ORACLE UNIVERSITY




# Oracle NoSQL Database for Developers

Activity Guide

D76021GC10

Edition 1.0 | November 2015 | D77501

Learn more from Oracle University at oracle.com/education/

ORACLE

## 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

**This book was published using:** oracle*tutor*

# Table of Contents

# Course Practice Environment: Security Credentials

**Chapter I**

## Course Practice Environment: Security Credentials

For OS usernames and passwords, see the following:

- If you are attending a classroom-based or live virtual class, ask your instructor or LVC producer for OS credential information.
- If you are using a self-study format, refer to the communication that you received from Oracle University for this course.

# Practices for Lesson 1: Course Overview

**Chapter 1**

# Practices for Lesson 1

## Practices Overview

- There are no practices for Lessons 1, 14, and 15.
- Practices 2- 5 contain paper-based questions that check your understanding of the concepts taught in the lessons.
- Practices 6-13 contain tasks that you perform in the lab environment. The lab environment details will be explained by your instructor. The files required to complete these practices are located in the `/home/oracle/labs/dev` folder.

# Practices for Lesson 2: Big Data and NoSQL - Overview

**Chapter 2**

# Practice 2-1: Understanding Big Data and NoSQL

## Overview

In this practice, you answer questions to check your understanding of the concepts taught in the lesson.

## Assumptions

None

## Questions

a. The four Vs that describe big data are _____, _____, _____, and _____.

b. True/false?

For big data solutions, it is recommended to evaluate the project needs and select a database technology that best suits the requirements, instead of assuming that NoSQL technologies are always a better fit than relational software.

c. Consider the following situations and identify if a NoSQL database will be an appropriate storage solution.

| Situation | NoSQL Database is Appropriate |
|---|---|
| Data to be stored is consistent, non-varying, and does not have a high processing speed requirements. | |
| Large amounts of unstructured data need to be written to a storage device in batches and made highly available so that the data can be read many times and processed as per requirements. | |
| Large volumes of unstructured and semi-structured data need to be stored such that the data is available for read operations and is also updatable. Operations should be processed with low latency. | |
| Data to be stored is structured and highly confidential. | |

d. Review the following notes written by Doris and check if her understanding is correct.

| Notes | True? |
|---|---|
| A key difference between NoSQL and RDBMS systems is that NoSQL stores unstructured data and RDBMS stores structured data. | |
| A NoSQL database typically does not support traditional SQL database queries. | |
| Big data is huge amounts of data with high business value. | |

| | |
|---|---|
| Big data is mostly unstructured and is received with high velocity and is rapidly changing. | |
| Big data is not supported by traditional RDBMS systems. | |
| NoSQL was designed with security in mind. Application developers or security teams don't need to worry about implementing a security layer. | |
| NoSQL databases provide faster access to data than relational database management systems. | |
| Every enterprise is moving all their storage requirements to NoSQL databases. | |

e. HDFS is a _____ file system that offers _____ storage. You can write _____ and read _____ using HDFS.

f. True/false? Data structures for HDFS and NoSQL databases are the same.

g. Which of the following options lists the primary categories of NoSQL technologies?
   1) CouchDB, MongoDB, Cassandra, and HBase
   2) Document databases, graph databases, key-value databases and wide column stores
   3) Those that manage data in the cloud and those that don't
   4) Oracle NoSQL database, NoSQL for Windows Azure, and IBM DB2

h. Place the following features in the correct column in the table.
   1) Stores high-value data.
   2) Writes data to storage only once.
   3) Schema is flexible.
   4) Is a file system.
   5) Data is highly structured.
   6) Stores low-value data.
   7) Data can be structured or unstructured.
   8) Schema is self-describing.
   9) Stores data in bulks.

| NoSQL Database | RDBMS | HDFS |
|---|---|---|
| | | |

# Solution 2-1: Understanding Big Data and NoSQL

## Overview

The answers to Practice 2-1 are formatted in **green bold**.

