Oracle

NoSQL Database Concepts Manual

12c Release 1

Library Version 12.1.4.3

ORACLE®
NOSQL DATABASE

Legal Notice

Published 17-Feb-2017

Table of Contents

Preface

This document introduces Oracle NoSQL Database.

This book is aimed at technical users, primarily database administrators and developers who are new to Oracle NoSQL Database.

Conventions Used in This Book

The following typographical conventions are used within this manual:

Information that you are to type literally is presented in `monospaced` font.

Variable or non-literal text is presented in *italics*. For example: "Go to your *KVHOME* directory."

Note

Finally, notes of special interest are represented using a note block such as this.

Chapter 1. Introduction to Oracle NoSQL Database

Welcome to Oracle NoSQL Database. Oracle NoSQL Database provides multi-terabyte distributed key/value pair storage that offers scalable throughput and performance. That is, it services network requests to store and retrieve data which is accessed as tables of information or, optionally, as key-value pairs. Oracle NoSQL Database services these types of data requests with a latency, throughput, and data consistency that is predictable based on how the store is configured.

Oracle NoSQL Database uses Oracle Berkeley DB Java Edition as its underlying storage engine. For more information about Oracle Berkeley DB Java Edition, start here: http://www.oracle.com/technetwork/database/berkeleydb/overview/index-093405.html

Oracle NoSQL Database offers full Create, Read, Update and Delete (CRUD) operations with adjustable durability guarantees. Oracle NoSQL Database is designed to be highly available, with excellent throughput and latency, while requiring minimal administrative interaction.

Oracle NoSQL Database provides performance scalability. If you require better performance, you use more hardware. If your performance requirements are not very steep, you can purchase and manage fewer hardware resources.

Oracle NoSQL Database is meant for any application that requires network-accessible data with user-definable read/write performance levels. The typical application is a web application which is servicing requests across the traditional three-tier architecture: web server, application server, and back-end database. In this configuration, Oracle NoSQL Database is meant to be installed behind the application server, causing it to either take the place of the back-end database, or work alongside it. To make use of Oracle NoSQL Database, code must be written that runs on the application server.

An application makes use of Oracle NoSQL Database by performing network requests against Oracle NoSQL Database's data store, which is referred to as the KVStore. The requests are made using the Oracle NoSQL Database Driver, which is linked into your application as a Java library (.jar file), and then accessed using a series of Java APIs.

The usage of these APIs is introduced in one of two manuals. Most developers will want to read *Oracle NoSQL Database Getting Started with the Table API*. Developers who want to use the older, legacy Key/Value API should read *Oracle NoSQL Database Getting Started with the Key/Value API*.

Note

Oracle NoSQL Database is tested using Java 8, and so Oracle NoSQL Database should be used only with that version of Java.

You can also access data stored in Oracle NoSQL Database tables by using a non Java language driver. C, Node.js and Python drivers are available. For more information, see the specific driver's Quick Start Guide.

Finally, Oracle NoSQL Database provides SQL for Oracle NoSQL Database, which is an easy to use SQL-like language that supports read-only queries and data definition (DDL) statements. This language can be used to access data in tables in a read-only fashion.

To follow along query examples run with the interactive shell, see *Getting Started with SQL for Oracle NoSQL Database*.

If you are interested in using the JAVA API to execute queries see *Getting Started with the Table API*.

For a more detailed description of the SQL language (both DDL and query statements) see the *SQL for Oracle NoSQL Database Specification*.

NoSQL Database Server Licensing

Oracle NoSQL Database Server is available one of two licensing options: Oracle NoSQL Database Community Edition (CE) and Oracle NoSQL Database Enterprise Edition (EE). For a description on these two licenses, see:

