



**Hardware and Software**  
Engineered to Work Together



# Oracle NoSQL Database for Developers

Student Guide

D76021GC10

Edition 1.0 | April 2015 | D77500

Learn more from Oracle University at [oracle.com/education/](http://oracle.com/education/)

**Author**

Salome Clement

**Technical Contributors  
and Reviewers**

Ashwin Agarwal  
Feisal Ahmad  
Yanti Chang  
Ron Cohen  
Steve Friedberg  
Mark Fuller  
Joel Goodman  
Nancy Greenberg  
Matthew Gregory  
Ashok Joshi  
Sailaja Pasupuleti  
Shankar Raman  
Bryan Roberts  
Anuj Sahni  
Swarnapriya Shridhar  
Sharon Stephen  
Drishya TM  
Christopher Wensley

**Editors**

Arijit Ghosh  
Raj Kumar

**Graphic Designer**

Rajiv Chandrabhanu

**Publishers**

Glenn Austin  
Srividya Rameshkumar  
Giri Venugopal

**Copyright © 2015, Oracle and/or its affiliates. All rights reserved.**

**Disclaimer**

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

**Restricted Rights Notice**

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

**U.S. GOVERNMENT RIGHTS**

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

**Trademark Notice**

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

## Contents

### 1 Introduction

- Objectives 1-2
- Course Goals 1-3
- Course Outline 1-4
- Course Environment 1-5
- Accessing the labs Directory 1-6
- Meet Doris 1-7
- Course Supplement 1-8
- Additional Resources 1-9
- Summary 1-10

### Unit I: Designing Application Schema

- Course Outline I-2
- Doris Designs the Applications I-3

### 2 Big Data and NoSQL: Overview

- Objectives 2-2
- Doris Prepares for Meeting with Team 2-3
- Defining Big Data 2-4
- The Changing Data World 2-5
- What Is a NoSQL Database? 2-6
- RDBMS Versus NoSQL 2-7
- What Is HDFS? 2-8
- HDFS Versus NoSQL 2-9
- Quiz 2-10
- Requirement in the Big Data World Today 2-11
- Oracle Integrated Software Solution 2-12
- Scenario 1: Retail Marketing System 2-13
- Scenario 2: Human Resources System 2-14
- Scenario 3: Healthcare System 2-15
- Big Data: Examples 2-16
- Summary 2-17
- Practice 2 Overview: Big Data and NoSQL 2-18

### **3 Oracle NoSQL Database: Overview**

- Objectives 3-2
- Doris Explores Oracle NoSQL Database 3-3
- Oracle NoSQL Database 3-4
- Key Features 3-5
- How Oracle NoSQL Database Works 3-6
- Architecture 3-7
- KVStore Components 3-8
- Quiz 3-10
- Request Processing: Example 3-11
- Performance and Throughput 3-12
- Oracle NoSQL Database Use Cases 3-13
- When to Use Oracle NoSQL Database 3-14
- Accessing the KVStore 3-17
- Schema: Overview 3-18
- KVLite: Introduction 3-19
- Starting and Stopping KVLite 3-20
- Verifying That a KVStore Is Running 3-21
- Restarting and Rerunning KVLite 3-22
- Summary 3-23
- Practice 3 Overview: Using KVLite 3-24

### **4 Schema Design**

- Objectives 4-2
- Doris Discusses with Team 4-3
- Importance of Schema Design 4-4
- Schema Design Options in ONDB 4-5
- Key-Value Data Model 4-6
- Table Data Model: Overview 4-8
- Designing Parent Tables 4-9
- Table Field Data Types 4-10
- Keys and Indexes for Table Data Model 4-11
- Parent Table: Examples 4-13
- Defining Child Tables 4-14
- Creating Child Tables Versus Record Fields 4-15
- Schema Design Options in ONDB: Summary 4-16
- Using Keys to Retrieve Data 4-17
- RDBMS Versus Table Data Model 4-18
- Quiz 4-19
- Introducing Sample Email Application 4-20
- Data Modeling for the Email App 4-21

Email App Schema 4-22  
Design Considerations 4-23  
Summary 4-24  
Practice 4 Overview: Designing Data Model 4-25

## 5 Application-Specific Requirements

Objectives 5-2  
Doris Finalizes Application Design 5-3  
Write and Read Process 5-4  
Consistency Policy: Definition 5-5  
Applying Consistency 5-6  
Default Consistency 5-7  
Types of Consistency Policies 5-8  
Predefined Consistency Policies 5-9  
Time-Based Consistency Policy 5-10  
Version-Based Consistency Policy 5-11  
Consistency Policies: Summary 5-12  
Quiz 5-13  
Write Process 5-14  
Durability: Definition 5-15  
Applying Durability 5-16  
Durability Policy 5-17  
Synchronization Policy 5-18  
Acknowledgment Policy 5-19  
Default Durability 5-20  
Durability Polices: Summary 5-21  
Quiz 5-22  
Summary 5-23  
Practice 5 Overview: Application-Specific Requirements 5-24

## 6 Creating Tables

Objectives 6-2  
Doris Creates Application Tables 6-3  
Creating Tables: Overview 6-4  
Data Definition Language Commands 6-5  
CREATE TABLE 6-6  
CHECK Constraint 6-8  
Creating a Table from a Java Application 6-9  
Introducing TableAPI 6-10  
Executing a DDL Command 6-11  
Executing a DDL Command: Example 6-12

Quiz	6-13
Creating a Table from the CLI	6-14
Accessing the CLI	6-15
Executing a DDL Command	6-16
Viewing Table Descriptions	6-17
Recommendation: Using Scripts	6-18
Modifying a Table	6-19
Deleting a Table	6-20
Quiz	6-21
Indexes: Introduction	6-22
Creating Indexes	6-23
Removing Indexes	6-24
Summary	6-25
Practice 6 Overview: Creating Table Data	6-26

## **Unit I: Summary of Designing Application Schema**

Course Outline	I-2
Doris Has a Design	I-3

## **Unit II: Creating, Retrieving, and Updating Data**

Course Outline	II-2
Doris Starts Coding	II-3
KVStore Handle	II-4
Creating a KVStore Handle	II-5
Using the KVStoreConfig Class	II-6
KVStoreConfig Class Definition	II-7
Using the KVStoreFactory Class	II-8
Creating a KVStore Handle: Example	II-9
Quiz	II-10

## **7 Creating Table Data**

Objectives	7-2
Doris Populates Created Tables	7-3
TableAPI	7-4
TableAPI Methods for Write Operations	7-5
Writing Rows to Tables: Steps	7-6
Constructing Handles	7-7
Creating a Row Object, Adding Fields, and Writing a Record	7-8
Write Method Definitions	7-9
Creating the Row Object	7-10
Using the Row Object	7-11

putIfAbsent(): Use Case	7-12
putIfPresent(): Use Case	7-13
Quiz	7-14
Writing Rows to Child Tables: Steps	7-15
Writing Rows to Child Tables: Example	7-16
TableAPIs for Delete Operations	7-17
Deleting Row(s) from a Table: Steps	7-18
Delete Method Definitions	7-19
Scenario	7-21
Versions	7-22
Scenario	7-23
Summary	7-24
Practice 7 Overview: Writing Data to Tables	7-25

## **8 Retrieving Table Data**

Objectives	8-2
Doris Reads Table Data	8-3
Reading a Table: Overview	8-4
Retrieving Table Data: Steps	8-5
Retrieving a Single Row	8-6
Retrieving Multiple Rows	8-7
Retrieving Child Tables	8-8
Iterating a Table	8-9
Using MultiRowOptions	8-11
Specifying Ranges	8-12
Retrieving Nested Tables	8-13
Reading Indexes	8-15
Parallel Scans	8-16
Quiz	8-17
Versions	8-18
Summary	8-19
Practice 8 Overview: Reading Data from Tables	8-20

## **9 Using Key-Value APIs**

Objectives	9-2
Doris Explores the Key-Value Model	9-3
Structure of a Record: Review	9-4
Creating a Key Component: Overview	9-5
Creating Key Components	9-6
Creating a Value Component: Overview	9-7
Creating a Value	9-8

Quiz	9-9
Retrieving Records: Overview	9-10
get() Method	9-11
multiGet() Method	9-12
multiGet(): Example	9-13
Creating a Key Range	9-14
Key Depth	9-15
Using multiGetIterator()	9-16
multiGetIterator(): Example	9-17
storeIterator() Method	9-18
storeIterator(): Example	9-19
Methods Summary	9-20
Quiz	9-21
Methods: Overview	9-22
Writing Key-Value Pair to KVStore	9-23
Deleting Records	9-24
Deleting a Set of Records: Example	9-25
Working with Versions	9-26
Summary	9-27
Practice 9 Overview: Working with Key-Value Data	9-28

## **Unit II: Summary of Creating, Retrieving, and Updating Data**

Course Outline	II-2
Doris Has a Functional Application	II-3

## **Unit III: Implementing Application-Specific Requirements**

Course Outline	III-2
Doris Configures Application Requirements	III-3

## **10 Configuring Consistency**

Objectives	10-2
Consistency Policy: Review	10-3
Viewing the Default Consistency Policy	10-4
Creating Consistency Policies: Overview	10-5
Using a Predefined Consistency Policy	10-6
Creating a Time-Based Consistency Policy	10-7
Consistency.Time: Example	10-8
Creating a Version-Based Consistency Policy	10-9
Consistency.Version: Example	10-10

Changing a Default Consistency Policy 10-11  
Summary 10-12  
Practice 10 Overview: Setting Consistency Policies 10-13

## **11 Configuring Durability**

Objectives 11-2  
Durability: Review 11-3  
Viewing the Default Durability Policy 11-4  
Creating Durability Policies: Overview 11-5  
Setting a Synchronization-Based Durability Policy 11-6  
Setting an Acknowledgment-Based Durability Policy 11-7  
Creating a New Durability Policy 11-8  
Changing the Default Durability Policy 11-9  
Summary 11-10  
Practice 11 Overview: Setting Durability Policies 11-11

## **12 Creating Transactions**

Objectives 12-2  
What Is a Transactional Operation? 12-3  
Creating Transactions: Points to Remember 12-4  
Creating and Running Transactional Operations: Process 12-5  
Creating and Running Transactional Operations: Example 12-6  
TableOperationFactory Methods 12-7  
Executing Operation Syntax 12-8  
Summary 12-9  
Practice 12 Overview: Creating Transactions 12-10

## **13 Handling Large Objects**

Objectives 13-2  
Introducing Large Objects 13-3  
Oracle NoSQL APIs for Large Objects 13-4  
Large Objects Storage 13-5  
Creating Large Object Keys 13-6  
Creating a Key for Table API Users 13-8  
Quiz 13-9  
Storing Large Objects: API Overview 13-10  
Storing Large Objects: Code Example 13-11  
Retrieving Large Objects: API Overview 13-12  
Retrieving Large Objects: Code Sample 13-13  
Deleting Large Objects: API Overview 13-14

Deleting Large Objects: Code Example 13-15  
Summary 13-16  
Practice 13 Overview: Handling Large Objects 13-17

## **14 Accessing a Secure Store**

Objectives 14-2  
Secure KVStore 14-3  
Security Features 14-4  
Obtaining Handle to Secure Store 14-5  
Security Parameters 14-6  
Accessing a KVStore: Example 14-8  
Specifying Security Properties 14-9  
Security Constants 14-10  
Summary 14-11  
Quiz 14-12

## **15 Handling Exceptions**

Objectives 15-2  
Understanding Exceptions 15-3  
Oracle NoSQL Database Exceptions 15-4  
ContingencyException 15-6  
FaultException 15-7  
RequestLimitException 15-8  
RequestLimitConfig 15-9  
Handling RequestTimeoutException 15-10  
Methods That Throw RequestTimeoutException 15-11  
Quiz 15-12  
Handling ConsistencyException 15-13  
Handling DurabilityException 15-14  
Handling TableOpExecutionException 15-15  
Handling PartialLOBException 15-16  
Handling Security Exceptions 15-17  
Summary 15-18

## **Unit III: Summary of Implementing Application-Specific Requirements**

Course Outline III-2

# 1

## Introduction

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Describe the course goals
- Identify the course environment

## Course Goals

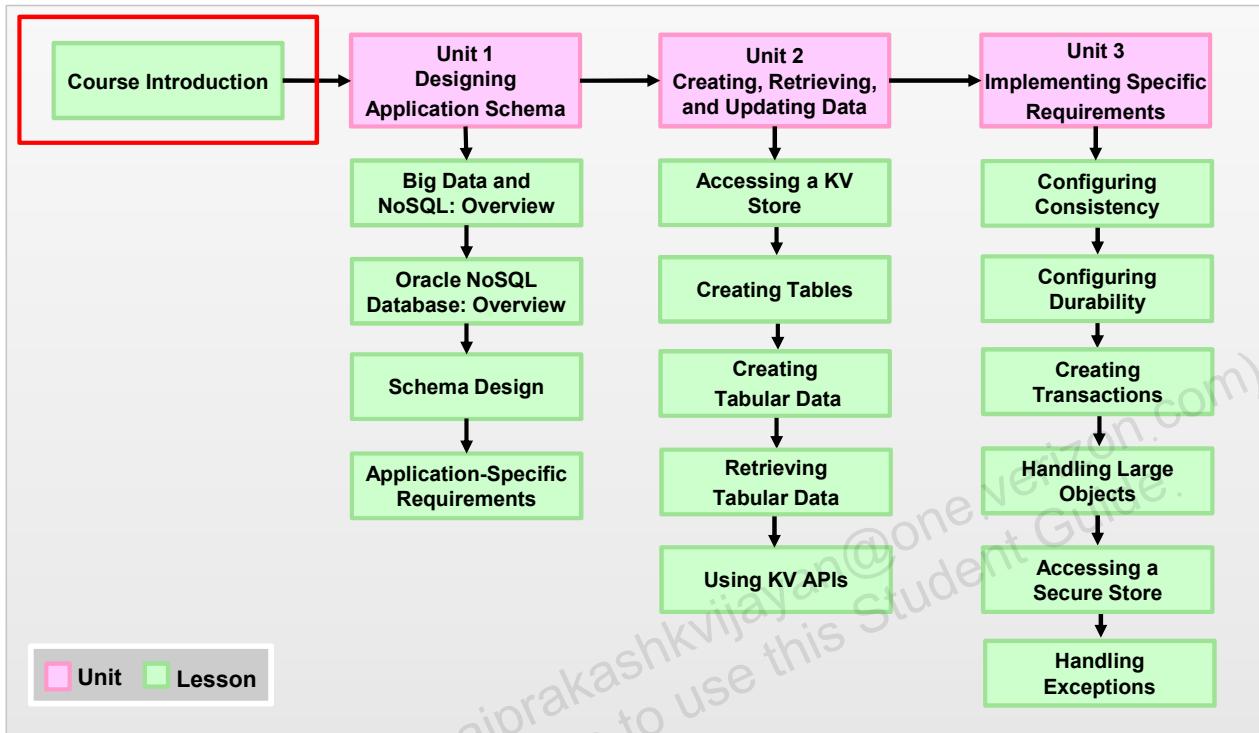
After completing the course, you should be able to:

- Design a data model for Oracle NoSQL Database
- Use Oracle NoSQL Database APIs in an application to create, retrieve, and update data in a KVStore
- Create consistency and durability policies according to application requirements and access a secure store



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Course Outline



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Course Environment

- Each student will get a set of four VM machines.
- Common software:
  - Oracle Enterprise Linux
  - Oracle NoSQL Database 12c (12.1.3.2.5)
- Additionally, VM 4 has NetBeans 7.1.



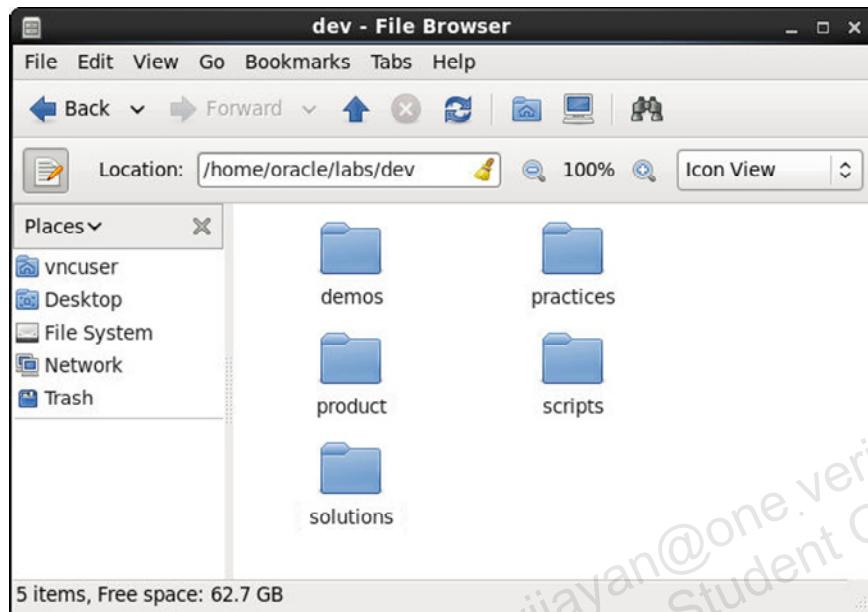
ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The instructor will provide a course environment for the practice exercises. The course environment is a set of four VMs hosting the KVStore for the class. The operating system is Oracle Enterprise Linux, and the latest Oracle NoSQL Database software has been installed.

You will use VM 4 to access NetBeans for the developer class.

## Accessing the labs Directory



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The files that you need for this course are available in the `/home/oracle/labs/dev` directory of VM4 (host4). You will see the following folders:

- **demos**: Contains scripts and files that are used by the instructor to demo in the class
- **practices**: Contains NetBeans applications with incomplete code that you will complete in the labs
- **product**: Contains the NetBeans setup file
- **scripts**: Contains scripts and files that you will use to complete practices for this course
- **solutions**: Contains NetBeans applications with completed practices code

# Meet Doris

**Name:** Doris

**Company:** XWHYZEE Co.

**Designation:** Application Developer

**Experience:** 5 Years Java experience



Doris is an experienced Java programmer and is assigned to work on a high-priority project that uses Oracle NoSQL Database. Doris, along with a team of developers, will modify an existing application, Earnback application, developed using RDBMS as the backend. Doris will be involved in designing and implementing some new requirements for the Earnback application.

## **Training required:**

Doris needs to learn how to program an application for Oracle NoSQL Database.

## **Doris's Technical Skill Set**

Java programming

IDE (NetBeans/Eclipse)

JDBC

Java UI (FX/Swings)

Java EE technologies

App Servers (weblogic/glassfish)

Other web technologies

System Administration

**ORACLE**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Course Supplement

- Access the course supplement from  
`/home/oracle/labs/dev/course_sup`
- The course supplement contains course assessments.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You will need to complete the assessments for each unit within the course supplement itself.

## Additional Resources

- Information about Oracle NoSQL Database:
  - <http://www.oracle.com/technetwork/database/nosqldb/overview/index.html>
  - Oracle NoSQL Database [FAQs](#)
- Information about Oracle Big Data Appliance:
  - <http://www.oracle.com/technetwork/server-storage/engineered-systems/bigdata-appliance/documentation/index.html>



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Summary

In this lesson, you should have learned to:

- Describe the course goals
- Identify the course environment

## **Unit I**

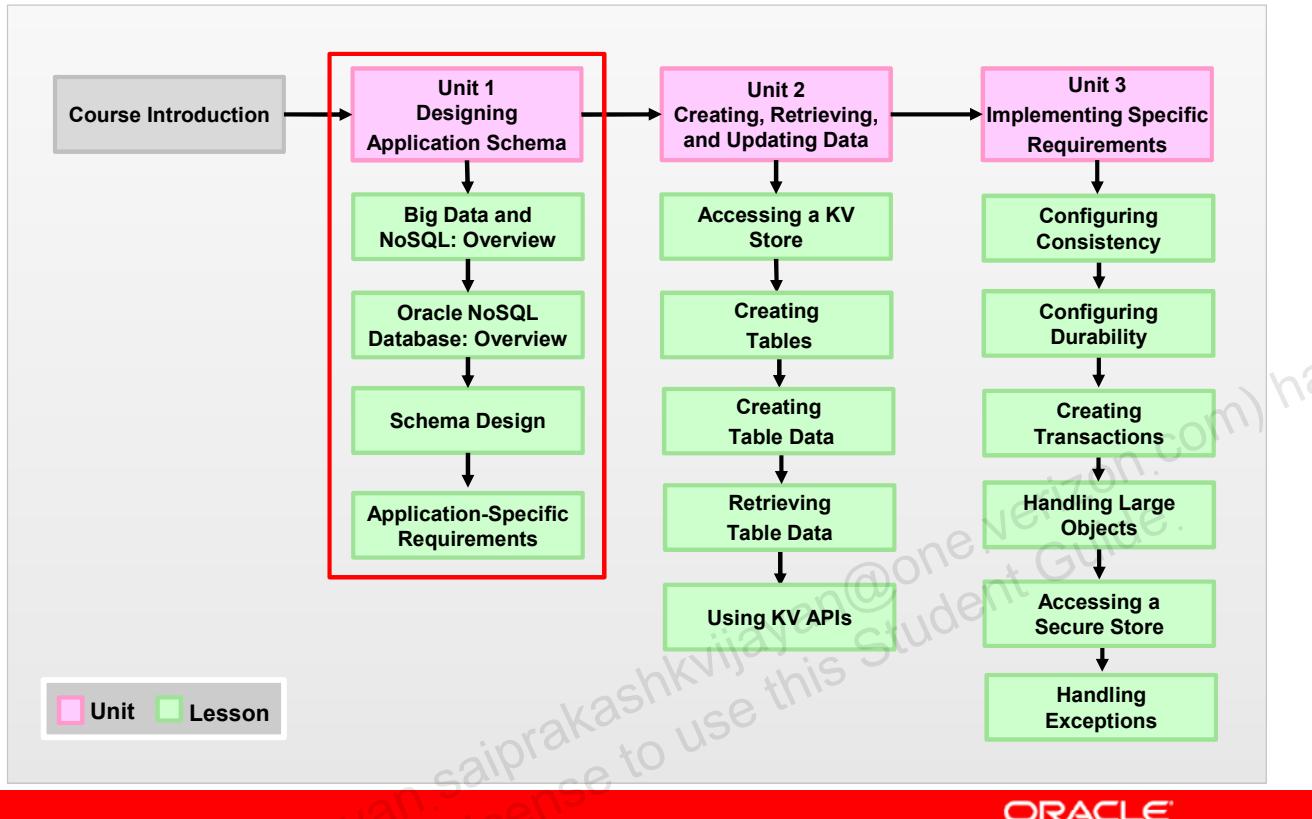
# **Designing Application Schema**



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Course Outline



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Doris Designs the Applications



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

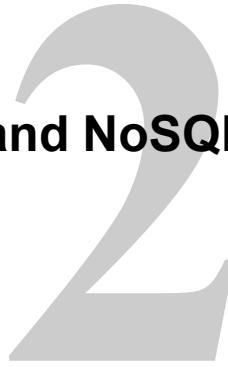
Doris has read the Earnback application specifications and is excited about implementing the new features using a new technology. She has researched on the Big Data and NoSQL technologies and come up with some notes to share with her team. Her next task is to identify how to store the data in the Oracle NoSQL Database in order to implement the required new features in the application.

In this unit, you review Doris's notes on Oracle NoSQL Database and also learn how to design a schema for Oracle NoSQL Database.

Unauthorized reproduction or distribution prohibited. Copyright© 2016, Oracle and/or its affiliates.

Sai Vijayan S (sai.vijayan.saiprakashkvijayan@one.verizon.com) has  
a non-transferable license to use this Student Guide.

## Big Data and NoSQL: Overview



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Define the term *big data*
- Differentiate between a NoSQL database and a relational database management system (RDBMS)
- Define Hadoop Distributed File System (HDFS)
- Differentiate between a NoSQL database and HDFS
- Identify when to use a NoSQL database



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In this lesson, you learn about the term *NoSQL* and its role in the world of “big data.” You also learn when to use a NoSQL database, as well as when not to use it.

## Doris Prepares for Meeting with Team

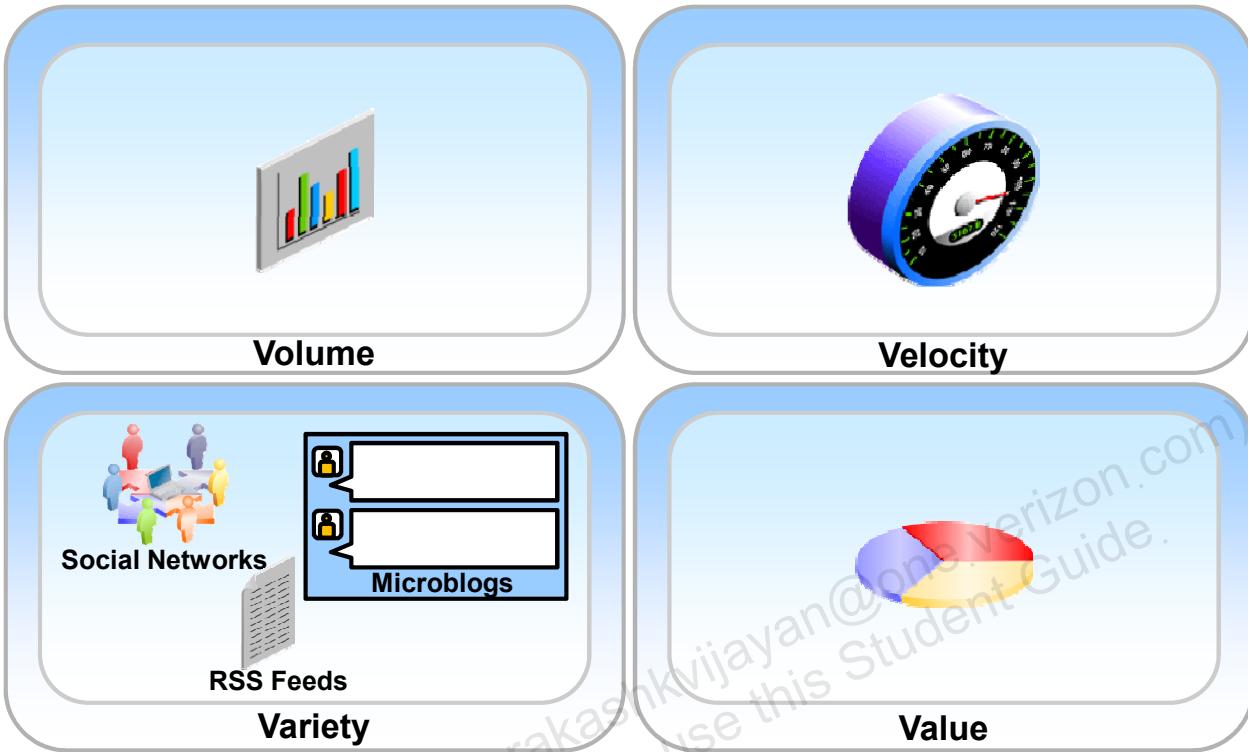


ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris is meeting with her development team to discuss the new features for the Earnback Application. She needs to brief the team about the new Oracle NoSQL Database technology the company is implementing. Doris is certain that her team will have some very specific questions regarding this new technology. This lesson introduces NoSQL and Big Data. In the practices for this lesson, you will help Doris answer her team's queries.

# Defining Big Data



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

What data can be defined as *big data*? As the term itself suggests, data sets that are very large in size are generally called *big data*. The slide shows the main characteristics of big data (referred to as “the four Vs”): volume, velocity, variety, and value.

You might be familiar with the first two characteristics. They mean that the data grows tremendously in volume with rapid velocity. The last two characteristics are unique to big data.

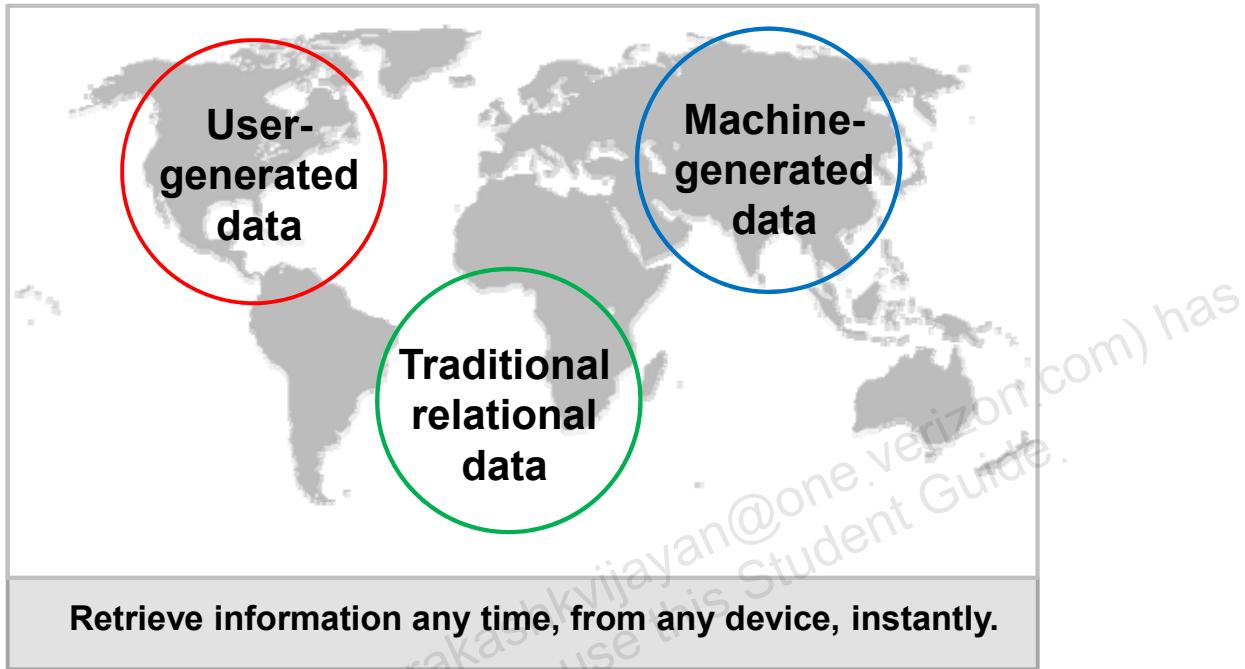
*Variety* means that the data sets for big data can be from different sources. This makes it difficult for these data sets to be stored in traditional relational databases. Because big data does not have a defined structure, it needs to be handled differently.

*Value* means that out of all the big data that is generated from these various sources only a little data is of value to drive business decisions. That is, a piece of information in big data is not valuable on its own, but it becomes valuable in the aggregate.

Why is big data valuable to business? If you can capture and analyze the various data that surround a business, you can make better business decisions.

Because traditional databases could not handle these volumes of data and process them instantly, there was a need for a different approach to storing data.

# The Changing Data World



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The current trend in the data world is capturing data from various sources. Namely, you have user-generated data such as social media data and images; machine-generated data such as log data, sensor data, and geospatial data; as well as the traditional relational data such as customer records and transactions, inventory, and ERP reports. Along with storing a wide variety of data, there is a demand for information delivery as well. You want to receive information at any time from any device instantly.

To support this trend, a new data center is required that is diverse and open and flexible to data changes and requirements. You should be able to scale out faster, accelerate application development, provide real-time interactions, and all this in a cost effective manner. NoSQL databases help in achieving this.

# What Is a NoSQL Database?

- Stores nonrelational data
- Uses one of the following data models:
  - Key-value
  - Columnar
  - Document
  - Graph
- Examples of NoSQL databases:
  - Oracle NoSQL
  - Cassandra
  - Voldemort
  - MongoDB



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

A NoSQL database is a nonrelational database that does not store information in the traditional relational format. There is no defined schema for the data.

The term *NoSQL* is an abbreviation of “Not Only SQL.” The slide lists some examples of NoSQL databases.

There are four data models for NoSQL databases:

- **Key-value:** This is the simplest data model for unstructured data. It is highly efficient and highly flexible. The drawback of this model is that the data is not self-describing.
- **Columnar:** This data model is good for sparse data sets, grouped subcolumns, and aggregated columns.
- **Document:** This data model is good for XML repositories and self-describing objects. However, storage in this model can be inefficient.
- **Graph:** This is a relatively new model that is good for relationship traversal. It is not efficient for general searches.

In this course, you learn about Oracle NoSQL Database, which belongs to the key-value data model category.

## RDBMS Versus NoSQL

RDBMS	NoSQL
High-value, high-density, complex data	Low-value, low-density, simple data
Complex data relationships	Very simple relationships
Joins	Avoids joins
Schema-centric, structured data	Unstructured or semi-structured data
Designed to scale up (not out)	Distributed storage and processing
Well-defined standards	Standards not yet evolved
Database-centric	Application-centric and developer-centric
High security	Minimal or no security

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

NoSQL databases differ from the traditional RDBMS in many ways. The slide lists some of the key differences between these two databases.

# What Is HDFS?

## Hadoop Distributed File System:

- Is a component of Hadoop
- Comprises data nodes
- Stores data in blocks across different nodes
- Replicates data nodes to increase availability



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Hadoop Distributed File System is a component of the Hadoop project. Hadoop is a technology designed for very large-scale data operations and is written in Java.

HDFS comprises a number of data nodes. The data that needs to be stored in HDFS is broken into blocks and distributed across different data nodes. The data blocks are also replicated to increase availability.

It may look like HDFS offers the same features as Oracle NoSQL Database. However, there are significant differences, which are discussed later in this lesson.

## HDFS Versus NoSQL

HDFS	NoSQL
File system	Database
No inherent structure	Simple data structure
Batch-oriented workload	Real-time workload
Processes data to use	Delivers a service
Bulk storage	Fast access to specific records
Write once, read many	Read, write, delete, update



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

NoSQL databases differ from HDFS in certain ways. The slide lists some of the key differences between these two storage systems.

# Quiz

NoSQL databases will completely replace RDBMS in the future.

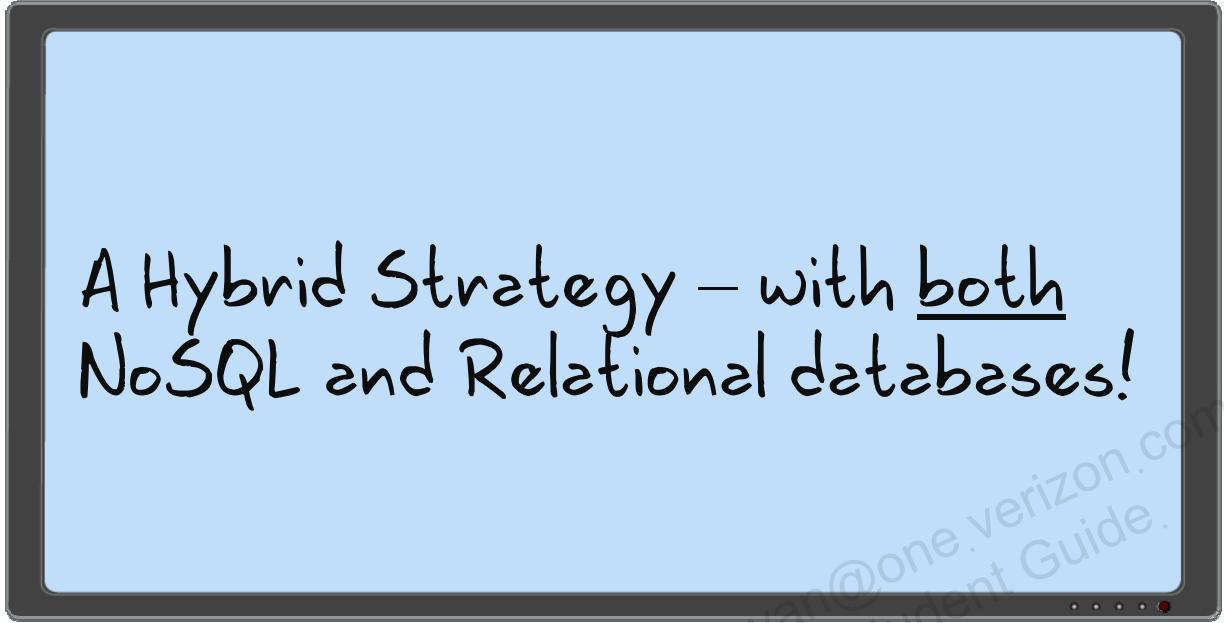
- a. True
- b. False



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: b**

## Requirement in the Big Data World Today



A Hybrid Strategy – with both NoSQL and Relational databases!

