.

By using Oracle Database SSL functionality to secure communications between clients and servers, you can

- Use SSL to encrypt the connection between clients and servers
- Authenticate any client or server, such as Oracle Application Server 10g, to any Oracle database server that is configured to communicate over SSL

You can use SSL features by themselves or in combination with other authentication methods supported by Oracle Database. For example, you can use the encryption provided by SSL in combination with the authentication provided by Kerberos. SSL supports any of the following authentication modes:

- Only the server authenticates itself to the client
- Both client and server authenticate themselves to each other
- Neither the client nor the server authenticates itself to the other, thus using the SSL encryption feature by itself

See Also:

The SSL Protocol, version 3.0, published by the Internet Engineering Task Force, for a more detailed discussion of SSL

How Secure Sockets Layer Works in an Oracle Environment: The SSL Handshake

When a network connection over Secure Sockets Layer is initiated, the client and server perform an SSL handshake before performing the authentication.

The handshake process is as follows:

1. The client and server establish which cipher suites to use. This includes which encryption algorithms are used for data transfers.
2. The server sends its certificate to the client, and the client verifies that the server's certificate was signed by a trusted CA. This step verifies the identity of the server.
3. Similarly, if client authentication is required, the client sends its own certificate to the server, and the server verifies that the client's certificate was signed by a trusted CA.
4. The client and server exchange key information using public key cryptography. Based on this information, each generates a session key. All subsequent communications between the client and the server is encrypted and decrypted by using this session key and the negotiated cipher suite.

The authentication process is as follows:

1. On a client, the user initiates an Oracle Net connection to the server by using SSL.
2. SSL performs the handshake between the client and the server.
3. If the handshake is successful, then the server verifies that the user has the appropriate authorization to access the database.

Public Key Infrastructure in an Oracle Environment

A public key infrastructure (PKI) is a substrate of network components that provide a security underpinning, based on trust assertions, for an entire organization.

About Public Key Cryptography

Traditional private-key or symmetric-key cryptography requires a single, secret key shared by two or more parties to establish a secure communication.

This key is used to both encrypt and decrypt secure messages sent between the parties, requiring prior, secure distribution of the key to each party. The problem with this method is that it is difficult to securely transmit and store the key.

Public-key cryptography provides a solution to this problem, by employing public and private key pairs and a secure method for key distribution. The freely available public key is used to encrypt messages that can *only* be decrypted by the holder of the associated private key. The private key is securely stored, together with other security credentials, in an encrypted container called a wallet.

Public-key algorithms can guarantee the secrecy of a message, but they do not necessarily guarantee secure communications because they do not verify the identities of the communicating parties. To establish secure communications, it is important to verify that the public key used to encrypt a message does in fact belong to the target recipient. Otherwise, a third party can potentially eavesdrop on the communication and intercept public key requests, substituting its own public key for a legitimate key (the man-in-the-middle attack).

In order to avoid such an attack, it is necessary to verify the owner of the public key, a process called authentication. Authentication can be accomplished through a certificate authority (CA), which is a third party that is trusted by both of the communicating parties.

The CA issues public key certificates that contain an entity's name, public key, and certain other security credentials. Such credentials typically include the CA name, the CA signature, and the certificate effective dates (From Date, To Date).

The CA uses its private key to encrypt a message, while the public key is used to decrypt it, thus verifying that the message was encrypted by the CA. The CA public key is well known and does not have to be authenticated each time it is accessed. Such CA public keys are stored in wallets.

Public Key Infrastructure Components in an Oracle Environment

Public key infrastructure (PKI) components in an Oracle environment include a certificate authority, certificates, certificate revocation lists, and wallets.

Certificate Authority

A certificate authority (CA) is a trusted third party that certifies the identity of entities, such as users, databases, administrators, clients, and servers.

When an entity requests certification, the CA verifies its identity and grants a certificate, which is signed with the CA's private key.

Different CAs may have different identification requirements when issuing certificates. Some CAs may verify a requester's identity with a driver's license, some may verify identity with the requester's fingerprints, while others may require that requesters have their certificate request form notarized.

The CA publishes its own certificate, which includes its public key. Each network entity has a list of trusted CA certificates. Before communicating, network entities exchange certificates and check that each other's certificate is signed by one of the CAs on their respective trusted CA certificate lists.

Network entities can obtain their certificates from the same or different CAs. By default, Oracle Database automatically installs trusted certificates from VeriSign, RSA, Entrust, and GTE CyberTrust when you create a new wallet.

Certificates

A certificate is created when an entity's public key is signed by a trusted certificate authority (CA).

A certificate ensures that an entity's identification information is correct and that the public key actually belongs to that entity.

A certificate contains the entity's name, public key, and an expiration date, as well as a serial number and certificate chain information. It can also contain information about the privileges associated with the certificate.

When a network entity receives a certificate, it verifies that it is a trusted certificate, that is, one that has been issued and signed by a trusted certificate authority. A certificate remains valid until it expires or until it is revoked.

Certificate Revocation Lists

When a CA signs a certificate binding a public key pair to a user identity, the certificate is valid for a specified time.

However, certain events, such as user name changes or compromised private keys, can render a certificate invalid before the validity period expires. When this happens, the CA revokes the certificate and adds its serial number to a Certificate Revocation List (CRL). The CA periodically publishes CRLs to alert the user population when it is no longer acceptable to use a particular public key to verify its associated user identity.

When servers or clients receive user certificates in an Oracle environment, they can validate the certificate by checking its expiration date, signature, and revocation status. Certificate revocation status is checked by validating it against published CRLs. If certificate revocation status checking is turned on, then the server searches for the appropriate CRL depending on how this feature has been configured. The server searches for CRLs in the following locations in this order:

1. Local file system
2. Oracle Internet Directory
3. CRL Distribution Point, a location specified in the CRL Distribution Point (CRL DP) X.509, version 3, certificate extension when the certificate is issued.

Note:

To use CRLs with other Oracle products, refer to the specific product documentation. This implementation of certificate validation with CRLs is only available in the Oracle Database 12*c* release 1 (12.1) SSL adapter.

Wallets

A wallet is a container that stores authentication and signing credentials, including private keys, certificates, and trusted certificates SSL needs.

In an Oracle environment, every entity that communicates over SSL must have a wallet containing an X.509 version 3 certificate, private key, and list of trusted certificates, with the exception of Diffie-Hellman.

Security administrators use Oracle Wallet Manager to manage security credentials on the server. Wallet owners use it to manage security credentials on clients. Specifically, you use Oracle Wallet Manager to do the following:

- Generate a public-private key pair and create a certificate request
- Store a user certificate that matches with the private key
- Configure trusted certificates

See Also:

- *Oracle Database Enterprise User Security Administrator's Guide* for information about Oracle Wallet Manager
- *Oracle Database Enterprise User Security Administrator's Guide* for information about creating a new Oracle wallet
- *Oracle Database Enterprise User Security Administrator's Guide* for information about managing trusted certificates in Oracle wallets

Hardware Security Modules

The hardware security modules for SSL include devices to handle various functions and hardware devices to store cryptographic information.

Oracle Database uses these devices for the following functions:

- Store cryptographic information, such as private keys
- Perform cryptographic operations to off load RSA operations from the server, freeing the CPU to respond to other transactions

Cryptographic information can be stored on two types of hardware devices:

- (Server-side) Hardware boxes where keys are stored in the box, but managed by using tokens.
- (Client-side) Smart card readers, which support storing private keys on tokens.

An Oracle environment supports hardware devices using APIs that conform to the RSA Security, Inc., Public-Key Cryptography Standards (PKCS) #11 specification.

Note:

Currently, SafeNET and nCipher devices are certified with Oracle Database

Secure Sockets Layer Combined with Other Authentication Methods

You can configure Oracle Database to use SSL concurrently with database user names and passwords, RADIUS, and Kerberos.

Architecture: Oracle Database and Secure Sockets Layer

It is important to understand the architecture of how Oracle Database works with SSL.

Figure 17-4 , which displays the Oracle Database implementation of Secure Sockets Layer architecture, shows that Oracle Databases operates at the session layer on top of SSL and uses TCP/IP at the transport layer.

This separation of functionality lets you employ SSL concurrently with other supported protocols.

See Also:

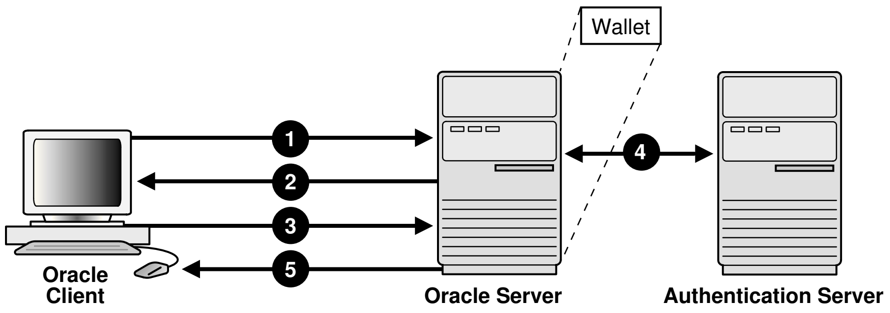
Oracle Database Net Services Administrator's Guide for information about stack communications in an Oracle networking environment

How Secure Sockets Layer Works with Other Authentication Methods

Secure Sockets Layer can be used with other authentication methods that Oracle Database supports.

Figure 20-1 illustrates a configuration in which Secure Sockets Layer is used in combination with another authentication method.

Figure 20-1 Secure Sockets Layer in Relation to Other Authentication Methods



In this example, Secure Sockets Layer is used to establish the initial handshake (server authentication), and an alternative authentication method is used to authenticate the client. The process is as follows:

1. The client seeks to connect to the Oracle database server.
2. Secure Sockets Layer performs a handshake during which the server authenticates itself to the client and both the client and server establish which cipher suite to use.
3. Once the Secure Sockets Layer handshake is successfully completed, the user seeks access to the database.
4. The Oracle database server authenticates the user with the authentication server using a non-SSL authentication method such as Kerberos or RADIUS.
5. Upon validation by the authentication server, the Oracle database server grants access and authorization to the user, and then the user can access the database securely by using SSL.

Secure Sockets Layer and Firewalls

Oracle Database supports two application proxy-based and stateful packet inspection of firewalls.

These firewalls are as follows:

- **Application proxy-based firewalls:** Examples are Network Associates Gauntlet, or Axent Raptor.
- **Stateful packet inspection firewalls:** Examples are Check Point Firewall-1, or Cisco PIX Firewall.

When you enable SSL, stateful inspection firewalls behave like application proxy firewalls because they do not decrypt encrypted packets.

Firewalls do not inspect encrypted traffic. When a firewall encounters data addressed to an SSL port on an intranet server, it checks the target IP address against its access rules and lets the SSL packet pass through to permitted SSL ports, rejecting all others.

With the Oracle Net Firewall Proxy kit, a product offered by some firewall vendors, firewall applications can provide specific support for database network traffic. If the proxy kit is implemented in the firewall, then the following processing takes place:

- The Net Proxy (a component of the Oracle Net Firewall Proxy kit) determines where to route its traffic.
- The database listener requires access to a certificate in order to participate in the SSL handshake. The listener inspects the SSL packet and identifies the target database, returning the port on which the target database listens to the client. This port must be designated as an SSL port.
- The client communicates on this server-designated port in all subsequent connections.

Secure Sockets Layer Usage Issues

You should be aware of SSL usage issues, such as communication with other Oracle products and types of supported authentication and encryption methods.

Consider the following issues when using SSL:

- SSL use enables secure communication with other Oracle products, such as Oracle Internet Directory.
- Because SSL supports both authentication and encryption, the client/server connection is somewhat slower than the standard Oracle Net TCP/IP transport (using native encryption).
- Each SSL authentication mode requires configuration settings.

Note:

If you configure SSL encryption, you must disable non-SSL encryption. To disable such encryption, refer to Disabling Strong Authentication and Network Encryption .

Enabling Secure Sockets Layer

You must configure Secure Sockets Layer on the server, and then the client.

Step 1: Configure Secure Sockets Layer on the Server

During installation, Oracle sets defaults on the Oracle database server and the Oracle client for SSL parameters, except the Oracle wallet location.

Step 1A: Confirm Wallet Creation on the Server

Before proceeding to the next step, confirm that a wallet has been created and that it has a certificate.

1. Start Oracle Wallet Manager.
   - (UNIX) From `$ORACLE_HOME/bin`, enter the following command:

     ```
     owm
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle-HOME_NAME**, **Integrated Management Tools**, **Wallet Manager**
2. From the **Wallet** menu, select **Open**.

   The wallet should contain a certificate with a status of `Ready` and auto-login turned on. If auto-login is not on, then select it from the **Wallet** menu and save the wallet again. This turns auto-login on.

See Also:

Oracle Database Enterprise User Security Administrator's Guide for information about creating a new Oracle wallet

Step 1B: Specify the Database Wallet Location on the Server

Next, you are ready to specify a location on the server for the wallet.

1. Start Oracle Net Manager.
   - (UNIX) From `$ORACLE_HOME/bin`, enter the following command at the command line:

     ```
     netmgr
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **SSL** tab and then select **Configure SSL for: Server**.
5. In the **Wallet Directory** box, enter the directory in which the Oracle wallet is located or click **Browse** to find it by searching the file system.

   Note that if you are configuring the database-to-directory SSL connection for Enterprise User Security, then Database Configuration Assistant automatically creates a database wallet while registering the database with the directory. You must use that wallet to store the database PKI credentials for SSL-authenticated Enterprise User Security.

Important:

- Use Oracle Wallet Manager to create the wallet. See *Oracle Database Enterprise User Security Administrator's Guide* for information about creating a new Oracle wallet.
- Use Oracle Net Manager to set the wallet location in the `sqlnet.ora` file. Be aware that in a multitenant environment, the settings in the `sqlnet.ora` file apply to all pluggable databases (PDBs).

Ensure that you enter the same wallet location when you create it and when you set the location in the `sqlnet.ora` file.

6. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` and `listener.ora` files are updated with the following entries:

```
wallet_location =
 (SOURCE=
  (METHOD=File)
  (METHOD_DATA=
   (DIRECTORY=wallet_location)))
```

Note:

The listener uses the wallet defined in the `listener.ora` file. It can use any database wallet. When SSL is configured for a server using Net Manager, the wallet location is entered into the `listener.ora` and the `sqlnet.ora` files. The `listener.ora` file is not relevant to the Oracle client.

To change the listener wallet location so that the listener has its own wallet, you can edit `listener.ora` to enter the new location.

Step 1C: Set the Secure Sockets Layer Cipher Suites on the Server (Optional)

Optionally, you can set the Secure Sockets Layer cipher suites.

About the Secure Sockets Layer Cipher Suites

A cipher suite is a set of authentication, encryption, and data integrity algorithms used for exchanging messages between network entities.

During a Secure Sockets Layer handshake, two entities negotiate to see which cipher suite they will use when transmitting messages back and forth.

When you install Oracle Database, the Secure Sockets Layer cipher suites listed in Table 20-1 are set for you by default and negotiated in the order they are listed. You can override the default order by setting the `SSL_CIPHER_SUITES` parameter. Ensure that you enclose the `SSL_CIPHER_SUITES` parameter setting in parentheses (for example, `SSL_CIPHER_SUITES=(ssl_rsa_with_aes_128_cbc_sha256)`). Otherwise, the cipher suite setting will not parse correctly.

You can prioritize the cipher suites. When the client negotiates with servers regarding which cipher suite to use, it follows the prioritization you set. When you prioritize the cipher suites, consider the following:

- **Compatibility.** Server and client must be configured to use compatible cipher suites for a successful connection.
- **Cipher priority and strength.** Prioritize cipher suites starting with the strongest and moving to the weakest to ensure the highest level of security possible.
- **The level of security you want to use.**
- **The impact on performance.**

Note:

Regarding Diffie-Hellman anonymous authentication:

– If you set the server to employ this cipher suite, then you must also set the same cipher suite on the client. Otherwise, the connection fails.

– If you use a cipher suite employing Diffie-Hellman anonymous, then you must set the `SSL_CLIENT_AUTHENTICATION` parameter to `FALSE`. For more information, refer to Step 1E: Set SSL Client Authentication on the Server (Optional).

– There is a known bug in which an OCI client requires a wallet even when using a cipher suite with DH_ANON, which does not authenticate the client.

SSL Cipher Suite Authentication, Encryption, Integrity, and TLS Versions

Oracle Database supports a set of cipher suites that are set by default when you install Oracle Database.

Table 20-1 lists the authentication, encryption, and data integrity types each cipher suite uses.

Table 20-1 Secure Sockets Layer Cipher Suites

Cipher Suites	Authentication	Encryption	Data Integrity	TLS Compatibility
`SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`	ECDHE_ECDSA	AES 128 GCM	SHA256 (SHA-2)	TLS 1.2 only
`SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA`	ECDHE_ECDSA	AES 128 CBC	SHA-1	TLS 1.0 and later
`SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256`	ECDHE_ECDSA	AES 128 CBC	SHA256 (SHA-2)	TLS 1.2 only
`SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`	ECDHE_ECDSA	AES 256 CBC	SHA-1	TLS 1.0 and later
`SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`	ECDHE_ECDSA	AES 256 CBC	SHA384 (SHA-2)	TLS 1.2 only

Table 20-1 (Cont.) Secure Sockets Layer Cipher Suites

Cipher Suites	Authentication	Encryption	Data Integrity	TLS Compatibility
`SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`	ECDHE_ECDSA	AES 256 GCM	SHA384 (SHA-2)	TLS 1.2 only
`SSL_RSA_WITH_AES_128_CBC_SHA256`	RSA	AES 128 CBC	SHA256 (SHA-2)	TLS 1.2 only
`SSL_RSA_WITH_AES_128_GCM_SHA256`	RSA	AES 128 GCM	SHA256 (SHA-2)	TLS 1.2 only
`SSL_RSA_WITH_AES_128_CBC_SHA`	RSA	AES 128 CBC	SHA-1	TLS 1.0 only
`SSL_RSA_WITH_AES_256_CBC_SHA`	RSA	AES 256 CBC	SHA-1	TLS 1.0 and later
`SSL_RSA_WITH_AES_256_CBC_SHA256`	RSA	AES 256 CBC	SHA256 (SHA-2)	TLS 1.2 only
`SSL_RSA_WITH_AES_256_GCM_SHA384`	RSA	AES 256 GCM	SHA384 (SHA-2)	TLS 1.2 only

Table 20-2 lists cipher suites that you can use, but be aware that they do not the provide authentication of the communicating parties, and hence can be vulnerable to man-in-the-middle attacks. Oracle recommends that you do not use these cipher suites to protect sensitive data. However, they are useful if the communicating parties want to remain anonymous or simply do not want the overhead caused by mutual authentication.

Table 20-2 SSL_DH Secure Sockets Layer Cipher Suites

Cipher Suites	Authentication	Encryption	Data Integrity	TLS Compatibility
`SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`	DH anon	3DES EDE CBC	SHA-1	TLS 3.0 and later

Specifying Secure Sockets Cipher Suites for the Database Server

First, you must specify the Secure Sockets cipher suites for the database server.

1. Start Oracle Net Manager.
   - (UNIX) From `$ORACLE_HOME/bin`, enter the following command at the command line:

     ```
     netmgr
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

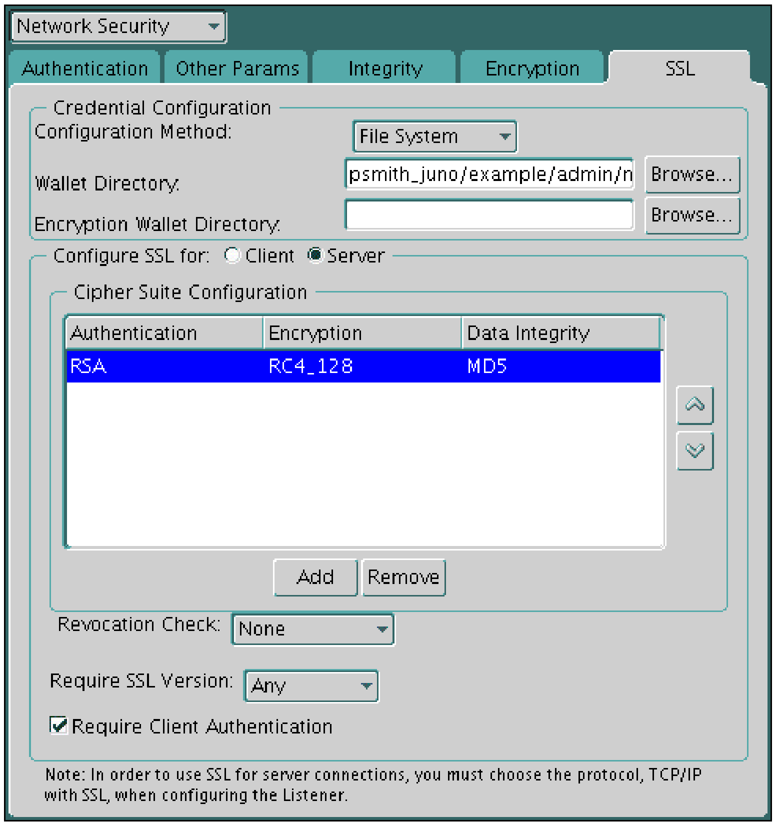
   The Network Security tabbed window appears.
4. Select the **SSL** tab and then select **Configure SSL for: Server**.

5. In the Cipher Suite Configuration area, click **Add**.

   A dialog box displays available cipher suites. To see the US domestic cipher suites, click the **Show US Domestic Cipher Suits** check box.

6. Select a suite and click **OK**.

   The **Cipher Suite Configuration** list is updated:



Credential Configuration
Configuration Method: File System
Wallet Directory: psmith_juno/example/admin/n
Encryption Wallet Directory:
Configure SSL for: ◯ Client ● Server
Cipher Suite Configuration

Authentication	Encryption	Data Integrity
RSA	RC4_128	MD5

Revocation Check: None
Require SSL Version: Any
☑ Require Client Authentication

Note: In order to use SSL for server connections, you must choose the protocol, TCP/IP with SSL, when configuring the Listener.

7. Use the up and down arrows to prioritize the cipher suites.

8. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry:

   ```
   SSL_CIPHER_SUITES= (SSL_cipher_suite1 [,SSL_cipher_suite2])
   ```

Step 1D: Set the Required Secure Sockets Layer Version on the Server (Optional)

The `SSL_VERSION` parameter defines the version of SSL that must run on the systems with which the server communicates.

Optionally, you can set the `SSL_VERSION` parameter in the `sqlnet.ora` or the `listener.ora` file.

You can require these systems to use any valid version. The default setting for this parameter in `sqlnet.ora` is `undetermined`, which is set by selecting **Any** from the list in the SSL tab of the Network Security window.

1. In the Require SSL Version list, the default is **Any**.

Accept this default or select the SSL version you want to use.

2. From the **File** menu, select **Save Network Configuration**.

   If you chose **Any**, then the `sqlnet.ora` file is updated with the following entry:

   ```
   SSL_VERSION=UNDETERMINED
   ```

> **Note:**
>
> SSL 2.0 is not supported on the server side.

> **See Also:**
>
> *Oracle Database Net Services Reference* for more information about the `SSL_VERSION` parameter

Step 1E: Set SSL Client Authentication on the Server (Optional)

The `SSL_CLIENT_AUTHENTICATION` parameter controls whether the client is authenticated using SSL.

You must set this parameter in the `sqlnet.ora` file on the server. The default value of `SSL_CLIENT_AUTHENTICATION` parameter is `TRUE`.

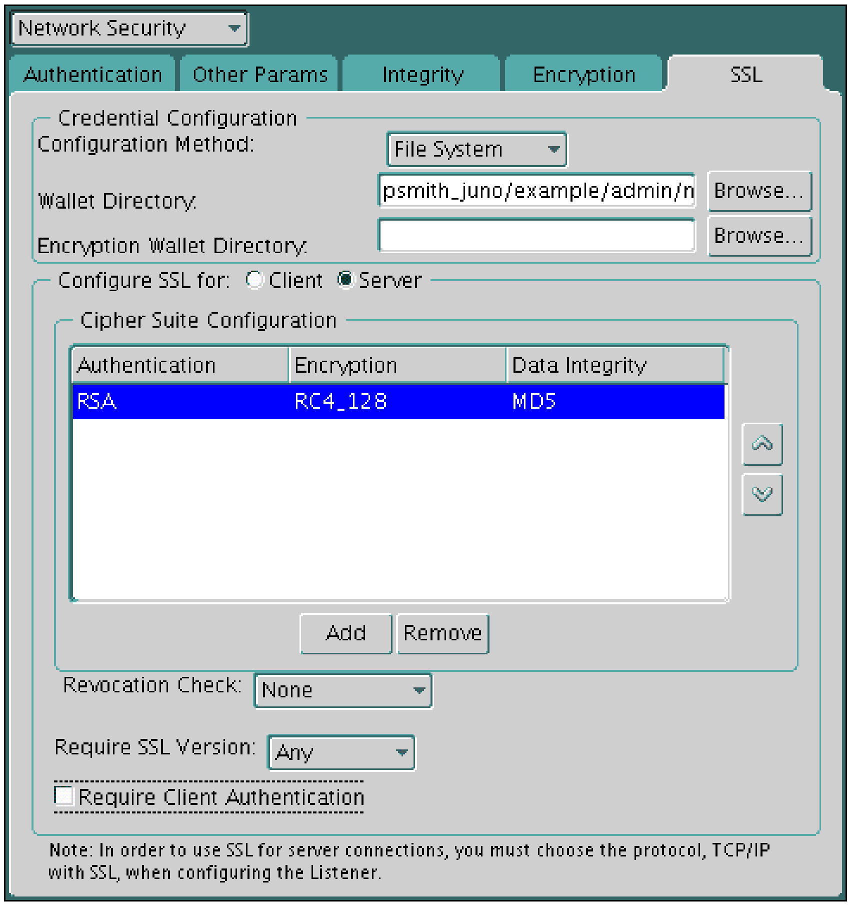
You can set the `SSL_CLIENT_AUTHENTICATION` to `FALSE` if you are using a cipher suite that contains Diffie-Hellman anonymous authentication (`DH_anon`).

Also, you can set this parameter to `FALSE` for the client to authenticate itself to the server by using any of the non-SSL authentication methods supported by Oracle Database, such as Kerberos or RADIUS.

> **Note:**
>
> There is a known bug in which an OCI client requires a wallet even when using a cipher suite with DH_ANON, which does not authenticate the client.

To set `SSL_CLIENT_AUTHENTICATION` to `FALSE` on the server:

1. In the SSL page Oracle Net Manager, deselect **Require Client Authentication**.



Network Security

Authentication | Other Params | Integrity | Encryption | SSL

Credential Configuration

Configuration Method: File System

Wallet Directory: psmith_juno/example/admin/n Browse...

Encryption Wallet Directory: Browse...

Configure SSL for: ○ Client ● Server

Cipher Suite Configuration

Authentication	Encryption	Data Integrity
RSA	RC4_128	MD5

Add Remove

Revocation Check: None

Require SSL Version: Any

☐ Require Client Authentication

Note: In order to use SSL for server connections, you must choose the protocol, TCP/IP with SSL, when configuring the Listener.

2. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry:

   ```
   SSL_CLIENT_AUTHENTICATION=FALSE
   ```

Step 1F: Set SSL as an Authentication Service on the Server (Optional)

The `SQLNET.AUTHENTICATION_SERVICES` parameter in the `sqlnet.ora` file sets the SSL authentication service.

Set this parameter if you want to use SSL authentication in conjunction with another authentication method supported by Oracle Database. For example, use this parameter if you want the server to authenticate itself to the client by using SSL and the client to authenticate itself to the server by using Kerberos.

- To set the `SQLNET.AUTHENTICATION_SERVICES` parameter on the server, add TCP/IP with SSL (TCPS) to this parameter in the `sqlnet.ora` file by using a text editor. For example, if you want to use SSL authentication in conjunction with RADIUS authentication, set this parameter as follows:

  ```
  SQLNET.AUTHENTICATION_SERVICES = (TCPS, radius)
  ```

If you do not want to use SSL authentication in conjunction with another authentication method, then do not set this parameter.

Step 1G: Disable SSLv3 on the Server and Client (Optional)

SSLv3 refers to Secure Sockets Layer version 3.

Applications that support Secure Sockets Layer version 3 (SSLv3) are vulnerable to Padding Oracle On Downgraded Legacy Encryption (POODLE) attacks, even if they use the most recent version of Transport Layer Security (TLS). To prevent POODLE attacks, you should set the `ADD_SSLV3_TO_DEFAULT` `sqlnet.ora` parameter to `FALSE` on both the server and the client. `ADD_SSLV3_TO_DEFAULT` only applies if the `SSL_VERSION` parameter is not set. (which means that you are using the default list of SSL versions).

1. Log in to the database server or the client server.
2. Edit the `sqlnet.ora` parameter file, which by default is located in the `$ORACLE_HOME/network/admin` directory, to include the `ADD_SSLV3_TO_DEFAULT` parameter, as follows:

   ```
   ADD_SSLV3_TO_DEFAULT=false
   ```

   `ADD_SSLV3_TO_DEFAULT` defaults to `FALSE`, and has no effect if `SSL_VERSION` is explicitly set. So, by default, without `SSL_VERSION` explicitly set, SSLv3 connections will not be allowed. Setting `ADD_SSLV3_TO_DEFAULT` to `TRUE` would allow SSLv3 connections to continue working by default.

Step 1H: Create a Listening Endpoint that Uses TCP/IP with SSL on the Server

You can configure a listening endpoint to use TCP/IP with SSL on the server.

- Configure the listener in the `listener.ora` file. Oracle recommends using port number 2484 for typical Oracle Net clients.

See Also:

Oracle Database Net Services Reference for detailed information about configuring the `listener.ora` file

Step 2: Configure Secure Sockets Layer on the Client

When you configure SSL on the client, you configure the server DNs and use TCP/IP with SSL on the client.

Step 2A: Confirm Client Wallet Creation

You must confirm that a wallet has been created on the client and that the client has a valid certificate.

- Use Oracle Wallet Manager to check that the wallet has been created. See Step 1A: Confirm Wallet Creation on the Server for information about checking a wallet.

Note:

Oracle recommends that you use Oracle Wallet Manager to remove the trusted certificate in your Oracle wallet that is associated with each certificate authority that you do not use.

See Also:

- *Oracle Database Enterprise User Security Administrator's Guide* for general information about wallets
- *Oracle Database Enterprise User Security Administrator's Guide* for information about opening an existing wallet
- *Oracle Database Enterprise User Security Administrator's Guide* for information about creating a new wallet

Step 2B: Configure the Server DNs and Use TCP/IP with SSL on the Client

Next, you are ready to configure the server DNs and user TCP/IP with SSL on the client.

About Configuring the Server DNS and Using TCP/IP with SSL on the Client

You can configure the Oracle Net Service name to include the server DNs and to use TCP/IP with SSL on the client.

To accomplish this, you must specify the server's distinguished name (DN) and `TCPS` as the protocol in the client network configuration files to enable server DN matching and TCP/IP with SSL connections. Server DN matching prevents the database server from faking its identity to the client during connections by matching the server's global database name against the DN from the server certificate.

You must manually edit the client network configuration files, `tnsnames.ora` and `listener.ora`, to specify the server's DN and the TCP/IP with SSL protocol. The `tnsnames.ora` file can be located on the client or in the LDAP directory. If it is located on the server, then it typically resides in the same directory as the `listener.ora` file. Depending on the operating system, these files reside in the following directory locations:

- (UNIX) `$ORACLE_HOME/network/admin/`
- (Windows) `ORACLE_BASE\ORACLE_HOME\network\admin\`

Configuring the Server DNS and Using TCP/IP with SSL on the Client

You must edit the `tnsnames.ora` and `listener.ora` files to configure the server DNS and user TCP/IP with SSL on the client.

1. In the client `tnsnames.ora` file, add the `SSL_SERVER_CERT_DN` parameter and specify the database server's DN, as follows:

```
(SECURITY=
(SSL_SERVER_CERT_DN="cn=finance,cn=OracleContext,c=us,o=acme"))
```

The client uses this information to obtain the list of DNs it expects for each of the servers, enforcing the server's DN to match its service name. The following example shows an entry for the `Finance` database in the `tnsnames.ora` file.

```
finance=
(DESCRIPTION=
(ADDRESS_LIST=
(ADDRESS= (PROTOCOL = tcps) (HOST = finance_server) (PORT = 1575)))
(CONNECT_DATA=
(SERVICE_NAME= Finance.us.example.com))
(SECURITY=
(SSL_SERVER_CERT_DN="cn=finance,cn=OracleContext,c=us,o=acme"))
```

By default, the tnsnames.ora and listener.ora files are in the `$ORACLE_HOME/network/admin` directory on UNIX systems and in *`ORACLE_HOME`*`\network\admin` on Windows.

Alternatively, you can ensure that the common name (CN) portion of the server's DN matches the service name.

2. In the client `tnsnames.ora` file, enter `tcps` as the `PROTOCOL` in the `ADDRESS` parameter.

   This specifies that the client will use TCP/IP with SSL to connect to the database that is identified in the `SERVICE_NAME` parameter. The following also shows an entry that specifies TCP/IP with SSL as the connecting protocol in the `tnsnames.ora` file.

   ```
   LISTENER=
   (DESCRIPTION_LIST=
   (DESCRIPTION=
   (ADDRESS= (PROTOCOL = tcps) (HOST = finance_server) (PORT = 1575))))
   ```

3. In the `listener.ora` file, enter `tcps` as the `PROTOCOL` in the `ADDRESS` parameter.

Step 2C: Specify Required Client SSL Configuration (Wallet Location)

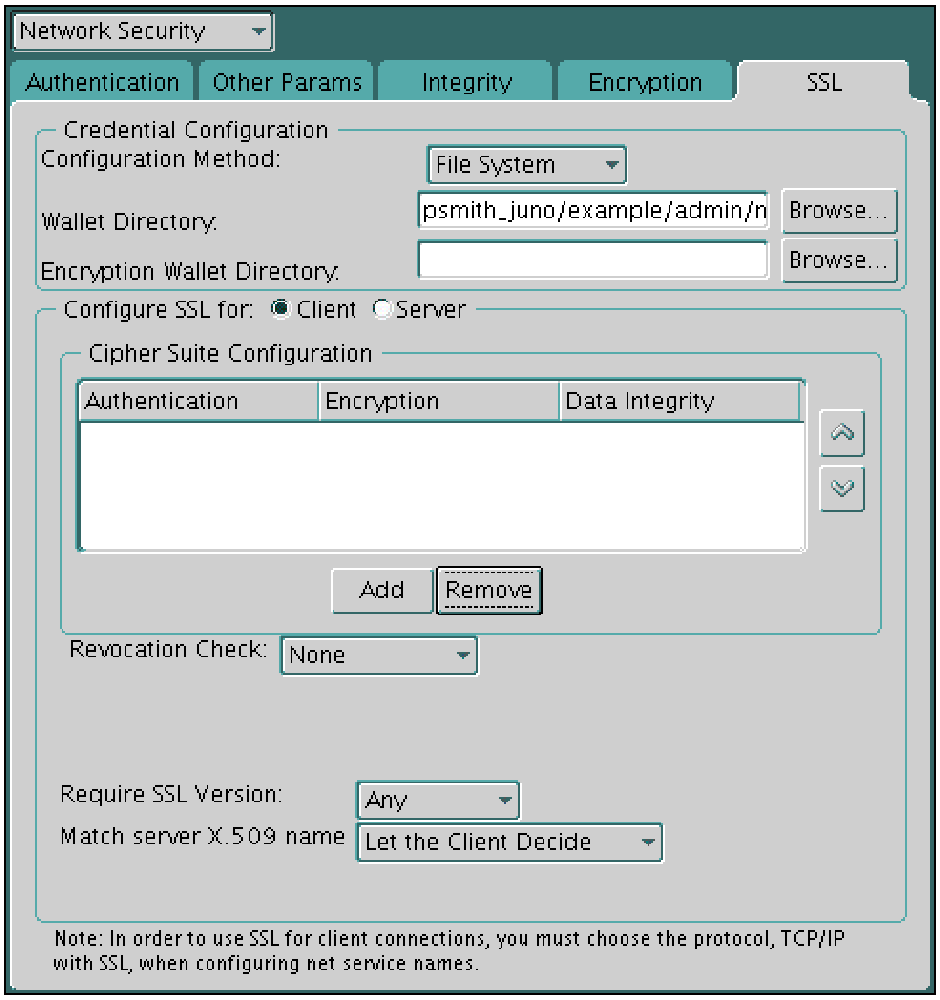
You can use Oracle Net Manager to specify the required client SSL configuration.

1. Start Oracle Net Manager.
   - (UNIX) From *`$ORACLE_HOME`*`/bin`, enter the following command at the command line:

     ```
     netmgr
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **SSL** tab.
5. Select **Configure SSL for: Client**.



Credential Configuration
Configuration Method: File System
Wallet Directory: psmith_juno/example/admin/n
Encryption Wallet Directory:
Configure SSL for: (●) Client ( ) Server
Cipher Suite Configuration

Authentication	Encryption	Data Integrity

Revocation Check: None
Require SSL Version: Any
Match server X.509 name Let the Client Decide

Note: In order to use SSL for client connections, you must choose the protocol, TCP/IP with SSL, when configuring net service names.

6. In the Wallet Directory box, enter the directory in which the Oracle wallet is located, or click **Browse** to find it by searching the file system.
7. From the Match server X.509 name list, select one of the following options:
   - **Yes:** Requires that the server's distinguished name (DN) match its service name. SSL ensures that the certificate is from the server and connections succeed only if there is a match.

     This check can be made only when RSA ciphers are selected, which is the default setting.
   - **No (default):** SSL checks for a match between the DN and the service name, but does not enforce it. Connections succeed regardless of the outcome but an error is logged if the match fails.
   - **Let Client Decide:** Enables the default.

     The following alert is displayed when you select **No**:

     ```
     Security Alert
     Not enforcing the server X.509 name match allows a server to potentially
     fake its identity. Oracle recommends selecting YES for this option so that
     connections are refused when there is a mismatch.
     ```
8. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file on the client is updated with the following entries:

```
SSL_CLIENT_AUTHENTICATION =TRUE
wallet_location =
 (SOURCE=
  (METHOD=File)
  (METHOD_DATA=
   (DIRECTORY=wallet_location)))

SSL_SERVER_DN_MATCH=(ON/OFF)
```

See Also:

For information about the server match parameters:

- Secure Sockets Layer X.509 Server Match Parameters

For information about using Oracle Net Manager to configure TCP/IP with SSL:

- *Oracle Database Net Services Administrator's Guide*
- *Oracle Database Net Services Reference*

Step 2D: Set the Client Secure Sockets Layer Cipher Suites (Optional)

Optionally, you can set the SSL cipher suites. Oracle Database provides default cipher suite settings.

About Setting the Client Secure Sockets Layer Cipher Suites

A cipher suite is a set of authentication, encryption, and data integrity algorithms used for exchanging messages between network entities.

During an SSL handshake, two entities negotiate to see which cipher suite they will use when transmitting messages back and forth.

When you install Oracle Database, the SSL cipher suites listed in Table 20-1 are set for you by default. This table lists them in the order they are tried when two entities are negotiating a connection. You can override the default by setting the `SSL_CIPHER_SUITES` parameter. For example, if you use Oracle Net Manager to add the cipher suite `SSL_RSA_WITH_RC4_128_SHA`, all other cipher suites in the default setting are ignored.

You can prioritize the cipher suites. When the client negotiates with servers regarding which cipher suite to use, it follows the prioritization you set. When you prioritize the cipher suites, consider the following:

- The level of security you want to use. For example, triple-DES encryption is stronger than DES.
- The impact on performance. For example, triple-DES encryption is slower than DES. Refer to Configuring Your System to Use Hardware Security Modules for information about using SSL hardware accelerators with Oracle Database.
- Administrative requirements. The cipher suites selected for a client must be compatible with those required by the server. For example, in the case of an Oracle Call Interface (OCI) user, the server requires the client to authenticate itself. You cannot, in this case, use a cipher suite employing Diffie-Hellman anonymous authentication, which disallows the exchange of certificates.

You typically prioritize cipher suites starting with the strongest and moving to the weakest.

Table 20-1 lists the currently supported Secure Sockets Layer cipher suites. These cipher suites are set by default when you install Oracle Database. The table also lists the authentication, encryption, and data integrity types each cipher suite uses.

Note:

If the `SSL_CLIENT_AUTHENTICATION` parameter is set to `true` in the `sqlnet.ora` file, then disable all cipher suites that use Diffie-Hellman anonymous authentication. Otherwise, the connection fails.

Setting the Client Secure Sockets Layer Cipher Suites

You can use Oracle Net Manager to set the client SSL cipher suites.

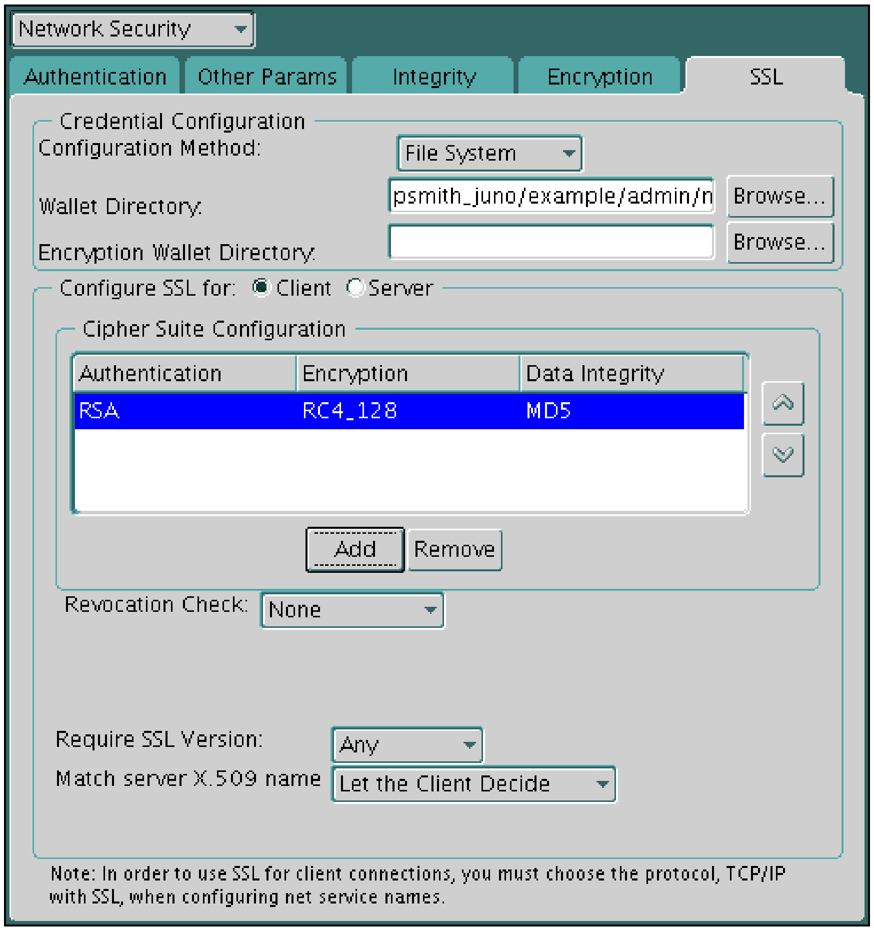
1. Start Oracle Net Manager.
   - (UNIX) From `$ORACLE_HOME/bin`, enter the following command at the command line:

     ```
     netmgr
     ```
   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **SSL** tab.
5. In the Cipher Suite Configuration region, click **Add**.

   A dialog box displays available cipher suites.
6. Select a suite and click **OK**.

   The **Cipher Suite Configuration** list is updated as shown as follows:



Credential Configuration

Configuration Method: File System

Wallet Directory: psmith_juno/example/admin/n

Encryption Wallet Directory:

Configure SSL for: [✓ Client] Server

Cipher Suite Configuration

Authentication	Encryption	Data Integrity
RSA	RC4_128	MD5

Revocation Check: None

Require SSL Version: Any

Match server X.509 name Let the Client Decide

Note: In order to use SSL for client connections, you must choose the protocol, TCP/IP with SSL, when configuring net service names.

7. Use the up and down arrows to prioritize the cipher suites.
8. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry:

   ```
   SSL_CIPHER_SUITES= (SSL_cipher_suite1 [,SSL_cipher_suite2])
   ```

Step 2E: Set the Required SSL Version on the Client (Optional)

The `SSL_VERSION` parameter defines the version of SSL that must run on the systems with which the client communicates.

You must set the `SSL_VERSION` parameter in the `sqlnet.ora` file. You can require these systems to use any valid version.

The default setting for this parameter in `sqlnet.ora` is `undetermined`, which is set by selecting **Any** from the list in the **SSL** tab of the Network Security window. When **Any** is selected, TLS 1.0 is tried first, then SSL 3.0, and SSL 2.0 are tried in that order. Ensure that the client SSL version is compatible with the version the server uses.

1. In the **Require SSL Version** list, select the SSL version that you want to configure.

   The default setting is **Any**.
2. From the **File** menu, select, **Save Network Configuration**.

The `sqlnet.ora` file is updated. If you selected **Any**, then it is updated with the following entry:

```
SSL_VERSION=UNDETERMINED
```

Step 2F: Set SSL as an Authentication Service on the Client (Optional)

The `SQLNET.AUTHENTICATION_SERVICES` parameter in the `sqlnet.ora` file sets the SSL authentication service.

About the SQLNET.AUTHENTICATION_SERVICES Parameter

The `SQLNET.AUTHENTICATION_SERVICES` parameter enables SSL authentication in conjunction with another authentication method supported by Oracle Database.

For example, use this parameter if you want the server to authenticate itself to the client by using SSL and the client to authenticate itself to the server by using RADIUS.

To set the `SQLNET.AUTHENTICATION_SERVICES` parameter, you must edit the `sqlnet.ora` file, which is located in the same directory as the other network configuration files.

Depending on the platform, the `sqlnet.ora` file is in the following directory location:

- (UNIX) *`$ORACLE_HOME`*`/network/admin`
- (Windows) *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\network\admin\`

Setting the SQLNET.AUTHENTICATION_SERVICES Parameter

You can set the `SQLNET.AUTHENTICATION_SERVICES` parameter in the `sqlnet.ora` file.

- To set the client `SQLNET.AUTHENTICATION_SERVICES` parameter, add TCP/IP with SSL (`TCPS`) to this parameter in the `sqlnet.ora` file by using a text editor.

  For example, if you want to use SSL authentication in conjunction with RADIUS authentication, then set this parameter as follows:

  ```
  SQLNET.AUTHENTICATION_SERVICES = (TCPS, radius)
  ```

If you do not want to use SSL authentication in conjunction with another authentication method, then do not set this parameter.

Step 2G: Specify the Certificate to Use for Authentication on the Client (Optional)

If you have multiple certificates, then you can set the `SQLNET.SSL_EXTENDED_KEY_USAGE` parameter in the `sqlnet.ora` file to specify the correct certificate.

About the SQLNET.SSL_EXTENDED_KEY_USAGE Parameter

The `SQLNET.SSL_EXTENDED_KEY_USAGE` parameter in the sqlnet.ora file specifies which certificate to use in authenticating to the database server

You should set the `SQLNET.SSL_EXTENDED_KEY_USAGE` parameter if you have multiple certificates in the security module, but there is only one certificate with extended key usage field of `client authentication`, and this certificate is exactly the one you want to use to authenticate to the database.

For example, use this parameter if you have multiple certificates in a smart card, only one of which has an extended key usage field of `client authentication`, and you want to use this certificate `C` to authenticate to the database. By setting this parameter on a Windows client to `client authentication`, the MSCAPI certificate selection box will not appear, and the certificate `C` is automatically used for the SSL authentication of the client to the server.

Setting the SQLNET.SSL_EXTENDED_KEY_USAGE Parameter

You can set the `SQLNET.SSL_EXTENDED_KEY_USAGE` to set the client authentication.

- To set the client `SQLNET.SSL_EXTENDED_KEY_USAGE` parameter, edit the `sqlnet.ora` file to have the following line:

  ```
  SQLNET.SSL_EXTENDED_KEY_USAGE = "client authentication"
  ```

If you do not want to use the certificate filtering, then remove the `SQLNET.SSL_EXTENDED_KEY_USAGE` parameter setting from the `sqlnet.ora` file.

Step 3: Log in to the Database Instance

After you have completed the configuration, you are ready to log in to the database.

- Start SQL*Plus and then enter one of the following connection commands:
  - If you are using SSL authentication for the client (`SSL_CLIENT_AUTHENTICATION=true` in the `sqlnet.ora` file):

    ```
    CONNECT/@net_service_name
    ```

  - If you are not using SSL authentication (`SSL_CLIENT_AUTHENTICATION=false` in the `sqlnet.ora` file):

    ```
    CONNECT username@net_service_name
    Enter password: password
    ```

Troubleshooting the Secure Sockets Layer Configuration

Common errors may occur while you use the Oracle Database SSL adapter.

It may be necessary to enable Oracle Net tracing to determine the cause of an error. For information about setting tracing parameters to enable Oracle Net tracing, refer to *Oracle Database Net Services Administrator's Guide*.

ORA-28759: Failure to Open File
Cause: The system could not open the specified file. Typically, this error occurs because the wallet cannot be found.

Action: Check the following:

- Ensure that the correct wallet location is specified in the `sqlnet.ora` file. This should be the same directory location where you saved the wallet.
- Enable Oracle Net tracing to determine the name of the file that cannot be opened and the reason.
- Ensure that auto-login was enabled when you saved the wallet. See *Oracle Database Enterprise User Security Administrator's Guide*.

ORA-28786: Decryption of Encrypted Private Key Failure
Cause: An incorrect password was used to decrypt an encrypted private key. Frequently, this happens because an auto-login wallet is not being used.

Action: Use Oracle Wallet Manager to turn the auto-login feature on for the wallet. Then save the wallet again. See *Oracle Database Enterprise User Security Administrator's Guide*
If the auto-login feature is not being used, then enter the correct password.

ORA-28858: SSL Protocol Error
Cause: This is a generic error that can occur during SSL handshake negotiation between two processes.

Action: Enable Oracle Net tracing and attempt the connection again to produce trace output. Then contact Oracle customer support with the trace output.

ORA-28859 SSL Negotiation Failure
Cause: An error occurred during the negotiation between two processes as part of the SSL protocol. This error can occur when two sides of the connection do not support a common cipher suite.

Action: Check the following:

- Use Oracle Net Manager to ensure that the SSL versions on both the client and the server match, or are compatible. For example, if the server accepts only SSL 3.0 and the client accepts only TLS 1.0, then the SSL connection will fail.
- Use Oracle Net Manager to check what cipher suites are configured on the client and the server, and ensure that compatible cipher suites are set on both.

  If the error still persists, then enable Oracle Net tracing and attempt the connection again. Contact Oracle customer support with the trace output.

See Also:

Step 2D: Set the Client Secure Sockets Layer Cipher Suites (Optional) for details about setting compatible cipher suites on the client and the server

Note:

If you do not configure any cipher suites, then all available cipher suites are enabled.

ORA-28862: SSL Connection Failed
Cause: This error occurred because the peer closed the connection.

Action: Check the following:

- Ensure that the correct wallet location is specified in the `sqlnet.ora` file so the system can find the wallet.

- Use Oracle Net Manager to ensure that cipher suites are set correctly in the `sqlnet.ora` file. Sometimes this error occurs because the `sqlnet.ora` has been manually edited and the cipher suite names are misspelled. Ensure that case sensitive string matching is used with cipher suite names.
- Use Oracle Net Manager to ensure that the SSL versions on both the client and the server match or are compatible. Sometimes this error occurs because the SSL version specified on the server and client do not match. For example, if the server accepts only SSL 3.0 and the client accepts only TLS 1.0, then the SSL connection will fail.
- For more diagnostic information, enable Oracle Net tracing on the peer.

ORA-28865: SSL Connection Closed
Cause: The SSL connection closed because of an error in the underlying transport layer, or because the peer process quit unexpectedly.

Action: Check the following:

- Use Oracle Net Manager to ensure that the SSL versions on both the client and the server match, or are compatible. Sometimes this error occurs because the SSL version specified on the server and client do not match. For example, if the server accepts only SSL 3.0 and the client accepts only TLS 1.0, then the SSL connection will fail.
- If you are using a Diffie-Hellman anonymous cipher suite and the `SSL_CLIENT_AUTHENTICATION` parameter is set to `true` in the server's `listener.ora` file, then the client does not pass its certificate to the server. When the server does not receive the client's certificate, it (the server) cannot authenticate the client so the connection is closed. To resolve this use another cipher suite, or set this `listener.ora` parameter to false.
- Enable Oracle Net tracing and check the trace output for network errors.
- For details, refer to Actions listed for ORA-28862: SSL Connection Failed

ORA-28868: Peer Certificate Chain Check Failed
Cause: When the peer presented the certificate chain, it was checked and that check failed. This failure can be caused by a number of problems, including:

- One of the certificates in the chain has expired.
- A certificate authority for one of the certificates in the chain is not recognized as a trust point.
- The signature in one of the certificates cannot be verified.

Action: See *Oracle Database Enterprise User Security Administrator's Guide* to use Oracle Wallet Manager to open your wallet and check the following:

- Ensure that all of the certificates installed in your wallet are current (not expired).
- Ensure that a certificate authority's certificate from your peer's certificate chain is added as a trusted certificate in your wallet. See *Oracle Database Enterprise User Security Administrator's Guide* to use Oracle Wallet Manager to import a trusted certificate.

ORA-28885: No certificate with the required key usage found.
Cause: Your certificate was not created with the appropriate X.509 version 3 key usage extension.

Action: Use Oracle Wallet Manager to check the certificate's key usage. See *Oracle Database Enterprise User Security Administrator's Guide* for information about key usage values.

ORA-29024: Certificate Validation Failure
Cause: The certificate sent by the other side could not be validated. This may occur if the certificate has expired, has been revoked, or is invalid for any other reason.

Action: Check the following:

- Check the certificate to determine whether it is valid. If necessary, get a new certificate, inform the sender that her certificate has failed, or resend.
- Check to ensure that the server's wallet has the appropriate trust points to validate the client's certificate. If it does not, then use Oracle Wallet Manager to import the appropriate trust point into the wallet. See *Oracle Database Enterprise User Security Administrator's Guide* for details about importing a trusted certificate.
- Ensure that the certificate has not been revoked and that certificate revocation list (CRL) checking is turned on. For details, refer to Configuring Certificate Validation with Certificate Revocation Lists

ORA-29223: Cannot Create Certificate Chain
Cause: A certificate chain cannot be created with the existing trust points for the certificate being installed. Typically, this error is returned when the peer does not give the complete chain and you do not have the appropriate trust points to complete it.

Action: Use Oracle Wallet Manager to install the trust points that are required to complete the chain. See *Oracle Database Enterprise User Security Administrator's Guide* for details about importing a trusted certificate.

Certificate Validation with Certificate Revocation Lists

Oracle provides tools that enable you to validate certificates using certificate revocation lists.

About Certificate Validation with Certificate Revocation Lists

The process of determining whether a given certificate can be used in a given context is referred to as certificate validation.

Certificate validation includes determining that the following takes place:

- A trusted certificate authority (CA) has digitally signed the certificate
- The certificate's digital signature corresponds to the independently-calculated hash value of the certificate itself and the certificate signer's (CA's) public key
- The certificate has not expired
- The certificate has not been revoked

The SSL network layer automatically performs the first three validation checks, but you must configure certificate revocation list (CRL) checking to ensure that certificates have not been revoked. CRLs are signed data structures that contain a list of revoked certificates. They are usually issued and signed by the same entity who issued the original certificate. (See certificate revocation list (CRL).)

What CRLs Should You Use?

You should have CRLs for all of the trust points that you honor.

The trust points are the trusted certificates from a third party identity that is qualified with a level of trust.

Typically, the certificate authorities you trust are called trust points.

How CRL Checking Works

Oracle Database checks the certificate revocation status against CRLs.

These CRLs are located in file system directories, Oracle Internet Directory, or downloaded from the location specified in the CRL Distribution Point (CRL DP) extension on the certificate.

Typically, CRL definitions are valid for a few days. If you store your CRLs on the local file system or in the directory, then you must update them regularly. If you use a CRL Distribution Point (CRL DP), then CRLs are downloaded each time a certificate is used, so there is no need to regularly refresh the CRLs.

The server searches for CRLs in the following locations in the order listed. When the system finds a CRL that matches the certificate CA's DN, it stops searching.

1. Local file system

   The system checks the `sqlnet.ora` file for the `SSL_CRL_FILE` parameter first, followed by the `SSL_CRL_PATH` parameter. If these two parameters are not specified, then the system checks the wallet location for any CRLs.

   Note: if you store CRLs on your local file system, then you must use the `orapki` utility to periodically update them. For more information, refer to Renaming CRLs with a Hash Value for Certificate Validation.

2. Oracle Internet Directory

   If the server cannot locate the CRL on the local file system and directory connection information has been configured in an `ldap.ora` file, then the server searches in the directory. It searches the CRL subtree by using the CA's distinguished name (DN) and the DN of the CRL subtree.

   The server must have a properly configured `ldap.ora` file to search for CRLs in the directory. It cannot use the Domain Name System (DNS) discovery feature of Oracle Internet Directory. Also note that if you store CRLs in the directory, then you must use the `orapki` utility to periodically update them. For details, refer to Uploading CRLs to Oracle Internet Directory

3. CRL DP

   If the CA specifies a location in the CRL DP X.509, version 3, certificate extension when the certificate is issued, then the appropriate CRL that contains revocation information for that certificate is downloaded. Currently, Oracle Database supports downloading CRLs over LDAP.

   Note the following:

   - For performance reasons, only user certificates are checked.

- Oracle recommends that you store CRLs in the directory rather than the local file system.

Configuring Certificate Validation with Certificate Revocation Lists

You can edit the `sqlnet.ora` file to configure certificate validation with certificate revocation lists.

About Configuring Certificate Validation with Certificate Revocation Lists

The `SSL_CERT_REVOCATION` parameter must be set to `REQUIRED` or `REQUESTED` in the `sqlnet.ora` file to enable certificate revocation status checking.

The `SSL_CERT_REVOCATION` parameter must be set to `REQUIRED` or `REQUESTED` in the `sqlnet.ora` file to enable certificate revocation status checking.

By default this parameter is set to `NONE` indicating that certificate revocation status checking is turned off.

Note:

If you want to store CRLs on your local file system or in Oracle Internet Directory, then you must use the command line utility, `orapki`, to rename CRLs in your file system or upload them to the directory.

Enabling Certificate Revocation Status Checking for the Client or Server

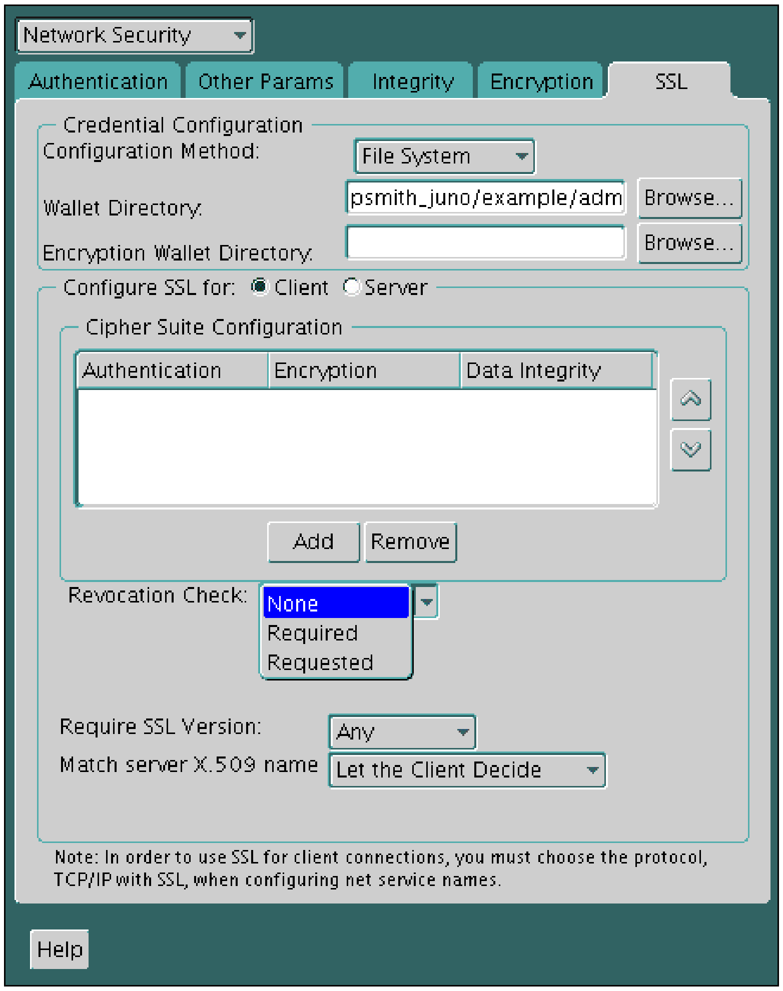
You can enable certificate the revocation status checking for a client or a server.

1. Start Oracle Net Manager.
   - (UNIX) From *`$ORACLE_HOME`*`/bin`, enter the following command at the command line:

     ```
     netmgr
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **SSL** tab.
5. Select one of the following options from the **Revocation Check** list:



Note: In order to use SSL for client connections, you must choose the protocol, TCP/IP with SSL, when configuring net service names.

- **Required:** Requires certificate revocation status checking. The SSL connection is rejected if a certificate is revoked or no CRL is found. SSL connections are accepted only if it can be verified that the certificate has not been revoked.
- **Requested:** Performs certificate revocation status checking if a CRL is available. The SSL connection is rejected if a certificate is revoked. SSL connections are accepted if no CRL is found or if the certificate has not been revoked.

  For performance reasons, only user certificates are checked for revocation.

6. (Optional) If CRLs are stored on your local file system, then set one or both of the following fields that specify where they are stored. These fields are available only when **Revocation Check** is set to **Required** or **REQUESTED**.
   - **Certificate Revocation Lists Path:** Enter the path to the directory where CRLs are stored or click **Browse** to find it by searching the file system. Specifying this path sets the `SSL_CRL_PATH` parameter in the `sqlnet.ora` file. If a path is not specified for this parameter, then the default is the wallet directory. Both DER-encoded (binary format) and PEM-encoded (BASE64) CRLs are supported.

- **Certificate Revocation Lists File:** Enter the path to a comprehensive CRL file (where PEM-encoded (BASE64) CRLs are concatenated in order of preference in one file) or click **Browse** to find it by searching the file system. Specifying this file sets the `SSL_CRL_FILE` parameter in the `sqlnet.ora` file. If this parameter is set, then the file must be present in the specified location, or else the application will error out during startup.

  If you want to store CRLs in a local file system directory by setting the **Certificate Revocation Lists Path**, then you must use the `orapki` utility to rename them so the system can locate them.

7. (Optional) If CRLs are fetched from Oracle Internet Directory, then directory server and port information must be specified in an `ldap.ora` file.

   When configuring your `ldap.ora` file, you should specify only a non-SSL port for the directory. CRL download is done as part of the SSL protocol, and making an SSL connection within an SSL connection is not supported.

   Oracle Database CRL functionality will not work if the Oracle Internet Directory non-SSL port is disabled.

8. Select **File**, **Save Network Configuration**. The `sqlnet.ora` file is updated.

Disabling Certificate Revocation Status Checking

You can disable certificate revocation status checking.

1. Start Oracle Net Manager.
   - (UNIX) From `$ORACLE_HOME/bin`, enter the following command at the command line:

     ```
     netmgr
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **SSL** tab.
5. Select **NONE** from the **Revocation Check** list.
6. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry:

   ```
   SSL_CERT_REVOCATION=NONE
   ```

Certificate Revocation List Management

Certificate revocation list management entails ensuring that the CRLs are the correct format before you enable certificate revocation checking.

About Certificate Revocation List Management

Oracle Database provides a command-line utility, `orapki`, that you can use to manage certificates.

Before you can enable certificate revocation status checking, you must ensure that the CRLs you receive from the CAs you use are in a form (renamed with a hash value) or in a location (uploaded to the directory) where your computer can use them.

You can also use LDAP command-line tools to manage CRLs in Oracle Internet Directory.

Note:

CRLs must be updated at regular intervals (before they expire) for successful validation. You can automate this task by using `orapki` commands in a script

Displaying orapki Help for Commands That Manage CRLs

You can display all the `orapki` commands that are available for managing CRLs.

- To display all the `orapki` available CRL management commands and their options, enter the following at the command line:

  ```
  orapki crl help
  ```

Note:

Using the `-summary`, `-complete`, or `-wallet` command options is always optional. A command will still run if these command options are not specified.

Renaming CRLs with a Hash Value for Certificate Validation

When the system validates a certificate, it must locate the CRL issued by the CA who created the certificate.

The system locates the appropriate CRL by matching the issuer name in the certificate with the issuer name in the CRL.

When you specify a CRL storage location for the **Certificate Revocation Lists Path** field in Oracle Net Manager, which sets the `SSL_CRL_PATH` parameter in the `sqlnet.ora` file, use the `orapki` utility to rename CRLs with a hash value that represents the issuer's name. Creating the hash value enables the server to load the CRLs.

On UNIX operating systems, `orapki` creates a symbolic link to the CRL. On Windows operating systems, it creates a copy of the CRL file. In either case, the symbolic link or the copy created by `orapki` are named with a hash value of the issuer's name. Then when the system validates a certificate, the same hash function is used to calculate the link (or copy) name so the appropriate CRL can be loaded.

- Depending on the operating system, enter one of the following commands to rename CRLs stored in the file system:
  - To rename CRLs stored in UNIX file systems:

    ```
    orapki crl hash -crl crl_filename [-wallet wallet_location] -symlink
    crl_directory [-summary]
    ```

– To rename CRLs stored in Windows file systems:

```
orapki crl hash -crl crl_filename [-wallet wallet_location] -copy
crl_directory [-summary]
```

In this specification, `crl_filename` is the name of the CRL file, `wallet_location` is the location of a wallet that contains the certificate of the CA that issued the CRL, and `crl_directory` is the directory where the CRL is located.

Using `-wallet` and `-summary` are optional. Specifying `-wallet` causes the tool to verify the validity of the CRL against the CA's certificate prior to renaming the CRL. Specifying the `-summary` option causes the tool to display the CRL issuer's name.

Uploading CRLs to Oracle Internet Directory

Publishing CRLs in the directory enables CRL validation throughout your enterprise, eliminating the need for individual applications to configure their own CRLs.

All applications can use the CRLs stored in the directory where they can be centrally managed, greatly reducing the administrative overhead of CRL management and use. The user who uploads CRLs to the directory by using `orapki` must be a member of the directory group `CRLAdmins` (`cn=CRLAdmins,cn=groups,%s_OracleContextDN%`). This is a privileged operation because these CRLs are accessible to the entire enterprise. Contact your directory administrator to get added to this administrative directory group.

- To upload CRLs to the directory, enter the following at the command line:

  ```
  orapki crl upload -crl crl_location -ldap hostname:ssl_port -user username [-
  wallet wallet_location] [-summary]
  ```

  In this specification, *`crl_location`* is the file name or URL where the CRL is located, *`hostname`* and *`ssl_port`* (SSL port with no authentication) are for the system on which your directory is installed, *`username`* is the directory user who has permission to add CRLs to the CRL subtree, and `wallet_location` is the location of a wallet that contains the certificate of the CA that issued the CRL.

Using `-wallet` and `-summary` are optional. Specifying `-wallet` causes the tool to verify the validity of the CRL against the CA's certificate prior to uploading it to the directory. Specifying the `-summary` option causes the tool to print the CRL issuer's name and the LDAP entry where the CRL is stored in the directory.

The following example illustrates uploading a CRL with the `orapki` utility:

```
orapki crl upload -crl /home/user1/wallet/crldir/crl.txt -ldap host1.example.com:
3533 -user cn=orcladmin
```

Note:

- The `orapki` utility will prompt you for the directory password when you perform this operation.
- Ensure that you specify the directory SSL port on which the Diffie-Hellman-based SSL server is running. This is the SSL port that does not perform authentication. Neither the server authentication nor the mutual authentication SSL ports are supported by the `orapki` utility.

Listing CRLs Stored in Oracle Internet Directory

You can display a list of all CRLs stored in the directory with `orapki`, which is useful for browsing to locate a particular CRL to view or download to your local computer.

This command displays the CA who issued the CRL (Issuer) and its location (DN) in the CRL subtree of your directory.

- To list CRLs in Oracle Internet Directory, enter the following at the command line:

  ```
  orapki crl list -ldap hostname:ssl_port
  ```

  where the *`hostname`* and *`ssl_port`* are for the system on which your directory is installed. Note that this is the directory SSL port with no authentication as described in the preceding section.

Viewing CRLs in Oracle Internet Directory

Oracle Internet Directory CRLS are available in a summarized format; you also can request a listing of revoked certificates for a CRL.

You can view CRLs stored in Oracle Internet Directory in a summarized format or you can request a complete listing of revoked certificates for a CRL.

A summary listing provides the CRL issuer's name and its validity period. A complete listing provides a list of all revoked certificates contained in the CRL.

- To view a summary listing of a CRL in Oracle Internet Directory, enter the following at the command line:

  ```
  orapki crl display -crl crl_location [-wallet wallet_location] -summary
  ```

  where *`crl_location`* is the location of the CRL in the directory. It is convenient to paste the CRL location from the list that displays when you use the `orapki crl list` command. See Listing CRLs Stored in Oracle Internet Directory.

