
CouchbaseConnect15
AT LEVI'S STADIUM | JUN 2 - 4, 2015
NoSQL
NoLIMITS
Choose from 30+ developer sessions led by the architects and engineers behind our Couchbase products.
See how SQL for Documents, our new SQL-based query language, enables faster, easier development of apps with complex queries.
Register now:
couchbase.com/connect15
Couchbase

NoSQL and Data Scalability 2.0

By Eugene Ciurana

This Refcard provides an introduction to basic NoSQL and Data Scalability terminology and techniques and exhibits in-depth examples of popular NoSQL technologies, including architectures, common uses, and more.

NoSQL and Data Scalability 2.0 demystifies the latest techniques in high-volume data storage, search, and management by explaining how they work and when to apply them.

SCALABLE DATA ARCHITECTURES

Scalable data architectures have evolved to improve overall system efficiency and reduce operational costs. Specific NoSQL databases may have different topological requirements, but the general architecture is the same.

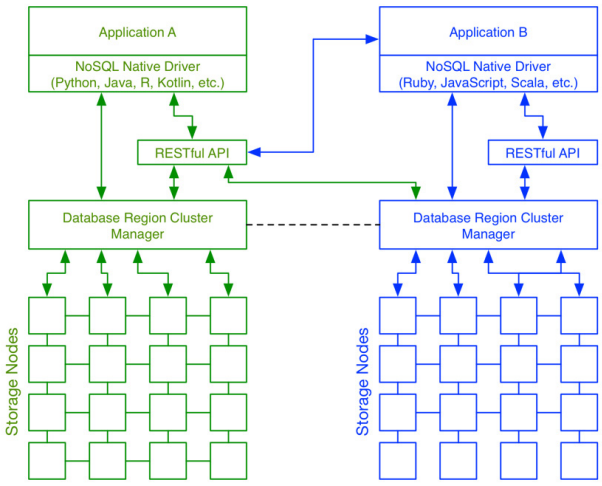


Figure 1: *NoSQL Architecture*

In general, NoSQL architectures offer:

- Eventual consistency
- High availability with partition tolerance
- Horizontal scalability (optimized for large volume of reads and queries)
- Cloud readiness
- Distributed, structured data storage

Cloud readiness describes the database being used as a service and the ability to deploy the storage grid and cluster manager to a cloud provider.

NOSQL

NoSQL describes a horizontally scalable, non-relational database with built-in replication support. Applications interact with the database through a simple API, and the data is stored in a schema-free repository as large files or data blocks. The repository is often a custom file system designed to support NoSQL operations with high replication.

NOSQL DATABASES CLASSIFICATION

DATABASE TYPE	USES
Document	Documents, semi-structured data
Column	Read/write raw time series data
Graph	Named entities, semantic queries, associative data sets
Key-value	Key-value pair, where the values can be complex and mixed data structures (e.g. a document)
Multi-model	Two or more database types, including relational databases and the types listed above, with a common database manager for all

While all database types are in common use, document stores are most often associated with NoSQL systems due to their pervasiveness in web and mobile content handling applications.

IS NOSQL FOR YOU?

DOES YOUR APP DESIGN...
☐ Require high-speed throughput?
☐ Need to handle high volumes of data?
☐ Work well with weak data consistency?
☐ Benefit from direct object-database entity mapping?
☐ Have to be Always On?

If you checked off four or more items from the list, then NoSQL is a good fit for you.

Always On just means that users will have access to complete app functionality at all times. In mobile app and gaming contexts, it can mean access to data that is "a bit behind" the effective system state (i.e. eventual consistency is acceptable).

NOSQL PERFORMANCE AND TCO COMPARISON

Total cost of ownership (TCO) depends on functionality and complexity. A higher TCO may be acceptable when performance (throughput or scalability) is a primary concern.

	Document	Graph	Column	Key:Value
Performance	high	variable	high	high
Scalability	high	variable	high	high
Complexity	low	high	low	none
Flexibility	high	high	so-so	high

(source: Ben Scoffeld, Viget Labs)

Figure 2: *Complexity and TCO*

Document and key-value stores are most popular because of their ease of use, flexibility, and applicability across many problem domains—at a reasonable TCO.

Tip: Graph databases are excellent replacements for complex relational models because relationships between entities (or graph edges) are more efficient and better suited for high-performance applications than using explicit joins and foreign-keys.