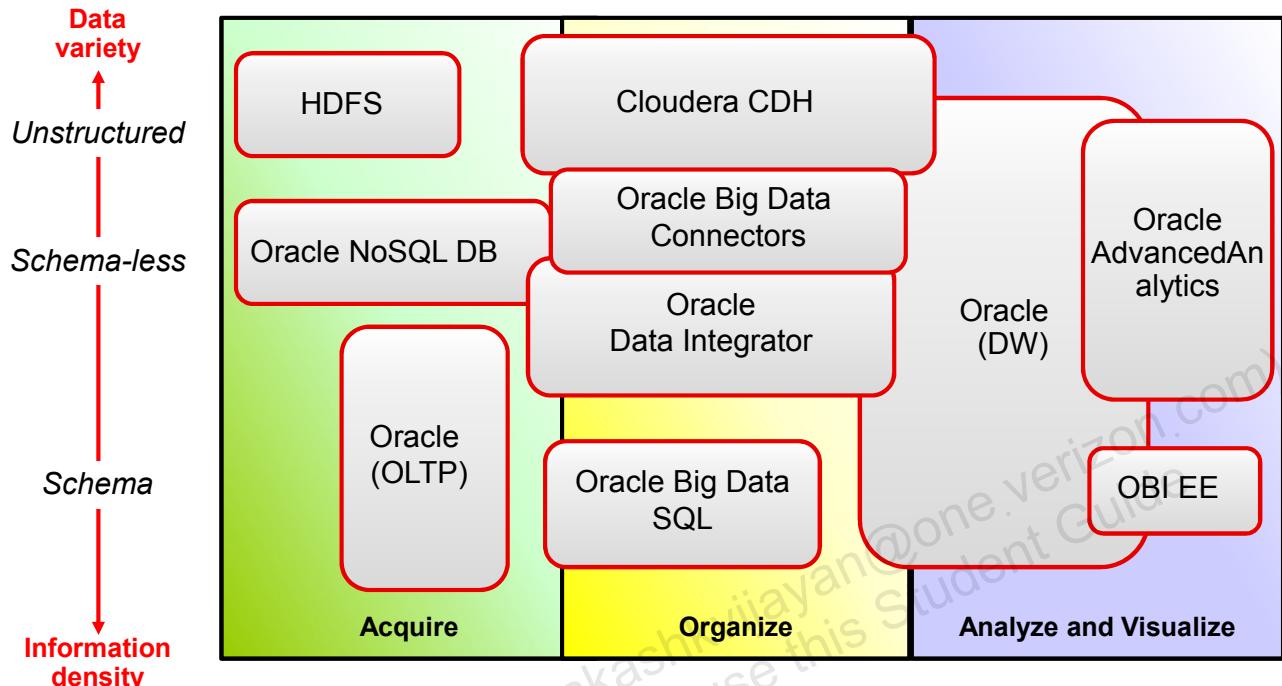
ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

There are a lot of innovations in database design. Years ago, you used to deploy relational databases, use a DBA, architect schemas, write database code to define the database interaction and build applications on top of the data models. This was a very secure approach, but it could not web scale.

The current need of enterprises is to scale to a massive deployment easily and quickly and to integrate data and code in a seamless manner. In order to achieve this, the database design strategy should be a hybrid model using the NoSQL and the relational databases.

# Oracle Integrated Software Solution



**ORACLE**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide is a high-level picture of the Oracle integrated software solution for conducting the Acquire, Organize, and Analyze phases of data conversion. Business decisions are derived from the analyzed data by using your choice of visualization tools.

- Cloudera CDH is a single, easy-to-install package from the Apache Hadoop core repository.
- Hadoop Distributed File System (HDFS) is a file system developed in Java, based on Google's GFS. It is used to store data on the cluster.
- Oracle NoSQL Database is the Oracle solution to acquiring and storing Big Data.
- Oracle Big Data Connectors are used to provide data communication between Oracle Big Data Appliance and Oracle Database.
- Oracle Data Integrator (ODI) extracts the unstructured data that is primarily stored in HDFS. It easily integrates data from any source.
- Oracle Big Data SQL provides powerful, high-performance SQL on Hadoop.
- Oracle Advanced Analytics is known for its statistical and advanced analytical functions using ORE. It also focuses on data mining, text mining, and predictive analytics using ODM.

In this course, you learn how to acquire big data by using Oracle NoSQL Database.

## Scenario 1: Retail Marketing System

In a new marketing strategy, you want to offer discount coupons to your customers when they are near your business. You will need to store:

- All details about a customer, such as name, date of birth, anniversaries, address, phone number, and so on
- All items a customer has purchased in the past
- Details of promotions and discounts the store is offering
- GPS signals from the customer's mobile device

***What technology would you recommend to build this application?***



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In the scenario described in the slide, you want to send discount coupons to your customers as they approach your business. You should process all information and send the coupons as soon as the customer is near the business. If you send the coupon when the customer has finished shopping and has already reached the parking lot, it is too late. Only when the customer is approaching the business does the information become valuable.

You will need to store many details (for example, the GPS feed). You also need to store the promotions and offers that your business is featuring—which might change on a daily basis. Given these requirements, you should choose a NoSQL database as the best storage option for this application.

## Scenario 2: Human Resources System

A multinational company needs a centralized human resources system with the following requirements:

- It should store the information of all employees in the organization (current as well as former employees).
- For each employee, it should store information such as date of hire, family details, job history, health history, benefits received from the company, and date of resignation or retirement.
- It should be able to store scans of important documents, employee fingerprints, voice samples, and so on.
- Company benefits and policies applying to an employee might change from time to time.

***What technology would you recommend to build this application?***



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This scenario is an example of high-value and confidential information. An RDBMS database should be used to acquire the data.

What are some of the considerations that helped to select the back-end storage technology for the application?

- **Data value:** The information that this application is required to handle is of very high value. Information about an employee is always considered to be confidential and should be securely stored.
- **Data structure:** Although this application needs to handle different kinds of data, you can still predict a structure for the information to be stored.

## Scenario 3: Healthcare System

A city needs a centralized healthcare system with the following requirements:

- It should store the health records of all the people in the city across all the different hospitals.
- Doctors should be able to use this system to understand the health history of patients.
- The system should be able to store different kinds of data, such as reports, monitoring signals, and so on.
- Data that needs to be stored might change over time.

***What technology would you recommend to build this application?***



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This scenario presents an example of big data. What are some of the considerations that can help in deciding the back-end storage technology for the application?

- **Data volume:** How big is the data that needs to be stored? In this scenario, suppose the city's population is 20 million. The resulting health information could be as much as 5.8 petabytes.
- **Real-time information:** The doctors should be able to retrieve information about a patient instantly. In many hospitals, doctors have less than five minutes to deal with each patient. So the response time should be immediate.
- **Data variety:** The application should be able to store any kind of information about the patient (X-ray reports, scans, laboratory results, doctor inputs, medical bills, and so on). Also, signals from heart monitoring devices or other monitoring devices need to be stored.
- **Data change:** Different hospitals will have different policies about the data that needs to be stored. These policies might change from time to time.

Considering the preceding points, a combination of a relational database and a NoSQL database should be used to acquire the data.

## Big Data: Examples

Data generated by an online movie rental application:

```
{"custId":1,"movieId":0,"activity":8"recommended":"Y","time":"2013-10-01:01:27:26"}  
{"custId":1,"movieId":0,"activity":6"recommended":"Y","time":"2013-10-01:01:27:26"}  
{"custId":1,"movieId":0,"activity":8"recommended":"Y","time":"2013-10-01:01:36:52"}  
{"custId":1,"movieId":0,"activity":8"recommended":"Y","time":"2013-10-01:01:38:53"}  
{"custId":1,"movieId":0,"activity":6"recommended":"Y","time":"2013-10-01:01:38:53"}  
{"custId":1,"genreId":7,"movieId":10386,"activity":5"recommended":"Y","time":"2013-10-01:01:39:18"}  
{"custId":1,"movieId":10386,"activity":11"recommended":"Y","time":"2013-10-01:01:40:15","price":1.99}  
{"custId":1,"movieId":10386,"activity":4"recommended":"Y","time":"2013-10-01:01:40:15"}  
{"custId":1,"movieId":10386,"activity":3"recommended":"Y","time":"2013-10-01:01:40:29","position":14}  
{"custId":1,"genreId":7,"movieId":10189,"activity":5"recommended":"Y","time":"2013-10-01:01:40:39"}  
{"custId":1,"movieId":10189,"activity":4"recommended":"Y","time":"2013-10-01:01:40:45"}  
{"custId":1,"movieId":10189,"activity":11"recommended":"Y","time":"2013-10-01:01:40:45","price":1.99}  
{"custId":1,"movieId":10189,"activity":3"recommended":"Y","time":"2013-10-01:01:40:45","position":1}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows some sample data generated by an online movie rental application. This is how big data is usually received by an application. There are various other types of data generated such as data streams from sensor devices.

## Summary

In this lesson, you should have learned how to:

- Define the term *big data*
- Differentiate between a NoSQL database and a relational database management system (RDBMS)
- Define Hadoop Distributed File System (HDFS)
- Differentiate between a NoSQL database and HDFS
- Identify when to use a NoSQL database



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Practice 2 Overview: Big Data and NoSQL

This practice covers the following topics:

- Defining *big data*
- Differentiating between a NoSQL database and an RDBMS
- Differentiating between a NoSQL database and HDFS
- Knowing when to use a NoSQL database



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This is a paper-based practice to check your understanding of the concepts covered in this lesson.

## Oracle NoSQL Database: Overview

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

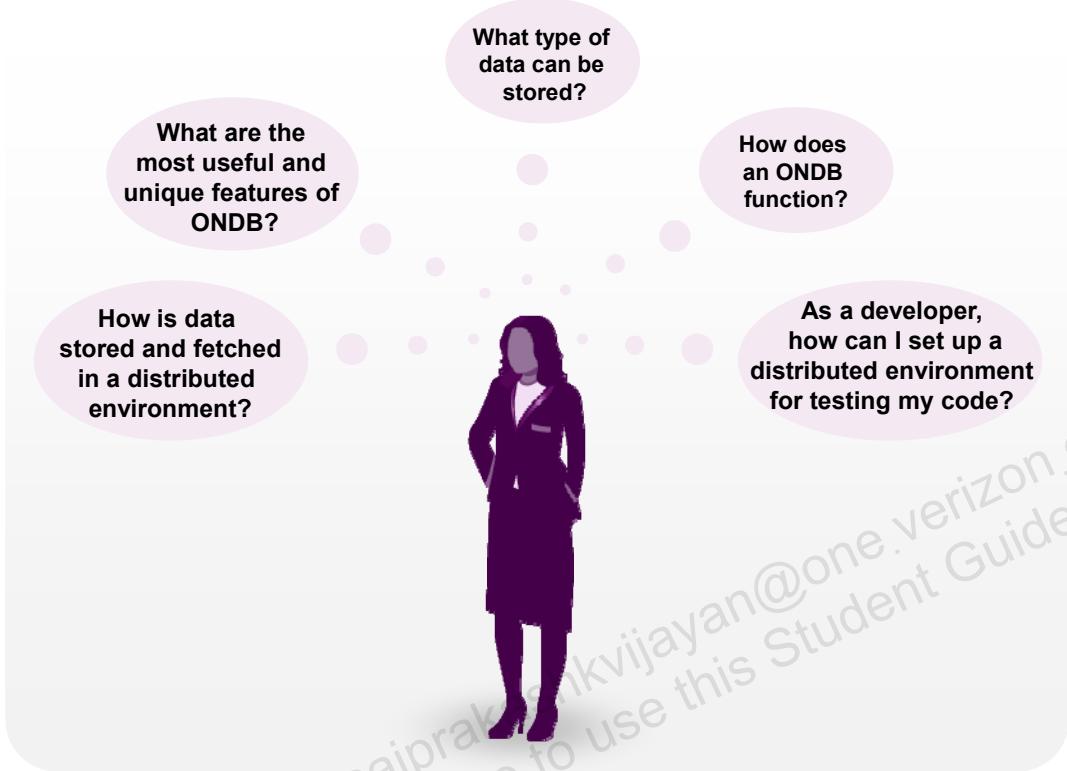
After completing this lesson, you should be able to:

- Identify the key features of Oracle NoSQL Database
- Explain Oracle NoSQL Database architecture
- Identify the components of Oracle NoSQL Database
- Identify when to use Oracle NoSQL Database
- Determine the structure for storing data in Oracle NoSQL Database
- Use KVLite



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Doris Explores Oracle NoSQL Database



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris wants to understand the working details of Oracle NoSQL Database before designing the new features for the Earnback application. She wants to make sure she understands all the aspects of Oracle NoSQL Database that should be identified before she and her team start designing and implementing the new features. Some specific questions in her mind are listed in the slide.

# Oracle NoSQL Database

## Oracle NoSQL Database:

- Is a key-value database
- Is accessible using Java APIs
- Is built on Oracle Berkeley DB Java Edition
- Stores data as byte arrays
- Is the Oracle solution to acquiring big data



### Benefits

- Easy to install and configure
- Highly reliable
- General-purpose database system
- Scalable throughput and predictable latency
- Configurable consistency and durability
- Web console for administration

**ORACLE®**

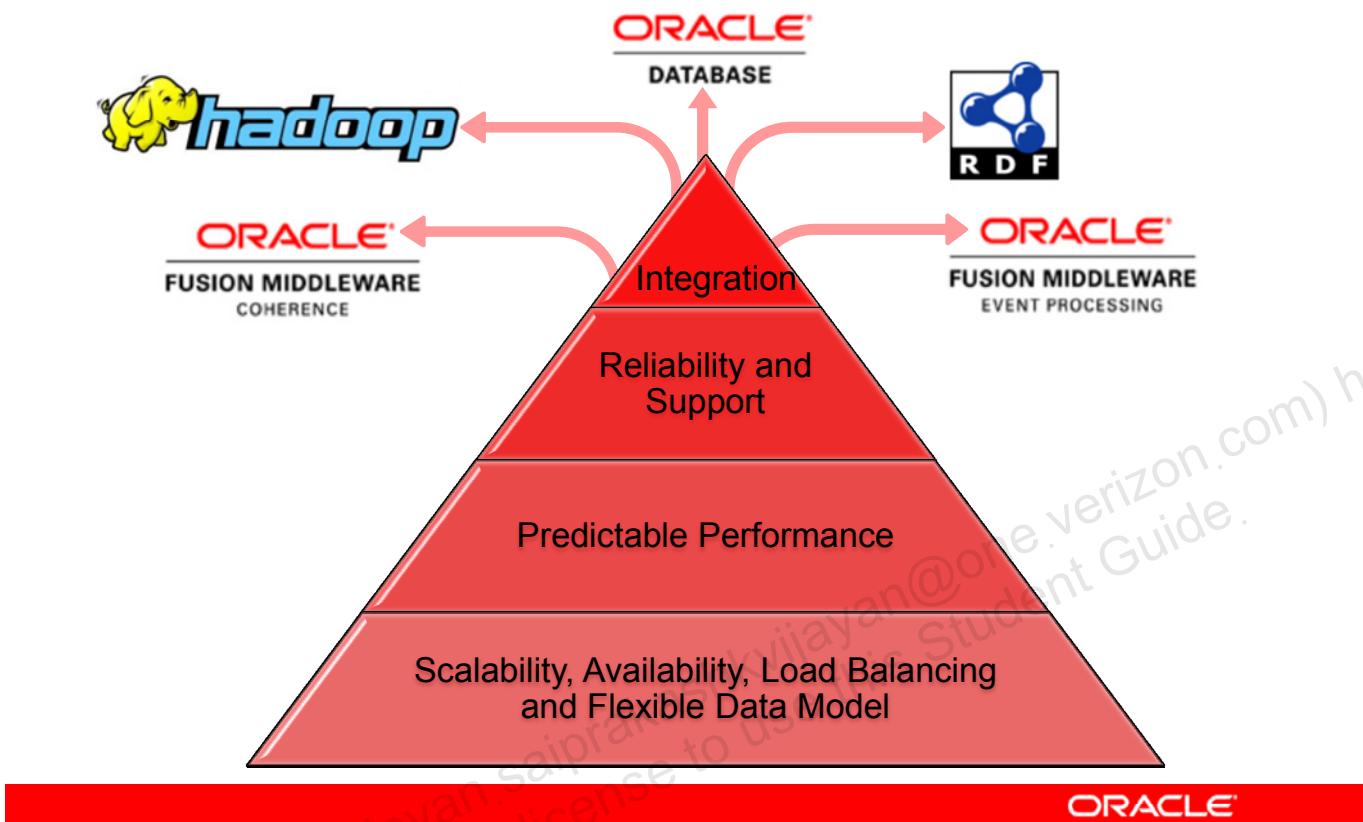
Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Oracle NoSQL Database (ONDB) is a nonrelational database. It is written in Java and uses the key-value data model to store data. Oracle NoSQL Database has evolved from Oracle Berkeley DB Java Edition, which is a library that provides fast performance for key-value data. To learn more about Oracle Berkeley DB Java Edition, refer to the product page on Oracle Technology Network (OTN).

To access the data stored in Oracle NoSQL Database, you must use Java-based APIs in an application. ONDB stores data as byte arrays. You can store any type of data in ONDB as long as you can convert it into the byte format.

Oracle NoSQL Database is the Oracle solution to acquiring and storing big data. The slide lists the benefits of using Oracle NoSQL Database. You learn about consistency and durability later in this unit. In the next few slides, you learn about ONDB features and components.

## Key Features

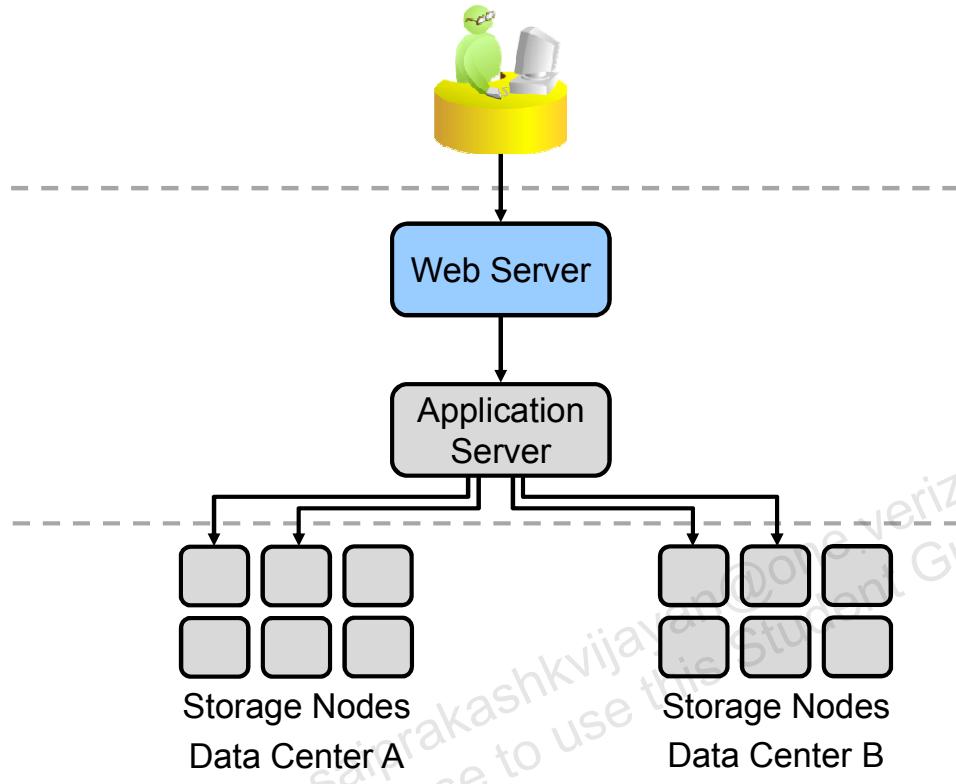


Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Oracle NoSQL Database is designed to provide the following key features:

- **Integration:** ONDB's ability to integrate with other Oracle's enterprise solutions distinguishes it from other NoSQL solutions. You can query ONDB from Oracle Database using external tables. You can access ONDB from Hadoop. You can store and query RDF data. You can cache event streams as well as share data for extensible in-memory cache grid.
- **Reliability and support:** ONDB provides reliable commercial grade software and offers excellent support for both the enterprise and community editions.
- **Predictable performance:** ONDB's performance is predictable as you scale out. The throughput increases linearly and the latencies are predictable.
- **Scalability:** When the volume of data increases, you can add additional storage nodes to store the data conveniently and quickly.
- **High availability:** You can configure Oracle NoSQL Database to have one or more replicas for each storage node. This guarantees high availability and no single point of failure.
- **Transparent load balancing:** Oracle NoSQL Database has an intelligent driver that is attached to your application. Depending on the load, this driver automatically distributes the requests among the master and replica nodes.
- **Flexible data model:** The data is stored in ONDB using the key-value data model. This is the simplest and most efficient data model for storing unstructured data.

# How Oracle NoSQL Database Works



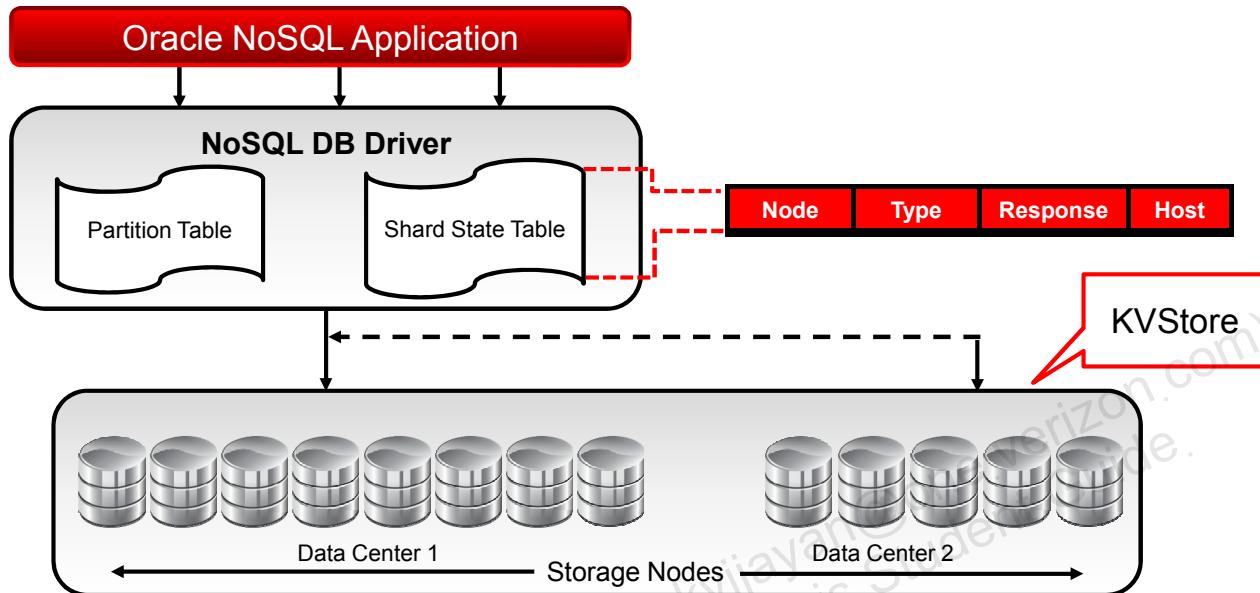
ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Consider a typical web application scenario in which the application services requests across the traditional three-tier architecture: web server, application server, and database server. In this scenario, Oracle NoSQL Database is installed behind the application server. Oracle NoSQL Database either takes the place of the back-end database server or runs alongside the back-end database.

Oracle NoSQL Database is installed in a set of storage nodes that may be located in different data centers.

# Architecture



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

An Oracle NoSQL Database consists of two parts: a NoSQL DB Driver and a collection of storage nodes called KVStore.

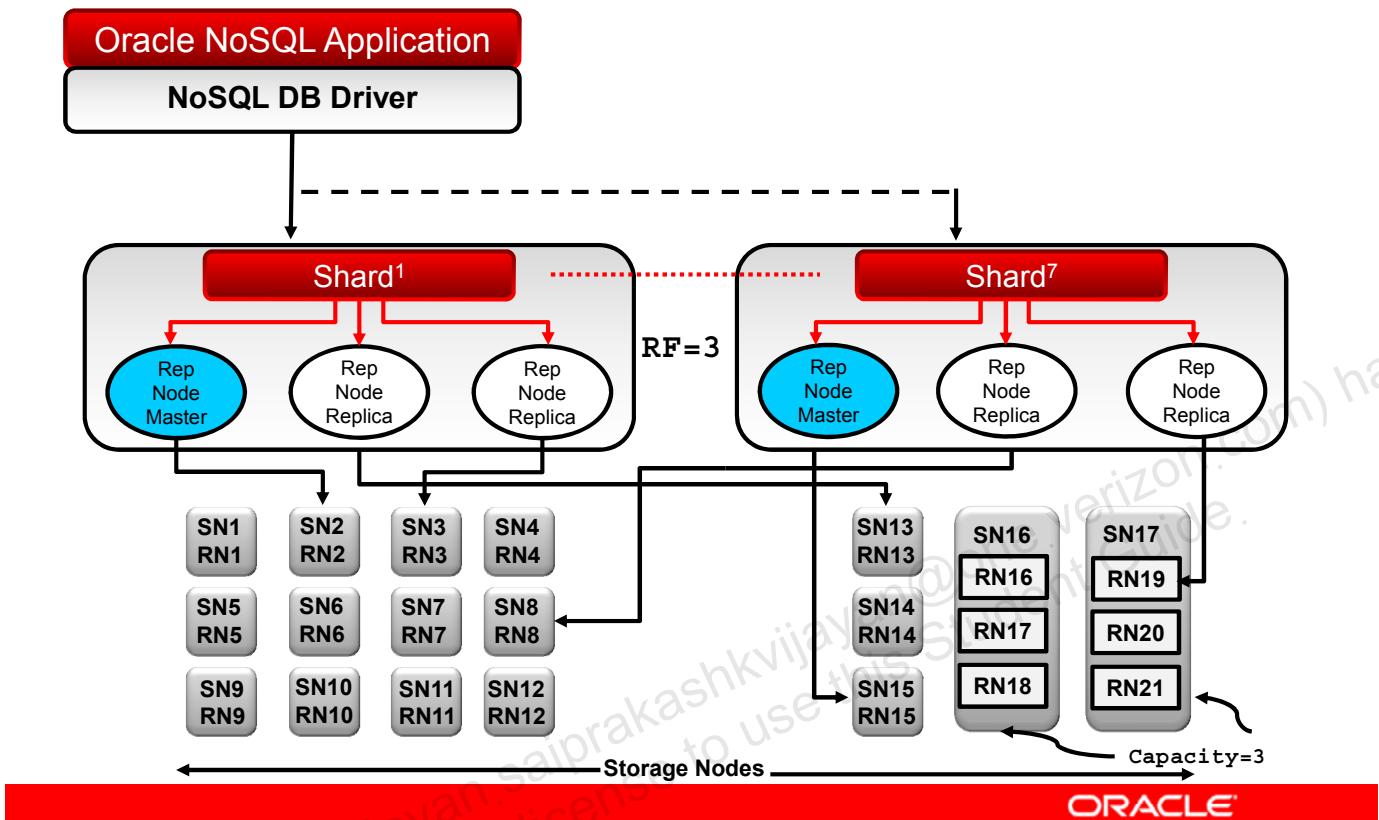
The NoSQL DB driver is an intelligent driver that transparently handles all the core operations of the ONDB.

- It transfers the read and write operations from the application to the KVStore.
- It evenly distributes the data across the several storage nodes in a KVStore by using a hash function-based data partitioning and distribution method.
- It transparently balances the load of read operations across replica nodes in the KVStore.
- It automatically directs the write operations to the appropriate master nodes in the KVStore.

To accomplish the preceding tasks, the NoSQL DB driver maintains a copy of the store topology (partition table) and a shard state table (SST). The topology maps a key to its partition and a partition to its shard. The SST stores data such as the host name of the storage node hosting each replication node in the group, the service name associated with the replication nodes and the data center in which each storage node resides. The NoSQL DB driver uses the SST for identifying the master node of a shard (for write operations) and for balancing load across all the nodes in a shard (for read requests).

The KVStore consists of storage nodes. You will learn about the components of a KVStore in the next slide.

# KVStore Components



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The main component of Oracle NoSQL Database is the Key-Value Store (KVStore). The KVStore is a collection of storage nodes that can be distributed across different data centers or zones. A zone can be physically located at a different location than other zones. There are two types of zones: primary and secondary. Primary zones are the default type and contain both master and replica replication nodes. Secondary zones contain only replica nodes and are used to increase the availability of the KVStore. The slide shows a KVStore with a number of distributed storage nodes.

A storage node (SN) is a physical or virtual machine with its own local storage. It is recommended that all the storage nodes in a KVStore be identical. A storage node hosts one or more replication nodes (RN). For better performance, it is recommended that each storage node host only one replication node. In the slide example, storage nodes 1 through 15 are considered to contain one replication node each. Storage nodes 16 and 17 have capacity = 3 and host three replication nodes each.

A replication node is the place where the key-value pairs of data are stored. The replication nodes are organized into shards (replication groups). Each shard contains a master node and one or more replica nodes. The master node in a shard handles all the database write operations and keeps the replica nodes updated. The replica nodes handle all the database read operations. In the slide example, the KVStore has seven shards and a replication factor of 3.

A shard is divided into partitions; each partition holds a key or a subset of keys. After a key is placed in a partition, it cannot be moved to a different partition. There are tools (shipped with the product) that enable you to plan how many partitions you need depending on your workload requirements.

To enable your application to communicate with the KVStore, you must link an Oracle NoSQL Database driver to your application. This driver is a Java library that you access from your application by using Java APIs.

# Quiz

Oracle NoSQL Database is just Oracle Berkeley DB Java Edition with some extra items added.

- a. True
- b. False

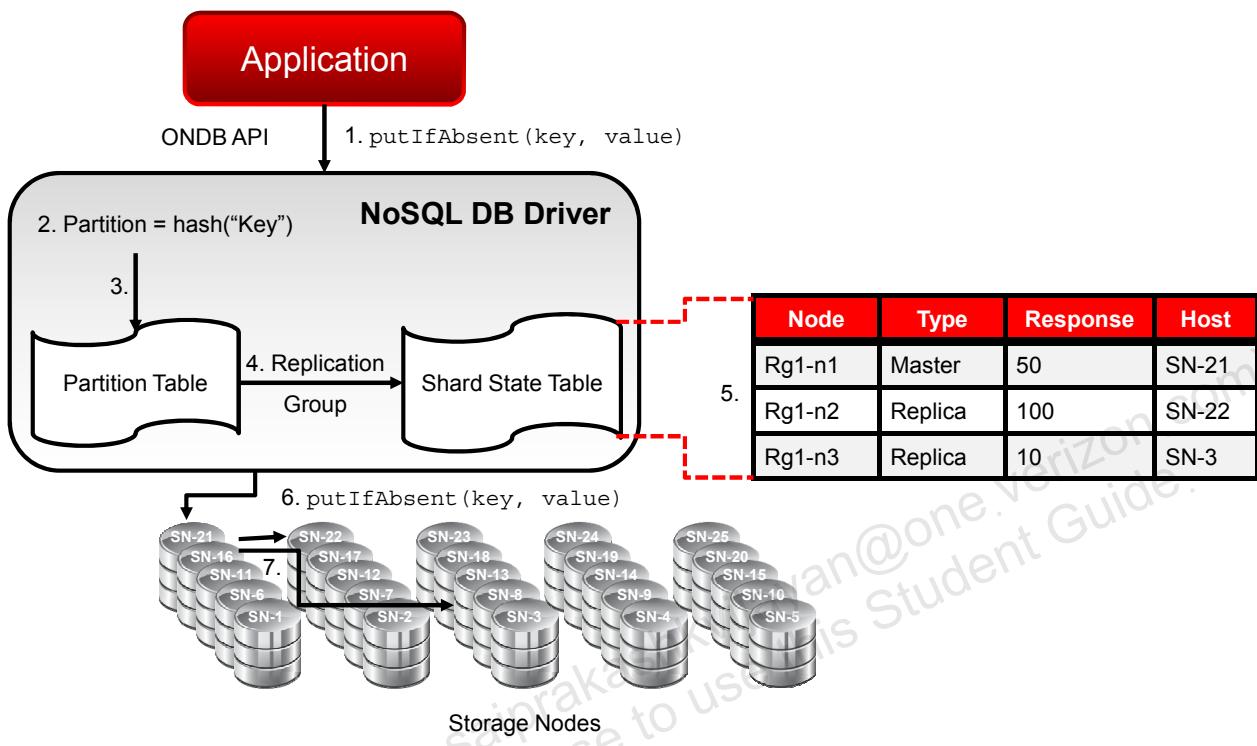


Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Answer: b

Although Oracle NoSQL Database is built using Oracle Berkeley DB Java Edition as the underlying, experience-tested storage system, Oracle NoSQL Database adds a large amount of infrastructure on top of it to bring it into the NoSQL realm.

## Request Processing: Example



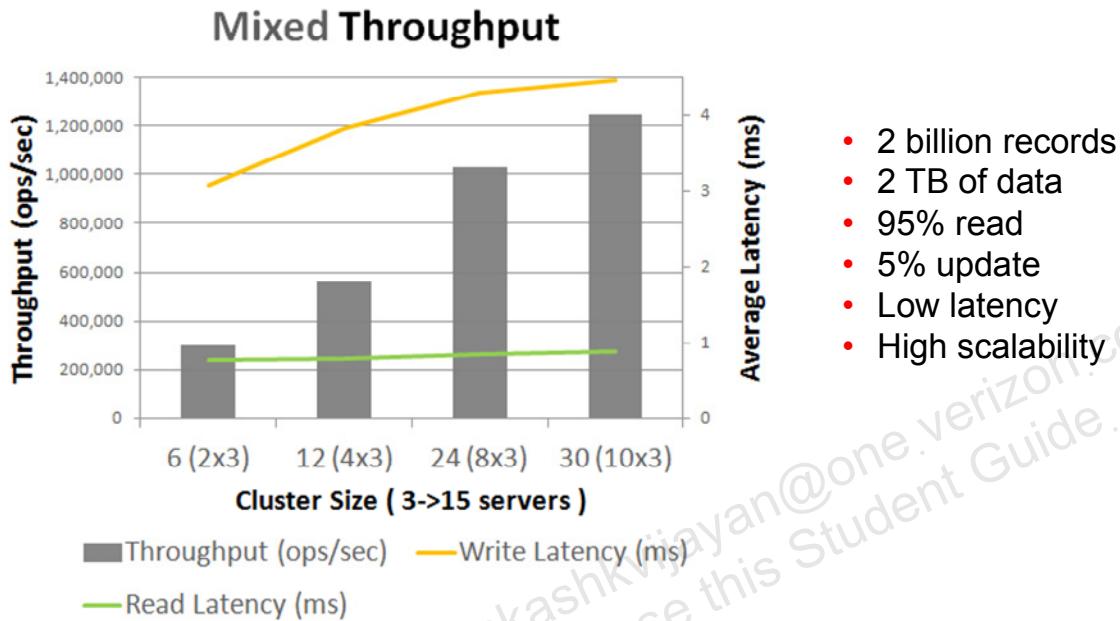
**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The Oracle NoSQL Database architecture processes a request in a logical sequence through the components of the system. In this example, you will see how the key and value of a key-value pair are created.

1. The application issues a `putIfAbsent` method to the NoSQL DB Driver.
2. The driver hashes the “key” to select one of a fixed number of partitions.
3. The driver consults the partition table to map the partition number to a shard.
4. The driver next consults the Shard State Table (SST).
5. The SST contains information about each replication node in the group.
6. The driver analyses the request and based on the information in the SST forwards the request to the appropriate node. In this case, because it is a write request, the request must go to a master node.
7. The replication node then performs the operation. If the key does not exist, the `putIfAbsent` method adds the key-value pair to the store and propagates the new key-value pair to the replica nodes. If the key exists, the operation returns an error stating that the specified entry is already present in the store.

## Performance and Throughput



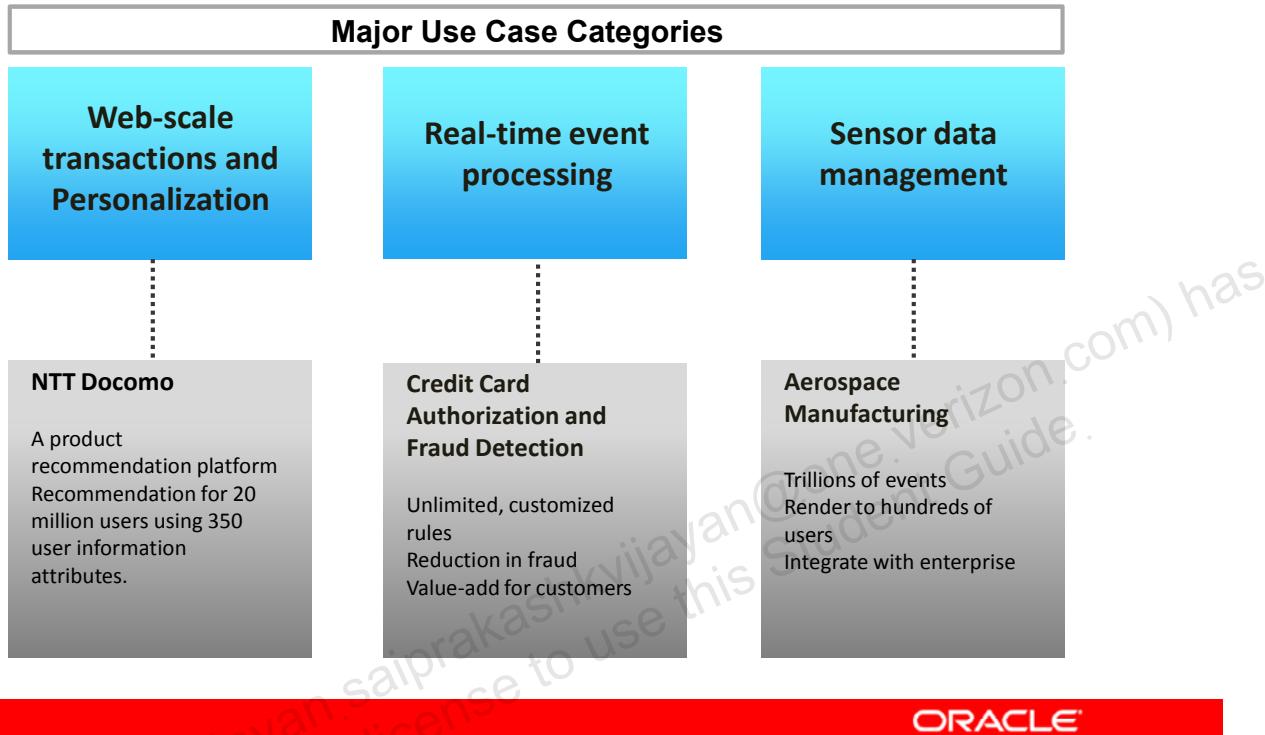
ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the Yahoo! Cloud Serving Benchmark statistics performed on SSD-backed (solid state drive) commodity servers. The graph shows that as the cluster size is increased there is a linear increase in the throughput with almost consistent write latency and very little increase in the read latency.

This graph is based on two billion records of data resulting in a total volume of two terabytes of data with a workload of 95% read and 5% update. The result shows that Oracle NoSQL Database has high scalability with low latency.

# Oracle NoSQL Database Use Cases



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

What kind of applications are using Oracle NoSQL Database? The applications can be summarized into three main categories:

**Web-scale transactions and personalization:** With the growth of e-commerce, more and more businesses are looking at providing an individualized and personal experience to their customers. Users' entire online experience is defined by who they are and how they use the application. This information is stored as a user profile in the ONDB and is used to personalize the user's online experience. Applications should also be able to scale up quickly to handle the large volumes of data as the business grows. NTT Docomo is an Oracle customer using Oracle NoSQL Database to store user attributes and various rules used to recommend the right services to its users.

**Real-time event processing:** Oracle NoSQL Database provides event processing systems with fast retrieval of data when an event occurs. An example of an application using ONDB for real-time event processing is a huge credit cards bureau that authorizes card transactions and detects fraudulent usage. Passoker is another Oracle NoSQL Database customer who wanted to revolutionize the online gaming industry. They created a platform where customers could place bets on events they are interested in during the course of the game itself.

**Sensor data management:** The challenge in these type of applications is to store huge amounts of sensor data in such a way that you can quickly retrieve subsets of data in order to analyze and evaluate the sensor data. An example is an aerospace manufacturing unit that uses Oracle NoSQL Database to capture sensor data.

## When to Use Oracle NoSQL Database

Functionality	Oracle NoSQL Database Not Required	Oracle NoSQL Database Required
Scale Out vs Scale Up	<ul style="list-style-type: none"><li>Application does not require multiple machines (scale out).</li><li>Scale up is sufficient to meet future growth.</li></ul>	<ul style="list-style-type: none"><li>Application requires distributed machines (scale out).</li><li>Application requires on-demand scaling by adding new hardware capacity.</li></ul>
Data velocity, low latency and distribution	<ul style="list-style-type: none"><li>No high-velocity data – no web streaming, no sensors, no devices</li><li>High write throughput is not critical.</li><li>Data does not require geographic distribution.</li></ul>	<ul style="list-style-type: none"><li>High throughput with low latency reads and writes</li><li>Distributed geographical servers</li><li>Application requires predictable response time for data access.</li></ul>



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows when to use Oracle NoSQL Database and when not to use it, keeping in mind the scaling and latency requirements.

## When to Use Oracle NoSQL Database

Functionality	Oracle NoSQL Database Not Required	Oracle NoSQL Database Required
Online Data Volume	<ul style="list-style-type: none"> <li>Applications do not need to keep large data volumes online.</li> <li>Older data is continuously purged.</li> </ul>	<ul style="list-style-type: none"> <li>Application requires access to large historic data (regulatory requirements).</li> <li>Require low latency access to both HOT (current) and COLD (historic) data</li> </ul>
Up Time Requirements	<ul style="list-style-type: none"> <li>Applications do not need to be available 24/7.</li> <li>Unplanned down time can be handled by traditional failover to backup.</li> </ul>	<ul style="list-style-type: none"> <li>Applications required 24/7 up time.</li> <li>Standard hot backups are not sufficient.</li> <li>Applications need transparent failover.</li> <li>Need to be able to add hardware capacity to the system while serving online operations</li> </ul>



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows when to use Oracle NoSQL Database and when not to use it, keeping in mind the online data volume and application up time requirements.

## When to Use Oracle NoSQL Database

Functionality	Oracle NoSQL Database Not Required	Oracle NoSQL Database Required
Data Variety	<ul style="list-style-type: none"><li>Applications have only tabular format data (rows and columns).</li><li>No social media, no text or semi-structured sources</li><li>Schemas change infrequently.</li><li>Data is transactional and queries are complicated.</li></ul>	<ul style="list-style-type: none"><li>Managing multiple types of data (structured, semi-structured and unstructured)</li><li>Schemas and data formats change frequently.</li><li>Applications need to process events by INTEGRATING different data sources and formats.</li><li>Queries are simple.</li></ul>



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows when to use Oracle NoSQL Database and when not to use it, keeping in mind the data variety requirements.

# Accessing the KVStore

You access the KVStore for two different reasons.

- For access to stored data:
  - Use Java APIs.
- For administrative actions:
  - Use the command-line interface.
  - Use the graphical web console.



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To perform an operation on Oracle NoSQL Database data, you must access the KVStore. You can do this by using Java APIs for Oracle NoSQL Database in your applications. You learn to use some of the APIs in later lessons in this course.

You will also access the KVStore while performing administrative operations. You can use the command-line interface to perform administrative tasks. Alternatively, you can configure and monitor the KVStore by using the graphical web administration console. Installation and administration of ONDB is covered in the course titled *Oracle NoSQL Database for Administrators*.

## Schema: Overview

You can perform data modeling using:

- Table data model (recommended)
- Key-Value data model



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

ONDB stores NoSQL data in a Key-Value format. However, you can model the data directly using the Key-Value APIs or use the Table APIs to model the data in a Table format. The Table data model is the recommended approach to model data for ONDB because it allows you to create primary keys and secondary indexes.

In the lesson titled “Schema Design,” you learn more about ONDB schemas and data modeling.

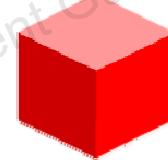
## KVLite: Introduction

KVLite is a simplified version of Oracle NoSQL Database.

- It provides a single storage node instance.
- It is not replicated.
- It is installed with Oracle NoSQL Database.
- It requires no configuration and runs as a single process.
- It is not intended for production deployment.

Why and when should you use KVLite?

- For a quick start with Oracle NoSQL Database
- To learn how to use the Java APIs
- To create and test applications



**ORACLE**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

KVLite is installed when Oracle NoSQL Database is installed. It runs as a single process in a single node and requires no configuration.

KVLite is available to help you quickly get started with Oracle NoSQL Database. It is not meant for production deployment purposes or performance measurement. You can start and stop KVLite by using a command-line interface (see the next slide).

## Starting and Stopping KVLite