To view a list of all revoked certificates contained in a specified CRL, which is stored in Oracle Internet Directory, you can enter the following at the command line:

```
orapki crl display -crl crl_location [-wallet wallet_location] -complete
```

For example, the following `orapki` command:

```
orapki crl display -crl $T_WORK/pki/wlt_crl/nzcrl.txt -wallet $T_WORK/pki/wlt_crl -
complete
```

produces the following output, which lists the CRL issuer's DN, its publication date, date of its next update, and the revoked certificates it contains:

```
issuer = CN=root,C=us, thisUpdate = Sun Nov 16 10:56:58 PST 2003, nextUpdate = Mon
Sep 30 11:56:58 PDT 2013, revokedCertificates = {(serialNo =
153328337133459399575438325845117876415, revocationDate - Sun Nov 16 10:56:58 PST
2003)}
CRL is valid
```

Using the `-wallet` option causes the `orapki crl display` command to validate the CRL against the CA's certificate.

Depending on the size of your CRL, choosing the `-complete` option may take a long time to display.

You can also use Oracle Directory Manager, a graphical user interface tool that is provided with Oracle Internet Directory, to view CRLs in the directory. CRLs are stored in the following directory location:

```
cn=CRLValidation,cn=Validation,cn=PKI,cn=Products,cn=OracleContext
```

Deleting CRLs from Oracle Internet Directory

The user who deletes CRLs from the directory by using `orapki` must be a member of the directory group `CRLAdmins`.

- To delete CRLs from the directory, enter the following at the command line:

  ```
  orapki crl delete -issuer issuer_name -ldap host:ssl_port -user username [-
  summary]
  ```

In this specification, *`issuer_name`* is the name of the CA who issued the CRL, the `hostname` and *`ssl_port`* are for the system on which your directory is installed, and `username` is the directory user who has permission to delete CRLs from the CRL subtree. Ensure that this must be a directory SSL port with no authentication.

Using the `-summary` option causes the tool to print the CRL LDAP entry that was deleted.

For example, the following `orapki` command:

```
orapki crl delete -issuer "CN=root,C=us" -ldap machine1:3500 -user cn=orcladmin -
summary
```

produces the following output, which lists the location of the deleted CRL in the directory:

```
Deleted CRL at cn=root
cd45860c.rN,cn=CRLValidation,cn=Validation,cn=PKI,cn=Products,cn=OracleContext
```

See Also:

Uploading CRLs to Oracle Internet Directory for information about the `CRLAdmins` directory administrative group and the SSL directory port

Troubleshooting CRL Certificate Validation

To determine whether certificates are being validated against CRLs, you can enable Oracle Net tracing.

When a revoked certificate is validated by using CRLs, then you will see the following entries in the Oracle Net tracing file without error messages logged between `entry` and `exit`:

```
nzcrlVCS_VerifyCRLSignature: entry
nzcrlVCS_VerifyCRLSignature: exit

nzcrlVCD_VerifyCRLDate: entry
nzcrlVCD_VerifyCRLDate: exit
```

```
nzcrlCCS_CheckCertStatus: entry
nzcrlCCS_CheckCertStatus: Certificate is listed in CRL
nzcrlCCS_CheckCertStatus: exit
```

Note:

Note that when certificate validation fails, the peer in the SSL handshake sees an `ORA-29024: Certificate Validation Failure`. If this message displays, refer to Oracle Net Tracing File Error Messages Associated with Certificate Validation for information about how to resolve the error.

See Also:

Oracle Database Net Services Administrator's Guide for information about setting tracing parameters to enable Oracle Net tracing

Oracle Net Tracing File Error Messages Associated with Certificate Validation

Oracle generates trace messages that are relevant to certificate validation.

These trace messages may be logged between the `entry` and `exit` entries in the Oracle Net tracing file. Oracle SSL looks for CRLs in multiple locations, so there may be multiple errors in the trace.

You can check the following list of possible error messages for information about how to resolve them.

CRL signature verification failed with RSA status
Cause: The CRL signature cannot be verified.

Action: Ensure that the downloaded CRL is issued by the peer's CA and that the CRL was not corrupted when it was downloaded. Note that the `orapki` utility verifies the CRL before renaming it with a hash value or before uploading it to the directory.
See Certificate Revocation List Management for information about using `orapki` for CRL management.

CRL date verification failed with RSA status
Cause: The current time is later than the time listed in the next update field. You should not see this error if CRL DP is used. The systems searches for the CRL in the following order:

1. File system
2. Oracle Internet Directory
3. CRL DP

The first CRL found in this search may not be the latest.

Action: Update the CRL with the most recent copy.

CRL could not be found
Cause: The CRL could not be found at the configured locations. This will return error ORA-29024 if the configuration specifies that certificate validation is require.

Action: Ensure that the CRL locations specified in the configuration are correct by performing the following steps:

1. Use Oracle Net Manager to check if the correct CRL location is configured. Refer to Configuring Certificate Validation with Certificate Revocation Lists
2. If necessary, use the `orapki` utility to configure CRLs for system use as follows:
   - For CRLs stored on your local file system, refer to Renaming CRLs with a Hash Value for Certificate Validation
   - CRLs stored in the directory, refer to Uploading CRLs to Oracle Internet Directory

Oracle Internet Directory host name or port number not set
Cause: Oracle Internet Directory connection information is not set. Note that this is not a fatal error. The search continues with CRL DP.

Action: If you want to store the CRLs in Oracle Internet Directory, then use Oracle Net Configuration Assistant to create and configure an `ldap.ora` file for your Oracle home.

Fetch CRL from CRL DP: No CRLs found
Cause: The CRL could not be fetched by using the CRL Distribution Point (CRL DP). This happens if the certificate does not have a location specified in its CRL DP extension, or if the URL specified in the CRL DP extension is incorrect.

Action: Ensure that your certificate authority publishes the CRL to the URL that is specified in the certificate's CRL DP extension.
Manually download the CRL. Then depending on whether you want to store it on your local file system or in Oracle Internet Directory, perform the following steps:
If you want to store the CRL on your local file system:

1. Use Oracle Net Manager to specify the path to the CRL directory or file. Refer to Configuring Certificate Validation with Certificate Revocation Lists
2. Use the `orapki` utility to configure the CRL for system use. Refer to Renaming CRLs with a Hash Value for Certificate Validation

If you want to store the CRL in Oracle Internet Directory:

1. Use Oracle Net Configuration Assistant to create and configure an `ldap.ora` file with directory connection information.
2. Use the `orapki` utility to upload the CRL to the directory. Refer to Uploading CRLs to Oracle Internet Directory

Configuring Your System to Use Hardware Security Modules

Oracle Database supports hardware security modules that use APIs that conform to the RSA Security, Inc., PKCS #11 specification.

Typically, these hardware devices are used to securely store and manage private keys in tokens or smart cards, or to accelerate cryptographic processing.

General Guidelines for Using Hardware Security Modules for SSL

Oracle provides a set of guidelines to follow if you are using a hardware security module with Oracle Database.

1. Contact your hardware device vendor to obtain the necessary hardware, software, and PKCS #11 libraries.
2. Install the hardware, software, and libraries where appropriate for the hardware security module you are using.
3. Test your hardware security module installation to ensure that it is operating correctly. Refer to your device documentation for instructions.
4. Create a wallet of the type `PKCS11` by using Oracle Wallet Manager and specify the absolute path to the PKCS #11 library (including the library name) if you wish to store the private key in the token. Oracle `PKCS11` wallets contain information that points to the token for private key access.

You can use the wallet containing PKCS #11 information just as you would use any Oracle wallet, except the private keys are stored on the hardware device and the cryptographic operations are performed on the device as well.

See Also:

Oracle Database Enterprise User Security Administrator's Guide for information about creating an Oracle wallet to store hardware security module credentials

Configuring Your System to Use nCipher Hardware Security Modules

You can configure your system to use nCipher hardware security modules for cryptographic processing.

About Configuring Your System to Use nCipher Hardware Security Modules

Hardware security modules made by nCipher Corporation are certified to operate with Oracle Database.

These modules provide a secure way to store keys and off-load cryptographic processing. Primarily, these devices provide the following benefits:

- Off-load cryptographic processing that frees your server to respond to other requests
- Secure private key storage on the device
- Allow key administration through the use of smart cards

> **Note:**
>
> You must contact your nCipher representative to obtain certified hardware and software to use with Oracle Database.

Oracle Components Required To Use an nCipher Hardware Security Module

To use an nCipher hardware security module, you must have a special set of components.

These components are as follows:

- nCipher Hardware Security Module
- Supporting nCipher PKCS #11 library

  The following platform-specific PKCS#11 library is required:

  – `libcknfast.so` library for UNIX 32-Bit
  – `libcknfast-64.so` library for UNIX 64-Bit
  – `cknfast.dll` library for Windows

> **Note:**
>
> You must contact your nCipher representative to have the hardware security module or the secure accelerator installed, and to acquire the necessary library.
>
> These tasks must be performed before you can use an nCipher hardware security module with Oracle Database.

Directory Path Requirements for Installing an nCipher Hardware Security Module

The nCipher hardware security module uses the nCipher PKCS #11 library.

To use the secure accelerator, you must provide the absolute path to the directory that contains the nCipher PKCS #11 library (including the library name) when you create the wallet by using Oracle Wallet Manager. This enables the library to be loaded at runtime.

Typically, the nCipher card is installed at the following locations:

- `/opt/nfast` for UNIX
- `C:\nfast` for Windows

The nCipher PKCS #11 library is located at the following location for typical installations:

- `/opt/nfast/toolkits/pkcs11/libcknfast.so` for UNIX 32-Bit
- `/opt/nfast/toolkits/pkcs11/libcknfast-64.so` for UNIX 64-Bit

- `C:\nfast\toolkits\pkcs11\cknfast.dll` for Windows

> **Note:**
>
> Use the 32-bit library version when using the 32-bit release of Oracle Database and use the 64-bit library version when using the 64-bit release of Oracle Database. For example, use the 64-bit nCipher PKCS #11 library for the Oracle Database for Solaris Operating System (SPARC 64-bit).

Configuring Your System to Use SafeNET Hardware Security Modules

You can configure your system to use SafeNET hardware security modules for cryptographic processing.

About Configuring Your System to Use SafeNET Hardware Security Modules

Hardware security modules made by SafeNET Incorporated are certified to operate with Oracle Database.

These modules provide a secure way to store keys and off-load cryptographic processing. Primarily, these devices provide the following benefits:

- Off-load of cryptographic processing to free your server to respond to more requests
- Secure private key storage on the device

> **Note:**
>
> You must contact your SafeNET representative to obtain certified hardware and software to use with Oracle Database.

Oracle Components Required for SafeNET Luna SA Hardware Security Modules

To use a SafeNET Luna SA hardware security module, you must have a special set of components.

These components are as follows:

- SafeNET Luna SA Hardware Security Module
- Supporting SafeNET Luna SA PKCS #11 library

  The following platform-specific PKCS#11 library is required:

  – `libCryptoki2.so` library for UNIX

  – `cryptoki.dll` library for Windows

Note:

You must contact your SafeNET representative to have the hardware security module or the secure accelerator installed, and to acquire the necessary library.

These tasks must be performed before you can use a SafeNET hardware security module with Oracle Database.

Directory Path Requirements for Installing a SafeNET Hardware Security Module

The SafeNET hardware security module uses the SafeNET PKCS #11 library.

To use the secure accelerator, you must provide the absolute path to the directory that contains the SafeNET PKCS #11 library (including the library name) when you create the wallet using Oracle Wallet Manager. This enables the library to be loaded at runtime.

Typically, the SafeNET Luna SA client is installed at the following location:

- `/usr/lunasa` for UNIX
- `C:\Program Files\LunaSA` for Windows

The SafeNET Luna SA PKCS #11 library is located at the following location for typical installations:

- `/usr/lunasa/lib/libCryptoki2.so` for UNIX
- `C:\Program Files\LunaSA\cryptoki2.dll` for Windows

Troubleshooting Using Hardware Security Modules

Oracle provides troubleshooting advice for hardware security modules.

Errors in the Oracle Net Trace Files

To detect whether the module is being used, you can turn on Oracle Net tracing.

If the wallet contains PKCS #11 information and the private key on the module is being used, then you will see the following entries in the Oracle Net tracing file without error messages logged between `entry` and `exit`:

```
nzpkcs11_Init: entry
nzpkcs11CP_ChangeProviders: entry
nzpkcs11CP_ChangeProviders: exit
nzpkcs11GPK_GetPrivateKey: entry
nzpkcs11GPK_GetPrivateKey: exit
nzpkcs11_Init: exit
...
nzpkcs11_Decrypt: entry
nzpkcs11_Decrypt: exit

nzpkcs11_Sign: entry
nzpkcs11_Sign: exit
```

See Also:

Oracle Database Net Services Administrator's Guide for information about setting tracing parameters to enable Oracle Net tracing

Error Messages Associated with Using Hardware Security Modules

Errors that are associated with using PKCS #11 hardware security modules can appear.

ORA-43000: PKCS11: library not found
Cause: The system cannot locate the PKCS #11 library at the location specified when the wallet was created. This happens only when the library is moved after the wallet is created.

Action: Copy the PKCS #11 library back to its original location where it was when the wallet was created.

ORA-43001: PKCS11: token not found
Cause: The smart card that was used to create the wallet is not present in the hardware security module slot.

Action: Ensure that the smart card that was used when the wallet was created is present in the hardware security module slot.

ORA-43002: PKCS11: passphrase is wrong
Cause: This can occur when an incorrect password is specified at wallet creation, or the PKCS #11 device password is changed after the wallet is created and not updated in the wallet by using Oracle Wallet Manager.

Action: Depending on the cause, take one of the following actions:
If you see this error during wallet creation, then check to ensure that you have the correct password and reenter it.
If the password changed after wallet creation, then use Oracle Wallet Manager to open the wallet and enter a new password.

See Also:

Oracle Database Enterprise User Security Administrator's Guide about creating an Oracle wallet to store hardware security credentials

Note:

The nCipher log file is in the directory where the module is installed at the following location:

```
/log/logfile
```

See Also:

nCipher and SafeNET documentation for more information about troubleshooting nCipher and SafeNET devices

21 Configuring RADIUS Authentication

RADIUS is a client/server security protocol widely used to enable remote authentication and access.

About Configuring RADIUS Authentication

An Oracle Database network can use any authentication method that supports the RADIUS standard.

The supported RADIUS standard includes token cards and smart cards when you install and configure the RADIUS protocol. Oracle Database uses RADIUS in a client/ server network environment. Moreover, when you use RADIUS, you can change the authentication method without modifying either the Oracle client or the Oracle database server.

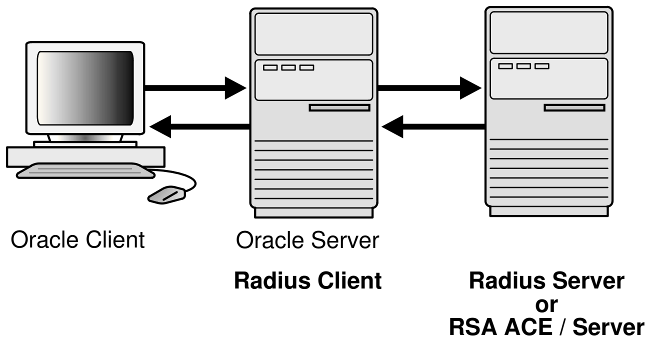
From an end user's perspective, the entire authentication process is transparent. When the user seeks access to an Oracle database server, the Oracle database server, acting as the RADIUS client, notifies the RADIUS server. The RADIUS server then:

- Looks up the user's security information
- Passes authentication and authorization information between the appropriate authentication server or servers and the Oracle database server
- Grants the user access to the Oracle database server
- Logs session information, including when, how often, and for how long the user was connected to the Oracle database server

> **Note:**
>
> Oracle Database does not support RADIUS authentication over database links.

Figure 21-1 illustrates the Oracle Database-RADIUS environment.

Figure 21-1 RADIUS in an Oracle Environment



The Oracle Database server acts as the RADIUS client, passing information between the Oracle client and the RADIUS server. Similarly, the RADIUS server passes information between the Oracle database server and the appropriate authentication servers.

A RADIUS server vendor is often the authentication server vendor as well. In this case authentication can be processed on the RADIUS server. For example, the RSA ACE/ Server is both a RADIUS server and an authentication server. It thus authenticates the user's pass code.

Note:

SecurID, an authentication product of RSA Security, Inc., though not directly supported by Oracle Database, has been certified as RADIUS-compliant. You can therefore, run SecurID under RADIUS.

Refer to the RSA Security SecurID documentation for further information.

See Also:

Oracle Database Net Services Reference for information about the `sqlnet.ora` file

RADIUS Components

RADIUS has a set of authentication components that enable you to manage configuration settings.

Table 21-1 lists the authentication components.

Table 21-1 RADIUS Authentication Components

Component	Stored Information
Oracle client	Configuration setting for communicating through RADIUS.

Table 21-1 (Cont.) RADIUS Authentication Components

Component	Stored Information
Oracle database server/ RADIUS client	Configuration settings for passing information between the Oracle client and the RADIUS server. The secret key file.
RADIUS server	Authentication and authorization information for all users. Each client's name or IP address. Each client's shared secret. Unlimited number of menu files enabling users already authenticated to select different login options without reconnecting.
Authentication server or servers	User authentication information such as pass codes and PINs, depending on the authentication method in use. **Note:** The RADIUS server can also be the authentication server.

RADIUS Authentication Modes

User authentication can take place either through synchronous authentication mode or challenge-response (asynchronous) authentication mode.

Synchronous Authentication Mode

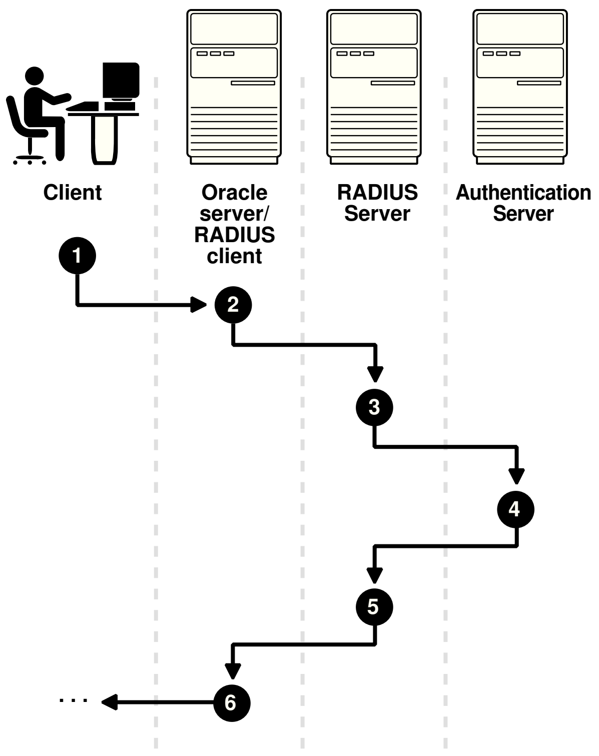
In the synchronous mode, RADIUS lets you use various authentication methods, including passwords and SecurID token cards.

Sequence for Synchronous Authentication Mode

The sequence of synchronous authentication mode is comprised of six steps.

Figure 21-2 shows the sequence in which synchronous authentication occurs.

Figure 21-2 Synchronous Authentication Sequence



The following steps describe the synchronous authentication sequence:

1. A user logs in by entering a connect string, pass code, or other value. The client system passes this data to the Oracle database server.
2. The Oracle database server, acting as the RADIUS client, passes the data from the Oracle client to the RADIUS server.
3. The RADIUS server passes the data to the appropriate authentication server, such as Smart Card or SecurID ACE for validation.
4. The authentication server sends either an Access Accept or an Access Reject message back to the RADIUS server.
5. The RADIUS server passes this response to the Oracle database server/RADIUS client.
6. The Oracle database server/RADIUS client passes the response back to the Oracle client.

Example: Synchronous Authentication with SecurID Token Cards

With SecurID authentication, each user has a token card that displays a dynamic number that changes every sixty seconds.

To gain access to the Oracle database server/RADIUS client, the user enters a valid pass code that includes both a personal identification number (PIN) and the dynamic number currently displayed on the user's SecurID card. The Oracle database server passes this authentication information from the Oracle client to the RADIUS server, which in this case is the authentication server for validation. Once the authentication server (RSA ACE/Server) validates the user, it sends an *accept* packet to the Oracle database server, which, in turn, passes it to the Oracle client. The user is now authenticated and able to access the appropriate tables and applications.

See Also:

Documentation provided by RSA Security, Inc.

Challenge-Response (Asynchronous) Authentication Mode

When the system uses the asynchronous mode, the user does not need to enter a user name and password at the SQL*Plus CONNECT string.

Sequence for Challenge-Response (Asynchronous) Authentication Mode

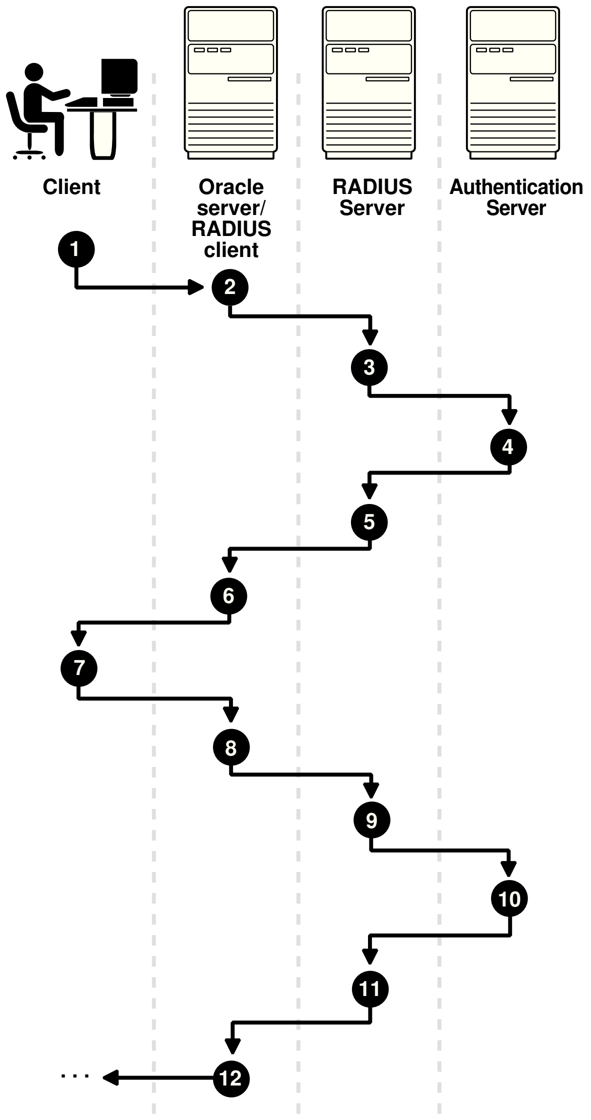
The sequence for challenge-response (asynchronous) authentication mode is comprised of 12 steps.

Figure 21-3 shows the sequence in which challenge-response (asynchronous) authentication occurs.

Note:

If the RADIUS server is the authentication server, Steps 3, 4, and 5, and Steps 9, 10, and 11 in Figure 21-3 are combined.

Figure 21-3 Asynchronous Authentication Sequence



The following steps describe the asynchronous authentication sequence:

1. A user initiates a connection to an Oracle database server. The client system passes the data to the Oracle database server.
2. The Oracle database server, acting as the RADIUS client, passes the data from the Oracle client to the RADIUS server.
3. The RADIUS server passes the data to the appropriate authentication server, such as a Smart Card, SecurID ACE, or token card server.
4. The authentication server sends a challenge, such as a random number, to the RADIUS server.
5. The RADIUS server passes the challenge to the Oracle database server/RADIUS client.
6. The Oracle database server/RADIUS client, in turn, passes it to the Oracle client. A graphical user interface presents the challenge to the user.

7. The user provides a response to the challenge. To formulate a response, the user can, for example, enter the received challenge into the token card. The token card provides a dynamic password that is entered into the graphical user interface. The Oracle client passes the user's response to the Oracle database server/RADIUS client.
8. The Oracle database server/RADIUS client sends the user's response to the RADIUS server.
9. The RADIUS server passes the user's response to the appropriate authentication server for validation.
10. The authentication server sends either an Access Accept or an Access Reject message back to the RADIUS server.
11. The RADIUS server passes the response to the Oracle database server/RADIUS client.
12. The Oracle database server/RADIUS client passes the response to the Oracle client.

Example: Asynchronous Authentication with Smart Cards

With smart card authentication, the user logs in by inserting the smart card into a smart card reader that reads the smart card.

The smart card is a plastic card, like a credit card, with an embedded integrated circuit for storing information.

The Oracle client sends the login information contained in the smart card to the authentication server by way of the Oracle database server/RADIUS client and the RADIUS server. The authentication server sends back a challenge to the Oracle client, by way of the RADIUS server and the Oracle database server, prompting the user for authentication information. The information could be, for example, a PIN as well as additional authentication information contained on the smart card.

The Oracle client sends the user's response to the authentication server by way of the Oracle database server and the RADIUS server. If the user has entered a valid number, the authentication server sends an *accept* packet back to the Oracle client by way of the RADIUS server and the Oracle database server. The user is now authenticated and authorized to access the appropriate tables and applications. If the user has entered incorrect information, the authentication server sends back a message rejecting user's access.

Example: Asynchronous Authentication with ActivCard Tokens

One particular ActivCard token is a hand-held device with a keypad and which displays a dynamic password.

When the user seeks access to an Oracle database server by entering a password, the information is passed to the appropriate authentication server by way of the Oracle database server/RADIUS client and the RADIUS server. The authentication server sends back a challenge to the client, by way of the RADIUS server and the Oracle database server. The user types that challenge into the token, and the token displays a number for the user to send in response.

The Oracle client then sends the user's response to the authentication server by way of the Oracle database server and the RADIUS server. If the user has typed a valid number, the authentication server sends an *accept* packet back to the Oracle client by way of the RADIUS server and the Oracle database server. The user is now authenticated and authorized to access the appropriate tables and applications. If the user has entered an incorrect response, the authentication server sends back a message rejecting the user's access.

Enabling RADIUS Authentication, Authorization, and Accounting

To enable RADIUS authentication, authorization, and accounting, you can use Oracle Net Manager.

Step 1: Configure RADIUS Authentication

To configure RADIUS authentication, you must first configure it on the Oracle client, then the server. Afterward, you can configure additional RADIUS features.

Note:

Unless otherwise indicated, perform these configuration tasks by using Oracle Net Manager or by using any text editor to modify the `sqlnet.ora` file. Be aware that in a multitenant environment, the settings in the `sqlnet.ora` file apply to all pluggable databases (PDBs).

Step 1A: Configure RADIUS on the Oracle Client

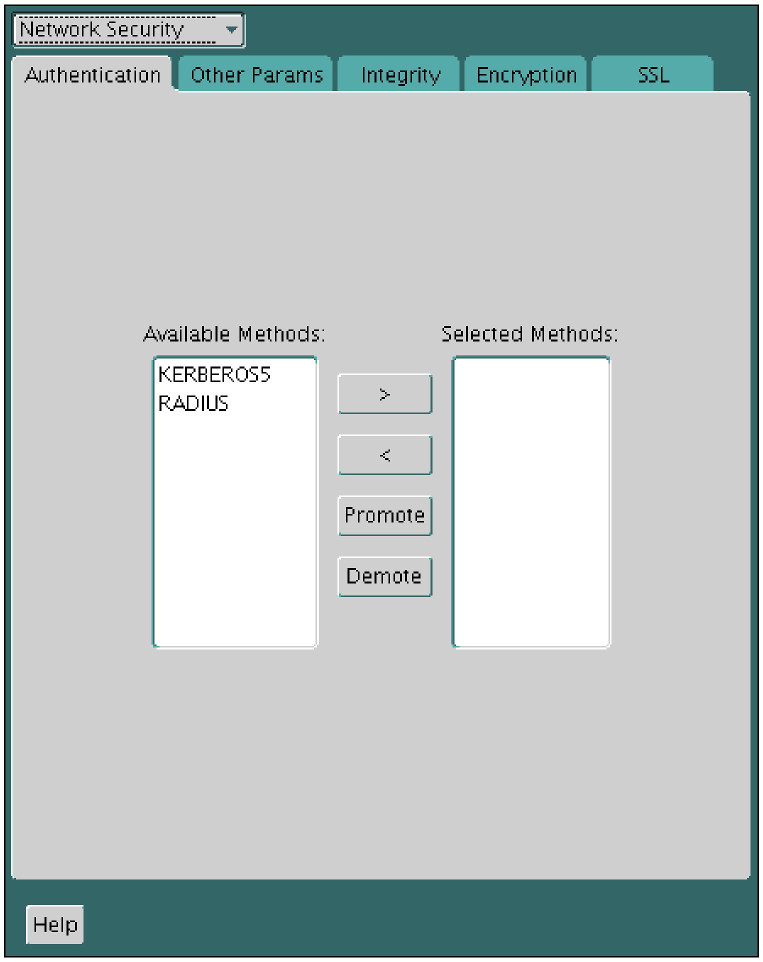
You can use Oracle Net Manager to configure RADIUS on the Oracle client.

1. Start Oracle Net Manager.
   - (UNIX) From *`$ORACLE_HOME`*`/bin`, enter the following command at the command line:

     ```
     netmgr
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **Authentication** tab. (It should be selected by default.)



5. From the Available Methods list, select **RADIUS**.
6. Select the right-arrow (**>**) to move RADIUS to the **Selected Methods** list.

   Move any other methods you want to use in the same way.
7. Arrange the selected methods in order of required usage by selecting a method in the Selected Methods list, and clicking **Promote** or **Demote** to position it in the list.

   For example, put RADIUS at the top of the list for it to be the first service used.
8. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry:

   ```
   SQLNET.AUTHENTICATION_SERVICES=(RADIUS)
   ```

Step 1B: Configure RADIUS on the Oracle Database Server

You must create a file to hold the RADIUS key and store this file on the Oracle database server. Then you must configure the appropriate parameters in the `sqlnet.ora` file.

Step 1B (1): Create the RADIUS Secret Key File on the Oracle Database Server

First, you must create the RADIUS secret key file.

1. Obtain the RADIUS secret key from the RADIUS server.

For each RADIUS client, the administrator of the RADIUS server creates a shared secret key, which must be less than or equal to 16 characters.

2. On the Oracle database server, create a directory:
   - (UNIX) *$ORACLE_HOME*`/network/security`
   - (Windows) *ORACLE_BASE*`\`*ORACLE_HOME*`\network\security`
3. Create the file `radius.key` to hold the shared secret copied from the RADIUS server. Place the file in the directory you created in Step 2.
4. Copy the shared secret key and paste it (and nothing else) into the `radius.key` file created on the Oracle database server.
5. For security purposes, change the file permission of `radius.key` to read only, accessible only by the Oracle owner.

   Oracle relies on the file system to keep this file secret.

   See Also:

   The RADIUS server administration documentation, for information about obtaining the secret key

Step 1B (2): Configure RADIUS Parameters on the Server (sqlnet.ora file)

After you create RADIUS secret key file, you are ready to configure the appropriate parameters in the `sqlnet.ora` file.

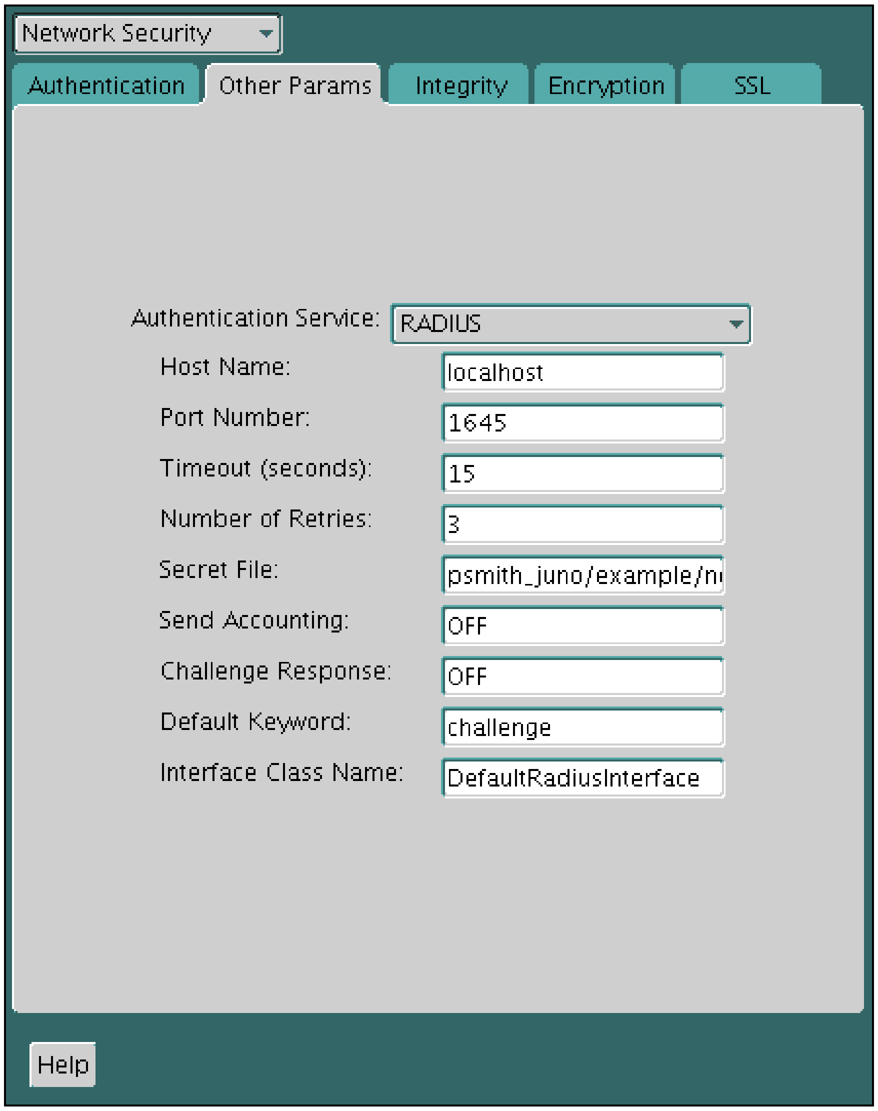
1. Start Oracle Net Manager.
   - (UNIX) From *$ORACLE_HOME*`/bin`, enter the following command at the command line:

     ```
     netmgr
     ```
   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **Authentication** tab.
5. From the Available Methods list, select **RADIUS**.
6. Move RADIUS to the **Selected Methods** list by choosing the right-arrow (**>**).
7. To arrange the selected methods in order of desired use, select a method in the Selected Methods list, and select **Promote** or **Demote** to position it in the list.

   For example, if you want RADIUS to be the first service used, then put it at the top of the list.
8. Select the **Other Params** tab.
9. From the **Authentication Service** list, select **RADIUS**.

10. In the **Host Name** field, accept the **localhost** as the default primary RADIUS server, or enter another host name.



Authentication Service:	RADIUS
Host Name:	localhost
Port Number:	1645
Timeout (seconds):	15
Number of Retries:	3
Secret File:	psmith_juno/example/n
Send Accounting:	OFF
Challenge Response:	OFF
Default Keyword:	challenge
Interface Class Name:	DefaultRadiusInterface

11. Ensure that the default value of the **Secret File** field is valid.
12. From the **File** menu, select **Save Network Configuration**.

The `sqlnet.ora` file is updated with the following entries:

```
SQLNET.AUTHENTICATION_SERVICES=RADIUS
SQLNET.RADIUS_AUTHENTICATION=RADIUS_server_{hostname|IP_address}
```

Note:

The `IP_address` can either be an Internet Protocol Version 4 (IPv4) or Internet Protocol Version 6 (IPv6) address. The RADIUS adapter supports both IPv4 and IPv6 based servers.

Step 1B (3): Set Oracle Database Server Initialization Parameters

After you configure the sqlnet.ora file, you must configure the `init.ora` initialization file.

1. Add the following setting to the `init.ora` file.

   ```
   OS_AUTHENT_PREFIX=""
   ```

   By default, the `init.ora` file is located in the *`ORACLE_HOME`*`/dbs` directory (or the same location of the data files) on Linux and UNIX systems, and in the *`ORACLE_HOME`*`\database` directory on Windows.

2. Restart the database.

   For example:

   ```
   SQL> SHUTDOWN
   SQL> STARTUP
   ```

See Also:

Oracle Database Reference for information about setting initialization parameters

Step 1C: Configure Additional RADIUS Features

You can change the default settings, configure the challenge-response mode, and set parameters for an alternate RADIUS server.

Step 1C(1): Change Default Settings

You can use Oracle Net Manager to change the default RADIUS settings.

1. Start Oracle Net Manager.
   - (UNIX) From *`$ORACLE_HOME`*`/bin`, enter the following command at the command line:

     ```
     netmgr
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Click the **Other Params** tab.
5. From the Authentication Service list, select **RADIUS**.
6. Change the default setting for any of the following fields:
   - **Port Number:** Specifies the listening port of the primary RADIUS server. The default value is 1645.

- **Timeout (seconds):** Specifies the time the Oracle database server waits for a response from the primary RADIUS server. The default is 15 seconds.
- **Number of Retries:** Specifies the number of times the Oracle database server resends messages to the primary RADIUS server. The default is three retries. For instructions on configuring RADIUS accounting, see Step 4: Configure RADIUS Accounting.
- **Secret File:** Specifies the location of the secret key on the Oracle database server. The field specifies the location of the secret key file, not the secret key itself. For information about specifying the secret key, see Step 1B (1): Create the RADIUS Secret Key File on the Oracle Database Server.

7. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entries:

   ```
   SQLNET.RADIUS_AUTHENTICATION_PORT=(PORT)
   SQLNET.RADIUS_AUTHENTICATION_TIMEOUT=(NUMBER OF SECONDS TO WAIT FOR response)
   SQLNET.RADIUS_AUTHENTICATION_RETRIES=(NUMBER OF TIMES TO RE-SEND TO RADIUS
   server)
   SQLNET.RADIUS_SECRET=(path/radius.key)
   ```

Step 1C(2): Configure Challenge-Response Mode

To configure challenge-response mode, you must specify information such as a dynamic password that you obtain from a token card.

With the RADIUS adapter, this interface is Java-based to provide optimal platform independence.

Note:

Third party vendors of authentication devices must customize this graphical user interface to fit their particular device. For example, a smart card vendor would customize the Java interface so that the Oracle client reads data, such as a dynamic password, from the smart card. When the smart card receives a challenge, it responds by prompting the user for more information, such as a PIN.

To configure challenge-response mode:

1. If you are using JDK 1.1.7 or JRE 1.1.7, then set the `JAVA_HOME` environment variable to the JRE or JDK location on the system where the Oracle client is run:

   - On UNIX, enter this command at the prompt:

     ```
     % setenv JAVA_HOME /usr/local/packages/jre1.1.7B
     ```

   - On Windows, select **Start**, **Settings**, **Control Panel**, **System**, **Environment**, and set the `JAVA_HOME` variable as follows:

     ```
     c:\java\jre1.1.7B
     ```

   This step is not required for any other JDK/JRE version.

2. Start Oracle Net Manager.

- (UNIX) From *$ORACLE_HOME*/bin, enter the following command at the command line:

  ```
  netmgr
  ```

- (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.

3. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
4. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
5. From the Authentication Service list, select **RADIUS**.
6. In the **Challenge Response** field, enter **ON** to enable challenge-response.
7. In the **Default Keyword** field, accept the default value of the challenge or enter a keyword for requesting a challenge from the RADIUS server.

   The keyword feature is provided by Oracle and supported by some, but not all, RADIUS servers. You can use this feature only if your RADIUS server supports it.

   By setting a keyword, you let the user avoid using a password to verify identity. If the user does *not* enter a password, the keyword you set here is passed to the RADIUS server which responds with a challenge requesting, for example, a driver's license number or birth date. If the user *does* enter a password, the RADIUS server may or may not respond with a challenge, depending upon the configuration of the RADIUS server.
8. In the **Interface Class Name** field, accept the default value of **DefaultRadiusInterface** or enter the name of the class you have created to handle the challenge-response conversation.

   If other than the default RADIUS interface is used, then you also must edit the `sqlnet.ora` file to enter `SQLNET.RADIUS_CLASSPATH=(location)`, where `location` is the complete path name of the jar file. It defaults to `$ORACLE_HOME/network/jlib/netradius.jar: $ORACLE_HOME/JRE/lib/vt.jar`
9. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entries:

   ```
   SQLNET.RADIUS_CHALLENGE_RESPONSE=([ON | OFF])
   SQLNET.RADIUS_CHALLENGE_KEYWORD=(KEYWORD)
   SQLNET.RADIUS_AUTHENTICATION_INTERFACE=(name of interface including the package
   name delimited by "/" for ".")
   ```

See Also:

Integrating Authentication Devices Using RADIUS for information about how to customize the challenge-response user interface

Step 1C(3): Set Parameters for an Alternate RADIUS Server

If you are using an alternate RADIUS server, then you must set additional parameters.

- Set the following parameters in the `sqlnet.ora` file:

```
SQLNET.RADIUS_ALTERNATE=(hostname or ip address of alternate radius server)
SQLNET.RADIUS_ALTERNATE_PORT=(1812)
SQLNET.RADIUS_ALTERNATE_TIMEOUT=(number of seconds to wait for response)
SQLNET.RADIUS_ALTERNATE_RETRIES=(number of times to re-send to radius server)
```

Step 2: Create a User and Grant Access

After you complete the RADIUS authentication, you must create an Oracle Database user who for the RADIUS configuration.

1. Start SQL*Plus and then execute these statements to create and grant access to a user identified externally on the Oracle database server.

   ```
   CONNECT system@database_name;
   Enter password: password
   CREATE USER username IDENTIFIED EXTERNALLY;
   GRANT CREATE SESSION TO USER username;
   EXIT
   ```

   If you are using Windows, you can use the Security Manager tool in Oracle Enterprise Manager.
2. Enter the same *username* in the RADIUS server's users file.

See Also:

Administration documentation for the RADIUS server

Step 3: Configure External RADIUS Authorization (Optional)

You must configure the Oracle server, the Oracle client, and the RADIUS server to RADIUS users who must connect to an Oracle database.

Step 3A: Configure the Oracle Server (RADIUS Client)

You can edit the `init.ora` file to configure an Oracle server for a RADIUS client.

To do so, you must modify the `init.ora` file, restart the database, and the set the RADIUS challenge-response mode.

1. Add the `OS_ROLES` parameter to the `init.ora` file and set this parameter to `TRUE` as follows:

   ```
   OS_ROLES=TRUE
   ```

   By default, the `init.ora` file is located in the *`ORACLE_HOME`*`/dbs` directory (or the same location of the data files) on Linux and UNIX systems, and in the *`ORACLE_HOME`*`\database` directory on Windows.
2. Restart the database so that the system can read the change to the `init.ora` file.

   For example:

   ```
   SQL> SHUTDOWN
   SQL> STARTUP
   ```

3. Set the RADIUS challenge-response mode to `ON` for the server if you have not already done so by following the steps listed in Step 1C(2): Configure Challenge-Response Mode.
4. Add externally identified users and roles.

Step 3B: Configure the Oracle Client Where Users Log In

Next, you must configure the Oracle client where users log in.

- Set the RADIUS challenge-response mode to `ON` for the client if you have not already done so by following the steps listed in Step 1C(2): Configure Challenge-Response Mode.

Step 3C: Configure the RADIUS Server

To configure the RADIUS server, you must modify the RADIUS server attribute configuration file.

1. Add the following attributes to the RADIUS server attribute configuration file:

ATTRIBUTE NAME	CODE	TYPE
`VENDOR_SPECIFIC`	26	Integer
`ORACLE_ROLE`	1	String

2. Assign a Vendor ID for Oracle in the RADIUS server attribute configuration file that includes the SMI Network Management Private Enterprise Code of `111`.

   For example, enter the following in the RADIUS server attribute configuration file:

```
VALUE      VENDOR_SPECIFIC      ORACLE      111
```

3. Using the following syntax, add the `ORACLE_ROLE` attribute to the user profile of the users who will use external RADIUS authorization:

```
ORA_databaseSID_rolename[_[A]|[D]]
```

   In this specification.:

   - `ORA` designates that this role is used for Oracle purposes
   - `databaseSID` is the Oracle system identifier that is configured in the database `init.ora` file.

     By default, the `init.ora` file is located in the *`ORACLE_HOME`*`/dbs` directory (or the same location of the data files) on Linux and UNIX systems, and in the *`ORACLE_HOME`*`\database` directory on Windows.
   - `rolename` is the name of role as it is defined in the data dictionary.
   - `A` is an optional character that indicates the user has administrator's privileges for this role.
   - `D` is an optional character that indicates this role is to be enabled by default.

   Ensure that RADIUS groups that map to Oracle roles adhere to the `ORACLE_ROLE` syntax.

   For example:

```
USERNAME      USERPASSWD="user_password",
              SERVICE_TYPE=login_user,
VENDOR_SPECIFIC=ORACLE,
ORACLE_ROLE=ORA_ora920_sysdba
```

See Also:

The RADIUS server administration documentation for information about configuring the server.

Step 4: Configure RADIUS Accounting

RADIUS accounting logs information about access to the Oracle database server and stores it in a file on the RADIUS accounting server.

Use this feature only if both the RADIUS server and authentication server support it.

Step 4A: Set RADIUS Accounting on the Oracle Database Server

To set RADIUS accounting on the server, you can use Oracle Net Manager.

1. Start Oracle Net Manager.
   - (UNIX) From `$ORACLE_HOME/bin`, enter the following command at the command line:

     ```
     netmgr
     ```
   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **Other Params** tab.
5. From the Authentication Service list, select **RADIUS**.
6. In the **Send Accounting** field, enter **ON** to enable accounting or **OFF** to disable accounting.
7. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry:

   ```
   SQLNET.RADIUS_SEND_ACCOUNTING= ON
   ```

Step 4B: Configure the RADIUS Accounting Server

RADIUS Accounting Server resides on the same host as the RADIUS authentication server or on a separate host.

- See the administration documentation for the RADIUS server, for information about configuring RADIUS accounting.

Step 5: Add the RADIUS Client Name to the RADIUS Server Database

The RADIUS server that you select must comply with RADIUS standards.

You can use any RADIUS server that complies with the Internet Engineering Task Force (IETF) RFC #2138, *Remote Authentication Dial In User Service (RADIUS)*, and RFC #2139 *RADIUS Accounting* standards. Because RADIUS servers vary, consult the documentation for your particular RADIUS server for any unique interoperability requirements.

To add the RADIUS client name to a Livingston RADIUS server:

1. Open the clients file, which is located in `/etc/raddb/clients`.

   The following text and table appear:

   ```
   @ (#) clients 1.1 2/21/96 Copyright 1991 Livingston Enterprises Inc
   This file contains a list of clients which are allowed to make authentication
   requests and their encryption key. The first field is a valid hostname. The
   second field (separated by blanks or tabs) is the encryption key.
   Client Name                      Key
   ```

2. In the `CLIENT NAME` column, enter the host name or IP address of the host on which the Oracle database server is running.

   In the `KEY` column, type the shared secret.

   The value you enter in the `CLIENT NAME` column, whether it is the client's name or IP address, depends on the RADIUS server.

3. Save and close the clients file.

See Also:

Administration documentation for the RADIUS server

Step 6: Configure the Authentication Server for Use with RADIUS

After you add the RADIUS client name to the RADIUS server database, you can configure the authentication server to use the RADIUS.

- Refer to the authentication server documentation for instructions about configuring the authentication servers.

Step 7: Configure the RADIUS Server for Use with the Authentication Server

After you configure the authentication server for use with RADIUS, you can configure the RADIUS server to use the authentication server.

- Refer to the RADIUS server documentation for instructions about configuring the RADIUS server for use with the authentication server.

Step 8: Configure Mapping Roles

If the RADIUS server supports vendor type attributes, then you can manage roles by storing them in the RADIUS server.

The Oracle database server downloads the roles when there is a `CONNECT` request using RADIUS.To use this feature, you must configure roles on both the Oracle database server and the RADIUS server.

1. Use a text editor to set the `OS_ROLES` parameter in the initialization parameters file on the Oracle database server.

   By default, the `init.ora` file is located in the *`ORACLE_HOME`*`/dbs` directory (or the same location of the data files) on Linux and UNIX systems, and in the *`ORACLE_HOME`* `\database` directory on Windows.

2. Stop and restart the Oracle database server.

   For example:

   ```
   SHUTDOWN
   STARTUP
   ```

3. Create each role that the RADIUS server will manage on the Oracle database server with the value `IDENTIFIED EXTERNALLY`.

   To configure roles on the RADIUS server, use the following syntax:

   ```
   ORA_DatabaseName.DatabaseDomainName_RoleName
   ```

   In this specification:

   - `DatabaseName` is the name of the Oracle database server for which the role is being created. This is the same as the value of the DB_NAME initialization parameter.
   - `DatabaseDomainName` is the name of the domain to which the Oracle database server belongs. The value is the same as the value of the `DB_DOMAIN` initialization parameter.
   - `RoleName` is name of the role created in the Oracle database server.

   For example:

   ```
   ORA_USERDB.US.EXAMPLE.COM_MANAGER
   ```

4. Configure RADIUS challenge-response mode.

Using RADIUS to Log in to a Database

You can use RADIUS to log into a database by using either synchronous authentication mode or challenge-response mode.

- Start SQL*Plus and use one of the following ways to log in to the database:
  – If you are using the synchronous authentication mode, first ensure that challenge-response mode is not turned to `ON`, and then enter the following command:

    ```
    CONNECT username@database_alias
    Enter password: password
    ```

  – If you are using the challenge-response mode, ensure that challenge-response mode is set to `ON` and then enter the following command:

    ```
    CONNECT /@database_alias
    ```

> **Note:**
>
> The challenge-response mode can be configured for all login cases.

RSA ACE/Server Configuration Checklist

If you are using an RSA ACE/Server RADIUS server, check the host agent and SecurID tokens for this server before making the initial connection.

- Ensure that the host agent in the RSA ACE/Server is set up to send a node secret. In version 5.0, this is done by leaving the SENT Node secret box unchecked. If the RSA ACE/Server fails to send a node secret to the agent, then a node verification failure message will be written to the RSA ACE/Server log.
- If you are using RSA SecurID tokens, then ensure that the token is synchronized with the RSA ACE/Server.

> **See Also:**
>
> RSA ACE/Server documentation for specific information about troubleshooting.

22

Customizing the Use of Strong Authentication

You can configure multiple authentication methods under Oracle Database network encryption and strong authentication.

Connecting to a Database Using Strong Authentication

You can use password authentication to connect to a database that is configured to use strong authentication.

1. To connect to an Oracle database server using a user name and password when an Oracle network and strong authentication method has been configured, disable the external authentication.

   You must first follow the instructions in Disabling Strong Authentication and Network Encryption to disable the external authentication before you can connect to an Oracle Database server using a user name and password when an Oracle network and strong authentication method has been configured.

2. With the external authentication disabled, connect to the database using the following format:

   ```
   % sqlplus username@net_service_name
   Enter password: password
   ```

   For example:

   ```
   % sqlplus hr@emp
   Enter password: password
   ```

> **Note:**
>
> You can configure multiple authentication methods, including both externally authenticated users and password authenticated users, on a single database.

Disabling Strong Authentication and Network Encryption

You can use Oracle Net Manager to disable strong authentication and network encryption.

1. Start Oracle Net Manager.

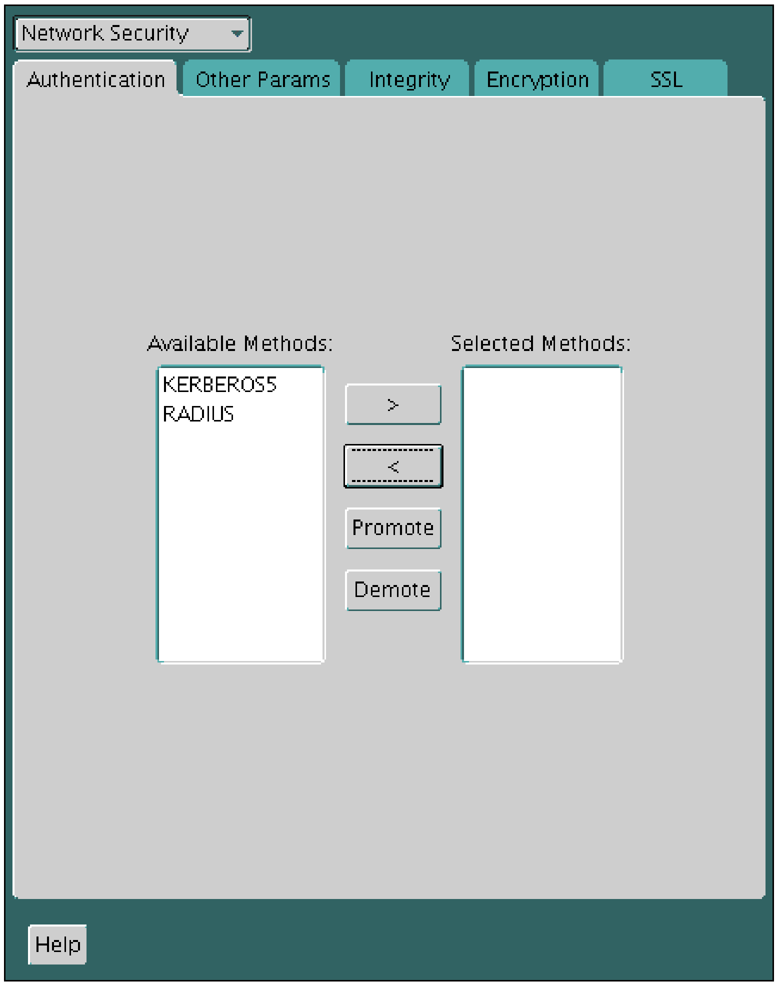
   - (UNIX) From *$ORACLE_HOME*/bin, enter the following command at the command line:

     ```
     netmgr
     ```

- (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.

2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **Authentication** tab (which is selected by default).
5. Sequentially move all authentication methods from the Selected Method list to the Available Methods list by selecting a method and choosing the left arrow [<].



6. Select the **Encryption** tab.
7. Do the following:
   - From the **Encryption** menu, select **SERVER**.
   - Set **Encryption Type** to **rejected**.
   - In the **Encryption Seed** field, enter a valid encryption seed if an encryption seed was used.
   - Under **Select Methods**, move any methods to the **Available Methods** field.
8. Repeat these steps disable network encryption for the client, by selecting **CLIENT** from the **Encryption** menu.

9. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entries to indicate that strong authentication and network encryption are disabled:

   Strong authentication:

   ```
   SQLNET.AUTHENTICATION_SERVICES = (NONE)
   ```

   For network encryption, you can set it individually, for the server side and for the client side. The following examples show network encryption being disabled for both the server and the client:

   ```
   SQLNET.ENCRYPTION_SERVER = REJECTED
   SQLNET.ENCRYPTION_CLIENT = REJECTED
   ```

   Be aware that in a multitenant environment, the settings in the `sqlnet.ora` file apply to all pluggable databases (PDBs).

Configuring Multiple Authentication Methods

Many networks use more than one authentication method on a single security server.

Accordingly, Oracle Database lets you configure your network so that Oracle clients can use a specific authentication method, and Oracle database servers can accept any method specified.

You can set up multiple authentication methods on both client and server systems either by using Oracle Net Manager, or by using any text editor to modify the `sqlnet.ora` file. Use Oracle Net Manager to add authentication methods to both clients and servers.

1. Start Oracle Net Manager.
   - (UNIX) From `$ORACLE_HOME/bin`, enter the following command at the command line:

     ```
     netmgr
     ```

   - (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, then **Net Manager**.
2. Expand **Oracle Net Configuration**, and from **Local**, select **Profile**.
3. From the **Naming** list, select **Network Security**.

   The Network Security tabbed window appears.
4. Select the **Authentication** tab.
5. Select a method listed in the Available Methods list.
6. Sequentially move selected methods to the Selected Methods list by clicking the right arrow (>).
7. Arrange the selected methods in order of desired use.

   To do this, select a method in the Selected Methods list, and select **Promote** or **Demote** to position it in the list.
8. From the **File** menu, select **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry, listing the selected authentication methods:

```
SQLNET.AUTHENTICATION_SERVICES = (KERBEROS5, RADIUS)
```

Note:

SecurID functionality is available through RADIUS; RADIUS support is built into the RSA ACE/Server.

Configuring Oracle Database for External Authentication

You can use parameters to configure Oracle Database for network authentication.

Setting the SQLNET.AUTHENTICATION_SERVICES Parameter in sqlnet.ora

The `SQLNET.AUTHENTICATION_SERVICES` parameter defines the authentication method and version to be used.

You must set the `SQLNET.AUTHENTICATION_SERVICES` parameter in the `sqlnet.ora` file for all clients and servers to enable each to use a supported authentication method.

- Set the `SQLNET.AUTHENTICATION_SERVICES` parameter using the following syntax:

  ```
  SQLNET.AUTHENTICATION_SERVICES=(oracle_authentication_method)
  ```

For example, for all clients and servers using Kerberos authentication:

```
SQLNET.AUTHENTICATION_SERVICES=(KERBEROS5)
```

By default, the `sqlnet.ora` file is located in the *`ORACLE_HOME`*`/network/admin` directory or in the location set by the `TNS_ADMIN` environment variable. Ensure that you have properly set the `TNS_ADMIN` variable to point to the correct `sqlnet.ora` file.

See Also:

*SQL*Plus User's Guide and Reference* for more information and examples of setting the `TNS_ADMIN` variable

Setting OS_AUTHENT_PREFIX to a Null Value

The `OS_AUTHENT_PREFIX` parameter specifies a prefix that Oracle Database uses to authenticate users who attempt to connect to the server.

Authentication service-based user names can be long, and Oracle user names are limited to 30 characters. Oracle strongly recommends that you set the `OS_AUTHENT_PREFIX` parameter to a null value.

- In the initialization file for the database instance, set `OS_AUTHENT_PREFIX` as follows:

  ```
  OS_AUTHENT_PREFIX=""
  ```

Note:

- The default value for `OS_AUTHENT_PREFIX` is `OPS$`; however, you can set it to any string.
- If a database already has the `OS_AUTHENT_PREFIX` set to a value other than `NULL` (" "), then *do not change it,* because it can inhibit previously created, externally identified users from connecting to the Oracle server.

After you have set `OS_AUTHENT_PREFIX` to null, then you can create external users by using the following syntax:

```
CREATE USER os_authent_prefix_username IDENTIFIED EXTERNALLY;
```

For example, to create the user `king`:

```
CREATE USER king IDENTIFIED EXTERNALLY;
```

The advantage of creating a user in this way is that you no longer need to maintain different user names for externally identified users. This is true for all supported authentication methods.

Part VI
Monitoring Database Activity with Auditing

Part VI describes how to monitor database activity with auditing.

23

Introduction to Auditing

Auditing tracks changes that users make in the database.

Note:

Except where noted, this part describes how to use pure unified auditing, in which all audit records are centralized in one place. If you have not yet migrated to use unified auditing, then see *Oracle Database Upgrade Guide*. Be aware that the upgrade process itself does not automatically enable unified auditing. You must manually migrate to unified auditing, as described in *Oracle Database Upgrade Guide*.

What Is Auditing?

Auditing is the monitoring and recording of configured database actions, from both database users and nondatabase users.

"Nondatabase users" refers to application users who are recognized in the database using the `CLIENT_IDENTIFIER` attribute. To audit this type of user, you can use a unified audit policy condition, a fine-grained audit policy, or Oracle Database Real Application Security.

You can base auditing on individual actions, such as the type of SQL statement executed, or on combinations of data that can include the user name, application, time, and so on.

You can configure auditing for both successful and failed activities, and include or exclude specific users from the audit. In a multitenant environment, you can audit individual actions of the pluggable database (PDB) or individual actions in the entire multitenant container database (CDB). In addition to auditing the standard activities the database provides, auditing can include activities from Oracle Database Real Application Security, Oracle Recovery Manager, Oracle Data Pump, Oracle Data Mining, Oracle Database Vault, Oracle Label Security, and Oracle SQL*Loader direct path events.

Auditing is enabled by default. All audit records are written to the unified audit trail in a uniform format and are made available through the `UNIFIED_AUDIT_TRAIL` view. These records reside in the `AUDSYS` schema. The audit records are stored in the `SYSAUX` tablespace by default. Oracle recommends that you configure a different tablespace for the unified audit trail. Be aware that for most Oracle Database editions except for Enterprise Edition, you can only associate the tablespace for unified auditing once. You should perform this association before you generate any audit records for the unified audit trail. After you have associated the tablespace, you cannot modify it because partitioning is only supported on Enterprise Edition.

You can configure auditing by using any of the following methods:

- **Group audit settings into one unified audit policy.** You can create one or more unified audit policies that define all the audit settings that your database needs. Auditing Activities with Unified Audit Policies and the AUDIT Statement describes how to accomplish this.
- **Use one of the default unified audit policies.** Oracle Database provides three default unified audit policies that encompass the standard audit settings that most regulatory agencies require. See Auditing Activities with the Predefined Unified Audit Policies.
- **Create fine-grained audit policies.** You can create fine-grained audit policies that capture data such as the time an action occurred. See Auditing Specific Activities with Fine-Grained Auditing.

Oracle recommends that you audit your databases. Auditing is an effective method of enforcing strong internal controls so that your site can meet its regulatory compliance requirements, as defined in the Sarbanes-Oxley Act. This enables you to monitor business operations, and find any activities that may deviate from company policy. Doing so translates into tightly controlled access to your database and the application software, ensuring that patches are applied on schedule and preventing ad hoc changes. By creating effective audit policies, you can generate an audit record for audit and compliance personnel. Be selective with auditing and ensure that it meets your business compliance needs.

Why Is Auditing Used?

You typically use auditing to monitor user activity.

Auditing can be used to accomplish the following:

- **Enable accountability for actions.** These include actions taken in a particular schema, table, or row, or affecting specific content.
- **Deter users (or others, such as intruders) from inappropriate actions based on their accountability.**
- **Investigate suspicious activity.** For example, if a user is deleting data from tables, then a security administrator can audit all connections to the database and all successful and unsuccessful deletions of rows from all tables in the database.
- **Notify an auditor of the actions of an unauthorized user.** For example, an unauthorized user could be changing or deleting data, or the user has more privileges than expected, which can lead to reassessing user authorizations.
- **Monitor and gather data about specific database activities.** For example, the database administrator can gather statistics about which tables are being updated, how many logical I/Os are performed, or how many concurrent users connect at peak times.
- **Detect problems with an authorization or access control implementation.** For example, you can create audit policies that you expect will never generate an audit record because the data is protected in other ways. However, if these policies generate audit records, then you will know the other security controls are not properly implemented.
- **Address auditing requirements for compliance.** Regulations such as the following have common auditing-related requirements:
  - Sarbanes-Oxley Act

  - Health Insurance Portability and Accountability Act (HIPAA)
  - International Convergence of Capital Measurement and Capital Standards: a Revised Framework (Basel II)
  - Japan Privacy Law
  - European Union Directive on Privacy and Electronic Communications

Best Practices for Auditing

You should follow best practices guidelines for auditing.

- **As a general rule, design your auditing strategy to collect the amount of information that you need to meet compliance requirements, but focus on activities that cause the greatest security concerns.** For example, auditing every table in the database is not practical, but auditing tables with columns that contain sensitive data, such as salaries, is. With both unified and fine-grained auditing, there are mechanisms you can use to design audit policies that focus on specific activities to audit.
- **Periodically archive and purge the audit trail data.** See Purging Audit Trail Records for more information.

What Is Unified Auditing?

In unified auditing, the unified audit trail captures audit information from a variety of sources.

Unified auditing enables you to capture audit records from the following sources:

- Audit records (including `SYS` audit records) from unified audit policies and `AUDIT` settings
- Fine-grained audit records from the `DBMS_FGA` PL/SQL package
- Oracle Database Real Application Security audit records
- Oracle Recovery Manager audit records
- Oracle Database Vault audit records
- Oracle Label Security audit records
- Oracle Data Mining records
- Oracle Data Pump
- Oracle SQL*Loader Direct Load

The unified audit trail, which resides in a read-only table in the `AUDSYS` schema in the `SYSAUX` tablespace, makes this information available in a uniform format in the `UNIFIED_AUDIT_TRAIL` data dictionary view, and is available in both single-instance and Oracle Database Real Application Clusters environments. In addition to the user `SYS`, users who have been granted the `AUDIT_ADMIN` and `AUDIT_VIEWER` roles can query these views. If your users only need to query the views but not create audit policies, then grant them the `AUDIT_VIEWER` role.

When the database is writeable, audit records are written to the unified audit trail. If the database is not writable, then audit records are written to new format operating system files in the `$ORACLE_BASE/audit/$ORACLE_SID` directory.

See Also:

Oracle Database Reference for detailed information about the `UNIFIED_AUDIT_TRAIL` data dictionary view

Benefits of the Unified Audit Trail

The benefits of a unified audit trail are many.

For example:

- After unified auditing is enabled, it does not depend on the initialization parameters that were used in previous releases. See Table G-1 for a list of these initialization parameters.
- The audit records, including records from the `SYS` audit trail, for all the audited components of your Oracle Database installation are placed in one location and in one format, rather than your having to look in different places to find audit trails in varying formats. This consolidated view enables auditors to co-relate audit information from different components. For example, if an error occurred during an `INSERT` statement, standard auditing can indicate the error number and the SQL that was executed. Oracle Database Vault-specific information can indicate whether this error happened because of a command rule violation or realm violation. Note that there will be two audit records with a distinct `AUDIT_TYPE`. With this unification in place, `SYS` audit records appear with `AUDIT_TYPE` set to `Standard Audit`.
- The management and security of the audit trail is also improved by having it in single audit trail.
- Overall auditing performance is greatly improved. By default, the audit records are automatically written to an internal relational table in the `AUDSYS` schema.
- You can create named audit policies that enable you to audit the supported components listed at the beginning of this section, as well as `SYS` administrative users. Furthermore, you can build conditions and exclusions into your policies.
- If you are using an Oracle Audit Vault and Database Firewall environment, then the unified audit trail greatly facilitates the collection of audit data, because all of this data will come from one location. See *Oracle Audit Vault and Database Firewall Administrator's Guide* for more information.

Checking if Your Database Has Migrated to Unified Auditing

The `V$OPTION` dynamic view indicates if your database has been migrated to unified auditing.

- Query the `VALUE` column of the `V$OPTION` dynamic view as follows, entering `Unified Auditing` in the case shown:

```
SELECT VALUE FROM V$OPTION WHERE PARAMETER = 'Unified Auditing';

PARAMETER         VALUE
----------------  ----------
Unified Auditing  TRUE
```

This output shows that unified auditing is enabled. If unified auditing has not been enabled, then the output is `FALSE`.

Mixed Mode Auditing

Mixed mode auditing is the default auditing in a newly installed database.

About Mixed Mode Auditing

Mixed mode auditing enables both traditional (that is, the audit facility from releases earlier than release 12c) and the new audit facilities (unified auditing).

When you create a new database, by default the database uses mixed mode auditing.

You can enable the database in either of these two modes: the mixed mode auditing or pure unified auditing mode. Even though the features of unified auditing are enabled in both these modes, there are differences between them. In mixed mode, you can use the new unified audit facility alongside the traditional auditing facility. In pure unified auditing, you only use the unified audit facility.

Table 23-1 summarizes the features of these two modes and how you enable them.

Table 23-1 Differences Between MIxed Mode Audting and Pure Unified Auditing

Mode	Features	How to Enable
Mixed mode auditing	Has both traditional and unified auditing	Enable any unified audit policy. There is no need to restart the database.
Pure unified auditing	Has only unified auditing	Link the `oracle` binary with `uniaud_on`, and then restart the database. *Oracle Database Upgrade Guide* describes how to enable pure unified auditing.

Mixed mode is intended to introduce unified auditing, so that you can have a feel of how it works and what its nuances and benefits are. Mixed mode enables you to migrate your existing applications and scripts to use unified auditing. Once you have decided to use pure unified auditing, you can relink the `oracle` binary with the unified audit option turned on and thereby enable it as the one and only audit facility the Oracle database runs. If you decide to revert back to mixed mode, you can.

As in previous releases, the traditional audit facility is driven by the `AUDIT_TRAIL` initialization parameter. Only for mixed mode auditing, you should set this parameter to the appropriate traditional audit trail. This traditional audit trail will then be populated with audit records, along with the unified audit trail.

When you upgrade your database to the current release, traditional auditing is preserved, and the new audit records are written to the traditional audit trail. After you complete the migration, the audit records from the previous release are still available in those audit trails. You then can archive and purge these older audit trails by using the `DBMS_AUDIT_MGMT` PL/SQL procedures, based on your enterprise retention policies.

See Also:

- How the Unified Auditing Migration Affects Individual Audit Features, for a comparison of the features available in the pre-migrated and post-migrated auditing environments
- Checking if Your Database Has Migrated to Unified Auditing
- *Oracle Database Upgrade Guide* for information about migrating your databases to unified auditing, and for references to the documentation you should use if you choose not to migrate

How Database Creation Determines the Type of Auditing You Have Enabled

Unified auditing uses the `$ORACLE_BASE/audit` directory as the location for the new format operating system files.