WHICH DATA STORE MODEL TO USE?

The flowchart in *Figure 3* describes how to choose the most appropriate database or store for the application.

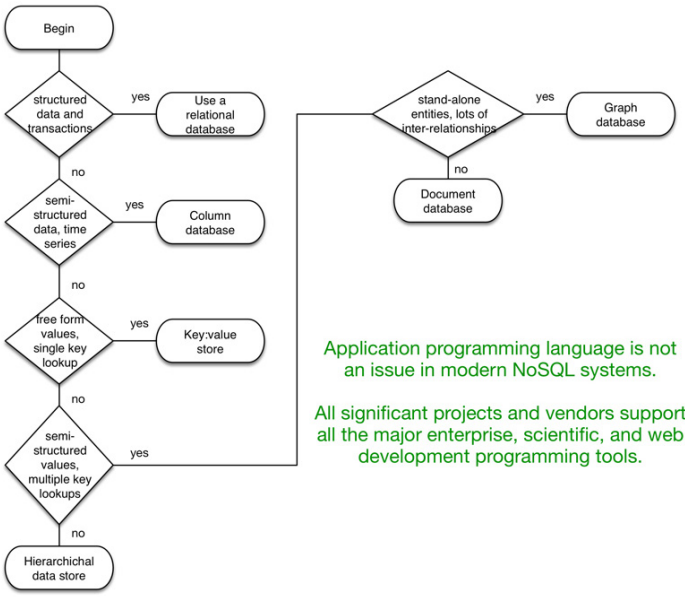


Figure 3: *Choosing the Right Data Store*

CLOUD DATABASES

Demand-based scaling is an attractive proposition for running NoSQL systems on the cloud; it maximizes the advantages of running the application on cloud-based providers like AWS, Azure, or Google Cloud Computing.

- **Database-as-a-Service** (DBaaS) offers turnkey managed functionality, which delegates all operational responsibilities to the provider.
- **Hosted VM** databases are provisioned on virtual images, much like they would be on premises, and all operational responsibility belongs to the user.

	DATABASE	PRIMARY MODEL
DB as a service	Amazon Dynamo	Key-value
	Amazon SimpleDB	Column
	Google Data Store	Column
	IBM Cloudant	Document
	Microsoft DocumentDB	Document
Hosted VM DB	MongoDB	Document
	Couchbase Server	Document
	Neo4j	Graph
	Apache Cassandra	Column

Tip: Billing overruns are very easy when using a Database-as-a-service. Engage a usage/cost monitoring system to help manage expenses and to avoid nasty surprises.

VERY HIGH-VOLUME DATA STORES

Many applications require the storage of very large binary data sets. Traditional data stores can't handle them because their size makes it impractical. Enter the High-Volume Data Store (HVDS).

Most very high-volume data applications are used in scientific or financial problem domains. These applications rely on dedicated, optimized binary data formats that allow quick access, manipulation, and data format description within a single scope.

HIGH-VOLUME DATA CHARACTERISTICS

- Very large data sets
- Multidimensional distribution
- Most data is numerical
- Batch processing (including Hadoop) is optimal
- Strong typing

HVDS CHARACTERISTICS

- Provided data structures (atoms, groups, arrays)
- Persistence management
- Self-describing mechanisms to store the data, obviating versioning issues
- ACID (Atomicity, Consistency, Isolation, Durability) properties
- Language independence
- No query language—access to structures is application and language/API dependent
- No security model
- Application or access APIs must provide concurrency

Rule of Thumb: High-Volume Data Stores handle very few I/O operations, but each consists of very large amounts of data; NoSQL handles lots of I/O operations on small amounts of data.

Data is stored in HVDS prior to initial processing, where the HVDS API provides more efficient access than the file system or a database system. The intermediate or final results move to NoSQL or relational stores for end-user reporting and manipulation.

HVDS maps onto local files, like Hadoop's HDFS, allowing volumes to move between files systems (e.g. HFS+ to NTFS) without issues. Most HVDS originates in scientific research organizations, and portability is a primary design concern.