```
> java -jar $KVHOME/lib/kvstore.jar kvlite  
-host <hostname>  
-admin <port>  
-port <port>  
-store <storename>  
-root <path>  
-usage
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To start KVLite, use the `kvlite` utility, which you run from the `KVHOME/lib/kvstore.jar` location. The location where you unzip the Oracle NoSQL Database software is called `KVHOME`.

The command-line options for this utility are:

- **-host:** Name of the host where KVLite is running. Default value is `localhost`.
- **-admin:** Port where administration process is started. Use this port to connect to the admin web console. Default port number is `5001`.
- **-port:** Port to listen for client connections. Default port number is `5000`.
- **-store:** Name of the new store. Default value is `kvstore`.
- **-root:** Path of `KVHOME`
- **-logging:** Turns on Java application logs
- **-usage:** Displays command Help

To stop KVLite, ensure that the terminal where KVLite is running is the active window, and then press `Ctrl + C`.

## Verifying That a KVStore Is Running

```
> java -jar $KVHOME/lib/kvstore.jar ping  
-host <hostname>  
-port <port>
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can verify that a KVStore is running by issuing a `ping` command. You must pass both of the following:

- The host name that you are trying to ping
- The port that is listening for the requests

## Restarting and Rerunning KVLite

- To restart KVLite, rerun the `kvlite` utility.
- To rerun KVLite:
  1. Delete the KVRoot directory.
  2. Run KVLite again with new options (if required).

**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

When KVLite has been stopped, you can either restart it with the same options that you specified earlier or delete the KVRoot directory and run KVLite again with new values.

## Summary

In this lesson, you should have learned how to:

- Identify the key features of Oracle NoSQL Database
- Explain Oracle NoSQL Database architecture
- Identify the components of Oracle NoSQL Database
- Identify when to use Oracle NoSQL Database
- Determine the structure for storing data in Oracle NoSQL Database
- Use KVLite



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Practice 3 Overview: Using KVLite

This practice covers the following hands-on using KVLite:

- Starting
- Verifying installation
- Stopping
- Restarting



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

For this hands-on practice, you will log in to the machine that is assigned to you by your instructor.

# Schema Design

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Describe the key-value data model
- Describe table data model
- Define major and minor keys
- Define primary and shard keys
- Describe how data is retrieved from the KVStore
- Describe design best practices



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Doris Discusses with Team

One important decision that we need to make is which of the two data models do we employ: table data model or key-value data model?

Table data model is meant for RDBMS folks. Because all of us are skilled in Java, I think we can stick to the key-value data model.

Do we design the schema differently for each data model?  
Or, can we choose the data model when we do the actual application coding when we implement the features?



Doris

Good question, I think we need to find that out.

We should probably review the pros and cons of both the design types before choosing one over the other.

I would like to know how Oracle NoSQL Database's Table model is different from the traditional table model for relational databases.



Doris' Team

**ORACLE**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris and her team have understood the features and working of Oracle NoSQL Database. They are now set to implement the application features. Review a part of the conversation between Doris and her team as given in the slide.

## Importance of Schema Design

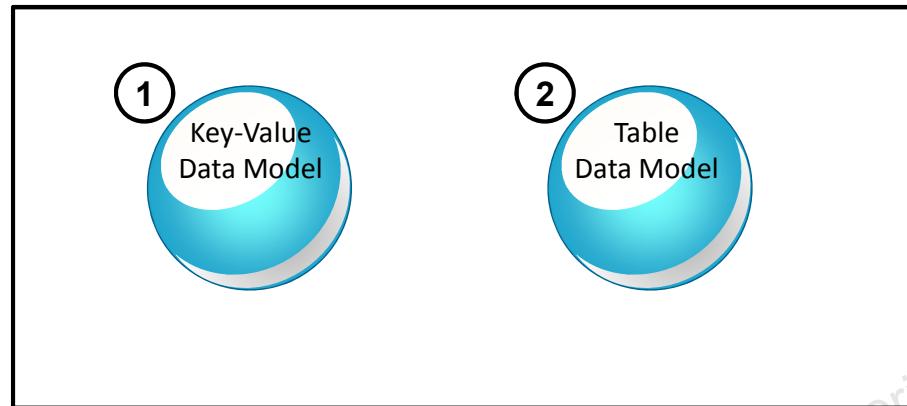
- Schema design impacts KVStore performance.
- A good data model:
  - Organizes data efficiently
  - Retrieves records efficiently
  - Prevents overhead
  - Uniquely locates a single record



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Before you develop an application for Oracle NoSQL Database, it is important to consider the schema structure of the records to be stored. A good data model can improve a KVStore's performance. Similarly, a poor data model can give poor KVStore performance. The structure of the data directly affects how it is stored and retrieved from a KVStore.

## Schema Design Options in ONDB



ORACLE®

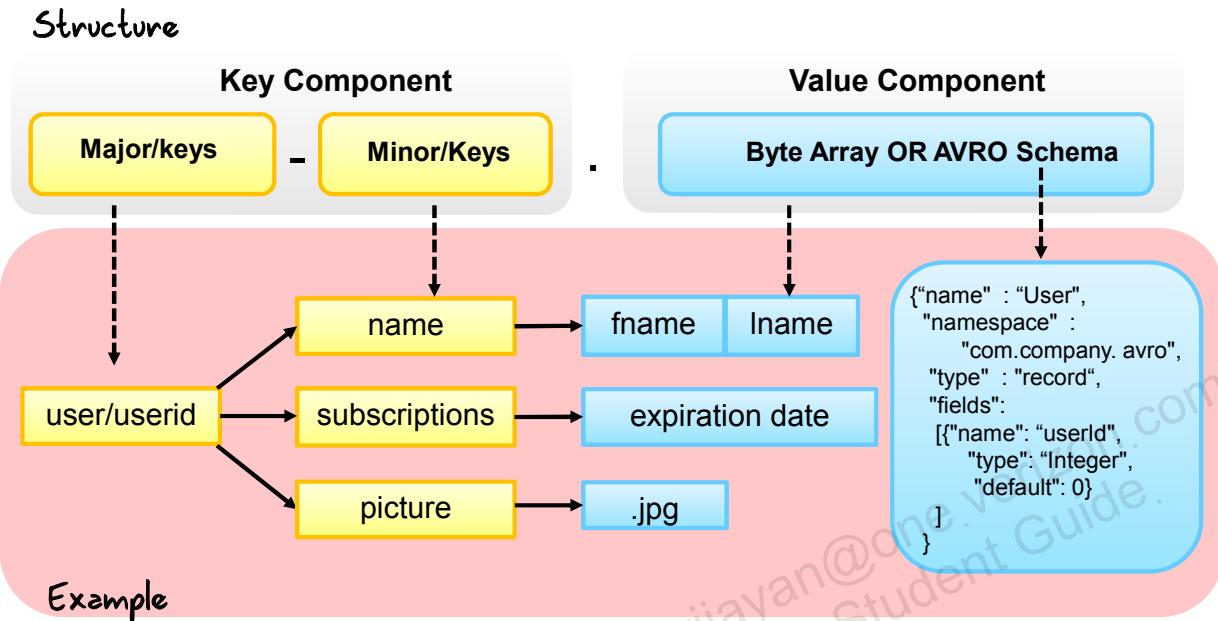
Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can create two data models for Oracle NoSQL Database:

- **Key-value data model:** Oracle NoSQL Database initially offered only the key-value model.
- **Table data model:** This model is one of the latest enhancements of the Oracle NoSQL Database product.

In this course, Table APIs will be the main focus for data storage and retrieval.

# Key-Value Data Model



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The schema structure of the key-value data model is very simple. Data in this model is not stored in fixed table-like structures. Each record consists of a key-value pair.

The key is used to uniquely identify a value in the KVStore. A key is a list of values of the String data type. You define the structure of a key within the application; all the logic to process the key and records is contained within the application itself. A key consists of major-key and minor-key components. It can have more than one major or minor component. However, all keys must have at least one major component. If a key has minor components, the combination of both the minor-key and major-key components uniquely identifies a record. In some situations, defining a key with only one major-key component is sufficient. For example, for an application tracking details of all employees in an organization, the employee ID can be defined as the key component. In some situations, you might want to define a person's name as the key component. In this case, the first name and last name of a person can be defined as the two major-key components.

It is recommended that you do not create all the records under a single major-key component. As the number of records grows in the store, performance problems can result. You can define minor-key components to further organize your data. When designing a key, remember that there is an overhead involved with each key that the store maintains. Defining too many key components might not be the best solution for the store's performance.

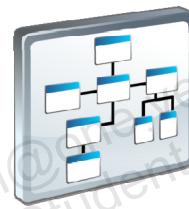
The value part of a record is the data that you want to store and manage using Oracle NoSQL Database. The value component can be any type of data. It is stored in Oracle NoSQL Database as a byte array. It can be of any data type as long as you can convert it to a byte array. Mapping of data structures, that is, serialization and de-serialization, should be taken care of in the application itself. The value component can be as large or as small as you want it to be. However, larger records take a longer time to read and write than shorter records. In the key-value data model, you can create the value component using AVRO to describe a schema for the value component. The AVRO schema describes the fields allowed in the value and their data types. You create and define an AVRO schema and then apply the schema to the value portion of a record using AVRO bindings.

The previous slide shows an example of a key-value pair. The key consists of two major keys (static text “user” and userid) and three minor keys (name, subscriptions, and picture). The value component can be raw byte values or can be stored using the AVRO schema.

The key-value data model is not self-describing. That is, there is no way to know what the various components of a key are and what data is stored as the value. The application developer using Oracle NoSQL Database should know the structure of the key and value fields.

## Table Data Model: Overview

- Data is stored in tables.
- Two types of tables:
  - Parent tables
  - Child tables



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In the table data model, you can store data in columns and rows of a table. There are two types of tables that you can create: parent tables and child tables. You can create as many child tables as you require. There is also no limit to the depth of nesting that can occur.

The table data model is eventually translated back to the key-value model. This is done internally and taken care of by Oracle NoSQL Database itself. The important point to note is that the more the number of tables, the more the keys that are created in the KVStore. This will have performance implications. So, you need to balance the number of tables you want to define in the KVStore.

# Designing Parent Tables

<Table Name 1>

Key 1	Field 1	Field 2	Field 3

<Table Name 2>

Key 1	Key 2	Field 1	Field 2	Field 3

<Table Name 3>

Key 1	Key 2	Key 3	Field 1	Field 2

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can define n number of parent tables in a KVStore. A parent table is defined with a table name, one or more primary keys, and n number of fields or values columns. Each column value is defined using a data type for that column.

## Table Field Data Types

- Simple data types
  - Boolean
  - Binary
  - Double
  - Enum
  - Fixed Binary
  - Float
  - Integer
  - Long
  - String
- Complex data types
  - Array
  - Map
  - Record



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Each column in a parent or child table is defined using a simple data type or a complex data type. The supported simple data types are boolean, binary, double, enum, fixed binary, float, integer, long, and string. The supported complex data types are array, record, and map.

## Keys and Indexes for Table Data Model

You can define two types of keys:

- Primary
- Shard

Alternative method to retrieve rows is by defining indexes:

- Secondary indexes



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In the ONDB table data model, you can create two types of keys: Primary and Shard. Every table must have one or more fields designated as the primary key. The primary key is defined when a table is created and cannot be changed later. A primary key must uniquely identify every row in the table. For example, a table might have five fields: `productName`, `productType`, `color`, `size`, and `inventoryCount`. To retrieve individual rows from the table, it might be enough to just know the product's name. In this case, you would set the primary key field as `productName` and then retrieve rows based on the product name that you want to examine. You can also define multiple fields as primary keys. This allows you to delete or retrieve multiple rows in your table in a single atomic operation.

For operations where atomicity is required, you can guarantee that certain rows are placed on the same shard by creating shard keys. Shard keys identify which primary key fields are meaningful in terms of shard storage. That is, rows that contain the same values for all the shard key fields are guaranteed to be stored on the same shard. Shard key fields must be a first-to-last subset of the primary key fields, and they must be specified in the same order as were the primary key fields.

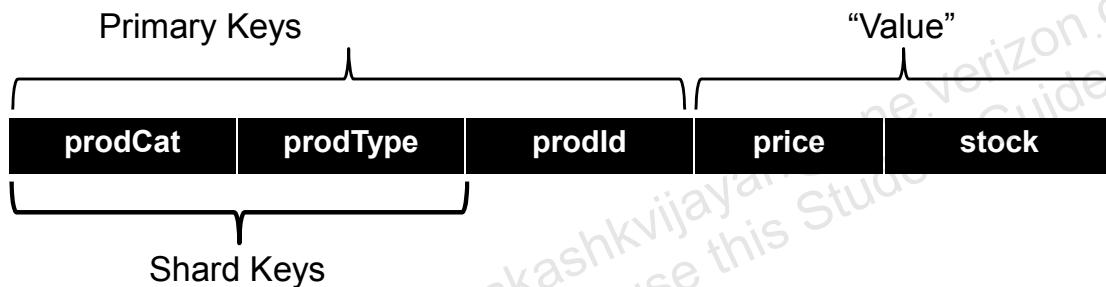
You can create indexes as an alternative way of retrieving table rows. By creating an index, you can retrieve rows with dissimilar primary-key values, but which share some other characteristic. For example, if you had a table representing types of automobiles, the primary keys for each row might be the automobile's manufacturer and model type. However, if you wanted to be able to query for all automobiles that are painted red, regardless of the manufacturer or model type, you could create an index on the table's field that contains color information. Indexes can take a long time to create because Oracle NoSQL Database must examine all of the data contained in the relevant table in your store. The smaller the data contained in the table, the faster your index creation will complete. Conversely, if a table contains a lot of data, then it can take a long time to create indexes for it.

## Parent Table: Examples

Table Name: User



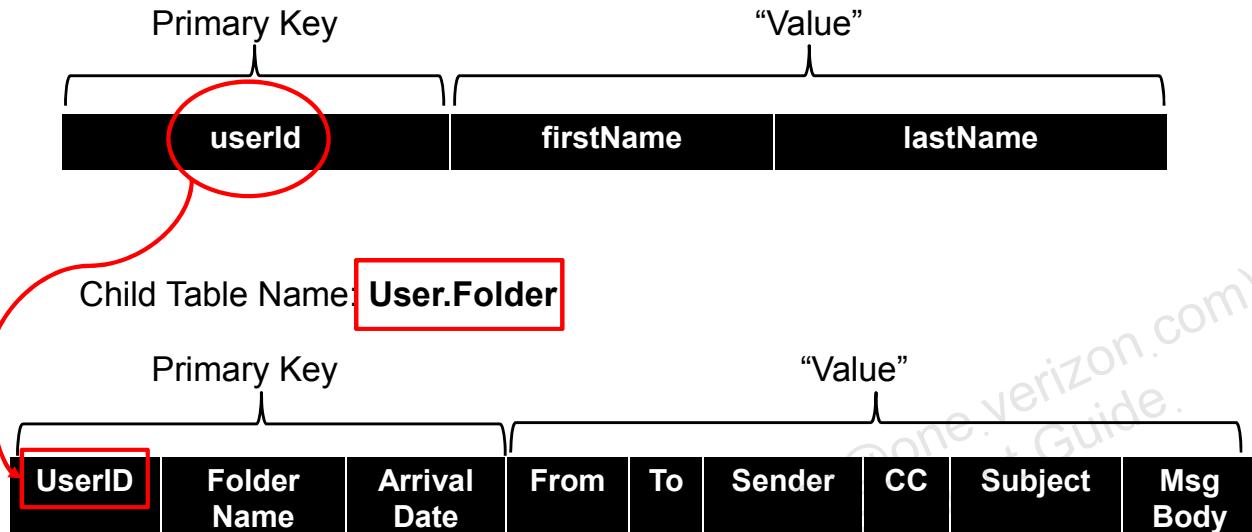
Table Name: Products



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Defining Child Tables

Table Name: **User**



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In the table data model, tables can be organized in parent/child hierarchy. Child tables are created to hold subordinate information. You can create multiple child tables for a given parent table. You have to be careful when using child tables because the child table records are stored separately from the parent table. So, when you manipulate the data in these data tables, it will result in multiple I/O requests, which may pose an issue for performance-constrained applications.

Child table records of any parent table record are stored in the same shard as the parent table record. The child table is also defined using primary keys and value fields. For any table, parent or child, the primary key can be defined using one or more columns. In ONDB, there are no joins or relationships defined between the tables. Therefore, child tables are not retrieved when you retrieve a parent table nor is the parent table retrieved when you retrieve a child table.

## Creating Child Tables Versus Record Fields

Child Table	Record Field
Used to hold subordinate data from the parent table	Used to hold subordinate data from the parent table
Child table rows are stored separate from the parent table row	Parent table data, including record fields, are stored together
Lower performance in retrieving data due to multiple I/O	Avoids multiple I/O
Child table rows can be indexed for faster retrieval	Record fields can also be indexed
Child table can hold unlimited number of rows	Record fields hold a fixed number of fields

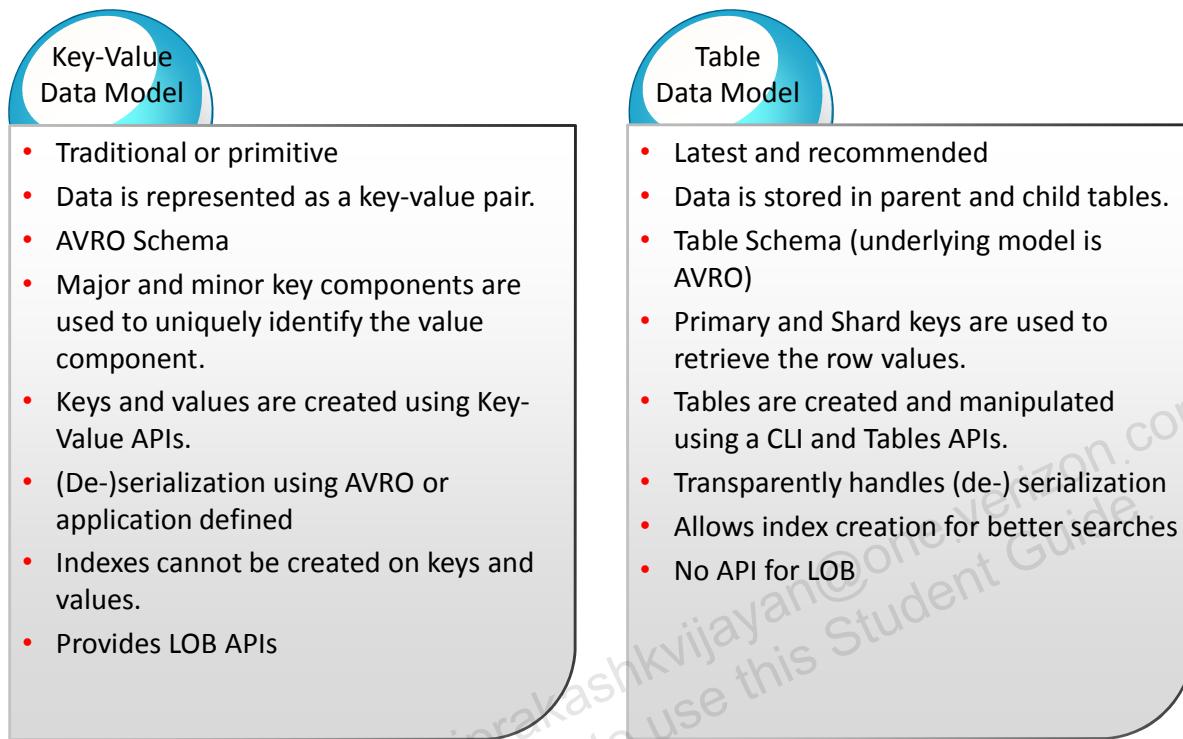


Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

While designing application schema, you will come across situations to store subordinate data. When designing your parent tables, you can store subordinate data using either child tables or as record fields within the parent table itself. The slide shows a comparison of both these data structures. If you have very simple requirements for subordinate data, you can use record fields instead of a child tables.

The assumption when using record fields is that you have a fixed known number of records that you will want to manage (unless you organize them as arrays). For example, for a contacts database, child tables allow you to have an unlimited number of addresses associated for each user. But by using records, you can associate a fixed number of addresses by creating a record field for each supported address (home and work, for example).

# Schema Design Options in ONDB: Summary



ORACLE

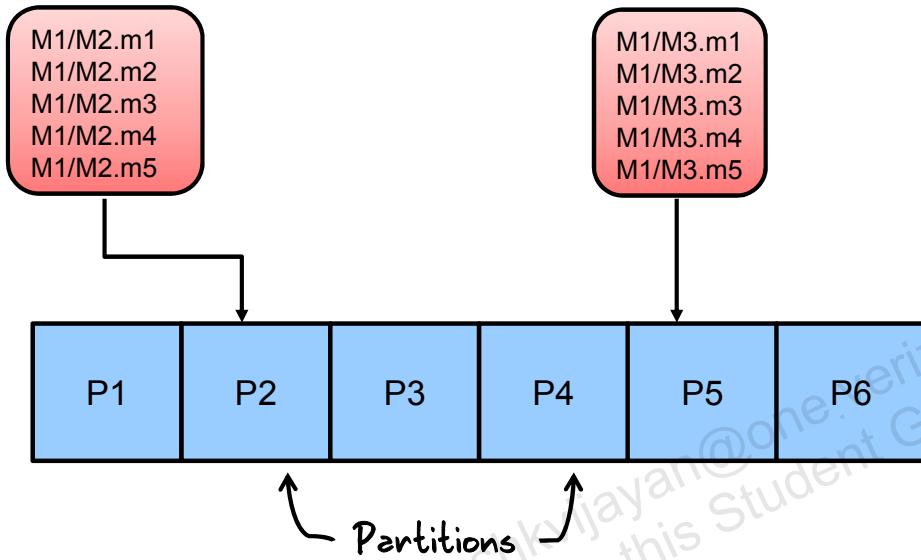
Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can create two data models for ONDB:

- **Key-value data model:** ONDB initially offered only the key-value model. This model can be considered as a schema-less model. AVRO schemas are now supported to offer some structure to the value component. Though very effective for data storage and retrieval, you cannot create indexes on the keys for searches, and so on. However, if you need to store large objects in the KVStore, you will need to use the Key-Value APIs because there are no Table APIs for LOBs yet.
- **Table data model:** This model is one of the latest enhancements of the ONDB product. It offers a flexible schema to store the data and is the recommended approach for data storage and retrieval.

In this course, Table APIs will be the main focus for data storage and retrieval.

## Using Keys to Retrieve Data



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The Oracle NoSQL driver evenly spreads an application's keys across the KVStore's partitions based on the key components. That is, in the key-value data model, records that share the same combination of major-key components are guaranteed to be in the same partition. And in the table data model, records that share the same shard keys are guaranteed to be in the same partition. If no shard keys are defined, then the primary keys are considered to be the shard keys too.

When a set of records that you want to work with are in the same partition, you can efficiently query them. This means that you can perform multiple operations on these records under a single atomic operation.

For example, the slide shows two sets of records from the key-value data model in pink boxes. The records that have the same combination of major-key components are stored together in a single partition.

**Note:** In this example and other examples in this lesson, a slash character (/) is used to separate the major-key components, and a “.” is used to specify minor-key components. This convention is used only for the purposes of illustration.

# RDBMS Versus Table Data Model

RDBMS	ONDB Data Model
Normalized data Tables are related. Constraints enforced	De-normalized data No relations No constraints

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Quiz

The performance of an application always improves when you increase the number of child tables in the table data model.

- a. True
- b. False



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: b**

# Introducing Sample Email Application

Email solution requirements:

- Store massive volume of data
- Highly available, with zero down time

Why ONDB?

- Highly available with zero down time
- Distributed environment
- High throughput and low latency
- Quick installation and implementation



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

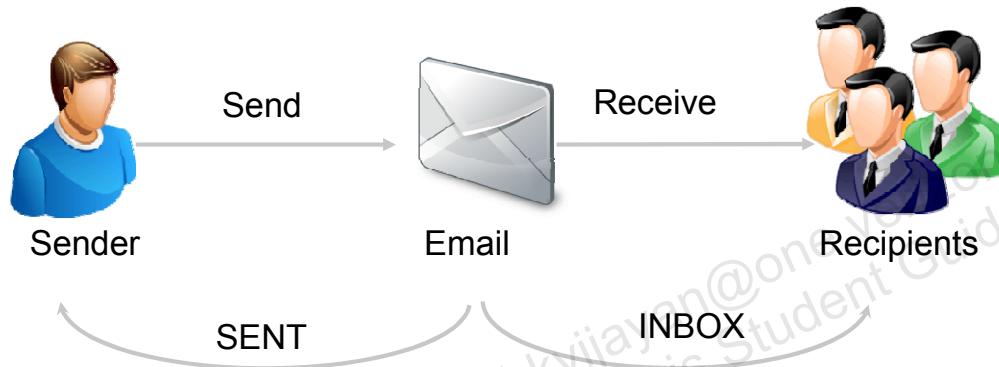
ONDB product team has developed a sample application to explain the usage of Table APIs. In this course, you will refer this application to understand data modeling and view Table API examples.

Now, why would you use ONDB to implement an email application? Consider a requirement to deliver email solutions to large corporations. The application must handle massive volumes of data (in the form of messages and attachments) for million of customers. In order to implement this application, the adopted technology should be highly available (that is, no down time), distributed across different geo-locations (in order to recover from any disaster), and process requests at lightning speeds. ONDB provides everything for implementing this solution.

## Data Modeling for the Email App

Approach is similar to RDBMS

- Business requirements
- Entities and relationships
- Query access patterns (CRUD, range, ACID)

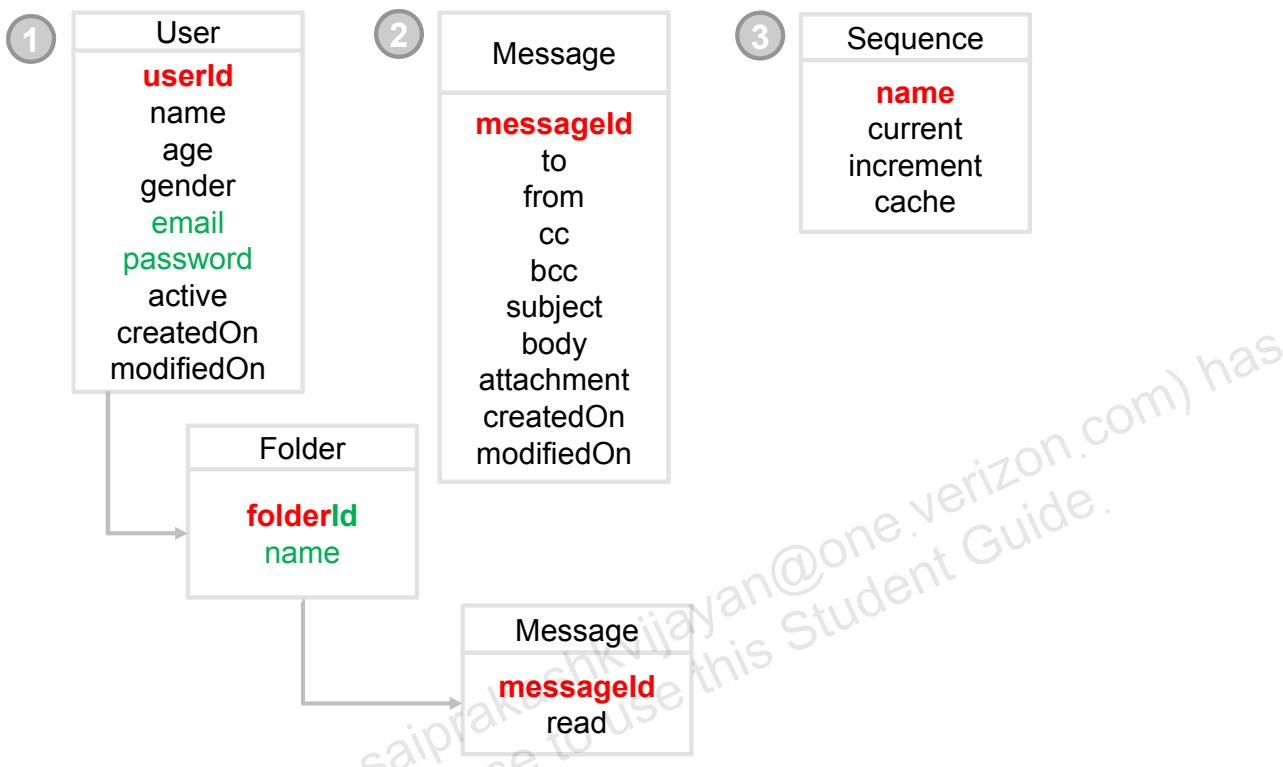


ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

When it comes to data modeling for the ONDB data model, the process is very similar to the relational data model. You first need to identify the business requirements for the application. You identify the entities for the application and how they interact with each other. For example, in the email app, the entities are a sender, a receiver, and the email. The sender sends an email and the receiver receives the email. The email is received in the inbox of the receiver and stored in the Sent folder of the sender. After identifying the entities and understanding how the entities interact with each other, you must identify how the queries will be written. All this information is then used to model the data structure for the application.

# Email App Schema



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The sample email application has the following tables:

- **User:** Holds the user information. `userId` is the primary key for the table and two indexes are created on the `email` and `password` fields.
- **Message:** Holds the message content and details. The `messagelId` field is the primary key.
- **Sequence:** Holds the current sequence IDs for all the sequence counts that are used in the application tables
- **User.Folder:** Adds the default folders for each user account
- **User.Folder.Message:** Holds pointers that add the messages to the respective folders

## Design Considerations

Consider the following before finalizing your key structure:

- What is the data (value) that you want to store?
- What data must always be accessed together?
- What data can be independently accessed?
- How big is the data (value) component?
- How frequently is the data accessed?
- How many partitions are there in the store?
- How large will you allow the individual records to grow?
- Strike a balance between too many small records or a small number of very large records.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

How you design your key components will greatly affect your application's performance. You should think about the issues listed in the slide when you design the key components.

From these considerations, you will see that you must have a clear picture of the type of data you are dealing with before finalizing the key structure. You need to know what data you need to store, how large or small that data is, how frequently it will be accessed, and so on.

## Summary

In this lesson, you should have learned how to:

- Describe the key-value data model
- Describe table data model
- Define major and minor keys
- Define primary and shard keys
- Describe how keys are used to retrieve data from the KVStore
- Describe key design best practices



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Practice 4 Overview: Designing Data Model

This practice covers designing a schema for Oracle NoSQL Database concepts.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This is a paper-based practice to check your understanding of the concepts covered in the lesson.

Unauthorized reproduction or distribution prohibited. Copyright© 2016, Oracle and/or its affiliates.

Sai Vijayan S (sai.vijayan.saiprakashkvijayan@one.verizon.com) has  
a non-transferable license to use this Student Guide.

## Application-Specific Requirements

5

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

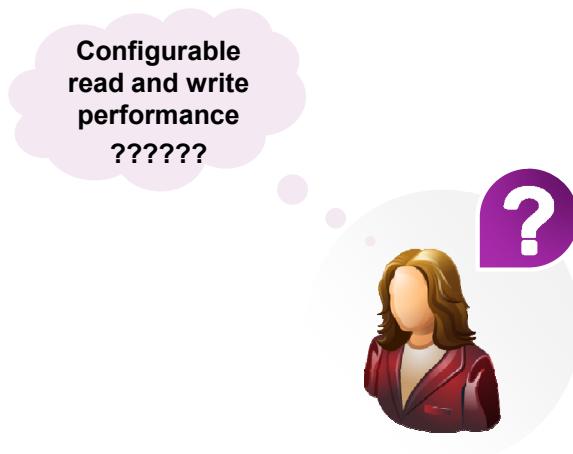
After completing this lesson, you should be able to:

- Describe the read and write processes
- Define consistency
- Identify the types of consistency policies
- Determine the consistency policy for an operation/application
- Define durability
- Identify the types of durability policies
- Determine the durability policy for an operation/application



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Doris Finalizes Application Design

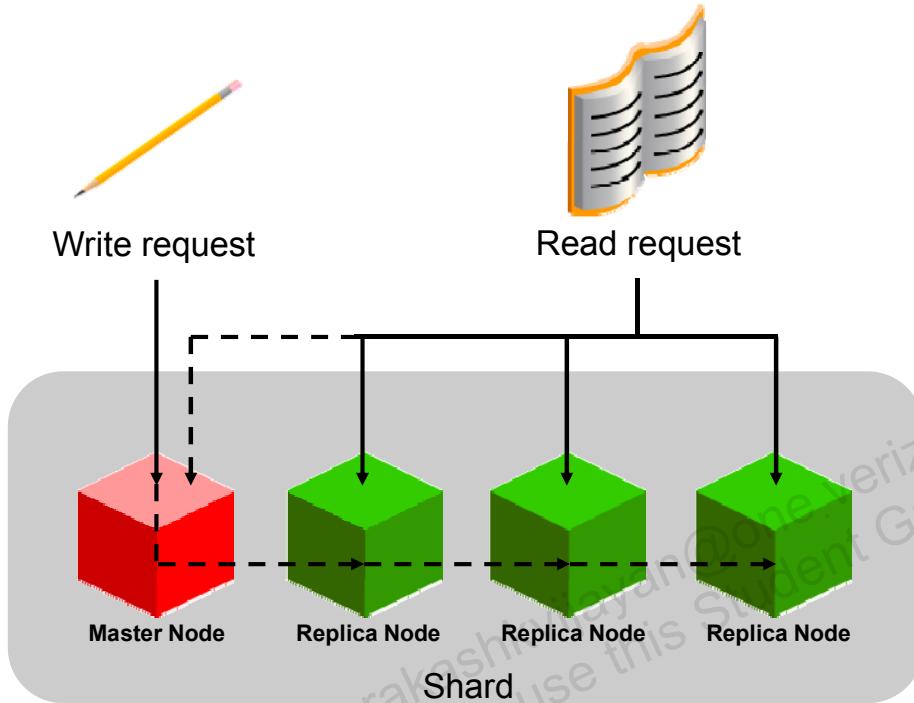


ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Earnback application has some specific requirements on data availability and performance throughputs. Doris recollects that one of ONDB's key features is that you can configure read and write operations as per application performance requirements. Doris decides to examine these capabilities before proceeding with Earnback application development.

## Write and Read Process



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Recall that an Oracle NoSQL Database consists of a number of storage nodes that may be spread across different data centers or zones. These nodes are grouped into shards, and each shard has a master node and one or more replica nodes.

All the write requests are performed at the master node. Therefore, at any given time, the master node always has the most up-to-date data.

After a write request is performed at the master node, it is then written to the replica nodes. At any point in time, the replica nodes might or might not have the most up-to-date data.

Depending on the consistency requirements of a read operation, the read requests are processed by either the replica nodes or the master node.

## Consistency Policy: Definition

- A consistency policy determines whether to read a record from the master node or from one of the replica nodes.
- Consistency policies affect the write performance of the KVStore.
- A consistency exception is thrown if the specified consistency policy is not met.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

A consistency policy applies to read operations. It determines the node (master or replica) from which a record should be read or fetched. The master nodes contain the most up-to-date data. Therefore, if you want the most up-to-date version of a record, you should specify a consistency policy that channels the read request to the master node. If your requirement is relaxed, you can specify a consistency policy so that the read request is serviced by one of the replica nodes.

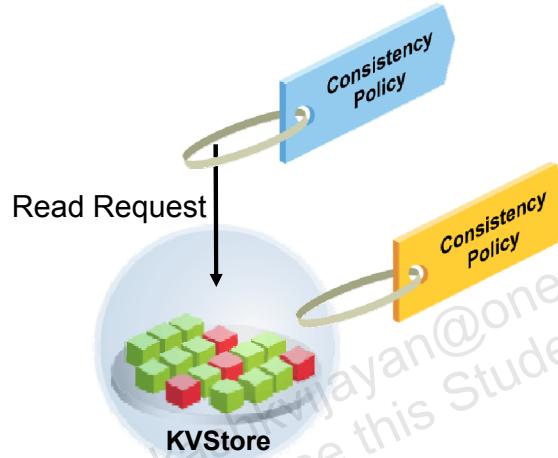
Because all write requests are always performed at the master node, channeling read requests to the master node brings down the write performance of your KVStore. Therefore, when you are deciding on a consistency policy, you should also consider the write performance requirements for your application.

If the read operation does not satisfy the specified consistency policy, an exception is thrown.

# Applying Consistency

You can apply a consistency policy to:

- An entire KVStore
- A specific read operation (overriding the policy set for the entire KVStore)



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can apply consistency policies as follows:

- For an entire KVStore. In this case, all read operations will use the specified consistency policy.
- For a single read operation. In this case, the consistency policy that exists for the entire store is overridden.

You learn how to use the Java APIs to set a consistency policy in the lesson titled "Configuring Consistency."

## Default Consistency

- A default consistency policy is automatically assigned to a KVStore.
- The default consistency policy is called `Consistency.NONE_REQUIRED`.
- The default consistency policy can be:
  - Overridden by a read operation
  - Changed for the entire store



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

After a KVStore has been installed and deployed, a default consistency policy is automatically assigned to the KVStore. This default consistency policy (`Consistency.NONE_REQUIRED`) is one of the predefined consistencies that are available for Oracle NoSQL Database. This is the most relaxed consistency policy. You learn more about it later in this lesson.

## Types of Consistency Policies

Oracle NoSQL Database enables you to specify three types of consistency policies:

- Predefined consistency policies
  - ABSOLUTE
  - NONE\_REQUIRED
  - NONE\_REQUIRED\_NO\_MASTER
- Time-based consistency policy
- Version-based consistency policy



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In Oracle NoSQL Database, you use the three types of policies listed in the slide to configure the consistency guarantee according to your application requirements.

## Predefined Consistency Policies

There are three predefined consistency policies:

- `Consistency.ABSOLUTE`
- `Consistency.NONE_REQUIRED`
- `Consistency.NONE_REQUIRED_NO_MASTER`



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

- **`Consistency.ABSOLUTE`**
  - This policy sends the read request to the master node. This means that you get the most up-to-date version of a record.
  - This is the strongest possible consistency guarantee.
  - Because the master node also handles write requests, channeling read requests to the master node can affect the write performance of an application.
- **`Consistency.NONE_REQUIRED`**
  - This policy sends the read requests to any one of the replica nodes, possibly resulting in a stale version of a record.
  - This is the most relaxed consistency policy.
  - Specifying this policy improves a KVStore's performance because the read requests can be handled by the replica nodes and the master node can handle the write operations.
- **`Consistency.NONE_REQUIRED_NO_MASTER`**
  - This policy sends the read requests only to replica nodes and never to the master nodes. This is to ensure that read requests are never processed by the master node because that can affect the write performance.

## Time-Based Consistency Policy

A time-based consistency policy:

- Specifies the amount of time that a replica node can lag behind a master node
- Requires that the clocks in all the nodes of a KVStore be synchronized using a protocol (such as NTP)



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can specify a time-based consistency policy while performing a read operation.

A time-based consistency policy specifies the amount of time a replica node is permitted to lag behind the master node. A consistency exception is thrown during the servicing of a read request if the record in the replica node is older than the permitted lag time.

This policy is effective only if the clocks running on all the nodes in a KVStore are synchronized by using a protocol, such as NTP.

## Version-Based Consistency Policy

A version-based consistency policy:

- Fetches a record from a replica node if it matches a specified version
- Requires that the version information be transferred between processes in an application



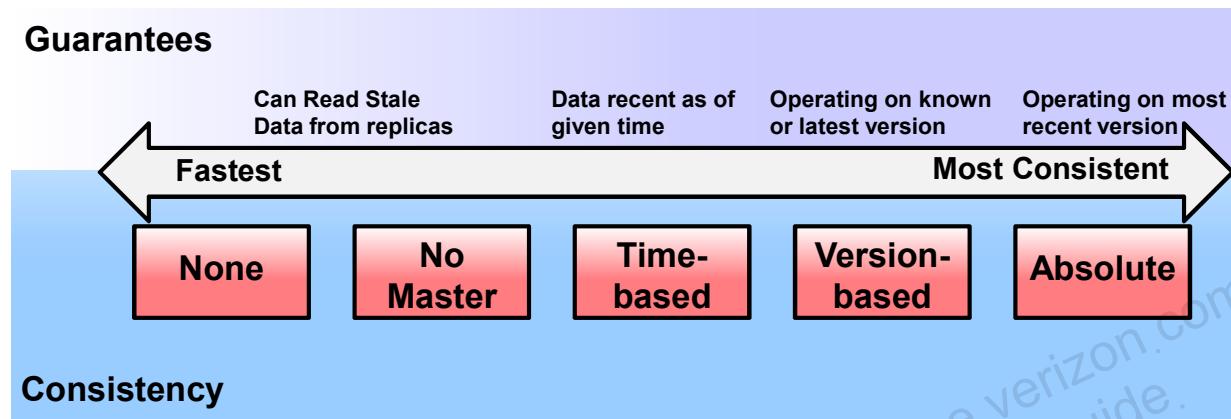
Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can specify a version-based consistency policy while performing a read operation.

A version-based consistency policy specifies the version of a record that the replica node must match. If the record in the replica node is older than the specified version during the servicing of a read request, a consistency exception is thrown.

This policy requires the transfer of version details between the processes in an application.

# Consistency Policies: Summary



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide summarizes all the consistency policies that you can set for a KVStore. Deciding which policy to specify depends on the requirements of your application and individual read operations. When making a decision, it is important to consider that a high consistency guarantee might result in slower KVStore performance. However, a low consistency guarantee will give you fast KVStore performance.

As a result, you must determine the right balance between consistency and the performance that you require.

## Quiz

Consistency. Absolute is the strongest form of consistency because the read operation is performed at the master node.

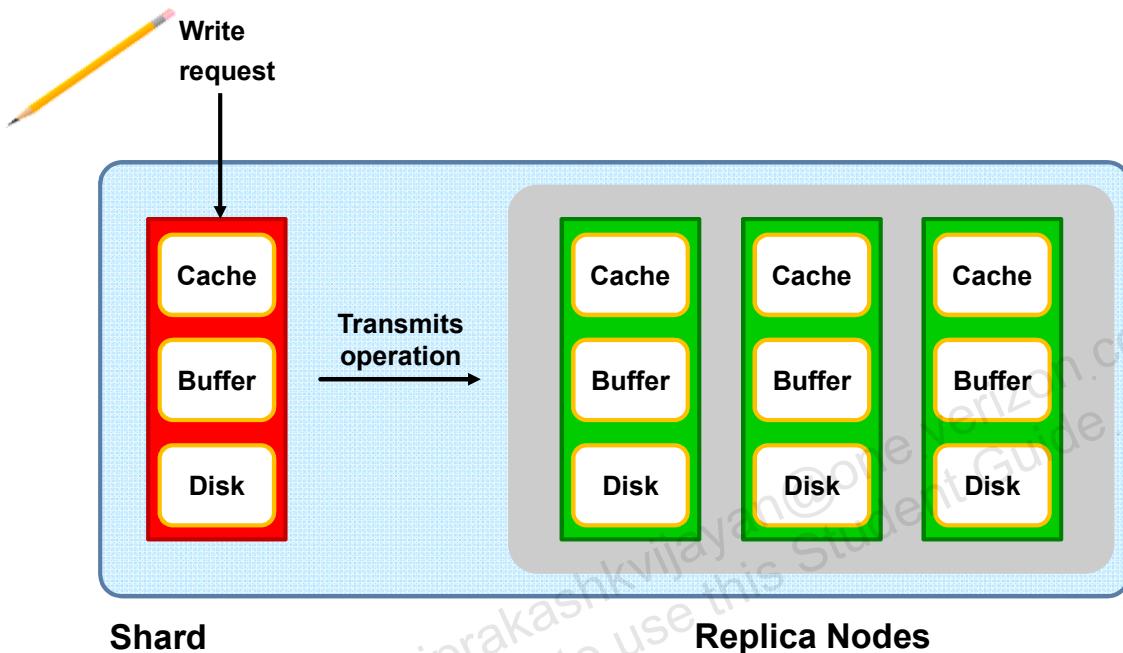
- a. True
- b. False



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

## Write Process



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You have already seen that a write operation is always performed at a master node. The master node is a replication node that resides in a storage node (that is, in a physical or virtual machine). A physical machine has a storage disk where data is permanently stored.

When a write operation request is received at a master node, the record is written first to the in-memory cache, then to the system buffers, and finally to the hardware disk. After the write operation is completed at the master node, the master node transmits the write operation to the replica nodes in its replication group.

## Durability: Definition

- A durability policy determines:
  - At what point in the entire write process the write request is considered to be completed
  - How persistent the data is in a catastrophic failure
- Durability policies affect the write performance of the KVStore.
- A durability exception is thrown if the specified durability policy is not met.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

A durability policy applies to write operations. In the previous slide, you saw the various stages of a write operation. A durability policy determines the point of the write process at which the master node acknowledges that the write operation is completed. Therefore, the durability policy determines the degree of persistence of the data in a KVStore in case of a catastrophic failure.

A catastrophic failure can take place due to any of the following events:

- Power outages
- Disk crashes
- Physical memory corruption
- Fatal application programming errors

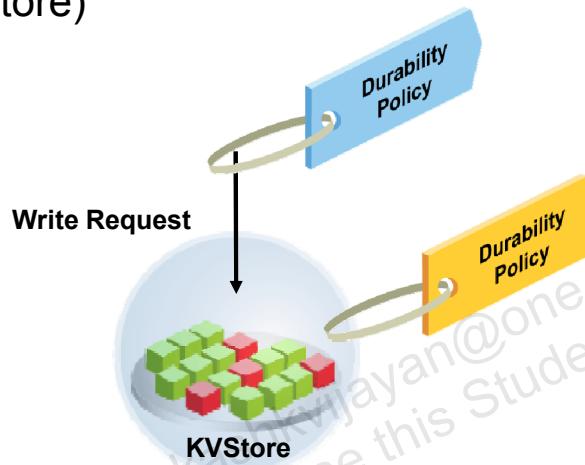
If your application requires that the data be strongly persistent, you should set your durability policy so that the write request is considered to be complete only after the record has been written to the disk of the master node as well as all replica nodes. However, such a policy significantly slows down the write operation.

If the write operation does not satisfy the specified durability policy, an exception is thrown.

# Applying Durability

You can apply a durability policy to:

- An entire KVStore
- A specific write operation (thus overriding the policy set for the entire KVStore)



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can apply durability policies as follows:

- For the entire KVStore. In this case, all write operations will use the specified durability policy.
- For a single write operation. In this case, the durability policy that exists for the entire store is overridden.

You learn how to use the Java APIs to set a durability policy in the lesson titled “Configuring Durability.”

# Durability Policy

A durability policy consists of two policies:

- Synchronization policy
- Acknowledgment policy



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

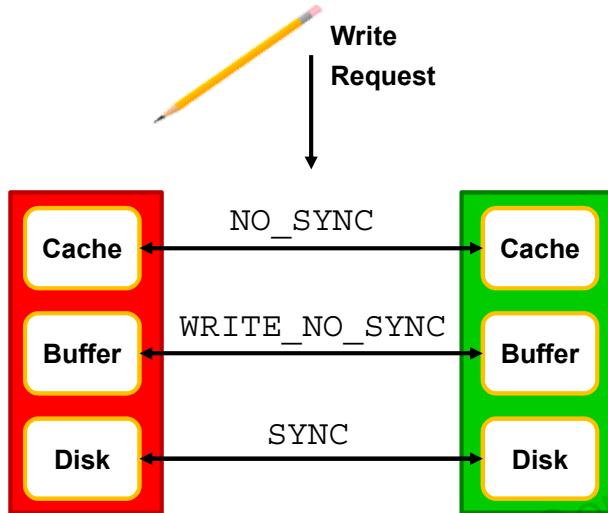
You set two policies to create a durability policy for an entire KVStore or for an individual write operation:

- Synchronization policy
- Acknowledgment policy

The synchronization policy can be set for the master node and the replica nodes separately. So the durability policy is a sum of the synchronization policy at the master node, the synchronization policy for the replica nodes, and the acknowledgment policy.

You learn more about these policies in the next few slides.

# Synchronization Policy



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

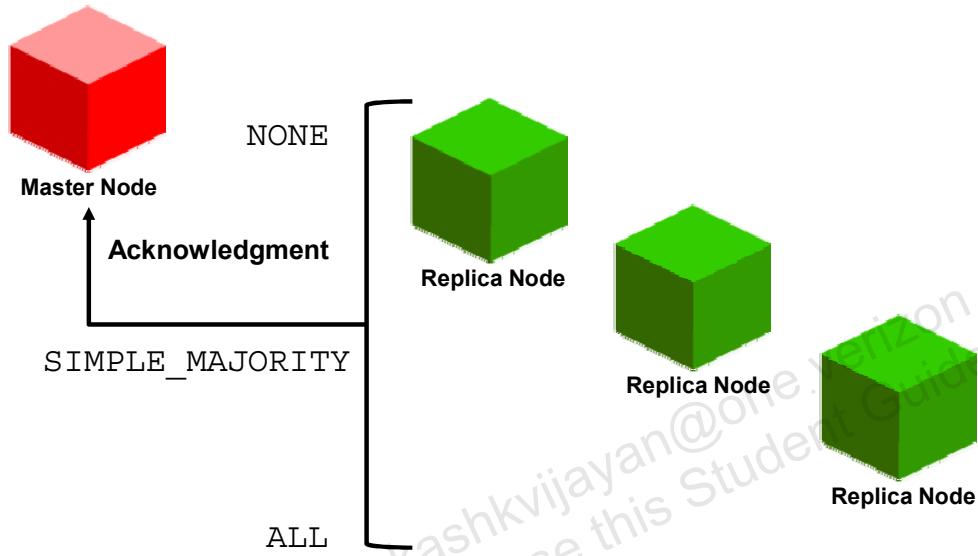
A write operation is performed by writing the record first to the in-cache memory, then to the system buffers, and finally to the disk. A synchronization policy determines whether the record should be written to stable storage (disk) before the write operation is considered complete in a node—whether it is a master node or a replica node.

A synchronization policy can be set for a node by using the following values:

- **NO\_SYNC**: A write operation is considered complete by the node when the record is written to the in-memory cache. This is the least durable policy setting.
- **WRITE\_NO\_SYNC**: A write operation is considered complete by the node when the record is written to the in-memory cache as well as the system buffer.
- **SYNC**: A write operation is considered complete by the node only after the record is written to the system disk. This is the most durable policy setting that can be applied to a node.

In all of the preceding cases, the write operation is eventually written to the system disk. The durability policy only specifies when the node considers the write to be complete and returns to the application so that the application can proceed to the next task.

# Acknowledgment Policy



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

After a write operation is completed at the master node, it is transmitted to the replica nodes. After the write operation is completed at the replica nodes, the replica nodes send an acknowledgment back to the master node.

An acknowledgment policy determines the number of acknowledgments the master node must wait for before it considers the write operation to be completed at the replica nodes.

An acknowledgment policy can be set by using the following values:

- **ALL:** The master node waits for acknowledgment from all the replica nodes. This setting offers the highest durability. However, the write operation is significantly slowed down.
- **NONE:** The master node does not wait for acknowledgment from any of the replica nodes. This setting offers the least durability.
- **SIMPLE\_MAJORITY:** The master node waits for acknowledgment from a majority of the replica nodes.

In all the preceding cases, the write operation is eventually written to all the replica nodes. The durability policy only specifies when the master node considers the write to be completed at the replica nodes and returns to the application so that the application can proceed to the next task.

## Default Durability

- A default durability is automatically assigned to a KVStore.
- The default durability policy has the following setting:
  - NO\_SYNC at master
  - NO\_SYNC at replicas
  - SIMPLE\_MAJORITY acknowledgment required
- A default durability policy can be:
  - Overridden by a write operation
  - Changed for the entire store

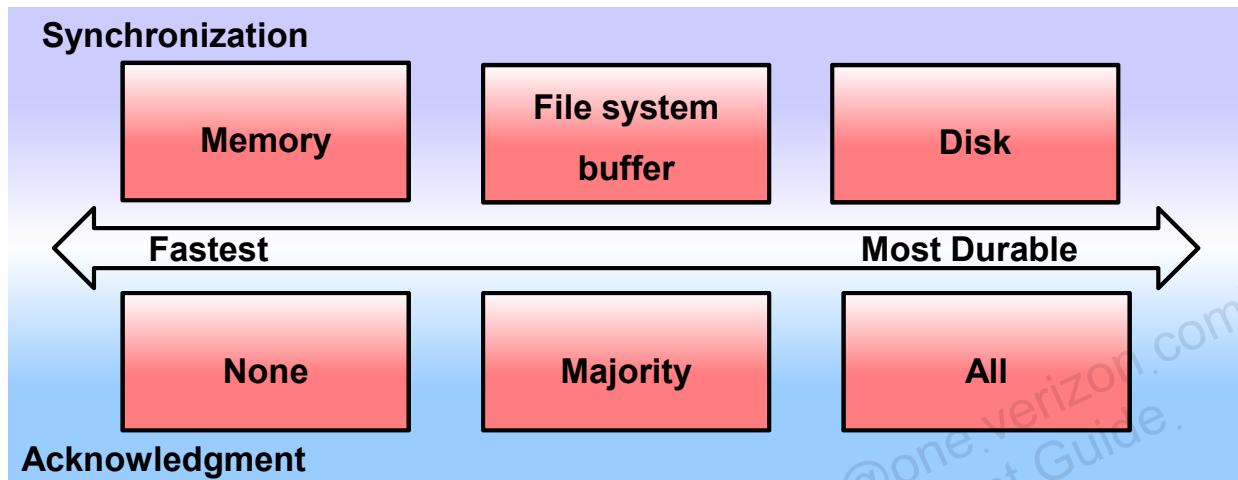


Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

After a KVStore has been installed and deployed, a default durability policy is automatically assigned to the KVStore. The default durability policy specifies that a write operation is considered complete when both of the following have occurred:

- A record is written to the cache memory in the master node and to the cache memory in the replica node.
- Acknowledgment is received from a majority of replica nodes.

## Durability Policies: Summary



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide summarizes the durability policies that you can set for a KVStore.

Deciding which policy to specify depends on the requirements of your application and individual write operations. When making a decision, it is important to consider that a high durability guarantee might result in slower KVStore performance. However, a low durability guarantee gives you fast KVStore performance.

As a result, you must set the right balance between durability and the performance that you require.

## Quiz

Write performance is faster if more acknowledgments are sent to the master.

- a. True
- b. False



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

### Answer: b

Waiting for acknowledgments means waiting for a write message to travel from the master to the replicas, then for the write operation to be performed at the replica, and finally for an acknowledgment message to travel from the replica back to the master.

For a computer application, this can take a long time.

## Summary

In this lesson, you should have learned how to:

- Describe the read and write processes
- Define consistency
- Identify the types of consistency policies
- Determine the consistency policy for an operation/application
- Define durability
- Identify the types of durability policies
- Determine the durability policy for an operation/application



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## **Practice 5 Overview: Application-Specific Requirements**

In this practice, you test your understanding of consistency and durability concepts.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This is a paper-based practice to check your understanding of the concepts covered in the lesson.

# Creating Tables

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

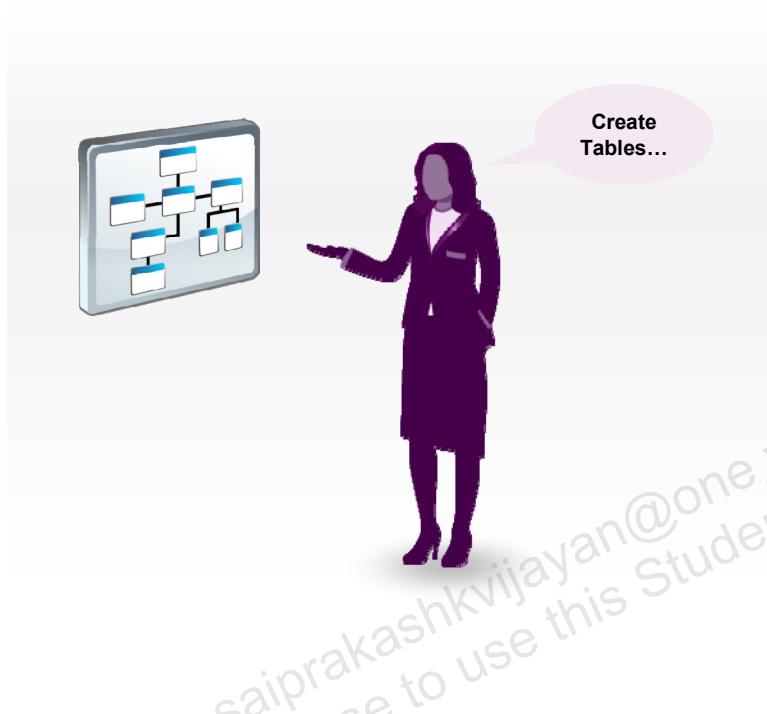
After completing this lesson, you should be able to:

- Identify ways to create tables
- Create tables
- Create child tables
- Modify a table
- Delete a table
- Create indexes



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Doris Creates Application Tables



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris has identified the tables required to support her application. She has written the code to fetch a KVStore handle. Now, she has to create the tables in the KVStore.

## Creating Tables: Overview



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can create tables in a KVStore using:

1. **Java APIs:** You can create, modify, and delete tables by creating DDL commands and invoking APIs to execute the commands from a Java application. This is the recommended approach.
2. **Data CLI:** You can create, modify, and delete tables by directly executing the DDL commands at the `kv` prompt.

You will learn how to create tables using both the methods. First, you will learn to create the DDL commands.

# Data Definition Language Commands

CREATE TABLE



DESCRIBE [AS JSON] TABLE  
SHOW [AS JSON] TABLES

- Keys words are reserved.
- Table and index names limited to 32 chars
- Field names can be 64 chars.
- Only alphanumeric + “\_” allowed
- All names to start with a letter

ALTER TABLE ADD FIELD



ALTER TABLE DROP FIELD

DROP TABLE



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The DDL commands available for Oracle NoSQL Database are listed in the slide. You will learn to use the CREATE, ALTER, and DROP commands in the coming slides. To view the description of the created tables, you use the DESCRIBE command. You can specify AS JSON to view the description in JSON format. To view the tables created in the KVStore, you use the SHOW command. Here also, you can specify AS JSON to view the output in JSON format.

While using these commands, remember the following:

- The DDL command keywords are reserved words and you cannot use them to identify table, index, or field names.
- The table and index names are limited to 32 characters.
- Field names can be 64 characters.
- All table, index, and field names are restricted to alphanumeric characters, plus underscore (“\_”).
- All names must start with a letter.

Each of these commands is executed by using a Java API or an execute command, which you will learn later in this lesson.

## CREATE TABLE