http://docs.oracle.com/cd/NOSQL/html/EE-CE.Differences.html

NoSQL Database Client Licensing

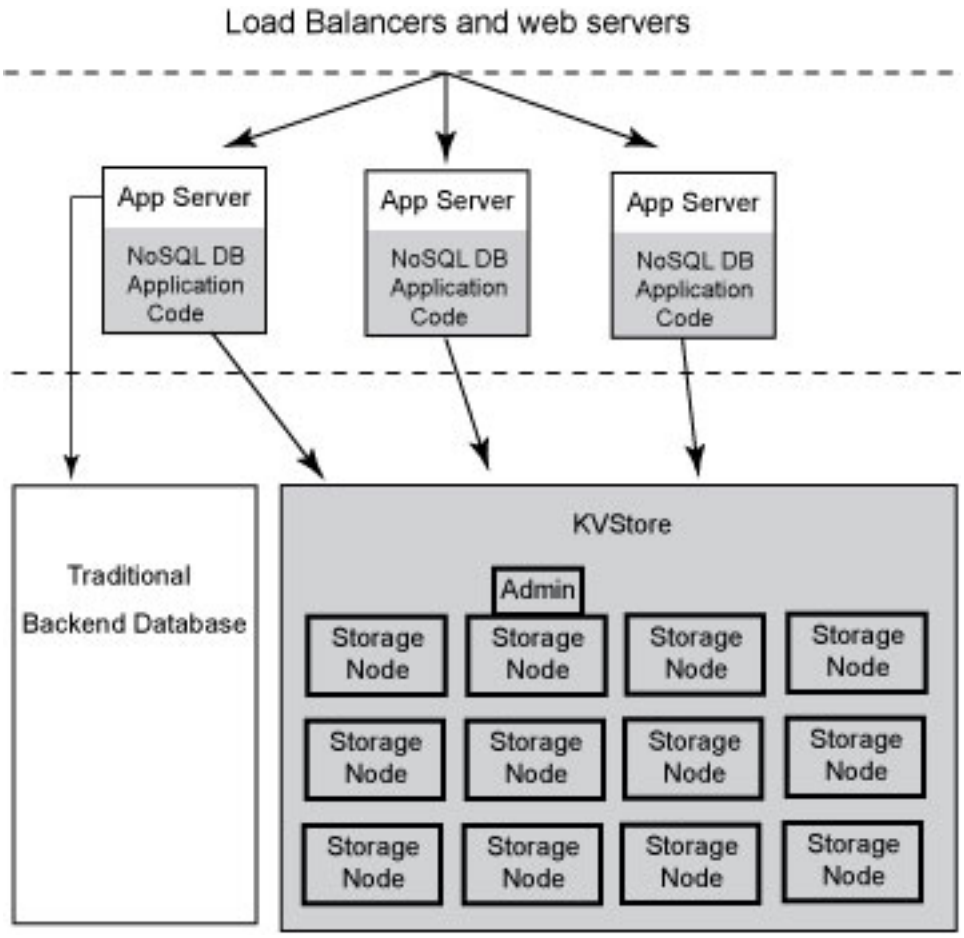
Oracle NoSQL Database client APIs are released as open source. They ship with source code and are released under the Apache 2.0 License. These client APIs may be used to access Oracle NoSQL Database servers using either the Community Edition (CE) or Enterprise Edition (EE) licenses.

Architecture

The KVStore is a collection of Storage Nodes which host a set of Replication Nodes. Data is spread across the Replication Nodes. Given a traditional three-tier web architecture, the KVStore either takes the place of your back-end database, or runs alongside it.

The store contains multiple Storage Nodes. A *Storage Node* is a physical (or virtual) machine with its own local storage. The machine is intended to be commodity hardware. It should be, but is not required to be, identical to all other Storage Nodes within the store.

The following illustration depicts the typical architecture used by an application that makes use of Oracle NoSQL Database:



Every Storage Node hosts one or more Replication Nodes as determined by its *capacity*. The capacity of a Storage Node serves as a rough measure of the hardware resources associated with it. A store can consist of Storage Nodes of different capacities. Oracle NoSQL Database will ensure that a Storage Node is assigned a load that is proportional to its capacity. A Replication Node in turn contains at least one and typically many partitions. Also, each Storage Node contains monitoring software that ensures the Replication Nodes which it hosts are running and are otherwise healthy.

For more information on how to associate capacity with a Storage Node and know the best way to balance the number of Storage Nodes and Replication Nodes, see *Oracle NoSQL Database Administrator's Guide*.