HVDS CLASSIFICATION

HVDS	USES
HDF5	Astronomy, genetics, finance
CDF	Astronomy, aeronautics
FITS	Astronomy
GRIB	Meteorology
Protocol Buffers	Network communication
RData	Finance, genetics, statistics, energy

HVDS WORKFLOW

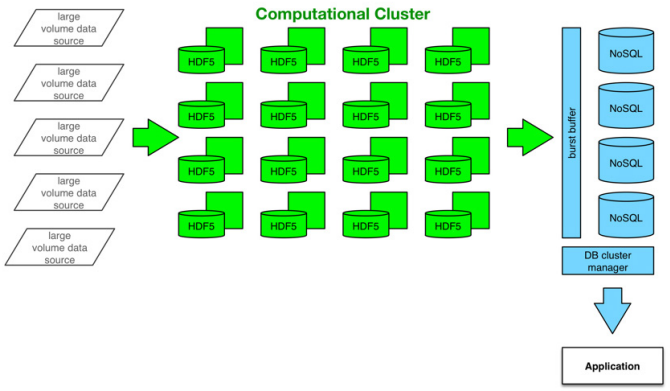


Figure 4: *High Volume Data Store Workflow*

DOCUMENT DATABASE: COUCHBASE SERVER

Couchbase Server is a document-based database that bridges the gaps between scalable key-value stores, relational database querying, and robustness capabilities. Its characteristics include:

- **Document-oriented storage** – data is manipulated as JSON documents
- **Querying** – uses a robust query language with document handling semantics, N1QL
- **Multi-dimensional scalability (MDS)** – different components may scale up or out, depending on load and performance required
- **MapReduce** – built-in, indispensable for querying on non-indexed document attributes
- **Caching** – integrated across all database services
- **Replication** – transparent replication across multiple data centers
- **Spatial views** – handles geometric, geospatial data definitions and allows mapping of attributes to multidimensional indices (e.g. table-like)

Couchbase Server provides datacenter consistency and partition tolerance. The database is based on the independent scaling and replication model shown in *Figure 5*. Data is handled across 3 different service zones: indexing, querying, and data.

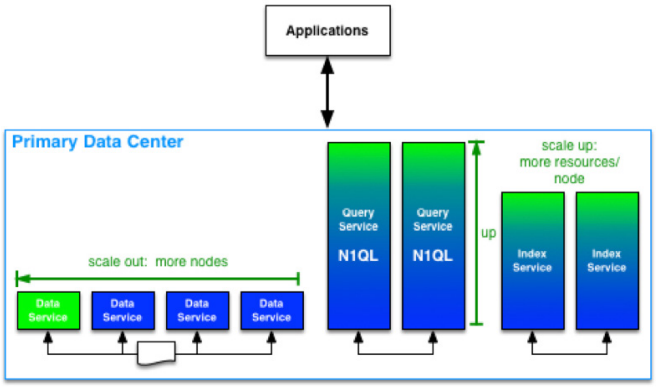


Figure 5: *Couchbase Server MDS Cluster*

The service zones have different scalability requirements according to their function.

SERVICE	FUNCTION
Data	Core operations, MapReduce
Query	N1QL execution engine with index and data service sync
Indexing	Global secondary index partitions and query fulfillment

The Couchbase Server software and general documentation is at http://docs.couchbase.com/admin/admin/Couchbase-intro.html

Each database service node can replicate data to its peer, and each cluster can replicate to other clusters. Couchbase Server provides facilities for cross datacenter replication (XDCR), simplifying disaster recovery, high availability, and data locality scenarios.

IN-MEMORY CACHE == HIGHER PERFORMANCE

Couchbase Server performs very well during writes because it uses a memory-first mechanism. Data is written to the in-memory cache with a fast response to the caller. Couchbase Server asynchronously replicates the data to other nodes or clusters, updates the indices, and persists the data to disk. Database clients may override any of these operations to make them synchronous.

A read request is guaranteed to always get the most recent result at the time of the beginning of the request.

DOCUMENT FORMAT

Couchbase Server handles JSON documents. Being a schemaless database, any valid document can be committed to the database. Couchbase Server assigns two additional attributes to each document upon creation for tracking the document's unique ID (_id) and revision number (_rev). These attributes are required for all operations other than creation. A typical document and its cross-language representation could be:

```
{
  "type" : "Person",
  "name" : "Tom",
  "age" : 42
}
```

LANGUAGE	REPRESENTATION
Python	`{ u"type" : u"Person", u"name" : u"Tom", u"age" : 42 }`
Ruby	`{ "type" => "Person", "name" => "Tom", "age" => 42 }`
PHP	`array( "type" => "Person", "name" => "Tom", "age" => 42);`

Dynamic languages offer a closer object mapping to JSON than compiled languages.