```
CREATE TABLE [IF NOT EXISTS] table-name (
    field-definition,
    field-definition-2 ...,
    PRIMARY KEY
    ([SHARD] field-name, field-name-2...),
    [COMMENT "comment string"] )
```

```
field-name type
[CHECK]
[NOT NULL] } Field constraints
[DEFAULT "default-value"]
[COMMENT "comment-string"]
```

### Supported Datatypes

- Array
- Boolean
- Binary
- Double
- Enum
- Fixed Binary
- Float
- Integer
- Long
- Map
- Record
- String

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the syntax for creating a table. The CREATE TABLE command has the following parts:

- **[IF NOT EXISTS]**: This is an optional phase. If specified, it creates a table only if a similar table is not already present in the KVStore. If not specified and if a table with the same name exists, an error is thrown.
- **table name**: The name you specify for the table. It should conform to the standards listed in the previous slide. To create a child table, prefix the child table name with the parent table name using a dot notation.
- **field definition**: A comma-separated list of field names mentioned as per the syntax for fields shown in the slide. The various types you can specify for a field are also listed in the slide.
  - All the elements of the `ARRAY` must be of the same data type that must be declared when you define the array.
  - When using an `ENUM`, all the acceptable values should be specified when you define the field.
  - Keys in a `MAP` are of String data type. You must specify the data type of the data portion of a `MAP`.
  - Use the `ELEMENTOF` keyword to refer to elements of an `ARRAY` or a `MAP`.

You can create the following constraints on the field values during the field definition itself.

- **CHECK:** Used to restrict the field values to a range of values. This parameter is discussed in detail in the next slide.
- **NOT NULL:** To enforce a value for this field. You can specify a default value for a field by using the **DEFAULT** keyword.

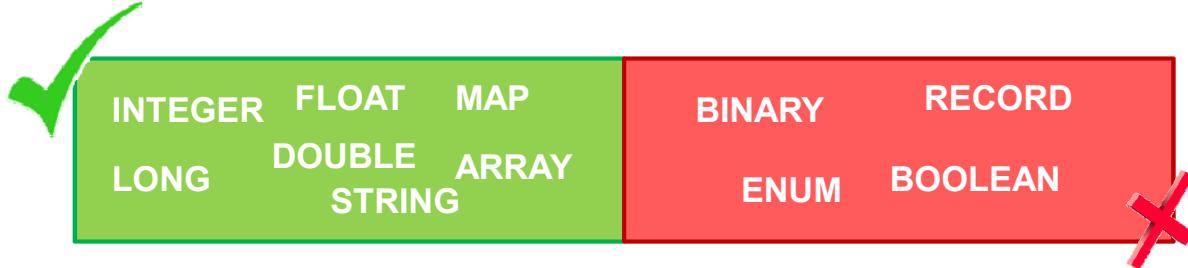
You can also include a description or comment for a field by using the **COMMENT** keyword.

Comments are written using any of the following styles:

```
/* This is a comment*/  
//This is a comment  
#This is a comment
```

- **PRIMARY KEY:** While creating a table, you need to specify a primary key field for the table. The values of this field are used to uniquely identify a row in the table. You can specify a composite primary key. That is, two or more fields can be used to create a primary key for the record.
- **SHARD:** This is an optional keyword used to identify the SHARD key for a table. As learned in Unit I, shard keys are used to ensure a set of records are always stored together in a single shard.
- **COMMENT:** This is used to document a comment or description for the table.

## CHECK Constraint



```
myInt INTEGER CHECK(myInt > 10 and myInt < 20)
```

```
myMap MAP(INTEGER CHECK(ELEMENTOF(myMap) > 10))  
myArray ARRAY(INTEGER CHECK(ELEMENTOF(myArray) > 100 AND \  
ELEMENTOF(myArray) < 1000))
```

```
myRec RECORD(a STRING, b INTEGER CHECK(b >= 0 AND b <= 10))
```

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can use the CHECK parameter to restrict the values of a field by specifying an allowable range of values. The relational operators AND, <, <=, >, and >= are all supported.

CHECK constraint can be specified for all the data types listed in the green box in the slide. The data types in the red box are not supported.

The slide shows examples of using CHECK in INTEGER fields, MAP fields, ARRAY fields, and within a field of a RECORD.



## Creating a Table from a Java Application

To create a table in a KVStore from a Java Application:

1. Create the DDL command.
2. Assign the command to a string variable.
3. Obtain a handle to the KVStore and TableAPI.
4. Execute the DDL command by using a Java API.



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

With the latest release of ONDB, you can create tables from a Java application itself. In order to do so, you first need to create the DDL command as per the syntax. Then, create a variable of type String and assign to it the DDL command.

As learned previously in the introduction for Unit II, to perform any operation in a KVStore, you need to obtain a KVStore handle. Even to create a table in the KVStore, you need to obtain a KVStore handle. Additionally, in order to perform any operation related to tables, you need to obtain a TableAPI handle.

Then, you execute the command using a Java API, about which you will learn later in this lesson.



## Introducing TableAPI

- TableAPI is an interface to the tables in a KVStore.
- To perform any operations on tables, you should obtain an instance of TableAPI.

```
TableAPI tableH = kvstore.getTableAPI();
```



Handle to the  
KVStore

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In order to perform any operation on the tables created in a KVStore, you need to use a TableAPI Java interface written for ONDB.



## Executing a DDL Command

```
ExecutionFuture execute(String statement)
    throws FaultException,
        IllegalArgumentException
```

```
StatementResult executeSync(String statement)
    throws FaultException,
        IllegalArgumentException
```

**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can execute a DDL command from a Java application by using two methods:

- **execute:** This method executes the table statement asynchronously. That is, when the method returns, the table statement might or might not have completed. You can use the return value `ExecutionFuture` to get information about the status of the application.
- **executeSync:** This method executes the table statement synchronously. That is, the method returns only when the table statement has finished and the result of the operation is returned.



## Executing a DDL Command: Example

```
try
{
    StatementResult sr = myStore.getTableAPI().executeSync(tbStmt);
    System.out.println("MESSAGE table created...");

    System.out.println(sr.getInfo());

} // try
catch (IllegalArgumentException e)
{
    System.out.println("The statement is invalid: " + e);
}
catch (FaultException e)
{
    System.out.println("There is a transient problem, retry the "
        + "operation: " + e);
} // catch
```

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of executing a table statement. You have to write the method inside a try-catch loop.

# Quiz

Which of the following data types are supported by the CREATE TABLE command?

- a. float
- b. string
- c. binary
- d. record
- e. enum



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer:** a, b, c, d, e



## Creating a Table from the CLI

To create a table in a KVStore using the CLI:

1. Create the DDL command.
2. Access the CLI.
3. Execute the DDL command.

**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can create tables from a CLI also. You first need to create the DDL command as per the syntax. You then access the CLI and use an `execute` command to create the tables or modify tables.



## Accessing the CLI

Access the CLI by invoking the `runadmin` utility or the `kvcli.jar` file.

1

```
java -jar $KVHOME/lib/kvstore.jar runadmin  
      -port <port number>  
      -host <host name>
```

Admin CLI

2

```
java -jar $KVHOME/lib/kvcli.jar  
      -port <port number>  
      -host <host name>  
      -store kvstore
```

Data CLI

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To create, delete, or update a table, you can use a command-line interface (CLI). There are two command-line interfaces that are provided with Oracle NoSQL Database: an admin CLI and a data CLI. Currently, both the CLIs provide the same functionality. That is, administration tasks as well as developer tasks can be performed using either of these CLIs.

You can access the admin CLI interface by invoking the `runadmin` utility of the `kvstore.jar` file. You need to pass the `port` and `host` parameter details. You can access the data CLI by invoking the `kvcli.jar` file. In addition to the host and port parameter values you need to pass the name of the store, which you mentioned while configuring the store. The complete syntax for invoking these CLIs is shown in the slide.

You use the CLI to create and define the application tables, fields, keys, as well as indexes. You can also insert and retrieve table data as well as key-value data using CLI.

You can run single line commands directly when invoking the CLI. You can also use a load command to run a script while invoking the CLI. If no other parameters are passed, then the CLI interface is shown and you can use it interactively to run commands.



## Executing a DDL Command

```
kv-> execute "CREATE TABLE user (userId STRING,\n>name RECORD (first STRING, middle STRING, last STRING),\n>age INTEGER CHECK(age >=0 AND age<=200),\n>gender ENUM(M,F),\n>email STRING, \n>password STRING, \n>active BOOLEAN DEFAULT TRUE, \n>createdOn LONG,\n>modifiedOn LONG,\n>PRIMARY KEY (userId))"■
```

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To run the DDL command at the `kv` prompt, use the `execute` command. The DDL command is passed inside single or double quotation marks to the `execute` command. The DDL command gets executed and the `kv` prompt is returned.



# **Viewing Table Descriptions**

## All Tables in KVStore

```
execute "show tables"  
execute "show AS JSON  
tables"
```

```
kv-> show tables
Tables:
  Message
  Sequence
  User
    User.Folder
    User.Folder.Message
```

## Specific Table

```
execute "describe AS  
JSON table <tablename>  
[fields]"
```

```
kv-> show table -name User.Folder.Message
{
  "type" : "table",
  "name" : "Message",
  "comment" : null,
  "parent" : "Folder",
  "shardKey" : [ "userId" ],
  "primaryKey" : [ "userId", "folderId", "messageId" ],
  "fields" : [ {
    "name" : "userId",
    "type" : "string",
    "comment" : "User ID"
  },
  {
    "name" : "folderId",
    "type" : "string",
    "comment" : "Folder ID"
  },
  {
    "name" : "messageId",
    "type" : "string",
    "comment" : "Message ID"
  },
  {
    "name" : "content",
    "type" : "string",
    "comment" : "Message Content"
  },
  {
    "name" : "timestamp",
    "type" : "long",
    "comment" : "Timestamp of the message"
  }
  ],
  "constraints" : [
    {
      "name" : "PK_Message",
      "type" : "primary",
      "columns" : [ "userId", "folderId", "messageId" ]
    }
  ]
}
```

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To view the tables existing in a KVStore, use the show tables command in the CLI. The parent and child tables are clearly listed in the output of this command.

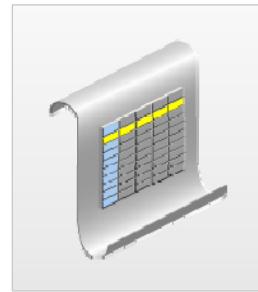
To view the schema of a specific table, use the show tables <tablename> command. The table schema is listed in JSON format. The slide output shows a part of the description for the User.Folder.Message table.



## Recommendation: Using Scripts

Use scripts for all database operations:

- This is the recommended approach.
- Prevents accidental errors/typos.
- Consistent store environment through all cycles of development, testing, and deployment.



1

```
java -jar $KVHOME/lib/kvstore.jar runadmin  
      -port <port number>  
      -host <host name>  
      load -file <path to script>
```

2

```
kv> load -file <path to script>
```

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

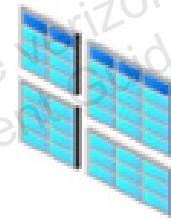
You can use the CLI interactively for creating tables and performing other operations. However, while working in an enterprise environment it is recommended that you create and run scripts to perform these operations. Database table operations include table creation, evolution, and deletion. Using scripts avoids typos and other errors as code moves from the development environment to a test environment and finally to the production environment.

A CLI script is a simple text file that contains a series of commands. To run the script:

1. You start the CLI and use a `load` command to run the file that has the commands.
2. You can also use the `load` command after connecting to the KVStore.

# Modifying a Table

```
ALTER TABLE table-name (ADD field-definition)  
ALTER TABLE table-name (DROP field-name)
```



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You may need to modify a table in the KVStore after it has been created. You can modify a table by adding new fields or removing existing fields. You cannot remove a field if it is defined as a primary field. To modify a table you use the `ALTER TABLE` command.

You cannot modify an existing field directly. Instead, you must delete the field, and then add the field back using the new definition. Note that this will cause all existing data associated with the current field to be deleted.

You can add a field to a nested record by using the dot notation.

## Deleting a Table



```
DROP TABLE [IF EXISTS] table-name
```



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Use the `DROP` command to delete a table from the KVStore.

Note that dropping a table is a lengthy operation because all table data currently existing in the store is deleted as a part of the drop operation.

## Quiz

Which of the following DDL commands takes times (comparatively) to complete execution?

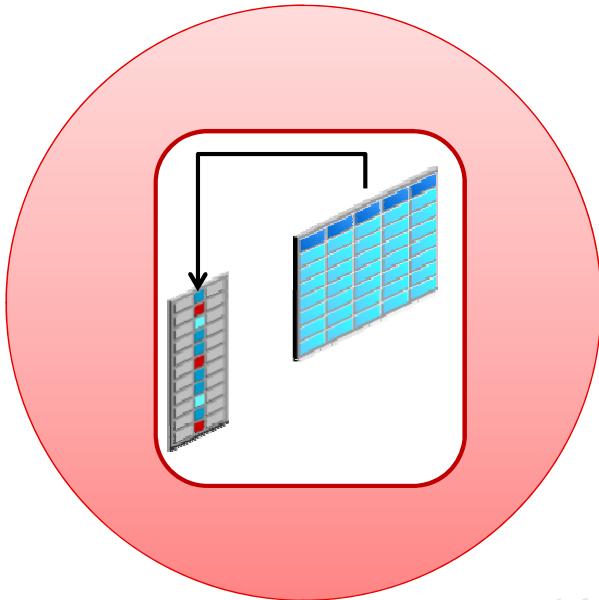
- a. CREATE
- b. ALTER
- c. DROP

 ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: c**

## Indexes: Introduction



### Supported Datatypes

- Double
- Enum
- Float
- Integer
- Long
- String

- Array
- Map
- Record

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

An index is a schema object that is used to provide faster access to data. In ONDB, usually you retrieve records using a table's primary key. Indexes provide an alternative way to fetch data. For example, if you had a table for storing a user profile, the primary keys for each row might be the user's email ID. However, if you wanted to query the table based on the user's age or gender, you would need to scan the complete table and retrieve the required records. Such lookups are time consuming when the size of the tables is voluminous. In such situations, you could create an index on the age and gender field of the table. ONDB stores the location information of the records based on the age and gender fields. This information is used when the fields are retrieved using the index and contribute to faster search results.

In ONDB, indexes can take a long time to create because Oracle NoSQL Database must examine all of the data contained in the relevant table in a KVStore. The smaller the data contained in the table, the faster the index creation will complete. Conversely, if a table contains a lot of data, then it can take a long time to create indexes for it.

Fields can be indexed only if they are declared to be one of the supported types listed in the slide. For all complex types (arrays, maps, and records), the field can be indexed if the ultimate target of the index is a scalar data type.

## Creating Indexes



```
CREATE INDEX [IF NOT EXISTS] index-name ON table-name  
          (field-name)
```

MAP

```
CREATE INDEX [IF NOT EXISTS] index-name ON table-name  
          (KEYOF field-name)  
CREATE INDEX [IF NOT EXISTS] index-name ON table-name \  
          (ELEMENTOF field-name)  
CREATE INDEX [IF NOT EXISTS] index-name ON table-name \  
          (KEYOF field-  
          name),ELEMENTOF field-name )
```

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the syntax of the `CREATE INDEX` command. You need to specify an index name, the table where the index needs to be created, and the field on which to create the index. Depending on the amount of data in a KVStore, creating indexes can take a long time. This is because index creation requires Oracle NoSQL Database to examine all the data in the store.

You can create an index for either the key entries or value entries of a MAP. To create an index on the key entries, use the `KEYOF` keyword. To create an index on the value entries of the MAP, use the `ELEMENTOF` keyword. Syntax examples are shown in the slide.

## Removing Indexes

```
DROP INDEX [IF EXISTS] index-name ON table-name
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can remove an index from a table. Use the `DROP INDEX` command to remove an index.