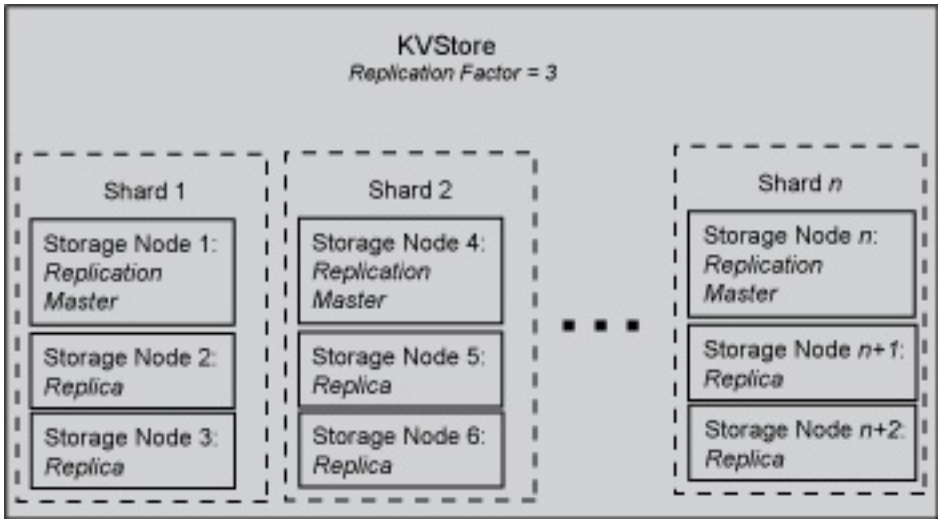
Replication Nodes and Shards

At a very high level, a *Replication Node* can be thought of as a single database which contains key-value pairs.

Replication Nodes are organized into *shards*. A shard contains a single Replication Node, called the *master* node, which is responsible for performing database writes, as well as one or more

read-only *replicas*. The *master* node copies all writes to the replicas. These replicas are used to service read-only operations. Although there can be only one master node at any given time, any of the members of the shard (with the exception of nodes in a secondary zone as described below) are capable of becoming a *master* node. In other words, each shard uses a single master/multiple replica strategy to improve read throughput and availability.

The following illustration shows how the KVStore is divided up into shards:



Note that if the machine hosting the master should fail in any way, then the master automatically fails over to one of the other nodes in the shard. That is, one of the replica nodes is automatically promoted to master.

Production KVStores should contain multiple shards. At installation time you provide information that allows Oracle NoSQL Database to automatically decide how many shards the store should contain. The more shards that your store contains, the better your write performance is because the store contains more nodes that are responsible for servicing write requests.

Replication Factor

The number of nodes belonging to a shard is called its *Replication Factor*. The larger a shard's Replication Factor, the faster its read throughput (because there are more machines to service the read requests) but the slower its write performance (because there are more machines to which writes must be copied).

Once you set the Replication Factor for each zone in the store, Oracle NoSQL Database makes sure the appropriate number of Replication Nodes are created for each shard residing in each zone making up your store. The number of copies, or replicas, maintained in a zone is called the *Zone Replication Factor*. The total number of replicas in all Primary zones is called the *Primary Replication Factor*, and the total number in all Secondary zones is called the *Secondary Replication Factor*. For all zones in the store, the total number of replicas across the entire store is called the *Store Replication Factor*.

For additional information on how to identify the *Primary Replication Factor* and its implications, as well on multiple zones and replication factors see the *Oracle NoSQL Database Administrator's Guide*.

Partitions

Each shard contains one or more *partitions*. Table rows (or key-value pairs) in the store are accessed by the data's key. Keys, in turn, are assigned to a partition. Once a key is placed in a partition, it cannot be moved to a different partition. Oracle NoSQL Database spreads records evenly across all available partitions by hashing each record's key.

As part of your planning activities, you must decide how many partitions your store should have. Note that this is not configurable after the store has been installed.

It is possible to expand and change the number of Storage Nodes in use by the store. When this happens, the store can be reconfigured to take advantage of the new resources by adding new shards. When this happens, partitions are balanced between new and old shards by redistributing partitions from one shard to another. For this reason, it is desirable to have enough partitions so as to allow fine-grained reconfiguration of the store. Note that there is a minimal performance cost for having a large number of partitions. As a rough rule of thumb, there should be at least 10 to 20 partitions per shard, and the number of partitions should be evenly divisible by the number of shards. Since the number of partitions cannot be changed after the initial deployment, you should consider the maximum future size of the store when specifying the number of partitions.