Tip: "Type" is just a JSON attribute in this example, like any other. It's good practice to define a document type to simplify queries, but it isn't compulsory.

VIEWS

Views are the primary query and reporting tool in Couchbase Server. A view is just a JavaScript function that maps view keys to values. Views are stored on the server and used when needed. They are only updated upon request (query or report), not upon document creation or updates. For example, in a database that contains **Person** and **Animal** objects, a view for listing all the instances of "Person" could be:

```
function (d) {  // d ::= document
  if (d.type == "Person")
    emit(d.name, { d.name, d.age });
}
```

Listing 1: *View-Based Query*

The output will be something like this:

```
{ "total_rows": 1, "offset": 0, "rows":
    [ { "id": "6921", "key": "Tom",
            "value": {
            "name": "Tom",
            "age": 42 } } ] }
```

View operations are defined in terms of MapReduce techniques.

STREAM-BASED VIEWS

Couchbase Server also introduced stream-based views based on the Data Change Protocol (DCP). A stream-based view submits the query to the managed cache. The managed cache asynchronously updates the disk queue, the actual disk, or replicates the query to another node.

View queries may include the `stale` data freshness flag with one of these settings:

- `false` – waits for the indexer to commit changes corresponding to the current key-value document set before returning the latest entries from the view index
  - responses may include up-to-date latest results that haven't been committed to disk
- `update_after` – returns the current index entries, then initiate an index update
- `ok` – returns the current entries from the index

A configurable, automatic process updates the indices at configurable intervals based on whether changes within a threshold have occurred.

SPATIAL VIEWS

Couchbase Server enables the creation of multi-dimensional spatial indices containing geometry data that can express information based on geometries within a multidimensional range. Some examples include:

- Geographical data for mobile, map, or other geo-location applications
- Geometrical data for statistical analysis and data science applications
- Arbitrary multidimensional collections for handling tables in a manner similar to how relational and column databases map structured and semi-structured data to schemas

The spatial views reference covering geospatial and arbitrary data collections information is available from http://docs.couchbase.com/4.0/admin/Views/spatial-views.html

N1QL – A SQL-LIKE LANGUAGE FOR DOCUMENTS

While views are powerful, they are somewhat cumbersome to manage. Couchbase Server introduced N1QL (pronounced "nickel") to ease integration with legacy reporting systems and to assist programmers in unleashing more efficient, maintainable, and robust queries. Its main features include:

- JSON attributes string concatenation and matching in `SELECT` statements
- Ability to remove duplicate but valid results through the `DISTINCT` keyword (e.g. `COUNT (DISTINCT someID)`)
- Semantics for handling document missing values—a valid NoSQL construct—to test conditions similar to IS `NULL` in SQL
- Dot-notation and `ARRAY` semantics for addressing JSON document attributes in query results
- `JOIN`, `NEST`, and `UNNEST` capabilities specific to handling variable column JSON result sets

The N1QL example in *Figure 6* shows the language's flexibility in dealing with schemaless documents:

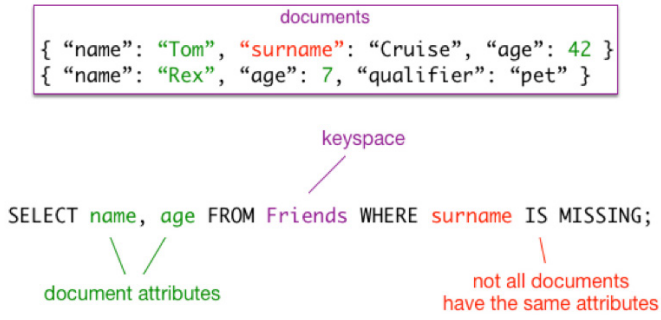


Figure 6: *N1QL and JSON Data*

Check out a Couchbase Server N1QL quick reference at http://query.couchbase.com

COUCHBASE SERVER COMMON APPLICATIONS

- **Caching** - more robust capabilities, indexing, and persistence with built-in replication and HA
- **Real-time analytics** - RDBMS may be too expensive or slow to run in comparison
- **Content management systems** - JSON objects can represent any kind of document, including those with a binary representation
- **Named entity index** – for scalable semantic search and machine learning applications