## Answers

a. The four Vs that describe big data are **Volume**, **Variety**, **Velocity**, and **Value**.

b. For big data solutions, it is recommended to evaluate the project needs and select a database technology that best suits the requirements, instead of assuming that NoSQL technologies are always a better fit than relational software. **True**

c. Consider the following situations and identify if a NoSQL database will be an appropriate storage solution.

| Situation | NoSQL Database is Appropriate |
|---|---|
| Data to be stored is consistent, non-varying, and does not have a high processing speed requirements. | **No** |
| Large amounts of unstructured data needs to be written to a storage device in batches and made highly available so that the data can be read many times and processed as per requirements. | **No** |
| Large volumes of unstructured and semi-structured data need to be stored such that the data is available for read operation and also updatable. Operations should be processed with low latency. | **Yes** |
| Data to be stored is structured and highly confidential. | **No** |

d. Review the following notes written by Doris and check if her understanding is correct.

| Notes | True? |
|---|---|
| A key difference between NoSQL and RDBMS systems is that NoSQL stores unstructured data and RDBMS stores structured data. | **yes** |
| A NoSQL database typically does not support traditional SQL database queries. | **No** |
| Big data is huge amounts of data with high business value. | **No** |
| Big data is mostly unstructured and is received with high velocity and is rapidly changing. | **yes** |
| Big data is not supported by traditional RDBMS systems. | **No** |
| NoSQL was designed with security in mind. Application developers or security teams don't need to worry about implementing a security layer. | **No** |
| NoSQL databases provide faster access to data than relational database management systems. | **Yes** |
| Every enterprise is moving all their storage requirements to NoSQL databases. | **No** |

e. HDFS is a __distributed__ file system that offers _bulk_ storage. You can write _once_ and read _many times__ using HDFS.

f. Data structures for HDFS and NoSQL databases are the same. **False.**

g. Which of the following options lists the primary categories of NoSQL technologies?
   1) CouchDB, MongoDB, Cassandra and HBase
   2) **Document databases, graph databases, key-value databases and wide column stores**
   3) Those that manage data in the cloud and those that don't
   4) Oracle NoSQL database, NoSQL for Windows Azure, and IBM DB2

h. Place the following features in the correct column in the table.
   1) Stores high-value data.
   2) Writes data to storage only once.
   3) Schema is flexible.
   4) Is a file system.
   5) Data is highly structured.
   6) Stores low-value data.
   7) Data can be structured or unstructured.
   8) Schema is self-describing.
   9) Stores data in bulks.

| NoSQL Database | RDBMS | HDFS |
|---|---|---|
| Schema is flexible. | Stores high-value data. | Writes data to storage only once. |
| Stores low-value data. | Data is highly structured. | Is a file system. |
| Data can be structured or unstructured. | Schema is self-describing. | Stores data in bulks. |

# Practices for Lesson 3: Oracle NoSQL Database - Overview

**Chapter 3**

## Practice 3-1: Using KVLite

### Overview

In this practice, you start and stop KVLite.

### Assumptions

You have been assigned an environment by your instructor and are able to access it.

### Tasks

a. Log in to the `host4` machine.

b. Run the `4x4.sh` script from the `/home/oracle/labs/dev/scripts` folder to install the Oracle NoSQL Database cluster required to perform all the practices in this course.

**Note:** During the execution of the script, you will be prompted to enter the oracle password as well as the OS password for the `root` user of the respective host. Note the prompt and enter the password accordingly.

c. Open a terminal and start KVLite with port `9000` and admin port `9001`. Use the default values for the rest of the parameters. What message is displayed in the terminal console?

d. Open another terminal and verify that KVLite is running successfully.

e. Stop KVLite.

f. Start KVLite again without specifying any parameters. What message is displayed in the terminal console? After viewing the output, stop KVLite.

# Practice 3-2: Understanding Oracle NoSQL Database

## Overview

In this practice, you answer questions to check your understanding of the concepts taught in the lesson.

## Assumptions

None

## Questions

a) Review the following notes written by Doris and check if her understanding is correct.

| Notes | True? |
|---|---|
| Java proficiency is required to be able to develop applications using Oracle NoSQL Database. | |
| Oracle NoSQL Database guarantees high data availability by preventing any single point of failure. | |
| ONDB is only suited for applications with high read performance requirements and not recommended for applications requiring high write performance. | |
| Oracle NoSQL Database does not allow any schema definition for big data. | |
| Oracle NoSQL Database can be installed to provide read and write performances according to the application requirements. | |
| Oracle NoSQL Database installation is very complicated and requires expert administrators for setting up a testing environment. | |

b) Match the given definitions to the correct terms.

| Definition | Terms |
|---|---|
| A collection of distributed systems communicating with each other and providing voluminous data storage | Storage Nodes |
| A physical machine with local storage space | Replication Nodes |
| A location where ONDB data is stored | Shards |
| A group of data nodes that contain a set of similar data in the KVStore | Secondary Zone |
| A collection of data nodes physically separate from the rest of the data nodes and allowing read as well as write operations | KVStore |
| A collection of nodes serving only read requests | Primary Zone |

## Solution 3-1: Using KVLite

### Overview

In this solution, the steps to start and stop KVLite are given.