Zones

A zone is a physical location that supports good network connectivity among the Storage Nodes deployed in it and has some level of physical separation from other zones. A zone generally includes redundant or backup power supplies, redundant data communications connections, environmental controls (for example: air conditioning, fire suppression) and security devices. A zone may represent an actual physical data center building, but could also represent a floor, room, pod, or rack, depending on the particular deployment. Oracle recommends you install and configure your store across multiple zones to guard against systemic failures affecting an entire physical location, such as a large scale power or network outage.

Multiple zones provide fault isolation and increase the availability of your data in the event of a single zone failure.

Zones come in two types. *Primary* zones contain nodes which can serve as masters or replicas. Zones are created as primary zones by default. *Secondary* zones contain nodes which can only serve as replicas. Secondary zones can be used to make a copy of the data available at a distant location, or to maintain an extra copy of the data to increase redundancy or read capacity.

Only primary zones can have a Replication Factor equal to zero. This type of zone is useful to host Arbiter Nodes. Zero capacity Storage Nodes would be added to this zone in order to host the Arbiter Nodes.

Note

Only primary zones can host Arbiter Nodes.

You can use the command line interface to create and deploy one or more `zones`. Each `zone` hosts the deployed storage nodes. For additional information on zones and how to create them see the *Oracle NoSQL Database Administrator's Guide*.

Topologies

A *topology* is the collection of zones, storage nodes, replication nodes and administration services that make up a NoSQL DB store. A deployed store has one topology that describes its state at a given time.

After initial deployment, the topology is laid out so as to minimize the possibility of a single point of failure for any given shard. This means that while a Storage Node might host more than one Replication Node, those Replication Nodes will never be from the same shard. This improves the chances of the shard continuing to be available for reads and writes even in the face of a hardware failure that takes down the host machine.

Arbiter Nodes are automatically configured in a topology if the primary replication factor is two and a zone is configured to host `Arbiter Nodes`.

Topologies can be changed to achieve different performance characteristics, or in reaction to changes in the number or characteristics of the Storage Nodes. Changing and deploying a topology is an iterative process. For information on how to use the command line interface to create, transform, view, validate and preview a topology, see the *Oracle NoSQL Database Administrator's Guide*.

Arbiter Nodes

An `Arbiter Node` is a lightweight process that supports write availability in two situations. First, when the primary replication factor is two and a single Replication Node becomes unavailable. Second, when two Replication Nodes cannot talk to each other to figure out who's master. The role of an `Arbiter Node` is to participate in elections and respond to acknowledge requests in these situations. An Arbiter Node does not hold any data, which is why Storage Nodes may be created with capacity equal to zero for the purpose of hosting an Arbiter Node.

The Arbiter Node will be allocated on a Storage Node outside of the shard. An error will result if there are not enough Storage Nodes to host an Arbiter Node located on a different Storage Node as other shard members. The Arbiter Node provides write availability in the absence of a single Storage Node. The pool of Storage Nodes in a primary zone configured to host Arbiter Nodes are used for Arbiter Node allocation. Storage Nodes may be created with capacity equal to zero for the purpose of hosting an Arbiter Node. Arbiter Nodes may also be allocated on nodes with capacity greater than zero, but zero capacity Storage Nodes have a higher priority in Arbiter Node allocation.

For information on Arbiter Nodes see the *Oracle NoSQL Database Administrator's Guide*.

Data Models

You can model your data in Oracle NoSQL Database by using Tables, JSON schemas or a raw key-value interface.

Tables are the easiest way to model data. They provide the highest level of abstraction, they are simple to model and should be familiar to any developer. This model also supports secondary indices and table evolution. For more information on the tables API, see *Oracle NoSQL Database Getting Started with the Table API*.

You can use JSON to model data for your JSON centric applications. If secondary indices or strongly typed keys are not a priority and if keys are going to be modeled manually, then this is a good choice. For more information on JSON, see the *Oracle NoSQL Database Administrator's Guide*.

Finally, if you want to serialize the data, manage the key structure, manage secondary indices through index views, manage evolution and security through your client code, then you can use the raw key-value interface. This is a good choice if you are willing to write code on to your store to manage things that are either available in the JSON or table interfaces. For more information on the key-value API, see *Oracle NoSQL Database Getting Started with the Key/Value API*.