For newly created databases, mixed mode auditing is enabled by default through the predefined policy `ORA_SECURECONFIG`.

To start using unified auditing, you must enable at least one unified audit policy, and to stop using it, disable all unified audit policies.

Capabilities of Mixed Mode Auditing

Mixed mode auditing provides the several capabilities.

These capabilities are as follows:

- It enables the use of all existing auditing initialization parameters: `AUDIT_TRAIL`, `AUDIT_FILE_DEST`, `AUDIT_SYS_OPERATIONS`, and `AUDIT_SYSLOG_LEVEL`.
- It writes mandatory audit records only to the traditional audit trails.
- It bases standard audit records on the standard audit configuration, and writes these records to the audit trail designated by the `AUDIT_TRAIL` initialization parameter.

  However, be aware that standard audit trail records are also generated based on unified audit policies and only these audit records are written to the unified audit trail. The standard audit records generated as a result of unified audit policies follow the semantics of unified audit policy enablement.
- Administrative user sessions generate `SYS` audit records. These records are written if the `AUDIT_SYS_OPERATIONS` initialization parameter is set to `TRUE`. This process writes the records only to the traditional audit trails. However, when unified audit policies are enabled for administrative users, these unified audit records are also written to unified audit trail.
- The format of the audit records that are written to traditional audit trails remains the same as in Oracle Database 11*g* Release 2.
- By default, Oracle Database immediately writes unified audit records to an internal relational table in the `AUDSYS` schema. See Writing the Unified Audit Trail Records to the AUDSYS Schema for more information.

- The performance cost of writing an audit record is equivalent to the sum of the times required for generating and writing an audit record to the traditional audit trail and the unified audit trail.
- Mixed mode auditing provides a glance of the unified audit mode features. Oracle recommends that you migrate to unified audit mode once you are comfortable with the new style of audit policies and audit trail. To migrate to unified auditing, see *Oracle Database Upgrade Guide*.

Who Can Perform Auditing?

Oracle provides two roles for users who perform auditing: `AUDIT_ADMIN` and `AUDIT_VIEWER`.

To perform any kind of auditing, you must be granted the `AUDIT_ADMIN` role. An auditor can view audit data after being granted the `AUDIT_VIEWER` role.

The privileges that these roles provide are as follows:

- **AUDIT_ADMIN role.** This role enables you to create unified and fine-grained audit policies, use the `AUDIT` and `NOAUDIT` SQL statements, view audit data, and manage the audit trail administration. Grant this role only to trusted users.
- **AUDIT_VIEWER role.** This role enables users to view and analyze audit data. It provides the `EXECUTE` privilege on the `DBMS_AUDIT_UTIL` PL/SQL package. The kind of user who needs this role is typically an external auditor.

> **Note:**
>
> In previous releases, users were allowed to add and remove audit configuration to objects in their own schemas without any additional privileges. This ability is no longer allowed.

About Auditing in a Multitenant Environment

You can use unified auditing in a multitenant environment.

You can apply audit settings to individual PDBs or to the CDB, depending on the type of policy. In a multitenant environment, each PDB, including the root, has own unified audit trail.

See the following sections for more information:

- **Unified audit policies created with the CREATE AUDIT POLICY and AUDIT statements:** You can create policies for both the root and individual PDBs.
- **Fine-grained audit policies:** You can create policies for individual PDBs only, not the root.
- **Purging the audit trail:** You can perform purge operations for both the root and individual PDBs.

Auditing in a Distributed Database

Auditing is site autonomous in that a database instance audits only the statements issued by directly connected users.

A local Oracle Database node cannot audit actions that take place in a remote database.

24 Configuring Audit Policies

Unified auditing supports custom unified audit policies, predefined unified auditing policies, and fine-grained auditing.

Selecting an Auditing Type

You can audit general activities (such as SQL statement actions), commonly used auditing activities, or fine-grained audit scenarios.

Auditing SQL Statements, Privileges, and Other General Activities

You can audit many types of objects, from SQL statements to other Oracle Database components, such as Oracle Database Vault..

In addition, you can create policies that use conditions. However, if you want to audit specific columns or use event handlers, you must use fine-grained auditing.

The general steps for performing this type of auditing are as follows:

1. In most cases, use the `CREATE AUDIT POLICY` statement to create an audit policy. If you must audit application context values, then use the `AUDIT` statement.

   See the relevant categories under Auditing Activities with Unified Audit Policies and the AUDIT Statement.

2. If you are creating an audit policy, then use the `AUDIT` statement to enable it and optionally apply (or exclude) the audit settings to one or more users, including administrative users who log in with the `SYSDBA` administrative privilege (for example, the `SYS` user).

   `AUDIT` also enables you to create an audit record upon an action's success, failure, or both.

   See Enabling and Applying Unified Audit Policies to Users and Roles.

3. Query the `UNIFIED_AUDIT_TRAIL` view to find the generated audit records.

   See also Audit Policy Data Dictionary Views for additional views.

4. Periodically archive and purge the contents of the audit trail.

   See Purging Audit Trail Records.

Auditing Commonly Used Security-Relevant Activities

Oracle Database provides a set default unified audit policies that you can choose from for commonly used security-relevant audits.

The general steps for performing this type of auditing are as follows:

1. See Auditing Activities with the Predefined Unified Audit Policies to learn about the default audit policies.

2. Use the `AUDIT` statement enable the policy and optionally apply (or exclude) the audit settings to one or more users.

   See Enabling and Applying Unified Audit Policies to Users and Roles.

3. Query the `UNIFIED_AUDIT_TRAIL` view to find the generated audit records.

   See also Audit Policy Data Dictionary Views for additional views.

4. Periodically archive and purge the contents of the audit trail.

   See Purging Audit Trail Records.

Auditing Specific, Fine-Grained Activities

Use fine-grained auditing if you want to audit individual columns and use event handlers.

This type of auditing provides all the features available in unified audit policies.

The general steps for fine-grained auditing are as follows:

1. See Auditing Specific Activities with Fine-Grained Auditing to understand more about auditing specific activities.

2. Use the `DBMS_FGA` PL/SQL package to configure fine-grained auditing policies. See Using the DBMS_FGA PL/SQL Package to Manage Fine-Grained Audit Policies.

3. Query the `UNIFIED_AUDIT_TRAIL` view to find the generated audit records.

   See also Audit Policy Data Dictionary Views for additional views.

4. Periodically archive and purge the contents of the audit trail.

   See Purging Audit Trail Records.

Auditing Activities with Unified Audit Policies and the AUDIT Statement

You can use the `CREATE AUDIT POLICY` and `AUDIT` statements to use unified auditing policies.

About Auditing Activities with Unified Audit Policies and AUDIT

You can audit the several types of activities, using unified audit policies and the `AUDIT` SQL statement.

The kinds of activities that you can audit are as follows:

- User accounts (including administrative users who log in with the `SYSDBA` administrative privilege), roles, and privileges
- Object actions, such as dropping a table or a running a procedure
- Application context values
- Activities from Oracle Database Real Application Security, Oracle Recovery Manager, Oracle Data Mining, Oracle Data Pump, Oracle SQL*Loader direct path events, Oracle Database Vault, and Oracle Label Security

To accomplish this, depending on what you want to audit, use the following:

- **Unified audit policies.** A unified audit policy is a named group of audit settings that enable you to audit a particular aspect of user behavior in the database. To create the policy, you use the `CREATE AUDIT POLICY` statement. The policy can be as simple as auditing the activities of a single user or you can create complex audit policies that use conditions. You can have more than one audit policy in effect at a time in a database. An audit policy can contain both system-wide and object-specific audit options. Most of the auditing that you will do for general activities (including standard auditing) requires the use of audit policies.
- **AUDIT and NOAUDIT SQL statements.** The `AUDIT` and `NOAUDIT` SQL statements enable you to, respectively, enable and disable an audit policy. The `AUDIT` statement also lets you include or exclude specific users for the policy. The `AUDIT` and `NOAUDIT` statements also enable you to audit application context values.
- **For Oracle Recovery Manager, you do not create unified audit policies.** The `UNIFIED_AUDIT_TRAIL` view automatically captures commonly audited Recovery Manager events.

Best Practices for Creating Unified Audit Policies

You can enable multiple policies at a time in the database, but ideally, limit the number of enabled policies.

The unified audit policy syntax is designed so that you can write one policy that covers all the audit settings that your database needs. A good practice is to group related options into a single policy instead of creating multiple small policies. This enables you to manage the policies much easier. As an example, the default audit policies described in Auditing Activities with the Predefined Unified Audit Policies each contain multiple audit settings within one unified audit policy.

Limiting the number of enabled audit policies for a user session has the following benefits:

- It reduces the logon overhead that is associated with loading the audit policy's details into the session's UGA memory. If the enabled policy count is less, then less time is spent in loading the policy information.
- It reduces the session's UGA memory consumption, because a fewer number of policies are required to be cached in UGA memory.
- It makes the internal audit check functionality more efficient, which determines whether to generate an audit record for its associated event.

Syntax for Creating a Unified Audit Policy

To create a unified audit policy, you must use the `CREATE AUDIT POLICY` statement.

When you create a unified audit policy, Oracle Database stores it in a first class object that is owned by the `SYS` schema, not in the schema of the user who created the policy.

Example 24-1 shows the syntax for the `CREATE AUDIT POLICY` statement.

Example 24-1 Syntax for the CREATE AUDIT POLICY Statement

```
CREATE AUDIT POLICY policy_name
    { {privilege_audit_clause [action_audit_clause ] [role_audit_clause ]}
        | { action_audit_clause  [role_audit_clause ] }
        | { role_audit_clause }
     }
```

```
[WHEN audit_condition EVALUATE PER {STATEMENT|SESSION|INSTANCE}]
[CONTAINER = {CURRENT | ALL}];
```

In this specification:

- *privilege_audit_clause* describes privilege-related audit options. See Auditing System Privileges for details. The detailed syntax for configuring privilege audit options is as follows:

  ```
  privilege_audit_clause  :=  PRIVILEGES  privilege1 [, privilege2]
  ```

- *action_audit_clause* and *standard_actions* describe object action-related audit options. See Auditing Object Actions. The syntax is as follows:

  ```
  action_audit_clause := {standard_actions | component_actions}
                                            [, component_actions ]
  standard_actions :=
       ACTIONS action1 [ ON {schema.obj_name
                                             | DIRECTORY directory_name
                                             | MINING MODEL schema.obj_name
                                              }
                  ]
             [, action2 [ ON {schema.obj_name
                                             | DIRECTORY directory_name
                                             | MINING MODEL schema.obj_name
                     }
                  ]
  ```

- *component_actions* enables you to create an audit policy for Oracle Label Security, Oracle Database Real Application Security, Oracle Database Vault, Oracle Data Pump, or Oracle SQL*Loader. See the appropriate section under Auditing Activities with Unified Audit Policies and the AUDIT Statement for more information. The syntax is:

  ```
  component_actions :=
       ACTIONS COMPONENT=[OLS|XS] action1 [,action2 ] |
       ACTIONS COMPONENT=DV DV_action ON DV_object_name |
       ACTIONS COMPONENT=DATAPUMP [ EXPORT | IMPORT | ALL ] |
       ACTIONS COMPONENT=DIRECT_LOAD [ LOAD | ALL ]
  ```

- *role_audit_clause* enables you to audit roles. See Auditing Roles. The syntax is:

  ```
  role_audit_clause := ROLES role1 [, role2]
  ```

- `WHEN` *audit_condition* `EVALUATE PER` enables you to specify a function to create a condition for the audit policy and the evaluation frequency. You must include the `EVALUATE PER` clause with the `WHEN` condition. See Creating a Condition for a Unified Audit Policy. The syntax is:

  ```
  WHEN 'audit_condition := function operation value_list'
  EVALUATE PER {STATEMENT|SESSION|INSTANCE}
  ```

- `CONTAINER`, used for multitenant environments, enables you to create an audit policy as either a local audit policy (for the local pluggable database (PDB)) or as a common audit policy. See Unified Audit Policies or AUDIT Settings in a Multitenant Environment.

This syntax is designed to audit any of the components listed in the policy. For example, suppose you create the following policy:

```
CREATE AUDIT POLICY table_pol
PRIVILEGES CREATE ANY TABLE, DROP ANY TABLE
ROLES emp_admin, sales_admin;
```

The audit trail will capture SQL statements that require the `CREATE ANY TABLE` system privilege or the `DROP ANY TABLE` system privilege or any system privilege directly granted to the role `emp_admin` or any system privilege directly granted to the role `sales_admin`. (Be aware that it audits privileges that are *directly* granted, not privileges that are granted recursively through a role.)

After you create the policy, you must enable it by using the `AUDIT` statement. Optionally, you can apply the policy to one or more users, exclude one or more users from the policy, and designate whether an audit record is written when the audited action succeeds, fails, or both succeeds or fails. See Enabling and Applying Unified Audit Policies to Users and Roles.

Auditing Roles

You can use the `CREATE AUDIT POLICY` statement to audit database roles.

About Role Auditing

When you audit a role, Oracle Database audits all system privileges that are directly granted to the role.

You can audit any role, including user-defined roles. If you create a common unified audit policy for roles with the `ROLES` audit option, then you must specify only common roles in the role list. When such a policy is enabled, Oracle Database audits all system privileges that are commonly and directly granted to the common role. The system privileges that are locally granted to the common role will not be audited. To find if a role was commonly granted, query the `DBA_ROLES` data dictionary view. To find if the privileges granted to the role were commonly granted, query the `ROLE_SYS_PRIVS` view.

Configuring Role Unified Audit Policies

To create a unified audit policy to capture role use, you must include the `ROLES` clause in the `CREATE AUDIT POLICY` statement.

- Use the following syntax to create a unified audit policy that audits roles:

  ```
  CREATE AUDIT POLICY policy_name
   ROLES role1 [, role2];
  ```

For example:

```
CREATE AUDIT POLICY audit_roles_pol
 ROLES IMP_FULL_DATABASE, EXP_FULL_DATABASE;
```

You can build more complex role unified audit policies, such as those that include conditions. Remember that after you create the policy, you must use the `AUDIT` statement to enable it.

Example: Auditing the DBA Role in a Multitenant Environment

The `CREATE AUDIT POLICY` statement can audit roles in a multitenant environment.

The following example shows how to audit a predefined common role `DBA` in a multitenant environment.

Example 24-2   Auditing the DBA Role in a Multitenant Environment

```
CREATE AUDIT POLICY role_dba_audit_pol
 ROLES DBA
 CONTAINER = ALL;

AUDIT POLICY role_dba_audit_pol;
```

Auditing System Privileges

You can use the `CREATE AUDIT POLICY` statement to audit system privileges.

About System Privilege Auditing

System privilege auditing audits activities that use a system privilege, such as `READ ANY TABLE`.

In this kind of auditing, SQL statements that require the audited privilege to succeed are recorded.

A single unified audit policy can contain both privilege and action audit options. Do not audit the privilege use of administrative users such as `SYS`. Instead, audit their object actions.

Note:

You can audit system privileges, objects, database events, and so on. However, if you must find database privilege usage (for example, which privileges that have been granted to a given role are used), and generate a report of the used and unused privileges, then you can create a privilege capture. See *Oracle Database Vault Administrator's Guide* for more information.

System Privileges That Can Be Audited

You can audit the use of almost any system privilege.

To find a list of auditable system privileges, you can query the `SYSTEM_PRIVILEGE_MAP` table.

For example:

```
SELECT NAME FROM SYSTEM_PRIVILEGE_MAP;

NAME
-------------
ALTER ANY CUBE BUILD PROCESS
SELECT ANY CUBE BUILD PROCESS
ALTER ANY MEASURE FOLDER
...
```

Similar to action audit options, privilege auditing audits the use of system privileges that have been granted to database users. If you set similar audit options for both SQL statement and privilege auditing, then only a single audit record is generated. For example, if two policies exist, with one auditing `EXECUTE PROCEDURE` specifically on the

HR.PROC procedure and the second auditing EXECUTE PROCEDURE in general (all procedures), then only one audit record is written.

Privilege auditing does not occur if the action is already permitted by the existing owner and object privileges. Privilege auditing is triggered only if the privileges are insufficient, that is, only if what makes the action possible is a system privilege. For example, suppose that user SCOTT has been granted the SELECT ANY TABLE privilege and SELECT ANY TABLE is being audited. If SCOTT selects his own table (for example, SCOTT.EMP), then the SELECT ANY TABLE privilege is not used. Because he performed the SELECT statement within his own schema, no audit record is generated. On the other hand, if SCOTT selects from another schema (for example, the HR.EMPLOYEES table), then an audit record *is* generated. Because SCOTT selected a table outside his own schema, he needed to use the SELECT ANY TABLE privilege.

System Privileges That Cannot Be Audited

Several system privileges cannot be audited.

These privileges are:

- INHERIT ANY PRIVILEGE
- INHERIT PRIVILEGE
- TRANSLATE ANY SQL
- TRANSLATE SQL

Configuring a Unified Audit Policy to Capture System Privilege Use

The PRIVILEGES clause in the CREATE AUDIT POLICY statement audits system privilege use.

- Use the following syntax to create a unified audit policy that audits privileges:

  ```
  CREATE AUDIT POLICY policy_name
   PRIVILEGES privilege1 [, privilege2];
  ```

  For example:

  ```
  CREATE AUDIT POLICY my_simple_priv_policy
   PRIVILEGES SELECT ANY TABLE, CREATE LIBRARY;
  ```

You can build more complex privilege unified audit policies, such as those that include conditions. Remember that after you create the policy, you must use the AUDIT statement to enable it.

Example: Auditing a User Who Has ANY Privileges

The CREATE AUDIT POLICY statement can audit users for ANY privileges.

Example 24-3 shows how to audit several ANY privileges of the user HR_MGR.

Example 24-3   Auditing a User Who Has ANY Privileges

```
CREATE AUDIT POLICY hr_mgr_audit_pol
 PRIVILEGES DROP ANY TABLE, DROP ANY CONTEXT, DROP ANY INDEX, DROP ANY LIBRARY;

AUDIT POLICY hr_mgr_audit_pol BY HR_MGR;
```

Example: Using a Condition to Audit a System Privilege

The CREATE AUDIT POLICY statement can create an audit policy that uses a condition to audit a system privilege.

Example 24-4 shows how to use a condition to audit privileges that are used by two operating system users, psmith and jrawlins.

Example 24-4 Using a Condition to Audit a System Privilege

```
CREATE AUDIT POLICY os_users_priv_pol
 PRIVILEGES SELECT ANY TABLE, CREATE LIBRARY
 WHEN 'SYS_CONTEXT (''USERENV'', ''OS_USER'') IN (''psmith'', ''jrawlins'')'
 EVALUATE PER SESSION;

AUDIT POLICY os_users_priv_pol;
```

How System Privilege Unified Audit Policies Appear in the Audit Trail

The UNIFIED_AUDIT_TRAIL data dictionary view lists system privilege audit events.

The following example, based on the unified audit policy os_users_priv_pol that was created in Example 24-4, shows a list of privileges used by the operating system user psmith.

```
SELECT SYSTEM_PRIVILEGE_USED FROM UNIFIED_AUDIT_TRAIL
 WHERE OS_USERNAME = 'PSMITH' AND UNIFIED_AUDIT_POLICIES = 'OS_USERS_PRIV_POL';

SYSTEM_PRIVILEGE_USED
----------------------
SELECT ANY TABLE
DROP ANY TABLE
```

Note:

If you have created an audit policy for the SELECT ANY TABLE system privilege, whether the user has exercised the READ object privilege or the SELECT object privilege will affect the actions that the audit trail captures.

Auditing Administrative Users

You can create unified audit policies to capture the actions of administrative user accounts, such as SYS.

Administrative User Accounts That Can Be Audited

Oracle Database provides administrative user accounts that are associated with administrative privileges.

Table 24-1 lists default administrative user accounts and the administrative privileges with which they are typically associated.

Table 24-1 Administrative Users and Administrative Privileges

Administrative User Account	Administrative Privilege
SYS	SYSDBA
PUBLIC[1]	SYSOPER
SYSASM	SYSASM
SYSBACKUP	SYSBACKUP
SYSDG	SYSDG
SYSKM	SYSKM

1 PUBLIC refers to the user PUBLIC, which is the effective user when you log in with the SYSOPER administrative privilege. It does not refer to the PUBLIC role.

Configuring a Unified Audit Policy to Capture Administrator Activities

The CREATE AUDIT POLICY statement can audit administrative users.

- To audit administrative users, create a unified audit policy and then apply this policy to the user, the same as you would for non-administrative users. Note that top-level statements by administrative users are mandatorily audited until the database opens.

Example: Auditing the SYS User

The CREATE AUDIT POLICY statement can audit the SYS user.

Example 24-5 shows how to audit grants of the DBMS_FGA PL/SQL package by user SYS.

Example 24-5 Auditing the SYS User

```
CREATE AUDIT POLICY dbms_fga_grants
 ACTIONS GRANT
 ON DBMS_FGA;

AUDIT POLICY dbms_fga_grants BY SYS;
```

Auditing Object Actions

You can use the CREATE AUDIT POLICY statement to audit object actions.

About Auditing Object Actions

You can audit actions performed on specific objects, such as UPDATE statements on the HR.EMPLOYEES table.

The audit can include both DDL and DML statements that were used on the object. A single unified audit policy can contain both privilege and action audit options, as well as audit options set for multiple objects.

Object Actions That Can Be Audited

Auditing object actions can be broad or focused (for example, auditing all user actions or only a select list of user actions).

Table 24-2 lists the object-level standard database action options. Audit policies for the `SELECT` SQL statement will capture `READ` actions as well as `SELECT` actions.

Table 24-2 Object-Level Standard Database Action Audit Option

Object	SQL Action That Can Be Audited
Table	`ALTER`, `AUDIT`, `COMMENT`, `DELETE`, `FLASHBACK`, `GRANT`, `INDEX`, `INSERT`, `LOCK`, `RENAME`, `SELECT`, `UPDATE`
View	`AUDIT`, `COMMENT`, `DELETE`, `FLASHBACK`, `GRANT`, `INSERT`, `LOCK`, `RENAME`, `SELECT`, `UPDATE`
Sequence	`ALTER`, `AUDIT`, `GRANT`, `SELECT`
Procedure (including triggers)	`AUDIT`, `EXECUTE`, `GRANT`
Function	`AUDIT`, `EXECUTE`, `GRANT`
Package	`AUDIT`, `EXECUTE`, `GRANT`
Materialized views	`ALTER`, `AUDIT`, `COMMENT`, `DELETE`, `INDEX`, `INSERT`, `LOCK`, `SELECT`, `UPDATE`
Mining Model	`AUDIT`, `COMMENT`, `GRANT`, `RENAME`, `SELECT`
Directory	`AUDIT`, `GRANT`, `READ`
Library	`EXECUTE`, `GRANT`
Object type	`ALTER`, `AUDIT`, `GRANT`
Java schema objects (source, class, resource)	`AUDIT`, `EXECUTE`, `GRANT`

Configuring an Object Action Unified Audit Policy

The `ACTIONS` clause in the `CREATE AUDIT POLICY` statement creates a policy that captures object actions.

- Use the following syntax to create a unified audit policy that audits object actions:

  ```
  CREATE AUDIT POLICY policy_name
   ACTIONS action1 [, action2 ON object1] [, action3 ON object2];
  ```

For example:

```
CREATE AUDIT POLICY my_simple_obj_policy
 ACTIONS SELECT ON OE.ORDERS, UPDATE ON HR.EMPLOYEES;
```

Note that you can audit multiple actions on multiple objects, as shown in this example.

You can build complex object action unified audit policies, such as those that include conditions. Remember that after you create the policy, you must use the `AUDIT` statement to enable it.

Example: Auditing Actions on SYS Objects

The CREATE AUDIT POLICY statement can audit actions on SYS objects.

Example 24-6 shows how to create an audit policy that audits SELECT statements on the SYS.USER$ system table. The audit policy applies to all users, including SYS and SYSTEM.

Example 24-6 Auditing Actions on SYS Objects

```
CREATE AUDIT POLICY select_user_dictionary_table_pol ACTIONS SELECT ON SYS.USER$;

AUDIT POLICY select_user_dictionary_table_pol;
```

Example: Auditing Multiple Actions on One Object

The CREATE AUDIT POLICY statement can audit multiple actions on one object.

Example 24-7 shows how to audit multiple SQL statements performed by users jrandolph and phawkins on the app_lib library.

Example 24-7 Auditing Multiple Actions on One Object

```
CREATE AUDIT POLICY actions_on_hr_emp_pol1
 ACTIONS EXECUTE, GRANT
 ON app_lib;

AUDIT POLICY actions_on_hr_emp_pol1 BY jrandolph, phawkins;
```

Example: Auditing Both Actions and Privileges on an Object

The CREATE AUDIT POLICY statement can audit both actions and privileges on an object, using a single policy.

Example 24-8 shows a variation of Example 24-7, in which all EXECUTE and GRANT statements on the app_lib library using the CREATE LIBRARY privilege are audited.

Example 24-8 Auditing Both Actions and Privileges on an Object

```
CREATE AUDIT POLICY actions_on_hr_emp_pol2
 PRIVILEGES CREATE LIBRARY
 ACTIONS EXECUTE, GRANT
 ON app_lib;

AUDIT POLICY actions_on_hr_emp_pol2 BY jrandolph, phawkins;
```

You can audit directory objects. For example, suppose you create a directory object that contains a preprocessor program that the ORACLE_LOADER access driver will use. You can audit anyone who runs this program within this directory object.

Example: Auditing All Actions on a Table

The CREATE AUDIT POLICY statement can audit all actions on a table.

You can use the keyword ALL to audit all actions. Example 24-9 shows how to audit all actions on the HR.EMPLOYEES table, except actions by user pmulligan.

Example 24-9 Auditing All Actions on a Table

```
CREATE AUDIT POLICY all_actions_on_hr_emp_pol
 ACTIONS ALL ON HR.EMPLOYEES;

AUDIT POLICY all_actions_on_hr_emp_pol EXCEPT pmulligan;
```

Example: Auditing All Actions in the Database

The `CREATE AUDIT POLICY` statement can audit all actions in the database.

Example 24-10 shows how to audit all actions in the entire database.

Example 24-10 Auditing All Actions in the Database

```
CREATE AUDIT POLICY all_actions_pol ACTIONS ALL;

AUDIT POLICY all_actions_pol;
```

How Object Action Unified Audit Policies Appear in the Audit Trail

The `UNIFIED_AUDIT_TRAIL` data dictionary view lists object action audit events.

For example:

```
SELECT ACTION_NAME, OBJECT_SCHEMA, OBJECT_NAME FROM UNIFIED_AUDIT_TRAIL
WHERE DBUSERNAME = 'SYS';

ACTION_NAME OBJECT_SCHEMA OBJECT_NAME
----------- ------------- ------------
SELECT      HR            EMPLOYEES
```

Auditing Functions, Procedures, Packages, and Triggers

You can audit functions, procedures, PL/SQL packages, and triggers.

The areas that you can audit are as follows:

- You can individually audit standalone functions, standalone procedures, and PL/SQL packages.
- If you audit a PL/SQL package, Oracle Database audits all functions and procedures within the package.
- If you enable auditing for all executions, Oracle Database audits all triggers in the database, as well as all the functions and procedures within PL/SQL packages.
- You cannot audit individual functions or procedures within a PL/SQL package.
- When you audit the `EXECUTE` operation on a PL/SQL stored procedure or stored function, the database considers only its ability to find the procedure or function and authorize its execution when determining the success or failure of the operation for the purposes of auditing. Therefore, if you specify the `WHENEVER NOT SUCCESSFUL` clause, then only invalid object errors, non-existent object errors, and authorization failures are audited; errors encountered during the execution of the procedure or function are not audited. If you specify the `WHENEVER SUCCESSFUL` clause, then all executions that are not blocked by invalid object errors, non-existent object errors, or authorization failures are audited, regardless of whether errors are encountered during execution.

Auditing of Oracle Virtual Private Database Predicates

The unified audit trail automatically captures the predicates that are used in Oracle Virtual Private Database (VPD) policies.

You do not need to create a unified audit policy to capture the VPD predicate audit information.

This type of audit enables you to identify the predicate expression that was run as part of a DML operation and thereby help you to identify other actions that may have occurred as part of the DML operation. For example, if a malicious attack on your database is performed using a VPD predicate, then you can track the attack by using the unified audit trail. In addition to predicates from user-created VPD policies, the internal predicates from Oracle Label Security and Oracle Real Application Security policies are captured as well. For example, Oracle Label Security internally creates a VPD policy while applying an OLS policy to a table. Oracle Real Application Security generates a VPD policy while enabling an Oracle RAS policy.

The unified audit trail writes this predicate information to the `RLS_INFO` column of the `UNIFIED_AUDIT_TRAIL` data dictionary view. If you have fine-grained audit policies, then the `RLS_INFO` column of these views captures VPD predicate information as well.

The audit trail can capture the predicates and their corresponding policy names if multiple VPD policies are enforced on the object. The audit trail captures the policy schema and policy name to enable you to differentiate predicates that are generated from different policies. By default, this information is concatenated in the `RLS_INFO` column, but Oracle Database provides a function in the `DBMS_AUDIT_UTIL` PL/SQL package that enables you to reformat the results in an easy-to-read format.

The following example shows how you can audit the predicates of a VPD policy:

1. Create the following VPD policy function:

```
CREATE OR REPLACE FUNCTION auth_orders(
  schema_var IN VARCHAR2,
  table_var  IN VARCHAR2
 )
 RETURN VARCHAR2
 IS
  return_val VARCHAR2 (400);
 BEGIN
  return_val := 'SALES_REP_ID = 159';
  RETURN return_val;
 END auth_orders;
/
```

2. Create the following VPD policy:

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema    => 'oe',
    object_name      => 'orders',
    policy_name      => 'orders_policy',
    function_schema  => 'sec_admin',
    policy_function  => 'auth_orders',
    statement_types  => 'select, insert, update, delete'
   );
 END;
/
```

3. Create and enable the following the unified audit policy:

```
CREATE AUDIT POLICY oe_pol
 ACTIONS SELECT ON OE.ORDERS;

AUDIT POLICY oe_pol;
```

4. Connect as user `OE` and query the `OE.ORDERS` table.

```
CONNECT OE
Enter password: password

SELECT COUNT(*) FROM ORDERS;
```

5. Connect as a user who has been granted the `AUDIT_ADMIN` role, and then query the `UNIFIED_AUDIT_TRAIL` data dictionary view.

```
CONNECT sec_admin
Enter password: password

SELECT RLS_INFO FROM UNIFIED_AUDIT_TRAIL;
```

   Output similar to the following should appear:

```
((POLICY_TYPE=[3]'VPD'),(POLICY_SCHEMA=[9]'SEC_ADMIN'),
(POLICY_NAME=[13]'ORDERS_POLICY'),(PREDICATE=[16]'SALES_REP_ID=159'));
```

6. To extract these details and add them to their own columns, run the appropriate function from the `DBMS_AUDIT_UTIL` PL/SQL package.

   For unified auditing, you must run the `DBMS_AUDIT_UTIL.DECODE_RLS_INFO_ATRAIL_UNI` function.

   For example:

```
SELECT DBUSERNAME, ACTION_NAME, OBJECT_NAME, SQL_TEXT,
  RLS_PREDICATE, RLS_POLICY_TYPE, RLS_POLICY_OWNER, RLS_POLICY_NAME
  FROM TABLE (DBMS_AUDIT_UTIL.DECODE_RLS_INFO_ATRAIL_UNI
  (CURSOR (SELECT * FROM UNIFIED_AUDIT_TRAIL)));
```

   The reformatted audit trail output appears similar to the following:

```
DBUSERNAME ACTION_NAME OBJECT_NAME SQL_TEXT
---------- ----------- ----------- ---------------------------
RLS_PREDICATE       RLS_POLICY_TYPE RLS_POLICY_OWNER RLS_POLICY_NAME
------------------ ---------------  ---------------- ---------------
OE         SELECT      ORDERS      SELECT COUNT(*) FROM ORDERS
SALES_REP_ID = 159  VPD              SEC_ADMIN        ORDERS_POLICY
```

See Also:

- Using Oracle Virtual Private Database to Control Data Access for more information about Oracle Virtual Private Database
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_AUDIT_UTIL` PL/SQL package

Audit Policies for Oracle Virtual Private Database Policy Functions

Auditing can affect dynamic VPD policies, static VPD policies, and context-sensitive VPD policies.

- **Dynamic policies:** Oracle Database evaluates the policy function twice, once during SQL statement parsing and again during execution. As a result, two audit records are generated for each evaluation.
- **Static policies:** Oracle Database evaluates the policy function once and then caches it in the SGA. As a result, only one audit record is generated.
- **Context-sensitive policies:** Oracle Database executes the policy function once, during statement parsing. As a result, only one audit record is generated.

Unified Auditing with Editioned Objects

When an editioned object has a unified audit policy, it applies in all editions in which the object is visible.

When an editioned object is actualized, any unified audit policies that are attached to it are newly attached to the new actual occurrence. When you newly apply a unified audit policy to an inherited editioned object, this action will actualize it.

You can find the editions in which audited objects appear by querying the `OBJECT_NAME` and `OBJ_EDITION_NAME` columns in the `UNIFIED_AUDIT_TRAIL` data dictionary view.

See Also:

Oracle Database Development Guide for detailed information about editions

Auditing the READ ANY TABLE and SELECT ANY TABLE Privileges

The `CREATE AUDIT POLICY` statement can audit the `READ ANY TABLE` and `SELECT ANY TABLE` privileges.

About Auditing the READ ANY TABLE and SELECT ANY TABLE Privileges

You can create unified audit policies that capture the use of the `READ ANY TABLE` and `SELECT ANY TABLE` system privileges.

Based on the action that the user tried to perform and the privilege that was granted to the user, the `SYSTEM_PRIVILEGE_USED` column of the `UNIFIED_AUDIT_TRAIL` data dictionary view will record either the `READ ANY TABLE` system privilege or the `SELECT ANY TABLE` system privilege. For example, suppose the user has been granted the `SELECT ANY TABLE` privilege and then performs a query on a table. The audit trail will record that the user used the `SELECT ANY TABLE` system privilege. If the user was granted `READ ANY TABLE` and performed the same query, then the `READ ANY TABLE` privilege is recorded.

Creating a Unified Audit Policy to Capture READ Object Privilege Operations

You can create unified audit policies that capture `READ` object privilege operations.

- To create a unified audit policy to capture any `READ` object operations, create the policy for the `SELECT` statement, not for the `READ` statement.

For example:

```
CREATE AUDIT POLICY read_hr_employees
 ACTIONS SELECT ON HR.EMPLOYEES;
```

For any `SELECT` object operations, also create the policy on the `SELECT` statement, as with other object actions that you can audit.

How the Unified Audit Trail Captures READ ANY TABLE and SELECT ANY TABLE

The unified audit trail captures `SELECT` behavior based on whether a user has the `READ ANY TABLE` or the `SELECT ANY TABLE` privilege.

Table 24-3 describes how the unified audit trail captures these actions.

Table 24-3 Auditing Behavior for READ ANY TABLE and SELECT ANY TABLE

Statement User Issues	Privilege Granted to User	System Privilege Being Audited	Expected UNIFIED_AUDIT_TRAIL Behavior
`SELECT`	`SELECT ANY TABLE`	`SELECT ANY TABLE`	Record inserted into `SYSTEM_PRIVILEGE_USED`: `SELECT ANY TABLE`
`SELECT`	`SELECT ANY TABLE`	`READ ANY TABLE`	No record
`SELECT`	`SELECT ANY TABLE`	Both `SELECT ANY TABLE` and `READ ANY TABLE`	Record inserted into `SYSTEM_PRIVILEGE_USED`: `SELECT ANY TABLE`
`SELECT`	`SELECT ANY TABLE`	Neither `SELECT ANY TABLE` nor `READ ANY TABLE`	No record
`SELECT`	`READ ANY TABLE`	`SELECT ANY TABLE`	No record
`SELECT`	`READ ANY TABLE`	`READ ANY TABLE`	Record inserted into `SYSTEM_PRIVILEGE_USED`: `READ ANY TABLE`
`SELECT`	`READ ANY TABLE`	Both `SELECT ANY TABLE` and `READ ANY TABLE`	Record inserted into `SYSTEM_PRIVILEGE_USED`: `READ ANY TABLE`
`SELECT`	`READ ANY TABLE`	Neither `SELECT ANY TABLE` nor `READ ANY TABLE`	No record
`SELECT`	Both `SELECT ANY TABLE` and `READ ANY TABLE`	`SELECT ANY TABLE`	No record, because `READ ANY TABLE` was used for access

Table 24-3 (Cont.) Auditing Behavior for READ ANY TABLE and SELECT ANY TABLE

Statement User Issues	Privilege Granted to User	System Privilege Being Audited	Expected UNIFIED_AUDIT_TRAIL Behavior
SELECT	Both SELECT ANY TABLE and READ ANY TABLE	READ ANY TABLE	Record inserted into SYSTEM_PRIVILEGE_USED: READ ANY TABLE
SELECT	Both SELECT ANY TABLE and READ ANY TABLE	Both SELECT ANY TABLE and READ ANY TABLE	Record inserted into SYSTEM_PRIVILEGE_USED: READ ANY TABLE
SELECT	Both SELECT ANY TABLE and READ ANY TABLE	Neither SELECT ANY TABLE nor READ ANY TABLE	No record
SELECT	Neither SELECT ANY TABLE nor READ ANY TABLE	SELECT ANY TABLE	No record
SELECT	Neither SELECT ANY TABLE nor READ ANY TABLE	READ ANY TABLE	No record
SELECT	Neither SELECT ANY TABLE nor READ ANY TABLE	Both SELECT ANY TABLE and READ ANY TABLE	No record
SELECT	Neither SELECT ANY TABLE nor READ ANY TABLE	Neither SELECT ANY TABLE nor READ ANY TABLE	No record
SELECT ... FOR UPDATE	SELECT ANY TABLE	SELECT ANY TABLE	Record inserted into SYSTEM_PRIVILEGE_USED: SELECT ANY TABLE
SELECT ... FOR UPDATE	SELECT ANY TABLE	READ ANY TABLE	No record
SELECT ... FOR UPDATE	SELECT ANY TABLE	Both SELECT ANY TABLE and READ ANY TABLE	Record inserted into SYSTEM_PRIVILEGE_USED: SELECT ANY TABLE
SELECT ... FOR UPDATE	SELECT ANY TABLE	Neither SELECT ANY TABLE nor READ ANY TABLE	No record
SELECT ... FOR UPDATE	READ ANY TABLE	SELECT ANY TABLE	No record
SELECT ... FOR UPDATE	READ ANY TABLE	READ ANY TABLE	No record
SELECT ... FOR UPDATE	READ ANY TABLE	Both SELECT ANY TABLE and READ ANY TABLE	No record
SELECT ... FOR UPDATE	READ ANY TABLE	Neither SELECT ANY TABLE nor READ ANY TABLE	No record
SELECT ... FOR UPDATE	Both SELECT ANY TABLE and READ ANY TABLE	SELECT ANY TABLE	Record inserted into SYSTEM_PRIVILEGE_USED: SELECT ANY TABLE
SELECT ... FOR UPDATE	Both SELECT ANY TABLE and READ ANY TABLE	READ ANY TABLE	No record, because READ ANY TABLE was used for access

Table 24-3 (Cont.) Auditing Behavior for READ ANY TABLE and SELECT ANY TABLE

Statement User Issues	Privilege Granted to User	System Privilege Being Audited	Expected UNIFIED_AUDIT_TRAIL Behavior
`SELECT ... FOR UPDATE`	Both `SELECT ANY TABLE` and `READ ANY TABLE`	Both `SELECT ANY TABLE` and `READ ANY TABLE`	Record inserted into `SYSTEM_PRIVILEGE_USED`: `SELECT ANY TABLE`
`SELECT ... FOR UPDATE`	Both `SELECT ANY TABLE` and `READ ANY TABLE`	Neither `SELECT ANY TABLE` nor `READ ANY TABLE`	No record
`SELECT ... FOR UPDATE`	Neither `SELECT ANY TABLE` nor `READ ANY TABLE`	`SELECT ANY TABLE`	No record
`SELECT ... FOR UPDATE`	Neither `SELECT ANY TABLE` nor `READ ANY TABLE`	`READ ANY TABLE`	No record
`SELECT ... FOR UPDATE`	Neither `SELECT ANY TABLE` nor `READ ANY TABLE`	Both `SELECT ANY TABLE` and `READ ANY TABLE`	No record
`SELECT ... FOR UPDATE`	Neither `SELECT ANY TABLE` nor `READ ANY TABLE`	Neither `SELECT ANY TABLE` or `READ ANY TABLE`	No record

Auditing SQL Statements and Privileges in a Multitier Environment

You can create a unified audit policy to audit the activities of a client in a multitier environment.

In a multitier environment, Oracle Database preserves the identity of a client through all tiers. Thus, you can audit actions taken on behalf of the client by a middle-tier application, by using the `BY` *`user`* clause in the `AUDIT` statement for your policy. The audit applies to all user sessions, including proxy sessions.

The middle tier can also set the user client identity in a database session, enabling the auditing of end-user actions through the middle-tier application. The end-user client identity then shows up in the audit trail.

The following example shows how to audit `SELECT TABLE` statements issued by the user `jackson`:

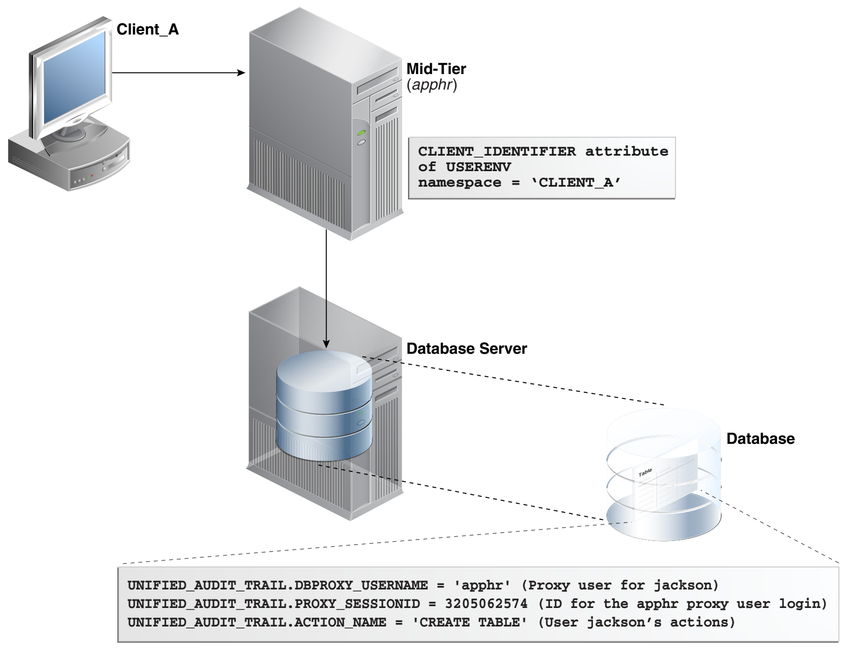
```
CREATE AUDIT POLICY tab_pol
 PRIVILEGES CREATE ANY TABLE
 ACTIONS CREATE TABLE;

AUDIT tab_pol BY jackson;
```

You can audit user activity in a multitier environment. Once audited, you can verify these activities by querying the `UNIFIED_AUDIT_TRAIL` data dictionary view.

Figure 24-1 illustrates how you can audit proxy users by querying the `PROXY_SESSIONID`, `ACTION_NAME`, and `SESSION_ID` columns of the `UNIFIED_AUDIT_TRAIL` view. In this scenario, both the database user and proxy user accounts are known to the database. Session pooling can be used.

Figure 24-1 Auditing Proxy Users

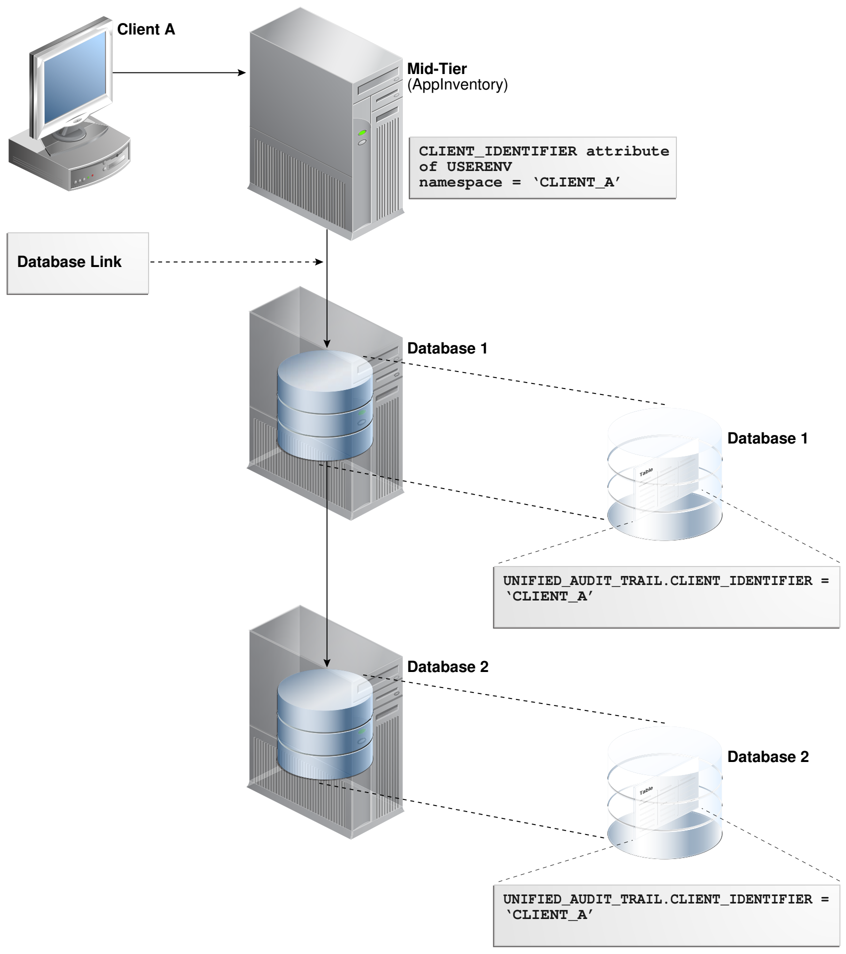


```
CLIENT_IDENTIFIER attribute
of USERENV
namespace = 'CLIENT_A'
```

```
UNIFIED_AUDIT_TRAIL.DBPROXY_USERNAME = 'apphr' (Proxy user for jackson)
UNIFIED_AUDIT_TRAIL.PROXY_SESSIONID = 3205062574 (ID for the apphr proxy user login)
UNIFIED_AUDIT_TRAIL.ACTION_NAME = 'CREATE TABLE' (User jackson's actions)
```

Figure 24-2 illustrates how you can audit client identifier information across multiple database sessions by querying the `CLIENT_ID` column of the `DBA_AUDIT_TRAIL` data dictionary view. In this scenario, the client identifier has been set to `CLIENT_A`. As with the proxy user-database user scenario described in Figure 24-1, session pooling can be used.

Figure 24-2 Auditing Client Identifier Information Across Sessions


Client A
Mid-Tier
(AppInventory)
CLIENT_IDENTIFIER attribute of USERENV namespace = 'CLIENT_A'
Database Link
Database 1
Database 1
UNIFIED_AUDIT_TRAIL.CLIENT_IDENTIFIER = 'CLIENT_A'
Database 2
Database 2
UNIFIED_AUDIT_TRAIL.CLIENT_IDENTIFIER = 'CLIENT_A'

Creating a Condition for a Unified Audit Policy

You can use the `CREATE AUDIT POLICY` statement to create conditions for a unified audit policy.

About Conditions in Unified Audit Policies

You can create a unified audit policy that uses a `SYS_CONTEXT` namespace-attribute pair to specify a condition.

For example, this audit condition can apply to a specific user who may fulfil the audit condition, or a computer host where the audit condition is fulfilled.

If the audit condition is satisfied, then Oracle Database creates an audit record for the event. As part of the condition definition, you must specify whether the audited condition is evaluated per statement occurrence, session, or database instance.

Note:

Audit conditions can use both secure and insecure application contexts.

Configuring a Unified Audit Policy with a Condition

The `WHEN` clause in the `CREATE AUDIT POLICY` statement defines the condition in the audit policy.

- Use the following syntax to create a unified audit policy that uses a condition:

```
CREATE AUDIT POLICY policy_name
 action_privilege_role_audit_option
[WHEN function_operation_value_list_1 [[AND | OR]
function_operation_value_list_n]
 EVALUATE PER STATEMENT | SESSION | INSTANCE];
```

In this specification:

- *`action_privilege_role_audit_option`* refers to audit options for system actions, object actions, privileges, and roles.
- `WHEN` defines the condition. It has the following components:
  - *`function`* uses the following types of functions:

    Numeric functions, such as `BITAND`, `CEIL`, `FLOOR`, and `LN POWER`

    Character functions that return character values, such as `CONCAT`, `LOWER`, and `UPPER`

    Character functions that return numeric values, such as `LENGTH` or `INSTR`

    Environment and identifier functions, such as `SYS_CONTEXT` and `UID`. For `SYS_CONTEXT`, in most cases, you may want to use the `USERENV` namespace, which is described in *Oracle Database SQL Language Reference*.
  - *`operation`* can be any the following operators: `AND`, `OR`, `IN`, `NOT IN`, =, <, >, <>
  - *`value_list`* refers to the condition for which you are testing.

  You can include additional conditions for each *`function_operation_value_list`* set, separated by `AND` or `OR`.

  When you write the `WHEN` clause, follow these guidelines:

  - Enclose the entire *`function operation value`* setting in single quotation marks. Within the clause, enclose each quoted component within two pairs of single quotation marks. Do not use double quotation marks.
  - Do not exceed 4000 bytes for the `WHEN` condition.
- `EVALUATE PER` refers to the following options:
  - `STATEMENT` evaluates the condition for each relevant auditable statement that occurs.
  - `SESSION` evaluates the condition only once during the session, and then caches and re-uses the result during the remainder of the session. Oracle Database evaluates the condition the first time the policy is used, and then stores the result in UGA memory afterward.

– `INSTANCE` evaluates the condition only once during the database instance lifetime. After Oracle Database evaluates the condition, it caches and re-uses the result for the remainder of the instance lifetime. As with the `SESSION` evaluation, the evaluation takes place the first time it is needed, and then the results are stored in UGA memory afterward.

For example:

```
CREATE AUDIT POLICY oe_orders_pol
 ACTIONS UPDATE ON OE.ORDERS
 WHEN 'SYS_CONTEXT(''USERENV'', ''IDENTIFICATION_TYPE'') = ''EXTERNAL'''
 EVALUATE PER STATEMENT;
```

Remember that after you create the policy, you must use the `AUDIT` statement to enable it.

See Also:

Oracle Database SQL Language Reference for more information about functions that you can use in conditions

Example: Auditing Access to SQL*Plus

The `CREATE AUDIT POLICY` statement can audit access to SQL*Plus.

Example 24-11 shows how to audit access to the database with SQL*Plus by users who have been directly granted the roles `emp_admin` and `sales_admin`.

Example 24-11   Auditing Access to SQL*Plus

```
CREATE AUDIT POLICY logon_pol
 ACTIONS LOGON
 WHEN 'INSTR(UPPER(SYS_CONTEXT(''USERENV'', ''CLIENT_PROGRAM_NAME'')), ''SQLPLUS'')
> 0'
 EVALUATE PER SESSION;

AUDIT POLICY logon_pol BY USERS WITH GRANTED ROLES emp_admin, sales_admin;
```

Example: Auditing Actions Not in Specific Hosts

The `CREATE AUDIT POLICY` statement can audit actions that are not in specific hosts.

Example 24-12 shows how to audit two actions (`UPDATE` and `DELETE` statements) on the `OE.ORDERS` table, but excludes the host names `sales_24` and `sales_12` from the audit. It performs the audit on a per session basis and writes audit records for failed attempts only.

Example 24-12   Auditing Actions Not in Specific Hosts

```
CREATE AUDIT POLICY oe_table_audit1
 ACTIONS UPDATE ON OE.ORDERS, DELETE ON OE.ORDERS
 WHEN 'SYS_CONTEXT (''USERENV'', ''HOST'') NOT IN (''sales_24'',''sales_12'')'
 EVALUATE PER SESSION;

AUDIT POLICY oe_table_audit1 WHENEVER NOT SUCCESSFUL;
```

Example: Auditing Both a System-Wide and a Schema-Specific Action

The CREATE AUDIT POLICY statement can audit both system-wide and schema-specific actions.

Example 24-13 shows a variation of Example 24-12 in which the UPDATE statement is audited system wide. The DELETE statement audit is still specific to the OE.ORDERS table.

Example 24-13 Auditing Both a System-Wide and a Schema-Specific Action

```
CREATE AUDIT POLICY oe_table_audit2
 ACTIONS UPDATE, DELETE ON OE.ORDERS
 WHEN 'SYS_CONTEXT (''USERENV'', ''HOST'') NOT IN (''sales_24'',''sales_12'')'
 EVALUATE PER SESSION;

AUDIT POLICY oe_table_audit2;
```

Example: Auditing a Condition Per Statement Occurrence

The CREATE AUDIT POLICY statement can audit conditions.

Example 24-14 shows how to audit a condition based on each occurrence of the DELETE statement on the OE.ORDERS table and exclude user jmartin from the audit.

Example 24-14 Auditing a Condition Per Statement Occurrence

```
CREATE AUDIT POLICY sales_clerk_pol
 ACTIONS DELETE ON OE.ORDERS
 WHEN 'SYS_CONTEXT(''USERENV'', ''CLIENT_IDENTIFIER'') = ''sales_clerk'''
 EVALUATE PER STATEMENT;

AUDIT POLICY sales_clerk_pol EXCEPT jmartin;
```

Example: Unified Audit Session ID of a Current Administrative User Session

The SYS_CONTEXT function can be used to find session IDs.

Example 24-15 shows how to find the unified audit session ID of current user session for an administrative user.

Example 24-15 Unified Audit Session ID of a Current Administrative User Session

```
CONNECT SYS AS SYSDBA
Enter password: password

SELECT SYS_CONTEXT('USERENV', 'UNIFIED_AUDIT_SESSIONID') FROM DUAL;
```

Output similar to the following appears:

```
SYS_CONTEXT('USERENV','UNIFIED_AUDIT_SESSIONID')
--------------------------------------------------------------------------------
2318470183
```

Note that in mixed mode auditing, the UNIFIED_AUDIT_SESSIONID value in the USERENV namespace is different from the value that is recorded by the SESSIONID parameter. Hence, if you are using mixed mode auditing and want to find the correct audit session ID, you should use the USERENV UNIFIED_AUDIT_SESSIONID parameter, not the SESSIONID parameter. In pure unified auditing, the `SESSIONID` and `UNIFIED_AUDIT_SESSIONID` values are the same.

Example: Unified Audit Session ID of a Current Non-Administrative User Session

The `SYS_CONTEXT` function can find the session ID of a current non-administrative user session.

Example 24-16 shows how to find the unified audit session ID of a current user session for a non-administrative user.

Example 24-16   Unified Audit Session ID of a Current Non-Administrative User Session

```
CONNECT mblake -- Or, CONNECT mblake@hrpdb for a PDB
Enter password: password

SELECT SYS_CONTEXT('USERENV', 'UNIFIED_AUDIT_SESSIONID') FROM DUAL;
```

Output similar to the following appears:

```
SYS_CONTEXT('USERENV','UNIFIED_AUDIT_SESSIONID')
--------------------------------------------------------------------------------
2776921346
```

How Audit Records from Conditions Appear in the Audit Trail

The audit record conditions from a unified audit policy do not appear in the audit trail.

If the condition evaluates to true and the record is written, then the record appears in the audit trail. You can check the audit trail by querying the `UNIFIED_AUDIT_TRAIL` data dictionary view.

Auditing Application Context Values

You can use the `AUDIT` statement to audit application context values.

About Auditing Application Context Values

You can capture application context values in the unified audit trail.

This feature enables you to capture any application context values set by the database applications, while executing the audited statement.

If you plan to audit Oracle Label Security, then this feature captures session label activity for the database audit trail. The audit trail records all the values retrieved for the specified context-attribute value pairs.

The application context audit setting or the audit policy have session static semantics. In other words, if a new policy is enabled for a user, then the subsequent user sessions will see an effect of this command. After the session is established, then the policies and contexts settings are loaded and the subsequent `AUDIT` statements have no effect on that session.

For multitenant environments, the application context audit policy applies only to the current PDB.

See Also:

- Using Application Contexts to Retrieve User Information, for detailed information about application contexts
- Unified Audit Policies or AUDIT Settings in a Multitenant Environment
- *Oracle Label Security Administrator's Guide* for detailed information about Oracle Label Security

Configuring Application Context Audit Settings

The `AUDIT` statement with the `CONTEXT` keyword configures auditing for application context values.

You do not create an unified audit policy for this type of auditing.

- Use the following syntax to configure auditing for application context values:

```
AUDIT CONTEXT NAMESPACE context_name1 ATTRIBUTES attribute1 [, attribute2]
 [, CONTEXT NAMESPACE context_name2 ATTRIBUTES attribute1 [, attribute2]]
 [BY user_list];
```

In this specification:

- *`context_name1`*: Optionally, you can include one additional `CONTEXT` name-attribute value pair.
- *`user_list`* is an optional list of database user accounts. Separate multiple names with a comma. If you omit this setting, then Oracle Database configures the application context policy for all users. When each user logs in, a list of all pertinent application contexts and their attributes is cached for the user session.

For example:

```
AUDIT CONTEXT NAMESPACE clientcontext3 ATTRIBUTES module, action,
 CONTEXT NAMESPACE ols_session_labels ATTRIBUTES ols_pol1, ols_pol3
 BY appuser1, appuser2;
```

To find a list of currently configured application context audit settings, query the `AUDIT_UNIFIED_CONTEXTS` data dictionary view.

Disabling Application Context Audit Settings

The `NOAUDIT` statement disables application context audit settings.

- To disable an application context audit setting, specify the namespace and attribute settings in the `NOAUDIT` statement. You can enter the attributes in any order (that is, they do not need to match the order used in the corresponding `AUDIT CONTEXT` statement.)

For example:

```
NOAUDIT CONTEXT NAMESPACE client_context ATTRIBUTES module,
 CONTEXT NAMESPACE ols_session_labels ATTRIBUTES ols_pol1, ols_pol3
 BY USERS WITH GRANTED ROLES emp_admin;
```

To find the currently audited application contexts, query the AUDIT_UNIFIED_CONTEXTS data dictionary view.

Example: Auditing Application Context Values in a Default Database

The AUDIT CONTEXT NAMESPACE statement can audit application context values.

Example 24-17 shows how to audit the clientcontext application values for the module and action attributes, by the user appuser1.

Example 24-17   Auditing Application Context Values in a Default Database

```
AUDIT CONTEXT NAMESPACE clientcontext ATTRIBUTES module, action
BY appuser1;
```

Example: Auditing Application Context Values from Oracle Label Security

The AUDIT CONTEXT NAMESPACE statement can audit application context values from Oracle Label Security.

Example 24-18 shows how to audit an application context for Oracle Label Security called ols_session_labels, for the attributes ols_pol1 and ols_pol2.

Example 24-18   Auditing Application Context Values from Oracle Label Security

```
AUDIT CONTEXT NAMESPACE ols_session_labels ATTRIBUTES ols_pol1, ols_pol2;
```

How Audited Application Contexts Appear in the Audit Trail

The UNIFIED_AUDIT_POLICIES data dictionary view lists application context audit events.

The APPLICATION_CONTEXTS column of the UNIFIED_AUDIT_TRAIL data dictionary view shows application context audit data. The application contexts appear as a list of semi-colon separated values.

For example:

```
SELECT APPLICATION_CONTEXTS FROM UNIFIED_AUDIT_TRAIL
 WHERE UNIFIED_AUDIT_POLICIES = 'app_audit_pol';

APPLICATION_CONTEXTS
---------------------------------------------------------
CLIENT_CONTEXT.APPROLE=MANAGER;E2E_CONTEXT.USERNAME=PSMITH
```

Auditing Oracle Database Real Application Security Events

You can use CREATE AUDIT POLICY statement to audit Oracle Database Real Application Security events.

About Auditing Oracle Database Real Application Security Events

You must have the AUDIT_ADMIN role to audit Oracle Database Real Application Security events.

To access the audit trail, you can query the UNIFIED_AUDIT_TRAIL data dictionary view, whose Real Application Security-specific columns begin with XS_. If you want to find audit information about the internally generated VPD predicate that is created while an

Oracle Real Application Security policy is being enabled, then you can query the `RLS_INFO` column.

Real Application Security-specific views are as follows:

- `DBA_XS_AUDIT_TRAIL` provides detailed information about Real Application Security events that were audited.
- `DBA_XS_AUDIT_POLICY_OPTIONS` describes the auditing options that were defined for Real Application Security unified audit policies.
- `DBA_XS_ENB_AUDIT_POLICIES` lists users for whom Real Application Security unified audit polices are enabled.

See Also:

- Auditing Application Context Values

  Oracle Database Real Application Security Predfined Audit Policies
- Auditing of Oracle Virtual Private Database Predicates for information about how to format the output of the `RLS_INFO` column
- *Oracle Database Real Application Security Administrator's and Developer's Guide* for detailed information about Oracle Database Real Application Security

Oracle Database Real Application Security Auditable Events

Oracle Database provides Real Application Security events that you can audit, such `CREATE USER`, `UPDATE USER`.

To find a list of auditable Real Application Security events that you can audit, you can query the `COMPONENT` and `NAME` columns of the `AUDITABLE_SYSTEM_ACTIONS` data dictionary view, as follows:

```
SELECT NAME FROM AUDITABLE_SYSTEM_ACTIONS WHERE COMPONENT = 'XS';

NAME
-------------
CREATE USER
UPDATE USER
DELETE USER
...
```

Oracle Database Real Application Security User, Privilege, and Role Audit Events

The unified audit trail can capture Oracle Database Real Application Security events for users, privileges, and roles.

Table 24-4 describes these events.

Table 24-4 Oracle Database Real Application Security User, Privilege, and Role Audit Events

Audit Event	Description
CREATE USER	Creates an Oracle Database Real Application Security user account through the XS_PRINCIPAL.CREATE_USER procedure
UPDATE USER	Updates an Oracle Database Real Application Security user account through the following procedures: • XS_PRINCIPAL.SET_EFFECTIVE_DATES • XS_PRINCIPAL.SET_USER_DEFAULT_ROLES_ALL • XS_PRINCIPAL.SET_USER_SCHEMA • XS_PRINCIPAL.SET_GUID • XS_PRINCIPAL.SET_USER_STATUS • XS_PRINCIPAL.SET_DESCRIPTION
DELETE USER	Deletes an Oracle Database Real Application Security user account through the through the XS_PRINCIPAL.DELETE_PRINCIPAL procedure
AUDIT_GRANT_PRIVILEGE	Audits the GRANT_SYSTEM_PRIVILEGE privilege
AUDIT_REVOKE_PRIVILEGE	Audits the REVOKE_SYSTEM_PRIVILEGE privilege
CREATE ROLE	Creates an Oracle Database Real Application Security role through the XS_PRINCIPAL.CREATE_ROLE procedure
UPDATE ROLE	Updates an Oracle Database Real Application Security role through the following procedures: • XS_PRINCIPAL.SET_DYNAMIC_ROLE_SCOPE • XS_PRINCIPAL.SET_DYNAMIC_ROLE_DURATION • XS_PRINCIPAL.SET_EFFECTIVE_DATES • XS_PRINCIPAL.SET_ROLE_DEFAULT
DELETE ROLE	Deletes an Oracle Database Real Application Security role through the XS_PRINCIPAL.DELETE_ROLE procedure
GRANT ROLE	Grants Oracle Database Real Application Security roles through the XS_PRINCIPAL.GRANT_ROLES procedure
REVOKE ROLE	Revokes Oracle Database Real Application Security roles through the XS_PRINCIPAL.REVOKE_ROLES procedure and revokes all granted roles through the XS_PRINCIPAL.REVOKE_ALL_GRANTED_ROLES procedure
ADD PROXY	Adds Oracle Database Real Application Security proxy user account through the XS_PRINCIPAL.ADD_PROXY_USER procedure, and adds proxies to database users through the XS_PRINCIPAL.ADD_PROXY_TO_SCHEMA procedure
REMOVE PROXY	Removes an Oracle Database Real Application Security proxy user account through the XS_PRINCIPAL.REMOVE_PROXY_USER, XS_PRINCIPAL.REMOVE_ALL_PROXY_USERS, and XS_PRINCIPAL.REMOVE_PROXY_FROM_SCHEMA PROCEDURES
SET USER PASSWORD	Sets the Oracle Database Real Application Security user account password through the XS_PRINCIPAL.SET_PASSWORD procedure
SET USER VERIFIER	Sets the Oracle Database Real Application Security proxy user account verifier through the XS_PRINCIPAL.SET_VERIFIER procedure

Oracle Database Real Application Security Security Class and ACL Audit Events

The unified audit trail can capture Oracle Database Real Application Security security class and ACL audit events.

Table 24-5 describes these events.

Table 24-5 Oracle Database Real Application Security Security Class and ACL Audit Events

Audit Event	Description
`CREATE SECURITY CLASS`	Creates a security class through the `XS_SECURITY_CLASS.CREATE_SECURITY_CLASS` procedure
`UPDATE SECURITY CLASS`	Creates a security class through the following procedures: • `XS_SECURITY_CLASS.SET_DEFAULT_ACL` • `XS_SECURITY_CLASS.ADD_PARENTS` • `XS_SECURITY_CLASS.REMOVE_ALL_PARENTS` • `XS_SECURITY_CLASS.REMOVE_PARENTS` • `XS_SECURITY_CLASS.ADD_PRIVILEGES` • `XS_SECURITY_CLASS.REMOVE_ALL_PRIVILEGES` • `XS_SECURITY_CLASS.ADD_IMPLIED_PRIVILEGES` • `XS_SECURITY_CLASS.REMOVE_IMPLIED_PRIVILEGES` • `XS_SECURITY_CLASS.REMOVE_ALL_IMPLIED_PRIVILEGES` • `XS_SECURITY_CLASS.SET_DESCRIPTION`
`DELETE SECURITY CLASS`	Deletes a security class through the `XS_SECURITY_CLASS.DELETE_SECURITY_CLASS` procedure
`CREATE ACL`	Creates an Access Control List (ACL) through the `XS_ACL.CREATE_ACL` procedure
`UPDATE ACL`	Updates an ACL through the following procedures: • `XS_ACL.APPEND_ACES` • `XS_ACL.REMOVE_ALL_ACES` • `XS_ACL.SET_SECURITY_CLASS` • `XS_ACL.SET_PARENT_ACL` • `XS_ACL.ADD_ACL_PARAMETER` • `XS_ACL.REMOVE_ALL_ACL_PARAMETERS` • `XS_ACL.REMOVE_ACL_PARAMETER` • `XS_ACL.SET_DESCRIPTION`
`DELETE ACL`	Deletes an ACL through the `XS_ACL.DELETE_ACL` procedure
`CREATE DATA SECURITY-`	Creates a data security policy through the `XS_DATA_SECURITY.CREATE_DATA_SECURITY` procedure
`UPDATE DATA SECURITY`	Updates a data security policy through the following procedures: • `XS_DATA_SECURITY.CREATE_ACL_PARAMETER` • `XS_DATA_SECURITY.DELETE_ACL_PARAMETER` • `XS_DATA_SECURITY.SET_DESCRIPTION`
`DELETE DATA SECURITY`	Deletes a data security policy through the `XS_DATA_SECURITY.DELETE_DATA_SECURITY` procedure

Table 24-5 (Cont.) Oracle Database Real Application Security Security Class and ACL Audit Events

Audit Event	Description
`ENABLE DATA SECURITY`	Enables extensible data security for a database table or view through the `XS_DATA_SECURITY.ENABLE_OBJECT_POLICY` procedure
`DISABLE DATA SECURITY`	Disables extensible data security for a database table or view through the `XS_DATA_SECURITY.DISABLE_XDS` procedure

Oracle Database Real Application Security Session Audit Events

The unified audit trail can capture Oracle Database Real Application Security session audit events.

Table 24-4 describes these events.

Table 24-6 Oracle Database Real Application Security Session Audit Events

Audit Event	Description
`CREATE SESSION`	Creates a session through the `DBMS_XS_SESSIONS.CREATE_SESSION` procedure
`DESTROY SESSION`	Destroys a session through the `DBMS_XS_SESSIONS.DESTROY_SESSION` procedure
`CREATE SESSION NAMESPACE`	Creates a namespace through the `DBMS_XS_SESSIONS.CREATE_NAMESPACE` procedure
`DELETE SESSION NAMESPACE`	Deletes a namespace through the `DBMS_XS_SESSIONS.DELETE_NAMESPACE` procedure
`CREATE NAMESPACE ATTRIBUTE`	Creates a namespace attribute through the `DBMS_XS_SESSIONS.CREATE_ATTRIBUTE` procedure
`SET NAMESPACE ATTRIBUTE`	Sets a namespace attribute through the `DBMS_XS_SESSIONS.SET_ATTRIBUTE` procedure
`GET NAMESPACE ATTRIBUTE`	Gets a namespace attribute through the `DBMS_XS_SESSIONS.GET_ATTRIBUTE` procedure
`DELETE NAMESPACE ATTRIBUTE`	Deletes a namespace attribute through the `DBMS_XS_SESSIONS.DELETE_ATTRIBUTE` procedure
`CREATE NAMESPACE TEMPLATE`	Creates a namespace attribute through the `XS_NS_TEMPLATE.CREATE_NS_TEMPLATE` procedure
`UPDATE NAMESPACE TEMPLATE`	Updates a namespace attribute through the following procedures: • `XS_NS_TEMPLATE.SET_HANDLER` • `XS_NS_TEMPLATE.ADD_ATTRIBUTES` • `XS_NS_TEMPLATE.REMOVE_ALL_ATTRIBUTES` • `XS_NS_TEMPLATE.REMOVE_ATTRIBUTES` • `XS_NS_TEMPLATE.SET_DESCRIPTION`
`DELETE NAMESPACE TEMPLATE`	Deletes a namespace through the `XS_NS_TEMPLATE.DELETE_NS_TEMPLATE` procedure
`ADD GLOBAL CALLBACK`	Adds a global callback through the `DBMS_XS_SESSIONS.ADD_GLOBAL_CALLBACK` procedure

Table 24-6 (Cont.) Oracle Database Real Application Security Session Audit Events

Audit Event	Description
`DELETE GLOBAL CALLBACK`	Deletes a global callback through the `DBMS_XS_SESSIONS.DELETE_GLOBAL_CALLBACK` procedure
`ENABLE GLOBAL CALLBACK`	Enables a global callback through the `DBMS_XS_SESSIONS.ENABLE_GLOBAL_CALLBACK` procedure
`SET COOKIE`	Sets a session cookie through the `DBMS_XS_SESSIONS.SET_SESSION_COOKIE` procedure
`SET INACTIVE TIMEOUT`	Sets the time-out time for inactive sessions through the `DBMS_XS_SESSIONS.SET_INACTIVITY_TIMEOUT` procedure
`SWITCH USER`	Sets the security context of the current lightweight user session to a newly initialized security context for a specified user through the `DBMS_XS_SESSIONS.SWITCH_USER` procedure
`ASSIGN USER`	Assigns or removes one or more dynamic roles for the specified user through the `DBMS_XS_SESSIONS.ASSIGN_USER` procedure
`ENABLE ROLE`	Enable a role for a lightweight user session through the `DBMS_XS_SESSIONS.ENABLE_ROLE` procedure
`DISABLE ROLE`	Disables a role for a lightweight user session through the `DBMS_XS_SESSIONS.DISABLE_ROLE` procedure

Oracle Database Real Application Security ALL Events

The unified audit trail can capture Oracle Database Real Application Security `ALL` events.

Table 24-7 describes these events.

Table 24-7 Oracle Database Real Application Security ALL Events

Audit Event	Description
`ALL`	Captures all Real Application Security actions

Configuring a Unified Audit Policy for Oracle Database Real Application Security

The `CREATE AUDIT POLICY` statement can create a unified audit policy for Oracle Real Application Security.

- Use the following syntax to create a unified audit policy for Oracle Database Real Application Security:

  ```
  CREATE AUDIT POLICY policy_name
   ACTIONS COMPONENT=XS component_action1 [, action2];
  ```

For example:

```
CREATE AUDIT POLICY audit_ras_pol
 ACTIONS COMPONENT=XS SWITCH USER, DISABLE ROLE;
```

You can build more complex policies, such as those that include conditions. Remember that after you create the policy, you must use the `AUDIT` statement to enable it.

Example: Auditing Real Application Security User Account Modifications

The `CREATE AUDIT POLICY` statement can audit Real Application Security user account modifications.

Example 24-19 shows how to audit user `bhurst`'s attempts to switch users and disable roles.

Example 24-19 Auditing Real Application Security User Account Modifications

```
CREATE AUDIT POLICY ras_users_pol
 ACTIONS COMPONENT=XS SWITCH USER, DISABLE ROLE;

AUDIT POLICY ras_users_pol BY bhurst;
```

Example: Using a Condition in a Real Application Security Unified Audit Policy

The `CREATE AUDIT POLICY` statement can set a condition for a Real Application Security unified audit policy.

Example 24-20 shows how to create Real Application Security unified audit policy that applies the audit only to actions from the `nemosity` computer host.

Example 24-20 Using a Condition in a Real Application Security Unified Audit Policy

```
CREATE AUDIT POLICY ras_acl_pol
 ACTIONS DELETE ON OE.CUSTOMERS
 ACTIONS COMPONENT=XS CREATE ACL, UPDATE ACL, DELETE ACL
 WHEN 'SYS_CONTEXT(''USERENV'', ''HOST'') = ''nemosity'''
 EVALUATE PER INSTANCE;

AUDIT POLICY ras_acl_pol BY pfitch;
```

How Oracle Database Real Application Security Events Appear in the Audit Trail

The `DBA_XS_AUDIT_TRAIL` data dictionary view lists Oracle Real Application Security audit events.

The following example queries the Real Application Security-specific view, `DBA_XS_AUDIT_TRAIL`:

```
SELECT XS_USER_NAME FROM DBA_XS_AUDIT_TRAIL
WHERE XS_ENABLED_ROLE = 'CLERK';

XS_USER_NAME
-------------
USER2
```

Auditing Oracle Recovery Manager Events

You can use the `CREATE AUDIT POLICY` statement to audit Oracle Recovery Manager events.

About Auditing Oracle Recovery Manager Events

The `UNIFIED_AUDIT_TRAIL` data dictionary view automatically stores Oracle Recovery Manager audit events in the `RMAN_`column.

Unlike other Oracle Database components, you do not create a unified audit policy for Oracle Recovery Manager events.

However, you must have the `AUDIT_ADMIN` or `AUDIT_VIEWER` role in order to query the `UNIFIED_AUDIT_TRAIL` view to see these events. If you have the `SYSBACKUP` or the `SYSDBA` administrative privilege, then you can find additional information about Recovery Manager jobs by querying views such as `V$RMAN_STATUS` or `V$RMAN_BACKUP_JOB_DETAILS`.

See Also:

Oracle Database Backup and Recovery User's Guide

Oracle Recovery Manager Unified Audit Trail Events

The unified audit trail can capture Oracle Recovery Manager events.

Table 24-8 describes these events.

Table 24-8 Oracle Recovery Manager Columns in UNIFIED_AUDIT_TRAIL View

Recovery Manager Column	Description
`RMAN_SESSION_RECID`	**Recovery Manager session identifier.** Together with the `RMAN_SESSION_STAMP` column, this column uniquely identifies the Recovery Manager job. The Recovery Manager session ID is a a `RECID` value in the control file that identifies the Recovery Manager job. (Note that the Recovery Manager session ID is not the same as a user session ID.)
`RMAN_SESSION_STAMP`	**Timestamp for the session.** Together with the `RMAN_SESSION_RECID` column, this column identifies Recovery Manager jobs.
`RMAN_OPERATION`	**The Recovery Manager operation executed by the job.** One row is added for each distinct operation within a Recovery Manager session. For example, a backup job contains `BACKUP` as the `RMAN_OPERATION` value.

Table 24-8 (Cont.) Oracle Recovery Manager Columns in UNIFIED_AUDIT_TRAIL View

Recovery Manager Column	Description
`RMAN_OBJECT_TYPE`	**Type of objects involved in a Recovery Manager session.** It contains one of the following values. If the Recovery Manager session does not satisfy more than one of them, then preference is given in the following order, from top to bottom of the list. 1. `DB FULL` (Database Full) refers to a full backup of the database 2. `RECVR AREA` refers to the Fast Recovery area 3. `DB INCR` (Database Incremental) refers to incremental backups of the database 4. `DATAFILE FULL` refers to a full backup of the data files 5. `DATAFILE INCR` refers to incremental backups of the data files 6. `ARCHIVELOG` refers to archived redo log files 7. `CONTROLFILE` refers to control files 8. `SPFILE` refers to the server parameter file 9. `BACKUPSET` refers to backup files
`RMAN_DEVICE_TYPE`	**Device associated with a Recovery Manager session.** This column can be `DISK`, `SBT` (system backup tape), or `*` (asterisk). An asterisk indicates more than one device. In most cases, the value will be `DISK` and `SBT`.

How Oracle Recovery Manager Audited Events Appear in the Audit Trail

The `UNIFIED_AUDIT_TRAIL` data dictionary view lists Oracle Recovery Manager audit events.

Table 24-8 lists the columns in the `UNIFIED_AUDIT_TRAIL` data dictionary view that you can query to find Oracle Recovery Manager-specific audit data.

For example:

```
SELECT RMAN_OPERATION FROM UNIFIED_AUDIT_TRAIL
WHERE RMAN_OBJECT_TYPE = 'DB FULL';

RMAN_OPERATION
---------------
BACKUP
```

Auditing Oracle Database Vault Events

In an Oracle Database Vault environment, the `CREATE AUDIT POLICY` statement can audit Database Vault activities.

About Auditing Oracle Database Vault Events

As with all unified auditing, you must have the `AUDIT_ADMIN` role before you can audit Oracle Database Vault events.

To create Oracle Database Vault unified audit policies, you must set the `CREATE AUDIT POLICY` statement's `COMPONENT` clause to `DV`, and then specify an action, such as `Rule Set Failure`, and an object, such as the name of a rule set.

To access the audit trail, you can query the following views:

- `UNIFIED_AUDIT_TRAIL`
- `AUDSYS.DV$CONFIGURATION_AUDIT`
- `AUDSYS.DV$ENFORCEMENT_AUDIT`

In the `UNIFIED_AUDIT_TRAIL` view, the Oracle Database Vault-specific columns begin with `DV_`. You must have the `AUDIT_VIEWER` role before you can query the `UNIFIED_AUDIT_TRAIL` view.

In addition to these views, the Database Vault reports capture the results of Database Vault-specific unified audit policies.

See Also:

- Oracle Database Vault Predefined Unified Audit Policy for DVSYS and LBACSYS Schemas
- *Oracle Database Vault Administrator's Guide* for detailed information about Oracle Database Vault audit policies

Who Is Audited in Oracle Database Vault?

Audited Oracle Database Vault users include administrators and users whose activities affect Database Vault enforcement policies.

These users are as follows:

- **Database Vault administrators.** All configuration changes that are made to Oracle Database Vault are mandatorily audited. The auditing captures activities such as creating, modifying, or deleting realms, factors, command rules, rule sets, rules, and so on. The `AUDSYS.DV$CONFIGURATION_AUDIT` data dictionary view captures configuration changes made by Database Vault administrators.
- **Users whose activities affect Oracle Database Vault enforcement policies.** The `AUDSYS.DV$ENFORCEMENT_AUDIT` data dictionary view captures enforcement-related audits

See Also:

Oracle Database Vault Administrator’s Guide for more information about the `AUDSYS.DV$CONFIGURATION_AUDIT` and `AUDSYS.DV$ENFORCEMENT_AUDIT` data dictionary views

About Oracle Database Vault Unified Audit Trail Events

The audit trail in an Oracle Database Vault environment captures all configuration changes or attempts at changes to Database Vault policies.

It also captures violations by users to existing Database Vault policies.

You can audit the following kinds of Oracle Database Vault events:

- **All configuration changes or attempts at changes to Oracle Database Vault policies.** It captures both Database Vault administrator changes and attempts made by unauthorized users.
- **Violations by users to existing Database Vault policies.** For example, if you create a policy to prevent users from accessing a specific schema table during non-work hours, the audit trail will capture this activity.

Oracle Database Vault Realm Audit Events

The unified audit trail captures Oracle Database Vault realm events.

Table 24-9 describes these events.

Table 24-9 Oracle Database Vault Realm Audit Events

Audit Event	Description
`CREATE_REALM`	Creates a realm through the `DVSYS.DBMS_MACADM.CREATE_REALM` procedure
`UPDATE_REALM`	Updates a realm through the `DVSYS.DBMS_MACADM.UPDATE_REALM` procedure
`RENAME_REALM`	Renames a realm through the `DVSYS.DBMS_MACADM.RENAME_REALM` procedure
`DELETE_REALM`	Deletes a realm through the `DVSYS.DBMS_MACADM.DELETE_REALM` procedure
`DELETE_REALM_CASCADE`	Deletes a realm and its related Database Vault configuration information through the `DVSYS.DBMS_MACADM.DELETE_REALM_CASCADE` procedure
`ADD_AUTH_TO_REALM`	Adds an authorization to the realm through the `DVSYS.DBMS_MACADM.ADD_AUTH_TO_REALM` procedure
`DELETE_AUTH_FROM_REALM`	Removes an authorization from the realm through the `DVSYS.DBMS_MACADM.DELETE_AUTH_FROM_REALM` procedure

Table 24-9 (Cont.) Oracle Database Vault Realm Audit Events

Audit Event	Description
`UPDATE_REALM_AUTH`	Updates a realm authorization through the `DVSYS.DBMS_MACADM.UPDATE_REALM_AUTHORIZATION` procedure
`ADD_OBJECT_TO_REALM`	Adds an object to a realm authorization through the `DVSYS.DBMS_MACADM.ADD_AUTH_TO_REALM` procedure
`DELETE_OBJECT_FROM_REALM`	Removes an object from a realm authorization through the `DVSYS.DBMS_MACADM.DELETE_OBJECT_FROM_REALM` procedure

Oracle Database Vault Rule Set and Rule Audit Events

The unified audit trail can capture Oracle Database Vault rule set and rule audit events.

Table 24-10 describes these events.

Table 24-10 Oracle Database Vault Rule Set and Rule Audit Events

Audit Event	Description
`CREATE_RULE_SET`	Creates a rule set through the `DVSYS.DBMS_MACADM.CREATE_RULE_SET` procedure
`UPDATE_RULE_SET`	Updates a rule set through the `DVSYS.DBMS_MACADM.UPDATE_RULE_SET` procedure
`RENAME_RULE_SET`	Renames a rule set through the `DVSYS.DBMS_MACADM.RENAME_RULE_SET` procedure
`DELETE_RULE_SET`	Deletes a rule set through the `DVSYS.DBMS_MACADM.DELETE_RULE_SET` procedure
`ADD_RULE_TO_RULE_SET`	Adds a rule to an existing rule set through the `DVSYS.DBMS_MACADM.ADD_RULE_TO_RULE_SET` procedure
`DELETE_RULE_FROM_RULE_SET`	Removes a rule from an existing rule set through the `DVSYS.DBMS_MACADM.DELETE_RULE_FROM_RULE_SET` procedure
`CREATE_RULE`	Creates a rule through the `DVSYS.DBMS_MACADM.CREATE_RULE` procedure
`UPDATE_RULE`	Updates a rule through the `DVSYS.DBMS_MACADM.UPDATE_RULE` procedure
`RENAME_RULE`	Renames a rule through the `DVSYS.DBMS_MACADM.RENAME_RULE` procedure
`DELETE_RULE`	Deletes a rule through the `DVSYS.DBMS_MACADM.DELETE_RULE` procedure
`SYNC_RULES`	Synchronizes the rules in Oracle Database Vault and Advanced Queuing Rules engine through the `DVSYS.DBMS_MACADM.SYNC_RULES` procedure

Oracle Database Vault Command Rule Audit Events

The unified audit trail can capture Oracle Database Vault command rule audit events.

Table 24-11 describes these events.

Table 24-11 Oracle Database Vault Command Rule Audit Events

Audit Event	Description
`CREATE_COMMAND_RULE`	Creates a command rule through the `DVSYS.DBMS_MACADM.CREATE_COMMAND_RULE` procedure
`DELETE_COMMAND_RULE`	Deletes a command rule through the `DVSYS.DBMS_MACADM.DELETE_COMMAND_RULE` procedure
`UPDATE_COMMAND_RULE`	Updates a command rule through the `DVSYS.DBMS_MACADM.UPDATE_COMMAND_RULE` procedure

Oracle Database Vault Factor Audit Events

The unified audit trail can capture Oracle Database Vault factor events.

Table 24-12 describes these events.

Table 24-12 Oracle Database Vault Factor Audit Events

Audit Event	Description
`CREATE_FACTOR_TYPE`	Creates a factor type through the `DVSYS.DBMS_MACADM.CREATE_FACTOR_TYPE` procedure
`DELETE_FACTOR_TYPE`	Deletes a factor type through the `DVSYS.DBMS_MACADM.DELETE_FACTOR_TYPE` procedure
`UPDATE_FACTOR_TYPE`	Updates a factor type through the `DVSYS.DBMS_MACADM.UPDATE_FACTOR_TYPE` procedure
`RENAME_FACTOR_TYPE`	Renames a factor type through the `DVSYS.DBMS_MACADM.RENAME_FACTOR_TYPE` procedure
`CREATE_FACTOR`	Creates a factor through the `DVSYS.DBMS_MACADM.CREATE_FACTOR` procedure
`UPDATE_FACTOR`	Updates a factor through the `DVSYS.DBMS_MACADM.UPDATE_FACTOR` procedure
`DELETE_FACTOR`	Deletes a factor through the `DVSYS.DBMS_MACADM.DELETE_FACTOR` procedure
`RENAME_FACTOR`	Renames a factor through the `DVSYS.DBMS_MACADM.RENAME_FACTOR` procedure

Table 24-12 (Cont.) Oracle Database Vault Factor Audit Events

Audit Event	Description
ADD_FACTOR_LINK	Specifies a parent-child relationship between two factors through the DVSYS.DBMS_MACADM.ADD_FACTOR_LINK procedure
DELETE_FACTOR_LINK	Removes the parent-child relationship between two factors through the DVSYS.DBMS_MACADM.DELETE_FACTOR_LINK procedure
ADD_POLICY_FACTOR	Specifies that the label for a factor contributes to the Oracle Label Security label for a policy, through the DVSYS.DBMS_MACADM.ADD_POLICY_FACTOR procedure
DELETE_POLICY_FACTOR	Removes factor label from being associated with an Oracle Label Security label for a policy, through the DBMS_MACADM.DELETE_POLICY_FACTOR procedure
CREATE_IDENTITY	Creates a factor identity through the DVSYS.DBMS_MACADM.CREATE_IDENTITY procedure
UPDATE_IDENTITY	Updates a factor identity through the DVSYS.DBMS_MACADM.UPDATE_IDENTITY procedure
CHANGE_IDENTITY_FACTOR	Associates an identity with a different factor through the DVSYS.DBMS_MACADM.CHANGE_IDENTITY_FACTOR procedure
CHANGE_IDENTITY_VALUE	Updates the value of an identity through the DVSYS.DBMS_MACADM.CHANGE_IDENTITY_VALUE procedure
DELETE_IDENTITY	Deletes an existing factor identity through the DVSYS.DBMS_MACADM.DELETE_IDENTITY procedure
CREATE_IDENTITY_MAP	Creates a factor identity map through the DVSYS.DBMS_MACADM.CREATE_IDENTITY_MAP procedure
DELETE_IDENTITY_MAP	Deletes a factor identity map through the DVSYS.DBMS_MACADM.DELETE_IDENTITY_MAP procedure
CREATE_DOMAIN_IDENTITY	Adds an Oracle Database Real Application Clusters database node to the domain factor identities and labels it according to the Oracle Label Security policy, through the DVSYS.DBMS_MACADM.CREATE_DOMAIN_IDENTITY procedure
DROP_DOMAIN_IDENTITY	Drops an Oracle RAC node from the domain factor identities through the DVSYS.DBMS_MACADM.DROP_DOMAIN_IDENTITY procedure

Oracle Database Vault Secure Application Role Audit Events

The unified audit trail can capture Oracle Database Vault secure application role audit events.

Table 24-13 describes these events.

Table 24-13   Oracle Database Vault Secure Application Role Audit Events

Audit Event	Description
`CREATE_ROLE`	Creates an Oracle Database Vault secure application role through the `DVSYS.DBMS_MACADM.CREATE_ROLE` procedure
`DELETE_ROLE`	Deletes an Oracle Database Vault secure application role through the `DVSYS.DBMS_MACADM.DELETE_ROLE` procedure
`UPDATE_ROLE`	Updates an Oracle Database Vault secure application role through the `DVSYS.DBMS_MACADM.UPDATE_ROLE` procedure
`RENAME_ROLE`	Renames an Oracle Database Vault secure application role through the `DVSYS.DBMS_MACADM.RENAME_ROLE` procedure

Oracle Database Vault Oracle Label Security Audit Events

The unified audit trail can capture Oracle Database Vault Oracle Label Security audit events.

Table 24-14 describes these events.

Table 24-14   Oracle Database Vault Oracle Label Security Audit Events

Audit Event	Description
`CREATE_POLICY_LABEL`	Creates an Oracle Label Security policy label through the `DVSYS.DBMS_MACADM.CREATE_POLICY_LABEL` procedure
`DELETE_POLICY_LABEL`	Deletes an Oracle Label Security policy label through the `DVSYS.DBMS_MACADM.DELETE_POLICY_LABEL` procedure
`CREATE_MAC_POLICY`	Specifies the algorithm that is used to merge labels when computing the label for a factor, or the Oracle Label Security Session label, through the `DVSYS.DBMS_MACADM.CREATE_MAC_POLICY` procedure
`UPDATE_MAC_POLICY`	Changes the Oracle Label Security merge label algorithm through the `DVSYS.DBMS_MACADM.UPDATE_MAC_POLICY` procedure
`DELETE_MAC_POLICY_CASCADE`	Deletes all Oracle Database Vault objects related to an Oracle Label Security policy, through the `DVSYS.DBMS_MACADM.DELETE_MAC_POLICY_CASCADE` procedure

Oracle Database Vault Oracle Data Pump Audit Events

The unified audit trail can capture Oracle Database Vault Oracle Data Pump audit events.

Table 24-15 describes these events.

Table 24-15 Oracle Database Vault Oracle Data Pump Audit Events

Audit Event	Description
`AUTHORIZE_DATAPUMP_USER`	Authorizes an Oracle Data Pump user through the `DVSYS.DBMS_MACADM.AUTHORIZE_DATAPUMP_USER` procedure
`UNAUTHORIZE_DATAPUMP_USER`	Removes from authorization an Oracle Data Pump user through the `DVSYS.DBMS_MACADM.UNAUTHORIZE_DATAPUMP_USER` procedure

Oracle Database Vault Enable and Disable Audit Events

The unified audit trail can capture Oracle Database Vault enable and disable audit events.

Table 24-16 describes these events.

Table 24-16 Oracle Database Vault Enable and Disable Audit Events

Event	Description
`ENABLE_EVENT`	`DBMS_MACADM.ENABLE_EVENT`
`DISABLE_EVENT`	`DBMS_MACADM.DISABLE_EVENT`

Configuring a Unified Audit Policy for Oracle Database Vault

The `ACTIONS` and `ACTIONS COMPONENT` clauses in the `CREATE AUDIT POLICY` statement can create unified audit policies for Oracle Database Vault events.

- Use the following syntax to create an Oracle Database Vault unified audit policy:

```
CREATE AUDIT POLICY policy_name
 ACTIONS action1 [,action2 ]
 ACTIONS COMPONENT= DV DV_action ON DV_object [,DV_action2 ON DV_object2]
```

In this specification:

- *`DV_action`* is one of the following:
  - `Realm Violation`, `Realm Success`, `Realm Access`
  - `Rule Set Failure`, `Rule Set Success`, `Rule Set Eval`
  - `Factor Error`, `Factor Null`, `Factor Validate Error`, `Factor Validate False`, `Factor Trust Level Null`, `Factor Trust Level Neg`, `Factor All`
- *`DV_objects`* is one of the following:

- *`Realm_Name`*
- *`Rule_Set_Name`*
- *`Factor_Name`*

If the object was created in lower or mixed case, then you must enclose *`DV_objects`* in double quotation marks. If you had created the object in all capital letters, then you can omit the quotation marks.

For example, to audit realm violations on the Database Vault Account Management realm:

```
CREATE AUDIT POLICY audit_dv
 ACTIONS CREATE TABLE, SELECT
 ACTIONS COMPONENT=DV Realm Violation ON "Database Vault Account Management";
```

You can build more complex policies, such as those that include conditions. Remember that after you create the policy, you must use the `AUDIT` statement to enable it.

Example: Auditing an Oracle Database Vault Realm

The `CREATE AUDIT POLICY` statement can audit Oracle Database Vault realms.

Example 24-21 shows how to audit a realm violation on the `HR` schema.

Example 24-21 Auditing a Realm Violation

```
CREATE AUDIT POLICY dv_realm_hr
 ACTIONS SELECT, UPDATE, DELETE
 ACTIONS COMPONENT=DV Realm Violation ON "HR Schema Realm";

AUDIT POLICY dv_realm_hr EXCEPT psmith;
```

Example: Auditing an Oracle Database Vault Rule Set

The `CREATE AUDIT POLICY` statement can audit Oracle Database Vault rule sets.

Example: Auditing an Oracle Database Vault Rule Set shows how to audit the Can Maintain Accounts/Profile rule set. The user `dbv_acctmgr`, who has the `DV_ACCTMGR` role and hence has privileges to manage user accounts and user profiles, is exempt from this audit policy.

Example 24-22 Auditing a Rule Set

```
CREATE AUDIT POLICY dv_rule_set_accts
 ACTIONS CREATE USER, ALTER USER, ALTER PROFILE
 ACTIONS COMPONENT=DV RULE SET FAILURE ON "Can Maintain Accounts/Profile";

AUDIT POLICY dv_rule_set_accts EXCEPT dbv_acctmgr;
```

Example: Auditing Two Oracle Database Vault Events

The `CREATE AUDIT POLICY` statement can audit multiple Oracle Database Vault events.

Example 24-23 shows how to audit a realm violation and a rule set failure.

Example 24-23   Auditing Two Oracle Database Vault Events

```
CREATE AUDIT POLICY audit_dv
 ACTIONS CREATE TABLE, SELECT
 ACTIONS COMPONENT=DV REALM VIOLATION ON "Oracle Enterprise Manager", Rule Set
 Failure ON "Allow Sessions";

AUDIT POLICY audit_dv EXCEPT psmith;
```

Example: Auditing Oracle Database Vault Factors

The `CREATE AUDIT POLICY` statement can audit Oracle Database Vault factors.

Example 24-24 shows how to audit two types of errors for one factor.

Example 24-24   Auditing Oracle Database Vault Factor Settings

```
CREATE AUDIT POLICY audit_dv_factor
 ACTIONS COMPONENT=DV FACTOR ERROR ON "Database_Domain", Factor Validate Error ON
"Client_IP";

AUDIT POLICY audit_dv_factor;
```

How Oracle Database Vault Audited Events Appear in the Audit Trail

The `UNIFIED_AUDIT_TRAIL` data dictionary view lists Oracle Database Vault audited events.

The `DV_*` columns of the `UNIFIED_AUDIT_TRAIL` view show Oracle Database Vault-specific audit data.

For example:

```
SELECT DV_RULE_SET_NAME FROM UNIFIED_AUDIT_TRAIL
WHERE ACTION_NAME = 'UPDATE';

DV_RULE_SET_NAME
-----------------------
Allow System Parameters
```

Auditing Oracle Label Security Events

In an Oracle Label Security environment, the `CREATE AUDIT POLICY` statement can audit Oracle Label Security activities.

About Auditing Oracle Label Security Events

As with all unified auditing, you must have the `AUDIT_ADMIN` role before you can audit Oracle Label Security (OLS) events.

To create Oracle Label Security unified audit policies, you must set the `CREATE AUDIT POLICY` statement `COMPONENT` clause to `OLS`.

To audit user session label information, you use the `AUDIT` statement to audit application context values.

To access the audit trail, you can query the `UNIFIED_AUDIT_TRAIL` data dictionary view. This view contains Oracle Label Security-specific columns whose names begin with `OLS_`. If you want to find audit information about the internally generated VPD predicate that is created when you apply an Oracle Label Security policy to a table, then you can query the RLS_INFO column.

See Also:

- Auditing of Oracle Virtual Private Database Predicates for information about how to format the output of the RLS_INFO column
- *Oracle Label Security Administrator's Guide* for more information about Oracle Label Security

Oracle Label Security Unified Audit Trail Events

The unified audit trail can capture Oracle Label Security audit events.

To find a list of auditable Oracle Label Security events that you can audit, you can query the COMPONENT and NAME columns of the AUDITABLE_SYSTEM_ACTIONS data dictionary view.

For example:

```
SELECT NAME FROM AUDITABLE_SYSTEM_ACTIONS WHERE COMPONENT = 'Label Security';

NAME
-------------
CREATE POLICY
ALTER POLICY
DROP POLICY
...
```

Table 24-17 describes the Oracle Label Security audit events.

Table 24-17 Oracle Label Security Audit Events

Audit Event	Description
CREATE POLICY	Creates an Oracle Label Security policy through the SA_SYSDBA.CREATE_POLICY procedure
ALTER POLICY	Alters an Oracle Label Security policy through the SA_SYSDBA.ALTER_POLICY procedure
DROP POLICY	Drops an Oracle Label Security policy through the SA_SYSDBA.DROP_POLICY procedure
APPLY POLICY	Applies a table policy through the SA_POLICY_ADMIN.APPLY_TABLE_POLICY procedure or a schema policy through the SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY procedure
REMOVE POLICY	Removes a table policy through the SA_POLICY_ADMIN.REMOVE_TABLE_POLICY procedure or a schema policy through the SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY procedure

Table 24-17 (Cont.) Oracle Label Security Audit Events

Audit Event	Description
`SET AUTHORIZATION`	Covers all Oracle Label Security authorizations, including Oracle Label Security privileges and user labels to either users or trusted stored procedures. The PL/SQL procedures that correspond to the `SET AUTHORIZATION` event are `SA_USER_ADMIN.SET_USER_LABELS`, `SA_USER_ADMIN.SET_USER_PRIVS`, and `SA_USER_ADMIN.SET_PROG_PRIVS`.
`PRIVILEGED ACTION`	Covers any action that requires the user of an Oracle Label Security privilege. These actions are logons, `SA_SESSION.SET_ACCESS_PROFILE` executions, and the invocation of trusted stored procedures.
`ENABLE POLICY`	Enables an Oracle Label Security policy through the following procedures: • `SA_SYSDBA.ENABLE_POLICY`: Enforces access control on the tables and schemas protected by the policy • `SA_POLICY_ADMIN.ENABLE_TABLE_POLICY`: Enables an Oracle Label Security policy for a specified table • `SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY`: Enables an Oracle Label Security policy for all the tables in a specified schema
`DISABLE POLICY`	Disables an Oracle Label Security policy through the following procedures: • `SA_SYSDBA.DISABLE_POLICY`: Disables the enforcement of an Oracle Label Security policy • `SA_POLICY_ADMIN.DISABLE_TABLE_POLICY`: Disables the enforcement an Oracle Label Security policy for a specified table • `SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY`: Disables the enforcement of an Oracle Label Security policy for all the tables in a specified schema
`SUBSCRIBE OID`	Subscribes to an Oracle Internet Directory-enabled Oracle Label Security policy through the `SA_POLICY_ADMIN.POLICY_SUBSCRIBE` procedure
`UNSUBSCRIBE OID`	Unsubscribes to an Oracle Internet Directory-enabled Oracle Label Security policy through the `SA_POLICY_ADMIN.POLICY_UNSUBSCRIBE` procedure
`CREATE DATA LABEL`	Creates an Oracle Label Security data label through the `SA_LABEL_ADMIN.CREATE_LABEL` procedure. `CREATE DATA LABEL` also corresponds to the `LBACSYS.TO_DATA_LABEL` function.
`ALTER DATA LABEL`	Alters an Oracle Label Security data label through the `SA_LABEL_ADMIN.ALTER_LABEL` procedure
`DROP DATA LABEL`	Drops an Oracle Label Security data label through the `SA_LABEL_ADMIN.DROP_LABEL` procedure
`CREATE LABEL COMPONENT`	Creates an Oracle Label Security component through the following procedures: • **Levels:** `SA_COMPONENTS.CREATE_LEVEL` • **Compartments:** `SA_COMPONENTS.CREATE_COMPARTMENT` • **Groups:** `SA_COMPONENTS.CREATE_GROUP`

Table 24-17 (Cont.) Oracle Label Security Audit Events

Audit Event	Description
`ALTER LABEL COMPONENTS`	Alters an Oracle Label Security component through the following procedures: • **Levels:** `SA_COMPONENTS.ALTER_LEVEL` • **Compartments:** `SA_COMPONENTS.ALTER_COMPARTMENT` • **Groups:** `SA_COMPONENTS.ALTER_GROUP` and `SA_COMPONENTS.ALTER_GROUP_PARENT`
`DROP LABEL COMPONENTS`	Drops an Oracle Label Security component through the following procedures: • **Levels:** `SA_COMPONENTS.DROP_LEVEL` • **Compartments:** `SA_COMPONENTS.DROP_COMPARTMENT` • **Groups:** `SA_COMPONENTS.DROP_GROUP`
`ALL`	Enables auditing of all Oracle Label Security actions

Oracle Label Security Auditable User Session Labels

The `ORA_OLS_SESSION_LABELS` application context can capture user session label usage for each Oracle Database event.

The attributes used by this application context refer to Oracle Label Security policies. .

The syntax is the same as the syntax used for application context auditing, described in Configuring Application Context Audit Settings. For example:

```
AUDIT CONTEXT NAMESPACE ORA_SESSION_LABELS ATTRIBUTES policy1, policy2;
```

Because the recording of session labels is not user-session specific, the `BY` *`user_list`* clause is not required for auditing Oracle Label Security application contexts.

To disable the auditing of user session label information, you use the `NOAUDIT` statement. For example, to stop auditing for policies `policy1` and `policy2`, enter the following statement:

```
NOAUDIT CONTEXT NAMESPACE ORA_SESSION_LABELS ATTRIBUTES policy1, policy2;
```

Configuring a Unified Audit Policy for Oracle Label Security

The `ACTIONS` and `ACTIONS COMPONENT` clauses in the `CREATE AUDIT POLICY` statement can be used to create Oracle Label Security event audit policies.

- Use the following syntax to create an Oracle Label Security unified audit policy:

  ```
  CREATE AUDIT POLICY policy_name
   ACTIONS action1 [,action2 ]
   ACTIONS COMPONENT=OLS component_action1 [, action2];
  ```

For example:

```
CREATE AUDIT POLICY audit_ols
 ACTIONS SELECT ON OE.ORDERS
 ACTIONS COMPONENT=OLS ALL;
```

You can build more complex policies, such as those that include conditions. Remember that after you create the policy, you must use the AUDIT statement to enable it.

Example: Auditing Oracle Label Security Session Label Attributes

The AUDIT CONTEXT NAMESPACE statement can audit Oracle Label Security session label attributes.

Example 24-25 shows how to audit ORA_OLS_SESSION_LABELS application context attributes for the Oracle Label Security policies usr_pol1 and usr_pol2.

Example 24-25 Auditing Oracle Label Security Session Label Attributes

```
AUDIT CONTEXT NAMESPACE ORA_SESSION_LABELS ATTRIBUTES usr_pol1, usr_pol2;
```

Example: Excluding a User from an Oracle Label Security Policy

The CREATE AUDIT POLICY statement can exclude users from policies.

Example 24-26 shows how to create a unified audit policy that excludes actions from user ols_mgr.

Example 24-26 Excluding a User from an Oracle Label Security Policy

```
CREATE AUDIT POLICY auth_ols_audit_pol
 ACTIONS SELECT ON HR.EMPLOYEES
 ACTIONS COMPONENT=OLS DROP POLICY, DISABLE POLICY;

AUDIT POLICY auth_ols_audit_pol EXCEPT ols_mgr;
```

Example: Auditing Oracle Label Security Policy Actions

The CREATE AUDIT POLICY statement can audit Oracle Label Security policy actions.

Example 24-27 shows how to audit the DROP POLICY, DISABLE POLICY, UNSUBSCRIBE OID events, and UPDATE and DELETE statements on the HR.EMPLOYEES table. Then this policy is applied to the HR and LBACSYS users, and audit records are written to the unified audit trail only when the audited actions are successful.

Example 24-27 Auditing Oracle Label Security Policy Actions

```
CREATE AUDIT POLICY generic_audit_pol
 ACTIONS UPDATE ON HR.EMPLOYEES, DELETE ON HR.EMPLOYEES
 ACTIONS COMPONENT=OLS DROP POLICY, DISABLE POLICY, UNSUBSCRIBE OID;

AUDIT POLICY generic_audit_pol BY HR, LBACSYS WHENEVER SUCCESSFUL;
```

Example: Querying for Audited OLS Session Labels

The LBACSYS.ORA_GET_AUDITED_LABEL function can be used in a UNIFIED_AUDIT_TRAIL query to find audited Oracle Label Security session labels.

Example 24-28 shows how to use the LBACSYS.ORA_GET_AUDITED_LABEL function in a UNIFIED_AUDIT_TRAIL data dictionary view query.

Example 24-28   Querying for Audited Oracle Label Security Session Labels

```
SELECT ENTRY_ID, SESSIONID,
       LBACSYS.ORA_GET_AUDITED_LABEL( APPLICATION_CONTEXTS,'GENERIC_AUDIT_POL1') AS
SESSION_LABEL1,
       LBACSYS.ORA_GET_AUDITED_LABEL( APPLICATION_CONTEXTS,'GENERIC_AUDIT_POL2') AS
SESSION_LABEL2
FROM UNIFIED_AUDIT_TRAIL;
/

ENTRY_ID  SESSIONID  SESSION_LABEL1  SESSION_LABEL2
--------  ---------  --------------  --------------
       1       1023  SECRET          LEVEL_ALPHA
       2       1024  TOP_SECRET      LEVEL_BETA
```

How Oracle Label Security Audit Events Appear in the Audit Trail

The UNIFIED_AUDIT_TRAIL data dictionary view lists Oracle Label Security audit events.

The OLS_* columns of the UNIFIED_AUDIT_TRAIL view show Oracle Label Security-specific audit data. For example:

```
SELECT OLS_PRIVILEGES_USED FROM UNIFIED_AUDIT_TRAIL WHERE DBUSERNAME = 'psmith';

OLS_PRIVILEGES_USED
-------------------
READ
WRITEUP
WRITEACROSS
```

The session labels that the audit trail captures are stored in the APPLICATION_CONTEXTS column of the UNIFIED_AUDIT_TRAIL view. You can use the LBACSYS.ORA_GET_AUDITED_LABEL function to retrieve session labels that are stored in the APPLICATION_CONTEXTS column. This function accepts the UNIFIED_AUDIT_TRAIL.APPLICATION_CONTEXTS column value, and the Oracle Label Security policy name as arguments, and then returns the session label that is stored in the column for the specified policy.

See Also:

Oracle Label Security Administrator’s Guide for more information about the ORA_GET_AUDITED_LABEL function

Auditing Oracle Data Mining Events

You can use the CREATE AUDIT POLICY statement to audit Oracle Data Mining events.

About Auditing Oracle Data Mining Events

You must have the AUDIT_ADMIN role to audit Oracle Data Mining events.

To access the audit trail, you can query the UNIFIED_AUDIT_TRAIL data dictionary view.

See Also:

Oracle Data Mining Concepts for more information about Oracle Data Mining

Oracle Data Mining Unified Audit Trail Events

The unified audit trail can capture Oracle Data Mining audit events..

Table 24-18 describes these events.

Table 24-18 Oracle Data Mining Audit Events

Audit Event	Description
`AUDIT`	Generates an audit record for a Data Mining model
`COMMENT`	Adds a comment to a Data Mining model
`GRANT`	Gives permission to a user to access the Data Mining model
`RENAME`	Changes the name of the Data Mining model
`SELECT`	Applies the Data Mining model or view its signature

Configuring a Unified Audit Policy for Oracle Data Mining

The `CREATE AUDIT POLICY` statement `ACTIONS` and `ON MINING MODEL` clauses can be used to create Oracle Data Mining event unified audit policies.

- Use the following syntax to create a unified audit policy for Oracle Data Mining:

```
CREATE AUDIT POLICY policy_name
ACTIONS {operation | ALL}
ON MINING MODEL schema_name.model_name;
```

For example:

```
CREATE AUDIT POLICY dm_ops ACTIONS RENAME ON MINING MODEL hr.dm_emp;
```

You can build more complex policies, such as those that include conditions. Remember that after you create the policy, you must use the `AUDIT` statement to enable it.

Example: Auditing Multiple Oracle Data Mining Operations by a User

The `CREATE AUDIT POLICY` statement can audit multiple Oracle Data Mining operations.

Example 24-29 shows how to audit multiple Oracle Data Mining operations by user `psmith`. Include the `ON MINING MODEL` *`schema_name.model_name`* clause for each event, and separate each with a comma. This example specifies the same *`schema_name.model name`* for both actions, but the syntax enables you to specify different *`schema_name.model_name`* settings for different schemas and data models.

Example 24-29 Auditing Multiple Oracle Data Mining Operations by a User

```
CREATE AUDIT POLICY dm_ops_pol
ACTIONS SELECT ON MINING MODEL dmuser1.nb_model, ALTER ON MINING MODEL
dmuser1.nb_model;
```

```
AUDIT POLICY dm_ops_pol BY psmith;
```

Example: Auditing All Failed Oracle Data Mining Operations by a User

The `CREATE AUDIT POLICY` statement can audit failed Oracle Data Mining operations by a user.

Example 24-30 shows how to audit all failed Oracle Data Mining operations by user `psmith`.

Example 24-30 Auditing All Failed Oracle Data Mining Operations by a User

```
CREATE AUDIT POLICY dm_all_ops_pol ACTIONS ALL ON MINING MODEL dmuser1.nb_model;

AUDIT POLICY dm_all_ops_pol BY psmith WHENEVER NOT SUCCESSFUL;
```

How Oracle Data Mining Events Appear in the Audit Trail

The `UNIFIED_AUDIT_TRAIL` data dictionary view lists Oracle Data Mining audit events.

The following example shows how to query the `UNIFIED_AUDIT_TRAIL` data dictionary view for Data Mining audit events.

```
SELECT DBUSERNAME, ACTION_NAME, SYSTEM_PRIVILEGE_USED, RETURN_CODE,
OBJECT_SCHEMA, OBJECT_NAME, SQL_TEXT
FROM UNIFIED_AUDIT_TRAIL;

DBUSERNAME ACTION_NAME          SYSTEM_PRIVILEGE_USED     RETURN_CODE
---------- -------------------- ------------------------- -----------
OBJECT_SCHEMA        OBJECT_NAME
-------------------- --------------------
SQL_TEXT
--------------------------------------------------------------------------------
DMUSER1    CREATE MINING MODEL  CREATE MINING MODEL                 0
DMUSER1
BEGIN
  dbms_data_mining.create_model(model_name => 'nb_model',
                mining_function => dbms_data_mining.classification,
                data_table_name => 'dm_data',
                case_id_column_name => 'case_id',
                target_column_name => 'target');
END;

DMUSER1    SELECT MINING MODEL                                      0
DMUSER1              NB_MODEL
select prediction(nb_model using *) from dual

DMUSER2    SELECT MINING MODEL                                  40284
DMUSER1              NB_MODEL
select prediction(dmuser1.nb_model using *) from dual

DMUSER1    ALTER MINING MODEL                                       0
DMUSER1              NB_MODEL
BEGIN dbms_data_mining.rename_model('nb_model', 'nb_model1'); END;


DMUSER2    ALTER MINING MODEL                                   40284
DMUSER1              NB_MODEL
BEGIN dbms_data_mining.rename_model('dmuser1.nb_model1', 'nb_model'); END;
```

```
DMUSER2    ALTER MINING MODEL                                  40284
DMUSER1              NB_MODEL
BEGIN dbms_data_mining.rename_model('dmuser1.nb_model1', 'nb_model'); END;
```

Auditing Oracle Data Pump Events

You can use the CREATE AUDIT POLICY statement to audit Oracle Data Pump.

About Auditing Oracle Data Pump Events

The CREATE AUDIT POLICY statement COMPONENT clause must be set to DATAPUMP to create Oracle Data Pump unified audit policies.

You can audit Data Pump export (expdp) and import (impdp) operations.

As with all unified auditing, you must have the AUDIT_ADMIN role before you can audit Oracle Data Pump events.

To access the audit trail, query the UNIFIED_AUDIT_TRAIL data dictionary view. The Data Pump-specific columns in this view begin with DP_.

See Also:

Oracle Database Utilities for detailed information about Oracle Data Pump

Oracle Data Pump Unified Audit Trail Events

The unified audit trail can capture Oracle Data Pump events.

The unified audit trail captures information about both export (expdp) and import (impdp) operations.

Configuring a Unified Audit Policy for Oracle Data Pump

The ACTIONS COMPONENT clause in the CREATE AUDIT POLICY statement can be used to create an Oracle Data Pump event unified audit policy.

- Use the following syntax to create a unified audit policy for Oracle Data Pump:

  ```
  CREATE AUDIT POLICY policy_name
  ACTIONS COMPONENT=DATAPUMP { EXPORT | IMPORT | ALL };
  ```

For example:

```
CREATE AUDIT POLICY audit_dp_export_pol
 ACTIONS COMPONENT=DATAPUMP EXPORT;
```

You can build more complex policies, such as those that include conditions. Remember that after you create the policy, you must use the AUDIT statement to enable it.

Example: Auditing Oracle Data Pump Import Operations

The `CREATE AUDIT POLICY` statement can audit Oracle Data Pump import operations.

Example 24-31 shows how to audit all Oracle Data Pump import operations.

Example 24-31 Auditing Oracle Data Pump Import Operations

```
CREATE AUDIT POLICY audit_dp_import_pol
 ACTIONS COMPONENT=DATAPUMP IMPORT;

AUDIT POLICY audit_dp_import_pol;
```

Example: Auditing All Oracle Data Pump Operations

The `CREATE AUDIT POLICY` statement can audit all Oracle Data Pump operations.

Example 24-32 shows how to audit both Oracle Database Pump export and import operations.

Example 24-32 Auditing All Oracle Data Pump Operations

```
CREATE AUDIT POLICY audit_dp_all_pol
 ACTIONS COMPONENT=DATAPUMP ALL;

AUDIT POLICY audit_dp_all_pol BY SYSTEM;
```

How Oracle Data Pump Audited Events Appear in the Audit Trail

The `UNIFIED_AUDIT_TRAIL` data dictionary view lists Oracle Data Pump audited events.

The `DP_*` columns of the `UNIFIED_AUDIT_TRAIL` view show Oracle Data Pump-specific audit data. For example:

```
SELECT DP_TEXT_PARAMETERS1, DP_BOOLEAN_PARAMETERS1 FROM UNIFIED_AUDIT_TRAIL
WHERE AUDIT_TYPE = 'DATAPUMP';

DP_TEXT_PARAMETERS1                              DP_BOOLEAN_PARAMETERS1
---------------------------------------------- ----------------------------------

MASTER TABLE:  "SCOTT"."SYS_EXPORT_TABLE_01",  MASTER_ONLY: FALSE,
JOB_TYPE: EXPORT,                              DATA_ONLY: FALSE,
METADATA_JOB_MODE: TABLE_EXPORT,               METADATA_ONLY: FALSE,
JOB VERSION: 12.1.0.0,                         DUMPFILE_PRESENT: TRUE,
ACCESS METHOD: DIRECT_PATH,                    JOB_RESTARTED: FALSE
DATA OPTIONS: 0,
DUMPER DIRECTORY: NULL
REMOTE LINK: NULL,
TABLE EXISTS: NULL,
PARTITION OPTIONS: NONE
```

(This output was reformatted for easier readability.)

Auditing Oracle SQL*Loader Direct Load Path Events

You can use the `CREATE AUDIT POLICY` statement to audit Oracle SQL*Loader direct load path events.

About Auditing in Oracle SQL*Loader Direct Path Load Events

You must have the `AUDIT_ADMIN` role to audit Oracle SQL*Loader direct path events.

To create SQL*Loader unified audit policies, you must set the `CREATE AUDIT POLICY` statement's `COMPONENT` clause to `DIRECT_LOAD`. You can audit direct path load operations only, not other SQL*Loader loads, such as conventional path loads.

To access the audit trail, you can query the `DIRECT_PATH_NUM_COLUMNS_LOADED` column in the `UNIFIED_AUDIT_TRAIL` data dictionary view.

See Also:

Oracle Database Utilities for detailed information about Oracle SQL*Loader

Oracle SQL*Loader Direct Load Path Unified Audit Trail Events

The unified audit trail can capture SQL*Loader Direct Load Path events.

The unified audit trail captures information about direct path loads that SQL*Loader performs (that is, when you set `direct=true` on the SQL*Loader command line or in the SQL*Loader control file).

It also audits Oracle Call Interface (OCI) programs that use the direct path API.

See Also:

Oracle Database Utilities for detailed information about direct path loads in Oracle SQL*Loader

Configuring a Unified Audit Trail Policy for Oracle SQL*Loader Direct Path Events

The `CREATE AUDIT POLICY` statement `ACTIONS COMPONENT` clause can create unified audit policies for Oracle SQL*Loader direct path events.

- Use the following syntax to create an Oracle SQL*Loader unified audit policy:

  ```
  CREATE AUDIT POLICY policy_name
  ACTIONS COMPONENT=DIRECT_LOAD { LOAD };
  ```

For example:

```
CREATE AUDIT POLICY audit_sqlldr_pol
 ACTIONS COMPONENT=DIRECT_LOAD LOAD;
```

You can build more complex policies, such as those that include conditions. Remember that after you create the policy, you must use the `AUDIT` statement to enable it.

Example: Auditing Oracle SQL*Loader Direct Path Load Operations

The `CREATE AUDIT POLICY` statement can audit Oracle SQL*Loader direct path load operations.

Example 24-31 shows how to audit SQL*Loader direct path load operations.

Example 24-33 Auditing Oracle SQL*Loader Direct Path Load Operations

```
CREATE AUDIT POLICY audit_sqlldr_load_pol
 ACTIONS COMPONENT=DIRECT_LOAD LOAD;

AUDIT POLICY audit_sqlldr_load_pol;
```

How SQL*Loader Direct Path Load Audited Events Appear in the Audit Trail

The `UNIFIED_AUDIT_TRAIL` data dictionary view lists SQL*Loader direct path load audited events.

The `DIRECT_PATH_NUM_COLUMNS_LOADED` column of the `UNIFIED_AUDIT_TRAIL` view shows the number of columns that were loaded using the SQL*Loader direct path load method. For example:

```
SELECT DBUSERNAME, ACTION_NAME, OBJECT_SCHEMA, OBJECT_NAME,
DIRECT_PATH_NUM_COLUMNS_LOADED FROM UNIFIED_AUDIT_TRAIL WHERE AUDIT_TYPE = 'DIRECT
PATH API';

DBUSERNAME  ACTION_NAME OBJECT_SCHEMA OBJECT_NAME  DIRECT_PATH_NUM_COLUMNS_LOADED
----------- ----------- ------------- ------------ ------------------------------
RLAYTON     INSERT       HR            EMPLOYEES    4
```

Unified Audit Policies or AUDIT Settings in a Multitenant Environment

In a multitenant environment, you can create unified audit policies for individual PDBs and in the root.

About Local, CDB Common, and Application Common Audit Policies

An audit policy can be either a local audit policy, a CDB common audit policy, or an application common audit policy.

This applies to both unified audit policies and policies that are created using the `AUDIT` SQL statement.

- **Local audit policy.** This type of policy can exist in either the root (CDB or application) or the PDB (CDB or application). A local audit policy that exists in the root can contain object audit options for both local and common objects. Both local and common users who have been granted the `AUDIT_ADMIN` role can enable local policies: local users from their PDBs and common users from the root or the PDB to which they have privileges. You can enable a local audit policy for both local and common users and roles.

  You can create local audit policies for application local objects and application local roles, as well as system action options and system privilege options. You cannot enforce a local audit policy for a common user across all containers, nor can you enforce a common audit policy for a local user.

- **CDB common audit policy.** This type of policy is available to all PDBs in the multitenant environment. Only common users who have been granted the `AUDIT_ADMIN` role can create and maintain common audit policies. You can enable common audit policies only for common users. You must create common audit policies only in the root. This type of policy can contain object audit options of only common objects, and be enabled only for common users. You can enable a common audit policy for common users and roles only.

  You cannot enforce a common audit policy for a local user across all containers.
- **Application common audit policy.** Similar to CDB common audit policies, this type of policy is available to all PDBs in the multitenant environment. You can create common audit policies for application common objects and application common roles, as well as system action options and system privilege options. You can only create this type of policy in the application root container, but you can enable it on both application common users and CDB common users. If you want to audit objects, then ensure that these objects are application common objects. You can determine whether an object is an application common object by querying the `SHARING` column of the `DBA_OBJECTS` data dictionary view.

By default, audit policies are local to the current PDB, for both CDB and application scenarios.

The following table explains how audit policies apply in different multitenant environments.

Table 24-19 How Audit Policies Apply to the CDB Root, Application Root, and Individual PDBs

Audit Option Type	CDB Root	Application Root	Individual PDB
Common audit statement or audit policy	Applies to CDB common users	Applies to CDB common users	Applies to CDB common users
Application container common audit statement or audit policy	Not applicable	• Applies to CDB common users and are valid for the current application container only • Applies to application container common users	• Applies to CDB common users and are valid for this application container only • Applies to application common users
Local audit statement or audit policy	Local configurations not allowed	Local configurations not allowed	• Applies to CDB common users • Applies to application common users

Traditional Auditing in a Multitenant Environment

In traditional auditing (not unified auditing), the `AUDIT` and `NOAUDIT` statements can audit statements and privileges in a multitenant environment.

To configure the audit policy to be either a local audit policy or a common audit policy, you must include the `CONTAINER` clause, as you normally do for other SQL creation or modification statements. If you want to audit an application container, then you can audit SQL statement and system privileges performed by local and common users and roles. The audit record will be created in the container in which the action was performed.

- If you want to apply the `AUDIT` or `NOAUDIT` statement to the current CDB or application PDB, then in this PDB, you must set `CONTAINER` to `CURRENT`. For example:

  ```
  AUDIT DROP ANY TABLE BY SYSTEM BY ACCESS CONTAINER = CURRENT;
  ```

- If you want to apply the `AUDIT` or `NOAUDIT` statement to the entire multitenant environment, then in the CDB root, then you must set `CONTAINER` to `ALL`. For an application container, you would set it in the application root. For example:

  ```
  AUDIT DROP ANY TABLE BY SYSTEM BY ACCESS CONTAINER = ALL;
  ```

To find if a traditional audit option is designed for use in an application container, perform a join query with the `DBA_OBJ_AUDIT_OPTS` and `DBA_OBJECTS` data dictionary views, by using the `OWNER` and `OBJECT_NAME` columns in both views, and the `APPLICATION` column in `DBA_OBJECTS`.

See Also:

Oracle Database SQL Language Reference for more information about the traditional `AUDIT` and `NOAUDIT` SQL statements

Configuring a Local Unified Audit Policy or Common Unified Audit Policy

The `CONTAINER` clause is specific to multitenant environment use for the `CREATE AUDIT POLICY` statement.

To create a local or common (CDB or application) unified audit policy in either the CDB environment or an application container environment, include the `CONTAINER` clause in the `CREATE AUDIT POLICY` statement.

- Use the following syntax to create a local or common unified audit policy:

  ```
  CREATE AUDIT POLICY policy_name
   action1 [,action2 ]
   [CONTAINER = {CURRENT | ALL}];
  ```

In this specification:

- `CURRENT` sets the audit policy to be local to the current PDB.
- `ALL` makes the audit policy a common audit policy, that is, available to the entire multitenant environment.

For example, for a common unified audit policy:

```
CREATE AUDIT POLICY dict_updates
 ACTIONS UPDATE ON SYS.USER$,
  DELETE ON SYS.USER$,
  UPDATE ON SYS.LINK$,
  DELETE ON SYS.LINK$
  CONTAINER = ALL;
```

Note the following:

- You can set the `CONTAINER` clause for the `CREATE AUDIT POLICY` statement but not for `ALTER AUDIT POLICY` or `DROP AUDIT POLICY`. If you want to change the scope of an existing unified audit policy to use this setting, then you must drop and re-create the policy.
- For `AUDIT` statements, you can set the `CONTAINER` clause for audit settings only if you have an Oracle database that has not been migrated to the Release 12.*x* audit features. You cannot use the `CONTAINER` clause in an `AUDIT` statement that is used to enable a unified audit policy.
- If you are in a PDB, then you can only set the `CONTAINER` clause to `CURRENT`, not `ALL`. If you omit the setting while in the PDB, then the default is `CONTAINER = CURRENT`.
- If you are in the root, then you can set the `CONTAINER` clause to either `CURRENT` if you want the policy to apply to the root only, or to `ALL` if you want the policy to apply to the entire CDB. If you omit the `CONTAINER` clause, then default is `CONTAINER = CURRENT`.
- For objects:
  - Common audit policies can have common objects only and local audit policies can have both local objects and common objects.
  - You cannot set `CONTAINER` to `ALL` if the objects involved are local. They must be common objects.
- For privileges:
  - You can set the `CONTAINER` to `CURRENT` (or omit the `CONTAINER` clause) if the user accounts involved are a mixture of local and common accounts. This creates a local audit configuration that applies only to the current PDB.
  - You cannot set `CONTAINER` to `ALL` if the users involved are local users. They must be common users.
  - If you set `CONTAINER` to `ALL` and do not specify a user list (using the `BY` clause in the `AUDIT` statement), then the configuration applies to all common users in each PDB.
- For application containers, you can run a common unified audit policy from the application container script that is used for application install, upgrade, patch, and uninstall operations. To do so:
  1. Create a common unified audit policy in the application container root, and set this policy to `CONTAINER = ALL`. Alternatively, you can include this policy in the script that is described in this next step.
  2. Create a custom version of the script you normally would use to install, upgrade, patch, or uninstall Oracle Database.
  3. Within this script, include the SQL statements that you want to audit within the following lines:

     ```
     ALTER PLUGGABLE DATABASE APPLICATION BEGIN INSTALL
     List SQL statements here. Separate each statement with a semi-colon.
     ALTER PLUGGABLE DATABASE APPLICATION END INSTALL
     ```

     If you include the unified audit policy in the script, then ensure that you include both the `CREATE AUDIT POLICY` and `AUDIT POLICY` statements.

  After the audit policy is created and enabled, all user access to the application common objects is audited irrespective of whether the audit policy is defined in the database or from the script.

- To audit application install, upgrade, patch, and uninstall operations locally in an application root or an application PDB, follow a procedure similar to the preceding procedure for common unified audit policies, but synchronize the application PDB afterward. For example:

  ```
  ALTER PLUGGABLE DATABASE APPLICATION application_name SYNC;
  ```

Example: Local Unified Audit Policy

The `CREATE AUDIT POLICY` statement can create a local unified audit policy in either the root or a PDB.

When you create a local unified audit policy in the root, it only applies to the root and not across the multitenant environment.

The following example shows a local unified audit policy that has been created by the common user `c##sec_admin` from a PDB and applied to common user `c##hr_admin`.

Example 24-34 Local Unified Audit Policy

```
CONNECT c##sec_admin@hrpdb
Enter password: password
Connected.

CREATE AUDIT POLICY table_privs
 PRIVILEGES CREATE ANY TABLE, DROP ANY TABLE
 CONTAINER = CURRENT;

AUDIT POLICY table_privs BY c##hr_admin;
```

Example: CDB Common Unified Audit Policy

The `CREATE AUDIT POLICY` statement can create a CDB common unified audit policy.

Example 24-35 shows a common unified audit policy that has been created by the common user `c##sec_admin` from the root and applied to common user `c##hr_admin`.

Example 24-35 Common Unified Audit Policy

```
CONNECT c##sec_admin
Enter password: password
Connected.

CREATE AUDIT POLICY admin_pol
 ACTIONS CREATE TABLE, ALTER TABLE, DROP TABLE
 ROLES c##hr_mgr, c##hr_sup
 CONTAINER = ALL;

AUDIT POLICY admin_pol BY c##hr_admin;
```

Example: Application Common Unified Audit Policy

For application container common unified audit policies, you can audit action options and system privilege options, and refer to common objects and roles.

You can create the application common audit policy only from the application root, and enable the policy for both application common users and CDB common users.

The following example shows how to create a policy that audits the application common user SYSTEM for the application container app_pdb. The audit policy audits SELECT actions on the SYSTEM.utils_tab table and on DROP TABLE actions on any of the PDBs in the container database, including the CDB root. The policy also audits the use of the SELECT ANY TABLE system privilege across all containers.

Example 24-36 Application Common Unified Audit Policy

```
CONNECT c##sec_admin@app_pdb
Enter password: password
Connected.

CREATE AUDIT POLICY app_pdb_admin_pol
 ACTIONS SELECT ON hr_app_cdb.utils_tab, DROP TABLE
 PRIVILEGES SELECT ANY TABLE
 CONTAINER = ALL;

AUDIT POLICY app_pdb_admin_pol by SYSTEM, c##hr_admin;
```

In the preceding example, setting CONTAINER to ALL applies the policy only to all the relevant object accesses in the application root and on all the application PDBs that belong to the application root. It does not apply the policy outside this scope.

How Local or Common Audit Policies or Settings Appear in the Audit Trail

You can query unified audit policy views from either the root or the PDB in which the action occurred.

You can perform the following types of queries:

- **Audit records from all PDBs.** The audit trail reflects audited actions that have been performed in the PDBs. For example, if user lbrown in PDB1 performs an action that has been audited by either a common or a local audit policy, then the audit trail will capture this action. The DBID column in the UNIFIED_AUDIT_TRAIL data dictionary view indicates the PDB in which the audited action takes place and to which the policy applies. If you want to see audit records from all PDBs, you should query the CDB_UNIFIED_AUDIT_TRAIL data dictionary view from the root.
- **Audit records from common audit policies.** This location is where the common audit policy results in an audit record. The audit record can be generated anywhere in the multitenant environment—the root or the PDBs, depending on where the action really occurred. For example, the common audit policy fga_pol audits the EXECUTE privilege on the DBMS_FGA PL/SQL package, and if this action occurs in PDB1, then the audit record is generated in PDB1 and not in the root. Hence, the audit record can be seen in PDB1.

  You can query the UNIFIED_AUDIT_TRAIL data dictionary view for the policy from either the root or a PDB if you include a WHERE clause for the policy name (for example, WHERE UNIFIED_AUDIT_POLICIES = 'FGA_POL').

The following example shows how to find the results of a common unified audit policy:

```
CONNECT c##sec_admin
Enter password: password
Connected.

SELECT DBID, ACTION_NAME, OBJECT_SCHEMA, OBJECT_NAME FROM CDB_UNIFIED_AUDIT_TRAIL
WHERE DBUSERNAME = 'c##hr_admin';
46892-1
DBID        ACTION_NAME  OBJECT_SCHEMA  OBJECT_NAME
```

```
----------- ----------- ------------- -----------
653916017   UPDATE      HR            EMPLOYEES
653916018   UPDATE      HR            JOB_HISTORY
653916017   UPDATE      HR            JOBS
```

Altering Unified Audit Policies

You can use the `ALTER AUDIT POLICY` statement to modify a unified audit policy.

About Altering Unified Audit Policies

You can change most properties in a unified audit policy, except for its `CONTAINER` setting.

You cannot alter unified audit policies in a multitenant environment. For example, you cannot turn a common unified audit policy into a local unified audit policy.

To find existing unified audit policies, query the `AUDIT_UNIFIED_POLICIES` data dictionary view. If you want to find only the enabled unified audit policies, then query the `AUDIT_UNIFIED_ENABLED_POLICIES` view. You can alter both enabled and disabled audit policies. If you alter an enabled audit policy, it remains enabled after you alter it.

After you alter an object unified audit policy, the new audit settings take place immediately, for both the active and subsequent user sessions. If you alter system audit options, or audit conditions of the policy, then they are activated for new user sessions, but not the current user session.

Altering a Unified Audit Policy

The `ALTER AUDIT POLICY` statement can modify a unified audit policy.

- Use the following syntax to alter a unified audit policy, you use the `ALTER AUDIT POLICY` statement.

  ```
  ALTER AUDIT POLICY  policy_name
  [ADD [privilege_audit_clause][action_audit_clause]
    [role_audit_clause]]
  [DROP [privilege_audit_clause][action_audit_clause]
      [role_audit_clause]]
  [CONDITION {DROP | audit_condition EVALUATE PER {STATEMENT|SESSION|INSTANCE}}]
  ```

In this specification:

- `ADD` enables you to alter the following the following settings:
  - *`privilege_audit_clause`* describes privilege-related audit options. See Auditing System Privileges for details. The detailed syntax for configuring privilege audit options is as follows:

    ```
    ADD privilege_audit_clause  :=  PRIVILEGES  privilege1 [, privilege2]
    ```

  - *`action_audit_clause`* and *`standard_actions`* describe object action-related audit options. See Auditing Object Actions. The syntax is as follows:

    ```
    ADD action_audit_clause := {standard_actions | component_actions}
                                         [, component_actions ]
    standard_actions :=
         ACTIONS action1 [ ON {schema.obj_name
                                          | DIRECTORY directory_name
                                          | MINING MODEL schema.obj_name
    ```

```
                                        }
                  ]
              [, action2 [ ON {schema.obj_name
                                       | DIRECTORY directory_name
                                       | MINING MODEL schema.obj_name
                     }
                  ]
```

  – *role_audit_clause* enables you to add or drop the policy for roles. See Auditing Roles. The syntax is:

    ```
    ADD role_audit_clause := ROLES role1 [, role2]
    ```

- DROP enables you to drop the same components that are described for the ADD clause. For example:

  ```
  DROP role_audit_clause := ROLES role1 [, role2]
  ```

- CONDITION {DROP... enables you to add or drop a condition for the policy. If you are altering an existing condition, then you must include the EVALUATE PER clause with the condition. See Creating a Condition for a Unified Audit Policy. The syntax is:

  ```
  CONDITION 'audit_condition := function operation value_list'
  EVALUATE PER {STATEMENT|SESSION|INSTANCE}
  ```

  If you want to drop a condition, then omit the condition definition and the EVALUATE PER clause. For example:

  ```
  CONDITION DROP
  ```

Example: Altering a Condition in a Unified Audit Policy

The ALTER AUDIT POLICY statement can alter conditions in unified audit policies.

Example 24-37 shows how to change a condition in an existing unified audit policy.

Example 24-37 Altering a Condition in a Unified Audit Policy

```
ALTER AUDIT POLICY orders_unified_audpol
 ADD ACTIONS INSERT ON SCOTT.EMP
CONDITION 'SYS_CONTEXT(''ENTERPRISE'', ''GROUP'') =  ''ACCESS_MANAGER'''
EVALUATE PER SESSION;
```

Example: Altering an Oracle Label Security Component in a Unified Audit Policy

The ALTER AUDIT POLICY statement can alter Oracle Label Security components in an audit policy.

Example 24-38 shows how to alter an Oracle Label Security component in an audit policy.

Example 24-38 Altering an Oracle Label Security Component in a Unified Audit Policy

```
ALTER AUDIT POLICY audit_ols
 ADD ACTIONS SELECT ON HR.EMPLOYEES
 ACTIONS COMPONENT=OLS DROP POLICY, DISABLE POLICY, REMOVE POLICY;
```

Example: Altering Roles in a Unified Audit Policy

The `ALTER AUDIT POLICY` statement can alter roles in a unified audit policy.

Example 24-39 shows how to add roles to a common unified audit policy.

Example 24-39 Altering Roles in a Unified Audit Policy

```
CONNECT c##sec_admin
Enter password: password
Connected.