## Summary

In this lesson, you should have learned how to:

- Identify ways to create tables
- Create tables
- Create child tables
- Modify a table
- Delete a table
- Create indexes



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Practice 6 Overview: Creating Table Data

This practice covers the following topics:

- Create a table and child tables
- Alter table



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This is a hands-on practice.

## **Unit I**

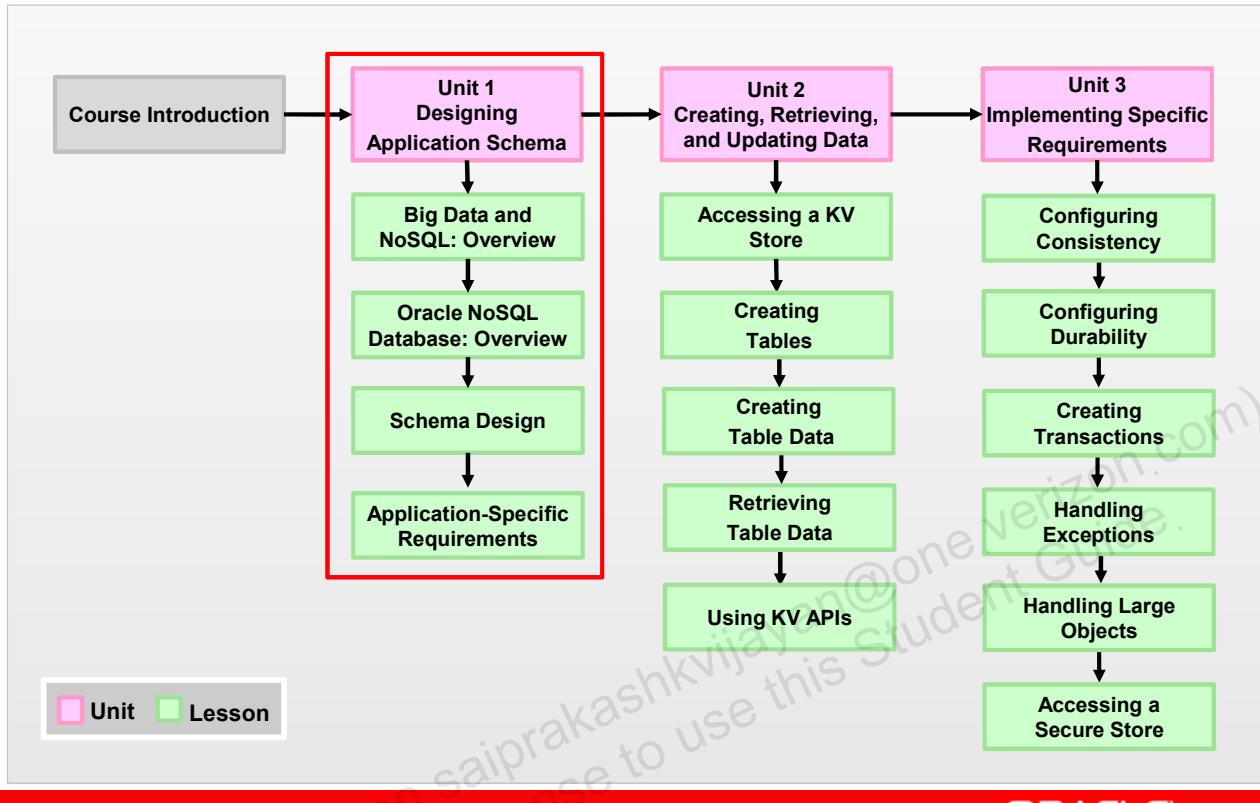
# **Summary of Designing Application Schema**



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Course Outline



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In Unit I, you completed four topics.

## Doris Has a Design

### Earnback App Design Specs

- Use table data model.
- Use key-value APIs for storing LOBs.
- Set default consistency policy as `Consistency.NONE`.
- Set the following as default durability policy: `SYNC` at master, `WRITE_NO_SYNC` at replicas, and `SIMPLE_MAJORITY` acknowledgment.
- Default consistency and durability will be overridden in certain read and write operations.
- Request for a secure installation.



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris and her team will use the Table data model. They have come up with a requirement of parent, child, and nested tables. They will use the key-value APIs for some large objects. In addition, they have identified the consistency and durability policies that they will apply.

Unauthorized reproduction or distribution prohibited. Copyright© 2016, Oracle and/or its affiliates.

Sai Vijayan S (sai.vijayan.saiprakashkvijayan@one.verizon.com) has  
a non-transferable license to use this Student Guide.

## **Unit II**

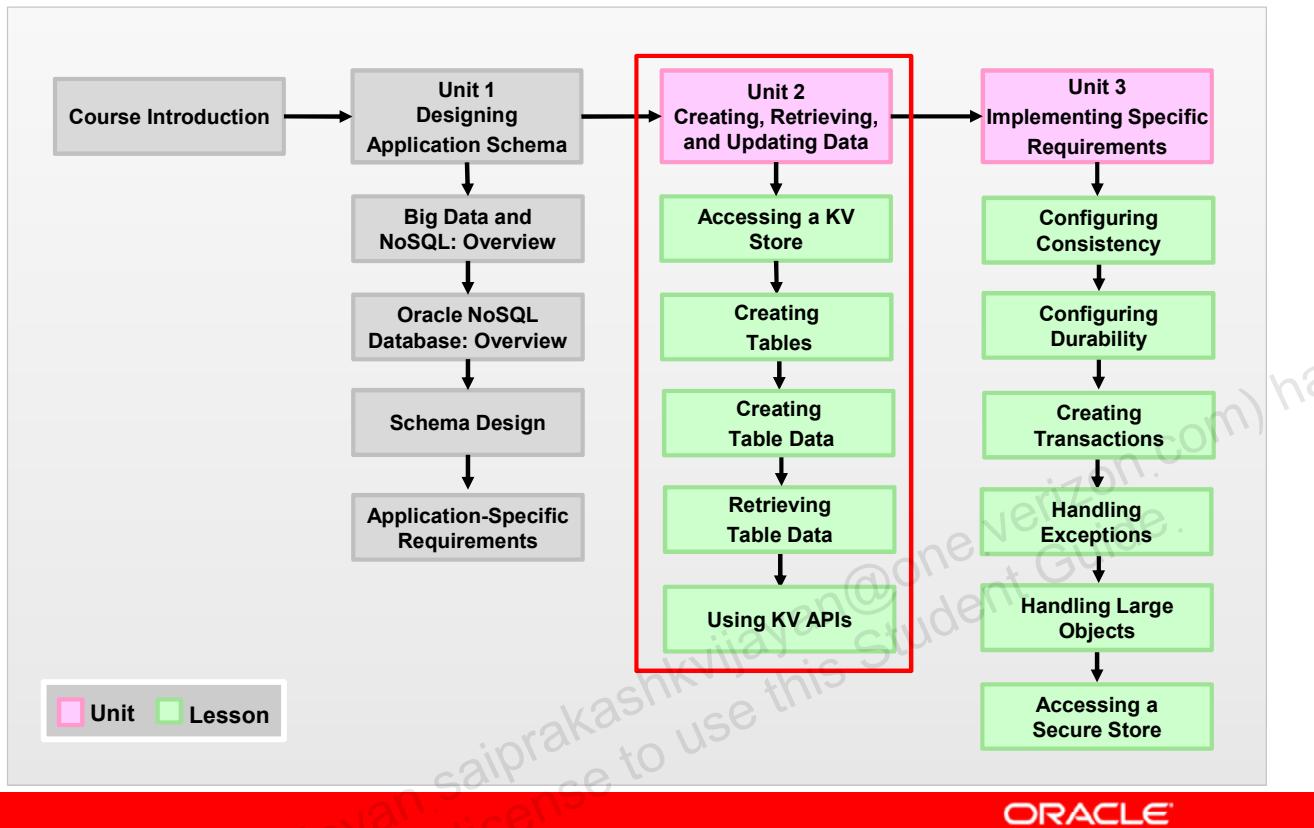
# **Creating, Retrieving, and Updating Data**



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Course Outline



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In Unit II, you learn to create tables and read and write table data. You also briefly review the key-value APIs.

## Doris Starts Coding

- ✓ Identify Application Requirements.
- ✓ Analyze Application Data.
- ✓ Design Data Model.

? What next?



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris has prepared the design for the application's data model and now is ready to create the identified tables in the KVStore. She needs to know how to create a connection to the KVStore. In other words, she needs to learn how to interact with the KVStore data.

## KVStore Handle

A KVStore handle:

- Is a resource that controls access to Oracle NoSQL Database
- Is used to open and close an already running KVStore
- Is required to perform any operation on the KVStore from a Java application



ORACLE®

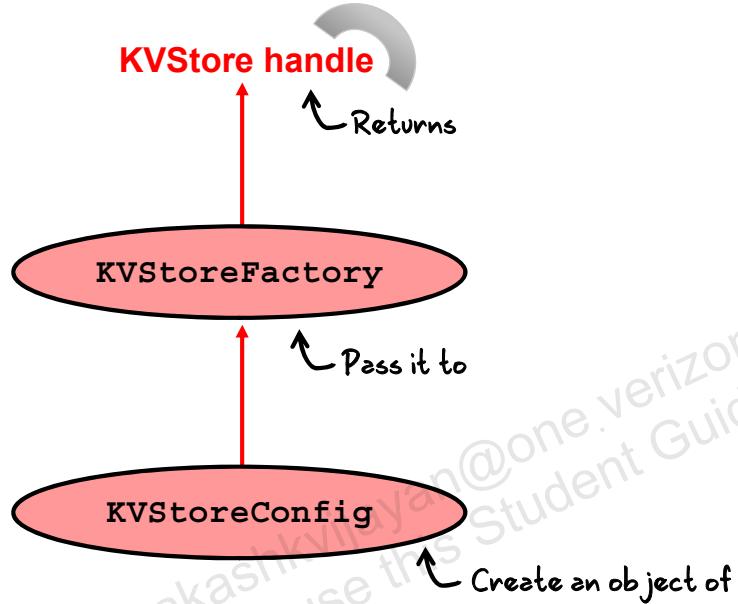
Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Before you perform any kind of access to Oracle NoSQL Database, you need to obtain a KVStore handle. A KVStore handle is a resource that is required to access the KVStore and is also used to open or close an already-running KVStore.

You need to access the KVStore whenever you perform the following operations in Oracle NoSQL Database:

- Create
- Read
- Update
- Delete

## Creating a KVStore Handle



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To create a KVStore handle, you use the `KVStoreFactory` and `KVStoreConfig` classes. You first create an object of the `KVStoreConfig` class. You then pass this `KVStoreConfig` object to the `KVStoreFactory` class, which returns the KVStore handle. When a KVStore handle is obtained, the store is automatically opened.

## Using the KVStoreConfig Class

The following information about a KVStore can be viewed or set using the KVStoreConfig class:

- Store name
- Request timeout
- Request limit
- Helper host names and port numbers
- Default store durability policy
- Default store consistency policy

KVStoreConfig

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

An object of the KVStoreConfig class contains important information about the KVStore you are accessing. You use the KVStoreConfig class methods to view or set the default configuration parameter values for a KVStore.

## KVStoreConfig Class Definition

```
public KVStoreConfig( string storeName,
                      string helperHostPort)
```

Methods to View Default Values	Methods to Set Default Values
getStoreName()	setStoreName()
getRequestLimit()	setRequestLimit()
getRequestTimeout()	setRequestTimeout()
getHelperHosts()	setHelperHosts()
getDurability()	setDurability()
getConsistency()	setConsistency()



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the KVStoreConfig constructor, which is used to create an instance of the KVStoreConfig class. The `storeName` parameter specifies the name of the KVStore. It must consist of uppercase or lowercase letters and digits. The `helperHostPort` parameter is a set of string values that specify the host name and port of an active node in the KVStore. Each string value must be in the format `hostname:port`. You must specify at least one helper host name and port.

The KVStoreConfig class has a list of methods that you can use to get and set `StoreName`, `Durability`, `Consistency`, `RequestTimeout`, and other parameters. For a complete list of these methods and their definitions, view the Javadoc from the installation documents.

## Using the KVStoreFactory Class

```
KVStore KVStoreFactory.getStore(KVStoreConfig config)
```

Method

KVStoreFactory

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The KVStorefactory class is a static class. It has one method, which is called getStore. The getStore method takes a KVStoreConfig object as the input parameter and returns the KVStore handle to the store, which is specified in the KVStoreConfig object.

## Creating a KVStore Handle: Example

```
import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;

public class OrclStore
{
    KVStore myStore;
    public static void main(String args[])
    {
        KVStoreConfig kconfig = new
            KVStoreConfig("orcl","localhost:5000");
        KVStore myStore = KVStoreFactory.getStore(kconfig);
    }
}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows a simple example to create a KVStore handle. The following packages should be imported into a Java class before creating a KVStore handle:

- oracle.kv.KVStore
- oracle.kv.KVStoreConfig
- oracle.kv.KVStoreFactory

Because this example is stand-alone code, the class has the `main` method defined.

First, an object of type `KVStoreConfig` is created and initialized.

- `kconfig` is the object name.
- `teachStore` is the name of the KVStore, which must already be running.
- `localhost:5000` is the host name and port number of the active node in the KVStore.

Then an object of type `KVStore` is created and initialized.

- `myStore` is the object name. The value for this object is obtained by calling the `getStore` method of the `KVStoreFactory` class and passing an object of type `KVStoreConfig`.
- `kconfig` is the object that is passed to the `getStore` method.

## Quiz

Which of the following returns a KVStore handle?

- a. KVStoreConfig.getStore()
- b. KVStoreConfig.getStoreName()
- c. KVStoreFactory.getStore()
- d. KVStoreFactory.getStoreName()



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: c**

## Creating Table Data

ORACLE®

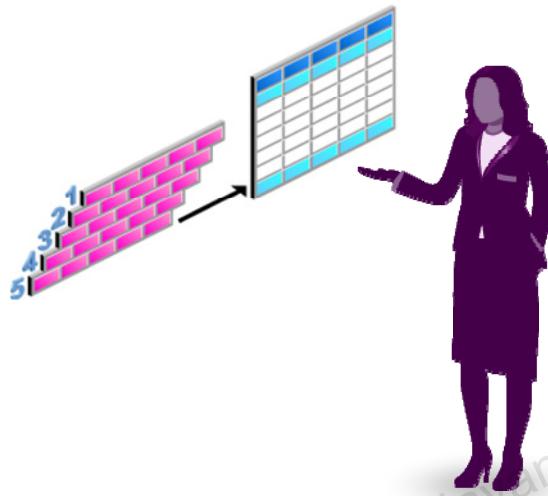
Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Write rows to a table
- Write to the child table
- Delete rows
- Use versions while writing rows

## Doris Populates Created Tables



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris has created the tables required for the Earnback application. Now she needs to write data into these tables. She needs to learn how to write records to a table as per her application requirements.

## TableAPI

- TableAPI is an interface to the tables in a KVStore.
- To perform any operation on tables, you should obtain an instance of TableAPI.

TableAPI Methods Summary			
3 for Deletes 4 for Inserts	5 for fetching records 3 for fetching keys only	2 for creating and executing transactions	Map<String, Table> getTables()  Table getTable(String tableName)



Discussed in this lesson      Discussed later

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In order to perform any operation on the tables created in a KVStore, you need to use a TableAPI Java interface written for ONDB. The TableAPI interface consists of a number of useful methods. The slide shows a summary of these methods. In this lesson, you learn how to use the methods available to perform insertion and deletion operations on a table. The methods to retrieve records are covered in the lesson titled “Retrieving Table Data” and the methods for creating transactions are covered in the lesson titled “Creating Transactions” in Unit III.

Apart from methods to write and read data and create transactions, TableAPI has two more methods, which are listed in the last column of the methods summary table in the slide. The getTables method returns a list of all the parent tables in the KVStore. The getTable method is used to obtain a handle to a table that you want to manipulate. You need to obtain this handle before performing any read or write operation to the table.

## TableAPI Methods for Write Operations

put	putIfAbsent	putIfPresent	putIfVersion
Writes to the table irrespective of whether the record is present or not	Writes to the table only if the record is not already present	Writes to the table only if the record is already present	Writes to the table only if the record present is the same as a specified version
<b>Definite write</b>	<b>Creation</b>	<b>Updation</b>	<b>Conditional Update</b>



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the four APIs available to write records to a table. Though all the methods are used to write records to a table, each of the methods has a specific functionality. The `put` method is used to ensure that the record you want to write is always written to the table irrespective of whether the same record is already present in the table. In other words, if the data is not present in the table, a new row is created. And in case the data is already present in the table, the row is updated with the field values you specify in the `put` operation.

The `putIfAbsent` method is used when you want to make sure that the write operation results in record creation only. If the same record exists in the table, then the write operation is not performed. Similarly, the `putIfPresent` method is used when you want to ensure that the write operation results in record updation only. If the same record is not present in the table, then the write operation is not performed.

Lastly, the `putIfVersion` method is used when you want to perform a write operation only if the version of the record in the table is same as the version specified in the write operation. If the versions do not match, then the write operation is not performed.

## Writing Rows to Tables: Steps

To write a row into a table, perform the following steps:

1. Obtain the TableAPI handle.
2. Construct the handle to the table.
3. Create a Row object.
4. Add field values to the row.
5. Write the row to the table.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To use the `put` methods discussed in the previous slide and to write records to a table, you have to code the steps listed in the slide. In the next few slides, you will look at these steps in detail.

## Constructing Handles

1. Obtain the TableAPI handle.
2. Construct the handle to the table.

```
TableAPI tableH = kvstore.getTableAPI();  
Table myTable = tableH.getTable("myTable");
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The first steps are to obtain the handles to the TableAPI interface as well as the table where you want to perform the write operation. The syntax for these commands is shown in the slide.

To obtain a handle to the TableAPI, you use a handle to the KVStore. Creating a KVStore handle was covered previously in the Unit II introduction.

Using the TableAPI handle, you obtain the handle to the required table by calling the getTable method. You need to pass the name of the table to which you want to write the data. This name should be the same as the name given to the table when it was created.

## Creating a Row Object, Adding Fields, and Writing a Record

3. Create a Row object.
4. Add field values to the row.
5. Write the row to the table.

```
Row row = myTable.createRow();

row.put("item", "Bolts");
row.put("description", "Hex head, stainless");
row.put("count", 5);
row.put("percentage", 0.2173913);

tableH.put(row, null, null);
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

After obtaining the required handles, you need to create a `Row` object for the table. Using the `put` method, specify the field name and field value for all the fields in the table. So far, a row object has been created and populated with the new row values. Next, you can write the data to the table by using an appropriate write method. In the example in the slide, the `put` method is used.

## Write Method Definitions

```
version put(Row row, ReturnRow prevRow, WriteOptions writeOptions)

version putIfAbsent(Row row, ReturnRow prevRow,
                   WriteOptions writeOptions)

version putIfPresent(Row row, ReturnRow prevRow,
                     WriteOptions writeOptions)

version putIfVersion(Row row, Version matchVersion,
                     ReturnRow prevRow, WriteOptions writeOptions)
```

```
WriteOptions(Durability durability,
             long timeout,
             TimeUnit timeoutUnit)
```

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The definitions of the four `write` methods are shown in the slide. After completing a write operation, each method returns the version of the record it just wrote. Each method takes as input three parameters: `Row`, `ReturnRow`, and `WriteOptions`. The `putIfVersion` method takes an additional parameter `Version`. Each of these parameters is described below.

- **Row:** This parameter holds the value of the record fields to be inserted into the table. You will learn more about this `Row` object in the next slide.
- **ReturnRow:** This parameter holds the value and version of the record before the write operation is performed. If this information is not required, you can specify the parameter value as null.
- **WriteOptions:** This parameter holds the durability requirements for the write operation. If you want to use the default durability configuration, you can specify the parameter value as null. You will learn how to configure durability for a write operation in the lesson titled “Configuring Durability” in Unit III.
- **Version:** This parameter is used in the `putIfVersion` method to hold the version that you want to match before writing the record.

# Creating the Row Object

```
Row Table.createRow()  
  
Row Table.createRow(RecordValue value)  
  
Row Table.createRowFromJson(InputStream jsonInput, boolean exact)  
  
Row Table.createRowFromJson(String jsonInput, boolean exact)  
  
Row Table.createRowWithDefaults()
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Using the Row Object

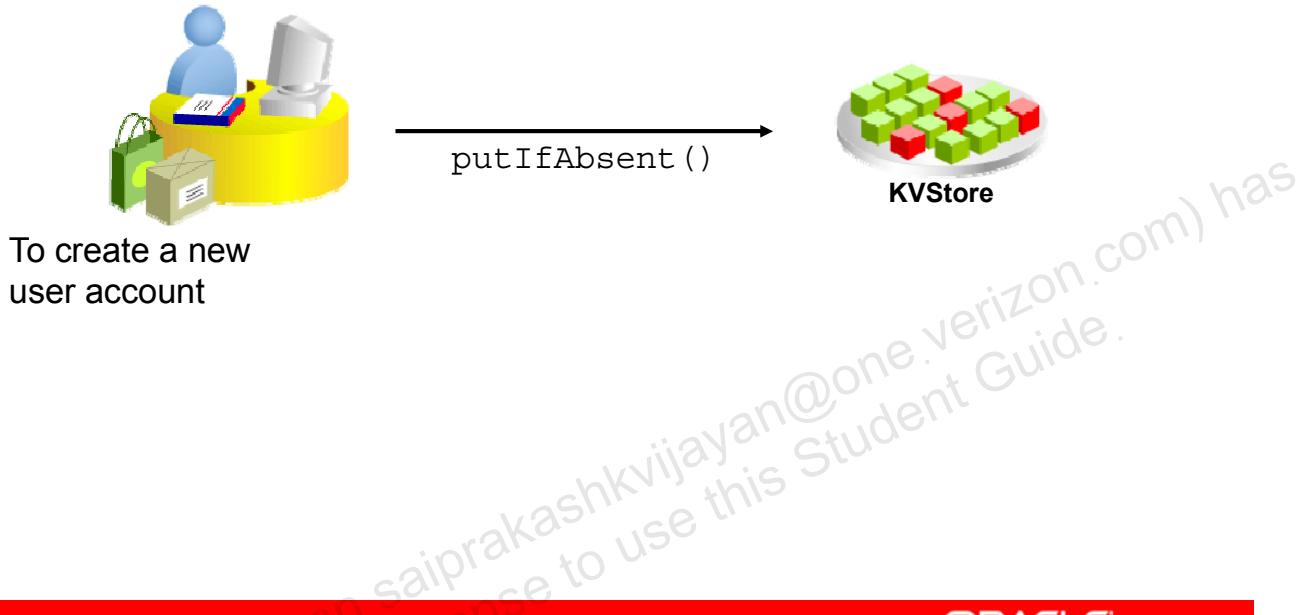
PrimaryKey	Row.createPrimaryKey()
FieldValue	Row.get(String fieldname)
RecordValue	Row.put()
RecordValue	Row.putEnum()
ArrayValue	Row.putArray()
RecordValue	Row.putRecord()
MapValue	Row.putMap()
RecordValue	Row.putFixed()
RecordValue	Row.putNull()
Table	Row.getTable()
Version	Row.getVersion()
Boolean	Row.equals(Object other)



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide lists the methods available for the Row object. The `createPrimaryKey()` method removes all other fields and creates a primary key from the Row object. The `get()` method returns the value associated with the specified field for that particular row. The various `put()` methods inserted the field values into the Row object. The `getTable()` method returns the handle to the table associated with this row. The `getVersion()` method returns the version of the row if it was retrieved from the KVStore. The `equals()` method compares the field values for two rows except the version details.

## putIfAbsent () : Use Case



ORACLE®

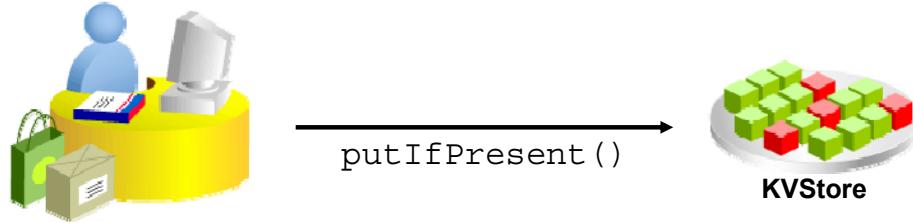
Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You will now understand when to use the `putIfAbsent` method by looking at a use case scenario. Suppose there is an online application where users can register and log in. You must make sure that each user is assigned a unique login `userId`. To uniquely identify each user record, you create a primary key on the `userId` field. Each time a new user account creation request is received, before you create the user in the KVStore, you need to match the entered username with the existing names in the `Users` table to ensure that the entered username does not already exist.

You can avoid this extra network trip to the KVStore by using the `putIfAbsent()` method. This method creates the new username record in the `Users` table only if the given username does not already exist in the primary key field. If there is another record with the same primary key, the method does not perform the write operation.

## putIfPresent(): Use Case

To reset a user's account password



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Consider again the online application where users can register and log in. If users forget their passwords, they need to submit a request to reset the password. When such a request is received, you must make sure that the submitted `userId` exists in the `Users` table before writing a new password for that user. Otherwise, you need to report back that the `userId` is invalid. To ensure this, you need to match the entered `userId` with all the `userId` records and ensure that the submitted `userId` is already present in the KVStore.

You can avoid this extra network trip to the KVStore by using the `putIfPresent()` method. This method creates a new password only if the submitted `userId` (primary key) record exists in the KVStore. If there is no record with the given primary key, the method does not perform the write operation.

## Quiz

You want to write sensor data to the KVStore. You want the write performance to be very fast and you need not worry about the previous value of a record. Which put method should you use?

- a. `put()`
- b. `putIfPresent()`
- c. `putIfAbsent()`
- d. `putIfVersion()`



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

## Writing Rows to Child Tables: Steps

To write a row into a child table, perform the following steps:

1. Obtain a TableAPI handle.
  2. Construct the handle to the parent table.
  3. Create a Row object.
  4. Add field values to the row.
  5. Write the row to the parent table.
  6. Construct the handle to the child table.
  7. Create a Row object.
  8. **Add a primary key of the parent table to the row.**
  9. Add field values to the row.
  10. Write the row to the child table.
- 
- Creating a parent table steps*



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Inserting data into the child table is similar to writing to the parent table. Ensure that the parent table record is written to the table. Then construct a handle to the child table and create the row object to be inserted. While adding the child table row fields, you should add the primary key of the parent table also. Finish adding the rest of the fields of the child table and write the row object to the child table.

## Writing Rows to Child Tables: Example

```
// row inserted into parent table

Table myChildTable = tableH.getTable("myTable.myChildTable");

Row childRow = myChildTable.createRow();

childRow.put("itemCategory", "Bolts");
childRow.put("itemSKU", "1392610");
childRow.put("itemDescription", "1/4-20 x 1/2 Grade 8 Hex");
childRow.put("price", new Float(11.99));
childRow.put("inventoryCount", 1457);

tableH.put(childRow, null, null);
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of a row object inserted into the child table. The creation and insertion of the row object for the parent table is not shown here.

## Table APIs for Delete Operations

delete	multiDelete	deleteIfVersion
Deletes a single record from a table	Deletes multiple records from tables	Deletes a single record from a table if the version of the record is the same as the specified version



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

There are three APIs available to delete records from a table. You use the `delete` method to delete a single record from the table. You can delete multiple rows from a table by using the `multidelete` command. The `multidelete` command uses a partial primary key to identify a set of records for deletion. If you want to match the version of a row before it is deleted, you use the `deleteIfVersion` method.

## Deleting Row(s) from a Table: Steps

To write a row into a table, perform the following steps:

1. Obtain a TableAPI handle.
2. Construct the handle to the table.
3. Create a primarykey object.
4. Specify primary key values to be deleted.
5. Delete the row from the table.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To delete a record from a table, you need to perform the steps listed in the slide. As in all other table operations, you need to obtain an instance of TableAPI and a handle to the table from which you want to delete the row or rows. You then create a primary key object and specify the value of the primary key for the rows you want to delete. After creating the primary key, you use the appropriate method to delete the rows.

## Delete Method Definitions

```
boolean delete(PrimaryKey key, ReturnRow prevRow,  
                WriteOptions writeOptions)  
  
int multiDelete(PrimaryKey key, MultiRowOptions getOptions,  
                 WriteOptions writeOptions)  
  
boolean deleteIfVersion(PrimaryKey key, Version matchVersion,  
                       ReturnRow prevRow, WriteOptions writeOptions)
```

```
PrimaryKey primaryKey = myTable.createPrimaryKey();  
primaryKey.put("item", "Bolts");
```

```
MultiRowOptions(FieldRange fieldRange, List<Table> ancestors,  
                List<Table> children)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The definitions of the three delete methods are shown in the slide. The single row delete methods (delete and deleteIfVersion) return a Boolean value specifying whether the delete operation was successful or not. The multiDelete method returns a count of the number of records it deleted. Each of these three methods takes as input three parameters: PrimaryKey, ReturnRow, and WriteOptions. The deleteIfVersion method takes an additional parameter Version. The multiDelete method takes an additional parameter called MultiRowOptions. Each of these parameters is described below.

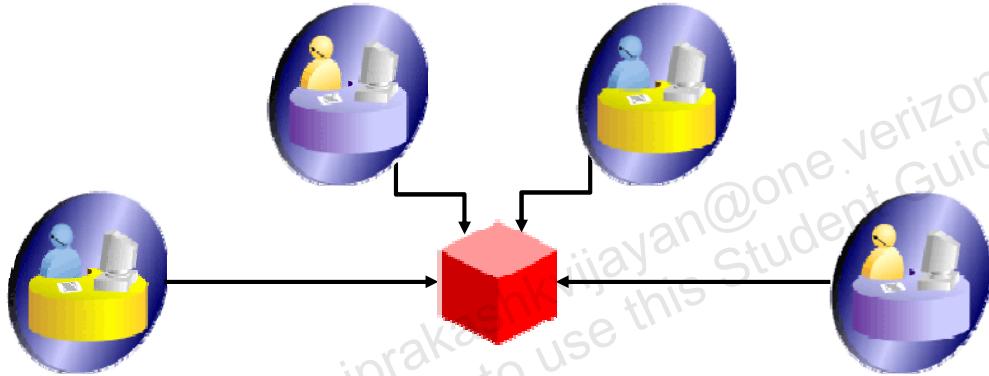
- **PrimaryKey:** This parameter holds the value of the primary key for the records to be deleted from the table. An example of a primary key is shown in the slide.
- **ReturnRow:** This parameter holds the value and version of the record before the delete operation is performed. If this information is not required, you can specify the parameter value as null.
- **WriteOptions:** This parameter holds the durability requirements for the delete operation. If you want to use the default durability configuration, you can specify the parameter value as null. You will learn how to configure durability in the lesson titled “Configuring Durability” in Unit III.

- **Version:** This parameter is used in the `deleteIfVersion` method to hold the version that you want to match before deleting the record.
- **MultiRowOptions:** This parameter is used to specify the range or depth of the child table records to be deleted. The definition of the `MultiRowOptions` is shown in the slide. You learn more about this parameter in the next lesson titled “Retrieving Records from Tables.”

## Scenario

In an application, multiple users should be able to access the same data at the same time.

Suppose you want to retrieve a record and update it with new data. How can you ensure that the record is not updated by another user between the time you read the record and the time you update the record?



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Consider the scenario described in the slide. After reading the next slide, you should be able to implement a solution for this scenario.

# Versions



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Oracle NoSQL Database maintains a version token for each record when the record is created for the first time and every time it is updated.

This version information is important for two reasons:

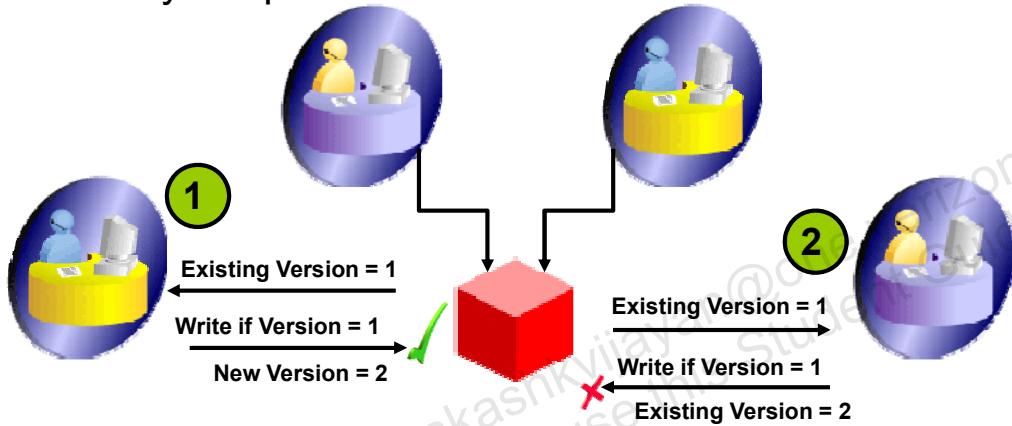
- When performing an update or a delete operation, you might want to perform the operation only if the record's value has not changed.
- When performing a read operation, you might want to ensure that you are reading a value that was previously written.