Consistency

Oracle NoSQL Database provides several different consistency policies. At one end of the spectrum, applications can specify absolute consistency, which guarantees that all reads return the most recently written value for a designated key. At the other end of the spectrum, applications capable of tolerating inconsistent data can specify weak consistency, allowing the database to return a value efficiently even if it is not entirely up to date. In between these two extremes, applications can specify time-based consistency to constrain how old a record might be or version-based consistency to support both atomicity for read-modify-write operations and reads that are at least as recent as the specified version.

The following illustration depicts the range of consistency policies that can be used by an application that makes use of Oracle NoSQL Database:

Flexible consistency policies enables developers to easily create business solutions providing data guarantees while meeting application latency and scalability requirements.

Durability

Oracle NoSQL Database provides a range of durability policies that specify what guarantees the system makes after a crash. At one extreme, applications can request that write requests block until the record has been written to stable storage on all copies. This has obvious performance and availability implications, but ensures that if the application successfully writes data, that data will persist and can be recovered even if all the copies become temporarily unavailable due to multiple simultaneous failures. At the other extreme, applications can request that write operations return as soon as the system has recorded the existence of the write, even if the data is not persistent anywhere. Such a policy provides the best write performance, but provides no durability guarantees.

The following illustration depicts the range of durability policies that can be used by an application that makes use of Oracle NoSQL Database:

By specifying when the database writes records to disk and what fraction of the copies of the record must be persistent (none, all, or a simple majority), applications can enforce a wide range of durability policies.

Quorum

Operations that modify data in Oracle NoSQL Database require that at least a simple majority of primary nodes be available to form a quorum in the shard that stores the specified key.

Quorum is the minimum number of primary nodes required in a shard, or in the set of admin nodes, to permit electing a master to support write operations. The quorum is the minimum number of nodes that represents a majority of the primary nodes in the group.

Note

Secondary nodes are not counted when computing the quorum.

Consider the following example using a store with four zones. Zones 1, 2, and 3 are primary zones with replication factor 1, and zone 4 is a secondary zone with replication factor 1. The number of primary nodes in each shard is 3, which is the sum of the replication factors for the primary zones. In a group of 3 nodes, 2 is the smallest number of nodes that represent a majority, so the quorum is 2. The secondary nodes in zone 4 have no impact on the quorum.

In general, to compute the quorum, first determine the primary replication factor, which is the sum of the replication factors of all primary zones. The quorum is one greater than half of the primary replication factor, rounding down when computing the half.

For example, for primary replication factor of 1, the quorum is 1. For primary replication factor of 5 the quorum is 3. For primary replication factor of 6, the quorum is 4.

Administration

The Administration command line interface (CLI) is the primary tool used to manage your store. It is used to configure, deploy, and change store components. It can also be used to verify the system, check service status, check for critical events and browse the store-wide log file. Alternatively, you can use a browser-based graphical user interface to do read-only monitoring. (Described in the next section.)

The CLI can also be used to get, put, and delete store records or tables, retrieve schema, and display general information about the store. It can also be used to diagnose problems or potential problems in the system, fix actual problems by adding, removing or modifying store data and/or verify that the store has data. It is particularly well-suited for a developer who is creating an Oracle NoSQL Database application, and who needs to either populate a store a with a small number of records so as to have data to develop against, or to examine the store's state as part of development debugging activities.

The command line interface is accessed using the following command:

```
java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar runadmin
```

> **Note**
> To avoid using too much heap space, you should specify -Xmx and -Xms flags for Java when running administrative and utility commands.

For a complete listing of all the commands available to you in the CLI as well as its usage, see the *Oracle NoSQL Database Administrator's Guide*.

KVLite

KVLite is a simplified version of Oracle NoSQL Database. It provides a single-node store that is not replicated. It runs in a single process without requiring any administrative interface. You configure, start, and stop KVLite using a command line interface.

KVLite is intended for use by application developers who need to unit test their Oracle NoSQL Database application. It is not intended for production deployment, or for performance measurements.

KVLite is installed when you install KVStore. It is available in the `kvstore.jar` file in the `lib` directory of your Oracle NoSQL Database distribution.

Note that KVLite cannot be configured as a secure store.