ALTER AUDIT POLICY RoleConnectAudit
 ADD ROLES c##role1, c##role2;
```

Example: Dropping a Condition from a Unified Audit Policy

The `ALTER AUDIT POLICY` statement can drop a condition from a unified audit policy.

Example 24-40 shows how to drop a condition from an existing unified audit policy.

Example 24-40 Dropping a Condition from a Unified Audit Policy

```
ALTER AUDIT POLICY orders_unified_audpol
CONDITION DROP;
```

Enabling and Applying Unified Audit Policies to Users and Roles

You can use the `AUDIT POLICY` statement to enable and apply unified audit policies to users and roles.

About Enabling Unified Audit Policies

The `AUDIT` statement with the `POLICY` clause enables a unified audit policy, applying for all types of audit options, including object-level options.

The policy does not take effect until after the audited users (or users who have been granted the roles associated with the policy) log into the database instance. In other words, if you create and enable a policy while the audited users are logged in, then the policy cannot collect audit data; the users must log out and then log in again before auditing can begin. Once the session is set up with auditing for it, then the setting lasts as long as the user session and then ends when the session ends.

You can enable the audit policy for individual users or for roles. Enabling the audit policy for roles allows you to enable the policy for a group of users who have been directly granted the role. When the role has been directly granted to a new user, then the policy automatically applies to the user. When the role is revoked from a user, then the policy no longer applies to the user.

You can check the results of the audit by querying the `UNIFIED_AUDIT_TRAIL` data dictionary view. To find a list of existing unified audit policies, query the `AUDIT_UNIFIED_POLICIES` data dictionary view.

The `AUDIT` statement lets you specify the following optional additional settings:

- **Whether to apply the unified audit policy to one or more users or roles.**To apply the policy to one or more users or roles, including administrative users who log in with the SYSDBA administrative privilege (such as SYS), use the BY clause. For example, to apply the policy to users SYS and SYSTEM:

For example, to apply the policy to two users:

```
AUDIT POLICY role_connect_audit_pol BY SYS, SYSTEM;
```

To apply a policy to users who have been directly granted the DBA and CDB_DBA roles:

```
AUDIT POLICY admin_audit_pol BY USERS WITH GRANTED ROLES DBA, CDB_DBA;
```

- **Whether to exclude users from the unified audit policy.** To exclude users from the audit policy, include the EXCEPT clause.

  For example:

  ```
  AUDIT POLICY role_connect_audit_pol EXCEPT rlee, jrandolph;
  ```

- **Whether to create an audit record if the activity succeeds or fails.** This method of auditing reduces the audit trail, helping you to focus on specific actions. This can aid in maintaining good database performance. Enter one of the following clauses:
  - WHENEVER SUCCESSFUL audits only successful executions of the user's activity.
  - WHENEVER NOT SUCCESSFUL audits only failed executions of the user's activity. Monitoring unsuccessful SQL statement can expose users who are snooping or acting maliciously, though most unsuccessful SQL statements are neither.

  For example:

  ```
  AUDIT POLICY role_connect_audit_pol WHENEVER NOT SUCCESSFUL;
  ```

  If you omit this clause, then both failed and successful user activities are written to the audit trail.

Note the following:

- The unified audit policy only can have either the BY, BY USERS WITH GRANTED ROLES, or the EXCEPT clause, but not more than one of these clauses for the same policy.
- If you run multiple AUDIT statements on the same unified audit policy but specify different BY users or different BY USERS WITH GRANTED ROLES roles, then Oracle Database audits all of these users or roles.
- If you run multiple AUDIT statements on the same unified audit policy but specify different EXCEPT users, then Oracle Database uses the last exception user list, not any of the users from the preceding lists. This means the effect of the earlier AUDIT POLICY ... EXCEPT statements are overridden by the latest AUDIT POLICY ... EXCEPT statement.
- You cannot use the EXCEPT clause for roles. It applies to users only.
- You can only enable common unified audit policies for common users or roles.
- In a multitenant environment, you can enable a common audit policy only from the root and a local audit policy only from the PDB to which it applies.

Enabling a Unified Audit Policy

The AUDIT POLICY statement can enable a unified audit policy.

- Use the following syntax to enable a unified audit policy:

```
AUDIT POLICY { policy_auditing }
 [WHENEVER [NOT] SUCCESSFUL]
```

In this specification:

- `policy_auditing` refers to the following components:
  - **The name of the unified audit policy.** To find all existing policies, query the `AUDIT_UNIFIED_POLICIES` data dictionary view. To find currently enabled policies, query `AUDIT_UNIFIED_ENABLED_POLICIES`.
  - **Users or roles to whom the unified audit policy applies.** To apply the policy to one or more users (including user `SYS`), enter the `BY` clause. For example:

    ```
    BY psmith, rlee
    ```

    To apply the policy to one or more users to whom the list of roles are directly granted, use the `BY USERS WITH GRANTED ROLES` clause. For example:

    ```
    BY USERS WITH GRANTED ROLES HS_ADMIN_ROLE, HS_ADMIN_SELECT_ROLE
    ```
  - **Users to exclude from the unified audit policy.** To exclude one or more users from the policy, enter the `EXCEPT` clause. For example:

    ```
    EXCEPT psmith, rlee
    ```

    Mandatory audit records are captured in the `UNIFIED_AUDIT_TRAIL` data dictionary view for the `AUDIT POLICY` SQL statement. To find users who have been excluded in the audit records, you can query the `EXCLUDED_USER` column in the `UNIFIED_AUDIT_TRAIL` view to list the excluded users.

  You cannot enable the same audit policy with the `BY`, `BY USERS WITH GRANTED ROLES`, and `EXCEPT` clauses in the same statement. This action throws an error for the subsequent `AUDIT` statement with the conflicting clause
- `WHENEVER [NOT] SUCCESSFUL` enables the policy to generate audit records based on whether the user's actions failed or succeeded. See About Enabling Unified Audit Policies for more information.

After you enable the unified audit policy and it is generating records, you can find the audit records by querying the `UNIFIED_AUDIT_TRAIL` data dictionary view.

Example: Enabling a Unified Audit Policy

The `AUDIT POLICY` statement can enable a unified audit policy using conditions, such as `WHENEVER NOT SUCCESSFUL`.

Example 24-41 shows how to enable a unified audit policy to record only failed actions by the user `dv_admin`.

Example 24-41 Enabling a Unified Audit Policy

```
AUDIT POLICY dv_admin_pol BY tjones
 WHENEVER NOT SUCCESSFUL;
```

Disabling Unified Audit Policies

You can use the `NOAUDIT POLICY` statement to disable a unified audit policy.

About Disabling Unified Audit Policies

The `NOAUDIT` statement with the `POLICY` clause can disable a unified audit policy.

In the `NOAUDIT` statement, you can specify a `BY` user or `BY USERS WITH GRANTED ROLES` role list, but not an `EXCEPT` user list. The disablement of a unified audit policy takes effect on subsequent user sessions.

You can find a list of existing unified audit policies by querying the `AUDIT_UNIFIED_POLICIES` data dictionary view.

In a multitenant environment, you can disable a common audit policy only from the root and a local audit policy only from the PDB to which it applies.

Disabling a Unified Audit Policy

The `NOAUDIT` statement can disable a unified audit policy using supported audit options.

- Use the following syntax to disable a unified audit policy:

  ```
  NOAUDIT POLICY {policy_auditing | existing_audit_options};
  ```

  In this specification:

  – *`policy_auditing`* is the name of the policy. To find all currently enabled policies, query the `AUDIT_UNIFIED_ENABLED_POLICIES` data dictionary view. As part of this specification, you optionally can include the `BY` or `BY USERS WITH GRANTED ROLES` clause, but not the `EXCEPT` clause. See About Enabling Unified Audit Policies for more information.

  – *`existing_audit_options`* refers to `AUDIT` options that were available in releases earlier than Oracle Database 12*c* release 1 (12.1), such as the following:

    * `SELECT ANY TABLE, UPDATE ANY TABLE BY SCOTT, HR`

    * `UPDATE ON SCOTT.EMP`

If the unified policy had been applied to all users, then you only need to specify the policy name. For example:

```
NOAUDIT POLICY logons_pol;
```

Example: Disabling a Unified Audit Policy

The `NOAUDIT POLICY` statement disable a unified audit policy using filtering, such as by user name.

Example 24-42 shows examples of how to disable a unified audit policy for a user and for a role.

Example 24-42 Disabling a Unified Audit Policy

```
NOAUDIT POLICY dv_admin_pol BY tjones;

NOAUDIT POLICY dv_admin_pol BY USERS WITH GRANTED ROLES emp_admin;
```

Dropping Unified Audit Policies

You can use the `DROP AUDIT POLICY` statement to drop a unified audit policy.

About Dropping Unified Audit Policies

The `DROP AUDIT POLICY` statement can be used to unified audit policies.

If a unified audit policy is already enabled for a session, the effect of dropping the policy is not seen by this existing session. Until that time, the unified audit policy's settings remain in effect. For object-related unified audit policies, however, the effect is immediate.

You can find a list of existing unified audit policies by querying the `AUDIT_UNIFIED_POLICIES` data dictionary view.

When you disable an audit policy before dropping it, ensure that you disable it using the same settings that you used to enable it. For example, suppose you enabled the `logon_pol` policy as follows:

```
AUDIT POLICY logon_pol BY HR, OE;
```

Before you can drop it, your `NOAUDIT` statement must include the `HR` and `OE` users as follows:

```
NOAUDIT POLICY logon_pol BY HR, OE;
```

In a multitenant environment, you can drop a common audit policy only from the root and a local audit policy only from the PDB to which it applies.

Dropping a Unified Audit Policy

To drop a unified audit policy, you must first disable it, and then run the `DROP AUDIT POLICY` statement to remove it.

- Use the following the following syntax to drop a unified audit policy:

  ```
  DROP AUDIT POLICY policy_name;
  ```

In a multitenant environment, the unified audit policy drop applies to the current PDB. If the unified audit policy was created as a common unified audit policy, then you cannot drop it from the local PDB.

Example: Disabling and Dropping a Unified Audit Policy

The `NOAUDIT POLICY` and `DROP AUDIT POLICY` statements can disable and drop a unified audit policy.

Example 24-43 shows how to disable and drop a common unified audit policy.

Example 24-43 Disabling and Dropping a Unified Audit Policy

```
CONNECT c##sec_admin
Enter password: password
Connected.

NOAUDIT POLICY dv_admin_pol;

DROP AUDIT POLICY dv_admin_pol
```

Tutorial: Auditing Nondatabase Users

This tutorial shows how to create a unified audit policy that uses a client identifier to audit a nondatabase user's actions.

Step 1: Create the User Accounts and Ensure the User OE Is Active

You must first create users and ensure that the user OE is active.

1. Log on as user SYS with the SYSDBA administrative privilege.

   ```
   sqlplus sys as sysdba
   Enter password: password
   ```

2. In a multitenant environment, connect to the appropriate PDB.

   For example:

   ```
   CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

3. Create the local user `policy_admin`, who will create the fine-grained audit policy.

   ```
   CREATE USER policy_admin IDENTIFIED BY password;
   GRANT CREATE SESSION, AUDIT_ADMIN TO policy_admin;
   ```

   Follow the guidelines in Minimum Requirements for Passwords to replace *password* with a password that is secure.

4. Create the local user account `auditor`, who will check the audit trail for this policy.

   ```
   CREATE USER policy_auditor IDENTIFIED BY password;
   GRANT CREATE SESSION, AUDIT_VIEWER TO policy_auditor;
   ```

5. The sample user OE will also be used in this tutorial, so query the DBA_USERS data dictionary view to ensure that OE is not locked or expired.

   ```
   SELECT USERNAME, ACCOUNT_STATUS FROM DBA_USERS WHERE USERNAME = 'OE';
   ```

   The account status should be OPEN. If the DBA_USERS view lists user OE as locked and expired, log in as user SYSTEM and then enter the following statement to unlock the OE account and create a new password:

   ```
   ALTER USER OE ACCOUNT UNLOCK IDENTIFIED BY password;
   ```

   Follow the guidelines in Minimum Requirements for Passwords to replace *password* with a password that is secure. For greater security, do **not** give the OE account the same password from previous releases of Oracle Database.

Step 2: Create the Unified Audit Policy

Next, you are ready to create the unified audit policy.

1. Connect to SQL*Plus as user `policy_admin`.

   ```
   CONNECT policy_admin -- Or, CONNECT policy_admin@hrpdb
   Enter password: password
   ```

2. Create the following policy:

   ```
   CREATE AUDIT POLICY orders_unified_audpol
     ACTIONS INSERT ON OE.ORDERS, UPDATE ON OE.ORDERS, DELETE ON OE.ORDERS, SELECT
   ON OE.ORDERS
     WHEN 'SYS_CONTEXT(''USERENV'', ''CLIENT_IDENTIFIER'') = ''robert'''
       EVALUATE PER STATEMENT;

   AUDIT POLICY orders_unified_audpol;
   ```

   In this example, the `AUDIT_CONDITION` parameter assumes that the nondatabase user is named `robert`. The policy will monitor any `INSERT`, `UPDATE`, `DELETE`, and `SELECT` statements that `robert` will attempt. Remember that the user's `CLIENT_IDENTITIFER` setting that you enter in the policy is case sensitive and that the policy only recognizes the case used for the identity that you specify here. In other words, later on, if the user session is set to `Robert` or `ROBERT`, the policy's condition will not be satisfied.

Step 3: Test the Policy

To test the policy, use `OE` must try to select from the `OE.ORDERS` table.

A unified auditing policy takes effect in the next user session for the users who are being audited. So, before their audit records can be captured, the users must connect to the database *after* the policy has been created.

1. Connect as user `OE` and select from the `OE.ORDERS` table.

   ```
   CONNECT OE -- Or, CONNECT OE@hrpdb
   Enter password: password

   SELECT COUNT(*) FROM ORDERS;
   ```

   The following output appears:

   ```
     COUNT(*)
   ----------
          105
   ```

2. Connect as user `policy_auditor` and then check if any audit records were generated.

   ```
   CONNECT policy_auditor -- Or, CONNECT policy_auditor@hrpdb
   Enter password: password

   col dbusername format a10
   col client_identifier format a20
   col sql_text format a29

   SELECT DBUSERNAME, CLIENT_IDENTIFIER, SQL_TEXT FROM UNIFIED_AUDIT_TRAIL
    WHERE SQL_TEXT LIKE '%FROM ORDERS%';
   ```

   The following output appears:

   ```
   no rows selected
   ```

3. Reconnect as user `OE`, set the client identifier to `robert`, and then reselect from the `OE.ORDERS` table.

   ```
   CONNECT OE -- Or, CONNECT OE@hrpdb
   Enter password: password
   ```

```
EXEC DBMS_SESSION.SET_IDENTIFIER('robert');

SELECT COUNT(*) FROM ORDERS;
```

The following output should appear:

```
  COUNT(*)
----------
       105
```

4. Reconnect as user `auditor` and then check the audit trail again.

   ```
   CONNECT policy_auditor -- Or, CONNECT policy_auditor@hrpdb
   Enter password: password

   SELECT DBUSERNAME, CLIENT_IDENTIFIER, SQL_TEXT FROM UNIFIED_AUDIT_TRAIL
    WHERE SQL_TEXT LIKE '%FROM ORDERS%';
   ```

   This time, because `robert` has made his appearance and queried the `OE.ORDERS` table, the audit trail captures his actions:

   ```
   DBUSERNAME CLIENT_IDENTIFIER SQL_TEXT
   ---------- ----------------- ----------------------------
   OE         robert            SELECT COUNT(*) FROM ORDERS;
   ```

Step 4: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. Connect to SQL*Plus as user `policy_admin`, and then manually disable and drop the `orders_unified_audpol` policy.

   ```
   CONNECT policy_admin -- Or, CONNECT policy_admin@hrpdb
   Enter password: password

   NOAUDIT POLICY orders_unified_audpol;
   DROP AUDIT policy orders_unified_audpol;
   ```

   (Unified audit policies reside in the `SYS` schema, not the schema of the user who created them.)

2. Connect to SQL*Plus as user `SYSTEM`.

   ```
   CONNECT SYSTEM -- Or, CONNECT SYSTEM@hrpdb
   Enter password: password
   ```

3. Drop users `policy_admin` and `policy_auditor`.

   ```
   DROP USER policy_admin;
   DROP USER policy_auditor;
   ```

4. If you want, lock and expire `OE`, unless other users want to use this account:

   ```
   ALTER USER OE PASSWORD EXPIRE ACCOUNT LOCK;
   ```

Auditing Activities with the Predefined Unified Audit Policies

Oracle Database provides predefined unified audit policies that cover commonly used security-relevant audit settings.

Logon Failures Predefined Unified Audit Policy

The `ORA_LOGON_FAILURES` unified audit policy tracks failed logons only, but not any other kinds of logons.

For new databases, this policy is enabled by default for both pure unified auditing and mixed-mode auditing environments. This policy is not enabled for databases that were upgraded from earlier versions, except if you have created a new database from the previous release and then upgrade it to the current release.

The following `CREATE AUDIT POLICY` statement shows the `ORA_LOGON_FAILURES` unified audit policy definition:

```
CREATE AUDIT POLICY ORA_LOGON_FAILURES ACTIONS LOGON;
```

You should enable the `ORA_LOGON_FAILURES` unified audit policy as follows:

```
AUDIT POLICY ORA_LOGON_FAILURES WHENEVER NOT SUCCESSFUL;
```

Secure Options Predefined Unified Audit Policy

The `ORA_SECURECONFIG` unified audit policy provides all the secure configuration audit options.

For new databases, this policy is enabled by default for both pure unified auditing and mixed-mode auditing environments. This policy is not enabled for databases that were upgraded from earlier versions, except if you have created a new database from the previous release and then upgrade it to the current release.

The following `CREATE AUDIT POLICY` statement shows the `ORA_SECURECONFIG` unified audit policy definition.

```
CREATE AUDIT POLICY ORA_SECURECONFIG
 PRIVILEGES ALTER ANY TABLE, CREATE ANY TABLE, DROP ANY TABLE,
            CREATE ANY PROCEDURE, DROP ANY PROCEDURE, ALTER ANY PROCEDURE,
            GRANT ANY PRIVILEGE, GRANT ANY OBJECT PRIVILEGE, GRANT ANY ROLE,
            AUDIT SYSTEM, CREATE EXTERNAL JOB, CREATE ANY JOB,
            CREATE ANY LIBRARY,
            EXEMPT ACCESS POLICY,
            CREATE USER, DROP USER,
            ALTER DATABASE, ALTER SYSTEM,
            CREATE PUBLIC SYNONYM, DROP PUBLIC SYNONYM,
            CREATE SQL TRANSLATION PROFILE, CREATE ANY SQL TRANSLATION
PROFILE,
            DROP ANY SQL TRANSLATION PROFILE, ALTER ANY SQL TRANSLATION
PROFILE,
            TRANSLATE ANY SQL,
            EXEMPT REDACTION POLICY,
            PURGE DBA_RECYCLEBIN, LOGMINING,
            ADMINISTER KEY MANAGEMENT, BECOME USER
 ACTIONS    ALTER USER, CREATE ROLE, ALTER ROLE, DROP ROLE,
            SET ROLE, CREATE PROFILE, ALTER PROFILE,
            DROP PROFILE, CREATE DATABASE LINK,
            ALTER DATABASE LINK, DROP DATABASE LINK,
            CREATE DIRECTORY, DROP DIRECTORY,
            CREATE PLUGGABLE DATABASE,
            DROP PLUGGABLE DATABASE,
            ALTER PLUGGABLE DATABASE,
            EXECUTE ON DBMS_RLS,
            ALTER DATABASE DICTIONARY;
```

Oracle Database Parameter Changes Predefined Unified Audit Policy

The `ORA_DATABASE_PARAMETER` policy audits commonly used Oracle Database parameter settings.

The following `CREATE AUDIT POLICY` statement shows the `ORA_DATABASE_PARAMETER` unified audit policy definition. By default, this policy is not enabled.

```
CREATE AUDIT POLICY ORA_DATABASE_PARAMETER
 ACTIONS ALTER DATABASE, ALTER SYSTEM, CREATE SPFILE;
```

User Account and Privilege Management Predefined Unified Audit Policy

The `ORA_ACCOUNT_MGMT` policy audits commonly used user account and privilege settings.

The following `CREATE AUDIT POLICY` statement shows the `ORA_ACCOUNT_MGMT` unified audit policy definition. By default, this policy is not enabled.

```
CREATE AUDIT POLICY ORA_ACCOUNT_MGMT
 ACTIONS CREATE USER, ALTER USER, DROP USER, CREATE ROLE, DROP ROLE,
  ALTER ROLE, SET ROLE, GRANT, REVOKE;
```

Center for Internet Security Recommendations Predefined Unified Audit Policy

The `ORA_CIS_RECOMMENDATIONS` policy performs audits that the Center for Internet Security (CIS) recommends.

The following `CREATE AUDIT POLICY` statement shows the `ORA_CIS_RECOMMENDATIONS` unified audit policy definition. By default, this policy is not enabled.

```
CREATE AUDIT POLICY ORA_CIS_RECOMMENDATIONS
PRIVILEGES SELECT ANY DICTIONARY, ALTER SYSTEM
ACTIONS CREATE USER, ALTER USER, DROP USER,
        CREATE ROLE, DROP ROLE, ALTER ROLE,
        GRANT, REVOKE, CREATE DATABASE LINK,
        ALTER DATABASE LINK, DROP DATABASE LINK,
        CREATE PROFILE, ALTER PROFILE, DROP PROFILE,
        CREATE SYNONYM, DROP SYNONYM,
        CREATE PROCEDURE, DROP PROCEDURE,
        ALTER PROCEDURE, ALTER SYNONYM, CREATE FUNCTION,
        CREATE PACKAGE, CREATE PACKAGE BODY,
        ALTER FUNCTION, ALTER PACKAGE, ALTER SYSTEM,
        ALTER PACKAGE BODY, DROP FUNCTION,
        DROP PACKAGE, DROP PACKAGE BODY,
        CREATE TRIGGER, ALTER TRIGGER,
        DROP TRIGGER;
```

Oracle Database Real Application Security Predfined Audit Policies

You can use predefined unified audit policies for Oracle Database Real Application Security events.

System Administrator Operations Predefined Unified Audit Policy

The ORA_RAS_POLICY_MGMT predefined unified audit policy audits policies for all Oracle Real Application Security administrative actions on application users, roles, and policies.

The following CREATE AUDIT POLICY statement describes the ORA_RAS_POLICY_MGMT audit policy. By default, this policy is not enabled.

```
CREATE AUDIT POLICY ORA_RAS_POLICY_MGMT
 ACTIONS COMPONENT=XS
  CREATE USER, UPDATE USER, DELETE USER,
  CREATE ROLE, UPDATE ROLE, DELETE ROLE, GRANT ROLE, REVOKE ROLE,
  ADD PROXY, REMOVE PROXY,
  SET USER PASSWORD, SET USER VERIFIER, SET USER PROFILE,
  CREATE ROLESET, UPDATE ROLESET, DELETE ROLESET,
  CREATE SECURITY CLASS, UPDATE SECURITY CLASS, DELETE SECURITY CLASS,
  CREATE NAMESPACE TEMPLATE, UPDATE NAMESPACE TEMPLATE, DELETE NAMESPACE TEMPLATE,
  CREATE ACL, UPDATE ACL, DELETE ACL,
  CREATE DATA SECURITY, UPDATE DATA SECURITY, DELETE DATA SECURITY,
  ENABLE DATA SECURITY, DISABLE DATA SECURITY,
  ADD GLOBAL CALLBACK, DELETE GLOBAL CALLBACK, ENABLE GLOBAL CALLBACK;
```

Session Operations Predefined Unified Audit Policy

The ORA_RAS_SESSION_MGMT predefined unified audit policy audits policies for all run-time Oracle Real Application Security session actions and namespace actions.

The following CREATE AUDIT POLICY statement describes the ORA_RAS_SESSION_MGMT policy. By default, this policy is not enabled.

```
CREATE AUDIT POLICY ORA_RAS_SESSION_MGMT
 ACTIONS COMPONENT=XS
  CREATE SESSION, DESTROY SESSION,
  ENABLE ROLE, DISABLE ROLE,
  SET COOKIE, SET INACTIVE TIMEOUT,
  SWITCH USER, ASSIGN USER,
  CREATE SESSION NAMESPACE, DELETE SESSION NAMESPACE,
  CREATE NAMESPACE ATTRIBUTE, GET NAMESPACE ATTRIBUTE, SET NAMESPACE ATTRIBUTE,
  DELETE NAMESPACE ATTRIBUTE;
```

Oracle Database Vault Predefined Unified Audit Policy for DVSYS and LBACSYS Schemas

The ORA_DV_AUDPOL predefined unified audit policy audits Oracle Database Vault DVSYS and LBACSYS schema objects.

The ORA_DV_AUDPOL policy audits all actions that are performed on the Oracle Database Vault DVSYS (including DVF) schema objects and the Oracle Label Security LBACSYS schema objects. It does not capture actions on the F$* factor functions in the DVF schema. By default, this policy is not enabled.

To view the complete definition of this policy, query the `AUDIT_UNIFIED_POLICIES` data dictionary view, where `policy_name` is `ORA_DV_AUDPOL`.

Oracle Database Vault Predefined Unified Audit Policy for Default Realms and Command Rules

The `ORA_DV_AUDPOL2` predefined unified audit policy audits the Oracle Database Vault default realms and command rules.

The `ORA_DV_AUDPOL2` policy constitutes the audit settings of the Oracle Database Vault-supplied default realms and command rules. By default, this policy is not enabled.

To view the complete definition of this policy, query the `AUDIT_UNIFIED_POLICIES` data dictionary view, where `policy_name` is `ORA_DV_AUDPOL2`.

Auditing Specific Activities with Fine-Grained Auditing

Fine-grained auditing enables you to create audit policies at the granular level.

About Fine-Grained Auditing

Fine-grained auditing enables you to create policies that define specific conditions that must take place for the audit to occur.

You cannot create unified audit policies using fine-grained auditing but you can use fine-grained auditing to create very customized audit settings, such as auditing the times that data is accessed.

This enables you to monitor data access based on content. It provides granular auditing of queries, and `INSERT`, `UPDATE`, and `DELETE` operations. You can use fine-grained auditing to audit the following types of actions:

- Accessing a table between 9 p.m. and 6 a.m. or on Saturday and Sunday
- Using an IP address from outside the corporate network
- Selecting or updating a table column
- Modifying a value in a table column

In general, fine-grained audit policies are based on simple, user-defined SQL predicates on table objects as conditions for selective auditing. During fetching, whenever policy conditions are met for a row, the query is audited.

The audit policies described in Auditing Activities with Unified Audit Policies and the AUDIT Statement can perform most of the operations that fine-grained audit policies can perform, except for the following actions:

- **Auditing specific columns.** You can audit specific relevant columns that hold sensitive information, such as salaries or Social Security numbers.
- **Using event handlers.** For example, you can write a function that sends an email alert to a security administrator when an audited column that should not be changed at midnight is updated.

Note:

- Fine-grained auditing is supported only with cost-based optimization. For queries using rule-based optimization, fine-grained auditing checks before applying row filtering, which could result in an unnecessary audit event trigger.
- Policies currently in force on an object involved in a flashback query are applied to the data returned from the specified flashback snapshot based on time or system change number (SCN).
- If you want to use fine-grained auditing to audit data that is being directly loaded (for example, using Oracle Warehouse Builder to execute DML statements), then Oracle Database transparently makes all direct loads that are performed in the database instance into conventional loads. If you want to preserve the direct loading of data, consider using unified audit policies instead.

Where Are Fine-Grained Audit Records Stored?

Fine-grained auditing records are stored in the `AUDSYS` schema.

These audit records are stored in the `SYSAUX` tablespace by default. You can supply a new tablespace by using the `DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION` procedure. This tablespace can be an encrypted tablespace. To find the records have been generated for the audit policies that are in effect, you can query `UNIFIED_AUDIT_TRAIL` data dictionary view.

The audit trail captures an audit record for each reference of a table or a view within a SQL statement. For example, if you run a `UNION` statement that references the `HR.EMPLOYEES` table twice, then an audit policy for statement generates two audit records, one for each access of the `HR.EMPLOYEES` table.

See Also:

- Activities That Are Mandatorily Audited
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION` procedure
- *Oracle Database Reference* for more information about the `UNIFIED_AUDIT_TRAIL` data dictionary view

Who Can Perform Fine-Grained Auditing?

Oracle provides roles for privileges needed to create fine-grained audit policies and to view and analyze fine-grained audit policy data.

The fine-grained audit privileges are as follows:

- To create fine-grained audit policies, you must be granted d the `AUDIT_ADMIN` role or the `EXECUTE` privilege on the `DBMS_FGA` package.
- To view and analyze fine-grained audit data, you must be granted the `AUDIT_VIEWER` role.

The PL/SQL package is already granted to `AUDIT_ADMIN` role. As with all privileges, grant these roles to trusted users only. You can find the roles that user have been granted by querying the `DBA_ROLE_PRIVS` data dictionary view.

Fine-Grained Auditing on Tables or Views That Have Oracle VPD Policies

The audit trail captures the VPD predicate for fine-grained audited tables or views that are included in an Oracle VPD policy.

This behavior is similar to how the unified audit trail captures the VPD predicate for unified audit policies.

The audit trail also captures internal predicates from Oracle Label Security and Oracle Real Application Security policies.

You do not need to create a special audit policy to capture the VPD predicate audit records. The predicate information is automatically stored in the `RLS_INFO` column of the `DBA_FGA_AUDIT_TRAIL` and `UNIFIED_AUDIT_TRAIL` data dictionary views.

If there are multiple VPD policies applied to the same table or view, then by default the predicates for these policies are concatenated in the `RLS_INFO` column. You can reformat the output so that each predicate is in its own row (identified by its corresponding VPD policy name and other information) by using the functions in the `DBMS_AUDIT_UTIL` PL/SQL package.

See Also:

- Auditing of Oracle Virtual Private Database Predicates for more information about the auditing of VPD predicates and for an example of how to use the `DBMS_AUDIT_UTIL` package functions to format captured audit data
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_AUDIT_UTIL` PL/SQL package

Fine-Grained Auditing in a Multitenant Environment

You can create fine-grained audit policies in the CDB root, application root, CDB PDBs, and application PDBs.

Note the following general rules about fine-grained audit policies in a multitenant environment:

- You cannot create fine-grained audit policies on `SYS` objects.
- You cannot create fine-grained audit policies, either local or application common, for extended data link objects.

- When you create a fine-grained audit policy in the CDB root, the policy cannot be applied to all PDBs. It only applies to objects within the CDB root. (In other words, there is no such thing as a common fine-grained audit policy for the CDB root.) If you want to create a fine-grained audit policy to audit a common object's access in all the PDBs, then you must explicitly create that policy in each PDB and then enable it on the common objects that is accessible in the PDB.
- When you create a fine-grained audit policy in a PDB, it applies only to objects within the PDB.
- You can create application common fine-grained audit policies only if you are connected to the application root and only within the `BEGIN`/`END` block. If you are connected to the application root and create the fine-grained audit policy outside the `BEGIN`/`END` block, then the fine-grained audit policy is created in the application root.
- You cannot create application common fine-grained audit policies on local PDB objects.
- If the application common fine-grained audit policy has a handler, then this handler must be owned by either an application common user or a CDB common user.
- You can create an application fine-grained audit policy on local (PDB) objects and CDB common objects. Because the policy is local to its container, the object on which the policy is defined is audited only in the particular container where the policy is defined. For example, if you create a fine-grained audit policy in the `hr_pdb` PDB, the object for which you create this policy must exist in the `hr_pdb` PDB.
- You cannot create local fine-grained audit policies in an application PDB on object linked and extended data link objects. On metadata-linked objects are allowed in the fine-grained audit policy.
- Application root local policies are allowed for all application common objects.
- When you create a fine-grained audit policy as a common audit policy in an application root, it will be effective in each PDB that belongs to this application root. Therefore, any access to the application common object and CDB common object (on which the application common fine-grained audit policy is defined) from the application PDB is audited in the fine-grained audit trail in that application PDB.
- When you create scripts for application install, upgrade, patch, or uninstall operations, you can include SQL statements within the `ALTER PLUGGABLE DATABASE` *`app_name`* `BEGIN INSTALL` and `ALTER PLUGGABLE DATABASE` *`app_name`* `END INSTALL` blocks to perform various operations. You can include fine-grained audit policy statements only within these blocks.
- You can only enable, disable, or drop application common fine-grained audit policies from the application root, and from within a `ALTER PLUGGABLE DATABASE` *`app_name`* `BEGIN INSTALL` and `ALTER PLUGGABLE DATABASE` *`app_name`* `END INSTALL` block in a script.

Fine-Grained Audit Policies with Editions

You can prepare an application for edition-based redefinition, and cover each table that the application uses with an editioning view.

If you do this, then you must move the fine-grained audit polices that protect these tables to the editioning view. You can find information about the currently configured editions by querying the `DBA_EDITIONS` data dictionary view. To find information about fine-grained audit policies, query `DBA_AUDIT_POLICIES`.

Using the DBMS_FGA PL/SQL Package to Manage Fine-Grained Audit Policies

The `DBMS_FGA` PL/SQL package manages fine-grained audit policies.

About the DBMS_FGA PL/SQL PL/SQL Package

The `DBMS_FGA` PL/SQL package can be used to combine statements into one policy and perform other fine-grained auditing management tasks.

However, unless you want to perform column-level auditing or use event handlers with your audit policy, you should create audit policies as described in Auditing Activities with Unified Audit Policies and the AUDIT Statement.

The `DBMS_FGA` PL/SQL package enables you to add all combinations of `SELECT`, `INSERT`, `UPDATE`, and `DELETE` statements to one policy. You also can audit `MERGE` statements, by auditing the underlying actions of `INSERT` and `UPDATE`. To audit `MERGE` statements, configure fine-grained access on the `INSERT` and `UPDATE` statements. Only one record is generated for each policy for successful `MERGE` operations.

To administer fine-grained audit policies, you must have be granted the `AUDIT_ADMIN` role. Note also that the `EXECUTE` privilege for the `DBMS_FGA` package is mandatorily audited.

The audit policy is bound to the table for which you created it. This simplifies the management of audit policies because the policy only needs to be changed once in the database, not in each application. In addition, no matter how a user connects to the database—from an application, a Web interface, or through SQL*Plus or Oracle SQL Developer—Oracle Database records any actions that affect the policy.

If any rows returned from a query match the audit condition that you define, then Oracle Database inserts an audit entry into the fine-grained audit trail. This entry excludes all the information that is reported in the regular audit trail. In other words, only one row of audit information is inserted into the audit trail for every fine-grained audit policy that evaluates to true.

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `DBMS_FGA` package

The DBMS_FGA PL/SQL Package with Editions

You can create `DBMS_FGA` policies for use in an editions environment.

If you plan to use the `DBMS_FGA` package policy across different editions, then you can control the results of the policy: whether the results are uniform across all editions, or specific to the edition in which the policy is used.

The DBMS_FGA PL/SQL Package in a Multitenant Environment

In a multitenant environment, the `DBMS_FGA` PL/SQL package applies only to the current local PDBs.

You cannot create one policy for the entire multitenant environment. The policy must be specific to objects within a PDB. To find PDBs, you can query the `DBA_PDBS` data dictionary view. To find the name of the current PDB, issue the `show con_name` command.

Creating a Fine-Grained Audit Policy

The `DBMS_FGA.ADD_POLICY` procedure creates a fine-grained audit policy.

About Creating a Fine-Grained Audit Policy

The `DBMS_FGA.ADD_POLICY` procedure creates an audit policy using the supplied predicate as the audit condition.

By default, Oracle Database executes the policy predicate with the privileges of the user who owns the policy. The maximum number of fine-grained policies on any table or view object is 256. Oracle Database stores the policy in the data dictionary table, but you can create the policy on any table or view that is not in the `SYS` schema. In a multitenant environment, the fine grained policy is only created in the local PDB.

You cannot modify a fine-grained audit policy after you have created it. If you must modify the policy, then drop and recreate it.

You can find information about a fine-grained audit policy by querying the `ALL_AUDIT_POLICIES`, `DBA_AUDIT_POLICIES`, and `ALL_AUDIT_POLICIES` views. The `UNIFIED_AUDIT_TRAIL` view contains a column entitled `FGA_POLICY_NAME`, which you can use to filter out rows that were generated using a specific fine-grained audit policy.

Syntax for Creating a Fine-Grained Audit Policy

The `DBMS_FGA.ADD_POLICY` procedure includes many settings, such as the ability to use a handler for complex auditing.

The `DBMS_FGA.ADD_POLICY` procedure syntax is as follows:

```
DBMS_FGA.ADD_POLICY(
   object_schema      IN  VARCHAR2 DEFAULT NULL
   object_name        IN  VARCHAR2,
   policy_name        IN  VARCHAR2,
   audit_condition    IN  VARCHAR2 DEFAULT NULL,
   audit_column       IN  VARCHAR2 DEFAULT NULL
   handler_schema     IN  VARCHAR2 DEFAULT NULL,
   handler_module     IN  VARCHAR2 DEFAULT NULL,
   enable             IN  BOOLEAN DEFAULT TRUE,
   statement_types    IN  VARCHAR2 DEFAULT SELECT,
   audit_trail        IN  BINARY_INTEGER DEFAULT NULL,
   audit_column_opts  IN  BINARY_INTEGER DEFAULT ANY_COLUMNS,
   policy_owner       IN  VARCHAR2 DEFAULT NULL);
```

In this specification:

- `object_schema` specifies the schema of the object to be audited. (If `NULL`, the current log-on user schema is assumed.)
- `object_name` specifies the name of the object to be audited.
- `policy_name` specifies the name of the policy to be created. Ensure that this name is unique.
- `audit_condition` specifies a Boolean condition in a row. `NULL` is allowed and acts as `TRUE`. See Audits of Specific Columns and Rows for more information. If you specify `NULL` or no audit condition, then any action on a table with that policy creates an audit record, whether or not rows are returned.

  Follow these guidelines:

  – Do not include functions, which execute the auditable statement on the same base table, in the `audit_condition` setting. For example, suppose you create a function that executes an `INSERT` statement on the `HR.EMPLOYEES` table. The policy's `audit_condition` contains this function and it is for `INSERT` statements (as set by `statement_types`). When the policy is used, the function executes recursively until the system has run out of memory. This can raise the error `ORA-1000: maximum open cursors exceeded` or `ORA-00036: maximum number of recursive SQL levels (50) exceeded`.

  – Do not issue the `DBMS_FGA.ENABLE_POLICY` or `DBMS_FGA.DISABLE_POLICY` statement from a function in a policy's condition.
- `audit_column` specifies one or more columns to audit, including hidden columns. If set to `NULL` or omitted, all columns are audited. These can include Oracle Label Security hidden columns or object type columns. The default, `NULL`, causes audit if any column is accessed or affected.
- `handler_schema`: If an alert is used to trigger a response when the policy is violated, specifies the name of the schema that contains the event handler. The default, `NULL`, uses the current schema. See also Tutorial: Adding an Email Alert to a Fine-Grained Audit Policy.
- `handler_module` specifies the name of the event handler. Include the package the event handler is in. This function is invoked only after the first row that matches the audit condition in the query is processed.

  Follow these guidelines:

  – Do not create recursive fine-grained audit handlers. For example, suppose you create a handler that executes an `INSERT` statement on the `HR.EMPLOYEES` table. The policy that is associated with this handler is for `INSERT` statements (as set by the `statement_types` parameter). When the policy is used, the handler executes recursively until the system has run out of memory. This can raise the error `ORA-1000: maximum open cursors exceeded` or `ORA-00036: maximum number of recursive SQL levels (50) exceeded`.

  – Do not issue the `DBMS_FGA.ENABLE_POLICY` or `DBMS_FGA.DISABLE_POLICY` statement from a policy handler. Doing so can raise the `ORA-28144: Failed to execute fine-grained audit handler` error.
- `enable` enables or disables the policy using true or false. If omitted, the policy is enabled. The default is `TRUE`.
- `statement_types`: Specifies the SQL statements to be audited: `INSERT`, `UPDATE`, `DELETE`, or `SELECT` only. If you want to audit a `MERGE` operation, then set `statement_types` to `'INSERT,UPDATE'`. The default is `SELECT`.

- `audit_trail`: If you have migrated to unified auditing, then Oracle Database ignores this parameter and writes the audit records immediately to the unified audit trail. If you have migrated to unified auditing, then omit this parameter.

  Be aware that sensitive data, such as credit card information, can be recorded in clear text.
- `audit_column_opts`: If you specify more than one column in the `audit_column` parameter, then this parameter determines whether to audit all or specific columns. See Audits of Specific Columns and Rows for more information.
- `policy_owner` is the user who owns the fine-grained auditing policy. However, this setting is not a user-supplied argument. The Oracle Data Pump client uses this setting internally to recreate the fine-grained audit policies appropriately.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_FGA.ADD_POLICY` syntax

Audits of Specific Columns and Rows

You can fine-tune audit behavior by targeting a specific column (*relevant column*) to be audited if a condition is met.

To accomplish this, you use the `audit_column` parameter to specify one or more sensitive columns. In addition, you can audit data in specific rows by using the `audit_condition` parameter to define a Boolean condition. (However, if your policy needs only to audit for conditions, consider using an audit policy condition described in Creating a Condition for a Unified Audit Policy.)

The following settings from Example 24-44 enable you to perform an audit if anyone in Department 50 (`DEPARTMENT_ID = 50`) tries to access the `SALARY` and `COMMISSION_PCT` columns.

```
audit_condition    => 'DEPARTMENT_ID = 50',
audit_column       => 'SALARY,COMMISSION_PCT,'
```

As you can see, this feature is enormously beneficial. It not only enables you to pinpoint particularly important types of data to audit, but it provides increased protection for columns that contain sensitive data, such as Social Security numbers, salaries, patient diagnoses, and so on.

If the `audit_column` lists more than one column, then you can use the `audit_column_opts` parameter to specify whether a statement is audited when the query references *any* column specified in the `audit_column` parameter or only when *all* columns are referenced. For example:

```
audit_column_opts   => DBMS_FGA.ANY_COLUMNS,

audit_column_opts   => DBMS_FGA.ALL_COLUMNS,
```

If you do not specify a relevant column, then auditing applies to all columns.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `audit_condition`, `audit_column`, and `audit_column_opts` parameters in the `DBMS_FGA.ADD_POLICY` procedure (see also the usage notes for the `ADD_POLICY` procedure in that section)

Example: Using DBMS_FGA.ADD_POLICY to Create a Fine-Grained Audit Policy

The `DBMS_FGA.ADD_POLICY` procedure can create a fine-grained audit policy using multiple statement types.

Example 24-44 shows how to audit statements `INSERT`, `UPDATE`, `DELETE`, and `SELECT` on table `HR.EMPLOYEES`.

Note that this example omits the `audit_column_opts` parameter, because it is not a mandatory parameter.

Example 24-44 Using DBMS_FGA.ADD_POLICY to Create a Fine-Grained Audit Policy

```
BEGIN
  DBMS_FGA.ADD_POLICY(
   object_schema      => 'HR',
   object_name        => 'EMPLOYEES',
   policy_name        => 'chk_hr_employees',
   audit_column       => 'SALARY',
   enable             =>  TRUE,
   statement_types    => 'INSERT, UPDATE, SELECT, DELETE');
END;
/
```

After you create the policy, if you query the `DBA_AUDIT_POLICIES` view, you will find the new policy listed:

```
SELECT POLICY_NAME FROM DBA_AUDIT_POLICIES;

POLICY_NAME
-------------------------------
CHK_HR_EMPLOYEES
```

Afterwards, any of the following SQL statements log an audit event record.

```
SELECT COUNT(*) FROM HR.EMPLOYEES WHERE COMMISSION_PCT = 20 AND SALARY > 4500;

SELECT SALARY FROM HR.EMPLOYEES WHERE DEPARTMENT_ID = 50;

DELETE FROM HR.EMPLOYEES WHERE SALARY > 1000000;
```

Disabling a Fine-Grained Audit Policy

The `DBMS_FGA.DISABLE_POLICY` procedure disables a fine-grained audit policy.

- Use the following syntax to disable a fine-grained audit policy:

```
DBMS_FGA.DISABLE_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    VARCHAR2);
```

For example, to disable the fine-grained audit policy that was created in Example 24-44.

```
BEGIN
 DBMS_FGA.DISABLE_POLICY(
  object_schema       => 'HR',
  object_name         => 'EMPLOYEES',
  policy_name         => 'chk_hr_employees');
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `DISABLE_POLICY` syntax

Enabling a Fine-Grained Audit Policy

The `DBMS_FGA.ENABLE_POLICY` procedure enables a fine-grained audit policy.

- Use the following syntax to enable a fine-grained audit policy:

```
DBMS_FGA.ENABLE_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    VARCHAR2,
   enable         BOOLEAN);
```

For example, to reenable the `chk_hr_emp` policy by using the `DBMS_FGA.ENABLE_POLICY` procedure

```
BEGIN
 DBMS_FGA.ENABLE_POLICY(
  object_schema       => 'HR',
  object_name         => 'EMPLOYEES',
  policy_name         => 'chk_hr_employees',
  enable              => TRUE);
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `ENABLE_POLICY` syntax

Dropping a Fine-Grained Audit Policy

The `DBMS_FGA.DROP_POLICY` procedure drops a fine-grained audit policy.

Oracle Database automatically drops the audit policy if you remove the object specified in the `object_name` parameter of the `DBMS_FGA.ADD_POLICY` procedure, or if you drop the user who created the audit policy.

- Use the following syntax to drop a fine-grained audit policy:

  ```
  DBMS_FGA.DROP_POLICY(
     object_schema  VARCHAR2,
     object_name    VARCHAR2,
     policy_name    IVARCHAR2);
  ```

For example, to drop a fine-grained audit policy manually by using the `DBMS_FGA.DROP_POLICY` procedure:

```
BEGIN
 DBMS_FGA.DROP_POLICY(
  object_schema      => 'HR',
  object_name        => 'EMPLOYEES',
  policy_name        => 'chk_hr_employees');
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `DROP_POLICY` syntax

Tutorial: Adding an Email Alert to a Fine-Grained Audit Policy

This tutorial demonstrates how to create a fine-grained audit policy that generates an email alert when users violate the policy.

About This Tutorial

This tutorial shows how you can add an email alert to a fine-grained audit policy that goes into effect when a user (or an intruder) violates the policy.

Note:

- To complete this tutorial, you must use a database that has an SMTP server.
- If you are using a multitenant environment, then this tutorial applies to the current PDB only.

To add an email alert to a fine-grained audit policy, you first must create a procedure that generates the alert, and then use the following `DBMS_FGA.ADD_POLICY` parameters to call this function when someone violates this policy:

- `handler_schema`: The schema in which the handler event is stored
- `handler_module`: The name of the event handler

The alert can come in any form that best suits your environment: an email or pager notification, updates to a particular file or table, and so on. Creating alerts also helps to meet certain compliance regulations, such as California Senate Bill 1386. In this tutorial, you will create an email alert.

In this tutorial, you create an email alert that notifies a security administrator that a Human Resources representative is trying to select or modify salary information in the `HR.EMPLOYEES` table. The representative is permitted to make changes to this table, but to meet compliance regulations, we want to create a record of all salary selections and modifications to the table.

Step 1: Install and Configure the UTL_MAIL PL/SQL Package

The `UTL_MAIL` PL/SQL manages email that includes commonly used email features, such as attachments, CC, and BCC.

You must install and configure this package before you can use it. It is not installed and configured by default.

1. Log on as user `SYS` with the `SYSDBA` administrative privilege.

   ```
   sqlplus sys as sysdba
   Enter password: password
   ```

2. In a multitenant environment, connect to the appropriate PDB.

   For example:

   ```
   CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

3. Install the `UTL_MAIL` package.

   ```
   @$ORACLE_HOME/rdbms/admin/utlmail.sql
   @$ORACLE_HOME/rdbms/admin/prvtmail.plb
   ```

   The `UTL_MAIL` package enables you to manage email.

   Be aware that currently, the `UTL_MAIL` PL/SQL package does not support SSL servers.

4. Check the current value of the `SMTP_OUT_SERVER` initialization parameter, and make a note of this value so that you can restore it when you complete this tutorial.

   For example:

   ```
   SHOW PARAMETER SMTP_OUT_SERVER
   ```

   If the `SMTP_OUT_SERVER` parameter has already been set, then output similar to the following appears:

```
NAME                   TYPE              VALUE
---------------------- ----------------- ----------------------------------
SMTP_OUT_SERVER        string            some_imap_server.example.com
```

5. Issue the following `ALTER SYSTEM` statement:

   ```
   ALTER SYSTEM SET SMTP_OUT_SERVER="imap_mail_server.example.com";
   ```

   Replace *`imap_mail_server.example.com`* with the name of your SMTP server, which you can find in the account settings in your email tool. Enclose these settings in quotation marks. For example:

   ```
   ALTER SYSTEM SET SMTP_OUT_SERVER="my_imap_server.example.com";
   ```

6. Connect as `SYS` using the `SYSOPER` privilege and then restart the database.

   ```
   CONNECT SYS AS SYSOPER -- Or, CONNECT SYS@hrpdb AS SYSOPER
   Enter password: password

   SHUTDOWN IMMEDIATE
   STARTUP
   ```

7. Ensure that the `SMTP_OUT_SERVER` parameter setting is correct.

   ```
   CONNECT SYS AS SYSDBA -- Or, CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password

   SHOW PARAMETER SMTP_OUT_SERVER
   ```

   Output similar to the following appears:

   ```
   NAME                   TYPE              VALUE
   ---------------------- ----------------- ----------------------------------
   SMTP_OUT_SERVER        string            my_imap_server.example.com
   ```

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `UTL_MAIL` package

Step 2: Create User Accounts

You must create an administrative account and an auditor user.

1. Ensure that you are connected as `SYS` using the `SYSDBA` administrative privilege, and then create the `fga_admin` user, who will create the fine-grained audit policy.

   For example:

   ```
   CONNECT SYS AS SYSDBA -- Or, CONNECT SYS@hrpdb AS SYSDBA
   Enter password: password

   CREATE USER fga_admin IDENTIFIED BY password;
   GRANT CREATE SESSION, CREATE PROCEDURE, AUDIT_ADMIN TO fga_admin;
   GRANT EXECUTE ON UTL_TCP TO fga_admin;
   GRANT EXECUTE ON UTL_SMTP TO fga_admin;
   GRANT EXECUTE ON UTL_MAIL TO fga_admin;
   GRANT EXECUTE ON DBMS_NETWORK_ACL_ADMIN TO fga_admin;
   ```

Follow the guidelines in Minimum Requirements for Passwordsto replace *password* with a password that is secure.

The UTL_TCP, UTL_SMTP, UTL_MAIL, and DBMS_NETWORK_ACL_ADMIN PL/SQL packages are used by the email security alert that you create.

2. Create the auditor user, who will check the audit trail for this policy.

```
GRANT CREATE SESSION TO fga_auditor IDENTIFIED BY password;
GRANT AUDIT_VIEWER TO fga_auditor;
```

3. Connect as user SYSTEM.

```
CONNECT SYSTEM -- Or, CONNECT SYSTEM@hrpdb
Enter password: password
```

4. Ensure that the HR schema account is unlocked and has a password. If necessary, unlock HR and grant this user a password.

```
SELECT USERNAME, ACCOUNT_STATUS FROM DBA_USERS WHERE USERNAME = 'HR';
```

The account status should be OPEN. If the DBA_USERS view lists user HR as locked and expired, then enter the following statement to unlock the HR account and create a new password:

```
ALTER USER HR ACCOUNT UNLOCK IDENTIFIED BY password;
```

Follow the guidelines in Minimum Requirements for Passwords to create a password that is secure. For greater security, do **not** give the HR account the same password from previous releases of Oracle Database.

5. Create a user account for Susan Mavris, who is an HR representative whose actions you will audit, and then grant this user access to the HR.EMPLOYEES table.

```
GRANT CREATE SESSION TO smavris IDENTIFIED BY password;
GRANT SELECT, INSERT, UPDATE, DELETE ON HR.EMPLOYEES TO SMAVRIS;
```

Step 3: Configure an Access Control List File for Network Services

An access control list (ACL) file can be used to enable fine-grained access to external network services.

Before you can use PL/SQL network utility packages such as UTL_MAIL, you must configure this type of access control list (ACL) file.

1. Connect to SQL*Plus as user fga_admin.

```
CONNECT fga_admin -- Or, CONNECT fga_admin@hrpdb
Enter password: password
```

2. Configure the following access control setting and its privilege definitions.

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
  host       => 'SMTP_OUT_SERVER_setting',
  lower_port => 25,
  ace        =>  xs$ace_type(privilege_list => xs$name_list('smtp'),
                             principal_name => 'FGA_ADMIN',
                             principal_type => xs_acl.ptype_db));
END;
/
```

In this example:

- *`SMTP_OUT_SERVER_setting`*: Enter the `SMTP_OUT_SERVER` setting that you set for the `SMTP_OUT_SERVER` parameter in Step 1: Install and Configure the UTL_MAIL PL/SQL Package. This setting should match exactly the setting that your email tool specifies for its outgoing server.
- *`lower_port`*: Enter the port number that your email tool specifies for its outgoing server. Typically, this setting is 25. Enter this value for the `lower_port` setting. (Currently, the `UTL_MAIL` package does not support SSL. If your email server is an SSL server, then enter 25 for the port number, even if the email server uses a different port number.)
- `ace`: Define the privileges here.

See Also:

Managing Fine-Grained Access in PL/SQL Packages and Types for detailed information about configuring an access control list (ACL) file

Step 4: Create the Email Security Alert PL/SQL Procedure

The email security alert PL/SQL procedure generates a message describing the violation and then sends this message to the appropriate users.

- As user `fga_admin`, create the following procedure.

```
CREATE OR REPLACE PROCEDURE email_alert (sch varchar2, tab varchar2, pol
varchar2)
AS
msg varchar2(20000) := 'HR.EMPLOYEES table violation. The time is: ';
BEGIN
  msg := msg||TO_CHAR(SYSDATE, 'Day DD MON, YYYY HH24:MI:SS');
UTL_MAIL.SEND (
    sender      => 'youremail@example.com',
    recipients  => 'recipientemail@example.com',
    subject     => 'Table modification on HR.EMPLOYEES',
    message     => msg);
END email_alert;
/
```

In this example:

  – `CREATE OR REPLACE PROCEDURE ...AS`: You must include a signature that describes the schema name (`sch`), table name (`tab`), and the name of the audit procedure (`pol`) that you will define in audit policy in the next step.
  – `sender` and `recipients`: Replace *`youremail@example.com`* with your email address, and *`recipientemail@example.com`* with the email address of the person you want to receive the notification.

Step 5: Create and Test the Fine-Grained Audit Policy Settings

The fine-grained audit policy will trigger the alert when the policy is violated.

1. As user `fga_admin`, create the `chk_hr_emp` policy fine-grained audit policy as follows.

```
BEGIN
 DBMS_FGA.ADD_POLICY (
  object_schema      =>  'HR',
  object_name        =>  'EMPLOYEES',
  policy_name        =>  'CHK_HR_EMP',
  audit_column       =>  'SALARY',
  handler_schema     =>  'FGA_ADMIN',
  handler_module     =>  'EMAIL_ALERT',
  enable             =>   TRUE,
  statement_types    =>  'SELECT, UPDATE');
END;
/
```

2. Commit the changes you have made to the database.

   ```
   COMMIT;
   ```

3. Test the settings that you have created so far.

   ```
   EXEC email_alert ('hr', 'employees', 'chk_hr_emp');
   ```

   SQL*Plus should display a `PL/SQL procedure successfully completed` message, and in a moment, depending on the speed of your email server, you should receive the email alert.

   If you receive an `ORA-24247: network access denied by access control list (ACL)` error followed by `ORA-06512: at` *`string`*`line` *`string`* errors, then check the settings in the access control list file.

Step 6: Test the Alert

With the components in place, you are ready to test the alert.

1. Connect to SQL*Plus as user `smavris`, check your salary, and give yourself a nice raise.

   ```
   CONNECT smavris -- Or, CONNECT smavris@hrpdb
   Enter password: password

   SELECT SALARY FROM HR.EMPLOYEES WHERE LAST_NAME = 'Mavris';

   SALARY
   -----------
   6500

   UPDATE HR.EMPLOYEES SET SALARY = 38000 WHERE LAST_NAME = 'Mavris';
   ```

   By now, depending on the speed of your email server, you (or your recipient) should have received an email with the subject header `Table modification on HR.EMPLOYEES` notifying you of the tampering of the `HR.EMPLOYEES` table. Now all you need to do is to query the `UNIFIED_AUDIT_TRAIL` data dictionary view to find who the violator is.

2. As user `fga_auditor`, query the `UNIFIED_AUDIT_TRAIL` data dictionary view as follows:

   ```
   CONNECT fga_auditor -- Or, CONNECT fga_auditor@hrpdb
   Enter password: password

   col dbusername format a20
   col sql_text format a66
   col audit_type format a17
   ```

```
SELECT DBUSERNAME, SQL_TEXT, AUDIT_TYPE
FROM UNIFIED_AUDIT_TRAIL
WHERE OBJECT_SCHEMA = 'HR' AND OBJECT_NAME = 'EMPLOYEES';
```

Output similar to the following appears:

```
DBUSERNAME  SQL_TEXT                                                          AUDIT_TYPE
----------  ----------------------------------------------------------------- ----------------
SMAVRIS     UPDATE HR.EMPLOYEES SET SALARY = 38000 WHERE LAST_NAME = 'Mavris' FineGrainedAudit
```

The audit trail captures the SQL statement that Susan Mavris ran that affected the `SALARY` column in the `HR.EMPLOYEES` table. The first statement she ran, in which she asked about her current salary, was not recorded because it was not affected by the audit policy. This is because Oracle Database executes the audit function as an autonomous transaction, committing only the actions of the `handler_module` setting and not any user transaction. The function has no effect on any user SQL transaction.

Step 7: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. Connect to SQL*Plus as user `SYSTEM` privilege, and then drop users `fga_admin` (including the objects in the `fga_admin` schema), `fga_auditor`, and `smavris`.

   ```
   CONNECT SYSTEM -- Or, CONNECT SYSTEM@hrpdb
   Enter password: password

   DROP USER fga_admin CASCADE;
   DROP USER fga_auditor;
   DROP USER smavris;
   ```

2. Connect as user `HR` and remove the loftiness of Susan Mavris's salary.

   ```
   CONNECT HR -- Or, CONNECT HR@hrpdb
   Enter password: password

   UPDATE HR.EMPLOYEES SET SALARY = 6500 WHERE LAST_NAME = 'Mavris';
   ```

3. If you want, lock and expire `HR`, unless other users want to use this account:

   ```
   ALTER USER HR PASSWORD EXPIRE ACCOUNT LOCK;
   ```

4. Issue the following `ALTER SYSTEM` statement to restore the `SMTP_OUT_SERVER` parameter to the previous value, from Step 5 under Step 1: Install and Configure the UTL_MAIL PL/SQL Package:

   ```
   ALTER SYSTEM SET SMTP_OUT_SERVER="previous_value";
   ```

   Enclose this setting in quotation marks. For example:

   ```
   ALTER SYSTEM SET SMTP_OUT_SERVER="some_imap_server.example.com"
   ```

5. Restart the database instance.

Audit Policy Data Dictionary Views

Data dictionary and dynamic views can be used to find detailed auditing information.

Table 24-20 lists these views.

Tip:

To find error information about audit policies, check the trace files. The USER_DUMP_DEST initialization parameter sets the location of the trace files.

Table 24-20 Views That Display Information about Audited Activities

View	Description
ALL_AUDIT_POLICIES	Displays information about all fine-grained audit policies
ALL_DEF_AUDIT_OPTS	Lists default object-auditing options that are to be applied when objects are created
AUDIT_UNIFIED_CONTEXTS	Describes application context values that have been configured to be captured in the audit trail
AUDIT_UNIFIED_ENABLED_POLICIES	Describes all unified audit policies that are enabled in the database
AUDIT_UNIFIED_POLICIES	Describes all unified audit policies created in the database
AUDIT_UNIFIED_POLICY_COMMENTS	Shows the description of each unified audit policy, if a description was entered for the unified audit policy using the COMMENT SQL statement
AUDITABLE_SYSTEM_ACTIONS	Maps the auditable system action numbers to the action names
CDB_UNIFIED_AUDIT_TRAIL	Similar to the UNIFIED_AUDIT_TRAIL view, displays the audit records but from all PDBs in a multitenant environment. This view is available only in the CDB root and must be queried from there.
DBA_AUDIT_POLICIES	Displays information about fine-grained audit policies
DBA_SA_AUDIT_OPTIONS	Describes audited Oracle Label Security events performed by users, and indicates if the user's action failed or succeeded
DBA_XS_AUDIT_TRAIL	Displays audit trail information related to Oracle Database Real Application Security
DV$CONFIGURATION_AUDIT	Displays configuration changes made by Oracle Database Vault administrators
DV$ENFORCEMENT_AUDIT	Displays user activities that are affected by Oracle Database Vault policies
SYSTEM_PRIVILEGE_MAP (table)	Describes privilege (auditing option) type codes. This table can be used to map privilege (auditing option) type numbers to type names.
USER_AUDIT_POLICIES	Displays information about all fine-grained audit policies on table and views owned by the current user
UNIFIED_AUDIT_TRAIL	Displays all audit records
V$OPTION	You can query the PARAMETER column for Unified Auditing to find if unified auditing is enabled
V$XML_AUDIT_TRAIL	Displays standard, fine-grained, SYS, and mandatory audit records written in XML format files.

See Also:

Oracle Database Reference for detailed information about these views

25

Administering the Audit Trail

Users who have been granted the `AUDIT_ADMIN` role can manage the audit trail, archive the audit trail, and purge audit trail records.

Managing the Unified Audit Trail

Auditing is enabled by default, but you can control when audit records are written to disk.

When and Where Are Audit Records Created?

Auditing is always enabled. Oracle Database generates audit records during or after the execution phase of the audited SQL statements.

Oracle Database individually audits SQL statements inside PL/SQL program units, as necessary, when the program unit is run.

To improve read performance of the unified audit trail, the unified audit records are written immediately to disk to an internal relational table in the `AUDSYS` schema. In the previous release, the unified audit records were written to the common logging infrastructure (CLI) SGA queues. If you had migrated to unified auditing in Oracle Database 12*c* release 1 (12.1), then you can manually transfer the unified audit records from the SGA queues to this internal table. If the version of the database that you are using supports partitioned tables, then this internal table is a partitioned table. In this case, you can modify the partition interval of the table by using the `DBMS_AUDIT_MGMT.ALTER_PARTITION_INTERVAL` procedure. The partitioned version of this table is based on the `EVENT_TIMESTAMP` timestamp as a partition key with a default partition interval of one month. If the database version does not support partitioning, then the internal table is a regular, non-partitioned table.

The generation and insertion of an audit trail record is independent of the user transaction being committed. That is, even if a user transaction is rolled back, the audit trail record remains committed.

Statement and privilege audit options from unified audit policies that are in effect at the time a database user connects to the database remain in effect for the duration of the session. When the session is already active, setting or changing statement or privilege unified audit options does not take effect in that session. The modified statement or privilege audit options take effect only when the current session ends and a new session is created.

In contrast, changes to schema object audit options become immediately effective for current sessions.

By default, audit trail records are written to the `AUDSYS` schema in the `SYSAUX` tablespace. You can designate a different tablespace, including one that is encrypted, by using the `DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION` procedure.

See Also:

- Writing the Unified Audit Trail Records to the AUDSYS Schema for more information about immediate-write mode and queued-write mode
- *Oracle Database Upgrade Guide* for information about transferring unified audit records after an upgrade

Activities That Are Mandatorily Audited

The `UNIFIED_AUDIT_TRAIL` data dictionary view captures activities from administrative users such as `SYSDBA`, `SYSBACKUP`, and `SYSKM`.

You do not need to audit the unified audit trail. The unified audit trail resides in a read-only table in the `AUDSYS` schema. Hence, DMLs are not permitted on the unified audit trail views. Even DML and DDL operations on the underlying dictionary tables from `AUDSYS` schema are not permitted.

The `SYSTEM_PRIVILEGE_USED` column shows the type of administrative privilege that was used for the activity.

The following audit-related activities, such as modifications to audit policies, are mandatorily audited:

- `CREATE AUDIT POLICY`
- `ALTER AUDIT POLICY`
- `DROP AUDIT POLICY`
- `AUDIT`
- `NOAUDIT`
- `EXECUTE` of the `DBMS_FGA` PL/SQL package
- `EXECUTE` of the `DBMS_AUDIT_MGMT` PL/SQL package
- `ALTER TABLE` attempts on the `AUDSYS` audit trail table (remember that this table cannot be altered)
- Top level statements by the administrative users `SYS`, `SYSDBA`, `SYSOPER`, `SYSASM`, `SYSBACKUP`, `SYSDG`, and `SYSKM`, until the database opens. When the database opens, Oracle Database audits these users using the audit configurations in the system—not just the ones that were applied using the `BY` clause in the `AUDIT` statement, for example, but those that were applied for all users when `AUDIT` statement does not have a `BY` clause or when the `EXCEPT` clause was used and these users were not excluded.
- All user-issued DML statements on the `SYS.AUD$` and `SYS.FGA_LOG$` dictionary tables
- Any attempts to modify the data or metadata of the unified audit internal table. `SELECT` statements on this table are not audited by default or mandatorily.
- All configuration changes that are made to Oracle Database Vault

How Do Cursors Affect Auditing?

For each execution of an auditable operation within a cursor, Oracle Database inserts one audit record into the audit trail.

Events that cause cursors to be reused include the following:

- An application, such as Oracle Forms, holding a cursor open for reuse
- Subsequent execution of a cursor using new bind variables
- Statements executed within PL/SQL loops where the PL/SQL engine optimizes the statements to reuse a single cursor

Auditing is *not* affected by whether or not a cursor is shared. Each user creates her or his own audit trail records on first execution of the cursor.

Writing the Unified Audit Trail Records to the AUDSYS Schema

Oracle Database writes audit records to the `AUDSYS` schema.

> **Note:**
>
> Starting with Oracle Database 12*c* release 2 (12.2), the ability to flush unified audit records is deprecated but is retained for backward compatibility.
>
> Instead, unified audit records are written immediately to an internal relational table in the `AUDSYS` schema. This feature improves the read performance of the unified audit trail. If you have upgraded from Oracle Database 12*c* release 1 (12.1) and had migrated to unified auditing in that release, then you can manually transfer audit records from their previous location to this new internal relational table. See *Oracle Database Upgrade Guide* for more information about transferring unified audit records after an upgrade.

About Writing Unified Audit Trail Records to AUDSYS

In a new or just-migrated Oracle Database installation, the `AUDSYS` schema is empty until unified auditing is initiated and records are generated.

> **Note:**
>
> Starting with Oracle Database 12c release 12.2, the ability to flush audit records deprecated but is retained for backward compatibility.

This design greatly improves the performance of the audit trail processes and the database as a whole. In the previous release, in the event of an instance crash or during `SHUTDOWN ABORT` operations, there was a chance that some audit records would be lost. Oracle recommends that you configure the audit trail to immediately write audit records to the `AUDSYS` schema audit table by using the

DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS procedure to transfer the audit records to an internal relational table in the AUDSYS schema.

From the previous release, the following modes, deprecated but retained for backward compatibility, are available. However, Oracle recommends that you use the DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS procedure instead, as described in *Oracle Database Upgrade Guide*.

- **Immediate-write mode.** This setting writes all audit records to the audit trail immediately. However, be aware that database performance may be affected.
- **Queued-write mode.** This setting, which is the default write mode, queues the audit records in memory to be written periodically to the AUDSYS schema audit table. To set the size of the SGA, set the UNIFIED_AUDIT_SGA_QUEUE_SIZE initialization parameter. The default size is 1 MB, and you can enter a range of 1 through 30.

Note:

The UNIFIED_AUDIT_SGA_QUEUE_SIZE initialization parameter has been deprecated, but is currently retained for backward compatibility.

See Also:

Oracle Database Reference for more information about the UNIFIED_AUDIT_SGA_QUEUE_SIZE initialization parameter

Setting the Write Mode for Unified Audit Trail Records

The DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY procedure sets the write mode for unified audit trail records.

In a multitenant environment, the DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY procedure applies to the current pluggable database (PDB) only.

If the database is read-only, then DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY sets the value directly in the SGA.

Note:

The use of the AUDIT_TRAIL_WRITE mode value in the AUDIT_TRAIL_PROPERTY parameter in DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY has been deprecated because it is no longer necessary. Starting with Oracle Database 12*c* release 2 (12.2), unified audit records are written directly to a new internal relational table, bypassing the common logging infrastructure queues.

1. Log in to SQL*Plus as a user who has been granted the AUDIT_ADMIN role.

   For example:

```
sqlplus audit_admin
Enter password: password
```

2. In a multitenant environment, connect to the appropriate PDB.

   For example:

   ```
   CONNECT audit_admin@hrpdb
   Enter password: password
   ```

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.

3. Set the `AUDIT_TRAIL_MODE` property of the `DBMS_AUDIT_MGMT` package, as follows:

   - To use immediate-write mode, run the following procedure:

     ```
     BEGIN
      DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
       DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
       DBMS_AUDIT_MGMT.AUDIT_TRAIL_WRITE_MODE,
       DBMS_AUDIT_MGMT.AUDIT_TRAIL_IMMEDIATE_WRITE);
     END;
     /
     ```

   - To use queued-write mode, run the following procedure:

     ```
     BEGIN
      DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
       DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
       DBMS_AUDIT_MGMT.AUDIT_TRAIL_WRITE_MODE,
       DBMS_AUDIT_MGMT.AUDIT_TRAIL_QUEUED_WRITE);
     END;
     /
     ```

The settings take effect on subsequent database sessions.

Manually Flushing Audit Records to the Audit Trail in Queued-Write Mode

Queued-write mode manually writes records to the unified audit trail.

Note:

The `DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL` procedure, described in this section, has been deprecated but is currently retained for backward compatibility. Starting in Oracle Database 12*c* release 2 (12.2), audit records are written directly to a new internal relational table, bypassing the common logging infrastructure queues. Oracle recommends that you use the `DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS` procedure instead, as described in *Oracle Database Upgrade Guide*, to transfer old unified audit records to the new internal relational table.

Be aware that there is a minimum flush threshold that is configured by default. It checks if the queues were flushed more than 3 seconds before and accordingly flushes the audit records in the SGA queue. However, owing to the database activity, the flush may not happen every 3 seconds and could take longer.

Writing Records to Disk for the Current Database Instance

You can manually write records to disk for the current database instance or the current Oracle Real Application Clusters (Oracle RAC) instance.

Note:

The ability to flush audit records to disk is deprecated in Oracle Database 12c release 12.2. Oracle recommends that you use the `DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS` procedure instead, as described in *Oracle Database Upgrade Guide*, to transfer old unified audit records to the new internal relational table.

- To manually write records to disk, run either of the following procedures:

```
EXEC DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL;

EXEC
DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL(DBMS_AUDIT_MGMT.FLUSH_CURRENT_INSTANCE)
;
```

Writing Records to Disk Across an Oracle RAC Environment

You can flush audit records across all Oracle Real Application Cluster (Oracle RAC) instances.

Note:

The ability to flush audit records to disk is deprecated in Oracle Database 12*c* release 2 (12.2). Oracle recommends that you use the `DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS` procedure instead, as described in *Oracle Database Upgrade Guide*, to transfer old unified audit records to the new internal relational table.

- Run the following procedure to flush audit records on Oracle RAC instances:

```
EXEC
DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL(DBMS_AUDIT_MGMT.FLUSH_ALL_INSTANCES);
```

Writing Records to Disk in a Multitenant Environment

In a multitenant environment, you can write the audit trail records to disk for the current PDB or across all PDBs in a multitenent environment.

> **Note:**
>
> The ability to flush audit records to disk is deprecated in Oracle Database 12*c* release 2 (12.2). Oracle recommends that you use the `DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS` procedure instead, as described in *Oracle Database Upgrade Guide*, to transfer old unified audit records to the new internal relational table.

- To write audit trail records to disk, use one of the following procedures:
  - For the current PDB:

    ```
    BEGIN
     DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL(
      CONTAINER  => DBMS_AUDIT_MGMT.CONTAINER_CURRENT);
    END;
    /
    ```

  - For all PDBs in the multitenant environment:

    ```
    BEGIN
     DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL(
      CONTAINER  => DBMS_AUDIT_MGMT.CONTAINER_ALL);
    END;
    /
    ```

Writing the Unified Audit Trail Records to SYSLOG or the Windows Event Viewer

You can write the unified audit trail records to SYSLOG or the Windows Event Viewer by setting an initialization parameter.

About Writing the Unified Audit Trail Records to SYSLOG or the Windows Event Viewer

For this feature, the unified audit trail will capture data such as session IDs and the audit type.

You can configure this feature on both UNIX and Microsoft Windows systems. On Windows systems, you either enable it or disable it. If enabled, it writes the records to the Windows Event Viewer.

On UNIX systems, you can fine-tune the capture of unified audit trail records for SYSLOG to specify the facility where the SYSLOG records are sent and the severity level of the records (for example, `DEBUG` if it is capturing debugging-related messages).

Table 25-1 maps the names given to the unified audit records fields that are written to SYSLOG and the Windows Event Viewer to the corresponding column names in the `UNIFIED_AUDIT_TRAIL` view.

Table 25-1 Audit Record Field Names for SYSLOG and the Windows Event Viewer

Field Name	Column Name in UNIFIED_AUDIT_TRAIL	Column Type	Column Description
`TYPE`	`AUDIT_TYPE`	`NUMBER`	Type of the audit record
`DBID`	`DBID`	`NUMBER`	Database identifier
`SESID`	`SESSION_ID`	`NUMBER`	Session identifier
`CLIENTID`	`CLIENT_IDENTIFIER`	`VARCHAR2`	Client identifier in the session
`ENTRYID`	`ENTRY_ID`	`NUMBER`	Identifier for each audit record in the system
`STMTID`	`STATEMENT_ID`	`NUMBER`	Identifier for each statement run in the system
`DBUSER`	`DB_USERNAME`	`VARCHAR2`	Session user
`CURUSER`	`CURRENT_USER`	`VARCHAR2`	Effective user for the audited event
`ACTION`	`ACTION`	`NUMBER`	Action code of the audited event
`RETCODE`	`RETURN_CODE`	`NUMBER`	Return code for the audited event
`SCHEMA`	`OBJECT_SCHEMA`	`VARCHAR2`	Schema name of the object
`OBJNAME`	`OBJECT_NAME`	`VARCHAR2`	Name of the object

Enabling syslog and Windows Event Viewer Captures for the Unified Audit Trail

To write unified audit trail records to the UNIX `syslog` or to the Windows Event Viewer, you must set the `UNIFIED_AUDIT_SYSTEMLOG` initialization parameter.

The default for `UNIFIED_AUDIT_SYSTEMLOG` is `FALSE` on Microsoft Windows systems. On UNIX systems, there is no default. In an Oracle Database Real Application Clusters (Oracle RAC) environment, set this parameter to the same value on each Oracle RAC instance.

1. Locate the `init.ora` initialzation file, which by default is in the the `$ORACLE_HOME/dbs` directory.
2. Edit the `init.ora` file to include the `UNIFIED_AUDIT_SYSTEMLOG` parameter.
   - On Windows, set `UNIFIED_AUDIT_SYSTEMLOG` to either `TRUE` or `FALSE`. `TRUE` writes the `syslog` values to the Windows Event Viewer; `FALSE` disables the parameter. For example:

     ```
     UNIFIED_AUDIT_SYSTEMLOG = TRUE
     ```

- On UNIX systems, use the following syntax:

  ```
  UNIFIED_AUDIT_SYSTEMLOG = 'facility_clause.priority_clause'
  ```

  In this specification:

  – `facility_clause` refers to the facility to which you will write the audit trail records. Valid choices are `USER` and `LOCAL`. If you enter `LOCAL`, then optionally append `0–7` to designate a local custom facility for the `syslog` records.

  – `priority_clause` refers to the type of warning in which to categorize the record. Valid choices are `NOTICE`, `INFO`, `DEBUG`, `WARNING`, `ERR`, `CRIT`, `ALERT`, and `EMERG`.

For example:

```
UNIFIED_AUDIT_SYSTEMLOG = 'LOCAL7.EMERG'
```

3. Exit SQL*Plus.
4. Add the audit file destination to the `syslog` configuration file `/etc/syslog.conf`.

   For example, assuming you had set the `UNIFIED_AUDIT_SYSTEMLOG` to `LOCAL7.EMERG`, enter the following:

   ```
   local7.emerg /var/log/audit.log
   ```

   This setting logs all emergency messages to the `/var/log/audit.log` file.
5. Restart the `syslog` logger.

   ```
   $/etc/rc.d/init.d/syslog restart
   ```

   Now, all audit records will be captured in the file `/var/log/audit.log` through the `syslog` daemon.
6. Log back in to the database instance.
7. Restart the database.

   For example:

   ```
   SHUTDOWN IMMEDIATE
   STARTUP
   ```

When Audit Records Are Written to the Operating System

In situations where the database table is unable to accept unified audit records, these records will be written to operating system spillover audit files (`.bin` format).

The ability to write to the database table can fail in situations such as the following: the audit tablespace is offline, the tablespace is read-only, the tablespace is full, the database is read-only, and so on. The unified audit records will continue to be written to OS spillover files until the OS disk space becomes full. At this point, when there is no room in the OS for the audit records, user auditable transactions will fail with `ORA-02002 error while writing to audit trail` errors. To prevent this problem, Oracle recommends that you purge the audit trail on a regular basis.

Moving Operating System Audit Records into the Unified Audit Trail

Audit records can be written to external files in the `$ORACLE_BASE/audit/$ORACLE_SID` directory.

When the database is not writable (such as during database mounts), if the database is closed, or if it is read-only, then Oracle Database writes the audit records to these external files.

You can load the files into the database by running the `DBMS_AUDIT_MGMT.LOAD_UNIFIED_AUDIT_FILES` procedure. Be aware that if you are moving a large number of operating system audit records to the unified audit trail, performance may be affected.

To move the audit records in these files to the `AUDSYS` schema audit table when the database is writable:

1. Log into the database instance as a user who has been granted the `AUDIT_ADMIN` role.

   For example:

   ```
   CONNECT aud_admin
   Enter password: password
   Connected.
   ```

   In a multitenant environment, log into the PDB in which you want to move the audit trail records to the unified audit trail.

   For example:

   ```
   CONNECT aud_admin@hrpdb
   Enter password: password
   Connected.
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

2. Ensure that the database is open and writable.

   For a non-CDB architecture, to find the databases that are open and writable, query the `V$DATABASE` view.