### Steps

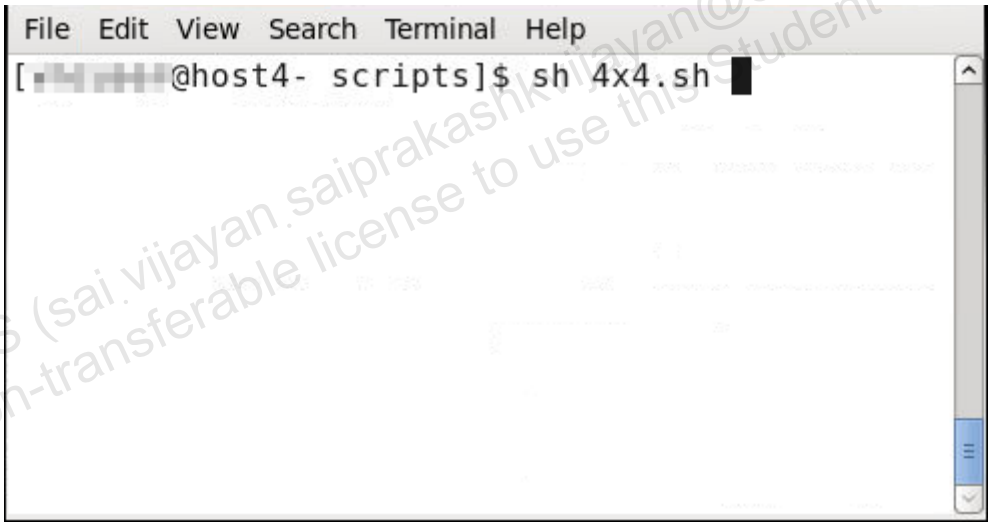
a. Log in to the `host4` machine as **oracle** user.

**Note:** All steps are to be executed as oracle user, unless specified otherwise.

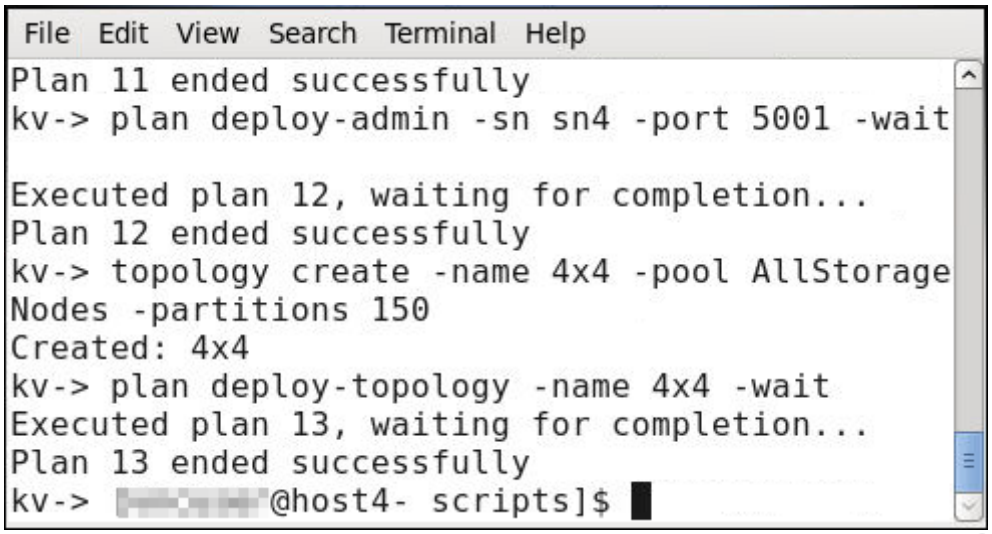
b. Run the `4x4.sh` script from the `/home/oracle/labs/dev/scripts` folder to install the Oracle NoSQL Database cluster required to perform all the practices in this course.

**Note:** During the execution of the script, you will be prompted to enter the oracle password as well as the OS password for the `root` user of the respective host. Note the prompt and enter the password accordingly.

1. Ensure you are in a terminal window and the directory is set to `/home/oracle/labs/dev/scripts`.
2. Run the `4x4.sh` file.