For more information on KVLite, see either *Oracle NoSQL Database Getting Started with the Table API* or *Oracle NoSQL Database Getting Started with the Key/Value API*.

The Administration Command Line Interface history

By default Oracle NoSQL Database uses the Java Jline library to support saveable command line history in the CLI. If you want to disable this feature, the following Java property should be set while running runadmin:

```
java -Doracle.kv.shell.jline.disable=true -jar KVHOME/kvstore.jar \
runadmin -host <hostname> -port <portname>
```

Command line history is saved to a file so that it is available after restart.
By default, Oracle NoSQL Database attempts to save the history in a KVHOME/.jlineoracle.kv.impl.admin.client.CommandShell.history file, which is created and opened automatically. The default history saved is 500 lines.

> **Note**
> If the history file cannot be opened, it will fail silently and the CLI will run without saved history.

The default history file path can be overridden by setting the `oracle.kv.shell.history.file="path"` Java property.

The default number of lines to save to the file can be modified by setting the `oracle.kv.shell.history.size=<int_value>` Java property.

Monitoring

Information about the performance and availability of your store is available. You can monitor the information through an API class, log files, and Java Management Extensions (JMX).

These agents provide interfaces on each Storage Node that allow management clients to poll them for information about the status, performance metrics, and operational parameters of the Storage Node and its managed services, including replication nodes, and admin instances. Also, JMX can be used to monitor Arbiter Nodes.

For more information, see the *Oracle NoSQL Database Administrator's Guide*.

Troubleshooting

Errors can occur in your store deployment. Tools, commands, logs and procedures can be used in order to solve problems.

To catch configuration errors early, you can use the `Diagnostics Utility`. You can also use this tool to package information and files to send them to Oracle Support, for example.

For more information on troubleshooting your store, see the *Oracle NoSQL Database Administrator's Guide*.

Access and Security

Access to the KVStore and its data is performed in two different ways. Routine access to the data is performed using Java APIs that the application developer uses to allow his application to interact with the Oracle NoSQL Database Driver, which communicates with the store's Storage Nodes in order to perform whatever data access the application developer requires.

In addition, administrative access to the store is performed using a command line interface or a browser-based graphical user interface. System administrators use these interfaces to perform the few administrative actions that are required by Oracle NoSQL Database. You can also monitor the store using these interfaces.

For most production stores, authentication over SSL is normally required by both the command line interface and the Java APIs. It is possible to install a store such that authentication is not required, but this is not recommended. For details on Oracle NoSQL Database's security features, see the *Oracle NoSQL Database Security Guide*.

Note

Oracle NoSQL Database is intended to be installed in a secure location where physical and network access to the store is restricted to trusted users. For this reason, at this time Oracle NoSQL Database's security model is designed to prevent accidental access to the data. It is *not* designed to prevent denial-of-service attacks.

Integration

Oracle NoSQL Database can be integrated with Apache Hadoop and products in the Oracle stack. In the following sections you will learn more about this.

Oracle Database Mobile Server Integration

As of Oracle Database Mobile Server (Oracle DMS) 12.1 release, Oracle NoSQL Database can be integrated with Oracle Database Mobile Server. Oracle DMS facilitates the development, deployment and management of mobile database applications for a large number of mobile users.

The main integration feature is synchronizing Oracle NoSQL with Oracle Berkeley DB/SQLite/ Java DB.

Oracle DMS can be used as a data synchronization engine between Oracle NoSQL and mobile client databases including Oracle Berkeley DB, SQLite and Java DB. Existing tables in an Oracle NoSQL database can be published to Oracle DMS using publish/subscribe APIs. Oracle DMS creates corresponding tables on the client-side and allows them to synchronously upload their data to Oracle NoSQL. Since, Oracle DMS completely manages the data format conversion between the local client database and Oracle NoSQL DB, no additional coding or scripting is required. The mobile to NoSQL paradigm fits best where the number of clients is large, and/ or the amount of collected data is large, and/or the server application requires NoSQL as data storage.

With Oracle NoSQL as data storage, mobile and embedded applications can use Oracle DMS as a complete and reliable solution for synchronizing their data to the highly scalable Oracle NoSQL database. For more information, see section 2.12.3 *"Creating NoSQL Queue Publication Item"* in the *Oracle Database Mobile Server Developer's Guide*. You can also find the FleetControl sample application in Oracle Mobile Development Kit (a development toolkit for Oracle DMS), which demonstrates how to use Oracle NoSQL as a back-end database and how to synchronize application data between mobile client database and Oracle NoSQL via Oracle DMS.