   For example:

   ```
   SELECT NAME, OPEN_MODE FROM V$DATABASE;

   NAME            OPEN_MODE
   --------------- ----------
   HRPDB           READ WRITE
   ```

   In a multitenant environment, you can run the `show pdbs` command to find information about PDBs associated with the current instance.

3. Run the `DBMS_AUDIT_MGMT.LOAD_UNIFIED_AUDIT_FILES` procedure.

   ```
   EXEC DBMS_AUDIT_MGMT.LOAD_UNIFIED_AUDIT_FILES;
   ```

The audit records are loaded into the `AUDSYS` schema audit table immediately, and then deleted from the `$ORACLE_BASE/audit/$ORACLE_SID` directory.

Disabling Unified Auditing

You can disable unified auditing.

1. Disable any unified audit policies that are currently enabled.

   This step prevents the database from going into mixed mode auditing after you complete this procedure.

   a. Log into the database instance as a user who has been granted the `AUDIT_ADMIN` role.

   b. Query the `POLICY_NAME` and `ENABLED_OPT` columns of the `AUDIT_UNIFIED_ENABLED_POLICIES` data dictionary view to find unified audit policies that are enabled.

   c. Run the `NOAUDIT POLICY` statement to disable each enabled policy.

      For example, to disable a policy that had been applied to user `psmith`:

      ```
      NOAUDIT POLICY audit_pol BY psmith;
      ```

2. Connect as user `SYS` with the `SYSOPER` privilege.

   ```
   CONNECT sys as sysoper
   Enter password: password
   ```

   In a multitenant environment, this command connects you to the root.

3. Shut down the database.

   For example:

   ```
   SHUTDOWN IMMEDIATE
   ```

   In a multitenant environment, this command shuts down all PDBs in the CDB.

4. Depending on your platform, do the following:

   - **UNIX systems:** Run the following commands:

     ```
     cd $ORACLE_HOME/rdbms/lib
     make -f ins_rdbms.mk uniaud_off ioracle
     ```

   - **Windows systems:** Rename the `%ORACLE_HOME%/bin/orauniaud12.dll` file to `%ORACLE_HOME%/bin/orauniaud12.dll.dbl`.

   In a multitenant environment, these actions disable unified auditing in all PDBs in the CDB.

5. In SQL*Plus, restart the database.

   ```
   STARTUP
   ```

   In a multitenant environment, this command restarts all PDBs in the CDB.

Exporting and Importing the Unified Audit Trail Using Oracle Data Pump

You can include the unified audit trail in Oracle Database Pump export and import dump files.

The unified audit trail is automatically included in either full database or partial database export and import operations using Oracle Data Pump. For example, for a partial database export operation, if you wanted to export only the unified audit trail tables, then you could enter the following `expdp` command:

```
expdp system
full=y
directory=aud_dp_dir
logfile=audexp_log.log
dumpfile=audexp_dump.dmp
version=12.02.00.02.00
INCLUDE=AUDIT_TRAILS

Password: password
```

Next, you can import all the exported content by reading the export dump file. This operation imports only the unified audit trial tables.

```
impdp system
full=y
directory=aud_dp_dir
dumpfile=audexp_dump.dmp
logfile=audimp_log.log

Password: password
```

You do not need to perform any special configuration to achieve this operation. However, you must have the `EXP_FULL_DATABASE` role if you are performing the export operation and the `IMP_FULL_DATABASE` role if you are performing the import operation.

Archiving the Audit Trail

You can archive the traditional operating system, unified database, and traditional database audit trails.

Archiving the Traditional Operating System Audit Trail

You can create an archive of the traditional operating system audit files after you have upgraded Oracle Database.

To archive the traditional operating system audit trail from an upgraded database, use your platform-specific operating system tools to create an archive of the traditional operating system audit files.

- Use the following methods to archive the traditional operating system audit files:
  - **Use Oracle Audit Vault and Database Firewall.** You install Oracle Audit Vault and Database Firewall separately from Oracle Database.
  - **Create tape or disk backups.** You can create a compressed file of the audit files, and then store it on tapes or disks. Consult your operating system documentation for more information.

Afterwards, you should purge (delete) the traditional operating system audit records both to free audit trail space and to facilitate audit trail management.

See Also:

- *Oracle Audit Vault and Database Firewall Administrator's Guide* for more information about Oracle Audit Vault and Database Firewall
- Moving Operating System Audit Records into the Unified Audit Trail
- Purging Audit Trail Records

Archiving the Unified and Traditional Database Audit Trails

You should periodically archive and then purge the audit trail to prevent it from growing too large.

Archiving and purging both frees audit trail space and facilitates the purging of the database audit trail.

You can create an archive of the unified and traditional database audit trail by using Oracle Audit Vault and Database Firewall. You install Oracle Audit Vault and Database Firewall separately from Oracle Database.

After you complete the archive, you can purge the database audit trail contents.

- To archive the unified, traditional standard, and traditional fine-grained audit records, copy the relevant records to a normal database table.

  For example:

  ```
  INSERT INTO table SELECT ... FROM UNIFIED_AUDIT_TRAIL ...;
  INSERT INTO table SELECT ... FROM SYS.AUD$ ...;
  INSERT INTO table SELECT ... FROM SYS.FGA_LOG$ ...;
  ```

See Also:

- *Oracle Audit Vault and Database Firewall Administrator's Guide* for more information about Oracle Audit Vault and Database Firewall
- Purging Audit Trail Records

Purging Audit Trail Records

The `DBMS_AUDIT_MGMT` PL/SQL package can schedule automatic purge jobs, manually purge audit records, and perform other audit trail operations.

About Purging Audit Trail Records

You can use a variety of ways to purge audit trail records.

You should periodically archive and then delete (purge) audit trail records. You can purge a subset of audit trail records or create a purge job that performs at a specified time interval. Oracle Database either purges the audit trail records that were created before the archive timestamp, or it purges all audit trail records. You can purge audit trail records in both read-write and read-only databases.

The purge process takes into account not just the unified audit trail, but audit trails from earlier releases of Oracle Database. For example, if you have migrated an upgraded database that still has operating system or XML audit records, then you can use the procedures in this section to archive and purge them.

To perform the audit trail purge tasks, in most cases, you use the `DBMS_AUDIT_MGMT` PL/SQL package. You must have the `AUDIT_ADMIN` role before you can use the `DBMS_AUDIT_MGMT` package. Oracle Database mandatorily audits all executions of the `DBMS_AUDIT_MGMT` PL/SQL package procedures.

If you have Oracle Audit Vault and Database Firewall installed, the audit trail purge process differs from the procedures described in this manual. For example, Oracle Audit Vault archives the audit trail for you.

Note:

Oracle Database audits all deletions from the audit trail, without exception.

See Also:

- *Oracle Audit Vault and Database Firewall Administrator's Guide* for information about Oracle Audit Vault and Database Firewall
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_AUDIT_MGMT` PL/SQL package
- *Oracle Database Reference* for detailed information about the `DBA_AUDIT_MGMT`-related views

Selecting an Audit Trail Purge Method

You can perform the purge on a regularly scheduled basis or at a specified times.

Purging the Audit Trail on a Regularly Scheduled Basis

You can purge all audit records, or audit records that were created before a specified timestamp, on a regularly scheduled basis.

For example, you can schedule the purge for every Saturday at 2 a.m.

1. If necessary, tune online and archive redo log sizes to accommodate the additional records generated during the audit table purge process.
2. Plan a timestamp and archive strategy.
3. Optionally, set an archive timestamp for the audit records.
4. Create and schedule the purge job.

Manually Purging the Audit Trail at a Specific Time

You can manually purge the audit records right away in a one-time operation, rather than creating a purge schedule.

1. If necessary, tune online and archive redo log sizes to accommodate the additional records generated during the audit table purge process.
2. Plan a timestamp and archive strategy.
3. Optionally, set an archive timestamp for the audit records.
4. Run the purge operation.

Scheduling an Automatic Purge Job for the Audit Trail

Scheduling an automatic purge job requires planning beforehand, such as tuning the online and archive redo log sizes.

About Scheduling an Automatic Purge Job

You can purge the entire audit trail, or only a portion of the audit trail that was created before a timestamp.

The individual audit records created before the timestamp can be purged.

Be aware that purging the audit trail, particularly a large one, can take a while to complete. Consider scheduling the purge job so that it runs during a time when the database is not busy.

You can create multiple purge jobs for different audit trail types, so long as they do not conflict. For example, you can create a purge job for the standard audit trail table and then the fine-grained audit trail table. However, you cannot then create a purge job for both or all types, that is, by using the `DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD` or `DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL` property. In addition, be aware that the jobs created by the `DBMS_SCHEDULER` PL/SQL package do not execute on a read-only database. An automatic purge job created with `DBMS_AUDIT_MGMT` uses the `DBMS_SCHEDULER` package to schedule the tasks. Therefore, these jobs cannot run on a database or PDB that is open in read-only mode.

Step 1: If Necessary, Tune Online and Archive Redo Log Sizes

The purge process may generate additional redo logs.

- If necessary, tune online and archive redo log sizes to accommodate the additional records generated during the audit table purge process.

In a unified auditing environment, the purge process does not generate as many redo logs as in a mixed mode auditing environment, so if you have migrated to unified auditing, then you may want to bypass this step.

See Also:

Oracle Database Administrator's Guide for more information about tuning log files

Step 2: Plan a Timestamp and Archive Strategy

You must record the timestamp of the audit records before you can archive them.

- To find the timestamp date, query the `DBA_AUDIT_MGMT_LAST_ARCH_TS` data dictionary view.

Later on, when the purge takes place, Oracle Database purges only the audit trail records that were created before the date of this archive timestamp.

After you have timestamped the records, you are ready to archive them.

Step 3: Optionally, Set an Archive Timestamp for Audit Records

If you want to delete all of the audit trail, then you can bypass this step.

You can set a timestamp for when the last audit record was archived. Setting an archive timestamp provides the point of cleanup to the purge infrastructure. If you are setting a timestamp for a read-only database, then you can use the `DBMS_AUDIT.MGMT.GET_LAST_ARCHIVE_TIMESTAMP` function to find the last archive timestamp that was configured for the instance on which it was run. For a read-write database, you can query the `DBA_AUDIT_MGMT_LAST_ARCH_TS` data dictionary view.

To find the last archive timestamps for the unified audit trail, you can query the `DBA_AUDIT_MGMT_LAST_ARCH_TS` data dictionary view. After you set the timestamp, all audit records in the audit trail that indicate a time earlier than that timestamp are purged when you run the `DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL` PL/SQL procedure. If you want to clear the archive timestamp setting, see Clearing the Archive Timestamp Setting.

If you are using Oracle Database Real Application Clusters, then use Network Time Protocol (NTP) to synchronize the time on each computer where you have installed an Oracle Database instance. For example, suppose you set the time for one Oracle RAC instance node at 11:00:00 a.m. and then set the next Oracle RAC instance node at 11:00:05. As a result, the two nodes have inconsistent times. You can use Network Time Protocol (NTP) to synchronize the times for these Oracle RAC instance nodes.

To set the timestamp for the purge job:

1. Log into the database instance as a user who has been granted the `AUDIT_ADMIN` role.

   In a multitenant environment, log into either the root or the PDB in which you want to schedule the purge job. In most cases, you may want to schedule the purge job on individual PDBs.

   For example, to log into a PDB called `hrpdb`:

   ```
   CONNECT aud_admin@hrpdb
   Enter password: password
   Connected.
   ```

2. Run the DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP PL/SQL procedure to set the timestamp.

   For example:

```
BEGIN
  DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP(
   AUDIT_TRAIL_TYPE     =>  DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
   LAST_ARCHIVE_TIME    =>  '12-OCT-2013 06:30:00.00',
   RAC_INSTANCE_NUMBER  =>  1,
   CONTAINER            => DBMS_AUDIT_MGMT.CONTAINER_CURRENT);
END;
/
```

   In this example:

   - AUDIT_TRAIL_TYPE specifies the audit trail type. DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED sets it for the unified audit trail.

     For upgraded databases that still have traditional audit data from previous releases:

     – DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD is used for the traditional standard audit trail table, AUD$. (This setting does not apply to read-only databases.)

     – DBMS_AUDIT_MGMT.AUDIT_TRAIL_FGA_STD is used for the traditional fine-grained audit trail table, FGA_LOG$. (This setting does not apply to read-only databases.)

     – DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS is used for the traditional operating system audit trail files with the .aud extension. (This setting does not apply to Windows Event Log entries.)

     – DBMS_AUDIT_MGMT.AUDIT_TRAIL_XML is used for the XML traditional operating system audit trail files.

   - LAST_ARCHIVE_TIME specifies the timestamp in YYYY-MM-DD HH:MI:SS.FF UTC (Coordinated Universal Time) format for AUDIT_TRAIL_UNIFIED, AUDIT_TRAIL_AUD_STD, and AUDIT_TRAIL_FGA_STD, and in the Local Time Zone for AUDIT_TRAIL_OS and AUDIT_TRAIL_XML.

   - RAC_INSTANCE_NUMBER specifies the instance number for an Oracle RAC installation. This setting is not relevant for single instance databases. If you specified the DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD or DBMS_AUDIT_MGMT.AUDIT_TRAIL_FGA_STD audit trail types, then you can omit the RAC_INSTANCE_NUMBER argument. This is because there is only one AUD$ or FGA_LOG$ table, even for an Oracle RAC installation. The default is NULL. You can find the instance number for the current instance by issuing the SHOW PARAMETER INSTANCE_NUMBER command in SQL*Plus.

   - CONTAINER applies the timestamp to a multitenant environment. DBMS_AUDIT_MGMT.CONTAINER_CURRENT specifies the current PDB; DBMS_AUDIT_MGMT.CONTAINER_ALL applies to all PDBs in the multitenant environment.

     Note that you can set CONTAINER to DBMS_MGMT.CONTAINER_ALL only from the root, and DBMS_MGMT.CONTAINER_CURRENT only from a PDB.

   Typically, after you set the timestamp, you can use the DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL PL/SQL procedure to remove the audit records that were created before the timestamp date.

Step 4: Create and Schedule the Purge Job

You can use the DBMS_AUDIT_MGMT PL/SQL package to create and schedule the purge job.

- Create and schedule the purge job by running the DBMS_AUDIT_MGMT.CREATE_PURGE_JOB PL/SQL procedure.

  For example:

```
CONNECT aud_admin@hrpdb
Enter password: password
Connected.

BEGIN
  DBMS_AUDIT_MGMT.CREATE_PURGE_JOB (
   AUDIT_TRAIL_TYPE            => DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
   AUDIT_TRAIL_PURGE_INTERVAL  => 12,
   AUDIT_TRAIL_PURGE_NAME      => 'Audit_Trail_PJ',
   USE_LAST_ARCH_TIMESTAMP      => TRUE,
   CONTAINER                   => DBMS_AUDIT_MGMT.CONTAINER_CURRENT);
END;
/
```

In this example:

- AUDIT_TRAIL_TYPE: Specifies the audit trail type. DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED sets it for the unified audit trail.

  For upgraded databases that still have audit data from previous releases:

  – DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD is used for the standard audit trail table, AUD$. (This setting does not apply to read-only databases.)

  – DBMS_AUDIT_MGMT.AUDIT_TRAIL_FGA_STD is used for the fine-grained audit trail table, FGA_LOG$. (This setting does not apply to read-only databases.)

  – DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD is used for both standard and fine-grained audit trail tables. (This setting does not apply to read-only databases.)

  – DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS is used for the operating system audit trail files with the .aud extension. (This setting does not apply to Windows Event Log entries.)

  – DBMS_AUDIT_MGMT.AUDIT_TRAIL_XML is used for the XML operating system audit trail files.

  – DBMS_AUDIT_MGMT.AUDIT_TRAIL_FILES is used for both operating system and XML audit trail files.

  – DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL is used for all audit trail records, that is, both database audit trail and operating system audit trail types. (This setting does not apply to read-only databases.)

- AUDIT_TRAIL_PURGE_INTERVAL specifies the hourly interval for this purge job to run. The timing begins when you run the DBMS_AUDIT_MGMT.CREATE_PURGE_JOB procedure, in this case, 12 hours after you run this procedure. Later on, if you want to update this value, run the DBMS_AUDIT_MGMT.SET_PURGE_JOB_INTERVAL procedure.

- USE_LAST_ARCH_TIMESTAMP accepts either of the following settings:

  - TRUE deletes audit records created before the last archive timestamp. To check the last recorded timestamp, query the LAST_ARCHIVE_TS column of the DBA_AUDIT_MGMT_LAST_ARCH_TS data dictionary view for read-write databases and the DBMS_AUDIT_MGMT.GET_LAST_ARCHIVE_TIMESTAMP function for read-only databases. The default value is TRUE. Oracle recommends that you set USE_LAST_ARCH_TIMESTAMP to TRUE.
  - FALSE deletes all audit records without considering last archive timestamp. Be careful about using this setting, in case you inadvertently delete audit records that should have been deleted.
- CONTAINER is used for a multitenant environment to define where to create the purge job. If you set CONTAINER to DBMS_AUDIT_MGMT.CONTAINER_CURRENT, then it is available, visible, and managed only from the current PDB. The DBMS_AUDIT_MGMT.CONTAINER_ALL setting creates the job in the root. This defines the job as a global job, which runs according to the defined job schedule. When the job is invoked, it cleans up audit trails in all the PDBs in the multitenant environment. If you create the job in the root, then it is visible only in the root. Hence, you can enable, disable, and drop it from the root only.

Manually Purging the Audit Trail

You can use the DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL procedure to manually purge the audit trail.

About Manually Purging the Audit Trail

You can manually purge the audit trail right away, without scheduling a purge job.

Similar to a purge job, you can purge audit trail records that were created before an archive timestamp date or all the records in the audit trail. Only the current audit directory is cleaned up when you run this procedure.

For upgraded databases that may still have audit trails from earlier releases, note the following about the DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL PL/SQL procedure:

- On Microsoft Windows, because the DBMS_AUDIT_MGMT package does not support cleanup of Windows Event Viewer, setting the AUDIT_TRAIL_TYPE property to DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS has no effect. This is because operating system audit records on Windows are written to Windows Event Viewer. The DBMS_AUDIT_MGMT package does not support this type of cleanup operation.
- On UNIX platforms, if you had set the AUDIT_SYSLOG_LEVEL initialization parameter, then Oracle Database writes the operating system log files to syslog files. (Be aware that when you configure the use of syslog files, the messages are sent to the syslog daemon process. The syslog daemon process does not return an acknowledgement to Oracle Database indicating a committed write to the syslog files.) If you set the AUDIT_TRAIL_TYPE property to DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS, then the procedure only removes .aud files under audit directory (This directory is specified by the AUDIT_FILE_DEST initialization parameter).

Using DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL to Manually Purge the Audit Trail

After you complete preparatory steps, you can use the DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL procedure to manually purge the audit trail.

1. Follow these steps under Scheduling an Automatic Purge Job for the Audit Trail:
   - Step 1: If Necessary, Tune Online and Archive Redo Log Sizes
   - Step 2: Plan a Timestamp and Archive Strategy
   - Step 3: Optionally, Set an Archive Timestamp for Audit Records
2. If you are using a multitenant environment, then connect to the database in which you created the purge job.

   If you created the purge job in the root, then you must log into the root. If you created the purge job in a specific PDB, then log into that PDB.

   For example:

   ```
   CONNECT aud_admin@hrpdb
   Enter password: password
   Connected.
   ```
3. Purge the audit trail records by running the `DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL` PL/SQL procedure.

   For example:

   ```
   BEGIN
     DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL(
      AUDIT_TRAIL_TYPE           =>  DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
      USE_LAST_ARCH_TIMESTAMP    =>  TRUE,
      CONTAINER                  =>  DBMS_AUDIT_MGMT.CONTAINER_CURRENT );
   END;
   /
   ```

   In this example:

   - `AUDIT_TRAIL_TYPE`: Specifies the audit trail type. `DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED` sets it for the unified audit trail.

     For upgraded databases that still have audit data from previous releases:

     – `DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD`: Standard audit trail table, `AUD$`. (This setting does not apply to read-only databases.)

     – `DBMS_AUDIT_MGMT.AUDIT_TRAIL_FGA_STD`: Fine-grained audit trail table, `FGA_LOG$`. (This setting does not apply to read-only databases.)

     – `DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD`: Both standard and fine-grained audit trail tables. (This setting does not apply to read-only databases)

     – `DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS`: Operating system audit trail files with the `.aud` extension. (This setting does not apply to Windows Event Log entries.)

     – `DBMS_AUDIT_MGMT.AUDIT_TRAIL_XML`: XML Operating system audit trail files.

     – `DBMS_AUDIT_MGMT.AUDIT_TRAIL_FILES`: Both operating system and XML audit trail files.

     – `DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL`: All audit trail records, that is, both database audit trail and operating system audit trail types. (This setting does not apply to read-only databases.)
   - `USE_LAST_ARCH_TIMESTAMP`: Enter either of the following settings:

     – `TRUE`: Deletes audit records created before the last archive timestamp. To set the archive timestamp, see Step 3: Optionally, Set an Archive

Timestamp for Audit Records. The default (and recommended) value is TRUE. Oracle recommends that you set USE_LAST_ARCH_TIMESTAMP to TRUE.

   - FALSE: Deletes all audit records without considering last archive timestamp. Be careful about using this setting, in case you inadvertently delete audit records that should have been deleted.

- CONTAINER: Applies the cleansing to a multitenant environment. DBMS_AUDIT_MGMT.CONTAINER_CURRENT specifies the local PDB; DBMS_AUDIT_MGMT.CONTAINER_ALL applies to all databases.

Other Audit Trail Purge Operations

Other kinds of audit trail purge include enabling or disabling the audit trail purge job or setting the default audit trail purge job interval.

Enabling or Disabling an Audit Trail Purge Job

The DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS procedure enables or disables an audit trail purge job.

In a multitenant environment, where you run the DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS procedure depends on the location of the purge job, which is determined by the CONTAINER parameter of the DBMS_MGMT.CREATE_PURGE_JOB procedure. If you had set CONTAINER to CONTAINER_ALL (to create the purge job in the root), then you must run the DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS procedure from the root. If you had set CONTAINER to CONTAINER_CURRENT, then you must run the DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS procedure from the PDB in which it was created.

- To enable or disable an audit trail purge job, use the DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS PL/SQL procedure.

  For example, assuming that you had created the purge job in a the hrpdb PDB:

  ```
  CONNECT aud_admin@hrpdb
  Enter password: password
  Connected.

  BEGIN
   DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS(
    AUDIT_TRAIL_PURGE_NAME      => 'Audit_Trail_PJ',
    AUDIT_TRAIL_STATUS_VALUE    => DBMS_AUDIT_MGMT.PURGE_JOB_ENABLE);
  END;
  /
  ```

In this example:

- AUDIT_TRAIL_PURGE_NAME specifies a purge job called Audit_Trail_PJ. To find existing purge jobs, query the JOB_NAME and JOB_STATUS columns of the DBA_AUDIT_MGMT_CLEANUP_JOBS data dictionary view.
- AUDIT_TRAIL_STATUS_VALUE accepts either of the following properties:
  - DBMS_AUDIT_MGMT.PURGE_JOB_ENABLE enables the specified purge job.
  - DBMS_AUDIT_MGMT.PURGE_JOB_DISABLE disables the specified purge job.

Setting the Default Audit Trail Purge Job Interval for a Specified Purge Job

You can set a default purge operation interval, in hours, that must pass before the next purge job operation takes place.

The interval setting that is used in the `DBMS_AUDIT_MGMT.CREATE_PURGE_JOB` procedure takes precedence over this setting.

- To set the default audit trail purge job interval for a specific purge job, run the `DBMS_AUDIT_MGMT.SET_PURGE_JOB_INTERVAL` procedure.

  For example, assuming that you had created the purge job in the `hrpdb` PDB:

  ```
  CONNECT aud_admin@hrpdb
  Enter password: password
  Connected.

  BEGIN
   DBMS_AUDIT_MGMT.SET_PURGE_JOB_INTERVAL(
    AUDIT_TRAIL_PURGE_NAME       => 'Audit_Trail_PJ',
    AUDIT_TRAIL_INTERVAL_VALUE   => 24);
  END;
  /
  ```

In this example:

- `AUDIT_TRAIL_PURGE_NAME` specifies the name of the audit trail purge job. To find a list of existing purge jobs, query the `JOB_NAME` and `JOB_STATUS` columns of the `DBA_AUDIT_MGMT_CLEANUP_JOBS` data dictionary view.
- `AUDIT_TRAIL_INTERVAL_VALUE` updates the default hourly interval set by the `DBMS_AUDIT_MGMT.CREATE_PURGE_JOB` procedure. Enter a value between `1` and `999`. The timing begins when you run the purge job.

In a multitenant environment, where you run the `DBMS_AUDIT_MGMT.SET_PURGE_JOB_INTERVAL` procedure depends on the location of the purge job, which is determined by the `CONTAINER` parameter of the `DBMS_MGMT.CREATE_PURGE_JOB` procedure. If you had set `CONTAINER` to `CONTAINER_ALL`, then the purge job exists in the root, so you must run the `DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS` procedure from the root. If you had set `CONTAINER` to `CONTAINER_CURRENT`, then you must run the `DBMS_AUDIT_MGMT.SET_PURGE_JOB_INTERVAL` procedure from the PDB in which it was created.

Deleting an Audit Trail Purge Job

You can delete existing audit trail purge jobs.

To find existing purge jobs, query the `JOB_NAME` and `JOB_STATUS` columns of the `DBA_AUDIT_MGMT_CLEANUP_JOBS` data dictionary view.

- To delete an audit trail purge job, use the `DBMS_AUDIT_MGMT.DROP_PURGE_JOB` PL/SQL procedure.

For example, assuming that you had created the purge job in the `hrpdb` PDB:

```
CONNECT aud_admin@hrpdb
Enter password: password
Connected.
```

```
BEGIN
 DBMS_AUDIT_MGMT.DROP_PURGE_JOB(
  AUDIT_TRAIL_PURGE_NAME  => 'Audit_Trail_PJ');
END;
/
```

In a multitenant environment, where you run the DBMS_AUDIT_MGMT.DROP_PURGE_JOB procedure depends on the location of the purge job, which is determined by the CONTAINER parameter of the DBMS_MGMT.CREATE_PURGE_JOB procedure. If you had set CONTAINER to CONTAINER_ALL, then the purge job exists in the root, so you must run the DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS procedure from the root. If you had set CONTAINER to CONTAINER_CURRENT, then you must run the DBMS_AUDIT_MGMT.DROP_PURGE_JOB_INTERVAL procedure from the PDB in which it was created.

Clearing the Archive Timestamp Setting

The DBMS_AUDIT_MGMT.CLEAR_LAST_ARCHIVE_TIMESTAMP procedure can clear the archive timestamp setting.

To find a history of audit trail log cleanup, you can query the UNIFIED_AUDIT_TRAIL data dictionary view, using the following criteria: OBJECT_NAME is DBMS_AUDIT_MGMT, OBJECT_SCHEMA is SYS, and SQL_TEXT is set to LIKE %DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL%.

- To clear the archive timestamp setting, use the DBMS_AUDIT_MGMT.CLEAR_LAST_ARCHIVE_TIMESTAMP PL/SQL procedure to specify the audit trail type and for a multitenant environment, the container type.

  For example, assuming that you had created the purge job in the hrpdb PDB:

  ```
  CONNECT aud_admin@hrpdb
  Enter password: password
  Connected.

  BEGIN
    DBMS_AUDIT_MGMT.CLEAR_LAST_ARCHIVE_TIMESTAMP(
     AUDIT_TRAIL_TYPE     =>  DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
     CONTAINER            =>  DBMS_AUDIT_MGMT.CONTAINER_CURRENT);
  END;
  /
  ```

In this example:

- AUDIT_TRAIL_TYPE is set for the unified audit trail. If the AUDIT_TRAIL_TYPE property is set to DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS or DBMS_AUDIT_MGMT.AUDIT_TRAIL_XML, then you cannot set RAC_INSTANCE_NUMBER to 0. You can omit the RAC_INSTANCE_NUMBER setting if you set AUDIT_TRAIL_TYPE to DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED.
- CONTAINER applies the timestamp to a multitenant environment. DBMS_AUDIT_MGMT.CONTAINER_CURRENT specifies the local PDB; DBMS_AUDIT_MGMT.CONTAINER_ALL applies to all databases.

Example: Directly Calling a Unified Audit Trail Purge Operation

You can create a customized archive procedure to directly call a unified audit trail purge operation.

The pseudo code in Example 25-1 creates a database audit trail purge operation that the user calls by invoking the `DBMS_ADUIT.CLEAN_AUDIT_TRAIL` procedure for the unified audit trail.

The purge operation deletes records that were created before the last archived timestamp by using a loop. The loop archives the audit records, calculates which audit records were archived and uses the `SetCleanUpAuditTrail` call to set the last archive timestamp, and then calls the `CLEAN_AUDIT_TRAIL` procedure. In this example, major steps are in **bold** typeface.

Example 25-1 Directly Calling a Database Audit Trail Purge Operation

```
-- 1. Set the last archive timestamp:
PROCEDURE SetCleanUpAuditTrail()
 BEGIN
  CALL FindLastArchivedTimestamp(AUD$);
  DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP(
   AUDIT_TRAIL_TYPE          => DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
   LAST_ARCHIVE_TIME         => '23-AUG-2013 12:00:00',
   CONTAINER                 => DBMS_AUDIT_MGMT.CONTAINER_CURRENT);
 END;
/
-- 2. Run a customized archive procedure to purge the audit trail records:
BEGIN
  CALL MakeAuditSettings();
  LOOP (/* How long to loop*/)
    -- Invoke function for audit record archival
    CALL DoUnifiedAuditRecordArchival();

    CALL SetCleanUpAuditTrail();
    IF(/* Clean up is needed immediately */)
      DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL(
       AUDIT_TRAIL_TYPE        => DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
       USE_LAST_ARCH_TIMESTAMP => TRUE,
       CONTAINER               => DBMS_AUDIT_MGMT.CONTAINER_CURRENT );
    END IF
  END LOOP /*LOOP*/
END; /* PROCEDURE */
/
```

If you want to modify this example for other audit trail types, be aware that additional steps may be required. For more information, see the Oracle Database 11*g* Release 2 (11.2) version of *Oracle Database Security Guide*, which is available from the following documentation library:

`http://www.oracle.com/pls/db112/homepage`

Audit Trail Management Data Dictionary Views

Oracle Database provides data dictionary views that list information about audit trail management settings.

Table 25-2 lists these views.

Table 25-2 Views That Display Information about Audit Trail Management Settings

View	Description
`DBA_AUDIT_MGMT_CLEAN_EVENTS`	Displays the history of purge events of the traditional (that is, non-unified) audit trails. Periodically, as a user who has been granted the `AUDIT_ADMIN` role, you should delete the contents of this view so that it does not grow too large. For example: `DELETE FROM DBA_AUDIT_MGMT_CLEAN_EVENTS;` This view applies to read-write databases only. For read-only databases, a history of purge events is in the alert log. For unified auditing, you can find a history of purged events by querying the `UNIFIED_AUDIT_TRAIL` data dictionary view, using the following criteria: `OBJECT_NAME` is `DBMS_AUDIT_MGMT`, `OBJECT_SCHEMA` is `SYS`, and `SQL_TEXT` is set to `LIKE %DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL%`.
`DBA_AUDIT_MGMT_CLEANUP_JOBS`	Displays the currently configured audit trail purge jobs
`DBA_AUDIT_MGMT_CONFIG_PARAMS`	Displays the currently configured audit trail properties that are used by the `DBMS_AUDIT_MGMT` PL/SQL package
`DBA_AUDIT_MGMT_LAST_ARCH_TS`	Displays the last archive timestamps that have set for audit trail purges

See Also:

Oracle Database Reference for detailed information about these views

Part VII
Appendixes

Part VII contains a set of reference appendixes.

A

Keeping Your Oracle Database Secure

Oracle provides guidelines for keeping your database secure, such as advice on securing user accounts, privileges, roles, passwords, and data.

About the Oracle Database Security Guidelines

Information security, and privacy and protection of corporate assets and data are critical in any business.

Oracle Database comprehensively addresses the need for information security by providing cutting-edge security features such as deep data protection, auditing, scalable security, secure hosting, and data exchange.

Oracle Database leads the industry in security. To maximize the security features offered by Oracle Database in any business environment, it is imperative that the database itself be well protected.

Security guidelines provide advice about how to configure Oracle Database to be secure by adhering to and recommending industry-standard and advisable security practices for operational database deployments. Many of the guidelines described in this section address common regulatory requirements such as those described in the Sarbanes-Oxley Act. For more information about how Oracle Database addresses regulatory compliance, protection of personally identifiable information, and internal threats, visit:

`http://www.oracle.com/technetwork/topics/security/whatsnew/index.html`

Downloading Security Patches and Contacting Oracle Regarding Vulnerabilities

You should always apply security patches as soon as they are available. If problems arise, then you should contact Oracle regarding vulnerabilities.

Downloading Security Patches and Workaround Solutions

Security patches apply to the operating system on which Oracle Database resides, Oracle Database itself, and all installed Oracle Database options and components.

- To download security patches and workaround solutions:
  - For security patches, periodically check the security site on Oracle Technology Network for details about security alerts released by Oracle at `http://www.oracle.com/technetwork/topics/security/alerts-086861.html`
  - Check the Oracle Worldwide Support Service site, My Oracle Support, for details about available and upcoming security-related patches at

    `https://support.oracle.com`

Contacting Oracle Security Regarding Vulnerabilities in Oracle Database

You can contact Oracle Security regarding vulnerabilities in Oracle Database.

- Contact Oracle Security using either of the following methods:
  - If you are an Oracle customer or an Oracle partner, use My Oracle Support to submit a Service Request on any potential Oracle product security vulnerability.
  - Send an email to `secalert_us@oracle.com` with a complete description of the problem, including product version and platform, together with any scripts and examples. Oracle encourages those who want to contact Oracle Security to employ email encryption, using our encryption key.

Guidelines for Securing User Accounts and Privileges

Oracle provides guidelines to secure user accounts and privileges.

1. **Lock and expire default (predefined) user accounts.**

   Oracle Database installs with several default database user accounts. Upon successful installation of the database, the Database Configuration Assistant automatically locks and expires most default database user accounts.

   If you perform a manual (without using Database Configuration Assistant) installation of Oracle Database, then no default database users are locked upon successful installation of the database server. Or, if you have upgraded from a previous release of Oracle Database, you may have default accounts from earlier releases. Left open in their default states, these user accounts can be exploited, to gain unauthorized access to data or disrupt database operations.

   You should *lock* and *expire* all default database user accounts. Oracle Database provides SQL statements to perform these operations. For example:

   ```
   ALTER USER ANONYMOUS PASSWORD EXPIRE ACCOUNT LOCK;
   ```

   See *Oracle Database SQL Language Reference* for more information about the `ALTER USER` statement.

   Installing additional products and components after the initial installation also results in creating more default database accounts. Database Configuration Assistant automatically locks and expires all additionally created database user accounts. Unlock only those accounts that need to be accessed on a regular basis and assign a strong, meaningful password to each of these unlocked accounts. Oracle provides SQL and password management to perform these operations.

   If any default database user account other than the ones left open is required for any reason, then a database administrator (DBA) must unlock and activate that account with a new, secure password.

   See *Oracle Database 2 Day + Security Guide* for a description of the predefined user accounts that are created when you install Oracle Database.

   If a default database user account, other than the ones left open, is required for any reason, then a database administrator (DBA) can unlock and activate that account with a new, secure password.

Securing Oracle Enterprise Manager Accounts

If you install Oracle Enterprise Manager, the `SYSMAN` and `DBSNMP` accounts are open, unless you configure Oracle Enterprise Manager for central administration. In this case, the `SYSMAN` account (if present) will be locked.

If you do not install Oracle Enterprise Manager, then only the `SYS` and `SYSTEM` accounts are open. Database Configuration Assistant locks and expires all other accounts (including `SYSMAN` and `DBSNMP`).

2. **Discourage users from using the NOLOGGING clause in SQL statements.**

   In some SQL statements, the user has the option of specifying the `NOLOGGING` clause, which indicates that the database operation is not logged in the online redo log file. Even though the user specifies the clause, a redo record is still written to the online redo log file. However, there is no data associated with this record. Because of this, using `NOLOGGING` has the potential for malicious code to be entered can be accomplished without an audit trail.

3. **Practice the principle of least privilege.**

   Oracle recommends the following guidelines:

   a. **Grant necessary privileges only.**

      Do not provide database users or roles more privileges than are necessary. (If possible, grant privileges to roles, not users.) In other words, the *principle of least privilege* is that users be given only those privileges that are actually required to efficiently perform their jobs.

      To implement this principle, restrict the following as much as possible:

      - The number of `SYSTEM` and `OBJECT` privileges granted to database users.
      - The number of people who are allowed to make `SYS`-privileged connections to the database.
      - The number of users who are granted the `ANY` privileges, such as the `DROP ANY TABLE` privilege. For example, there is generally no need to grant `CREATE ANY TABLE` privileges to a non-DBA-privileged user.
      - The number of users who are allowed to perform actions that create, modify, or drop database objects, such as the `TRUNCATE TABLE`, `DELETE TABLE`, `DROP TABLE` statements, and so on.

   b. **Limit granting the CREATE ANY EDITION and DROP ANY EDITION privileges.**

      To maintain additional versions of objects, editions can increase resource and disk space consumption in the database. Only grant the `CREATE ANY EDITION` and `DROP ANY EDITION` privileges to trusted users who are responsible for performing upgrades.

   c. **Re-evaluate the SELECT object privilege and SELECT ANY TABLE system privileges that you have granted to users.**

      If you want to restrict users to only being able to query tables, views, materialized views, and synonyms, then grant users the `READ` object privilege, or for trusted users only, the `READ ANY TABLE` system privilege. If in addition to performing query operations, you want users to be able to lock tables in exclusive mode or perform `SELECT ... FOR UPDATE` statements, then grant the user the `SELECT` object privilege or, for trusted users only, the `SELECT ANY TABLE` system privilege.

d. **Restrict the CREATE ANY JOB, BECOME USER, EXP_FULL_DATABASE, and IMP_FULL_DATABASE privileges. Also restrict grants of the CREATE DIRECTORY and CREATE ANY DIRECTORY privileges.**

   These are powerful security-related privileges. Only grant these privileges to users who need them.

e. **Restrict the BECOME USER privilege to users of Oracle Data Pump, Oracle Streams, and the DBMS_WORKLOAD_CAPTURE and DBMS_WORKLOAD_REPLAY packages.**

   The `BECOME USER` privilege is used only for the following subsystems:

   - Oracle Data Pump Import utilities `impdp` and `imp`, to assume the identity of another user to perform operations that cannot be directly performed by a third party (for example, loading objects such as object privilege grants). In an Oracle Database Vault environment, Database Vault provides several levels of required authorization that affect grants of `BECOME USER`. See *Oracle Database Vault Administrator's Guide*.
   - Oracle Streams, to enable administrators to create or alter capture users and apply users in a Streams environment. By default, `BECOME USER` is part of the `DBA` role. However, if Database Vault is enabled, then the `BECOME USER` privilege is removed from the `DBA` role. Therefore, `BECOME USER` is needed by Streams only in an environment where Database Vault is enabled.
   - `DBMS_WORKLOAD_CAPTURE` and `DBMS_WORKLOAD_REPLAY` PL/SQL packages, as a required privilege to be granted to users who must use these packages.

   If you use the `AUTHID CURRENT_USER` clause when invoking one of these subsystems (for example, in static references in PL/SQL code), then ensure that the `CURRENT_USER` is granted the `BECOME USER` privilege, either by a direct grant or through a role.

f. **Restrict library-related privileges to trusted users only.**

   The `CREATE LIBRARY`, `CREATE ANY LIBRARY`, `ALTER ANY LIBRARY`, and `EXECUTE ANY LIBRARY` privileges, and grants of `EXECUTE ON` *`library_name`* convey a great deal of power to users. If you plan to create PL/SQL interfaces to libraries, only grant the `EXECUTE` privilege to the PL/SQL interface. Do not grant `EXECUTE` on the underlying library. You must have the `EXECUTE` privilege on a library to create the PL/SQL interface to it. However, users have this privilege implicitly on libraries that they create in their own schemas. Explicit grants of `EXECUTE ON` *`library_name`* are rarely required. Only make an explicit grant of these privileges to trusted users, and never to the `PUBLIC` role.

g. **Restrict synonym-related privileges to trusted users only.**

   The `CREATE PUBLIC SYNONYM` and `DROP PUBLIC SYNONYM` system privileges convey a great deal of power to these users. Do not grant these privileges to users, unless they are trusted.

h. **Do not allow non-administrative users access to objects owned by the SYS schema.**

   Do not allow users to alter table rows or schema objects in the `SYS` schema, because doing so can compromise data integrity. Limit the use of statements such as `DROP TABLE`, `TRUNCATE TABLE`, `DELETE`, `INSERT`, or similar object-modification statements on `SYS` objects only to highly privileged administrative users.

The SYS schema owns the data dictionary. You can protect the data dictionary by setting the O7_DICTIONARY_ACCESSIBILITY parameter to FALSE. See Guidelines for Securing Data for more information.

i. **Only grant the EXECUTE privilege on the DBMS_RANDOM PL/SQL package to trusted users.**

   The EXECUTE privilege on the DBMS_RANDOM package could permit users who normally should have only minimal access to execute the functions associated with this package.

j. **Restrict permissions on run-time facilities.**

   Many Oracle Database products use run-time facilities, such as Oracle Java Virtual Machine (OJVM). Do not assign all permissions to a database run-time facility. Instead, grant specific permissions to the explicit document the root file paths for facilities that might run files and packages outside the database.

   Here is an example of a vulnerable run-time call, which individual files are specified:

   ```
   call dbms_java.grant_permission('wsmith',
   'SYS:java.io.FilePermission','<<ALL FILES>>','read');
   ```

   Here is an example of a better (more secure) run-time call, which specifies a directory path instead:

   ```
   call dbms_java.grant_permission('wsmith',
   'SYS:java.io.FilePermission','<<actual directory path>>','read');
   ```

4. **Revoke access to the following:**
   - The SYS.USER_HISTORY$ table from all users except SYS and DBA accounts
   - The RESOURCE role from typical application accounts
   - The CONNECT role from typical application accounts
   - The DBA role from users who do not need this role
5. **Grant privileges only to roles.**

   Granting privileges to roles and not individual users makes the management and tracking of privileges much easier.
6. **Limit the proxy account (for proxy authorization) privileges to CREATE SESSION only.**
7. **Use secure application roles to protect roles that are enabled by application code.**

   Secure application roles allow you to define a set of conditions, within a PL/SQL package, that determine whether or not a user can log on to an application. Users do not need to use a password with secure application roles.

   Another approach to protecting roles from being enabled or disabled in an application is the use of role passwords. This approach prevents a user from directly accessing the database in SQL (rather than the application) to enable the privileges associated with the role. However, Oracle recommends that you use secure application roles instead, to avoid having to manage another set of passwords.
8. **Create privilege captures to find excessively granted privileges.** See *Oracle Database Vault Administrator’s Guide* for more information.

9. **Monitor the granting of the following privileges only to users and roles who need these privileges.**

   By default, Oracle Database audits the following privileges:

   - `ALTER SYSTEM`
   - `AUDIT SYSTEM`
   - `CREATE EXTERNAL JOB`

   Oracle recommends that you also audit the following privileges:

   - `ALL PRIVILEGES` (which includes privileges such as `BECOME USER`, `CREATE LIBRARY`, and `CREATE PROCEDURE`)
   - `DBMS_BACKUP_RESTORE` package
   - `EXECUTE` to `DBMS_SYS_SQL`
   - `SELECT ANY TABLE`
   - `SELECT` on `PERFSTAT.STATS$SQLTEXT`
   - `SELECT` on `PERFSTAT.STATS$SQL_SUMMARY`
   - `SELECT` on `SYS.SOURCE$`
   - Privileges that have the `WITH ADMIN` clause
   - Privileges that have the `WITH GRANT` clause
   - Privileges that have the `CREATE` keyword

10. **Use the following data dictionary views to find information about user access to the database.**

    - `DBA_*`
    - `DBA_ROLES`
    - `DBA_SYS_PRIVS`
    - `DBA_ROLE_PRIVS`
    - `DBA_TAB_PRIVS`
    - `DBA_AUDIT_TRAIL` (if standard auditing is enabled)
    - `DBA_FGA_AUDIT_TRAIL` (if fine-grained auditing is enabled)

Guidelines for Securing Roles

Oracle provides guidelines for role management.

1. **Grant a role to users only if they need all privileges of the role.**

   Roles (groups of privileges) are useful for quickly and easily granting permissions to users. Although you can use Oracle-defined roles, you have more control and continuity if you create your own roles containing only the privileges pertaining to your requirements. Oracle may change or remove the privileges in an Oracle Database-defined role, as it has with the `CONNECT` role, which now has only the `CREATE SESSION` privilege. Formerly, this role had eight other privileges.

   Ensure that the roles you define contain only the privileges that reflect job responsibility. If your application users do not need all the privileges encompassed by an existing role, then apply a different set of roles that supply just the correct privileges. Alternatively, create and assign a more restricted role.

For example, it is imperative to strictly limit the privileges of user `SCOTT`, because this is a well known account that may be vulnerable to intruders. Because the `CREATE DBLINK` privilege allows access from one database to another, drop its privilege for `SCOTT`. Then, drop the entire role for the user, because privileges acquired by means of a role cannot be dropped individually. Re-create your own role with only the privileges needed, and grant that new role to that user. Similarly, for better security, drop the `CREATE DBLINK` privilege from all users who do not require it.

2. **Do not grant user roles to application developers.**

   Roles are not meant to be used by application developers, because the privileges to access schema objects within stored programmatic constructs need to be granted directly. Remember that roles are not enabled within stored procedures except for invoker's right procedures. See How Roles Work in PL/SQL Blocks for information about this topic.

3. **Create and assign roles specific to each Oracle Database installation.**

   This principle enables the organization to retain detailed control of its roles and privileges. This also avoids the necessity to adjust if Oracle Database changes or removes Oracle Database-defined roles, as it has with `CONNECT`, which now has only the `CREATE SESSION` privilege. Formerly, it also had eight other privileges.

4. **For enterprise users, create global roles.**

   Global roles are managed by an enterprise directory service, such as Oracle Internet Directory. See the following sections for more information about global roles:

   - Global User Authentication and Authorization
   - Authorizing a Global Role by an Enterprise Directory Service
   - *Oracle Database Enterprise User Security Administrator's Guide*

Guidelines for Securing Passwords

Oracle provides guidelines for securing passwords.

When you create a user account, Oracle Database assigns a default password policy for that user. The password policy defines rules for how the password should be created, such as a minimum number of characters, when it expires, and so on. You can strengthen passwords by using password policies. See also Configuring Password Protection for additional ways to protect passwords.

Follow these guidelines to further strengthen passwords:

1. **Choose passwords carefully.**

   Minimum Requirements for Passwords describes the minimum requirements for passwords. Follow these additional guidelines when you create or change passwords:

   - Make the password between 12 and 30 characters and numbers.
   - Have the password contain at least one digit, one upper-case character, and one lower-case character.

- Use mixed case letters and special characters in the password. (See Ensuring Against Password Security Threats by Using the 12C Password Version for more information.)
- You can include multibyte characters in the password.
- Use the database character set for the password's characters, which can include the underscore (_), dollar ($), and number sign (#) characters.
- You must enclose the following passwords in double-quotation marks:
  - Passwords containing multibyte characters.
  - Passwords starting with numbers or special characters and containing alphabetical characters. For example:

    `"123abc"`

    `"#abc"`

    `"123dc$"`
  - Passwords containing any character other than alphabetical characters, numbers, and special characters. For example:

    `"abc>"`

    `"abc@",`

    `" "`
- You do not need to specify the following passwords in double-quotation marks.
  - Passwords starting with an alphabet character (a–z, A–Z) and containing numbers(0–9) or special characters ($, #, _). For example:

    `abc123`

    `ab23a`

    `ab$#_`
  - Passwords containing only numbers.
  - Passwords containing only alphabetical characters.
- Do not include double-quotation marks within the password.
- Do not use an actual word for the entire password.

2. **To create a longer, more complex password from a shorter, easier to remember password, follow these techniques:**
   - Create passwords from the first letters of the words of an easy-to-remember sentence. For example, "I usually work until 6:00 almost every day of the week" can be `Iuwu6aedotw`.
   - Combine two weaker passwords, such as `welcome1` and `binky` into `WelBinkyCome1`.
   - Repeat a character at the beginning or end of the password.
   - Add a string, another password, or part the same password to the beginning or end of the password that you want to create. For example, ways that you can modify the password `fussy2all` are as follows:
     - `fussy2all34hj2`
     - `WelBinkyCome1fussy2all`

    – `fusfussy2all`

  - Double some or all of the letters. For example, `welcome13` can become `wwellCcooMmee13`.

3. **Ensure that the password is sufficiently complex.**

   Oracle Database provides a password complexity verification routine, the PL/SQL script `utlpwdmg.sql`, that you can run to check whether or not passwords are sufficiently complex. Ideally, edit the `utlpwdmg.sql` script to provide stronger password protections. See also About Password Complexity Verification for a sample routine that you can use to check passwords.

4. **Associate a password complexity function with the user profile or the default profile.**

   The `PASSWORD_VERIFY_FUNCTION` clause of the `CREATE PROFILE` and `ALTER PROFILE` statements associates a password complexity function with a user profile or the default profile. Password complexity functions ensure that users create strong passwords using guidelines that are specific to your site. Having a password complexity function also requires a user changing his or her own password (without the `ALTER USER` system privilege) to provide both the old and new passwords. You can create your own password complexity functions or use the password complexity functions that Oracle Database provides.

   See Managing the Complexity of Passwords for more information.

5. **Change default user passwords.**

   Oracle Database installs with a set of predefined, default user accounts. Security is most easily broken when a default database user account still has a default password *even after installation*. This is particularly true for the user account `SCOTT`, which is a well known account that may be vulnerable to intruders. In Oracle Database, default accounts are installed locked with the passwords expired, but if you have upgraded from a previous release, you may still have accounts that use default passwords.

   To find user accounts that have default passwords, query the `DBA_USERS_WITH_DEFPWD` data dictionary view. See Finding User Accounts That Have Default Passwords for more information.

6. **Change default passwords of administrative users.**

   You can use the same or different passwords for the `SYS`, `SYSTEM`, `SYSMAN`, and `DBSNMP` administrative accounts. Oracle recommends that you use different passwords for each. In any Oracle environment (production or test), assign strong, secure, and distinct passwords to these administrative accounts. If you use Database Configuration Assistant to create a new database, then it requires you to enter passwords for the `SYS` and `SYSTEM` accounts, disallowing the default passwords `CHANGE_ON_INSTALL` and `MANAGER`.

   Similarly, for production environments, do not use default passwords for administrative accounts, including `SYSMAN` and `DBSNMP`.

   See *Oracle Database 2 Day + Security Guide* for information about changing a default password.

7. **Enforce password management.**

   Apply basic password management rules (such as password length, history, complexity, and so forth) to all user passwords. Oracle Database has password policies enabled for the default profile. Guideline 1 in this section lists these password policies. *Oracle Database 2 Day + Security Guide* lists initialization parameters that you can use to further secure user passwords.

You can find information about user accounts by querying the `DBA_USERS` view. The `PASSWORD` column of the `DBA_USERS` view indicates whether the password is global, external, or null. The `DBA_USERS` view provides useful information such as the user account status, whether the account is locked, and password versions.

Oracle also recommends, if possible, using Oracle strong authentication with network authentication services (such as Kerberos), token cards, smart cards, or X.509 certificates. These services provide strong authentication of users, and provide protection against unauthorized access to Oracle Database.

8. **Do not store user passwords in clear text in Oracle tables.**

   For better security, do not store passwords in clear text (that is, human readable) in Oracle tables. You can correct this problem by using a secure external password store to encrypt the table column that contains the password. See Managing the Secure External Password Store for Password Credentials for information.

   When you create or modify a password for a user account, Oracle Database automatically creates a cryptographic hash or digest of the password. If you query the `DBA_USERS` view to find information about a user account, the data in the `PASSWORD` column indicates if the user password is global, external, or null.

Guidelines for Securing Data

Oracle provides guidelines for securing data on your system.

1. **Enable data dictionary protection.**

   Oracle recommends that you protect the data dictionary to prevent users that have the `ANY` system privilege from using those privileges on the data dictionary. Altering or manipulating the data in data dictionary tables can permanently and detrimentally affect the operation of a database.

   To enable data dictionary protection, set the following initialization parameter to `FALSE` (which is the default) in the `init`*`sid`*`.ora` control file:

   ```
   O7_DICTIONARY_ACCESSIBILITY = FALSE
   ```

   You can set the `O7_DICTIONARY_ACCESSIBILITY` parameter in a server parameter file. For more information about server parameter files, see *Oracle Database Vault Administrator's Guide*.

   After you set `O7_DICTIONARY_ACCESSIBILTY` to `FALSE`, only users who have the `SELECT ANY DICTIONARY` privilege and those authorized users making DBA-privileged (for example `CONNECT / AS SYSDBA`) connections can use the `ANY` system privilege on the data dictionary. If `O7_DICTIONARY_ACCESSIBILITY` parameter is not set to `FALSE`, then any user with the `DROP ANY TABLE` (for example) system privilege will be able to drop parts of the data dictionary. However, if a user *needs* view access to the data dictionary, then you can grant that user the `SELECT ANY DICTIONARY` system privilege.

Note:

- In a default installation, the `O7_DICTIONARY_ACCESSIBILITY` parameter is set to `FALSE`. However, in Oracle8*i*, this parameter is set to `TRUE` by default, and must be changed to `FALSE` to enable this security feature.
- The `SELECT ANY DICTIONARY` privilege is not included in the `GRANT ALL PRIVILEGES` statement, but you can grant it through a role. Configuring Privilege and Role Authorization, describes roles in detail.

2. **Restrict operating system access.**

   Follow these guidelines:

   - Limit the number of operating system users.
   - Limit the privileges of the operating system accounts (administrative, root-privileged, or database administrative) on the Oracle Database host computer to the least privileges required for a user to perform necessary tasks.
   - Restrict the ability to modify the default file and directory permissions for the Oracle Database home (installation) directory or its contents. Even privileged operating system users and the Oracle owner should not modify these permissions, unless instructed otherwise by Oracle.
   - Restrict symbolic links. Ensure that when you provide a path or file to the database, neither the file nor any part of the path is modifiable by an untrusted user. The file and all components of the path should be owned by the database administrator or trusted account, such as *root*.

     This recommendation applies to all types of log files, trace files, external tables, BFILE data types, and so on.

3. **Encrypt sensitive data and all backup media that contains database files.**

   According to common regulatory compliance requirements, you must encrypt sensitive data such as credit card numbers and passwords. When you delete sensitive data from the database, encrypted data does not linger in data blocks, operating system files, or sectors on disk.

   In most cases, you may want to use Transparent Data Encryption to encrypt your sensitive data. See *Oracle Database Advanced Security Guide* for more information. See also Security Problems That Encryption Does Not Solve for when you should not encrypt data.

4. **For Oracle Automatic Storage Management (Oracle ASM) environments on Linux and UNIX systems, use Oracle ASM File Access Control to restrict access to the Oracle ASM disk groups.**

   If you use different operating system users and groups for Oracle Database installations, then you can configure Oracle ASM File Access Control to restrict the access to files in Oracle ASM disk groups to only authorized users. For example, a database administrator would only be able to access the data files for the databases he or she manages. This administrator would not be able to see or overwrite the data files belonging (or used by) other databases.

   For more information about managing Oracle ASM File Access Control for disk groups, see *Oracle Automatic Storage Management Administrator's Guide*. For information about the various privileges required for multiple software owners, see also *Oracle Automatic Storage Management Administrator's Guide*.

Guidelines for Securing the ORACLE_LOADER Access Driver

Oracle provides guidelines to secure the `ORACLE_LOADER` access driver.

1. **Create a separate operating system directory to store the access driver preprocessors.** You (or the operating system manager) may need to create multiple directories if different Oracle Database users will run different preprocessors. If you want to prevent one set of users from using one preprocessor while allowing those users access to another preprocessor, then place the preprocessors in separate directories. If all the users need equal access, then you can place the preprocessors together in one directory. After you create these operating system directories, in SQL*Plus, you can create a directory object for each directory.
2. **Grant the operating system user ORACLE the correct operating system privileges to run the access driver preprocessor.** In addition, protect the preprocessor program from `WRITE` access by operating system users other than the user responsible for managing the preprocessor program.
3. **Grant the EXECUTE privilege to each user who will run the preprocessor program in the directory object.** Do not grant this user the `WRITE` privilege on the directory object. Never grant users both the `EXECUTE` and `WRITE` privilege for directory objects.
4. **Grant the WRITE privilege sparingly to anyone who will manage directory objects that contain preprocessors.** This prevents database users from accidentally or maliciously overwriting the preprocessor program.
5. **Create a separate operating system directory and directory object for any data files that are required for external tables.** Ensure that these are separate from the directory and directory object used by the access directory preprocessor.

   Work with the operating system manager to ensure that only the appropriate operating system users have access to this directory. Grant the `ORACLE` operating system user `READ` access to any directory that has a directory object with `READ` privileges granted to database users. Similarly, grant the `ORACLE` operating system user `WRITE` access to any directory that has the `WRITE` privilege granted to database users.
6. **Create a separate operating system directory and directory object for any files that the access driver generates.** This includes log files, bad files, and discarded files. You and the operating system manager must ensure that this directory and directory object have the proper protections, similar to those described in Guideline 5. The database user may need to access these files when resolving problems in data files, so you and the operating system manager must determine a way for this user to read those files.
7. **Grant the CREATE ANY DIRECTORY and DROP ANY DIRECTORY privileges sparingly.** Users who have these privileges and users who have been granted the `DBA` role have full access to all directory objects.
8. **Consider auditing the DROP ANY DIRECTORY privilege.** See Auditing System Privileges for more information about auditing privileges.

9. **Consider auditing the directory object.** See Auditing Object Actions for more information.

> **See Also:**
>
> *Oracle Database Utilities* for more information about the `ORACLE_DATAPUMP` access driver

Guidelines for Securing a Database Installation and Configuration

Oracle provides guidelines to secure the database installation and configuration.

Changes were made to the default configuration of Oracle Database to make it more secure. The recommendations in this section augment the new, secure default configuration.

1. **Before you begin an Oracle Database installation on UNIX systems, ensure that the umask value is 022 for the Oracle owner account.**

   See *Oracle Database Administrator's Reference for Linux and UNIX-Based Operating Systems* for more information about managing Oracle Database on Linux and UNIX systems.

2. **Install only what is required.**

   Options and Products: The Oracle Database CD pack contains products and options in addition to the database. Install additional products and options only as necessary. Use the Custom Installation feature to avoid installing unnecessary products, or perform a typical installation, and then deinstall options and products that are not required. There is no need to maintain additional products and options if they are not being used. They can always be properly installed, as required.

   Sample Schemas: Oracle Database provides sample schemas to provide a common platform for examples. If your database will be used in a production environment, then do not install the sample schema. If you have installed the sample schema on a test database, then before going to production, remove or relock the sample schema accounts. See *Oracle Database Sample Schemas* for more information about the sample schemas.

3. **During installation, when you are prompted for a password, create a secure password.**

   Follow Guidelines 1, 5, and 6 in Guidelines for Securing Passwords.

4. **Immediately after installation, lock and expire default user accounts.**

   See Guideline 1 in Guidelines for Securing User Accounts and Privileges.

Guidelines for Securing the Network

Security for network communications is improved by using client, listener, and network guidelines to ensure thorough protection.

Client Connection Security

Authenticating clients stringently, configuring encryption for the connection, and using strong authentication strengthens client connections.

Because authenticating client computers is problematic, typically, user authentication is performed instead. This approach avoids client system issues that include falsified IP addresses, hacked operating systems or applications, and falsified or stolen client system identities.

Nevertheless, the following guidelines improve the security of client connections:

1. **Enforce access controls effectively and authenticate clients stringently.**

   By default, Oracle allows operating system-authenticated logins only over secure connections, which precludes using Oracle Net and a shared server configuration. This default restriction prevents a user over a network connection.

   Setting the initialization parameter `REMOTE_OS_AUTHENT` to `TRUE` forces the database to accept the client operating system user name received over an unsecure connection and use it for account access. Because clients, such as PCs, are not trusted to perform operating system authentication properly, it is poor security practice to use this feature.

   The default setting, `REMOTE_OS_AUTHENT = FALSE`, creates a more secure configuration that enforces proper, server-based authentication of clients connecting to an Oracle database. Be aware that the `REMOTE_OS_AUTHENT` was deprecated in Oracle Database Release 11*g* (11.1) and is retained only for backward compatibility.

   You should not alter the default setting of the `REMOTE_OS_AUTHENT` initialization parameter, which is `FALSE`.

   Setting this parameter to `FALSE` does not mean that users cannot connect remotely. It means that the database will not trust that the client has already authenticated, and will therefore apply its standard authentication processes.

   Be aware that the `REMOTE_OS_AUTHENT` parameter was deprecated in Oracle Database 11*g* Release 1 (11.1), and is retained only for backward compatibility.

2. **Configure the connection to use encryption.**

   Oracle network encryption makes eavesdropping difficult. To learn how to configure encryption, see *Oracle Database Advanced Security Guide*.

3. **Set up strong authentication.**

   See Configuring Kerberos Authentication , for more information about using Kerberos and public key infrastructure (PKI).

4. **In an Oracle Data Guard environment, set the `ADG_ACCOUNT_INFO_TRACKING` initialization parameter.**

   The `ADG_ACCOUNT_INFO_TRACKING` parameter controls login attempts on Oracle Active Data Guard standby databases. It provides more security against login attacks across an Oracle Database production environment and all Active Data Guard standby databases. Use one of the following settings:

   - `LOCAL` (default) enforces the existing behavior, which maintains a local copy of user account information in the standby database's in-memory view. This setting only tracks login failures locally on a per-database basis. It denies the login when the maximum of failed logins is reached.

- `GLOBAL` increases the security of logins by maintaining a single global copy of user account information across all Data Guard primary and standby databases. Login failures across all databases in the Data Guard environment count toward the maximum count. When this count is reached, then logins anywhere are denied access.

To learn more about the `ADG_ACCOUNT_INFO_TRACKING` parameter, see *Oracle Database Reference*

Network Connection Security

Protecting the network and its traffic from inappropriate access or modification is the essence of network security.