COUCHBASE SERVER DRAWBACKS

- **Complex queries** - some complex queries and indices that require pivot tables are better suited for SQL or graph databases than for N1QL or Views
- **No full text search**
- **No built-in collections or tables** – ad hoc attribute definitions are used for logical groupings; spatial views and new N1QL features help in table creation but are harder than native collection/table support in other database technologies

GRAPH DATABASE: NEO4J

Neo4j is an embeddable database with transactional capabilities that stores data in graphs. Entities are stored as graph nodes, and relationships between nodes are stored as edges connecting them. Its main features include:

- Relationships and nodes have the same priority during searches
- Relationships are first class objects, not compound constructs like joins, and look up tables in other databases
- High performance through memory mapping of entities and indices
- ACID and transactional integrity
- Cypher, an expressive and efficient declarative language for querying the database

The Neo4j downloads and documentation are available from: http://neo4j.com.

Figure 7 shows how Neo4j may be embedded in a JVM-based application, where Neo4j exposes a set of Java packages to make direct calls to the database.

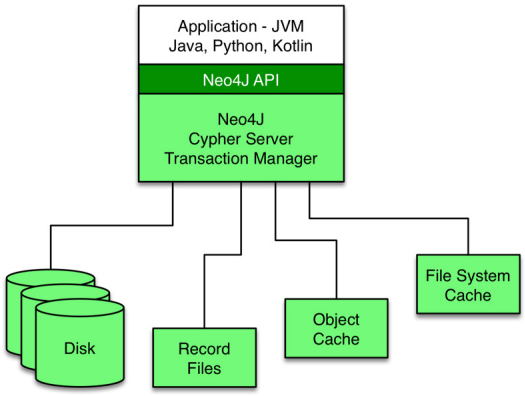


Figure 7: *Embedded Neo4j*

The Neo4j stand-alone configuration in *Figure 8* exposes a RESTful API and runs on dedicated servers to optimize memory usage since objects and indices are memory-mapped.

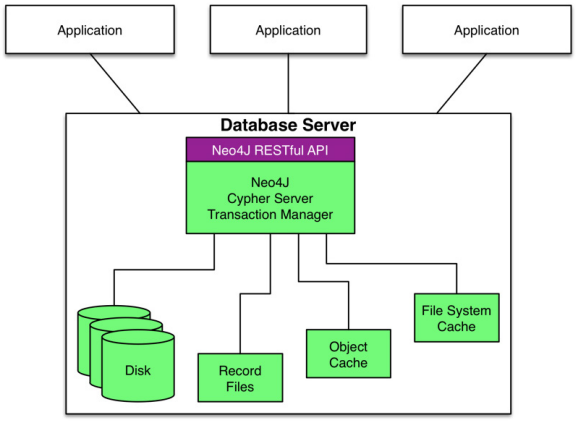


Figure 8: *Stand-Alone Neo4j*

CACHING

Neo4j excels at content delivery and query speed because it offers two different caches:

- **Low-level** – file system cache for data stored in the actual medium; it's configured to assume that the database runs in a dedicated server
- **Objects** – cache of individual nodes and relationships optimized for quick graph traversal

Both caches have a number of configuration options; consult the Neo4j web documentation for details.

CORE: PROPERTY GRAPH

The property graph is made up of nodes, relationships, and properties. A graph database manages all the storage and searching aspects of property graph traversal.

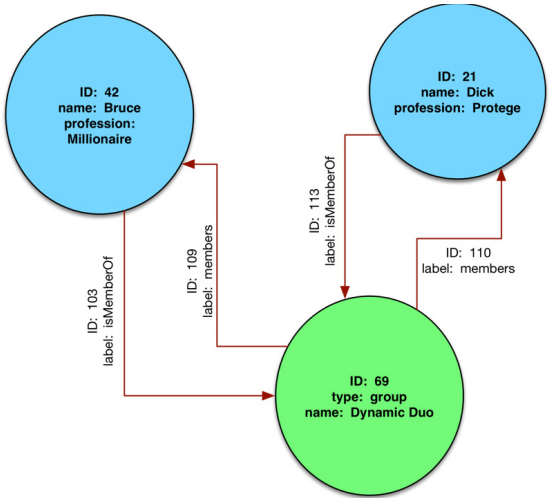


Figure 9: *Property Graph*

- Nodes are aggregations of properties
- Properties are arbitrary key-value pairs
  - Keys must be strings
  - Values can be any of any type
- Relationships connect nodes in a directed graph with a start and an end node:
  - No dangling relationships allowed!
  - Relationships also have properties, often in the form of metadata that aids in graph manipulation and defining run-time query constraints