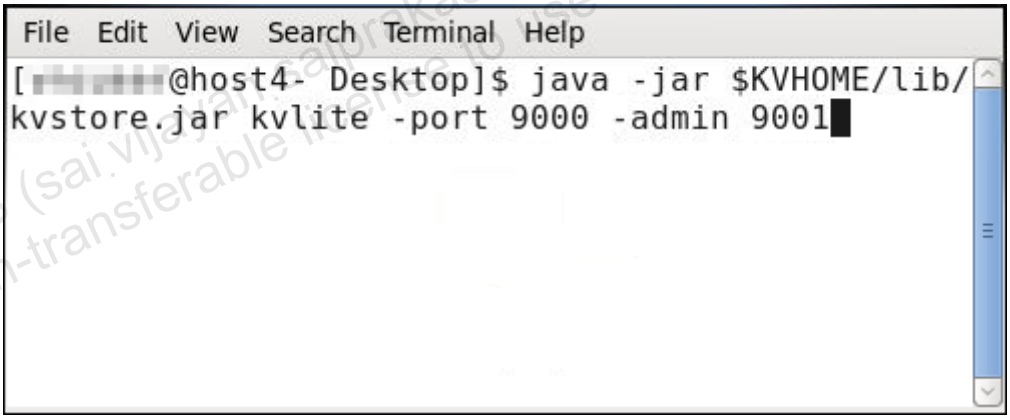
3. Enter the password details as prompted.



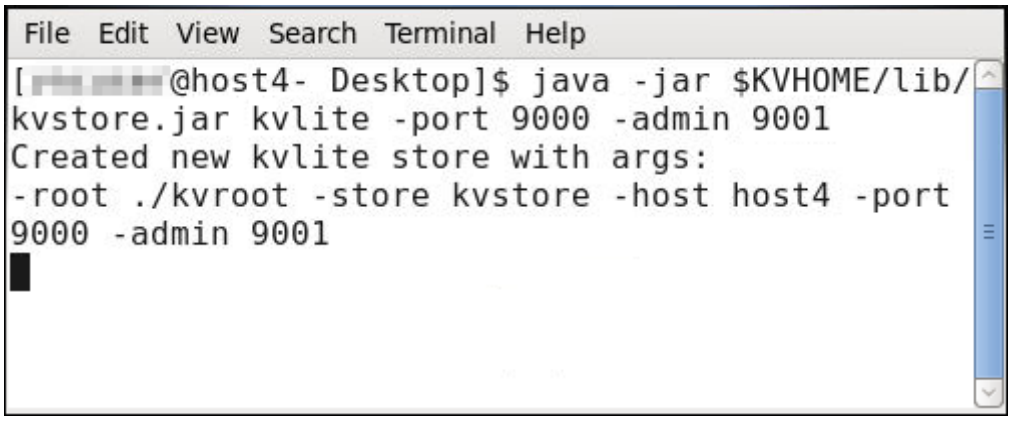

c. Open a terminal and start KVLite with port `9000` and admin port `9001`. Use the default values for the rest of the parameters. What message is displayed in the terminal console?

1. Open a terminal window and run the following command:

```
java -jar $KVHOME/lib/kvstore.jar kvlite -port 9000 -admin 9001
```



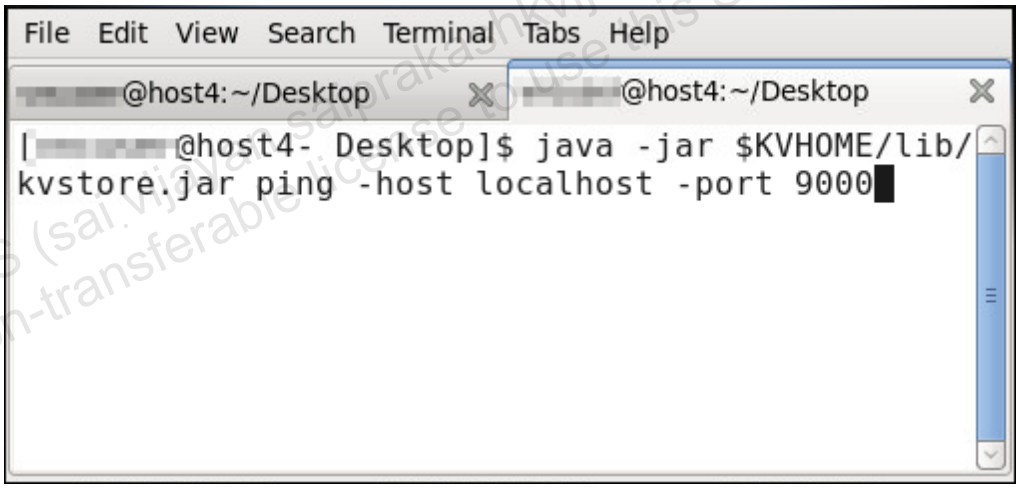
2. Note the message that is displayed. Because this is the first time you are starting KVLite, a new store is created with the parameter values you specified and default values for the rest of the parameters.