You should consider all paths the data travels, and assess the threats on each path and node. Then, take steps to lessen or eliminate those threats and the consequences of a security breach. In addition, monitor and audit to detect either increased threat levels or penetration attempts.

To manage network connections, you can use Oracle Net Manager. For more information about Net Manager, see *Oracle Database Net Services Administrator's Guide*.

The following practices improve network security:

1. **Use Secure Sockets Layer (SSL) when administering the listener.**

   See Secure Sockets Layer Connection Security for more information.

2. **Prevent online administration by requiring the administrator to have the write privilege on the listener password and on the listener.ora file on the server.**

   a. Add or alter this line in the `listener.ora` file:

      ```
      ADMIN_RESTRICTIONS_LISTENER=ON
      ```

   b. Use `RELOAD` to reload the configuration.

   c. Use SSL when administering the listener by making the TCPS protocol the first entry in the address list, as follows:

      ```
      LISTENER=
        (DESCRIPTION=
          (ADDRESS_LIST=
            (ADDRESS=
              (PROTOCOL=tcps)
              (HOST = sales.us.example.com)
              (PORT = 8281)))
      ```

      To administer the listener remotely, you define the listener in the `listener.ora` file on the client computer. For example, to access listener USER281 remotely, use the following configuration:

      ```
      user281 =
        (DESCRIPTION =
          (ADDRESS =
            (PROTOCOL = tcps)
            (HOST = sales.us.example.com)
      ```

```
          (PORT = 8281))
        )
      )
```

For more information about the parameters in `listener.ora`, see *Oracle Database Net Services Reference*.

3. **Do not set the listener password.**

   Ensure that the password has not been set in the `listener.ora` file. The local operating system authentication will secure the listener administration. The remote listener administration is disabled when the password has not been set. This prevents brute force attacks of the listener password.

   The listener password has been deprecated in this release. It will not be supported in the next release of Oracle Database.

4. **When a host computer has multiple IP addresses associated with multiple network interface controller (NIC) cards, configure the listener to the specific IP address.**

   This allows the listener to listen on all the IP addresses. You can restrict the listener to listen on a specific IP address. Oracle recommends that you specify the specific IP addresses on these types of computers, rather than allowing the listener to listen on all IP addresses. Restricting the listener to specific IP addresses helps to prevent an intruder from stealing a TCP end point from under the listener process.

5. **Restrict the privileges of the listener, so that it cannot read or write files in the database or the Oracle server address space.**

   This restriction prevents external procedure agents spawned by the listener (or procedures executed by an agent) from inheriting the ability to perform read or write operations. The owner of this separate listener process should not be the owner that installed Oracle Database or executes the Oracle Database instance (such as `ORACLE`, the default owner).

   For more information about configuring external procedures in the listener, see *Oracle Database Net Services Administrator's Guide*.

6. **Use encryption to secure the data in flight.**

   See *Oracle Database 2 Day + Security Guide* and Introduction to Strong Authentication, for more information about network data encryption.

7. **Use a firewall.**

   Appropriately placed and configured firewalls can prevent outside access to your databases.

   - Keep the database server behind a firewall. Oracle Database network infrastructure, Oracle Net Services (formerly known as SQL*Net), provides support for a variety of firewalls from various vendors. Supported proxy-enabled firewalls include Gauntlet from Network Associates and Raptor from Axent. Supported packet-filtering firewalls include PIX Firewall from Cisco, and supported stateful inspection firewalls (more sophisticated packet-filtered firewalls) include Firewall-1 from CheckPoint.
   - Ensure that the firewall is placed outside the network to be protected.
   - Configure the firewall to accept only those protocols, applications, or client/server sources that you know are safe.

- Use a product such as Net8 and Oracle Connection Manager to manage multiplex multiple client network sessions through a single network connection to the database. It can filter on source, destination, and host name. This product enables you to ensure that connections are accepted only from physically secure terminals or from application Web servers with known IP addresses. (Filtering on IP address alone is not enough for authentication, because it can be falsified.)

8. **Prevent unauthorized administration of the Oracle listener.**

   For more information about the listener, see *Oracle Database Net Services Administrator's Guide*.

9. **Check network IP addresses.**

   Use the Oracle Net *valid node checking* security feature to allow or deny access to Oracle server processes from network clients with specified IP addresses. To use this feature, set the following `sqlnet.ora` configuration file parameters:

   ```
   tcp.validnode_checking = YES

   tcp.excluded_nodes = {list of IP addresses}

   tcp.invited_nodes = {list of IP addresses}
   ```

   The `tcp.validnode_checking` parameter enables the feature. The `tcp.excluded_nodes` and `tcp.invited_nodes` parameters deny and enable specific client IP addresses from making connections to the Oracle listener. This helps to prevent potential Denial of Service attacks.

   You can use Oracle Net Manager to configure these parameters. See *Oracle Database Net Services Administrator's Guide* for more information.

10. **Encrypt network traffic.**

    If possible, use Oracle network data encryption to encrypt network traffic among clients, databases, and application servers. *Oracle Database 2 Day + Security Guide* provides an introduction to network encryption. For detailed information about network encryption, see Configuring Oracle Database Network Encryption and Data Integrity.

11. **Secure the host operating system (the system on which Oracle Database is installed).**

    Secure the host operating system by disabling all unnecessary operating system services. Both UNIX and Windows provide a variety of operating system services, most of which are not necessary for typical deployments. These services include FTP, TFTP, TELNET, and so forth. Be sure to close both the UDP and TCP ports for each service that is being disabled. Disabling one type of port and not the other does not make the operating system more secure.

12. **Configure database link communication protocol.**

    To specify the protocols over which the database link communication takes place, set the `OUTBOUND_DBLINK_PROTOCOLS` initialization parameter to one of the following settings:

    - `ALL` (default) enables all net protocols to be used for the database links.
    - *`comma-separated_list_of_protocols`* can be set `TPC`, `TCPS`, or `IPC`. For example, for a single protocol:

      ```
      ALTER SYSTEM SET OUTBOUND_DBLINK_PROTOCOLS=TCPS;
      ```

For multiple protocols:

```
ALTER SYSTEM SET OUTBOUND_DBLINK_PROTOCOLS=TCP,TCPS,IPC;
```

- `NONE` disables any database link communication.

13. **If necessary, disable LDAP lookup for global database links.**

    Set the `ALLOW_GLOBAL_DBLINKS` initialization parameter to enable or disable LDAP lookup for global database links. Settings are as follows:

    - `ON` enables LDAP lookup for global database links.
    - `OFF` (default) disables LDAP lookup for global database links.

Secure Sockets Layer Connection Security

Oracle provides guidelines for securing Secure Sockets Layer (SSL).

Secure Sockets Layer (SSL) is the Internet standard protocol for secure communication, providing mechanisms for data integrity and data encryption. These mechanisms can protect the messages sent and received by you or by applications and servers, supporting secure authentication, authorization, and messaging through certificates and, if necessary, encryption. Good security practices maximize protection and minimize gaps or disclosures that threaten security.

1. **Ensure that configuration files (for example, for clients and listeners) use the correct port for SSL, which is the port configured upon installation.**

   You can run HTTPS on any port, but the standards specify port 443, where any HTTPS-compliant browser looks by default. The port can also be specified in the URL, for example:

   ```
   https://secure.example.com:4445/
   ```

   If a firewall is in use, then it too must use the same ports for secure (SSL) communication.

2. **Ensure that TCPS is specified as the PROTOCOL in the ADDRESS parameter in the tnsnames.ora file (typically on the client or in the LDAP directory).**

   An identical specification must appear in the `listener.ora` file (typically in the `$ORACLE_HOME/network/admin` directory).

3. **Ensure that the SSL mode is consistent for both ends of every communication. For example, the database (on one side) and the user or application (on the other) must have the same SSL mode.**

   The mode can specify either client or server authentication (one-way), both client and server authentication (two-way), or no authentication.

4. **Ensure that the server supports the client cipher suites and the certificate key algorithm in use.**

5. **Enable DN matching for both the server and client, to prevent the server from falsifying its identity to the client during connections.**

   This setting ensures that the server identity is correct by matching its global database name against the DN from the server certificate.

   You can enable DN matching in the `tnsnames.ora` file. For example:

   ```
   set:SSL_SERVER_CERT_DN="cn=finance,cn=OracleContext,c=us,o=example"
   ```

Otherwise, a client application would not check the server certificate, which could allow the server to falsify its identity.

6. **Do not remove the encryption from your RSA private key inside your server.key file, which requires that you enter your pass phrase to read and parse this file.**

   Note:

   A server without SSL does not require a pass phrase.

   If you decide your server is secure enough, you could remove the encryption from the RSA private key while preserving the original file. This enables system boot scripts to start the database server, because no pass phrase is needed. Ideally, restrict permissions to the root user only, and have the Web server start as `root`, but then log on as another user. Otherwise, anyone who gets this key can impersonate you on the Internet, or decrypt the data that was sent to the server.

   See Also:

   - Configuring Secure Sockets Layer Authentication, for general SSL information, including configuration
   - *Oracle Database Net Services Reference* for TCP-related parameters in `sqlnet.ora`

Guideline for Securing External Procedures

The `ENFORCE_CREDENTIAL` environment variable controls how an `extproc` process authenticates user credentials and callout functions.

You can specify this variable in the `extproc.ora` file. Before modifying this variable, review your site's security requirements for the handling of external libraries. For maximum security, set the `ENFORCE_CREDENTIAL` variable to `TRUE`. The default setting is `FALSE`.

See Also:

Securing External Procedures

Guidelines for Auditing

Oracle provides guidelines for auditing.

Manageability of Audited Information

Although auditing is relatively inexpensive, limit the number of audited events as much as possible.

This minimizes the performance impact on the execution of audited statements and the size of the audit trail, making it easier to analyze and understand.

Follow these guidelines when devising an auditing strategy:

1. **Evaluate your reason for auditing.**

   After you have a clear understanding of the reasons for auditing, you can devise an appropriate auditing strategy and avoid unnecessary auditing.

   For example, suppose you are auditing to investigate suspicious database activity. This information by itself is not specific enough. What types of suspicious database activity do you suspect or have you noticed? A more focused auditing strategy might be to audit unauthorized deletions from arbitrary tables in the database. This purpose narrows the type of action being audited and the type of object being affected by the suspicious activity.

2. **Audit knowledgeably.**

   Audit the minimum number of statements, users, or objects required to get the targeted information. This prevents unnecessary audit information from cluttering the meaningful information and using valuable space in the `SYSTEM` tablespace. Balance your need to gather sufficient security information with your ability to store and process it.

   For example, if you are auditing to gather information about database activity, then determine exactly what types of activities you want to track, audit only the activities of interest, and audit only for the amount of time necessary to gather the information that you want. As another example, do not audit *objects* if you are only interested in logical I/O information for each session.

3. **Before you implement an auditing strategy, consult your legal department.**

   You should have the legal department of your organization review your audit strategy. Because your auditing will monitor other users in your organization, you must ensure that you are correctly following the compliance and corporate policy of your site.

Audits of Typical Database Activity

Oracle provides guidelines for when you must gather historical information about particular database activities.

1. **Audit only pertinent actions.**

   At a minimum, audit user access, the use of system privileges, and changes to the database schema structure. To avoid cluttering meaningful information with useless audit records and reduce the amount of audit trail administration, only audit the targeted database activities. Remember also that auditing too much can affect database performance.

   For example, auditing changes to all tables in a database produces far too many audit trail records and can slow down database performance. However, auditing changes to critical tables, such as salaries in a Human Resources table, is useful.

You can audit specific actions by using fine-grained auditing, which is described in Auditing Specific Activities with Fine-Grained Auditing.

2. **Archive audit records and purge the audit trail.**

   After you collect the required information, archive the audit records of interest and then purge the audit trail of this information. See the following sections:

   - Archiving the Audit Trail
   - Purging Audit Trail Records

3. **Remember your company's privacy considerations.**

   Privacy regulations often lead to additional business privacy policies. Most privacy laws require businesses to monitor access to personally identifiable information (PII), and monitoring is implemented by auditing. A business-level privacy policy should address all relevant aspects of data access and user accountability, including technical, legal, and company policy concerns.

4. **Check the Oracle Database log files for additional audit information**

   The log files generated by Oracle Database contain useful information that you can use when auditing a database. For example, an Oracle database creates an alert file to record `STARTUP` and `SHUTDOWN` operations, and structural changes such as adding data files to the database.

   For example, if you want to audit committed or rolled back transactions, you can use the redo log files.

Audits of Suspicious Database Activity

Oracle provides guidelines for when you audit to monitor suspicious database activity.

1. **First audit generally, and then specifically.**

   When you start to audit for suspicious database activity, often not much information is available to target specific users or schema objects. Therefore, audit generally first, that is, by using the unified audit policies. Configuring Audit Policies explains how you can audit SQL statements, schema objects, privileges, and so on.

   After you have recorded and analyzed the preliminary audit information, alter your audit policies to audit specific actions and privileges. You can add conditions to your policies to exclude unnecessary audit records. You an also use the `EXCEPT` clause in the `AUDIT POLICY` statement to exclude specific users who do not need to be audited. See Auditing Activities with Unified Audit Policies and the AUDIT Statement for more information about unified audit policies.

   You can use fine-grained auditing, which is described in Auditing Specific Activities with Fine-Grained Auditing, to audit specific actions.

   Continue this process until you have gathered enough evidence to draw conclusions about the origin of the suspicious database activity.

2. **Audit common suspicious activities.**

   Common suspicious activities are as follows:

   - Users who access the database during unusual hours
   - Multiple failed user login attempts
   - Login attempts by non-existent users

In addition, be aware that sensitive data, such as credit card numbers, can appear in the audit trail columns, such as SQL text when used in the SQL query. You should also monitor users who share accounts or multiple users who are logging in from the same IP address. You can query the `UNIFIED_AUDIT_TRAIL` data dictionary view to find this kind of activity. For a very granular approach, create fine-grained audit policies.

Recommended Audit Settings

Oracle provides predefined policies that contain recommended audit settings that apply to most sites.

For example:

- `ORA_SECURECONFIG` audits the same default audit settings from Oracle Database Release 11*g*. It tracks the use of a number of privileges such as `ALTER ANY TABLE`, `GRANT ANY PRIVILEGE`, and `CREATE USER`. The actions that it tracks include `ALTER USER`, `CREATE ROLE`, `LOGON`, and other commonly performed activities. This policy is enabled by default only when the database is created in Oracle Database Release 12*c*.
- `ORA_DATABASE_PARAMETER` audits commonly used Oracle Database parameter settings: `ALTER DATABASE`, `ALTER SYSTEM`, and `CREATE SPFILE`. By default, this policy is not enabled.
- `ORA_ACCOUNT_MGMT` audits the commonly used user account and privilege settings: `CREATE USER`, `ALTER USER`, `DROP USER`, `CREATE ROLE`, `DROP ROLE`,`ALTER ROLE`, `SET ROLE`, `GRANT`, and `REVOKE`. By default, this policy is not enabled.

See Also:

Auditing Activities with the Predefined Unified Audit Policies for detailed information about these and other predefined audit policies

Addressing the CONNECT Role Change

The `CONNECT` role, introduced with Oracle Database version 7, added new and robust support for database roles.

Why Was the CONNECT Role Changed?

The `CONNECT` role is used in sample code, applications, documentation, and technical papers.

In Oracle Database 10*g* Release 2 (10.2), the `CONNECT` role was changed. If you are upgrading from a release earlier than Oracle Database 10.2 to the current release, then you should be aware of how the `CONNECT` role has changed in the most recent release.

The `CONNECT` role was originally established a special set of privileges. These privileges were as follows: `ALTER SESSION`, `CREATE CLUSTER`, `CREATE DATABASE LINK`, `CREATE SEQUENCE`, `CREATE SESSION`, `CREATE SYNONYM`, `CREATE TABLE`, `CREATE VIEW`.

Beginning in Oracle Database 10*g* Release 2, the `CONNECT` role has only the `CREATE SESSION` privilege, all other privileges are removed. Starting with Oracle Database 12*c* Release 1, the `CONNECT` role had the `CREATE SESSION` and `SET CONTAINER` privileges.

Although the `CONNECT` role was frequently used to provision new accounts in Oracle Database, connecting to the database does not require all those privileges. Making this change enables you to enforce good security practices more easily.

Each user should have only the privileges needed to perform his or her tasks, an idea called the principle of least privilege. Least privilege mitigates risk by limiting privileges, so that it remains easy to do what is needed while concurrently reducing the ability to do inappropriate things, either inadvertently or maliciously.

How the CONNNECT Role Change Affects Applications

The `CONNECT` role changes can be seen in database upgrades, account provisioning, and installation of applications using new databases.

How the CONNECT Role Change Affects Database Upgrades

You should be aware of how the `CONNECT` role affects database upgrades.

Upgrading your existing Oracle database to Oracle Database 10*g* Release 2 (10.2) automatically changes the `CONNECT` role to have only the `CREATE SESSION` privilege.

Most applications are not affected because the applications objects already exist: no new tables, views, sequences, synonyms, clusters, or database links need to be created.

Applications that create tables, views, sequences, synonyms, clusters, or database links, or that use the `ALTER SESSION` command dynamically, may fail due to insufficient privileges.

How the CONNECT Role Change Affects Account Provisioning

You should be aware of how the `CONNECT` role affects accounts provisioning.

If your application or DBA grants the `CONNECT` role as part of the account provisioning process, then only `CREATE SESSION` privileges are included. Any additional privileges must be granted either directly or through another role.

This issue can be addressed by creating a new customized database role.

See Also:

Approaches to Addressing the CONNECT Role Change

How the CONNECT Role Change Affects Applications Using New Databases

You should be aware of how the `CONNECT` role affects applications that use new databases.

New databases created using the Oracle Database 10*g* Release 2 (10.2) Utility (DBCA), or using database creation templates generated from DBCA, define the `CONNECT` role with only the `CREATE SESSION` privilege.

Installing an application to use a new database may fail if the database schema used for the application is granted privileges solely through the `CONNECT` role.

How the CONNECT Role Change Affects Users

The change to the `CONNECT` role affects general users, application developers, and client/server applications differently.

How the CONNECT Role Change Affects General Users

You should be aware of how the `CONNECT` role affects general users.

The new `CONNECT` role supplies only the `CREATE SESSION` privilege. Users who connect to the database to use an application are not affected, because the `CONNECT` role still has the `CREATE SESSION` privilege.

However, appropriate privileges will not be present for a certain set of users if they are provisioned solely with the `CONNECT` role. These are users who create tables, views, sequences, synonyms, clusters, or database links, or use the `ALTER SESSION` command. The privileges they need are no longer provided with the `CONNECT` role. To authorize the additional privileges needed, the database administrator must create and apply additional roles for the appropriate privileges, or grant them directly to the users who need them.

Note that the `ALTER SESSION` privilege is required for setting events. Few database users should require the `ALTER SESSION` privilege.

The `ALTER SESSION` privilege is *not* required for other alter session commands.

How the CONNECT Role Change Affects Application Developers

You should be aware of how the `CONNECT` role affects application developers.

Application developers provisioned solely with the `CONNECT` role do not have appropriate privileges to create tables, views, sequences, synonyms, clusters, or database links, nor to use the `ALTER SESSION` statement.

You must either create and apply additional roles for the appropriate privileges, or grant them directly to the application developers who need them.

How the CONNECT Role Change Affects Client Server Applications

You should be aware of how the `CONNECT` role affects client server applications.

Most client/server applications that use dedicated user accounts will not be affected by this change.

However, applications that create private synonyms or temporary tables using dynamic SQL in the user schema during account provisioning or run-time operations will be affected. They will require additional roles or grants to acquire the system privileges appropriate to their activities.

Approaches to Addressing the CONNECT Role Change

Oracle recommends three approaches to address the impact of the `CONNECT` role change.

Creating a New Database Role

The privileges removed from the `CONNECT` role can be managed by creating a new database role.

1. Connect to the upgraded Oracle database and create a new database role.

   The following example uses a role called `my_app_developer`.

   ```
   CREATE ROLE my_app_developer;
   GRANT CREATE TABLE, CREATE VIEW, CREATE SEQUENCE, CREATE SYNONYM, CREATE
   CLUSTER, CREATE DATABASE LINK, ALTER SESSION TO my_app_developer;
   ```

2. Determine which users or database roles have the `CONNECT` role, and grant the new role to these users or roles.

   ```
   SELECT USER$.NAME, ADMIN_OPTION, DEFAULT_ROLE
    FROM USER$, SYSAUTH$, DBA_ROLE_PRIVS
    WHERE PRIVILEGE# =
    (SELECT USER# FROM USER$ WHERE NAME = 'CONNECT')
    AND USER$.USER# = GRANTEE#
    AND GRANTEE = USER$.NAME
    AND GRANTED_ROLE = 'CONNECT';

   NAME                           ADMIN_OPTI DEF
   ------------------------------ ---------- ---
   R1                             YES        YES
   R2                             NO         YES

   GRANT my_app_developer TO R1 WITH ADMIN OPTION;
   GRANT my_app_developer TO R2;
   ```

3. Determine the privileges that users require by creating a privilege analysis policy.

   The information that you gather can then be analyzed and used to create additional database roles with finer granularity. Privileges that are not used can then be revoked for specific users.

   For example:

   ```
   BEGIN
    DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE(
     name          => 'my_app_dev_role_pol',
     description   => 'Captures my_app_developer role use',
     type          => DBMS_PRIVILEGE_CAPTURE.G_ROLE,
     roles         => role_name_list('my_app_developer');
   END;
   /
   EXEC DBMS_PRIVILEGE_CAPTURE.ENABLE_CAPTURE ('my_app_dev_role_pol');
   ```

4. After a period of time, isable the privilege analysis policy and then generate a report.

```
EXEC DBMS_PRIVILEGE_CAPTURE.DISABLE_CAPTURE ('my_app_dev_role_pol');

EXEC DBMS_PRIVILEGE_CAPTURE.GENERATE_RESULT ('my_app_dev_role_pol');
```

5. After you generate the report, query the privilege analysis data dictionary views.

   For example:

   ```
   SELECT USERNAME, SYS_PRIV, OBJECT_OWNER, OBJECT_NAME FROM DBA_USED_PRIVS;
   ```

See Also:

Oracle Database Vault Administrator's Guide for more information about privilege analysis

Restoring the CONNECT Privilege

The `rstrconn.sql` script restores the `CONNECT` privileges.

After a database upgrade or new database creation, you can use this script to grant the privileges that were removed from the `CONNECT` role in Oracle Database 10*g* release 2 (10.2). If you use this approach, then you should revoke privileges that are not used from users who do not need them.

To restore the `CONNECT` privilege:

1. Run the `rstrconn.sql` script, which is in the `$ORACLE_HOME/rdbms/admin` directory.

   ```
   @$ORACLE_HOME/rdbm_admin/rstrconn.sql
   ```

2. Monitor the privileges that are used.

   For example:

   ```
   CREATE AUDIT POLICY connect_priv_pol
     PRIVILEGES AUDIT CREATE TABLE, CREATE SEQUENCE, CREATE SYNONYM, CREATE
   DATABASE LINK, CREATE CLUSTER, CREATE VIEW, ALTER SESSION;

   AUDIT POLICY connect_priv_pol BY psmith;
   ```

3. Periodically, monitor database privilege usage.

   For example:

   ```
   SELECT USERID, NAME FROM AUD$, SYSTEM_PRIVILEGE_MAP WHERE - PRIV$USED =
   PRIVILEGE;

   USERID                         NAME
   ------------------------------ ----------------
   ACME                           CREATE TABLE
   ACME                           CREATE SEQUENCE
   ACME                           CREATE TABLE
   ACME                           ALTER SESSION
   APPS                           CREATE TABLE
   APPS                           CREATE TABLE
   APPS                           CREATE TABLE
   APPS                           CREATE TABLE
   8 rows selected.
   ```

Data Dictionary View to Show CONNECT Grantees

The DBA_CONNECT_ROLE_GRANTEES data dictionary view enables administrators who continue using the old CONNECT role to see which users have that role.

Table A-1 shows the columns in the DBA_CONNECT_ROLE_GRANTEES view.

Table A-1 Columns and Contents for DBA_CONNECT_ROLE_GRANTEES

Column	Datatype	NULL	Description
GRANTEE	VARCHAR2(128)	NULL	User granted the CONNECT role
PATH_OF_CONNECT_ROLE_GRANT	VARCHAR2(4000	NULL	Role (or nested roles) by which the user is granted CONNECT
ADMIN_OPT	VARCHAR2(3)	NULL	YES if user has the ADMIN option on CONNECT; otherwise, NO

Least Privilege Analysis Studies

Oracle partners and application providers should conduct a least privilege analysis so that they can deliver more secure products to their Oracle customers.

The principle of least privilege mitigates risk by limiting privileges to the minimum set required to perform a given function.

For each class of users that the analysis shows need the same set of privileges, create a role with only those privileges. Remove all other privileges from those users, and assign that role to those users. As needs change, you can grant additional privileges, either directly or through these new roles, or create new roles to meet new needs. This approach helps to ensure that inappropriate privileges have been limited, thereby reducing the risk of inadvertent or malicious harm.

You can create policies that analyze the use of privileges by database users. The policies capture this information and make it available in data dictionary views. Based on these reports, you can determine who should have access to your data.

See Also:

Oracle Database Vault Administrator’s Guide for more information about privilege analysis

B
Data Encryption and Integrity Parameters

The `sqlnet.ora` file has data encryption and integrity parameters.

About Using sqlnet.ora for Data Encryption and Integrity

You can use the default parameter settings as a guideline for configuring data encryption and integrity.

This `sqlnet.ora` file is generated when you perform the network configuration described in Configuring Oracle Database Network Encryption and Data Integrity and Configuring Secure Sockets Layer Authentication. Also provided are encryption and data integrity parameters.

Sample sqlnet.ora File

The sample `sqlnet.ora` configuration file is based on a set of clients with similar characteristics and a set of servers with similar characteristics.

The file includes examples of Oracle Database encryption and data integrity parameters.

By default, the `sqlnet.ora` file is located in the *`ORACLE_HOME`*`/network/admin` directory or in the location set by the `TNS_ADMIN` environment variable. Ensure that you have properly set the `TNS_ADMIN` variable to point to the correct `sqlnet.ora` file. See *SQL*Plus User's Guide and Reference* for more information and examples of setting the `TNS_ADMIN` variable.

Trace File Setup

```
#Trace file setup
trace_level_server=16
trace_level_client=16
trace_directory_server=/orant/network/trace
trace_directory_client=/orant/network/trace
trace_file_client=cli
trace_file_server=srv
trace_unique_client=true
```

Oracle Database Network Encryption

```
sqlnet.encryption_server=accepted
sqlnet.encryption_client=requested
sqlnet.encryption_types_server=(RC4_40)
sqlnet.encryption_types_client=(RC4_40)
```

Oracle Database Network Data Integrity

```
#ASO Checksum
sqlnet.crypto_checksum_server=requested
sqlnet.crypto_checksum_client=requested
sqlnet.crypto_checksum_types_server = (SHA256)
sqlnet.crypto_checksum_types_client = (SHA256)
```

Secure Sockets Layer

```
#SSL
WALLET_LOCATION = (SOURCE=
                   (METHOD = FILE)
                      (METHOD_DATA =
                         DIRECTORY=/wallet)

SSL_CIPHER_SUITES=(SSL_DH_anon_WITH_RC4_128_MD5)
SSL_VERSION= 3
SSL_CLIENT_AUTHENTICATION=FALSE
```

Common

```
#Common
automatic_ipc = off
sqlnet.authentication_services = (beq)
names.directory_path = (TNSNAMES)
```

Kerberos

```
#Kerberos
sqlnet.authentication_services = (beq, kerberos5)
sqlnet.authentication_kerberos5_service = oracle
sqlnet.kerberos5_conf= /krb5/krb.conf
sqlnet.kerberos5_keytab= /krb5/v5srvtab
sqlnet.kerberos5_realms= /krb5/krb.realm
sqlnet.kerberos5_cc_name = /krb5/krb5.cc
sqlnet.kerberos5_clockskew=900
sqlnet.kerberos5_conf_mit=false
```

RADIUS

```
#Radius
sqlnet.authentication_services = (beq, RADIUS )
sqlnet.radius_authentication_timeout = (10)
sqlnet.radius_authentication_retries = (2)
sqlnet.radius_authentication_port = (1645)
sqlnet.radius_send_accounting = OFF
sqlnet.radius_secret = /orant/network/admin/radius.key
sqlnet.radius_authentication = radius.us.example.com
sqlnet.radius_challenge_response = OFF
sqlnet.radius_challenge_keyword = challenge
sqlnet.radius_challenge_interface =
oracle/net/radius/DefaultRadiusInterface
sqlnet.radius_classpath = /jre1.1/
```

Data Encryption and Integrity Parameters

Oracle provides data and integrity parameters that you can set in the `sqlnet.ora` file.

About the Data Encryption and Integrity Parameters

The data encryption and integrity parameters control the type of encryption algorithm you are using.

If you do not specify any values for Server Encryption, Client Encryption, Server Checksum, or Client Checksum, the corresponding configuration parameters do not appear in the `sqlnet.ora` file. However, the defaults are `ACCEPTED`.

For both data encryption and integrity algorithms, the server selects the first algorithm listed in its `sqlnet.ora` file that matches an algorithm listed in the client `sqlnet.ora` file, or in the client installed list if the client lists no algorithms in its `sqlnet.ora` file. If there are no entries in the server `sqlnet.ora` file, the server sequentially searches its installed list to match an item on the client side—either in the client `sqlnet.ora` file or in the client installed list. *If no match can be made and one side of the connection REQUIRED the algorithm type (data encryption or integrity), then the connection fails.* Otherwise, the connection succeeds with the algorithm type `inactive`.

Data encryption and integrity algorithms are selected independently of each other. Encryption can be activated without integrity, and integrity can be activated without encryption, as shown by Table B-1:

Table B-1 Algorithm Type Selection

Encryption Selected?	Integrity Selected?
Yes	No
Yes	Yes
No	Yes
No	No

SQLNET.ENCRYPTION_SERVER

The `SQLNET.ENCRYPTION_SERVER` parameter specifies the encryption behavior when a client or a server acting as a client connects to this server.

The behavior of the server partially depends on the `SQLNET.ENCRYPTION_CLIENT` setting at the other end of the connection.

Table B-2 describes the `SQLNET.ENCRYPTION_SERVER` parameter attributes.

Table B-2 SQLNET.ENCRYPTION_SERVER Parameter Attributes

Attribute	Description
Syntax	`SQLNET.ENCRYPTION_SERVER = valid_value`
Valid Values	`ACCEPTED`, `REJECTED`, `REQUESTED`, `REQUIRED`
Default Setting	`ACCEPTED`

See Also:

Oracle Database Net Services Reference for more information about the `SQLNET.ENCRYPTION_SERVER` parameter

SQLNET.ENCRYPTION_CLIENT

The `SQLNET.ENCRYPTION_CLIENT` parameter specifies the encryption behavior when this client or server acting as a client connects to a server.

The behavior of the client partially depends on the value set for `SQLNET.ENCRYPTION_SERVER` at the other end of the connection.

Table B-3 describes the `SQLNET.ENCRYPTION_CLIENT` parameter attributes.

Table B-3 SQLNET.ENCRYPTION_CLIENT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.ENCRYPTION_CLIENT = `*`valid_value`*
Valid Values	`ACCEPTED`, `REJECTED`, `REQUESTED`, `REQUIRED`
Default Setting	`ACCEPTED`

See Also:

Oracle Database Net Services Reference for more information about the `SQLNET.ENCRYPTION_CLIENT` parameter

SQLNET.CRYPTO_CHECKSUM_SERVER

The `SQLNET.CRYPTO_CHECKSUM_SERVER` parameter specifies the data integrity behavior when a client or another server acting as a client connects to this server.

The behavior partially depends on the `SQLNET.CRYPTO_CHECKSUM_CLIENT` setting at the other end of the connection.

Table B-4 describes the `SQLNET.CRYPTO_CHECKSUM_SERVER` parameter attributes.

Table B-4 SQLNET.CRYPTO_CHECKSUM_SERVER Parameter Attributes

Attribute	Description
Syntax	`SQLNET.CRYPTO_CHECKSUM_SERVER = `*`valid_value`*
Valid Values	`ACCEPTED`, `REJECTED`, `REQUESTED`, `REQUIRED`
Default Setting	`ACCEPTED`

See Also:

Oracle Database Net Services Reference for more information about the `SQLNET.CRYPTO_CHECKSUM_SERVER` parameter

SQLNET.CRYPTO_CHECKSUM_CLIENT

The `SQLNET.CRYPTO_CHECKSUM_CLIENT` parameter specifies the desired data integrity behavior when this client or server acting as a client connects to a server.

The behavior partially depends on the `SQLNET.CRYPTO_CHECKSUM_SERVER` setting at the other end of the connection.

Table B-5 describes the `SQLNET.CRYPTO_CHECKSUM_CLIENT` parameter attributes.

Table B-5 SQLNET.CRYPTO_CHECKSUM_CLIENT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.CRYPTO_CHECKSUM_CLIENT = ` *`valid_value`*
Valid Values	`ACCEPTED`, `REJECTED`, `REQUESTED`, `REQUIRED`
Default Setting	`ACCEPTED`

SQLNET.ENCRYPTION_TYPES_SERVER

The `SQLNET.ENCRYPTION_TYPES_SERVER` parameter specifies encryption algorithms this server uses in the order of the intended use.

This list is used to negotiate a mutually acceptable algorithm with the client end of the connection. Each algorithm is checked against the list of available client algorithm types until a match is found. If an algorithm that is not installed is specified on this side, the connection terminates with the error message `ORA-12650: No common encryption or data integrity algorithm`.

Table B-6 describes the `SQLNET.ENCRYPTION_TYPES_SERVER` parameter attributes.

Table B-6 SQLNET.ENCRYPTION_TYPES_SERVER Parameter Attributes

Attribute	Description
Syntax	`SQLNET.ENCRYPTION_TYPES_SERVER = (`*`valid_encryption_algorithm`* `[,`*`valid_encryption_algorithm`*`])`
Valid Values	• `AES256`: AES (256-bit key size) • `AES192`: AES (192-bit key size) • `AES128`: AES (128-bit key size)
Default Setting	If no algorithms are defined in the local `sqlnet.ora` file, then all installed algorithms are used in a negotiation in the preceding sequence.

Usage Notes	You can specify multiple encryption algorithms. It can be either a single value or a list of algorithm names. For example, either of the following encryption parameters is acceptable:
	`SQLNET.ENCRYPTION_TYPES_SERVER=(AES256)`
	`SQLNET.ENCRYPTION_TYPES_SERVER=(AES256,AES192,AES128)`

See Also:

Oracle Database Net Services Reference for more information about the `SQLNET.ENCRYPTION_TYPES_SERVER` parameter

SQLNET.ENCRYPTION_TYPES_CLIENT

The `SQLNET.ENCRYPTION_TYPES_CLIENT` parameter specifies encryption algorithms this client or the server acting as a client uses.

This list is used to negotiate a mutually acceptable algorithm with the other end of the connection. If an algorithm that is not installed is specified on this side, the connection terminates with the `ORA-12650: No common encryption or data integrity algorithm error` message.

Table B-7 describes the `SQLNET.ENCRYPTION_TYPES_CLIENT` parameter attributes.

Table B-7 SQLNET.ENCRYPTION_TYPES_CLIENT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.ENCRYPTION_TYPES_CLIENT = (`*`valid_encryption_algorithm`*` [,`*`valid_encryption_algorithm`*`])`
Valid Values	• `AES256`: AES (256-bit key size). • `AES192`: AES (192-bit key size). • `AES128`: AES (128-bit key size).
Default Setting	If no algorithms are defined in the local `sqlnet.ora` file, all installed algorithms are used in a negotiation.
Usage Notes	You can specify multiple encryption algorithms by separating each one with a comma. For example: `SQLNET.ENCRYPTION_TYPES_CLIENT=(AES256,AES192,AES128)`

See Also:

Oracle Database Net Services Reference for more information about the `SQLNET.ENCRYPTION_TYPES_CLIENT` parameter

SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER

The `SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER` parameter specifies data integrity algorithms that this server or client to another server uses, in order of intended use.

This list is used to negotiate a mutually acceptable algorithm with the other end of the connection. Each algorithm is checked against the list of available client algorithm types until a match is found. If an algorithm is specified that is not installed on this side, the connection terminates with the `ORA-12650: No common encryption or data integrity algorithm error` error message.

Table B-8 describes the `SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER` parameter attributes.

Table B-8 SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER Parameter Attributes

Attribute	Description
Syntax	`SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER = (`*`valid_crypto_checksum_algorithm`*` [,`*`valid_crypto_checksum_algorithm`*`])`
Valid Values	• `SHA512`: SHA-2, produces a 512-bit hash. • `SHA384`: SHA-2, produces a 384-bit hash. • `SHA256`: SHA-2, produces a 256-bit hash. This is the default value. • `SHA1`: Secure Hash Algorithm • `MD5`: Message Digest 5
Default Setting	If no algorithms are defined in the local `sqlnet.ora` file, all installed algorithms are used in a negotiation starting with `SHA256`.

See Also:

Oracle Database Net Services Reference for more information about the `SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER` parameter

SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT

The `SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT` parameter specifies a list of data integrity algorithms that this client or server acting as a client uses.

This list is used to negotiate a mutually acceptable algorithm with the other end of the connection. If an algorithm that is not installed on this side is specified, the connection terminates with the `ORA-12650: No common encryption or data integrity algorithm error` error message.

Table B-9 describes the `SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT` parameter attributes.

Table B-9 SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT = (`*`valid_crypto_checksum_algorithm`*` [,`*`valid_crypto_checksum_algorithm`*`])`
Valid Values	• `SHA512`: SHA-2, produces a 512-bit hash. • `SHA384`: SHA-2, produces a 384-bit hash. • `SHA256`: SHA-2, produces a 256-bit hash. This is the default value. • `SHA1`: Secure Hash Algorithm • `MD5`: Message Digest 5

Table B-9 (Cont.) SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT Parameter Attributes

Attribute	Description
Default Setting	If no algorithms are defined in the local `sqlnet.ora` file, all installed algorithms are used in a negotiation starting with `SHA256.`

See Also:

Oracle Database Net Services Reference for more information about the `SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT` parameter

C

Kerberos, SSL, and RADIUS Authentication Parameters

The `sqlnet.ora` and the database initialization files provide Kerberos, RADIUS, or SSL authentication parameters.

Parameters for Clients and Servers Using Kerberos Authentication

Oracle Database provides client and server parameters for using Kerberos authentication.

Table C-1 lists parameters to insert into the configuration files for clients and servers using Kerberos.

Table C-1 Kerberos Authentication Parameters

File Name	Configuration Parameters
`sqlnet.ora`	`SQLNET.AUTHENTICATION_SERVICES=(KERBEROS5) SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=oracle SQLNET.KERBEROS5_CC_NAME=/usr/tmp/DCE-CC SQLNET.KERBEROS5_CLOCKSKEW=1200 SQLNET.KERBEROS5_CONF=/krb5/krb.conf SQLNET.KERBEROS5_CONF_MIT=(FALSE) SQLNET.KERBEROS5_REALMS=/krb5/krb.realms SQLNET.KERBEROS5_KEYTAB=/krb5/v5srvtabSQLNET.FALLBACK_AUTHENTICATION=FALSE`
`initialization parameter file`	`OS_AUTHENT_PREFIX=""`

Parameters for Clients and Servers Using Secure Sockets Layer

Oracle provides parameters to control Secure Sockets Layer authentication.

Ways to Configure a Parameter for Secure Sockets Layer

There are two ways to configure a parameter for Secure Sockets Layer (SSL).

- **Static:** The name of the parameter that exists in the `sqlnet.ora` file. Parameters like `SSL_CIPHER_SUITES` and `SSL_VERSION` can also be configured using the `listener.ora` file.
- **Dynamic:** The name of the parameter used in the security subsection of the Oracle Net address.

Secure Sockets Layer Authentication Parameters for Clients and Servers

Oracle provides both static and dynamic Secure Sockets Layer (SSL) authentication parameters.

Table C-2 describes the static and dynamic parameters for configuring SSL on the server.

Table C-2 SSL Authentication Parameters for Clients and Servers

Attribute	Description
Parameter Name (static)	`SQLNET.AUTHENTICATION_SERVICES`
Parameter Name (dynamic)	`AUTHENTICATION`
Parameter Type	String `LIST`
Parameter Class	Static
Permitted Values	Add TCPS to the list of available authentication services.
Default Value	No default value.
Description	To control which authentication services a user wants to use. **Note:** The dynamic version supports only the setting of one type.
Existing/New Parameter	Existing
Syntax (static)	`SQLNET.AUTHENTICATION_SERVICES = (TCPS, *selected_method_1*, *selected_method_2*)`
Example (static)	`SQLNET.AUTHENTICATION_SERVICES = (TCPS, radius)`
Syntax (dynamic)	`AUTHENTICATION = *string*`
Example (dynamic)	`AUTHENTICATION = (TCPS)`

Cipher Suite Parameters for Secure Sockets Layer

You can configure cipher suite parameters for Secure Sockets Layer (SSL).

Table C-3 describes the static and dynamic parameters for configuring cipher suites.

Table C-3 Cipher Suite Parameters for Secure Sockets Layer

Attribute	Description
Parameter Name (static)	`SSL_CIPHER_SUITES`
Parameter Name (dynamic)	`SSL_CIPHER_SUITES`
Parameter Type	String `LIST`
Parameter Class	Static
Permitted Values	Any known SSL cipher suite
Default Value	No default

Table C-3 (Cont.) Cipher Suite Parameters for Secure Sockets Layer

Attribute	Description
Description	Controls the combination of encryption and data integrity used by SSL.
Existing/New Parameter	Existing
Syntax (static)	`SSL_CIPHER_SUITES=(`*`SSL_cipher_suite1`*`[,` *`SSL_cipher_suite2`*`, ...` *`SSL_cipher_suiteN`*`])`
Example (static)	`SSL_CIPHER_SUITES=(SSL_DH_DSS_WITH_DES_CBC_SHA)`
Syntax (dynamic)	`SSL_CIPHER_SUITES=(`*`SSL_cipher_suite1`* `[,` *`SSL_cipher_suite2`*`, ...`*`SSL_cipher_suiteN`*`])`
Example (dynamic)	`SSL_CIPHER_SUITES=(SSL_DH_DSS_WITH_DES_CBC_SHA)`

Supported Secure Sockets Layer Cipher Suites

Oracle Database supports a large number of cipher suites for Secure Sockets Layer (SSL).

The cipher suites are as follows:

- `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
- `SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA`
- `SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256`
- `SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`
- `SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`
- `SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
- `SSL_RSA_WITH_AES_128_CBC_SHA256`
- `SSL_RSA_WITH_AES_128_GCM_SHA256`
- `SSL_RSA_WITH_AES_128_CBC_SHA`
- `SSL_RSA_WITH_AES_256_CBC_SHA`
- `SSL_RSA_WITH_AES_256_CBC_SHA256`

Secure Sockets Layer Version Parameters

You can set a range of Secure Sockets Layer (SSL) parameters to configure the version of SSL to use.

Table C-4 describes the `SSL_VERSION` static and dynamic parameters for configuring the version of SSL to be used.

Table C-4 Secure Sockets Layer Version Parameters

Attribute	Description
Parameter Name (static)	`SSL_VERSION`

Table C-4 (Cont.) Secure Sockets Layer Version Parameters

Attribute	Description
Parameter Name (dynamic)	`SSL_VERSION`
Parameter Type	string
Parameter Class	Static
Permitted Values	Any version which is valid to SSL. Values are as follows: `undetermined | 1.0 | 1.1 | 1.2` If you want to specify one version or another version, then use "or". The following values are permitted: • `1.0` • `1.1` • `1.2` • `1.0 or 1.1` • `1.0 or 1.2` • `1.0 or 1.1 or 1.2`
Default Value	`1.2`
Description	To force the version of the SSL connection.
Existing/New Parameter	New
Syntax (static)	`SSL_VERSION=`*`version`*
Example (static)	`SSL_VERSION=1.1`
Syntax (dynamic)	`SSL_VERSION=`*`version`*
Example (dynamic)	`SSL_VERSION=1.1 or 1.2`

Note:

The `ADD_SSLv3_IMPLICITLY` initialization parameter has no effect on the `SSL_VERSION` parameter.

Secure Sockets Layer Client Authentication Parameters

You can configure static and dynamic parameters for Secure Sockes Layer (SSL) on the client.

Table C-5 describes the `SSL_CLIENT_AUTHENTICATION` parameters.

Table C-5 Secure Sockets Layer Client Authentication Parameters

Attribute	Description
Parameter Name (static)	`SSL_CLIENT_AUTHENTICATION`
Parameter Name (dynamic)	`SSL_CLIENT_AUTHENTICATION`
Parameter Type	Boolean
Parameter Class	Static
Permitted Values	`TRUE` or `FALSE`

Table C-5 (Cont.) Secure Sockets Layer Client Authentication Parameters

Attribute	Description
Default Value	`TRUE`
Description	To control whether a client, in addition to the server, is authenticated using SSL.
Existing/New Parameter	New
Syntax (static)	`SSL_CLIENT_AUTHENTICATION={`*`TRUE`* | *`FALSE`*`}`
Example (static)	`SSL_CLIENT_AUTHENTICATION=FALSE`
Syntax (dynamic)	`SSL_CLIENT_AUTHENTICATION={`*`TRUE`* | *`FALSE`*`}`
Example (dynamic)	`SSL_CLIENT_AUTHENTICATION=FALSE`

Secure Sockets Layer X.509 Server Match Parameters

The `SSL_SERVER_DN_MATCH` and `SSL_SERVER_CERT_DN` parameters validate the identity of the server to which a client connects.

SSL_SERVER_DN_MATCH

The `SSL_SERVER_DN_MATCH` parameter forces the server’s distinguished name (DN) to match the name of the servivce.

Table C-6 describes the `SSL_SERVER_DN_MATCH` parameter.

Table C-6 SSL_SERVER_DN_MATCH Parameter

Attribute	Description
Parameter Name	`SSL_SERVER_DN_MATCH`
Where stored	`sqlnet.ora`
Purpose	Use this parameter to force the server's distinguished name (DN) to match its service name. If you force the match verifications, SSL ensures that the certificate is from the server. If you choose not to enforce the match verification, SSL performs the check but permits the connection, regardless of whether there is a match. *Not forcing the match lets the server potentially fake its identity.*
Values	`yes|on|true`. Specify to enforce a match. If the DN matches the service name, the connection succeeds; otherwise, the connection fails. `no|off|false`. Specify to not enforce a match. If the DN does not match the service name, the connection is successful, but an error is logged to the `sqlnet.log` file.
Default	Oracle8*i*, or later:.FALSE. SSL client (always) checks server DN. If it does not match the service name, the connection succeeds but an error is logged to `sqlnet.log` file.
Usage Notes	Additionally configure the `tnsnames.ora` parameter `SSL_SERVER_CERT_DN` to enable server DN matching.

SSL_SERVER_CERT_DN

The SSL_SERVER_CERT_DN specifies the distinguished name (DN) of a server.

Table C-7 describes the `SSL_SERVER_CERT_DN` parameter.

Table C-7 SSL_SERVER_CERT_DN Parameter

Attribute	Description
Parameter Name	`SSL_SERVER_CERT_DN`
Where stored	`tnsnames.ora`. It can be stored on the client, for every server it connects to, or it can be stored in the LDAP directory, for every server it connects to, updated centrally.
Purpose	This parameter specifies the distinguished name (DN) of the server. The client uses this information to obtain the list of DNs it expects for each of the servers to force the server's DN to match its service name.
Values	Set equal to distinguished name (DN) of the server.
Default	N/A
Usage Notes	Additionally configure the `sqlnet.ora` parameter `SSL_SERVER_DN_MATCH` to enable server DN matching.
Example	`dbalias=(description=address_list=(address=(protocol=tcps)(host=hostname)(port=portnum)))(connect_data=(sid=Finance))(security=(SSL_SERVER_CERT_DN="CN=Finance,CN=OracleContext,C=US,O=Acme"))`

Oracle Wallet Location

You must specify wallet location parameters for applications that must access an Oracle wallet for loading the security credentials into the process space.

Table C-8 lists the configuration files in which you must specify the wallet locations.

- `sqlnet.ora`
- `listener.ora`

Table C-8 Wallet Location Parameters

Static Configuration	Dynamic Configuration

```
WALLET_LOCATION =
(SOURCE=
  (METHOD=File)
  (METHOD_DATA=
     (DIRECTORY=your_wallet_dir)
      )

)
```
	`MY_WALLET_DIRECTORY = ` *`your_wallet_dir`*

The default wallet location is the `ORACLE_HOME` directory.

Parameters for Clients and Servers Using RADIUS Authentication

Oracle provides parameters for RADIUS authentication.

sqlnet.ora File Parameters

You can include RADIUS-specific parameters in the `sqlnet.ora` file.

SQLNET.AUTHENTICATION_SERVICES

The `SQLNET.AUTHENTICATION_SERVICES` parameter configures the client or the server to use the RADIUS adapter.

Table C-9 describes the `SQLNET.AUTHENTICATION_SERVICES` parameter attributes.

Table C-9 SQLNET.AUTHENTICATION_SERVICES Parameter Attributes

Attribute	Description
Syntax	`SQLNET.AUTHENTICATION_SERVICES=(radius)`
Default setting	None

SQLNET.RADIUS_ALTERNATE

The `SQLNET.RADIUS_ALTERNATE` parameter sets the location of an alternate RADIUS server to be used if the primary server is unavailable for fault tolerance.

Table C-10 describes the `SQLNET.RADIUS_ALTERNATE` parameter attributes.

Table C-10 SQLNET.RADIUS_ALTERNATE Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_ALTERNATE=`*`alternate_RADIUS_server_hostname_or_IP_address`*
Default setting	`off`

SQLNET.RADIUS_ALTERNATE_PORT

The `SQLNET.RADIUS_ALTERNATE_PORT` parameter sets the listening port for the alternate RADIUS server.

Table C-11 describes the `SQLNET.RADIUS_ALTERNATE_PORT` parameter attributes.

Table C-11 SQLNET.RADIUS_ALTERNATE_PORT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_ALTERNATE_PORT=`*`alternate_RADIUS_server_listening_port_number`*
Default setting	`1645`

SQLNET.RADIUS_ALTERNATE_TIMEOUT

The `SQLNET.RADIUS_ALTERNATE_TIMEOUT` parameter sets the time for an alternate RADIUS server to wait for a response.

Table C-12 describes the `SQLNET.RADIUS_ALTERNATE_TIMEOUT` parameter attributes.

Table C-12 SQLNET.RADIUS_ALTERNATE_TIMEOUT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_ALTERNATE_TIMEOUT=`*`time_in_seconds`*
Default setting	5

SQLNET.RADIUS_ALTERNATE_RETRIES

The `SQLNET.RADIUS_ALTERNATE_RETRIES` parameter sets the number of times that the alternate RADIUS server resends messages.

Table C-13 describes the `SQLNET.RADIUS_ALTERNATE_RETRIES` parameter attributes.

Table C-13 SQLNET.RADIUS_ALTERNATE_RETRIES Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_ALTERNATE_RETRIES=`*`n_times_to_resend`*
Default setting	3

SQLNET.RADIUS_AUTHENTICATION

The `SQLNET.RADIUS_AUTHENTICATION` parameter sets the location of the primary RADIUS server, either host name or dotted decimal format.

If the RADIUS server is on a different computer from the Oracle server, you must specify either the host name or the IP address of that computer.

Table C-14 describes the `SQLNET.RADIUS_AUTHENTICATION` parameter attributes.

Table C-14 SQLNET.RADIUS_AUTHENTICATION Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_AUTHENTICATION=`*`RADIUS_server_IP_address`*
Default setting	`localhost`

SQLNET.RADIUS_AUTHENTICATION_INTERFACE

The `SQLNET.RADIUS_AUTHENTICATION_INTERFACE` parameter sets the name of the Java class that contains the GUI when RADIUS is in challenge-response (asynchronous) mode.

Table C-15 describes the `SQLNET.RADIUS_AUTHENTICATION_INTERFACE` parameter attributes.

Table C-15 SQLNET.RADIUS_AUTHENTICATION_INTERFACE Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_AUTHENTICATION_INTERFACE=`*`Java_class_name`*
Default setting	`DefaultRadiusInterface (oracle/net/radius/ DefaultRadiusInterface)`

SQLNET.RADIUS_AUTHENTICATION_PORT

The `SQLNET.RADIUS_AUTHENTICATION_PORT` parameter sets the listening port of the primary RADIUS server.

Table C-16 describes the `SQLNET.RADIUS_AUTHENTICATION_PORT` parameter attributes.

Table C-16 SQLNET.RADIUS_AUTHENTICATION_PORT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_AUTHENTICATION_PORT=`*`port_number`*
Default setting	`1645`

SQLNET.RADIUS_AUTHENTICATION_TIMEOUT

The `SQLNET.RADIUS_AUTHENTICATION_TIMEOUT` parameter sets the time to wait for response.

Table C-17 describes the `SQLNET.RADIUS_AUTHENTICATION_TIMEOUT` parameter attributes.

Table C-17 SQLNET.RADIUS_AUTHENTICATION_TIMEOUT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_AUTHENTICATION_TIMEOUT=`*`time_in_seconds`*
Default setting	5

SQLNET.RADIUS_AUTHENTICATION_RETRIES

The `SQLNET.RADIUS_AUTHENTICATION_RETRIES` parameter sets the number of times to resend authentication information.

Table C-18 describes the `SQLNET.RADIUS_AUTHENTICATION_RETRIES` parameter attributes.

Table C-18 SQLNET.RADIUS_AUTHENTICATION_RETRIES Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_AUTHENTICATION_RETRIES=`*`n_times_to_resend`*
Default setting	3

SQLNET.RADIUS_CHALLENGE_RESPONSE

The `SQLNET.RADIUS_CHALLENGE_RESPONSE` parameter turns on or turns off the challenge-response or asynchronous mode support.

Table C-19 describes the `SQLNET.RADIUS_CHALLENGE_RESPONSE` parameter attributes.

Table C-19 SQLNET.RADIUS_CHALLENGE_RESPONSE Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_CHALLENGE_RESPONSE=`*`on`*
Default setting	`off`

SQLNET.RADIUS_CHALLENGE_KEYWORD

The `SQLNET.RADIUS_CHALLENGE_KEYWORD` parameter sets the keyword to request a challenge from the RADIUS server.

The user types no password on the client.

Table C-20 describes the `SQLNET.RADIUS_CHALLENGE_KEYWORD` parameter attributes.

Table C-20 SQLNET.RADIUS_CHALLENGE_KEYWORD Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_CHALLENGE_KEYWORD=`*`keyword`*
Default setting	`challenge`

SQLNET.RADIUS_CLASSPATH

The `SQLNET.RADIUS_CLASSPATH` parameter sets the path for Java classes and the JDK Java libraries.

If you decide to use the challenge-response authentication mode, then RADIUS presents the user with a Java-based graphical interface requesting first a password, then additional information, for example, a dynamic password that the user obtains from a token card.

Add the `SQLNET.RADIUS_CLASSPATH` parameter in the `sqlnet.ora` file to set the path for the Java classes for that graphical interface, and to set the path to the JDK Java libraries.

Table C-21 describes the `SQLNET.RADIUS_CLASSPATH` parameter attributes.

Table C-21 SQLNET.RADIUS_CLASSPATH Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_CLASSPATH=`*`path_to_GUI_Java_classes`*
Default setting	`$ORACLE_HOME/jlib/netradius.jar:$ORACLE_HOME/JRE/lib/sparc/native_threads`

SQLNET.RADIUS_SECRET

The `SQLNET.RADIUS_SECRET` parameter specifies the file name and location of the RADIUS secret key.

Table C-22 describes the `SQLNET.RADIUS_SECRET` parameter attributes.

Table C-22 SQLNET.RADIUS_SECRET Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_SECRET=`*`path_to_RADIUS_secret_key`*
Default setting	`$ORACLE_HOME/network/security/radius.key`

SQLNET.RADIUS_SEND_ACCOUNTING

The `SQLNET.RADIUS_SEND_ACCOUNTING` parameter turns accounting on or off.

If you enable accounting, packets will be sent to the active RADIUS server at the listening port plus one. By default, packets are sent to port 1646. You need to turn this feature on only when your RADIUS server supports accounting and you want to keep track of the number of times the user is logging on to the system.

Table C-23 describes the `SQLNET.RADIUS_SEND_ACCOUNTING` parameter attributes.

Table C-23 SQLNET.RADIUS_SEND_ACCOUNTING Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_SEND_ACCOUNTING=`*`on`*
Default setting	`off`

Minimum RADIUS Parameters

At minimum, you should use the `SQLNET.AUTHENTICATION_SERVICES` and `SQLNET.RADIUS.AUTHENTICATION` parameters.

Use the following settings:

```
sqlnet.authentication_services = (radius)
sqlnet.radius.authentication   = IP-address-of-RADIUS-server
```

Initialization File Parameter for RADIUS

For RADIUS, you should set the `OS_AUTHENT_PREFIX` initialization parameter.

For example:

```
OS_AUTHENT_PREFIX=""
```

D

Integrating Authentication Devices Using RADIUS

The RADIUS challenge-response user interface further enhances authentication in a RADIUS configuration.

About the RADIUS Challenge-Response User Interface

You can use third-party authentication vendors to customize the RADIUS challenge-response user interface to fit a particular device.

You can set up any authentication device that supports the RADIUS standard to authenticate Oracle users. When your authentication device uses the challenge-response mode, a graphical interface prompts the end user first for a password and then for additional information (for example, a dynamic password that the user obtains from a token card). This interface is Java-based to provide optimal platform independence.

Third-party vendors of authentication devices must customize this graphical user interface to fit their particular device. For example, a smart card vendor customizes the Oracle client to issue the challenge to the smart card reader. Then, when the smart card receives a challenge, it responds by prompting the user for more information, such as a PIN.

Customizing the RADIUS Challenge-Response User Interface

You can customize `OracleRadiusInterface` interface by creating your own class.

1. Open the `sqlnet.ora` file.

   By default, the `sqlnet.ora` file is located in the `*ORACLE_HOME*/network/admin` directory or in the location set by the `TNS_ADMIN` environment variable. Ensure that you have properly set the `TNS_ADMIN` variable to point to the correct `sqlnet.ora` file.

2. Locate the `SQLNET.RADIUS_AUTHENTICATION_INTERFACE` parameter, and replace the name of the class listed there (`DefaultRadiusInterface`), with the name of the new class that you have created.

   When you make this change in the `sqlnet.ora` file, the class is loaded on the Oracle client in order to handle the authentication process.

3. Save and exit the `sqlnet.ora` file

The third party must implement the `OracleRadiusInterface` interface, which is located in the `ORACLE.NET.RADIUS` package.

See Also:

*SQL*Plus User's Guide and Reference* for more information and examples of setting the `TNS_ADMIN` variable

Example: Using the OracleRadiusInterface Interface

You can use the `OracleRadiusInterface` interface to retrieve a user name and password.

Example D-1 shows how to use the `OracleRadiusInterface` interface.

Example D-1 Using the OracleRadiusInterface Interface

```
public interface OracleRadiusInterface {
  public void radiusRequest();
  public void radiusChallenge(String challenge);
  public String getUserName();
  public String getPassword();
}
```

In this specification:

- `radiusRequest` prompts the end user for a user name and password, which will later be retrieved through `getUserName` and `getPassword`.
- `getUserName` extracts the user name the user enters. If this method returns an empty string, it is assumed that the user wants to cancel the operation. The user then receives a message indicating that the authentication attempt failed.
- `getPassword` extracts the password the user enters. If `getUserName` returns a valid string, but `getPassword` returns an empty string, the challenge keyword is replaced as the password by the database. If the user enters a valid password, a challenge may or may not be returned by the RADIUS server.
- `radiusChallenge` presents a request sent from the RADIUS server for the user to respond to the server's challenge.
- `getResponse` extracts the response the user enters. If this method returns a valid response, then that information populates the `User-Password` attribute in the new `Access-Request` packet. If an empty string is returned, the operation is aborted from both sides by returning the corresponding value.

E

Oracle Database FIPS 140-2 Settings

Oracle supports the Federal Information Processing Standard (FIPS) standard for 140-2.

About the Oracle Database FIPS 140-2 Settings

The Federal Information Processing Standard (FIPS) standard, 140-2, is a U.S. government standard that defines cryptographic module security requirements.

The FIPS 140-2 cryptographic libraries are designed to protect data at rest and in transit over the network.

Oracle Database uses these cryptographic libraries for Secure Sockets Layer (SSL), Transparent Data Encryption (TDE), and `DBMS_CRYPTO` PL/SQL package.

To verify the current status of the certification, you can find information at the Computer Security Resource Center (CSRC) Web site address from the National Institute of Standards and Technology:

http://csrc.nist.gov/groups/STM/cmvp/validation.html

You can find information specific to FIPS by searching for `Validated FIPS 140 Cryptographic Modules`. The security policy, which is available on this site upon successful certification, includes requirements for secure configuration of the host operating system.

Configuring FIPS 140-2 for Transparent Data Encryption and DBMS_CRYPTO

The `DBFIPS_140` initialization parameter configures FIPS mode.

1. To configure Transparent Data Encryption and the `DBMS_CRYPTO` PL/SQL package program units to run in FIPS mode, set the `DBFIPS_140` initialization parameter to `TRUE`.

   The effect of this parameter depends on the platform.

2. Restart the database.

Table E-1 describes how the `DBFIPS_140` parameter affects various platforms.

Table E-1 How the DBFIPS_140 Initialization Parameter Affects Platforms

Platform	Effect of Setting DBFIPS_140 to TRUE or FALSE
Linux or Windows on Intel x86_64	• `TRUE`: TDE and `DBMS_CRYPTO` program units use Micro Edition Suite (MES) 4.0.5.1 FIPS mode • `FALSE`: TDE and `DBMS_CRYPTO` program units use Intel Performance Primitives (IPP)

Table E-1 (Cont.) How the DBFIPS_140 Initialization Parameter Affects Platforms

Platform	Effect of Setting DBFIPS_140 to TRUE or FALSE
Solaris 11.1+ on either SPARC T-series or Intel x86_64	• `TRUE`: TDE and `DBMS_CRYPTO` program units use MES 4.1.2 FIPS mode • `FALSE`: TDE and `DBMS_CRYPTO` program units use Solaris Cryptographic Framework (SCF)/UCrypto (separately validated for FIPS 140)
Other operating systems or hardware	• `TRUE`: TDE and `DBMS_CRYPTO` program units use MES 4.1.2 FIPS mode • `FALSE`: TDE and `DBMS_CRYPTO` program units use MES 4.1.2 non-FIPS mode

Be aware that setting `DBFIPS_140` to `TRUE` and thus using the underlying library in FIPS mode incurs a certain amount of overhead when the library is first loaded. This is due to the verification of the signature and the execution of the self tests on the library. Once the library is loaded, then there is no other impact on performance.

See Also:

Oracle Database Reference for more information about the `DBFIPS_140` initialization parameter

Configuration of FIPS 140-2 for Secure Sockets Layer

The `SSLFIPS_140` parameter configures Secure Sockets Layer (SSL).

Configuring the SSLFIPS_140 Parameter for Secure Sockets Layer

Setting the `SSLFIPS_140` parameter to `TRUE` in the `fips.ora` file configures the Secure Sockets Layer (SSL) adapter to run in FIPS mode.

1. Ensure that the `fips.ora` file is either located in the `$ORACLE_HOME/ldap/admin` directory, or is in a location pointed to by the `FIPS_HOME` environment variable.
2. In the `fips.ora` file, set `SSLFIPS_140`.

   For example, to set `SSLFIPS_140` to `TRUE`:

   ```
   SSLFIPS_140=TRUE
   ```

   This parameter is set to `FALSE` by default. You must set it to `TRUE` on both the client and the server for FIPS mode operation.
3. Repeat this procedure in any Oracle Database home for any database server or client.

When you set `SSLFIPS_140` to `TRUE`, Secure Sockets Layer cryptographic operations take place in the embedded RSA/Micro Edition Suite (MES) library in FIPS mode. These cryptographic operations are accelerated by the CPU when hardware acceleration is available and properly configured in the host hardware and software.

If you set `SSLFIPS_140` to `FALSE`, then Secure Sockets Layer cryptographic operations take place in the embedded RSA/Micro Edition Suite (MES) library in *non*-FIPS mode, and as with the `TRUE` setting, the operations are accelerated if possible.

For native encryption, this behavior of cryptographic operations landing in RSA/Micro Edition Suite (MES) and being accelerated is similar to the above, except that it is determined by the `FIPS_140` setting in `sqlnet.ora` (instead of the `SSL_FIPS140` setting in `fips.ora`).

Note:

The `SSLFIPS_140` parameter replaces the `SQLNET.SSLFIPS_140` parameter used in Oracle Database 10*g* release 2 (10.2). You must set the parameter in the `fips.ora` file, and not the `sqlnet.ora` file.

Approved SSL Cipher Suites for FIPS 140-2

A cipher suite is a set of authentication, encryption, and data integrity algorithms that exchange messages between network nodes.

During an SSL handshake, for example, the two nodes negotiate to see as to which cipher suite they will use when transmitting messages back and forth.

Only the following cipher suites are approved for FIPS validation:

- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_RSA_WITH_AES_256_CBC_SHA`
- `SSL_RSA_WITH_AES_128_CBC_SHA`
- `SSL_RSA_WITH_AES_256_GCM_SHA384`
- `SSL_RSA_WITH_3DES_EDE_CBC_SHA`

Oracle Database SSL cipher suites are automatically set to FIPS approved cipher suites. If you wish to configure specific cipher suites, you can do so by editing the `SSL_CIPHER_SUITES` parameter in the `sqlnet.ora` or the `listener.ora` file.

```
SSL_CIPHER_SUITES=(SSL_cipher_suite1[,SSL_cipher_suite2[,..]])
```

You can also use Oracle Net Manager to set this parameter on the server and the client.

Postinstallation Checks for FIPS 140-2

After you configure the FIPS 140-2 settings, you must verify permissions in the operating system.

The permissions are as follows:

- Set execute permissions on all Oracle executable files to prevent the execution of Oracle Cryptographic Libraries by users who are unauthorized to do so, in accordance with the system security policy.

- Set read and write permissions on all Oracle executable files to prevent accidental or deliberate reading or modification of Oracle Cryptographic Libraries by any user.

To comply with FIPS 140-2 Level 2 requirements, in the security policy, include procedures to prevent unauthorized users from reading, modifying or executing Oracle Cryptographic Libraries processes and the memory they are using in the operating system.

Verifying FIPS 140-2 Connections

To check if FIPS mode is enabled for SSL, you can enable tracing in the `sqlnet.ora` file.

You can find FIPS self-test messages in the trace file.

1. Add the following lines to `sqlnet.ora` to enable tracing:

   ```
   trace_directory_server=trace_dir
   trace_file_server=trace_file
   trace_level_server=trace_level
   ```

   For example:

   ```
   trace_directory=/private/oracle/owm
   trace_file_server=fips_trace.trc
   trace_level_server=6
   ```

2. To check if FIPS mode is enabled for TDE and `DBMS_CRYPTO`, log into SQL*Plus and run the following command:

   ```
   SHOW PARAMETER DBFIPS_140
   ```

Trace level 6 is the minimum trace level required to check the results of the FIPS self-tests.

F

Managing Public Key Infrastructure (PKI) Elements

You can use the `orapki` command line utility and sqlnet.ora parameters to manage public key infrastructure (PKI) elements.

Uses of the orapki Utility

The `orapki` utility manages public key infrastructure (PKI) elements, such as wallets and certificate revocation lists, from the command line.

This way, you can automate these tasks by using scripts. Providing a way to incorporate the management of PKI elements into scripts makes it possible to automate many of the routine tasks of maintaining a PKI.

You can use the `orapki` command-line utility to perform the following tasks:

- Creating and viewing signed certificates for testing purposes
- Manage Oracle wallets (except for Transparent Data Encryption keystores):
  - Create and display Oracle wallets
  - Add and remove certificate requests
  - Add and remove certificates
  - Add and remove trusted certificates
- Manage certificate revocation lists (CRLs):
  - Renaming CRLs with a hash value for certificate validation
  - Uploading, listing, viewing, and deleting CRLs in Oracle Internet Directory

Note:

The use of PKI encryption with Transparent Data Encryption is deprecated. To configure Transparent Data Encryption, use the `ADMINISTER KEY MANAGEMENT` SQL statement. See *Oracle Database Advanced Security Guide* for more information.

orapki Utility Syntax

The `orapki` utility syntax specifies an Oracle wallet, a certificate revocation list, or a PKI digital certificate.

The syntax of the `orapki` command-line utility is as follows:

```
orapki module command -parameter value
```

In this specification, *module* can be `wallet` (Oracle wallet), `crl` (certificate revocation list), or `cert` (PKI digital certificate). The available commands depend on the `module` you are using.

For example, if you are working with a `wallet`, then you can add a certificate or a key to the wallet with the `add` command. The following example adds the user certificate located at `/private/lhale/cert.txt` to the wallet located at `$ORACLE_HOME/wallet/ewallet.p12`:

```
orapki wallet add -wallet $ORACLE_HOME/wallet/ewallet.p12 -user_cert -cert /private/
lhale/cert.txt
```

Creating Signed Certificates for Testing Purposes

The `orapki` utility provides a convenient, lightweight way to create signed certificates for testing purposes.