It is important to understand that the KVStore does not store various versions of a record. At any given time, only one version of each record is stored in the database and is assigned a unique version token. This version token is an incremental integer number.

## Scenario

In an application, multiple users should be able to access the same data at the same time.

Use the Versions feature to ensure that a record is not updated by another user between the time that you read the record and the time that you update the record.



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Recall the scenario discussed previously. You need to use the versioning capability of Oracle NoSQL Database to implement a solution for this scenario. Another point to be noted is that in case the operation failed due to version mismatch, you should try a retry logic for some N number of times. If the retries do not result in a successful operation, then you should throw an exception.

## Summary

In this lesson, you should have learned how to:

- Write rows to a table
- Write to the child table
- Use versions while writing rows
- Delete rows



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Practice 7 Overview: Writing Data to Tables

This practice covers the following topics:

- Inserting rows to parent and child tables
- Deleting rows



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This is a hands-on practice.

Unauthorized reproduction or distribution prohibited. Copyright© 2016, Oracle and/or its affiliates.

Sai Vijayan S (sai.vijayan.saiprakashkvijayan@one.verizon.com) has  
a non-transferable license to use this Student Guide.

## Retrieving Table Data

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

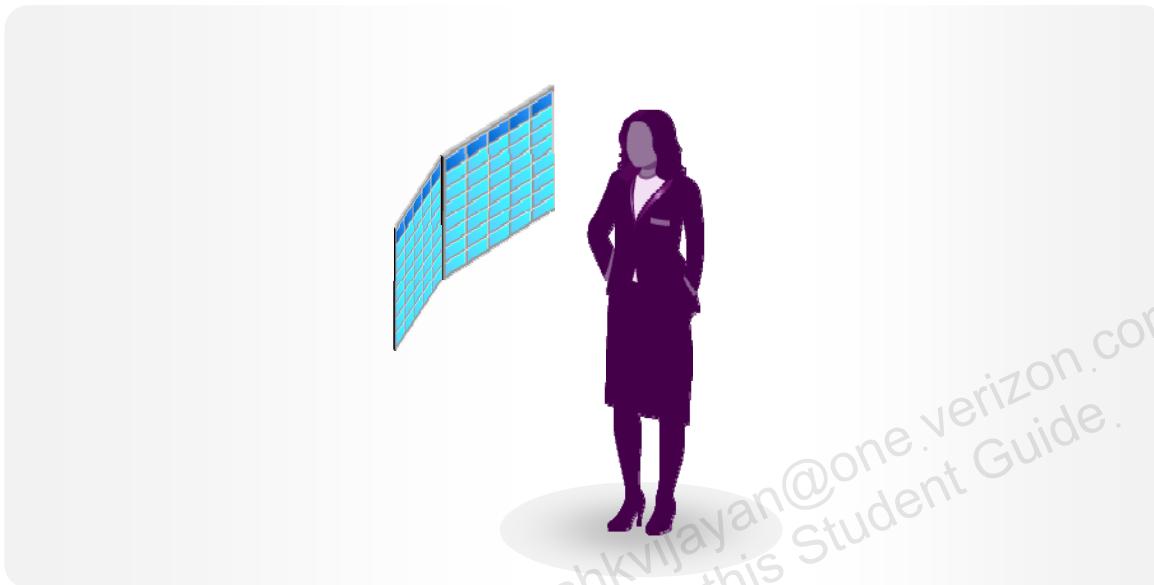
After completing this lesson, you should be able to:

- Retrieve a single row
- Retrieve child table rows
- Retrieve multiple rows
- Specify field ranges
- Read indexes



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Doris Reads Table Data



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris has created tables for the application and also learned to write to the tables using table APIs. There are several places in the application where she needs to read the data from the tables and process it. She needs to learn how to retrieve the table data and use it in the various processes.

## Reading a Table: Overview

get	multiGet	multiGetKeys	tableIterator
Retrieves a single record from the table	Retrieves a set of records from the table	Retrieves a set of record keys from the table	Retrieves a set of records or indexes from the table
complete key	complete shard key	complete shard key	partial primary key
-	atomic	atomic	non-atomic
-	-	single thread	multiple threads



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the four methods available to read records from tables in a KVStore.

- Use the `get()` method to retrieve a single record from the table. If the table has a composite primary key then you must specify the complete primary key to uniquely identify a single record in the table.
- Use the `multiGet()` method to retrieve more than one record from the table. This method fetches all the records in a single atomic operation. You should use this method only when you are sure that the result set will completely fit in the memory.
- If you have a very large set of records that need to be fetched, then you can use the `tableIterator()` method. This method fetches the records by iterating through the entire table. However, atomicity of the operation is lost. If you want to retrieve all the records from the table based on indexes, then also you use the `tableIterator()` method. This method can issue multiple threads to perform parallel scans of the table to improve the performance of the operation.

In the next few slides, you will learn each of these methods in detail.

## Retrieving Table Data: Steps

To fetch data from a table, perform the following steps:

1. Obtain a TableAPI handle.
2. Construct the handle to the table.
3. Create a primary key.
4. Use an appropriate method to fetch rows.
5. Retrieve field values from rows.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Similar to the write operations, to perform a read operation you need to follow the steps listed in the slide. First, create a TableAPI and a Table handle. Next, create a complete or partial primary key and pass to the appropriate get method to fetch a single or multiple records. You can then retrieve the field values from the fetched rows.

## Retrieving a Single Row

```
1 TableAPI tableH = kvstore.getTableAPI();  
2 Table myTable = tableH.getTable("myTable");  
3 PrimaryKey primaryKey = myTable.createPrimaryKey();  
primaryKey.put("item", "Bolts");  
4 Row row = tableH.get(primaryKey, null);  
5 String item = row.get("item").asString.get();  
String description = row.get("description").asString.get();  
Integer count = row.get("count").asInteger.get();  
Double percentage = row.get("percentage").asDouble.get();
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows a sample code to fetch a single record from a table:

1. Obtain a TableAPI handle.
2. Construct the handle to the table.
3. Create the primary key.
4. Use get().
5. Retrieve field values from the row.

## Retrieving Multiple Rows

```
ArrayList<Row> myRows = null;
myRows = tableH.multiGet(key, null, null);

for (Iterator<Row> iter = myRows.iterator(); iter.hasNext();)
{
    Row theRow = iter.next();
    String itemType = theRow.get("itemType").asString.get();
    String itemCategory = theRow.get("itemCategory").asString.get();
    String itemClass = theRow.get("itemClass").asString.get();
    String itemColor = theRow.get("itemColor").asString.get();
    String itemSize = theRow.get("itemSize").asString.get();
    Float price = theRow.get("price").asFloat.get();
    Integer price = theRow.get("itemCount").asInteger.get();
}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can retrieve multiple rows from a table by using the `multiGet()` method. You should use this method only when you know the resulting rows will fit completely in the memory. The initial steps are the same as retrieving a single row. You need to obtain a TableAPI handle and a handle to the table you want to retrieve. Then, create the partial primary key and use the `multiget()` method to fetch the rows. You have to use the methods within a try catch loop. Then, you can read the individual rows by using an iterator and retrieve each row and field value from the rows.

## Retrieving Child Tables

```
TableAPI tableH = kvstore.getTableAPI();

Table myChildTable = tableH.getTable("myTable.myChildTable");

PrimaryKey primaryKey = myChildTable.createPrimaryKey();
primaryKey.put("itemCategory", "Bolts");
primaryKey.put("itemSKU", "1392610");

Row row = tableH.get(primaryKey, null);

String description = row.get("itemDescription").asString().get();
Float price = row.get("price").asFloat().get();
Integer invCount = row.get("inventoryCount").asInteger().get();
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To retrieve a child table record, you perform the same steps as discussed in the previous slide. When you create the PrimaryKey object, you need to specify the primary key of the parent table as well as the parent key of the child table.

## Iterating a Table

```
TableIterator<Row> tableIterator( PrimaryKey key,  
                                MultiRowOptions getOptions,  
                                TableIteratorOptions Options)
```

```
TableIteratorOptions (Direction direction,  
                     Consistency consistency,  
                     long timeout,  
                     TimeUnit timeoutUnit,  
                     int maxConcurrentRequests,  
                     int batchResultsSize,  
                     int maxResultsBatches)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

If you want to iterate through an entire table or use indexes, then you use the `tableIterator` method to fetch rows. `tableIterator()` performs a non-atomic table iteration. That is, this method does not result in a single atomic operation. It does not return the entire set of rows all at once. Instead, it fetches rows in batches to minimize the number of network round trips and avoid monopolizing the available bandwidth. Since the retrieval is batched, the return set can change over the course of the entire retrieval operation. As a result, the atomicity of the operation is lost. If you do not provide a primary key, then the `tableIterator` will iterate over all of the table's rows. The `tableIterator` method takes as input the following parameters:

- **PrimaryKey:** This is a primary key object specifying a partial primary key common to the set of rows you want to retrieve.
- **MultiRowOptions:** Use this to specify a field range and the ancestor and parent tables you want to include in this iteration. You will look at this parameter in detail in the next slide.

- **TableIteratorOptions:** Use this to identify the number of keys to fetch during each network round trip. If you provide a value of 0, an internally determined default is used. You can also use this parameter to specify a traversal order while retrieving the rows. FORWARD, REVERSE, and UNORDERED are the two options available. You also use this parameter to control how many threads are used to perform the store read. By default, this method performs a single-threaded retrieval of table rows. You might be able to achieve better performance by using parallel scans, which use multiple threads to retrieve rows from the store. Parallel scans are discussed later in this lesson.

## Using MultiRowOptions

### MultiRowOptions

```
MultiRowOptions(FieldRange fieldRange,  
                 List<Table> ancestors,  
                 List<Table> children)
```

multiGet  
multiGetKeys  
tableIterator  
tableKeysIterator  
multiDelete



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

MultiRowOptions is used with methods working on multiple table rows. A list of these methods is shown in the slide as a quick recap.

MultiRowOptions is used to specify whether the operation should affect (return or delete) records from ancestor and/or descendant tables for matching records.

MultiRowOptions can also be used to specify subranges within a table or index for all operations it supports using FieldRange.

## Specifying Ranges

```
FieldRange fh = myTable.createFieldRange("familiarName");
fh.setStart("Bob", true);
fh.setEnd("Patricia", true);
MultiRowOptions mro = fh.createMultiRowOptions();

TableIterator<Row> iter = tableH.tableIterator(key, mro, null);

while (iter.hasNext())
{
Row row = iter.next();
}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

When performing multi-key operations in the store, you can specify a range of rows to operate upon. Use the `FieldRange` parameter in the methods performing bulk reads. This parameter is used to restrict the selected rows to those matching a range of field values.

To specify a range, you create an instance of the `FieldRange` class by using `createFieldRange()`. This method takes one argument: the name of the primary key for which you want to set the range. You can then set the start and end values for the range. You can specify whether the range values are inclusive.

The `FieldRange` object is used in the methods retrieving multiple rows. It is passed to some methods using a `MultiRowOptions` class instance. This is created using the `createMultiRowOptions()` method.

## Retrieving Nested Tables

```
tableH = kvstore.getTableAPI();

prodTable = tableH.getTable("prodTable");
categoryTable = tableH.getTable("prodTable.prodCategory");
itemTable = tableH.getTable("prodTable.prodCategory.item");

PrimaryKey key = prodTable.createPrimaryKey();

MultiRowOptions mro = new MultiRowOptions(null, null,
Arrays.asList(categoryTable, itemTable));

TableIterator<Row> iter = tableH.tableIterator(key, mro, null);
while (iter.hasNext()) {
Row row = iter.next();
displayRow(row);
}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

When you are iterating over a table, or performing a multi-get operation, rows are retrieved from the table on which you are operating. You can use `MultiRowOptions` to specify that parent and child tables are to be retrieved as well. When you do this, parent tables are retrieved first, then the table you are operating on, and then child tables. In other words, the tables' hierarchical order is observed. The parent and child tables retrieved are identified by specifying a list of table objects to the `ancestors` and `children` parameters on the class constructor. You can also specify these using the

`MultiRowOptions.setIncludedChildTables()` or  
`MultiRowOptions.setIncludedParentTables()` methods. When operating on rows retrieved from multiple tables, it is your responsibility to determine which table the row belongs to.

Now you need `PrimaryKey` and `MultiRowOptions` that you will use to iterate over the top-level table. Because you want all the rows in the top-level table, you create an empty `PrimaryKey`.

MultiRowOptions identifies the two child tables in the constructor's child parameter. This causes the iteration to return all the rows from the top-level table, as well as all the rows from the nested children tables. The `displayRow()` method is used to determine which table a row belongs to, and then display it in the appropriate way.

Note that the retrieval order remains the top-most ancestor to the lowest child, even if you retrieve by lowest child.

## Reading Indexes

```
TableIterator<Row> tableIterator( IndexKey key,  
                                MultiRowOptions getOptions,  
                                TableIteratorOptions Options)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

If you want to retrieve table rows using a table's indexes, use the `tableIterator` method. You use the method in the same way as you use it to retrieve rows using primary keys. The only difference is that you provide an instance of `IndexKey` instead of a primary key. You use the rest of the parameters to specify the range and other options.

To read rows using an index, you use the `getIndex()` method to retrieve the index name. You then create an `IndexKey` from the index object.

You can limit the rows returned by using a field range. You do that by using `Index.createFieldRange()` to create a `FieldRange` object. You must specify the field to base the range on. Recall that an index can be based on more than one table field, so the field name you give the method must be one of the indexed fields.

## Parallel Scans

```
TableIteratorOptions (Direction direction,
                      Consistency consistency,
                      long timeout,
                      TimeUnit timeoutUnit,
                      int maxConcurrentRequests,
                      int batchResultsSize,
                      int maxResultsBatches)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Reads are usually performed one partition at a time, in sequence, until all the desired rows are retrieved. This has obvious performance implications if you are retrieving a large number of rows that span multiple shards (such as might occur if you are iterating over an index, or even over a partial primary key). However, you can speed up the read performance by using parallel scans.

A parallel scan retrieves records from each shard in parallel and allows the client to receive and process them in parallel. You can specify how many threads to use to perform the retrieval. If more threads are specified on the client side, then the user can expect better retrieval performance—until processor or network resources are saturated.

To specify that a parallel scan is to be performed, you use `TableIteratorOptions` to identify the maximum number of client-side threads to be used for the scan, as well as the number of results per request and the maximum number of result batches that the Oracle NoSQL Database client can hold before the scan pauses. You pass this to `TableAPI.tableIterator()`. This creates a `TableIterator` that uses the specified parallel scan configuration.

## Quiz

You want to count the number of rows in a particular table.  
Which of the following methods should you use?

- a. get()
- b. multiGet()
- c. multiGetKeys()
- d. tableIterator()



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: d**

# Versions

- Version is a unique token assigned to a row when it is written.
- Version information is important:
  - To ensure data has not changed since the last read while performing update or delete operations
  - To ensure data is updated with the latest record that was written while performing a read operation



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Versions were introduced in the previous lesson. When a row is initially inserted in the store, and each time it is updated, it is assigned a unique version token. The version is always returned by the method that wrote to the store. The version information is also returned by methods that retrieve rows from the store.

Version information is very useful to an application. When an update or delete is to be performed, it may be important to only perform the operation if the row's value has not changed. This is particularly interesting in an application where there can be multiple threads or processes simultaneously operating on the row. In this case, read the row, examining its version when you do so. You can then perform a put operation, but only allow the put to proceed if the version has not changed. You use the version information in the `putIfVersion()` or `deleteIfVersion()` methods to perform the write or delete operation.

Version information is also useful when a client reads data that was previously written. It may be important to ensure that the Oracle NoSQL Database node servicing the read operation has been updated with the information previously written. This can be accomplished by passing the version of the previously written data as a consistency parameter to the read operation.

## Summary

In this lesson, you should have learned how to:

- Retrieve a single row
- Retrieve child table rows
- Retrieve multiple rows
- Specify field ranges
- Read indexes



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Practice 8 Overview: Reading Data from Tables

This practice covers the following topics:

- Retrieving a row
- Retrieving multiple rows



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This is a hands-on practice.

# Using Key-Value APIs

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

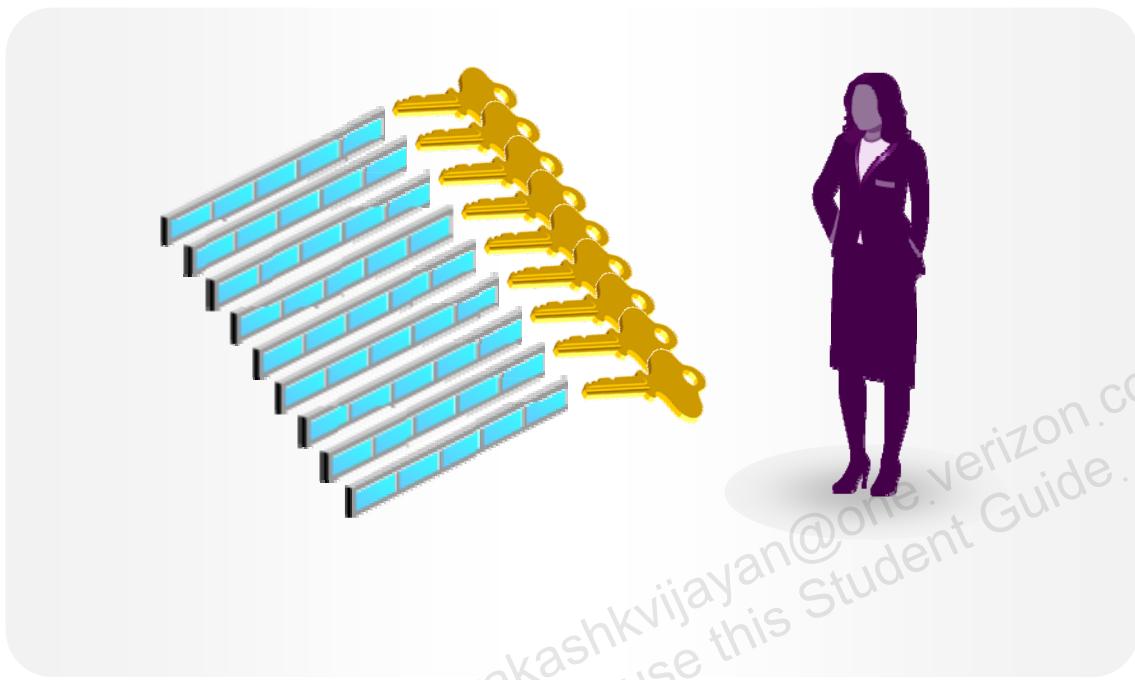
After completing this lesson, you should be able to:

- Create key and value components
- Identify APIs to write key-value pairs to a KVStore
- Perform a create and update operation
- Delete records
- Use versions
- Identify APIs to read key-value pairs from a KVStore
- Use APIs to read single and multiple key-value pairs
- Specify subranges



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Doris Explores the Key-Value Model

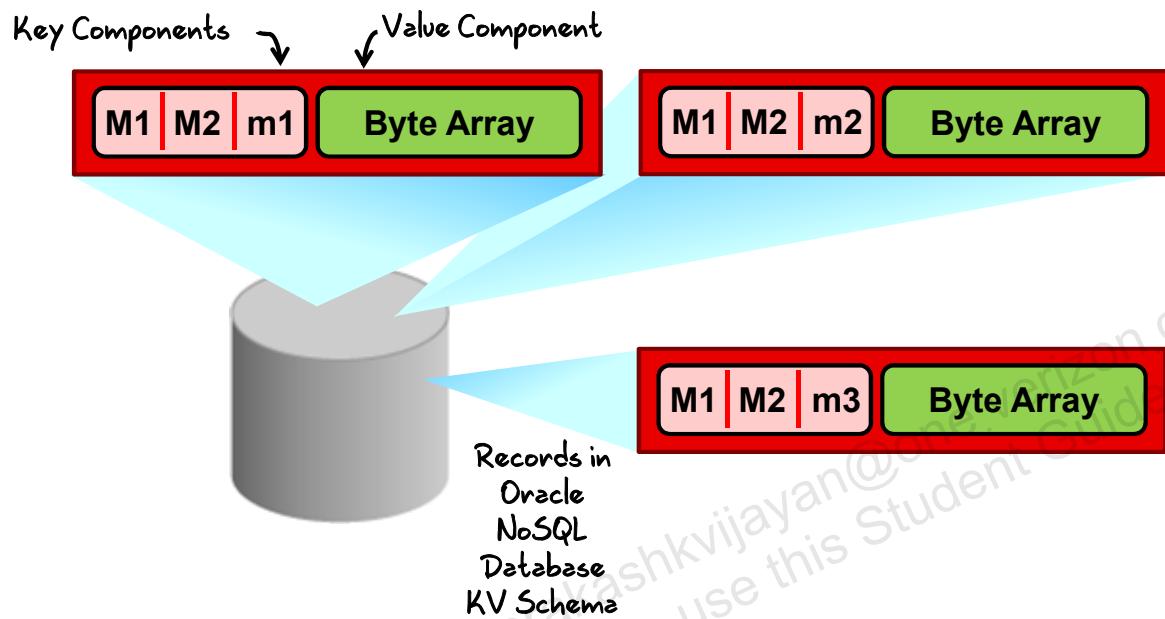


ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris has learnt how to read and write records by using the Table model. To complete her knowledge and understanding of Oracle NoSQL Database, she decides to examine how to work with the key-value pair APIs as well.

## Structure of a Record: Review



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Recall that in a key-value schema, a record consists of a key component and a value component. Also, a key component can consist of major and minor components. If a record consists of major and minor components, each unique combination of major and minor components results in a single record.

For example, the slide illustrates a case where there are two major components and one minor component with three different values. In the next few slides, you learn how to create the key and value components of a record.

# Creating a Key Component: Overview

Methods to View Default Values	Structure
createKey(String majorComponent)	M
createKey(List<String> majorPath)	M1/M2/M3 ...
createKey(String majorComponent, String minorComponent)	M - m
createKey(List<String> majorPath, List<String> minorPath)	M1/M2 ... - m1/m2 ...
createKey(List<String> majorPath, String minorComponent)	M1/M2 ... - m
createKey(String majorComponent, List<String> minorPath)	M - m1/m2 ...



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You create a key component whenever you need to perform a create, read, update, or delete operation for a key-value pair in a KVStore. Each key-value pair you want to operate on is identified by the key component.

You use the `createKey` method of the `Key` class to create a key component. The `createKey` method has various definitions. The method that is invoked depends on the parameters you pass to the `createKey` method. The slide lists all the method definitions and the structure of the key components that you can create.

## Creating Key Components

```
List<String> majorPath = new ArrayList<String>();  
majorPath.add("Smith");  
majorPath.add("Bob");
```

```
List<String> minorPath = new ArrayList<String>();  
minorPath.add("info");  
minorPath.add("personal");
```

```
Key myKey = Key.createKey(majorPath, minorPath);  
  
Key myKey = Key.createKey(majorPath, 'info');  
  
Key myKey = Key.createKey('smith.bob@abc.com', minorPath);
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The simplest way to create major components is to define an array of the data type `String`. As a naming-convention best practice, you can use `majorPath` as the name of the array to store the major-key components. After the array is defined, you can add the major-key component values to the array. The example in the slide adds `Smith` and `Bob` to the `majorPath` array.

It is mandatory that you define the data type as `String`, because the key component in Oracle NoSQL Database is restricted to the `String` data type.

Creating the minor components is similar to creating the major components. You should define an array of the data type `String`. As a naming-convention best practice, you can use `minorPath` as the name of the array to store the minor-key components. After the array is defined, you can add the minor-key component values to the array. The example in the slide adds `info` and `personal` to the `minorPath` array.

**Note:** You use an `ArrayList` when the major or minor key has more than one component. If the major or minor key has only one component, you can directly specify the `String` value to the `createKey` method.

After you have defined the major-key and minor-key components, use the `createKey` method to create the key. The slide shows some examples of using the `createKey` method.

## Creating a Value Component: Overview

- Oracle NoSQL Database can store information in the value component of a record in byte format only.
- You should implement appropriate serializing and de-serializing techniques.
- Use an AVRO schema to store values in JSON format



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Recollect that all information you want to store as the value component of a record should be converted into byte format.

You should write the appropriate code for serializing and de-serializing the value component. You can also use AVRO schemas to define a value component in JSON format.

## Creating a Value

Use the `createValue` method of the `Value` class.

```
List<String> data = new ArrayList<String>();  
  
data.add("35");  
data.add("male");  
data.add("developer");  
  
Value myValue = Value.createValue(data.getBytes());
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The value component of a record is created by using a `createValue` method. Before using this method, create the data content of the record. Either you create a variable of the `String` data type and store the data in it, or you create an `ArrayList` and add all the data values to the array.

In the example in the slide, an `ArrayList` called `data` of the `String` data type is created. The data values are added to this `ArrayList`.

The `createValue` method accepts values of the `byte` data type and creates a value that can be stored in a KVStore.

## Quiz

Which of the following statements is true about keys in Oracle NoSQL Database?

- a. Major-key and minor-key components are stored in a single ArrayList of the String data type.
- b. Minor-key components cannot be stored in an ArrayList.
- c. The structure of key components is completely application- and developer-defined.
- d. Key components can be of only the String data type.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: d**

## Retrieving Records: Overview

API	Usage
<code>get()</code>	Used to retrieve the value associated with the specified key
<code>multiGet()</code>	Used to retrieve multiple records that will fit entirely in the available memory
<code>multiGetIterator()</code>	Used to retrieve multiple records in batches
<code>storeIterator()</code>	Used to retrieve all or some records from the entire store



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

There are four methods that are available to retrieve records from a KVStore.

- Use the `get()` method when you want to fetch the value associated with a single key.
- Use the `multiGet()` method to fetch key-value pairs that are the descendants of a parent key that has a complete major path. The result set should fit in memory.
- Use the `multiGetIterator()` method to fetch key-value pairs that are descendants of a parent key that has a complete major path but has a result set that is too large to fit in memory.
- Use the `storeIterator()` method to fetch key-value pairs that are descendants of a parent key that is null or has a partial major path.

## get () Method

```
ValueVersion get(Key key, Consistency consistency,  
                 long timeout, TimeUnit timeoutUnit)
```

```
ValueVersion vv = get(myKey);  
Value v = vv.getValue();  
Version vr = vv.getVersion();
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the definition of the `get ()` method. The only mandatory argument to this method is the key component of the record whose value you want to retrieve. The rest of the parameters are optional. If you do not specify the optional parameters, the default values are used. You learn more about these parameters in Unit III.

The key component that you specify in this method must be a complete set of major-key and minor-key components that uniquely identifies a single record. The method returns an object of class `ValueVersion`, which contains the value and version associated with the specified key. If there is no value attached to the specified key, a null is returned.

The slide shows an example of using the `get ()` method. You retrieve the value component of the record by using the `getValue ()` method. The `Value` object is in byte format. You can convert it to `String` and use it as required in an application. Similarly, you extract the version information of the record by using the `getVersion ()` method.

## multiGet() Method

```
SortedMap<Key, ValueVersion>
multiGet( Key parentKey,
          KeyRange subRange,
          Depth depth,
          Consistency consistency,
          long timeout,
          TimeUnit timeoutUnit)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You use the multiGet() method to retrieve multiple records. You should use this method:

- When the result set will fit entirely in the memory cache. If the results cannot all be held in memory at one time, an OutOfMemoryError or excessive Garbage Collection activity could result.
- When you know the complete major components path of the records you want to retrieve. This method is useful only when you have more than one minor-key component.
- When you want to perform a transactional operation. All the key-value records are fetched in a single operation. This ensures that the value of the records does not change within the scope of this fetch operation.

The only mandatory argument to this method is a complete major-key component that forms the parent key of the records to retrieve. You can restrict the number of rows fetched in multirow operations by specifying a key range.

The records are returned in a SortedMap. The keys in this map are the key components of the records. The version and value of the records are stored as values in this map. You can use the methods available in the SortedMap class to retrieve the required information from this map.

## multiGet(): Example

```
SortedMap<Key, ValueVersion> myrecords = null;  
  
myrecords = myStore.multiGet(myKey,null,null);  
  
for (SortedMap.Entry<Key, ValueVersion> entry :  
    myrecords.entrySet())  
{  
    ValueVersion vv = entry.getValue();  
    Value v = vv.getValue();  
    String data = new String(v.getValue());  
    System.out.println(data);  
}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Creating a Key Range

```
KeyRange(String start,  
         boolean startInclusive,  
         String end,  
         boolean endInclusive)
```

```
KeyRange kr = new KeyRange(" history ", true,  
                           " purchases ", true);
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

By creating a key range, you can identify a subset of keys to use from the actual matching set. To create a key range, use the `KeyRange` class. This class accepts `String` values to define a range for the key components immediately following a key that is used in a multiple get operation.

The slide shows the syntax for the `KeyRange` method and an example of how to use this class. The object `kr` can now be passed as a parameter to the multirow operations.

## Key Depth

Depth Parameter Value	Usage
<code>depth.CHILDREN_ONLY</code>	Selects only immediate children; does not select the parent
<code>depth.DESCENDENTS_ONLY</code>	Selects all descendants; does not select the parent
<code>depth.PARENT_AND_CHILDREN</code>	Selects immediate children and the parent
<code>depth.PARENT_AND_DESCENDENTS</code>	Selects all descendants and the parent



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In multirow operations, you specify the depth of records on which you want to operate by passing a parameter value. The parameter values are listed in the slide.

## Using multiGetIterator()

```
Iterator<KeyValueVersion>
multiGetIterator(Direction direction,
                int batchSize,
                Key parentKey,
                KeyRange subRange,
                Depth depth,
                Consistency consistency,
                long timeout,
                TimeUnit timeoutUnit)
```

Values for Direction Parameter	Action
Direction.FORWARD	Iterates in ascending key order
Direction.REVERSE	Iterates in descending key order
Direction.UNORDERED	Iterates in no particular key order



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Similar to the `multiGet()` method, the `multiGetIterator` method is used to retrieve multiple records. You should use this method:

- When the result set is so large that it does not fit entirely in the memory cache. This method retrieves records in batches.
- When you know the complete major components path of the records that you want to retrieve. This method is useful only when you have more than one minor-key component.
- When you do not want to perform a transactional operation. Because the records are fetched in batches, the return set can change over the course of the entire retrieval operation. Thus, you lose the atomicity of the operation.

You should specify the direction of traversal, the number of records to retrieve in each batch, and a complete major-key component that forms the parent key of the records to retrieve. The values for the `Direction` parameter are listed in the slide.

You can restrict the records that are fetched by specifying a subrange and depth. When you leave the optional parameters as null, the default values are used. The records are returned in `Iterator`.

## multiGetIterator(): Example

```
Iterator<KeyValueVersion> i = null;  
  
i = myStore.multiGetIterator(Direction.FORWARD, 0,  
                             myKey, null,  
                             Depth.DESCENDANTS_ONLY);  
  
while(i.hasNext())  
{  
    Value v = i.next().getValue();  
    String data = new String(v.getValue());  
    System.out.println(data);  
}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of using the `multiGetIterator()` method.

## storeIterator() Method

```
Iterator<KeyValueVersion>
storeIterator(Direction direction,
              int batchSize,
              Key parentKey,
              KeyRange subRange,
              Depth depth,
              Consistency consistency,
              long timeout,
              TimeUnit timeoutUnit)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The `storeIterator()` method has the same parameters as the `multiGetIterator()` method. Also, it functions like the `multiGetIterator()` parameter by retrieving the records in batches.

However, unlike the `multiGetIterator()` method, it does not require a complete major-key component path. It can fetch records that match a partial set of major-key components. Also, if the key path is set as null, it can iterate over the entire store.

## storeIterator(): Example

```
Iterator <KeyValueVersion> i = myStore.storeIterator
                           (Direction.UNORDERED, 0,
                            null, null, null);
while (i.hasNext())
{
    KeyValueVersion kvv= i.next();
    Key key = kvv.getKey();
    System.out.println(key.toString());
}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of using the `storeIterator()` method.

Note that no key has been passed to this method. Therefore, it iterates over the entire store and prints the key components of all the records.

## Methods Summary

multiGet()	multiGetIterator()	storeIterator()
Fetches multiple records	Fetches multiple records	Fetches multiple records
Stores all the records in memory in one go	Fetches records in batches	Fetches records in batches
Performs transactional operation	Not transactional	Not transactional
Requires complete major path	Requires complete major path	Accepts partial or null key major path
Returns key and value-version of each record in a sorted map	Returns key-value-version in an iterator	Returns key-value-version in an iterator



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The table in the slide summarizes the three methods that you can use to fetch multiple records.

## Quiz

For a particular transaction, you need to count all the keys in the store using a partial primary key. Which method should you use?

- a. multiGet()
- b. storeIterator()
- c. multiGetIterator()
- d. getAll()



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: b**

## Methods: Overview

API	Usage
<code>kvstore.put()</code>	To create or update a record in the KVStore
<code>kvstore.putIfAbsent()</code>	To ensure that a record is only created
<code>kvstore.putIfPresent()</code>	To ensure that a record is only updated
<code>kvstore.putIfVersion()</code>	To update a record if it matches the specified version
<code>kvstore.delete()</code>	To delete a record
<code>kvstore.multiDelete()</code>	To delete multiple records
<code>kvstore.deleteIfVersion()</code>	To delete a record only if it matches a specified version



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide lists a summary of the methods available to write and delete key-value pairs,

- `put()`: Writes a record to the KVStore by either creating a new record or overwriting an existing record as appropriate
- `putIfAbsent()`: Writes a record to the KVStore only if there is no existing record with the specified key (that is, always creates a new record)
- `putIfPresent()`: Writes a record to the KVStore only if there is already an existing record with the specified key (that is, always updates a record)
- `putIfVersion()`: Writes a record to the KVStore only if the existing record's version matches the Version argument specified with this method
- `delete()`: Deletes a single record from the KVStore based on a specified key
- `multiDelete()`: Deletes multiple records that share the same major-key component from the KVStore based on a specified parent key
- `deleteIfVersion()`: Deletes a single record from the KVStore if it matches a specified version token

## Writing Key-Value Pair to KVStore

```
Version put(Key key,  
           Value value,  
           ReturnValueVersion prevValue,  
           Durability durability,  
           long timeout,  
           TimeUnit timeoutUnit)
```

```
Version putIfAbsent( Key key, Value value,  
                     ReturnValueVersion prevValue,  
                     Durability durability,  
                     long timeout,  
                     TimeUnit timeoutUnit)
```

```
Version putIfPresent( Key key, Value value,  
                     ReturnValueVersion prevValue,  
                     Durability durability,  
                     long timeout,  
                     TimeUnit timeoutUnit)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the definition of the write methods.

The `put` method takes two mandatory arguments: the key part and the value part of the record to be inserted. It returns the version number of the inserted record. If the optional parameters are not specified, the default values are used.

Use the `putIfAbsent()` method to perform a create operation. If a record with the specified key is already present in the KVStore, the method returns without performing a write. The slide shows the syntax for the `putIfAbsent()` method. You should, at a minimum, specify the key and value information while using this method. If the rest of the parameters are not specified, the default values are used. A version token for the newly created record is returned.

Use the `putIfPresent()` method to perform an update operation. If a record with the specified key does not exist in the KVStore, the method returns without performing a write. The slide shows the syntax for the `putIfPresent()` method. The parameters are similar to the `putIfAbsent()` method.

## Deleting Records

```
boolean (Key key,  
        ReturnValueVersion prevValue,  
        Durability durability,  
        long timeout,  
        TimeUnit timeoutUnit)
```

```
Integer multiDelete( Key parentKey,  
                     KeyRange subRange,  
                     Depth depth,  
                     Durability durability,  
                     long timeout,  
                     TimeUnit timeoutUnit)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the definition of the `delete()` method. The only mandatory argument you should specify is the key of the record to be deleted. It returns a Boolean value depending on whether the operation was successful or not. That is, if the record was deleted, a `TRUE` value is returned. If there was no record to be deleted, a `FALSE` value is returned. Dependent records, if any, are not deleted using this method. If the optional parameters are not specified, the default values are used.

You call the `multiDelete()` method to delete multiple records that share the same major path component. This method reduces the number of trips to the KVStore that occur if you use the `delete()` method. The mandatory parameter for this method is the parent key. This key should be a complete major component path. You can specify a subrange and depth to restrict the number of records to be deleted. After the method completes the delete operation, it returns a count of the number of records that were deleted.

## Deleting a Set of Records: Example

```
List<String> majorPath = new ArrayList<String>();  
majorPath.add("emp");  
majorPath.add("japan");  
majorPath.add("abc@gmail.com");  
  
Key myKey = Key.createKey(majorPath);  
  
Integer count = myStore.multiDelete(myKey, null, null);  
  
System.out.println(count + "Records Deleted");  
myStore.close();
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of using the `multiDelete()` method.

# Working with Versions

Version	ValueVersion	KeyValueVersion
<code>put()</code>	<code>get()</code>	<code>multiGetIterator()</code>
<code>putIfPresent()</code>	<code>multiGet()</code>	<code>storeIterator()</code>
<code>putIfAbsent()</code>		
<code>putIfVersion()</code>		



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The version information of a key-value pair is returned by methods that read or write records to the KVStore. These methods are listed in the slide. The version information is returned as objects of the following three classes: Version, ValueVersion, and KeyValueVersion. You can retrieve the version information from the objects of each of these classes by using a `.getVersion()` method.