Source: Graph Databases, Robinson, Webber, & Eifrem, O'Reilly, 2014

QUERYING NEO4J WITH CYPHER

Cypher is a declarative query language specific to Neo4j that describes database operation patterns. It's loosely based on SQL, though it features ASCII text constructs to represent patterns and directionality.

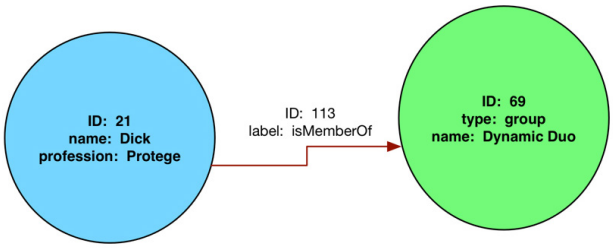


Figure 10: *Cypher Example*

Cypher enables users to describe what to create, update, delete, or select from the graph without requiring an explicit description of how to do it. It describes the nodes, attributes, and relationships in the property graph.

- Parentheses encapsulate a node

- Operators show directionality arrows and attributes
- SQL-like statements describe database operations

Check out the DZone Refcard *Querying Graphs with Neo4j* for more details on writing queries with Cypher, available at http://refcardz.dzone.com/refcardz/querying-graphs-neo4j.

CYPHER AND JAVA

- Cypher queries can be executed as payloads in RESTful calls or through the various native language wrappers
- Database operations are made more efficient if Neo4j is embedded mode if the program make direct calls the Neo4j Java API, bypassing the RESTful API and Cypher; all Cypher operations have counterparts in the Java API

COMMON USE CASES

- **Recommendations** – e-commerce, social media, entertainment
- **Route optimization** – actual geographical routes or operations research algorithms
- **Logistics** – B2B, disaster management
- **Authorization and access control** – better capabilities than traditional LDAP and other ACL
- **Finance** – stock analysis, historical performance analysis
- **Named entity analysis** – semantic web, fraud detection, natural language processing

NEO4J DRAWBACKS

- Poor horizontal scalability for load distribution
- Cypher lacks end-user exact traversal patterns definition
- Lack of support for composite keys
- No auto-sharding – users must model the property graphs, databases, and server allocations manually

STAYING CURRENT

Do you want to know about specific projects and use cases where NoSQL and data scalability are the hot topics? Follow the author's data science and scalability feed: http://twitter.com/ciurana

PUBLICATIONS

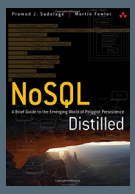
By Eugene Ciurana

- DZone Refcard #105: *NoSQL and Data Scalability*
- DZone Refcard #43: *Scalability and High Availability*
- DZone Refcard #128: *Apache Hadoop Deployment*
- DZone Refcard #117: *Getting Started with Apache Hadoop*
- *Developing with Google App Engine*, Apress
- DZone Refcard #38: *SOA Patterns*
- *The Tesla Testament: A Thriller*, CIMEntertainment

ABOUT THE AUTHOR

Eugene Ciurana is an open source evangelist and entrepreneur who specializes in the design and implementation of mission-critical, high availability systems. As CTO of Summly, he led the team that built one of the highest throughput natural language and automatic summarization systems in the world, leveraging NoSQL technologies like document and graph databases. Eugene is the founder and CEO of Cosmify, Inc., where his team is democratizing high-volume unstructured data analysis by humanizing machine learning, knowledge discovery, and data visualization—doing away with the need for Big Data shamans and expensive infrastructure.

RECOMMENDED BOOK



NoSQL Distilled by Martin Fowler and Pramod Sadalage offers a glimpse into the universe of non-relational database systems. This book begins by introducing the reader to high-level concepts like CAP theorem before diving into specific databases and use cases. It includes information on such databases as MongoDB, Cassandra, and Neo4j, offering a wide level of coverage within the NoSQL space.

BUY NOW

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795
9 781936 502776

VERSION 1.0 $7.95