- To create a signed certificate for testing purposes, use the following command:

  ```
  orapki cert create [-wallet wallet_location] -request
  certificate_request_location -cert certificate_location -validity number_of_days
  [-summary]
  ```

This command creates a signed certificate from the certificate request. The `-wallet` parameter specifies the wallet containing the user certificate and private key that will be used to sign the certificate request. The `-validity` parameter specifies the number of days, starting from the current date, that this certificate will be valid. Specifying a certificate and certificate request is mandatory for this command.

Viewing a Certificate

After you create a certificate, you can use the `orapki` utility to view it.

- To view a certificate, use the following command:

  ```
  orapki cert display -cert certificate_location [-summary | -complete]
  ```

This command enables you to view a test certificate that you have created with `orapki`. You can choose either `-summary` or `-complete`, which determines how much detail the command will display. If you choose `-summary`, the command will display the certificate and its expiration date. If you choose `-complete`, it will display additional certificate information, including the serial number and public key.

Controlling MD5 and SHA-1 Certificate Use

You can use the `sqlnet.ora` file to control whether MD5 and SHA-1 signed certificates are accepted.

To control whether the MD5 and SHA-1 signed certificates are accepted, you can edit the `sqlnet.ora` file to enable or disable their use.

1. Log in to the server where the Oracle database resides.
2. Edit the `sqlnet.ora` file.

By default, the `sqlnet.ora` file is located in the `$ORACLE_HOME/dbs` directory or in the location set by the `TNS_ADMIN` environment variable.

3. Set the following parameters:
   - `ACCEPT_MD5_CERTS` controls the use of MD5 certificates. The default is `FALSE`. This parameter replaces the `ORACLE_SSL_ALLOW_MD5_CERT_SIGNATURES` environment variable.
   - `ACCEPT_SHA1_CERTS` controls the use of SHA-1 certificates. The default is `TRUE`. .

Managing Oracle Wallets with orapki Utility

The `orapki` utility can create, view, modify wallets; it can add and export certificates and certificate requests.

About Managing Wallets with orapki

You should understand the `orapki` command-line utility syntax used to create and manage Oracle wallets.

You can use the `orapki` utility `wallet` module commands in scripts to automate the wallet creation process. For example, you can create PKCS#12 wallets and auto-login wallets. You can create auto-login wallets that are associated with PKCS#12 wallets or auto-login wallets that are local to the computer on which they were created and the user who created them. You can view wallets, modify wallet passwords, and convert wallets to use the AES256 algorithm.

Note:

The `-wallet` parameter is mandatory for all `wallet` module commands.

Creating, Viewing, and Modifying Wallets with orapki

You can use `orapki` to perform a range of management activities with Oracle wallets.

Creating a PKCS#12 Wallet

You can use the `orapki` utility to create a PKCS#12 Oracle wallet.

- To create an Oracle PKCS#12 wallet (`ewallet.p12`), use the `orapki wallet create` command.

  ```
  orapki wallet create -wallet wallet_location [-pwd password]
  ```

This command prompts you to enter and reenter a wallet password, if no password has been specified on the command line. It creates a wallet in the location specified for `-wallet`.

Note:

For security reasons, Oracle recommends that you do not specify the password at the command line. You should supply the password only when prompted to do so.

Creating an Auto-Login Wallet

You can use the `orapki` utility to create an auto-login wallet.

- To create an auto-login wallet (`cwallet.sso`), which does not need a password to open the wallet, use the `orapki wallet create` command:

```
orapki wallet create -wallet wallet_location -auto_login_only
```

You can modify or delete the wallet without using a password. File system permissions provide the necessary security for such auto-login wallets.

You cannot move local auto-login wallets to another computer. They must be used on the host on which they are created.

Even though a local auto-login wallet does not need a password to open, you must supply the password for the associated PKCS#12 wallet in order to modify or delete the wallet. Any update to the PKCS#12 wallet also updates the associated auto-login wallet.

Creating an Auto-Login Wallet That Is Associated with a PKCS#12 Wallet

You can create an auto-login wallet that is associated with a PKCS#12 wallet.

The auto-login wallet does not need a password to open.

However, you must supply the password for the associated PKCS#12 wallet in order to modify or delete the wallet. Any update to the PKCS#12 wallet also updates the associated auto-login wallet.

- To create an auto-login wallet (`cwallet.sso`) that is associated with a PKCS#12 wallet (`ewallet.p12`), use the following command:

```
orapki wallet create -wallet wallet_location -auto_login [-pwd password]
```

This command creates a wallet with auto-login enabled (`cwallet.sso`) and associates it with a PKCS#12 wallet (`ewallet.p12`). The command prompts you to enter the password for the PKCS#12 wallet, if no password has been specified at the command line.

If the *`wallet_location`* already contains a PKCS#12 wallet, then auto-login is enabled for it. You must supply the password for the existing PKCS#12 wallet in order to enable auto-login for it.

If the *`wallet_location`* does not contain a PKCS#12 wallet, then a new PKCS#12 wallet is created. You must specify a password for the new PKCS#12 wallet.

If you want to turn the auto-login feature off for a PKCS#12 wallet, then use Oracle Wallet Manager.

See Also:

Oracle Database Enterprise User Security Administrator's Guide for more information

Creating an Auto-Login Wallet That Is Local to the Computer and User Who Created It

The `orapki` utility can create an auto-login wallet that is local to the computer of the user who created it.

- To create a local auto-login wallet that is local to both the computer on which it is created and the user who created it, use the following command:

```
orapki wallet create -wallet wallet_location -auto_login_local [-pwd password]
```

This command creates an auto-login wallet (`cwallet.sso`). It associates it with a PKCS#12 wallet (`ewallet.p12`). The command prompts you to enter the password for the PKCS#12 wallet, if no password has been specified at the command line.

Viewing a Wallet

You can use the `orapki` utility to view a wallet.

- To view an Oracle wallet, use the `orapki wallet display` command.

```
orapki wallet display -wallet wallet_location
```

This command displays the certificate requests, user certificates, and trusted certificates contained in the wallet, which must be a binary `PKCS12` file, with extension `.p12`. Other files will fail.

Modifying the Password for a Wallet

You can use the `orapki` utility to modify the password of a wallet.

- To change the wallet password, use the following command:

```
orapki wallet change_pwd -wallet wallet_location [-oldpwd password ] [-newpwd
password]
```

This command changes the current wallet password to the new password. The command prompts you for the old and new passwords if no password is supplied at the command line.

Note:

For security reasons, Oracle recommends that you do not specify the password options at the command line. You should supply the password when prompted to do so.

Converting an Oracle Wallet to Use the AES256 Algorithm

By default , an Oracle wallet with the `ADMINISTER KEY MANAGEMENT` or `ALTER SYSTEM` statement is encrypted with 3DES.

You can use the `orapki convert` command to convert the wallet to use the AES256 algorithm, which is stronger than the 3DES algorithm. Note that if you had created the wallet using `orapki` and not the `ADMINISTER KEY MANAGEMENT` or `ALTER SYSTEM` statement, then by default it uses the AES256 algorithm.

- To change the wallet algorithm from 3DES to AES256:

  ```
  orapki wallet convert -wallet wallet_location [-pwd password] [-compat_v12]
  ```

The `compat_v12` setting performs the conversion from 3DES to AES256.

Adding Certificates and Certificate Requests to Oracle Wallets with orapki

You can use the `orapki` utiltiy to perform a range of certificate-related tasks.

Adding a Certificate Request to an Oracle Wallet

You can use the `orapki` utility to add certificates and certificate requests to Oracle wallets.

- To add a certificate request to an Oracle wallet, use the `orapki wallet add` command.

  ```
  orapki wallet add -wallet wallet_location -dn user_dn -keySize 512|1024|2048
  ```

This command adds a certificate request to a wallet for the user with the specified distinguished name (`user_dn`). The request also specifies the requested certificate's key size (512, 1024, or 2048 bits). To sign the request, export it with the export option.

Adding a Trusted Certificate to an Oracle Wallet

You can use the `orapki` utility to add trusted certificates to an Oracle wallet.

- To add a trusted certificate to an Oracle wallet, use the following command:

  ```
  orapki wallet add -wallet wallet_location -trusted_cert -cert
  certificate_location
  ```

This command adds a trusted certificate, at the specified location (`-cert certificate_location`), to a wallet. You must add all trusted certificates in the certificate chain of a user certificate before adding a user certificate, or the command to add the user certificate will fail.

Adding a Root Certificate to an Oracle Wallet

You can use the `orapki` utility to add a root certificate to an Oracle wallet.

- To add a root certificate to an Oracle wallet, use the following command:

  ```
  orapki wallet add -wallet wallet_location -dn certificate_dn -keySize 512|1024|
  2048 -self_signed -validity number_of_days
  ```

This command creates a new self-signed (root) certificate and adds it to the wallet. The `-validity` parameter (mandatory) specifies the number of days, starting from the current date, that this certificate will be valid. You can specify a key size for this root certificate (`-keySize`) of 512, 1024, or 2048 bits.

Adding a User Certificate to an Oracle Wallet

You can use the `orapki` utility to add a user certificate to an Oracle wallet.

- To add a user certificate to an Oracle wallet, use the following command:

  ```
  orapki wallet add -wallet wallet_location -user_cert -cert certificate_location
  ```

This command adds the user certificate at the location specified with the `-cert` parameter to the Oracle wallet at the `wallet_location`. Before you add a user certificate to a wallet, you must add all the trusted certificates that make up the certificate chain. If all trusted certificates are not installed in the wallet before you add the user certificate, then adding the user certificate will fail.

Note:

For security reasons, Oracle recommends that you do not specify the password at the command line. You should supply the password when prompted to do so.

Verifying Credentials on the Hardware Device That Uses a PKCS#11 Wallet

You can verify credentials on the hardware device using the PKCS#11 wallet.

- Use the following command to verify the credential details:

  ```
  orapki wallet p11_verify -wallet wallet_location [-pwd password]
  ```

Adding PKCS#11 Information to an Oracle Wallet

A wallet that contains PKCS#11 information can be used like any Oracle wallet.

The private keys are stored on a hardware device. The cryptographic operations are also performed on the device.

- Use the following command to add PKCS#11 information to a wallet:

  ```
  orapki wallet p11_add -wallet wallet_location -p11_lib pkcs11Lib
  [-p11_tokenlabel tokenLabel] [-p11_tokenpw tokenPassphrase]
  [-p11_certlabel certLabel] [-pwd password]
  ```

In this specification:

- `wallet` specifies the wallet location.
- `p11_lib` specifies the path to the PKCS#11 library. This includes the library filename.
- `p11_tokenlabel` specifies the token or smart card used on the device. Use this when there are multiple tokens on the device. Token labels are set using vendor tools.

- `p11_tokenpw` specifies the password that is used to access the token. Token passwords are set using vendor tools.
- `p11_certlabel` is used to specify a certificate label on the token. Use this when a token contains multiple certificates. Certificate labels are set using vendor tools.
- `pwd` is used to specify the wallet password.

Exporting Certificates and Certificate Requests from Oracle Wallets with orapki

You can use the `orapki` utility to export certificates and certificate requests from Oracle wallets.

- To export a certificate from an Oracle wallet, use the following command:

  ```
  orapki wallet export -wallet wallet_location -dn certificate_dn -cert
  certificate_filename
  ```

This command exports a certificate with the subject's distinguished name (`-dn`) from a wallet to a file that is specified by `-cert`.

To export a certificate request from an Oracle wallet, use the following command:

```
orapki wallet export -wallet wallet_location -dn certificate_request_dn -request
certificate_request_filename
```

This command exports a certificate request with the subject's distinguished name (`-dn`) from a wallet to a file that is specified by `-request`.

Management of Certificate Revocation Lists (CRLs) with orapki Utility

You must manage certificate revocation lists (CRLs) with the `orapki` utility.

This utility creates a hashed value of the CRL issuer's name to identify the CRLs location in your system. If you do not use `orapki`, your Oracle server cannot locate CRLs to validate PKI digital certificates.

orapki Usage

Examples of `orapki` commands include creating wallets, user certificates, and wallets with self-signed certificates, and exporting certificates.

Example: Wallet with a Self-Signed Certificate and Export of the Certificate

The `orapki wallet add` command can create a wallet with a self-signed certificate; the `orapki wallet export` can export the certificate.

Example F-1 illustrates the steps to create a wallet with a self-signed certificate, view the wallet, and then export the certificate to a file.

Example F-1 Creating a Wallet with a Self-Signed Certificate and Exporting the Certificate

1. Create a wallet.

   For example:

   ```
   c -wallet /private/user/orapki_use/root
   ```

   The wallet is created at the location, `/private/user/orapki_use/root`.
2. Add a self-signed certificate to the wallet.

   ```
   orapki wallet add -wallet /private/user/orapki_use/root -dn
   'CN=root_test,C=US' -keysize 2048 -self_signed -validity 3650
   ```

   This creates a self-signed certificate with a validity of 3650 days. The distinguished name of the subject is `CN=root_test,C=US`. The key size for the certificate is 2048 bits.
3. View the wallet.

   ```
   orapki wallet display -wallet /private/user/orapki_use/root
   ```

   This is used to view the certificate contained in the wallet.
4. Export the certificate.

   ```
   orapki wallet export -wallet /private/user/orapki_use/root -dn
   'CN=root_test,C=US' -cert /private/user/orapki_use/root/b64certificate.txt
   ```

   This exports the self-signed certificate to the file, `b64certificate.txt`. Note that the distinguished name used is the same as in step 2.

Example: Creating a Wallet and a User Certificate

The `orapki` utility can create wallets and user certificates.

Example F-2 illustrates miscellaneous tasks related to creating user certificates.

The following steps illustrate creating a wallet, creating a certificate request, exporting the certificate request, creating a signed certificate from the request for testing, viewing the certificate, adding a trusted certificate to the wallet and adding a user certificate to the wallet.

Example F-2 Creating a Wallet and a User Certificate

1. Create a wallet with auto-login enabled.

   For exmaple:

   ```
   orapki wallet create -wallet /private/user/orapki_use/server -auto_login
   ```

   This creates a wallet at `/private/user/orapki_use/server` with auto-login enabled.
2. Add a certificate request to the wallet.

   ```
   orapki wallet add -wallet /private/user/orapki_use/server/ewallet.p12 -dn
   'CN=server_test,C=US' -keysize 2048
   ```

   This adds a certificate request to the wallet that was created (`ewallet.p12`). The distinguished name of the subject is `CN=server_test,C=US`. The key size specified is 2048 bits.

3. Export the certificate request to a file.

   ```
   orapki wallet export -wallet /private/user/orapki_use/server -dn
   'CN=server_test,C=US' -request /private/user/orapki_use/server/creq.txt
   ```

   This exports the certificate request to the specified file, which is `creq.txt` in this case.

4. Create a signed certificate from the request for test purposes.

   ```
   orapki cert create -wallet /private/user/orapki_use/root -request /private/user/
   orapki_use/server/creq.txt -cert /private/user/orapki_use/server/cert.txt -
   validity 3650
   ```

   This creates a certificate, `cert.txt` with a validity of 3650 days. The certificate is created from the certificate request generated in the preceding step.

5. View the certificate.

   ```
   orapki cert display -cert /private/user/orapki_use/server/cert.txt -complete
   ```

   This displays the certificate generated in the preceding step. The `-complete` option enables you to display additional certificate information, including the serial number and public key.

6. Add a trusted certificate to the wallet.

   ```
   orapki wallet add -wallet /private/user/orapki_use/server/ewallet.p12 -
   trusted_cert -cert /private/user/orapki_use/root/b64certificate.txt
   ```

   This adds a trusted certificate, `b64certificate.txt` to the `ewallet.p12` wallet. You must add all trusted certificates in the certificate chain of a user certificate before adding a user certificate.

7. Add a user certificate to the wallet.

   ```
   orapki wallet add -wallet /private/user/orapki_use/server/ewallet.p12 -user_cert
   -cert /private/user/orapki_use/server/cert.txt
   ```

   This command adds the user certificate, `cert.txt` to the `ewallet.p12` wallet.

orapki Utility Commands Summary

The `orapki` commands perform a variety of wallet, certificate revocation lists (CRL), and certificate management tasks.

orapki cert create

The `orapki cert create` command creates a signed certificate for testing purposes.

Syntax

```
orapki cert create [-wallet wallet_location] -request certificate_request_location -
cert certificate_location -validity number_of_days [-summary]
```

- `wallet` specifies the wallet containing the user certificate and private key that will be used to sign the certificate request.
- `request` (mandatory) specifies the location of the certificate request for the certificate you are creating.

- `cert` (mandatory) specifies the directory location where the tool places the new signed certificate.
- `validity` (mandatory) specifies the number of days, starting from the current date, that this certificate will be valid.

orapki cert display

The `orapki cert display` command displays details of a specific certificate.

Syntax

```
orapki cert display -cert certificate_location [-summary|-complete]
```

- `cert` specifies the location of the certificate you want to display.
- You can use either the `-summary` or the `-complete` parameter to display the following information:
  - `summary` displays the certificate and its expiration date
  - `complete` displays additional certificate information, including the serial number and public key

orapki crl delete Command

The `orapki crl delete` command deletes a certificate revocation list (CRL) from Oracle Internet Directory.

The user who deletes the CRLs from the directory by using `orapki` must be a member of the `CRLAdmins` (`cn=CRLAdmins,cn=groups,%s_OracleContextDN%`) directory group.

Prerequisites

None

Syntax

```
orapki crl delete -issuer issuer_name -ldap hostname:ssl_port -user username [-
wallet wallet_location] [-summary]
```

- `issuer` specifies the name of the certificate authority (CA) who issued the CRL.
- `ldap` specifies the host name and SSL port for the directory where the CRLs are to be deleted. Note that this must be a directory SSL port with no authentication.

  See also Uploading CRLs to Oracle Internet Directory for more information about this port.
- `user` specifies the user name of the directory user who has permission to delete CRLs from the CRL subtree in the directory.
- `wallet` (optional) specifies the location of the wallet that contains the certificate of the certificate authority (CA) who issued the CRL. Using it causes the tool to verify the validity of the CRL against the CA's certificate prior to deleting it from the directory.
- `summary` is optional. It displays the CRL LDAP entry that was deleted.

orapki crl display

The `orapki crl display` command displays a specified certificate revocation list (CRL) that is stored in Oracle Internet Directory.

Syntax

```
orapki crl display -crl crl_location [-wallet wallet_location] [-summary|-complete]
```

- `crl` parameter specifies the location of the CRL in the directory. It is convenient to paste the CRL location from the list that displays when you use the `orapki crl list` command. See orapki crl list.
- `wallet` (optional) specifies the location of the wallet that contains the certificate of the certificate authority (CA) who issued the CRL. Using it causes the tool to verify the validity of the CRL against the CA's certificate prior to displaying it.
- `summary` and `complete` display the following information:
  - `summary` provides a listing that contains the CRL issuer's name and the CRL's validity period
  - `complete` provides a list of all revoked certificates that the CRL contains. Note that this option may take a long time to display, depending on the size of the CRL.

orapki crl hash

The `orapki crl hash` command generates a hash value of the certificate revocation list (CRL) issuer to identify the CRL file system location for certificate validation.

Syntax

```
orapki crl hash -crl crl_filename|URL [-wallet wallet_location] [-symlink|-copy]
crl_directory [-summary]
```

- `crl` specifies the filename that contains the CRL or the URL where it can be found.
- `wallet` (optional) specifies the location of the wallet that contains the certificate of the certificate authority (CA) who issued the CRL. Using it causes the tool to verify the validity of the CRL against the CA's certificate prior to uploading it to the directory.
- Depending on the operating system, use either the `-symlink` or the `-copy` parameter:
  - (UNIX) `symlink` creates a symbolic link to the CRL at the `crl_directory` location
  - (Windows) `copy` creates a copy of the CRL at the `crl_directory` location
- `summary` (optional) displays the CRL issuer's name.

orapki crl list

The `orapki crl list` command displays a list of certificate revocation lists (CRLs) stored in Oracle Internet Directory.

Syntax

This is useful for browsing to locate a particular CRL to view or download to your local file system.

```
orapki crl list -ldap hostname:ssl_port
```

`ldap` specifies the host name and SSL port for the directory server from where you want to list CRLs. Note that this must be a directory SSL port with no authentication.

See Also:

Uploading CRLs to Oracle Internet Directory for more information about this port

orapki crl upload

The `orapki crl upload` command uploads a certificate revocation list (CRL) to the CRL subtree in Oracle Internet Directory.

Note that you must be a member of the directory administrative group `CRLAdmins` (`cn=CRLAdmins,cn=groups,%s_OracleContextDN%`) to upload CRLs to the directory.

Syntax

```
orapki crl upload -crl crl_location -ldap hostname:ssl_port -user username [-wallet
wallet_location] [-summary]
```

- `crl` specifies the directory location or the URL where the CRL is located that you are uploading to the directory.
- `ldap` specifies the host name and SSL port for the directory where you are uploading the CRLs. Note that this must be a directory SSL port with no authentication.

  See also Uploading CRLs to Oracle Internet Directory for more information about this port.
- `user` specifies the user name of the directory user who has permission to add CRLs to the CRL subtree in the directory.
- `wallet` specifies the location of the wallet that contains the certificate of the certificate authority (CA) who issued the CRL. This is an optional parameter. Using it causes the tool to verify the validity of the CRL against the CA's certificate prior to uploading it to the directory.
- `summary` is optional. It displays the CRL issuer's name and the LDAP entry where the CRL is stored in the directory.

orapki wallet add

The `orapki wallet add` command adds certificate requests and certificates to an Oracle wallet.

Syntax

To add certificate requests:

```
orapki wallet add -wallet wallet_location -dn user_dn -keySize 512|1024|2048
```

- `wallet` specifies the location of the wallet to which you want to add a certificate request.
- `dn` specifies the distinguished name of the certificate owner.
- `keySize` specifies the key size for the certificate.
- To sign the request, export it with the export option. Refer to orapki wallet export

To add trusted certificates:

```
orapki wallet add -wallet wallet_location -trusted_cert -cert certificate_location
```

- `trusted_cert` adds the trusted certificate, at the location specified with `-cert`, to the wallet.

To add root certificates:

```
orapki wallet add -wallet wallet_location -dn certificate_dn -keySize 512|1024|2048 -
self_signed -validity number_of_days
```

- `self_signed` creates a root certificate.
- `validity` is mandatory. Use it to specify the number of days, starting from the current date, that this root certificate will be valid.

To add user certificates:

```
orapki wallet add -wallet wallet_location -user_cert -cert certificate_location
```

- `user_cert` adds the user certificate at the location specified with the `-cert` parameter to the wallet. Before you add a user certificate to a wallet, you must add all the trusted certificates that make up the certificate chain. If all trusted certificates are not installed in the wallet before you add the user certificate, then adding the user certificate will fail.

orapki wallet convert

The `orapki wallet convert` command converts the 3DES algorithm in an Oracle wallet to use the AES256 algorithm.

Syntax

```
orapki wallet convert -wallet wallet_location [-pwd password] [-compat_v12]
```

- `wallet` specifies a location for the new wallet or the location of the wallet for which you want to turn on auto-login.
- `pwd` is the wallet password.

- `compat_v12` performs the conversion from 3DES to AES256.

orapki wallet create

The `orapki wallet create` command creates an Oracle wallet or enables auto-login for an Oracle wallet.

Syntax

```
orapki wallet create -wallet wallet_location [-auto_login|-auto_login_local]
```

- `wallet` specifies a location for the new wallet or the location of the wallet for which you want to turn on auto-login.
- `auto_login` creates an auto-login wallet, or it turns on automatic login for the wallet specified with the `-wallet` option.

  See also *Oracle Database Enterprise User Security Administrator's Guide* for details about auto-login wallet.
- `auto_login_local` creates a local auto-login wallet, or it turns on local automatic login for the wallet specified with the `-wallet` option.

orapki wallet display

The `orapki wallet display` command displays the certificate requests, user certificates, and trusted certificates in an Oracle wallet.

Syntax

```
orapki wallet display -wallet wallet_location
```

- `wallet` specifies a location for the wallet you want to open if it is not located in the current working directory.

orapki wallet export

The `orapki wallet export` command exports certificate requests and certificates from an Oracle wallet.

Syntax

To export a certificate from an Oracle wallet:

```
orapki wallet export -wallet wallet_location -dn certificate_dn -cert
certificate_filename
```

- `wallet` specifies the location of the wallet from which you want to export the certificate.
- `dn` specifies the distinguished name of the certificate.
- `cert` specifies the name of the file that contains the exported certificate.

To export a certificate request from an Oracle wallet:

```
orapki wallet export -wallet wallet_location -dn certificate_request_dn -request
certificate_request_filename
```

- `request` specifies the name of the file that contains the exported certificate request.

G

How the Unified Auditing Migration Affects Individual Audit Features

Most of the pre-Oracle Database 12*c* release 1 (12.1) auditing features can be used before a unified auditing migration.

Table G-1 describes how the pre-Oracle Database 12*c* audit features change in the migration.

Table G-1 Availability of Unified Auditing Features Before and After Migration

Feature	Availability in Pre-Migrated Environment	Availability in Post-Migrated Environment
General Auditing Features	-	-
Operating system audit trail	Yes	No
XML file audit trail	Yes	No
Network auditing	Yes	No
The ability of users to audit and to removing auditing from their own schema objects	Yes	No
Mandatory auditing of audit administrative actions	No	Yes
Auditing Roles	-	-
`AUDIT_ADMIN`	Yes, but not needed for users who want to audit their own objects, nor for users who already have the `ALTER SYSTEM` privilege and want to change the auditing initialization parameters	Yes
`AUDIT_VIEWER`	Yes	Yes
System Tables	-	-
`SYS.AUD$`	Yes	Yes, but will only have pre-unified audit records
`SYS.FGA_LOG$`	Yes	Yes, but will only have pre-unified audit records
Initialization Parameters	-	-
`AUDIT_TRAIL`	Yes	Yes, but will not have any effect
`AUDIT_FILE_DEST`	Yes	Yes, but will not have any effect
`AUDIT_SYS_OPERATIONS`	Yes	Yes, but will not have any effect
`AUDIT_SYSLOG_LEVEL`	Yes	Yes, but will not have any effect
`UNIFIED_AUDIT_SGA_QUEUE_SIZE`	Yes, but note that this parameter has been deprecated, but is currently retained for backward compatibility.	Yes, but note that this parameter has been deprecated, but is currently retained for backward compatibility.

Table G-1 (Cont.) Availability of Unified Auditing Features Before and After Migration

Feature	Availability in Pre-Migrated Environment	Availability in Post-Migrated Environment
Data Dictionary Views [1]	-	-
ALL_AUDIT_POLICIES	Yes	Yes, but only if fine-grained audit policies are created using the DBMS_FGA PL/SQL package
DBA_AUDIT_POLICIES	Yes	Yes, but only if fine-grained audit policies are created using the DBMS_FGA PL/SQL package
DBA_AUDIT_POLICY_COLUMNS	Yes	Yes, but only if fine-grained audit policies are created using the DBMS_FGA PL/SQL package
DBA_COMMON_AUDIT_TRAIL	Yes	Yes, but will only have pre-unified audit records
DBA_AUDIT_EXISTS	Yes	Yes
DBA_AUDIT_OBJECT	Yes	Yes
DBA_AUDIT_POLICIES	Yes	Yes, but only if fine-grained audit policies are created using the DBMS_FGA PL/SQL package
DBA_AUDIT_POLICY_COLUMNS	Yes	Yes, but only if fine-grained audit policies are created using the DBMS_FGA PL/SQL package
DBA_AUDIT_SESSION	Yes	Yes, but will only have pre-unified audit records
DBA_AUDIT_STATEMENT	Yes	Yes, but will only have pre-unified audit records
DBA_AUDIT_TRAIL	Yes	Yes, but will only have pre-unified audit records. The RLS_INFO column captures audited Oracle VPD predicates.
DBA_FGA_AUDIT_TRAIL	Yes	Yes, but will only have pre-unified audit records. The RLS_INFO column captures audited Oracle VPD predicates.
DBA_OBJ_AUDIT_OPTS	Yes	Yes
DBA_PRIV_AUDIT_OPTS	Yes	Yes
DBA_STMT_AUDIT_OPTS	Yes	Yes
UNIFIED_AUDIT_TRAIL	Yes, but does not collect any audit records	Yes, and collects audit records
USER_AUDIT_OBJECT	Yes	Yes
USER_AUDIT_POLICY_COLUMN	Yes	Yes, but only if fine-grained audit policies are created using the DBMS_FGA PL/SQL package
USER_AUDIT_POLICIES	Yes	Yes, but only if fine-grained audit policies are created using the DBMS_FGA PL/SQL package

Table G-1 (Cont.) Availability of Unified Auditing Features Before and After Migration

Feature	Availability in Pre-Migrated Environment	Availability in Post-Migrated Environment
`USER_AUDIT_SESSION`	Yes	Yes
`USER_AUDIT_STATEMENT`	Yes	Yes
`USER_AUDIT_TRAIL`	Yes	Yes, but will only have pre-unified audit records
`USER_OBJ_AUDIT_OPTS`	Yes	Yes
`V$XML_AUDIT_TRAIL`	Yes	Yes, but will only have pre-unified audit records. The `RLS_INFO` column captures audited Oracle VPD predicates.
CREATE AUDIT POLICY, ALTER AUDIT POLICY, and DROP AUDIT POLICY Statements	The statements are available, but the audit policies will not write to the old audit trails. When a policy is enabled, its audit records are written to the unified audit trail.	Yes, but writes the audit record to the unified audit trail only
AUDIT and NOAUDIT Statements	-	-
`AUDIT`	Yes, and can be used in a multitenant environment	Yes, but enhanced to enable audit policies; create application context audit settings; create audit records on success, failure, or both; and use in a multitenant environment
`NOAUDIT`	Yes, and can be used in a multitenant environment	Yes, but changed to disable audit policies, disable application context audit settings, and use in a multitenant environment
DBMS_FGA.ADD_POLICY Procedure Parameters	-	-
`audit_trail`	Yes, and is used as in previous releases	Yes, but when unified auditing is enabled, you can omit this parameter because all records will be written to the unified audit trail.
DBMS_AUDIT_MGMT Package AUDIT_TRAIL_TYPE Property Options	-	-
`DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD`	Yes	Yes, but only pre-unified audit records
`DBMS_AUDIT_MGMT.AUDIT_TRAIL_FGA_STD`	Yes	Yes, but only pre-unified audit records
`DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD`	Yes	Yes, but only pre-unified audit records
`DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS`	Yes	Yes, but only pre-unified audit records
`DBMS_AUDIT_MGMT.AUDIT_TRAIL_XML`	Yes	Yes, but only pre-unified audit records
`DBMS_AUDIT_MGMT.AUDIT_TRAIL_FILES`	Yes	Yes, but only pre-unified audit records

Table G-1 (Cont.) Availability of Unified Auditing Features Before and After Migration

Feature	Availability in Pre-Migrated Environment	Availability in Post-Migrated Environment
DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL	Yes	Yes, but only pre-unified audit records
Oracle Database Vault Features	-	-
DVSYS.AUDIT_TRAIL$ system table	Yes	Is renamed to DVSYS.OLD_AUDIT_TRAIL$ and retains the old audit records. The previous DVSYS.AUDIT_TRAIL$ table is made into a view named DVSYS.AUDIT_TRAIL$. No new audit records are added.
Oracle Label Security Features	-	-
SA_AUDIT_ADMIN PL/SQL package	Yes	No

1 These data dictionary views will continue to show audit data from audit records that are still in the SYS.AUD$ and SYS.FGA_LOG$ system tables. Unified audit trail records are shown only in the unified audit trail-specific views. You must be granted the AUDIT_ADMIN or AUDIT_VIEWER role to query any views that are not prefaced with USER_.

Glossary

access control

The ability of a system to grant or limit access to specific data for specific clients or groups of clients.

Access Control Lists (ACLs)

The group of access directives that you define. The directives grant levels of access to specific data for specific clients, or groups of clients, or both.

Advanced Encryption Standard

Advanced Encryption Standard (AES) is a new cryptographic algorithm that has been approved by the National Institute of Standards and Technology as a replacement for DES. The AES standard is available in Federal Information Processing Standards Publication 197. The AES algorithm is a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits.

AES

See Advanced Encryption Standard

application context

A name-value pair that enables an application to access session information about a user, such as the user ID or other user-specific information, and then securely pass this data to the database.

See also global application context.

attribute

An item of information that describes some aspect of an entry in an LDAP directory. An entry comprises a set of attributes, each of which belongs to an object class. Moreover, each attribute has both a *type*, which describes the kind of information in the attribute, and a *value,* which contains the actual data.

application role

A database role that is granted to application users and that is secured by embedding passwords inside the application.

See also secure application role.

authentication

The process of verifying the identity of a user, device, or other entity in a computer system, often as a prerequisite to granting access to resources in a system. A recipient of an authenticated message can be certain of the message's origin (its sender). Authentication is presumed to preclude the possibility that another party has impersonated the sender.

authentication method

A security method that verifies a user's, client's, or server's identity in distributed environments. Network authentication methods can also provide the benefit of single sign-on (SSO) for users. The following authentication methods are supported:

- Kerberos
- RADIUS
- Secure Sockets Layer (SSL)
- Windows native authentication

authorization

Permission given to a user, program, or process to access an object or set of objects. In Oracle, authorization is done through the role mechanism. A single person or a group of people can be granted a role or a group of roles. A role, in turn, can be granted other roles. The set of privileges available to an authenticated entity.

auto-login wallet

Password-based access to services without providing credentials at the time of access. This auto-login access stays in effect until the auto-login feature is disabled for that wallet. File system permissions provide the necessary security for auto-login wallet. When auto-login is enabled for a wallet, it is only available to the operating system user who created that wallet. Sometimes these are called "SSO wallets" because they provide single sign-on capability.

CDB

Multitenant container database. An Oracle Database installation that contains one root zero or more pluggable databases (PDBs). Every Oracle database is either a CDB or a non-CDB.

base

The root of a subtree search in an LDAP-compliant directory.

CA

See certificate authority

certificate

An ITU x.509 v3 standard data structure that securely binds an identify to a public key.

A certificate is created when an entity's public key is signed by a trusted identity, a certificate authority. The certificate ensures that the entity's information is correct, and that the public key belongs to that entity.

A certificate contains the entity's name, identifying information, and public key. It is also likely to contain a serial number, expiration date, and information about the rights, uses, and privileges associated with the certificate. Finally, it contains information about the certificate authority that issued it.

certificate authority

A trusted third party that certifies that other entities—users, databases, administrators, clients, servers—are who they say they are. When it certifies a user, the certificate authority first seeks verification that the user is not on the certificate revocation list (CRL), then verifies the user's identity and grants a certificate, signing it with the certificate authority's private key. The certificate authority has its own certificate and public key which it publishes. Servers and clients use these to verify signatures the certificate authority has made. A certificate authority might be an external company that offers certificate services, or an internal organization such as a corporate MIS department.

certificate chain

An ordered list of certificates containing an end-user or subscriber certificate and its certificate authority certificates.

certificate request

A certificate request, which consists of three parts: certification request information, a signature algorithm identifier, and a digital signature on the certification request information. The certification request information consists of the subject's distinguished name, public key, and an optional set of attributes. The attributes may provide additional information about the subject identity, such as postal address, or a challenge password by which the subject entity may later request certificate revocation. See PKCS #10.

certificate revocation list (CRL)

(CRLs) Signed data structures that contain a list of revoked certificate **s**. The authenticity and integrity of the CRL is provided by a digital signature appended to it. Usually, the CRL signer is the same entity that signed the issued certificate.

checksumming

A mechanism that computes a value for a message packet, based on the data it contains, and passes it along with the data to authenticate that the data has not been tampered with. The recipient of the data recomputes the cryptographic checksum and compares it with the cryptographic checksum passed with the data; if they match, it is "probabilistic" proof the data was not tampered with during transmission.

cleartext

Unencrypted plain text.

Cipher Block Chaining (CBC)

An encryption method that protects against block replay attacks by making the encryption of a cipher block dependent on all blocks that precede it; it is designed to make unauthorized decryption incrementally more difficult. Oracle Database employs *outer* cipher block chaining because it is more secure than *inner* cipher block chaining, with no material performance penalty.

CIDR

The standard notation used for IP addresses. In CIDR notation, an IPv6 subnet is denoted by the subnet prefix and the size in bits of the prefix (in decimal), separated by the slash (/) character. For example, `fe80:0000:0217:f2ff::/64` denotes a subnet with addresses `fe80:0000:0217:f2ff:0000:0000:0000:0000` through `fe80:0000:0217:f2ff:ffff:ffff:ffff:ffff`. The CIDR notation includes support for IPv4 addresses. For example, `192.0.2.1/24` denotes the subnet with addresses `192.0.2.1` through `192.0.2.255`.

cipher suite

A set of authentication, encryption, and data integrity algorithms used for exchanging messages between network nodes. During an SSL handshake, for example, the two nodes negotiate to see which cipher suite they will use when transmitting messages back and forth.

cipher suite name

Cipher suites describe the kind of cryptographics protection that is used by connections in a particular session.

ciphertext

Message text that has been encrypted.

Classless Inter-Domain Routing

See CIDR .

client

A client relies on a service. A client can sometimes be a user, sometimes a process acting on behalf of the user during a database link (sometimes called a proxy).

common privilege grant

A privilege that a common user grants to another common user or to a common role. Common privilege grants can be either system privileges or object privileges, and they apply across all PDBs in a CDB.

See also local privilege grant.

common role

A role that exists in all containers in a CDB.

common user

In a CDB, a database user that exists with the same identity in every existing and future PDB.

confidentiality

A function of cryptography. Confidentiality guarantees that only the intended recipient(s) of a message can view the message (decrypt the ciphertext).

connect descriptor

A specially formatted description of the destination for a network connection. A connect descriptor contains destination service and network route information. The destination service is indicated by using its service name for Oracle9*i* or Oracle8*i* databases or its Oracle system identifier (SID) for Oracle databases version 8.0. The network route provides, at a minimum, the location of the listener through use of a network address. See connect identifier

connect identifier

A name, net service name, or service name that resolves to a connect descriptor. Users initiate a connect request by passing a user name and password along with a connect identifier in a connect string for the service to which they want to connect.

For example:

```
CONNECT username@connect_identifier
Enter password: password
```

connect string

Information the user passes to a service to connect, such as user name, password and net service name. For example:

```
CONNECT username@net_service_name
Enter password: password
```

container

In a CDB either, a root or a PDB.

container data object

In a CDB, a table or view containing data pertaining to multiple containers and possibly the CDB as a whole, along with mechanisms to restrict data visible to specific common users through such objects to one or more containers. Examples of container data objects are Oracle-supplied views whose names begin with `V$` and `CDB_`.

credentials

A user name, password, or certificate used to gain access to the database.

CRL

See certificate revocation list (CRL)

CRL Distribution Point

(CRL DP) An optional extension specified by the X.509 version 3 certificate standard, which indicates the location of the Partitioned CRL where revocation information for a certificate is stored. Typically, the value in this extension is in the form of a URL. CRL DPs allow revocation information within a single certificate authority domain to be posted in multiple CRLs. CRL DPs subdivide revocation information into more manageable pieces to avoid proliferating voluminous CRLs, thereby providing performance benefits. For example, a CRL DP is specified in the certificate and can point to a file on a Web server from which that certificate's revocation information can be downloaded.

CRL DP

See CRL Distribution Point

cryptography

The practice of encoding and decoding data, resulting in secure messages.

data dictionary

A set of read-only tables that provide information about a database.

Data Encryption Standard (DES)

An older Federal Information Processing Standards encryption algorithm superseded by the Advanced Encryption Standard (AES).

database administrator

(1) A person responsible for operating and maintaining an Oracle Server or a database application. (2) An Oracle user name that has been given DBA privileges and can perform database administration functions. Usually the two meanings coincide. Many sites have multiple DBAs.

database alias

See net service name

Database Installation Administrator

Also called a database creator. This administrator is in charge of creating new databases. This includes registering each database in the directory using the Database Configuration Assistant. This administrator has create and modify access to database service objects and attributes. This administrator can also modify the Default domain.

database link

A network object stored in the local database or in the network definition that identifies a remote database, a communication path to that database, and optionally, a user name and password. Once defined, the database link is used to access the remote database.

A public or private database link from one database to another is created on the local database by a DBA or user.

A global database link is created automatically from each database to every other database in a network with Oracle Names. Global database links are stored in the network definition.

database password version

An irreversible value that is derived from the user's database password. It is also called a password verifier. This value is used during password authentication to the database to prove the identity of the connecting user.

Database Security Administrator

The highest level administrator for database enterprise user security. This administrator has permissions on all of the enterprise domains and is responsible for:

- Administering the Oracle `DBSecurityAdmins` and `OracleDBCreators` groups.

Creating new enterprise domain**s**.

- Moving databases from one domain to another within the enterprise.

decryption

The process of converting the contents of an encrypted message (ciphertext) back into its original readable format (plaintext).

definer's rights procedure

A procedure (or program unit) that executes with the privileges of its owner, not its current user. Definer's rights subprograms are bound to the schema in which they are located.

For example, assume that user `blake` and user `scott` each have a table called `dept` in their respective user schemas. If user `blake` calls a definer's rights procedure, which is owned by user `scott`, to update the `dept` table, then this procedure will update the `dept` table in the `scott` schema. This is because the procedure executes with the privileges of the user who owns (defined) the procedure (that is, `scott`).

See also invoker's rights procedure.

DES

See Data Encryption Standard (DES)

dictionary attack

A common attack on passwords. The attacker creates a list of many common passwords and encrypts them. Then the attacker steals a file containing encrypted passwords and compares it to his list of encrypted common passwords. If any of the encrypted password values (called verifiers) match, then the attacker can steal the corresponding password. Dictionary attacks can be avoided by using "salt" on the password before encryption. See salt.

Diffie-Hellman key negotiation algorithm

This is a method that lets two parties communicating over an insecure channel to agree upon a random number known only to them. Though the parties exchange information over the insecure channel during execution of the Diffie-Hellman key negotiation algorithm, it is computationally infeasible for an attacker to deduce the random number they agree upon by analyzing their network communications. Oracle Database uses the Diffie-Hellman key negotiation algorithm to generate session keys.

digital signature

A digital signature is created when a public key algorithm is used to sign the sender's message with the sender's private key. The digital signature assures that the document is authentic, has not been forged by another entity, has not been altered, and cannot be repudiated by the sender.

directory information tree (DIT)

A hierarchical tree-like structure consisting of the DNs of the entries in an LDAP directory. See distinguished name (DN)

directory naming

A naming method that resolves a database service, net service name, or net service alias to a connect descriptor stored in a central directory server. A

directory naming context

A subtree which is of significance within a directory server. It is usually the top of some organizational subtree. Some directories only permit one such context which is fixed; others permit none to many to be configured by the directory administrator.

distinguished name (DN)

The unique name of a directory entry. It is comprised of all of the individual names of the parent entries back to the root entry of the directory information tree. See directory information tree (DIT)

domain

Any tree or subtree within the Domain Name System (DNS) namespace. Domain most commonly refers to a group of computers whose host names share a common suffix, the domain name.

Domain Name System (DNS)

A system for naming computers and network services that is organized into a hierarchy of domain**s**. DNS is used in TCP/IP networks to locate computers through user-friendly names. DNS resolves a friendly name into an IP address, which is understood by computers.

In Oracle Net Services, DNS translates the host name in a TCP/IP address into an IP address.

denial-of-service (DoS) attack

An attack that renders a Web site inaccessible or unusable. The denial-of-service attack can occur in many different ways but frequently includes attacks that cause the site to crash, reject connections, or perform too slowly to be usable. DoS attacks come in two forms:

- Basic denial-of-service attacks, which require only one or a few computers

- Distributed DoS attacks, which require many computers to execute

directly granted role

A role that has been granted directly to the user, as opposed to an indirectly granted role.

encrypted text

Text that has been encrypted, using an encryption algorithm; the output stream of an encryption process. On its face, it is not readable or decipherable, without first being subject to decryption. Also called ciphertext. Encrypted text ultimately originates as plaintext.

encryption

Disguising a message, rendering it unreadable to all but the intended recipient.

enterprise domain

A directory construct that consists of a group of databases and enterprise roles. A database should only exist in one enterprise domain at any time. Enterprise domains are different from Windows 2000 domains, which are collections of computers that share a common directory database.

Enterprise Domain Administrator

User authorized to manage a specific enterprise domain, including the authority to add new enterprise domain administrators.

enterprise role

Access privileges assigned to enterprise users. A set of Oracle role-based authorization**s** across one or more databases in an enterprise domain. Enterprise roles are stored in the directory and contain one or more global role**s**.

enterprise user

A user defined and managed in a directory. Each enterprise user has a unique identify across an enterprise.

entry

The building block of a directory, it contains information about an object of interest to directory users.

external authentication

Verification of a user identity by a third party authentication service, such as Kerberos or RADIUS.

Federal Information Processing Standard (FIPS)

A U.S. government standard that defines security requirements for cryptographic modules—employed within a security system protecting unclassified information within computer and telecommunication systems. Published by the National Institute of Standards and Technology (NIST).

FIPS

See Federal Information Processing Standard (FIPS).

forced cleanup

The ability to forcibly cleanup (that is, remove) all audit records from the database. To accomplish this, you set the `USE_LAST_ARCH_TIMESTAMP` argument of the `DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL` procedure to `FALSE`.

See also purge job.

forest

A group of one or more Active Directory trees that trust each other. All trees in a forest share a common schema, configuration, and global catalog. When a forest contains multiple trees, the trees do not form a contiguous namespace. All trees in a given forest trust each other through transitive bidirectional trust relationships.

Forwardable Ticket Granting Ticket

A special Kerberos ticket that can be forwarded to proxies, permitting the proxy to obtain additional Kerberos tickets on behalf of the client for proxy authentication.

See also Kerberos ticket.

global role

A role managed in a directory, but its privileges are contained within a single database. A global role is created in a database by using the following syntax:

```
CREATE ROLE role_name IDENTIFIED GLOBALLY;
```

global application context

A name-value pair that enables application context values to be accessible across database sessions.

See also application context.

grid computing

A computing architecture that coordinates large numbers of servers and storage to act as a single large computer. Oracle Grid Computing creates a flexible, on-demand computing resource for all enterprise computing needs. Applications running on the Oracle Database grid computing infrastructure can take advantage of common infrastructure services for failover, software provisioning, and management. Oracle Grid Computing analyzes demand for resources and adjusts supply accordingly.

HTTP

Hypertext Transfer Protocol: The set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. Relative to the TCP/IP suite of protocols (which are the basis for information exchange on the Internet), HTTP is an application protocol.

HTTPS

The use of Secure Sockets Layer (SSL) as a sublayer under the regular HTTP application layer.

indirectly granted role

A role granted to a user through another role that has already been granted to this user. Then you grant the `role2` and `role3` roles to the `role1` role. Roles `role2` and `role3` are now under `role1`. This means `psmith` has been indirectly granted the roles `role2` and `role3`, in addition to the direct grant of `role1`. Enabling the direct role1 for psmith enables the indirect roles role2 and role3 for this user as well.

identity

The combination of the public key and any other public information for an entity. The public information may include user identification data such as, for example, an e-mail address. A user certified as being the entity it claims to be.

identity management

The creation, management, and use of online, or digital, entities. Identity management involves securely managing the full life cycle of a digital identity from creation (provisioning of digital identities) to maintenance (enforcing organizational policies regarding access to electronic resources), and, finally, to termination.

identity management realm

A subtree in Oracle Internet Directory, including not only an Oracle Context, but also additional subtrees for users and groups, each of which are protected with access control lists.

initial ticket

In Kerberos authentication, an initial ticket or ticket granting ticket (TGT) identifies the user as having the right to ask for additional service tickets. No tickets can be obtained without an initial ticket. An initial ticket is retrieved by running the `okinit` program and providing a password.

instance

Every running Oracle database is associated with an Oracle instance. When a database is started on a database server (regardless of the type of computer), Oracle allocates a memory area called the System Global Area (SGA) and starts an Oracle process. This combination of the SGA and an Oracle process is called an instance. The memory and the process of an instance manage the associated database's data efficiently and serve the one or more users of the database.

integrity

A guarantee that the contents of a message received were not altered from the contents of the original message sent.

invoker's rights procedure

A procedure (or program unit) that executes with the privileges of the current user, that is, the user who invokes the procedure. These procedures are not bound to a particular schema. They can be run by a variety of users and allow multiple users to manage their own data by using centralized application logic. Invoker's rights procedures are created with the `AUTHID` clause in the declaration section of the procedure code.

For example, assume that user `blake` and user `scott` each have a table called `dept` in their respective user schemas. If user `blake` calls an invoker's rights procedure, which is owned by user `scott`, to update the `dept` table, then this procedure will update the `dept` table in the `blake` schema. This is because the procedure executes with the privileges of the user who invoked the procedure (that is, `blake`.).

See also definer's rights procedure.

java code obfuscation

Java code obfuscation is used to protect Java programs from reverse engineering. A special program (an obfuscator) is used to scramble Java symbols found in the code. The process leaves the original program structure intact, letting the program run correctly while changing the names of the classes, methods, and variables in order to hide the intended behavior. Although it is possible to decompile and read non-obfuscated Java code, the obfuscated Java code is sufficiently difficult to decompile to satisfy U.S. government export controls.

Java Database Connectivity (JDBC)

An industry-standard Java interface for connecting to a relational database from a Java program, defined by Sun Microsystems.

JDBC

See Java Database Connectivity (JDBC)

KDC

See Key Distribution Center (KDC).

Kerberos

A network authentication service developed under Massachusetts Institute of Technology's Project Athena that strengthens security in distributed environments. Kerberos is a trusted third-party authentication system that relies on shared secrets and assumes that the third party is secure. It provides single sign-on capabilities and database link authentication (MIT Kerberos only) for users, provides centralized password storage, and enhances PC security.

Kerberos ticket

A temporary set of electronic credentials that verify the identity of a client for a particular service. Also referred to as a service ticket.

Key Distribution Center (KDC)

In Kerberos authentication, the KDC maintains a list of user principals and is contacted through the `kinit` (`okinit` is the Oracle version) program for the user's initial ticket. Frequently, the KDC and the Ticket Granting Service are combined into the same entity and are simply referred to as the KDC. The Ticket Granting Service maintains a list of service principals and is contacted when a user wants to authenticate to a server providing such a service. The KDC is a trusted third party that must run on a secure host. It creates ticket-granting tickets and service tickets.

See also Kerberos ticket.

key pair

A public key and its associated private key. See public and private key pair.

keytab file

A Kerberos key table file containing one or more service keys. Hosts or services use *keytab* files in the same way as users use their passwords.

kinstance

An instantiation or location of a Kerberos authenticated service. This is an arbitrary string, but the host Computer name for a service is typically specified.

kservice

An arbitrary name of a Kerberos service object.

last archive timestamp

A timestamp that indicates the timestamp of the last archived audit record. For the database audit trail, this timestamp indicates the last audit record archived. For operating system audit files, it indicates the highest last modified timestamp property of the audit file that was archived. To set this timestamp, you use the `DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP` PL/SQL procedure.

See also purge job.

LDAP

See Lightweight Directory Access Protocol (LDAP)

ldap.ora file

A file created by Oracle Net Configuration Assistant that contains the following directory server access information:

- Type of directory server
- Location of the directory server
- Default identity management realm or Oracle Context (including ports) that the client or server will use

Lightweight Directory Access Protocol (LDAP)

A standard, extensible directory access protocol. It is a common language that LDAP clients and servers use to communicate. The framework of design conventions supporting industry-standard directory products, such as the Oracle Internet Directory.

listener

A process that resides on the server whose responsibility is to listen for incoming client connection requests and manage the traffic to the server.

Every time a client requests a network session with a server, a listener receives the actual request. If the client information matches the listener information, then the listener grants a connection to the server.

listener.ora file

A configuration file for the listener that identifies the:

- Listener name
- Protocol addresses that it is accepting connection requests on
- Services it is listening for

The `listener.ora` file typically resides in `$ORACLE_HOME/network/admin` on UNIX platforms and `ORACLE_BASE\ORACLE_HOME\network\admin` on Windows.

lightweight user session

A user session that contains only information pertinent to the application that the user is logging onto. The lightweight user session does not hold its own database resources, such as transactions and cursors; hence it is considered "lightweight." Lightweight user sessions consume far less system resources than traditional database session. Because lightweight user sessions consume much fewer server resources, a lightweight user session can be dedicated to each end user and can persist for as long as the application deems necessary.

local privilege grant

A privilege that applies only to the PDB in which it was granted.

See also common privilege grant.

local role

In a CDB, a role that exists only in a single PDB, just as a role in a non-CDB exists only in the non-CDB. Unlike a common role, a local role may only contain roles and privileges that apply within the container in which the role exists.

local user

In a CDB, any user that is not a common user.

man-in-the-middle

A security attack characterized by the third-party, surreptitious interception of a message, wherein the third-party, the *man-in-the-middle*, decrypts the message, reencrypts it (with or without alteration of the original message), and re-transmits it to the originally-intended recipient—all without the knowledge of the legitimate sender and receiver. This type of security attack works only in the absence of authentication.

mandatory auditing

Activities that are audited by default. Examples are modifications to unified audit trail policies (such as `ALTER AUDIT POLICY` statements) and top level statements by the administrative users `SYS`, `SYSDBA`, `SYSOPER`, `SYSASM`, `SYSBACKUP`, `SYSDG`, and `SYSKM`, until the database opens. See "Activities That Are Mandatorily Audited" for more information.

MD5

Message Digest 5. An algorithm that assures data integrity by generating a 128-bit cryptographic message digest value from given data. If as little as a single bit value in the data is modified, the MD5 checksum for the data changes. Forgery of data in a way that will cause MD5 to generate the same result as that for the original data is considered computationally infeasible.

message authentication code

Also known as data authentication code (DAC). A checksumming with the addition of a secret key. Only someone with the key can verify the cryptographic checksum.

message digest

See checksumming

CDB

See CDB.

namespace

In Oracle Database security, the name of an application context. You create this name in a `CREATE CONTEXT` statement.

naming method

The resolution method used by a client application to resolve a connect identifier to a connect descriptor when attempting to connect to a database service.

National Institute of Standards and Technology (NIST)

An agency within the U.S. Department of Commerce responsible for the development of security standards related to the design, acquisition, and implementation of cryptographic-based security systems within computer and telecommunication systems, operated by a Federal agency or by a contractor of a Federal agency or other organization that processes information on behalf of the Federal Government to accomplish a Federal function.

net service alias

An alternative name for a directory naming object in a directory server. A directory server stores net service aliases for any defined net service name or database service. A net service alias entry does not have connect descriptor information. Instead, it only references the location of the object for which it is an alias. When a client requests a directory lookup of a net service alias, the directory determines that the entry is a net service alias and completes the lookup as if it was actually the entry it is referencing.

net service name

A simple name for a service that resolves to a connect descriptor. Users initiate a connect request by passing a user name and password along with a net service name in a connect string for the service to which they want to connect:

```
CONNECT username@net_service_name
Enter password: password
```

Depending on your needs, net service names can be stored in a variety of places, including:

- Local configuration file, `tnsnames.ora`, on each client
- Directory server
- External naming service, such as NIS

network authentication service

A means for authenticating clients to servers, servers to servers, and users to both clients and servers in distributed environments. A network authentication service is a repository for storing information about users and the services on different servers to which they have access, as well as information about clients and servers on the network. An authentication server can be a physically separate computer, or it can be a facility co-located on another server within the system. To ensure availability, some authentication services may be replicated to avoid a single point of failure.

network listener

A listener on a server that listens for connection requests for one or more databases on one or more protocols. See listener.

NIST

See National Institute of Standards and Technology (NIST).

non-CDB

An Oracle database that is not a CDB.

non-repudiation

Incontestable proof of the origin, delivery, submission, or transmission of a message.

obfuscation

A process by which information is scrambled into a non-readable form, such that it is extremely difficult to de-scramble if the algorithm used for scrambling is not known.

obfuscator

A special program used to obfuscate Java source code. See obfuscation.

object class

A named group of attribute**s**. When you want to assign attributes to an entry, you do so by assigning to that entry the object classes that hold those attributes. All objects associated with the same object class share the same attributes.

Oracle Context

1. An entry in an LDAP-compliant internet directory called `cn=OracleContext`, under which all Oracle software relevant information is kept, including entries for Oracle Net Services directory naming and checksumming security.

There can be one or more Oracle Contexts in a directory. An Oracle Context is usually located in an identity management realm.

Oracle Virtual Private Database

A set of features that enables you to create security policies to control database access at the row and column level. Essentially, Oracle Virtual Private Database adds a dynamic `WHERE` clause to a SQL statement that is issued against the table, view, or synonym to which an Oracle Virtual Private Database security policy was applied.

Oracle Net Services

An Oracle product that enables two or more computers that run the Oracle server or Oracle tools such as Designer/2000 to exchange data through a third-party network. Oracle Net Services support distributed processing and distributed database capability. Oracle Net Services is an open system because it is independent of the communication protocol, and users can interface Oracle Net to many network environments.

Oracle PKI certificate usages

Defines Oracle application types that a certificate supports.

Password-Accessible Domains List

A group of enterprise domain**s** configured to accept connections from password-authenticated users.

PCMCIA cards

Small credit card-sized computing devices that comply with the Personal Computer Memory Card International Association (PCMCIA) standard. These devices, also called PC cards, are used for adding memory, modems, or as hardware security modules. PCMCIA cards that are used as hardware security modules securely store the private key component of a public and private key pair and some also perform the cryptographic operations as well.

PDB

An individual database that is part of a CDB.

See also root.

peer identity

SSL connect sessions are between a particular client and a particular server. The identity of the peer may have been established as part of session setup. Peers are identified by X.509 certificate chain**s**.

PEM

The Internet Privacy-Enhanced Mail protocols standard, adopted by the Internet Architecture Board to provide secure electronic mail over the Internet. The PEM protocols provide for encryption, authentication, message integrity, and key management. PEM is an inclusive standard, intended to be compatible with a wide range of key-management approaches, including both symmetric and public-key schemes to encrypt data-encrypting keys. The specifications for PEM come from four Internet Engineering Task Force (IETF) documents: RFCs 1421, 1422, 1423, and 1424.

PKCS #10

An RSA Security, Inc., Public-Key Cryptography Standards (PKCS) specification that describes a syntax for certification requests. A certification request consists of a distinguished name, a public key, and optionally a set of attributes, collectively signed by the entity requesting certification. Certification requests are referred to as certificate requests in this manual. See certificate request

PKCS #11

An RSA Security, Inc., Public-Key Cryptography Standards (PKCS) specification that defines an application programming interface (API), called Cryptoki, to devices which hold cryptographic information and perform cryptographic operations. See PCMCIA cards

PKCS #12

An RSA Security, Inc., Public-Key Cryptography Standards (PKCS) specification that describes a transfer syntax for storing and transferring personal authentication credentials—typically in a format called a wallet.

PKI

See public key infrastructure (PKI)

plaintext

Message text that has not been encrypted.

pluggable database

See PDB.

principal

A string that uniquely identifies a client or server to which a set of Kerberos credentials is assigned. It generally has three parts: `kservice/kinstance@REALM`. In the case of a user, `kservice` is the user name. See also kservice, kinstance, and realm

private key

In public-key cryptography, this key is the secret key. It is primarily used for decryption but is also used for encryption with digital signatures. See public and private key pair.

proxy authentication

A process typically employed in an environment with a middle tier such as a firewall, wherein the end user authenticates to the middle tier, which thence authenticates to the directory on the user's behalf—as its *proxy*. The middle tier logs into the directory as a *proxy user*. A proxy user can switch identities and, once logged into the directory, switch to the end user's identity. It can perform operations on the end user's behalf, using the authorization appropriate to that particular end user.

public key

In public-key cryptography, this key is made public to all. It is primarily used for encryption but can be used for verifying signatures. See public and private key pair.

public and private key pair

A set of two numbers used for encryption and decryption, where one is called the private key and the other is called the public key. Public keys are typically made widely available, while private keys are held by their respective owners. Though mathematically related, it is generally viewed as computationally infeasible to derive the private key from the public key. Public and private keys are used only with asymmetric encryption algorithms, also called public-key encryption algorithms, or public-key cryptosystems. Data encrypted with either a public key or a private key from a key pair can be decrypted with its associated key from the key-pair. However, data encrypted with a public key cannot be decrypted with the same public key, and data enwrapped with a private key cannot be decrypted with the same private key.

public key infrastructure (PKI)

Information security technology utilizing the principles of public key cryptography. Public key cryptography involves encrypting and decrypting information using a shared public and private key pair. Provides for secure, private communications within a public network.

PUBLIC role

A special role that every database account automatically has. By default, it has no privileges assigned to it, but it does have grants to many Java objects. You cannot drop the `PUBLIC` role, and a manual grant or revoke of this role has no meaning, because the user account will always assume this role. Because all database user accounts assume the `PUBLIC` role, it does not appear in the `DBA_ROLES` and `SESSION_ROLES` data dictionary views.

purge job

A database job created by the `DBMS_AUDIT_MGMT.CREATE_PURGE_JOB` procedure, which manages the deletion of the audit trail. A database administrator schedules, enables, and disables the purge job. When the purge job becomes active, it deletes audit records from the database audit tables, or it deletes Oracle Database operating system audit files.

See also forced cleanup, last archive timestamp.

RADIUS

Remote Authentication Dial-In User Service (RADIUS) is a client/server protocol and software that enables remote access servers to communicate with a central server to authenticate dial-in users and authorize their access to the requested system or service.

realm

1. Short for identity management realm. 2. A Kerberos object. A set of clients and servers operating under a single key distribution center/ticket-granting service (KDC/TGS). Services (see kservice) in different realms that share the same name are unique.

realm Oracle Context

An Oracle Context that is part of an identity management realm in Oracle Internet Directory.

registry

A Windows repository that stores configuration information for a computer.

remote computer

A computer on a network other than the local computer.

role

A named group of related privileges that you grant as a group to users or other roles.

See also indirectly granted role.

root

In a multitenant environment, a collection of Oracle-supplied and user-created schemas to which all PDBs belong. The container database has only one root. Each PDB is considered to be a child of this root. Root has an entry in its data dictionary that indicates the existence of each PDB.

See also container, CDB, PDB.

root key certificate

See trusted certificate

salt

In cryptography, a way to strengthen the security of encrypted data. Salt is a random string that is added to the data before it is encrypted, making it more difficult for attackers to steal the data by matching patterns of ciphertext to known ciphertext samples. Salt is often also added to passwords, before the passwords are encrypted, to avoid dictionary attacks, a method that unethical hackers (attackers) use to steal passwords. The encrypted salted values make it difficult for attackers to match the hash value of encrypted passwords (sometimes called verifiers) with their dictionary lists of common password hash values.

schema

1. Database schema: A named collection of objects, such as tables, views, clusters, procedures, packages, attributes, object class**es**, and their corresponding matching rules, which are associated with a particular user. 2. LDAP directory schema: The collection of attributes, object classes, and their corresponding matching rules.

schema mapping

See user-schema mapping

secure application role

A database role that is granted to application users, but secured by using an invoker's right stored procedure to retrieve the role password from a database table. A secure application role password is not embedded in the application.

See also application role.

Secure Hash Algorithm (SHA)

An algorithm that assures data integrity by generating a 160-bit cryptographic message digest value from given data. If as little as a single bit in the data is modified, the Secure Hash Algorithm checksum for the data changes. Forgery of a given data set in a way that will cause the Secure Hash Algorithm to generate the same result as that for the original data is considered computationally infeasible.

An algorithm that takes a message of less than 264 bits in length and produces a 160-bit message digest. The algorithm is slightly slower than MD5, but the larger message digest makes it more secure against brute-force collision and inversion attacks.

Secure Sockets Layer (SSL)

An industry standard protocol designed by Netscape Communications Corporation for securing network connections. SSL provides authentication, encryption, and data integrity using public key infrastructure (PKI).

The Transport Layer Security (TLS) protocol is the successor to the SSL protocol.

separation of duty

Restricting activities only to those users who must perform them. For example, you should not grant the `SYSDBA` administrative privilege to any user. Only grant this privilege to administrative users. Separation of duty is required by many compliance policies. See "Guidelines for Securing User Accounts and Privileges" for guidelines on granting privileges to the correct users.

server

A provider of a service.

service

1. A network resource used by clients; for example, an Oracle database server.

2. An executable process installed in the Windows registry and administered by Windows. Once a service is created and started, it can run even when no user is logged on to the computer.

service name

For Kerberos-based authentication, the kservice portion of a service principal.

service principal

See principal

service key table

In Kerberos authentication, a service key table is a list of service principals that exist on a kinstance. This information must be extracted from Kerberos and copied to the Oracle server computer before Kerberos can be used by Oracle.

service ticket

A service ticket is trusted information used to authenticate the client, to a specific service or server, for a predetermined period of time. It is obtained from the KDC using the initial ticket. See also Kerberos ticket.

session key

A key shared by at least two parties (usually a client and a server) that is used for data encryption for the duration of a single communication session. Session keys are typically used to encrypt network traffic; a client and a server can negotiate a session key at the beginning of a session, and that key is used to encrypt all network traffic between the parties for that session. If the client and server communicate again in a new session, they negotiate a new session key.

session layer

A network layer that provides the services needed by the presentation layer entities that enable them to organize and synchronize their dialogue and manage their data exchange. This layer establishes, manages, and terminates network sessions between the client and server. An example of a session layer is Network Session.

SHA

See Secure Hash Algorithm (SHA).

shared schema

A database or application schema that can be used by multiple enterprise users. Oracle Database supports the mapping of multiple enterprise users to the same shared schema on a database, which lets an administrator avoid creating an account for each user in every database. Instead, the administrator can create a user in one location, the enterprise directory, and map the user to a shared schema that other enterprise users can also map to. Sometimes called user/schema separation.

single key-pair wallet

A PKCS #12-format wallet that contains a single user certificate and its associated private key. The public key is imbedded in the certificate.

single password authentication

The ability of a user to authenticate with multiple databases by using a single password. In the Oracle Database implementation, the password is stored in an LDAP-compliant directory and protected with encryption and Access Control Lists.

single sign-on (SSO)

The ability of a user to *authenticate once,* combined with strong authentication occurring transparently in subsequent connections to other databases or applications. Single sign-on lets a user access multiple accounts and applications with a single password, entered during a single connection. *Single password, single authentication.* Oracle Database supports Kerberos and SSL-based single sign-on.

smart card

A plastic card (like a credit card) with an embedded integrated circuit for storing information, including such information as user names and passwords, and also for performing computations associated with authentication exchanges. A smart card is read by a hardware device at any client or server.

A smartcard can generate random numbers which can be used as one-time use passwords. In this case, smartcards are synchronized with a service on the server so that the server expects the same password generated by the smart card.

sniffer

Device used to surreptitiously listen to or capture private data traffic from a network.

SSO

See single sign-on (SSO)

System Global Area (SGA)

A group of shared memory structures that contain data and control information for an Oracle instance.

system identifier (SID)

A unique name for an Oracle instance. To switch between Oracle databases, users must specify the desired SID. The SID is included in the `CONNECT DATA` parts of the connect descriptor in a tnsnames.ora file, and in the definition of the network listener in a listener.ora file.

ticket

A piece of information that helps identify who the owner is. See initial ticket and service ticket.

tnsnames.ora

A file that contains connect descriptors; each connect descriptor is mapped to a net service name. The file may be maintained centrally or locally, for use by all or individual clients. This file typically resides in the following locations depending on your platform:

- (UNIX) `ORACLE_HOME/network/admin`
- (Windows) `ORACLE_BASE\ORACLE_HOME\network\admin`

token card

A device for providing improved ease-of-use for users through several different mechanisms. Some token cards offer one-time passwords that are synchronized with an authentication service. The server can verify the password provided by the token card at any given time by contacting the authentication service. Other token cards operate on a challenge-response basis. In this case, the server offers a challenge (a number) which the user types into the token card. The token card then provides another number (cryptographically-derived from the challenge), which the user then offers to the server.

transport layer

A networking layer that maintains end-to-end reliability through data flow control and error recovery methods. Oracle Net Services uses *Oracle protocol supports* for the transport layer.

Transport Layer Security (TLS)

An industry standard protocol for securing network connections. The TLS protocol is a successor to the SSL protocol. It provides authentication, encryption, and data integrity using public key infrastructure (PKI). The TLS protocol is developed by the Internet Engineering Task Force (IETF).

trusted certificate

A trusted certificate, sometimes called a root key certificate, is a third party identity that is qualified with a level of trust. The trusted certificate is used when an identity is being validated as the entity it claims to be. Typically, the certificate authorities you trust are called trusted certificates. If there are several levels of trusted certificates, a trusted certificate at a lower level in the certificate chain does not need to have all its higher level certificates reverified.

trusted certificate authority

See certificate authority.

trust point

See trusted certificate.

user name

A name that can connect to and access objects in a database.

user-schema mapping

An LDAP directory entry that contains a pair of values: the base in the directory at which users exist, and the name of the database schema to which they are mapped. The users referenced in the mapping are connected to the specified schema when they connect to the database. User-schema mapping entries can apply only to one database or they can apply to all databases in a domain. See shared schema.

user/schema separation

See shared schema.

user search base

The node in the LDAP directory under which the user resides.

views

Selective presentations of one or more tables (or other views), showing both their structure and their data.

wallet

A data structure used to store and manage security credentials for an individual entity.

Windows native authentication

An authentication method that enables a client single login access to a Windows server and a database running on that server.

X.509

An industry-standard specification for digital certificate **s**.

Index