Use the `putIfVersion()` method to update a single record with a new value only if it matches a specified version. The method returns the new version of the record if the operation was successful. Otherwise, a null value is returned. The version value is also used in a delete operation. Use the `deleteIfVersion()` method to delete a single record from the KVStore only if it matches a specified version. The method returns a Boolean value specifying whether the operation was successful or not.

## Summary

In this lesson, you should have learned how to:

- Create key and value components
- Identify APIs to write key-value pairs to a KVStore
- Perform a create and update operation
- Delete records
- Use versions
- Identify APIs to read key-value pairs from a KVStore
- Use APIs to read single and multiple key-value pairs
- Specify subranges



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Practice 9 Overview: Working with Key-Value Data

This practice covers the following topics:

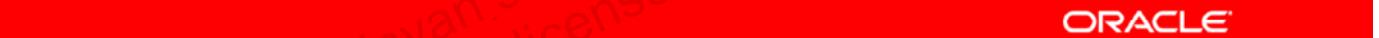
- Insert rows to parent and child tables
- Deleting rows



This is a hands-on practice.

## **Unit II**

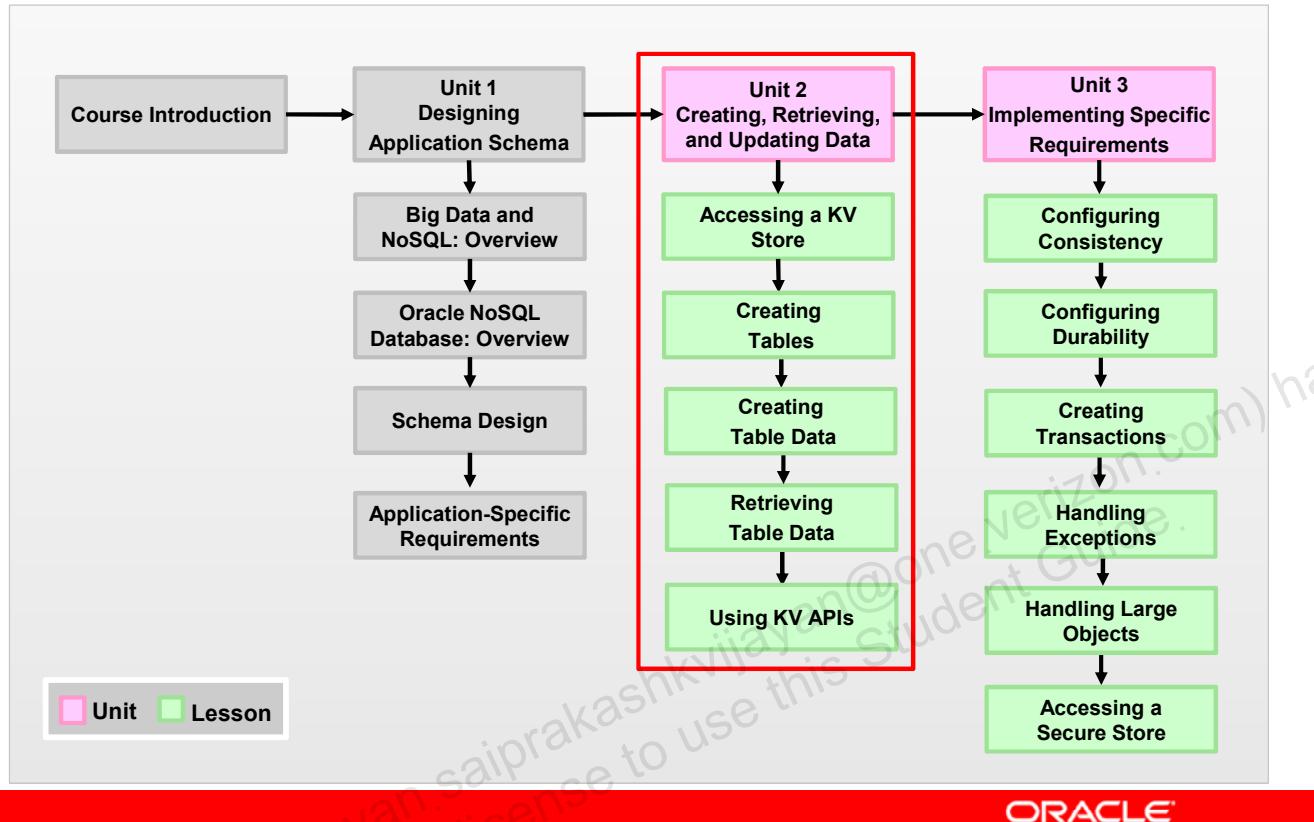
# **Summary of Creating, Retrieving, and Updating Data**



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Course Outline



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In Unit II, you completed five topics.

## Doris Has a Functional Application

### Earnback Application

- ✓ Read and write functionality working.



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris has created the application tables and is able to read and write data from and to the tables.

Unauthorized reproduction or distribution prohibited. Copyright© 2016, Oracle and/or its affiliates.

Sai Vijayan S (sai.vijayan.saiprakashkvijayan@one.verizon.com) has  
a non-transferable license to use this Student Guide.

## **Unit III**

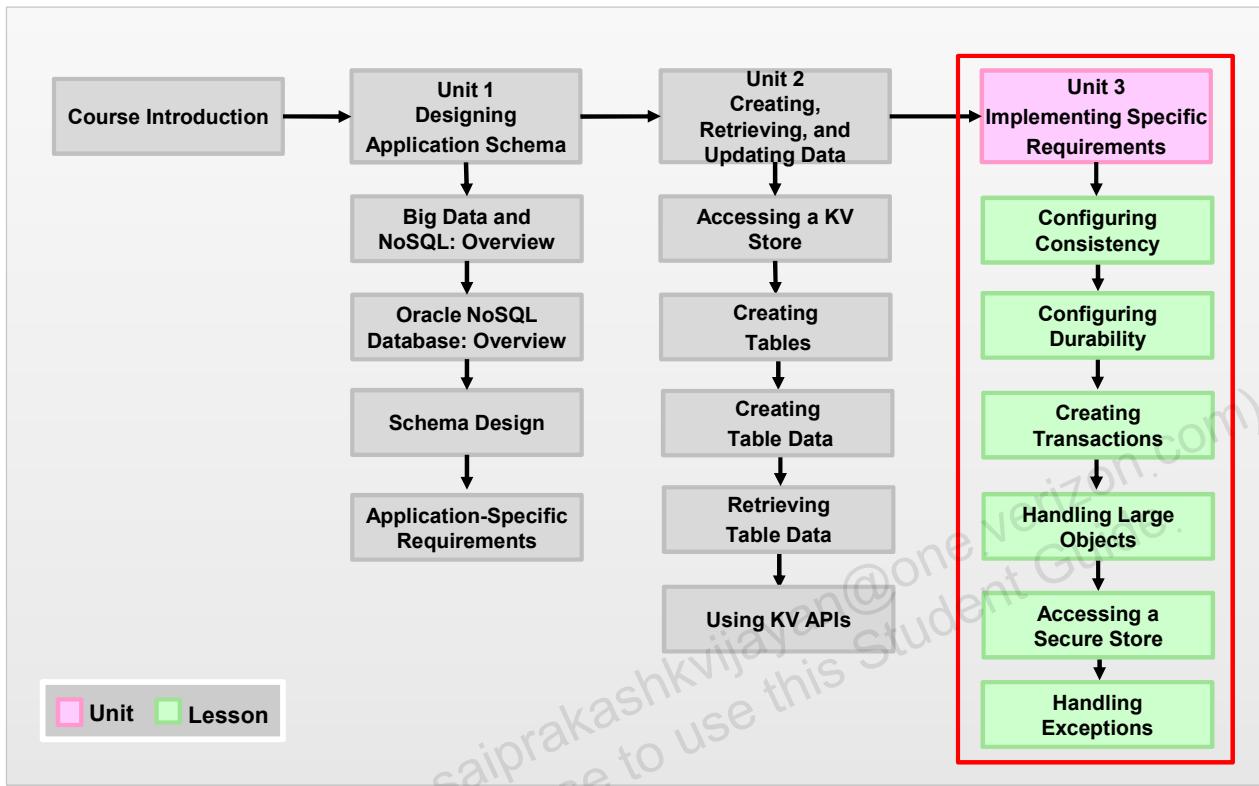
# **Implementing Application-Specific Requirements**



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Course Outline



## Doris Configures Application Requirements



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Doris and team now have a functional Earnback application. They now need to implement some specific requirements that include consistency, durability, transactional code, and security.

Unauthorized reproduction or distribution prohibited. Copyright© 2016, Oracle and/or its affiliates.

Sai Vijayan S (sai.vijayan.saiprakashkvijayan@one.verizon.com) has  
a non-transferable license to use this Student Guide.

# 10

## Configuring Consistency

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- View the default consistency policy
- Use predefined consistency policies
- Create time-based and version-based consistency policies
- Change the default consistency policy

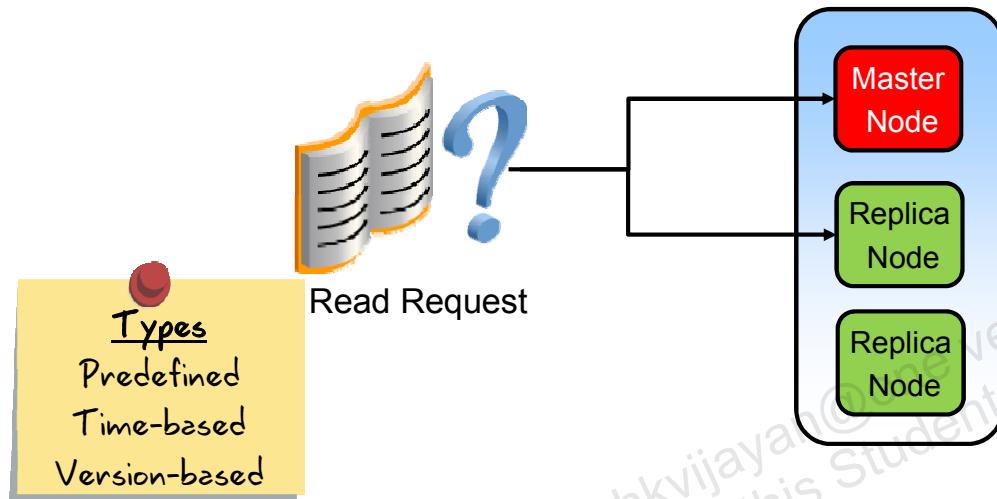


ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Consistency Policy: Review

A consistency policy determines whether to service a read request from a master node or from one of the replica nodes.



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You learned about the types of consistency in Unit I in the lesson titled “Application Specific Requirements.” In this lesson, you learn how to implement the different types of consistency in a Java program.

## Viewing the Default Consistency Policy

```
KVStoreConfig kconfig = new KVStoreConfig("orcl",
                                         "localhost",
                                         "5000");
System.out.println(kconfig.getConsistency());
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

When you set up a KVStore, a default consistency policy is applied to the entire store. The default policy is one of the predefined consistency policies called `Consistency.NONE_REQUIRED`. You can view this default value by calling the `getConsistency()` method of the `KVStoreConfig` class, as shown in the slide.

## Creating Consistency Policies: Overview

### ReadOptions

```
ReadOptions (Consistency consistency,  
             long timeout,  
             TimeUnit timeoutUnit)
```

get  
multiGet  
multiGetKeys

### TableIteratorOptions

```
TableIteratorOptions (Direction direction,  
                     Consistency consistency,  
                     long timeout,  
                     TimeUnit timeoutUnit,  
                     int maxConcurrentRequests,  
                     int batchResultsSize,  
                     int maxResultsBatches)
```

tableIterator  
tableKeysIterator

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The default consistency policy is applied to all the read operations for which no consistency parameter is explicitly specified. There may be cases where you want to specify a consistency policy for a particular read operation that differs from the default consistency policy for the KVStore.

In order to create consistency policies for read operations, you need to use either the `ReadOptions` class or the `TableIteratorOptions` class. Both these classes take consistency policies as parameters. The slide shows the read APIs that take these class objects as input parameters.

## Using a Predefined Consistency Policy

- Consistency.ABSOLUTE
- Consistency.NONE\_REQUIRED
- Consistency.NONE\_REQUIRED\_NO\_MASTER

```
ReadOptions ro = new ReadOptions(Consistency.ABSOLUTE,  
                                0, null);
```

```
TableIteratorOptions tio = new  
TableIteratorOptions(FORWARD,  
                     Consistency.NONE_REQUIRED_NO_MASTER,  
                     0, null);
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can specify the predefined consistency policies listed in the slide. To use the predefined consistencies in a read operation, you need to create an instance of the `ReadOptions` class or the `TableIteratorOptions` class depending on the read API you use. Pass the appropriate consistency policy as the `consistency` parameter. In the example in the slide, the `ABSOLUTE` consistency is used in `ReadOptions` and the `NONE_REQUIRED_NO_MASTER` consistency is used in `TableIteratorOptions`. A value of 0 for the `timeout` parameter specifies that the default timeout value is to be used for the operation. The `ReadOptions` or `TableIteratorOptions` object is then passed to the API performing the read operation.

## Creating a Time-Based Consistency Policy

```
Consistency.Time(long permissibleLag,  
                  TimeUnit permissibleLagUnit,  
                  long timeout,  
                  TimeUnit timeoutUnit)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You create a time-based consistency policy by using the `Consistency.Time` class. Its constructor takes the following parameters:

- `permissibleLag`: Specifies the amount of time that the replica is allowed to lag behind the master
- `permissibleLagUnit`: Specifies the unit of time used by `PermissibleLag`
- `timeout`: Specifies the amount of time to wait to achieve consistency
- `timeoutunit`: Specifies the unit of time used by the `timeout` parameter

The specified policy is then passed as a parameter to either `ReadOptions` or `TableIteratorOptions`, which is then passed to the read API used.

## Consistency.Time: Example

```
import oracle.kv.Consistency.Time;

Consistency.Time cpolicy = new Consistency.Time
    (2, TimeUnit.SECONDS, 4, TimeUnit.SECONDS);

ReadOptions ro = new ReadOptions(cpolicy,
    0, null);
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows a time-based consistency policy in which the replica can lag by two seconds and the waiting time is four seconds.

## Creating a Version-Based Consistency Policy

```
public Consistency.Version(Version version,  
                           long timeout,  
                           TimeUnit TimeoutUnit)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can create a version-based consistency policy by using the `Consistency.Version` class. The parameters specified in this policy are:

- `version`: Specifies the token that the read operation must match
- `timeout`: Specifies the amount of time to wait for the replica to be updated
- `timeoutUnit`: Specifies the unit of time for the `timeout` parameter

The specified policy is then passed as a parameter to either `ReadOptions` or `TableIteratorOptions`, which is then passed to the read API used.

## Consistency.Version: Example

```
import oracle.kv.Version;  
  
Consistency.Version vConsistency = new  
    Consistency.Version(matchVersion, 200,  
    TimeUnit.NANOSECONDS);
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows a version-based consistency policy in which the transaction waits 200 nanoseconds to match the `matchVersion` value before throwing an exception.

**Note:** `matchVersion` is a user-defined method that is written to capture the version information.

## Changing a Default Consistency Policy

```
KVStoreConfig kconfig = new KVStoreConfig("teachStore",
                                         "localhost",
                                         "5000");

kconfig.setConsistency(Consistency.ABSOLUTE);

System.out.println(kconfig.getConsistency());
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can change the default consistency policy that is applied to the entire KVStore. To do this, use the `setConsistency()` method of the `KVStoreConfig` class. The default consistency policy shown in the slide is set to the predefined consistency policy `Consistency.ABSOLUTE`. Any of the predefined consistencies or a time-based consistency can be set as the default consistency. Version-based consistencies are used only on a per-operation basis.

## Summary

In this lesson, you should have learned how to:

- View the default consistency policy
- Use predefined consistency policies
- Create time-based and version-based consistency policies
- Change the default consistency policy



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Practice 10 Overview: Setting Consistency Policies

This practice covers setting the following:

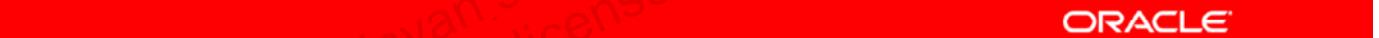
- A predefined consistency policy
- A time-based consistency policy

Unauthorized reproduction or distribution prohibited. Copyright© 2016, Oracle and/or its affiliates.

Sai Vijayan S (sai.vijayan.saiprakashkvijayan@one.verizon.com) has  
a non-transferable license to use this Student Guide.

# 11

## Configuring Durability



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- View the default durability policy
- Create synchronization-based and acknowledgment-based durability policies
- Change the default durability policy



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Durability: Review

A durability policy consists of two policies:

- Synchronization policy
- Acknowledgment policy



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

A durability policy is the sum of the synchronization policy at the master node, the synchronization policy for the replica nodes, and the acknowledgment policy.

## Viewing the Default Durability Policy

```
KVStoreConfig kconfig = new KVStoreConfig("teachStore",
                                         "localhost",
                                         "5000");
System.out.println(kconfig.getDurability());
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

When you set up a KVStore, a default durability policy is applied to the entire store. You view this default value by calling the `getDurability()` method of the `KVStoreConfig` class (as shown in the slide).

## Creating Durability Policies: Overview

### WriteOptions

```
WriteOptions (Durability durability,  
             long timeout,  
             TimeUnit timeoutUnit)
```

```
put  
putIfAbsent  
putIfPresent  
putIfVersion  
delete  
deleteIfVersion  
multiDelete  
execute
```

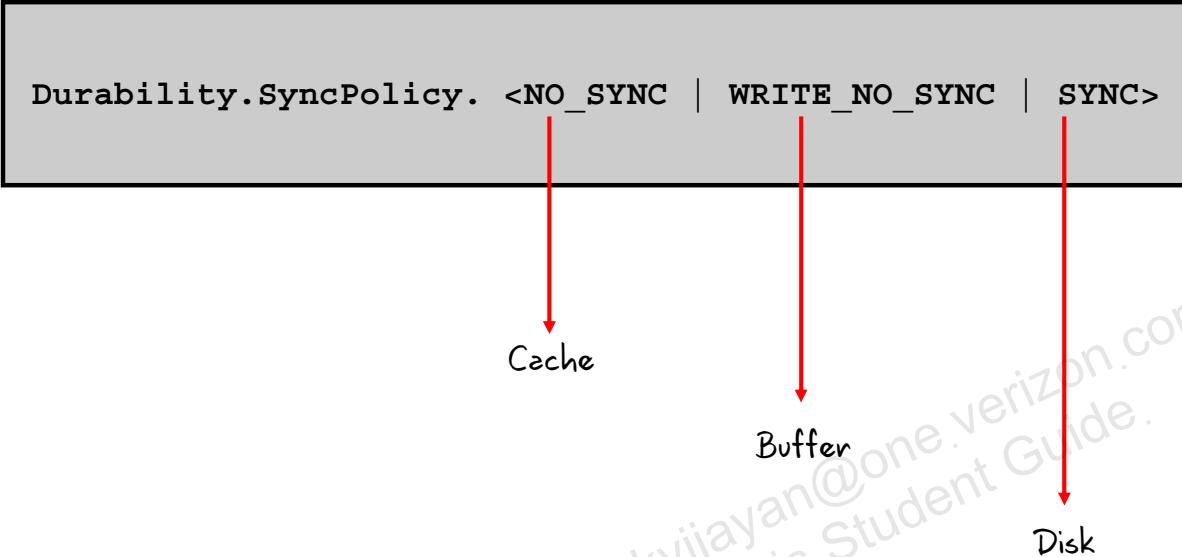
ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The default durability policy is applied to all the write operations for which no durability parameter is explicitly specified. There may be cases where you want to specify a durability policy for a particular write operation that differs from the default durability policy for the KVStore.

In order to create durability policies for write operations, you need to use the WriteOptions class, which takes durability policies as parameters. The slide shows the write APIs that take these class objects as input parameters. You learn the execute API in the lesson titled “Creating Transactions” next in this unit.

# Setting a Synchronization-Based Durability Policy



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To create a synchronization policy for the master or replica nodes, specify the durability parameter (as shown in the slide).

# Setting an Acknowledgment-Based Durability Policy

```
Durability.ReplicaAckPolicy.<NONE | SIMPLE_MAJORITY| ALL>
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To create an acknowledgment policy, specify the durability parameter (as shown in the slide).

## Creating a New Durability Policy

```
Durability(Durability.SyncPolicy masterSync,  
           Durability.SyncPolicy replicaSync,  
           Durability.ReplicaAckPolicy replicaAck)
```

```
Durability durability =  
    new Durability(Durability.SyncPolicy.NO_SYNC,  
                  Durability.SyncPolicy.NO_SYNC,  
                  Durability.ReplicaAckPolicy.NONE);  
  
WriteOptions wo = new WriteOptions(durability, 0, null);
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You create a new durability policy by creating an object of class `Durability`. The slide shows the constructor for the `Durability` class. You must specify the synchronization policy at the master node and replica node, as well as the acknowledgment policy.

The slide also shows an example of setting a durability policy. After you create a durability policy, you can pass the policy to the `WriteOptions` class's `Durability` parameter. The `WriteOptions` object is then passed to the appropriate write API.

## Changing the Default Durability Policy

```
Durability durability =
    new Durability(Durability.SyncPolicy.NO_SYNC,
                   Durability.SyncPolicy.NO_SYNC,
                   Durability.ReplicaAckPolicy.NONE);

kconfig.setDurability(durability);

System.out.println(kconfig.getDurability());
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You change the default KVStore's consistency by creating a new policy and using the `setDurability()` method of the `KVStoreConfig` class.

## Summary

In this lesson, you should have learned how to:

- View the default durability policy
- Create synchronization-based and acknowledgment-based durability policies
- Change the default durability policy



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Practice 11 Overview: Setting Durability Policies

This practice covers setting a durability policy.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This is a hands-on practice.

Unauthorized reproduction or distribution prohibited. Copyright© 2016, Oracle and/or its affiliates.

Sai Vijayan S (sai.vijayan.saiprakashkvijayan@one.verizon.com) has  
a non-transferable license to use this Student Guide.

# 12

## Creating Transactions

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

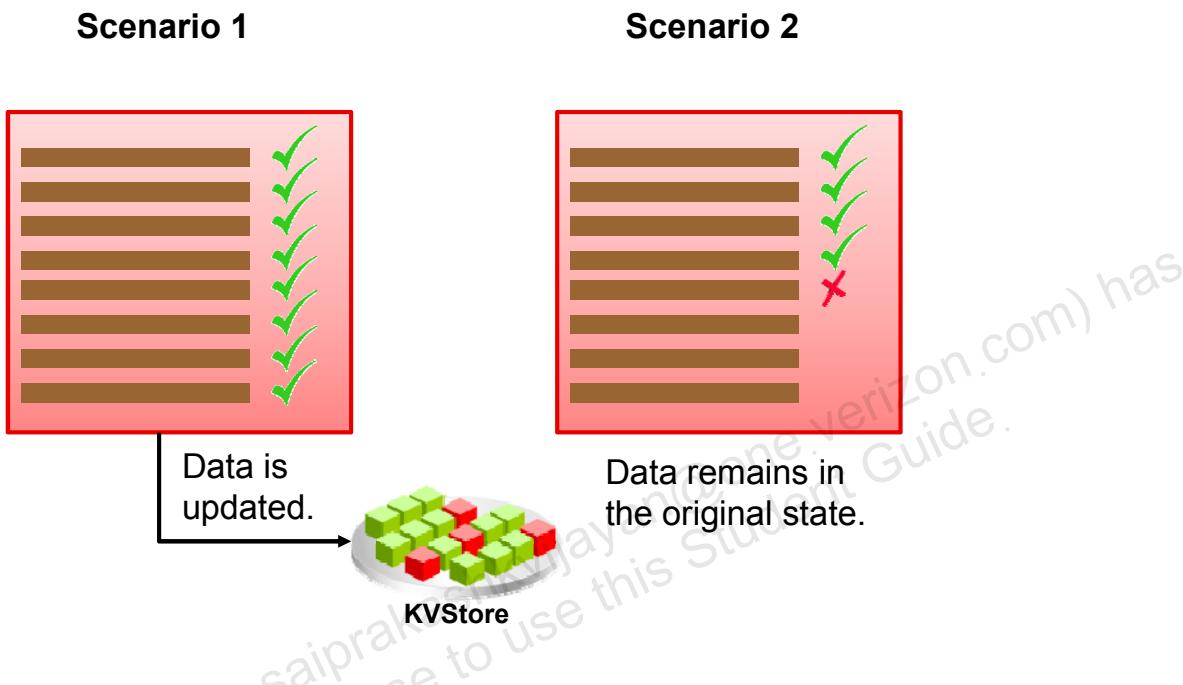
- Explain a transactional operation
- Create a transactional operation
- Execute a transactional operation



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# What Is a Transactional Operation?



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

A transactional operation is a sequence of write and/or delete operations that will all either succeed or fail. This sequence of operations is also referred to as an *atomic unit*.

The data that is in use by the transaction cannot be updated by any other process or operation. The updates made by the transaction's operations are not visible to any other process until the entire sequence of operations is completed.

If all the individual operations in the transaction complete successfully, the transaction is successful and the updates are visible in the KVStore. If any one operation in the transaction fails, the transaction aborts immediately and the data in the KVStore remains in the original state (as it was before the transaction started).

The operations in a transaction are not executed in the order you create them. Rather, they are executed in an internally defined order that prevents deadlock.

## Creating Transactions: Points to Remember

When creating a sequence of operations, you must ensure the following:

- Create only write or delete operations.
- All the operations must use the same shard key.
- Avoid creating two operations that operate on the same record.

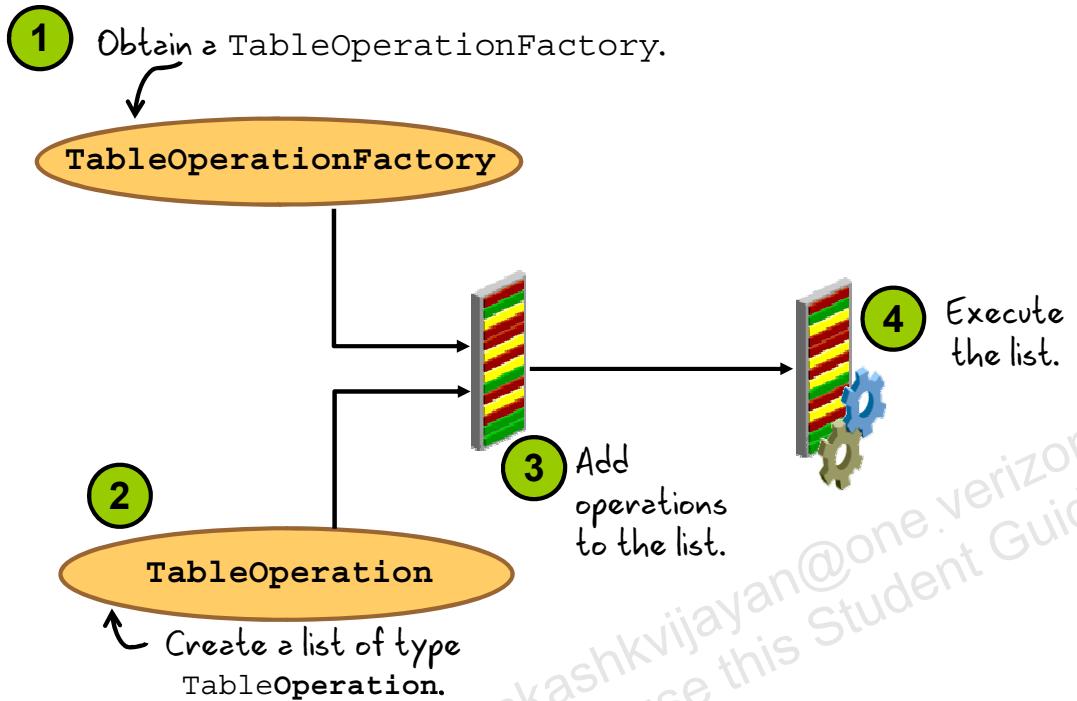


Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide lists some of the points you must keep in mind when creating a transaction.

- Transactional operations support only write operations. You can create, update, or delete data from the KVStore. You cannot read data from the store by using a transactional operation (sequence of operations).
- All the individual operations in a transactional operation must have the same shard key. Each operation should operate on a single record only.
- Do not create two operations that operate on the same record. As you saw in the previous slide, the operations are executed in an internally decided order (not the order in which you created the operations for a transaction). If there is any confusion while setting the order for the operations, you will find that when the transaction is executed, an exception is raised and the transaction aborts immediately.

# Creating and Running Transactional Operations: Process



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To create and run a transactional operation, perform the following steps:

1. Create an instance of the `TableOperationFactory` class by using the `getTableOperationFactory()` method.
2. Create a list of type `TableOperation` that holds all the individual operations of the transactional operation.
3. Add the individual operations to the list by using the `TableOperationFactory` instance.
4. Run the transaction operation by passing the list to the `execute()` method. A transactional operation is usually written within a `try-catch` structure.

These steps are only for creating and running a transactional operation. In addition to these steps, you must create a KVStore handle, a TableAPI handle, a Table handle, the row to be inserted or deleted, and so on, as required by the individual operations.

# Creating and Running Transactional Operations: Example

```
1 TableOperationFactory tof =  
mytableapi.getTableOperationFactory();  
  
2 List<TableOperation> opList = new  
ArrayList<TableOperation>();  
  
3 opList.add(tof.createPut(row1, null, true));  
opList.add(tof.createPut(row2, null, true));  
opList.add(tof.createPut(row3, null, true));  
  
4 mytableapi.execute(opList,null);
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## TableOperationFactory Methods

Methods
<code>createDelete(PrimaryKey key, ReturnRow.Choice prevReturn, boolean abortIfUnsuccessful)</code>
<code>createDeleteByVersion(PrimaryKey key, Version versionMatch, ReturnRow.Choice prevReturn, boolean abortIfUnsuccessful)</code>
<code>createPut(Row row, ReturnRow.Choice prevReturn, boolean abortIfUnsuccessful)</code>
<code>createPutIfAbsent(Row row, ReturnRow.Choice prevReturn, boolean abortIfUnsuccessful)</code>
<code>createPutIfPresent(Row row, ReturnRow.Choice prevReturn, boolean abortIfUnsuccessful)</code>
<code>createPutByVersion(Row row, Version versionMatch, ReturnRow.Choice prevReturn, boolean abortIfUnsuccessful)</code>



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

While adding the individual operations in a transactional operation to a TableOperation list, you must use the methods listed in the slide. These methods are called by using the TableOperationFactory instance. They are similar to the respective methods of the TableAPI class. However, they differ in the following:

- The WriteOptions parameter is not passed in these methods. It is passed to the execute() method.
- The ReturnRow.Choice parameter is passed instead of the ReturnRow parameter. This parameter is an ENUM constant that can have the following values:
  - **ALL**: Specifies that both the value and the version are returned
  - **NONE**: Specifies that neither the value nor the version is returned
  - **VALUE**: Specifies that only the value is returned
  - **VERSION**: Specifies that only the version is returned

**Note:** For best performance, you should retrieve only the required values.

- An additional parameter called abortIfUnsuccessful is passed.
- The return type is an instance of TableOperation.