Hadoop Integration

Oracle NoSQL Database can be integrated with Apache Hadoop systems using the `oracle.kv.hadoop.KVInputFormat` class. This class allows you to read data from Oracle

NoSQL Database and then prepare it for insertion into a Hadoop system. To move data in the reverse, you can read data from the Hadoop system using the standard mechanisms, and then write the records to Oracle NoSQL Database using the APIs described in this book.

An example of using `KVInputFormat` to read data from Oracle NoSQL Database in a Map/Reduce job can be found in the `<KVHOME>/examples/hadoop` directory.

In addition, Oracle NoSQL Database provides the `oracle.kv.AvroFormatter` interface. This is used to support Oracle Loader for Hadoop (OLH), which can read data directly from Oracle NoSQL Database and write it to Oracle Database as a Map/Reduce job. OLH is described in the *Oracle Loader for Hadoop* chapter of the *Oracle Big Data Connectors User's Guide*, which you can find here:

http://docs.oracle.com/cd/E37231_01/doc.20/e36961/olh.htm#CBHGEJDE

Property Graph Integration

Oracle Big Data Spatial and Graph can be configured to use Oracle NoSQL Database to support its property graph feature. This feature supports graph operations, indexing, queries, search and in-memory analytics.

Graphs are commonly used to model, store, and analyze relationships found in social networks, cyber security, utilities and telecommunications, life sciences and clinical data, and knowledge networks. Typical graph analyses encompass graph traversal, recommendations, finding communities and influencers, and pattern matching.

For more information, see the Big Data Spatial and Graph User's Guide and Reference

Oracle External Tables Integration

Oracle NoSQL Database data can be accessed using Oracle Database's External Tables feature. This capability allows NoSQL Database data to be read into Oracle Database. NoSQL Database data cannot be modified using the External Tables feature.

Note that this is a feature which is only available to users of the Oracle NoSQL Database Enterprise Edition.

To use the Oracle Database External Table feature to read Oracle NoSQL Database data, you must use the `<KVHOME>/exttab/bin/nosql_stream` preprocessor to populate our Oracle tables with the data. You must then configure your Oracle Database to use the External Tables feature.

For information on how to use the `nosql_stream` preprocessor, and how to configure Oracle Database to use External Tables, see the oracle.kv.exttab package summary.

Coherence Integration

Oracle Coherence is a middleware application that provides data management support for clustered applications. The data that an application delegates to Oracle Coherence are automatically available to and accessible by all servers in the application cluster. By distributing data across multiple machines, Oracle Coherence solves problems related to

achieving availability, reliability, scalability, performance, serviceability and manageability of clustered applications.

Oracle Coherence is described here: http://docs.oracle.com/cd/E24290_01/index.htm

To provide these solutions, Oracle Coherence implements a cache. This cache can be customized to use a number of different data repositories to store the cached data. One such data repository is Oracle NoSQL Database.

Note that this is a feature which is only available to users of the Oracle NoSQL Database Enterprise Edition.

To integrate with Oracle NoSQL Database, Oracle Coherence must be customized using a combination of configuration XML, stock Coherence code, and custom Oracle NoSQL Database code. The Oracle NoSQL Database code is implemented using classes provided by the `oracle.kv.coherence` package.

Oracle NoSQL Database can be used to support two different caching strategies for Oracle Coherence. The first of these is implemented by oracle.kv.coherence.NoSQLBinaryStore. This class allows you to implement cache data that is not meant to be shared with non-cache-based applications, and so uses a data format that is fairly opaque. This is an efficient and easy-to-configure caching option.

Alternatively, you can implement a caching strategy that results in data which is meant to be shared with non-cache-based applications. You do this using oracle.kv.coherence.NoSQLAvroCacheStore. This caching mechanism is Avro-aware, and so any Avro-compliant application will be able to read and write these data records.

In order for Oracle Coherence to use these Avro-based objects, it must be able to serialize the Avro records for transmission over its network. To enable this, you use the oracle.kv.coherence.NoSQLAvroSerializer class.