## Executing Operation Syntax

```
List<TableOperationResult>
    execute(List<TableOperation> operations,
           WriteOptions writeOptions)
    throws TableOpExecutionException,
           DurabilityException,
           FaultException
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the syntax for the `execute` method. This method provides an efficient and transactional mechanism for executing a sequence of operations associated with tables that share the same shard key portion of their primary keys. The efficiency results from the use of a single network interaction to accomplish the entire sequence of operations. The `execute()` method takes as argument the `TableOperation` list and the `WriteOptions` parameter.

## Summary

In this lesson, you should have learned how to:

- Explain a transactional operation
- Create a transactional operation
- Execute a transactional operation



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## Practice 12 Overview: Creating Transactions

This practice covers the following topics:

- Creating a transaction
- Executing a transaction



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

In this practice, you use the APIs that you learned in this lesson to create Java programs.

# 13

## Handling Large Objects

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

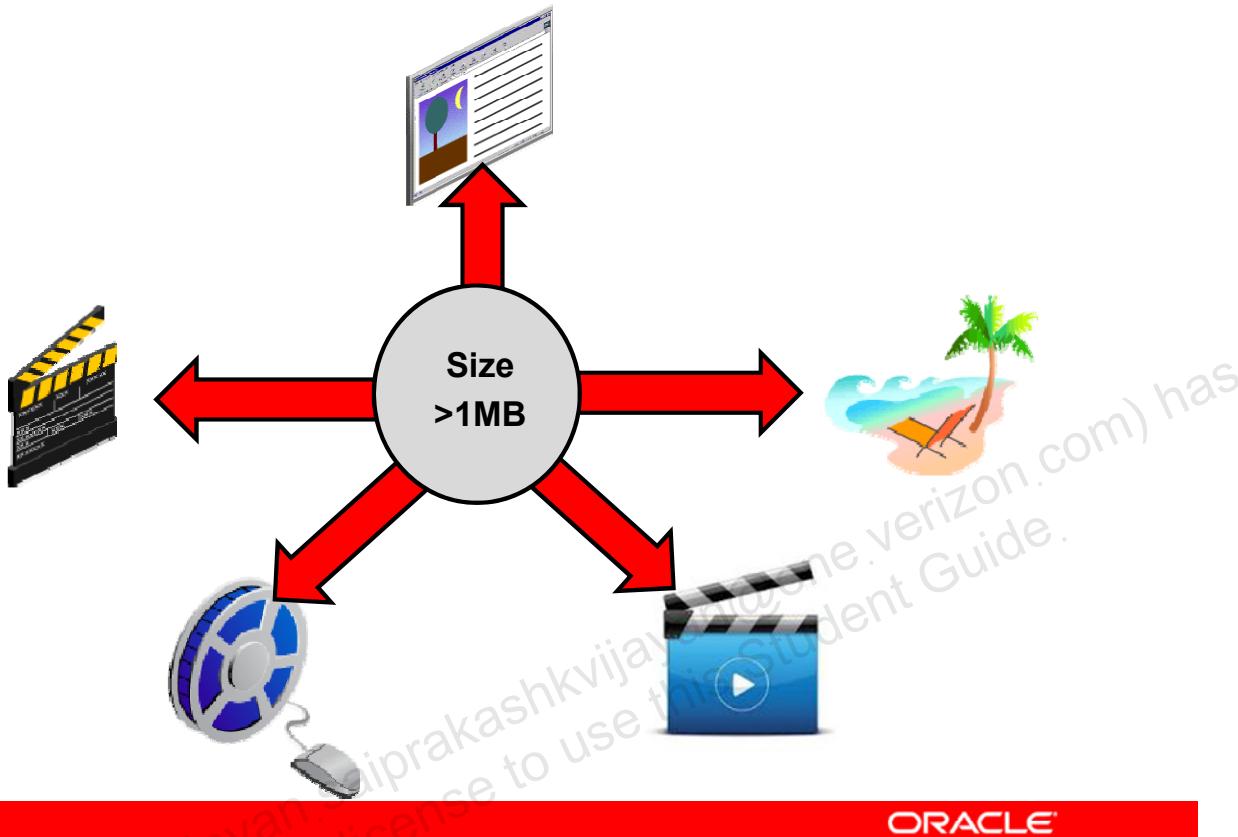
- Identify large objects
- Explain the need to store large objects
- Use Oracle NoSQL APIs to:
  - Store large objects
  - Retrieve large objects



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Introducing Large Objects



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Data objects, such as image, audio, and video files that are huge in size are termed as large objects. Typically, any data object that is more than 1 MB in size can be considered as a large object.

## How Are Large Objects Useful?

As more and more businesses are looking at providing personalization and networking features in their applications, there is a greater need today for storing large objects. Even in the field of healthcare and medicine, there is a need to store huge amounts of patient data, such as X-rays, continuous monitoring streams of data, and so on.

These business requirements justify a need for an efficient system to store and retrieve large objects.

The large objects APIs available with the latest release of Oracle NoSQL Database allow access to large objects without having to materialize the large value in the application. This improves the performance and reduces memory consumption when dealing with large values.

# Oracle NoSQL APIs for Large Objects

Writing LOB	Reading LOB	Deleting LOB
<code>KVLargeObject. putLOB()</code>	<code>KVLargeObject. getLOB()</code>	<code>KVLargeObject. deleteLOB()</code>
<code>KVLargeObject. putLOBIfAbsent()</code>		
<code>KVLargeObject. putLOBIfPresent()</code>		

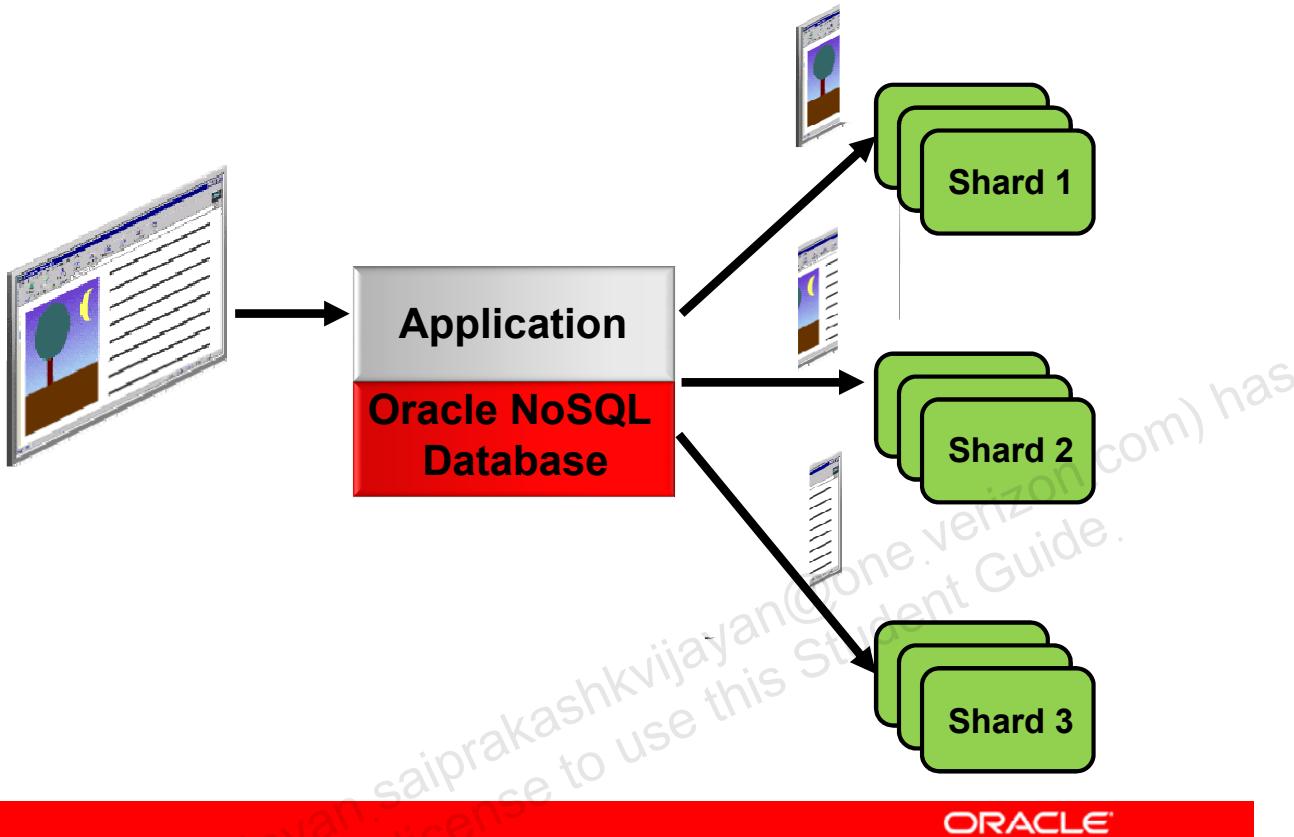


Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Oracle NoSQL Database provides APIs to enable you to work with large objects. You can group the APIs into three categories depending on the function you want to perform. Namely, writing large objects, reading large objects, and deleting large objects. The large object APIs are very similar to their corresponding APIs for regular data. To access the LOB APIs, you need to create a KVStore handle.

For writing large objects into the Oracle NoSQL Database, you have three APIs. The `putLOB` API creates a new record if the record does not already exist in the database. In case a record with the specified key already exists in the database, the `putLOB` API overwrites the existing record. The `putLOBIfAbsent` API always creates a new record in the database. If a record with the specified key already exists, then the API returns without performing any write. The `putLOBIfPresent` API always performs an update in the database. That is, if a record with the specified key exists, then the API performs a write with the new value. If a record with the specified key does not exist, then the API returns without performing any write. To read and delete large objects, you have two APIs as listed in the slide. You will learn more about these APIs in the next section of this course.

## Large Objects Storage



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

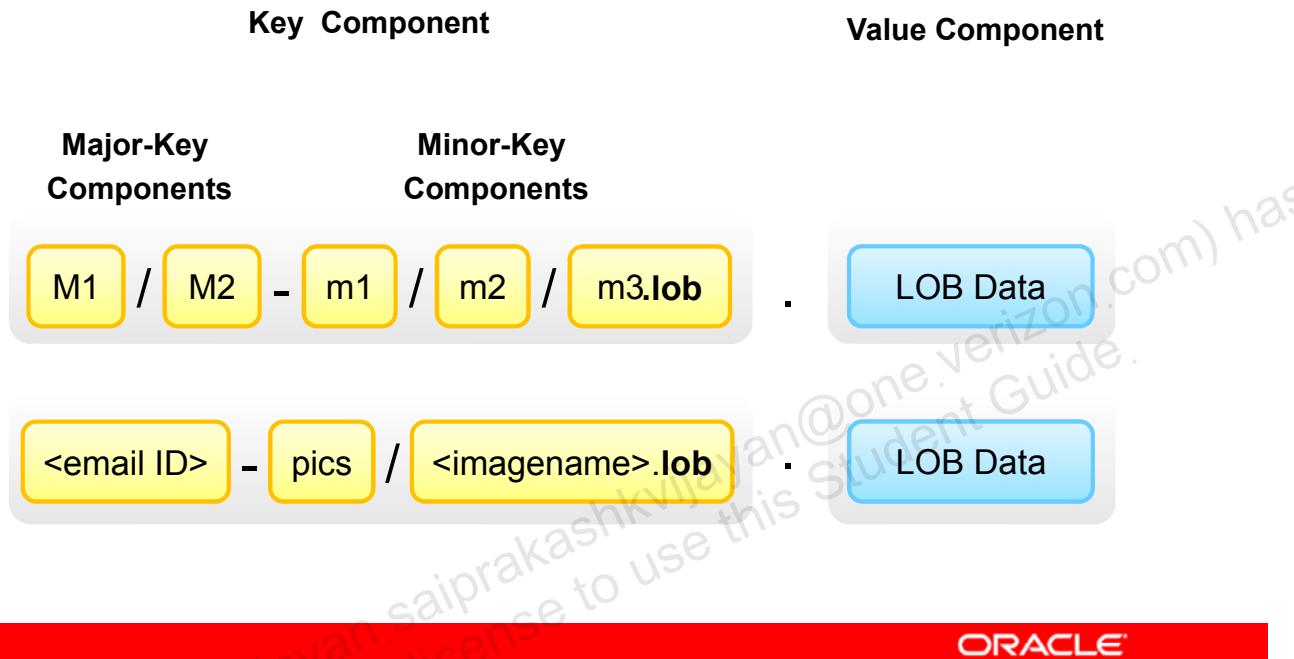
Oracle NoSQL Database splits a large object into chunks and stores it across different shards in a KVStore. The large object chunks are of different sizes, which are chosen by the NoSQL system automatically. These sizes are optimized for the underlying storage architecture and hardware.

Since a large object is split into chunks and stored across a KVStore in parallel, the read and write operations on these large objects are faster.

Unlike other key-value pairs, large objects are not stored in shards based on the major- and minor-key path components in the key path. Instead, LOB data uses a hidden keyspace, and its various chunks are distributed across partitions based on this keyspace, instead of being based on the key that you provide.

# Creating Large Object Keys

A record in a KVStore consists of a key-value pair.



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

To store large objects in an Oracle NoSQL Database you need to use `oracle.kv.Key` objects. Each such object contains a series of strings representing a key path. The LOB interface makes use of these text-only keys that can be used with either the Tables API or the Key/Value API.

In Unit II, you learned about the key structure and key components of a key-value pair in an Oracle NoSQL Database. A key-value pair consists of a key component and a value component. The key component is made up of at least one major-key component and one or more minor-key components. A key should uniquely identify a record in the KVStore. The principles to design a key remain the same for large objects as well. You can have one or more major-key components and one or more minor-key components to store the large object data in the KVStore. The only other requirement is that the last key component must contain a trailing suffix. This is how the NoSQL engine will identify that the record it is storing is a large object. The default suffix is `.lob`. This default suffix value can be changed by using the `setLOBSuffix` method of the `KVStoreConfig` class. All the large object APIs check for the presence of the large object suffix. If the check fails, an `IllegalArgumentException` is thrown.

For example, consider a scenario where you want to allow users of your application to store their pictures. To design the key, without considering any other performance or application requirements, you can create a major-key component using the application user's email ID and create two minor-key components. One can be a static string called `pics` and the other can be the file name or image name of the picture that you are storing. Note that the last minor key, `imagename` in our case, has a suffix `.lob` attached to it.

## Creating a Key for Table API Users

- Table APIs users can use LOB APIs.
- A key can be created by using:
  - The `Key.fromString()` method
  - One or more arrays
  - Information stored in table cells



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can use LOBs with Table APIs. However, you still need to create a key to identify the LOB. The LOB cannot be stored in a table row.

In a key-value pair, arrays are used to represent the major- and minor-key path components. These arrays are passed to the `Key.createKey()` method. Table API users can use the `Key.fromString()` method as it is easy to store a string representation of a key path in a table cell. Alternatively, Table API users can store key path components as one or more arrays in table cells, or construct the key path array using information found in table cells. One key point to ensure is that the LOB key paths do not collide with the keys used internally by the tables.

## Quiz

Which of the following checks are made by the LOB APIs?

- a. Size of the LOB is more than 1 MB.
- b. Key of the LOB has a trailing suffix.
- c. Key of the LOB has a minor-key component.
- d. Size of the LOB key is less than 1 MB.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: b**

## Storing Large Objects: API Overview

```
putLOB  
Version putIfPresentLOB (Key lobKey,  
                         putIfAbsentLOB InputStream lobStream,  
                         Durability durability,  
                         long lobTimeout,  
                         TimeUnit timeoutUnit)  
throws DurabilityException,  
       RequestTimeoutException,  
       ConcurrentModificationException,  
       FaultException,  
       IOException
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide displays the syntax for the APIs used to store large objects into the Oracle NoSQL Database. It displays the return type, the input parameters, as well as the exceptions thrown by the methods.

There are three APIs to store large objects into the KVStore. These three APIs differ only in their functionality of performing the write operation. The return type, input parameters, and the exceptions thrown remain the same for all the three APIs or methods.

These methods take five input parameters. You must be familiar with the `Key`, `Durability`, `timeout`, and `timeoutunit` parameters. You need to pass the key for the key-value pair that will be inserted. This key is created similar to the keys for regular key-value pairs. The only other consideration for large objects is the `.lob` suffix at the end of the last key component. The `InputStream` parameter is unique to these large object methods. The large object that you want to insert should be converted to an input stream format and then passed to these methods. The `Durability` parameter accepts the durability policy that you want to specify for this write operation. The last two parameters are used to specify the amount of time to wait before quitting the write operation.

If the write operation is successful, the record is inserted into the KVStore and a version token is returned. If the write operation does not meet the durability policy specified, a durability exception is thrown. If the write operation does not complete within the specified time, a request timeout exception is thrown.

## Storing Large Objects: Code Example

```
KVStoreConfig kconfig = new
    KVStoreConfig("kvstore", "localhost:5000");
KVStore mystore = KVStoreFactory.getStore(kconfig);
Key key1 = Key.createKey("hr_logo.jpeg", "image.lob");
File lobfile = new File("home/oracle/pictures/hr_logo.jpeg");
try{
    FileInputStream fis = new FileInputStream(lobfile);
    Version vrl = mystore.putLOB(key1, fis,
        Durability.COMMIT_WRITE_NO_SYNC,
        5, TimeUnit.SECONDS);
}
catch (FileNotFoundException fnf){}
catch (IOException io){}
finally{
    mystore.close();
}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide displays a piece of code that reads a file from the local file system and inserts it into the Oracle NoSQL Database. Let us look at this code bit by bit.

The first task is to obtain a KVStore handle. You can do this by creating an object of the KVStoreConfig class. Recollect that you need to pass the name of a running KVStore instance and the host and port numbers. You then pass this object to the getStore method of the KVStoreFactory class.

The next step is to create the key for the key-value pair to be inserted. In this example, the key is created with one major-key component and one minor-key component. The major-key component is the file name and the minor-key component is the static text image with a suffix .lob.

After creating the key, you also need to read the image to be inserted. You can do this by using the Java File class. In this example, the path to an hr\_logo.jpeg file is used to create the file.

The putLOB method takes the large object as an InputStream. So you need to pass the Java file object to the FileInputStream class and obtain the InputStream.

You then use the key you created and the InputStream object along with the durability and request timeout parameters to write the file into the KVStore using the putLOB method. You need to insert the final two steps inside the Java try-catch statements.

After completing the write operation, it is a best practice to close the KVStore.

## Retrieving Large Objects: API Overview

```
InputStreamVersion getLOB(Key lobKey,  
                           Consistency consistency,  
                           long lobTimeout,  
                           TimeUnit timeoutUnit)  
throws ConsistencyException,  
       RequestTimeoutException,  
       FaultException,  
       ConcurrentModificationException
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide displays the syntax for the `getLOB` API used to fetch large objects from the Oracle NoSQL Database. This method takes four input parameters. You need to pass the key for the key-value pair that needs to be retrieved. The `Consistency` parameter specifies the consistency policy that you want for this read operation. And the last two parameters are used to specify the amount of time to wait before quitting the read operation. If the read operation is successful, the large object data is fetched as an `InputStream` object along with the version token of the record. If the read operation does not meet the consistency policy specified, a consistency exception is thrown. If the read operation does not complete within the specified time, a request timeout exception is thrown.

## Retrieving Large Objects: Code Sample

```
KVStoreConfig kconfig = new  
KVStoreConfig("kvstore","localhost:5000");  
KVStore mystore = KVStoreFactory.getStore(kconfig);  
Key key = Key.createKey("hr_logo.jpeg","image.lob");  
InputStream stream;  
try{  
    InputStreamVersion isv = mystore.getLOB(key,  
        Consistency.NONE_REQUIRED, 5,  
        TimeUnit.SECONDS);  
    stream = isv.getInputStream();  
    //process the stream as per application requirement}  
catch (Exception io)  
{  
}  
finally{  
    mystore.close();}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows a code sample to understand the `getLOB` method. As you saw in the previous code sample, you first need to obtain a `KVStore` handle and also create the key for the value to be retrieved. After completing these tasks, you can use the `getLOB` method to fetch the large object from the `KVStore`. The `getLOB` method returns the large object in `InputStream` format. So you need to declare an object of class `InputStream`. The `getLOB` method should be written inside the Java try-catch statements. To this `getLOB` method, you must pass the key you created, a consistency policy for the read operation and the request timeout parameters. You can then retrieve the large object from the `InputStreamVersion` object by using the `getInputStream` method. You can use this stream object to process the large object you retrieved from the `KVStore` as per your application requirements.

## Deleting Large Objects: API Overview

```
boolean deleteLOB(Key lobKey,  
                  Durability durability,  
                  long lobTimeout,  
                  TimeUnit timeoutUnit)  
throws DurabilityException,  
      RequestTimeoutException,  
      FaultException,  
      ConcurrentModificationException
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the syntax for the `deleteLOB` API used to delete large objects from the Oracle NoSQL Database. This method takes four input parameters. You need to pass the key for the key-value pair that needs to be deleted. The `Durability` parameter specifies the durability policy that you want for this delete operation. The last two parameters are used to specify the amount of time to wait before quitting the delete operation. If the delete operation is successful, the method returns a “true” Boolean value. Otherwise, “false” is returned. If the delete operation does not meet the durability policy specified, a durability exception is thrown. If the delete operation does not complete within the specified time, a request timeout exception is thrown.

## Deleting Large Objects: Code Example

```
KVStoreConfig kconfig = new  
KVStoreConfig("kvstore","localhost:5000");  
KVStore mystore = KVStoreFactorygetStore(kconfig);  
  
Key key = Key.createKey("hr_logo.jpeg","image.lob");  
  
mystore.deleteLOB(key, Durability.COMMIT_WRITE_NO_SYNC,  
                  5, TimeUnit.SECONDS);  
mystore.close();
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The code sample in the slide shows how to use the deleteLOB method. You first create a KVStore handle, then create the key for the value you want to delete, and pass this key to the deleteLOB method.

## Summary

In this lesson, you should have learned how to:

- Identify large objects
- Explain the need to store large objects
- Use Oracle NoSQL APIs to:
  - Store large objects
  - Retrieve large objects



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Practice 13 Overview: Handling Large Objects

This practice covers writing and reading large objects.



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This is a hands-on practice.

Unauthorized reproduction or distribution prohibited. Copyright© 2016, Oracle and/or its affiliates.

Sai Vijayan S (sai.vijayan.saiprakashkvijayan@one.verizon.com) has  
a non-transferable license to use this Student Guide.

# 14

## Accessing a Secure Store

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Access a secure store
- Identify security constants



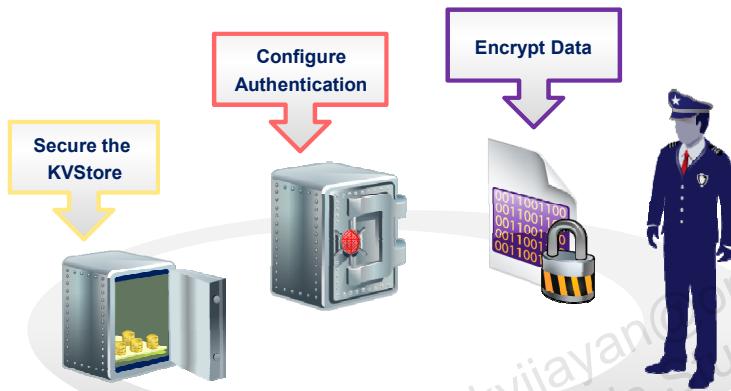
ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Secure KVStore

A KVStore can be made secure by:

- Performing a secure installation
- Configuring security to an existing installation



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

A KVStore can be made secure by configuring the security features available for Oracle NoSQL Database. The KVStore administrator can secure a KVStore during installation itself or configure security to an existing installation. Secure installation is covered in detail in the *Oracle NoSQL Database for Administrators* course.

In this course, you will learn how to access a secure store from within a Java application.

# Security Features

Two levels of security:

- Network-level security
  - Secure installation
  - External password storage
- Application-level security
  - Authentication
  - Authorization or Encryption
  - Security Policies



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can configure two levels of security in ONDB.

- **Network-level security:** This is configured at the file-system level as part of an ONDB installation and configuration process. You use a security configuration utility to add security to a new or existing ONDB installation.
- **Application-level security:** You can use the default security features to configure application-level security.

Network-level security is covered in detail in the *Oracle NoSQL Database for Administrators* course. Application-level security is covered later in this course. For further details and guidelines on security, refer to the *Oracle NoSQL Database Security Guide*.

## Obtaining Handle to Secure Store

```
KVStoreConfig(String storeName,  
String helperHostPort)
```

```
KVStore KVStoreFactorygetStore(KVStoreConfig config)
```

```
KVStore KVStoreFactorygetStore(KVStoreConfig config,  
LoginCredentials creds,  
ReauthenticateHandler reauthHandler)
```

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Recollect that you used the `getStore()` method of the `KVStoreFactory` class to obtain a handle to the `KVStore`. A `KVStoreConfig` object containing the connection details is passed as a parameter to this `getStore()` method.

To connect to a secure store, you use the same `getStore()` method and along with the `KVStoreConfig` object you need to pass additional authentication arguments required for accessing a secure `KVStore`.

- `LoginCredentials` is a common interface of `KVStore` credential class implementations. This object is used in an application to authenticate a user accessing a `KVStore`.
- `ReauthenticateHandler` is a callback interface used when `KVStore` authentication has expired and requires renewal.

## Security Parameters

**LoginCredentials**

```
 PasswordCredentials(String username, char[] password)  
  
 PasswordCredentials.clear();
```

```
 ReauthenticateHandler.reauthenticate(KVStore kvstore)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You pass the following security parameters to the `getStore()` method.

A `PasswordCredentials` class implements the `LoginCredentials` interface. The constructor for the `PasswordCredentials` class takes as input the username and password required to authenticate a user. When no longer required, the `clear()` method should be invoked to wipe out the password value from the Java memory heap.

When you pass a `LoginCredentials` object, you also need to pass in the `ReauthenticateHandler` interface parameter. The interface has a `reauthenticate` method that takes a `KVStore` handle as input. The `ReauthenticateHandler` will be used to re-establish a login connection without interrupting the sequence of `KVStore` calls. If no `LoginCredentials` are provided to the `getStore()` method (discussed in the previous slide), the method will attempt to locate credentials through other sources, in the following search order.

- `KVStoreConfig.getLoginProperties()`
- A login file referenced by the `oracle.kv.login` Java system property

If both security parameters are null, but login information is located using either of the preceding lookup methods, an internally supplied reauthentication handler is automatically provided that will re-read login credentials as needed for reauthentication. User passwords are not retained in memory by the KVStore client, so if you explicitly provide `LoginCredentials`, you are also responsible for supplying a reauthentication handler, if desired.

## Accessing a KVStore: Example

```
KVStoreConfig kconfig = new KVStoreConfig  
    ("orcl","localhost:5000");  
  
PasswordCredentials pc =  
new PasswordCredentials("user01",  
    "user01".toCharArray());  
  
KVStore mySecureStore =  
    KVStoreFactory.getStore(kconfig,  
    pc,  
    null)  
pc.clear();
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of obtaining a handle to a secure KVStore. As with an unsecured store, you need to create a KVStoreConfig object. Also, create an object of the PasswordCredentials class. If you pass null for the ReauthenticateHandler parameter, the system will automatically generate one.

## Specifying Security Properties

```
KVStoreConfig setSecurityProperties  
          (Properties securityProps)  
  
Properties prop = new Properties();  
  
prop.setProperty(String key, String value);
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

You can configure the security parameters for a KVStore by using the `setSecurityProperties()` method of the `KVStoreConfig` class. You configure the properties using the `setProperty()` method of the `Properties` Java class.

## Security Constants

```
public interface KVSecurityConstants
```

Fields	
AUTH_PWDFILE_PROPERTY	AUTH_USERNAME_PROPERTY
AUTH_WALLET_PROPERTY	SECURITY_FILE_PROPERTY
SSL_CIPHER_SUITES_PROPERTY	TRANSPORT_PROPERTY
SSL_TRUSTSTORE_FILE_PROPERTY	SSL_PROTOCOLS_PROPERTY
SSL_TRUSTSTORE_TYPE_PROPERTY	SSL_TRANSPORT_NAME
SSL_HOSTNAME_VERIFIER_PROPERTY	



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the KVSecurityConstants interface constants used for security configuration. These are most commonly used when populating a set of properties to be passed to the KVStoreConfig.setSecurityProperties (java.util.Properties) method. They may be used as a reference when configuring a security property file.

## Summary

In this lesson, you should have learned how to:

- Access a secure store
- Identify security constants

## Quiz

Which of the following classes should be used to pass the authentication credentials?

- a. KVStore()
- b. KVStoreFactory()
- c. KVStoreConfig()
- d. KVSecurityConstants()

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

**Answer: c**

# 15

## Handling Exceptions

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Define exceptions
- Identify the exceptions that are thrown when using Oracle NoSQL Database APIs
- Describe how to handle exceptions effectively



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Understanding Exceptions

## Exceptions:

- Occur when flow of a program is disrupted during execution
- Are handled within a `try-catch` block

```
try{
    //code...
}
catch{
}
finally{
}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

An exception is an event that disrupts the normal flow of instructions during the execution of a program. When you handle an exception, you specify the details of what to do when the exception occurs.

You handle exceptions within the Java `try/catch` statements. You write the code to be executed in the `try` block, and you write the code to handle the exception in the `catch` block. If an exception is thrown, the `catch` block handles the exception.

There are two benefits of using a `try/catch` block:

- You can fix the error.
- The program is prevented from terminating abruptly.

# Oracle NoSQL Database Exceptions

Exception	Indication
ContingencyException	Result occurred that cannot be expressed by the return value
NoSQLRuntimeException	Generic exception class that extends Java RuntimeException
FaultException	Error occurred that cannot be handled by an application
RequestLimitException	Many active requests waiting to be processed
RequestTimeoutException	Request cannot be processed within the specified time interval



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide lists some of the exceptions you might encounter while using Oracle NoSQL Database for an application. The exceptions are described below.

- **ContingencyException:** This exception indicates that the status or result of an operation cannot be expressed through the return value. This exception should be handled by the caller of the method.
- **NoSQLRuntimeException:** This is a generic exception used for generating runtime exceptions whose messages are derived from a locale-specific message file.
- **FaultException:** This exception indicates an error condition that cannot normally be handled by the caller of the method, except by retrying the operation.
- **RequestLimitException:** This exception is thrown when a request cannot be processed because it would exceed the maximum number of active requests for a node.
- **RequestTimeoutException:** This exception is thrown when a request cannot be processed because the configured timeout interval is exceeded.

# Oracle NoSQL Database Exceptions

Topic (Unit III)	Exception
Consistency	ConsistencyException
Durability	DurabilityException
Transactions	TableOpExecutionException
Large Objects	PartialLOBException
Security	KVSecurityException AuthenticationRequiredException AuthenticationFailureException UnauthorizedException



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

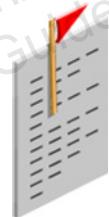
In this slide, some of the more API-specific exceptions are listed. You learned these APIs in the lesson for Unit III. A description of these exceptions is given below.

- **ConsistencyException:** This exception indicates that a read operation would not be completed as a specified consistency policy was not met.
- **DurabilityException:** This exception indicates that a write operation would not be completed as a specified durability policy was not met.
- **TableOpExecutionException:** This exception indicates that a sequence of operations or transaction could not be executed successfully.
- **PartialLOBException:** This exception indicates that a read operation was performed on a partial large object.
- **KVSecurityException:** This is the base class for all security-related exceptions. These exceptions are thrown only when the KVStore you are accessing is a secure store.
- **AuthenticationRequiredException:** This exception is thrown when a client is not authenticated while performing a secured operation.
- **AuthenticationFailureException:** This exception is thrown when an application passes invalid credentials to a KVStore authentication operation.
- **UnauthorizedException:** This exception is thrown when an authenticated user is attempting to perform an operation for which he or she is not authorized.

## ContingencyException

This exception occurs when the result of an operation cannot be expressed in the return type.

Handle this exception from within the calling method.



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This exception occurs when there is an error within the caller method, causing an unexpected return value. It occurs rarely.

## FaultException

This exception occurs when an error cannot be handled by the method.

Handle this exception by:

- Retrying the request
- Reporting an error

C

R

U

D

put ()

get ()

multiGet ()

tableIterator ()

putIfAbsent () delete ()

putIfPresent () deleteIfVersion ()

putIfVersion () multiDelete ()

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This exception is thrown by all the methods that are used to perform CRUD operations.

The exception is thrown if the error occurred remotely due to a server exception and it is not logged in the client library.

This exception cannot be normally handled because it is not an error that occurred due to your application. The error could have occurred remotely due to an internal server exception. In such cases, the exception returns the following methods that will give some information about the error, and also helps the user to log, test, and debug the error.

- **getFaultClassName ()**: This method returns the name of the remote exception's class name.
- **getRemoteStackTrace()**: This method returns the stack trace of the remote exception or null if the error occurred locally.
- **toString()**: This method returns the fault class name and remote stack trace.
- **wasLoggedRemotely()**: To keep track of the errors and make the information available for both the client and the administrator, the client also logs the errors that occur locally. This method returns if the client application has logged it.

## RequestLimitException

This exception occurs when:

- A node is already servicing the maximum number of active requests permitted
- There is a network issue while communicating with a node
- The node itself is facing an issue

Handle this exception by:

- Waiting and retrying
- Relaxing the request limit
- Freeing the thread



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This exception is thrown when a request cannot be processed because it would exceed the maximum number of active requests for a node. This limit is configured using the `setRequestLimit()` method of the `KVStoreConfig` class. You learn to set this limit in the next slide.

This exception might also indicate that there is a network issue with the communication path to the node or that the node itself has problems. This exception is quite rare because the KVStore request dispatcher handles node failures automatically.

You can handle this exception by:

- Waiting for a small unit of time and then retrying the operation
- Relaxing the request limit. That is, you increase the request limit.
- Stopping the request and freeing up the thread that contains the failure. Freeing up the thread enables other nodes to process the pending requests.

## RequestLimitConfig

```
RequestLimitConfig(int maxActiveRequests,  
                    int requestThresholdPercent,  
                    int nodeLimitPercent)
```

```
KVStoreConfig.setRequestLimit(RequestLimitConfig rlc)
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The slide shows the constructor you can use to configure the request limit for a node. The threshold and limit parameters are expressed as percentages of `maxActiveRequests`.

There are three parameters:

- **maxActiveRequests**: Specifies the maximum number of active requests permitted by the KV client
- **requestThresholdPercent**: Specifies the percentage of `maxActiveRequests` at which requests are limited. The default is 90.
- **nodeLimitPercent**: Specifies the maximum number of active requests that can be associated with a node when the request limiting mechanism is active. The default is 80.

This mechanism is activated only when the number of active requests exceeds the specified `requestThresholdPercent` parameter. When the mechanism is active, the number of active requests to a node is not allowed to exceed `nodeRequestLimitPercent`. Any new requests that exceed this limit are rejected and a `NodeRequestLimitException` is thrown. This exception occurs rarely.

## Handling RequestTimeoutException

This exception occurs when:

- A request is not completed within the given time interval (default – 5 seconds)
- A store attempts to service too many requests at the same time
- There is a network problem

Handle this exception by:

- Retrying the request
- Increasing the timeout interval
- Reporting an error



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This exception is thrown when the operation cannot be completed within the specified amount of time set in the timeout property. This indicates that there are too many requests at the same time or there is a network problem that is causing the network to slow down or be unresponsive.

The default timeout interval is five seconds. You can change this value by using the `setRequestTimeout()` method of the `KVStoreConfig` class.

You handle this exception by:

- Waiting for a short period and then retrying the request
- Increasing the timeout interval
- Reporting an error

## Methods That Throw RequestTimeoutException

C

R

U

D

put()	get()	putIfAbsent()	delete()
	multiGet()	putIfPresent()	deleteIfVersion()
	tableIterator()	putIfVersion()	multiDelete()

The methods include two parameters:

- long timeout
- TimeUnit timeoutunit

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The methods you use to perform the CRUD operations throw RequestTimeoutException. Along with other parameters, these methods include the following optional parameters:

- **timeout**: Specifies the time required for completing the operation. If the parameter is set to zero or not specified, the default request timeout is used.
- **timeoutUnit**: Represents the unit of the `timeout` parameter. The unit can be nanoseconds, microseconds, milliseconds, seconds, minutes, hours, or days.

If an operation exceeds the given timeout, the `RequestTimeoutException` is thrown.

## Quiz

In the following example, identify the exception that is thrown if the operation is not completed within 10 seconds.

```
get(myKey, myValue, null, null, 10,  
    TimeUnit.SECONDS);
```

- a. RequestLimitException
- b. FaultException
- c. RequestTimeoutException
- d. ContingencyException



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

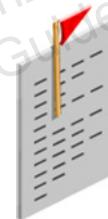
**Answer: c**

**Reason:** The `timeout` parameter is set to 10, which means that the application waits for a maximum of 10 seconds and then throws a `RequestTimeoutException`.

## Handling ConsistencyException

A consistency exception is thrown when a read operation cannot be completed in the specified consistency policy.

- This exception can be handled by:
  - Retrying the read operation
  - Setting a larger timeout value
  - Logging an error



ORACLE®

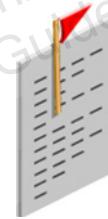
Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This exception is thrown when the specified consistency policy is not met. The likelihood of this exception being thrown depends on the specified consistency and the general health of the KVStore system. If the exception is thrown frequently, the specified consistency policy must be relaxed to keep the load balanced between the system and hardware resources.

## Handling DurabilityException

This exception is thrown when a write operation cannot be completed in the shard.

- You handle this exception by:
  - Resending the write operation
  - Retrying the operation a little later
  - Logging an error



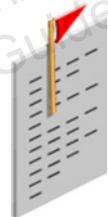
**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This exception is thrown when write operations cannot be initiated because a quorum of Replicas as determined by the Durability.ReplicaAckPolicy was not available.

## Handling TableOpExecutionException

This exception provides information about a failure from the sequence of operations executed by TableAPI.execute().



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This exception is thrown during the execution of a transactional operation.

TableOpExecutionException is raised if an error prevents the sequence from executing successfully.

- You can use the `getFailedOperation()` method to determine which operation failed.
- You can use the `getFailedOperationIndex()` method to determine the list index of the operation that failed.
- You can also use the `getFailedOperationResult()` method to obtain an `OperationResult` object, which you can further use to troubleshoot the cause of the execution or exception.

## Handling PartialLOBException

- This exception is thrown when KVStore.getLOB() is invoked on a partial LOB.
- A partial LOB is typically the result of an incomplete deleteLOB() or putLOB() operation.



ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

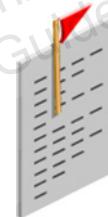
When you modify a LOB and there is some failure during the LOB modification operation, it results in the creation of a partial LOB. The LOB value of a partial LOB is in some intermediate state, where it cannot be read by the application. Any attempts to perform a getLOB() on a partial LOB will result in a PartialLOBException.

A partial LOB resulting from an incomplete putLOB() operation can be repaired by retrying the operation. Or the LOB can be deleted and a new key-value LOB pair can be created in its place. A partial LOB resulting from an incomplete delete operation must have the delete retried.

# Handling Security Exceptions

The following security exceptions are thrown while accessing a secured KVStore:

- `AuthenticationRequiredException`
- `AuthenticationFailureException`
- `UnauthorizedException`



**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

- **AuthenticationRequiredException:** This exception is thrown when a secured operation is attempted and the client is not currently authenticated. This can occur when the client does not supply login credentials either directly or by specifying a login file. It can also occur if login credentials were specified but the login session has expired. The client application should reauthenticate before retrying the operation.
- **AuthenticationFailureException:** This exception is thrown if an application passes invalid credentials to a KVStore authentication operation.
- **UnauthorizedException:** This exception is thrown from methods where an authenticated user is attempting to perform an operation for which he or she is not authorized. An application that receives this exception typically should not retry the operation.

## Summary

In this lesson, you should have learned how to:

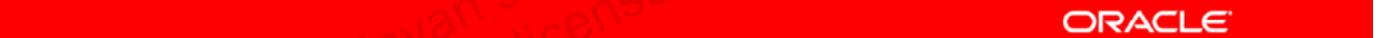
- Define exceptions
- Identify the exceptions that are thrown when using Oracle NoSQL Database APIs
- Describe how to handle exceptions effectively



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

## **Unit III**

# **Summary of Implementing Application-Specific Requirements**

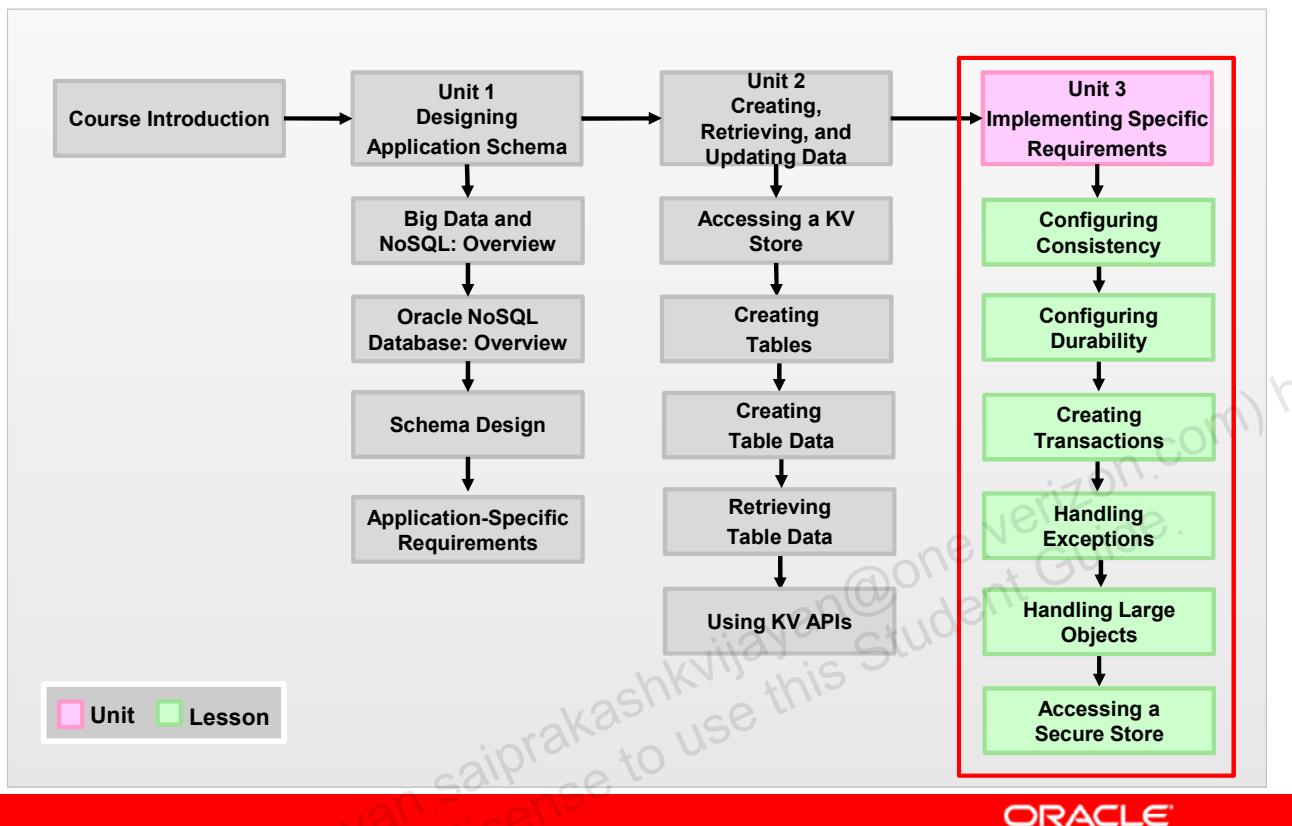


**ORACLE®**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

# Course Outline

Unauthorized reproduction or distribution prohibited. Copyright© 2016, Oracle and/or its affiliates.



ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.