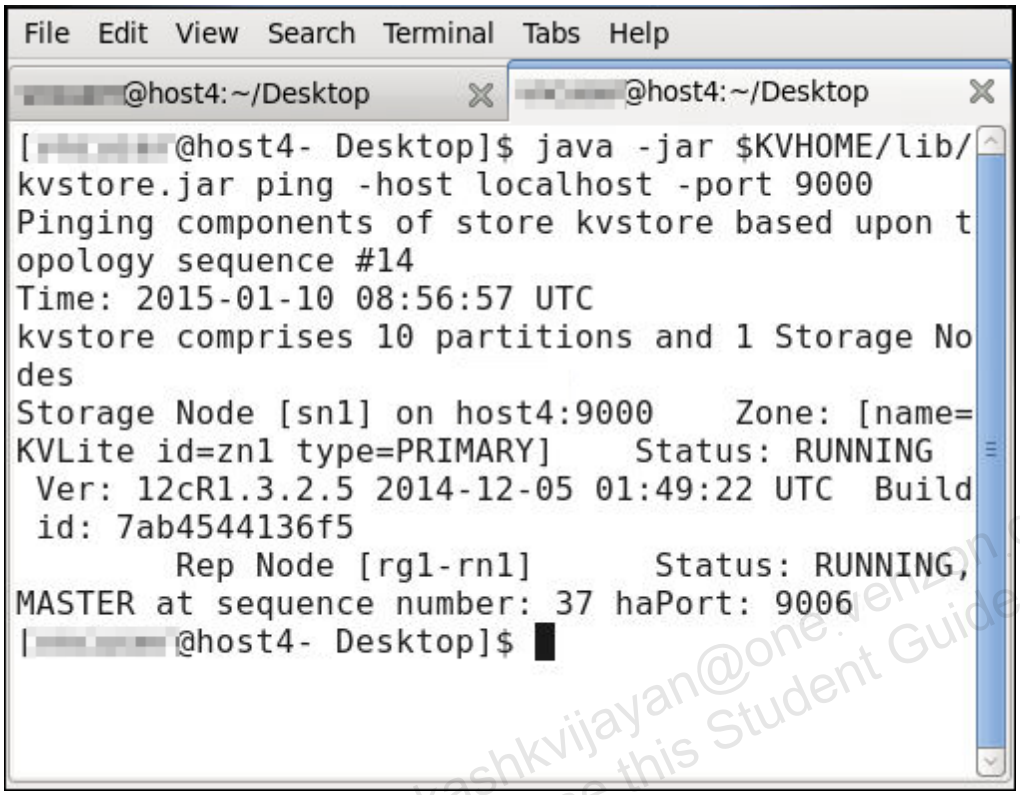

d. Open another terminal and verify that KVLite is running successfully.

1. Open a terminal window or a tab in the existing window and run the following command:

```
java -jar $KVHOME/lib/kvstore.jar ping -host localhost -port
9000
```

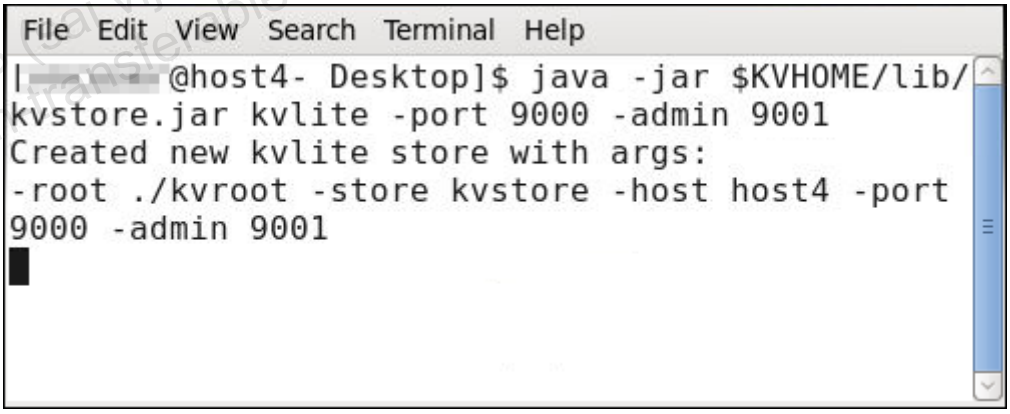


2. Note the message that is displayed.

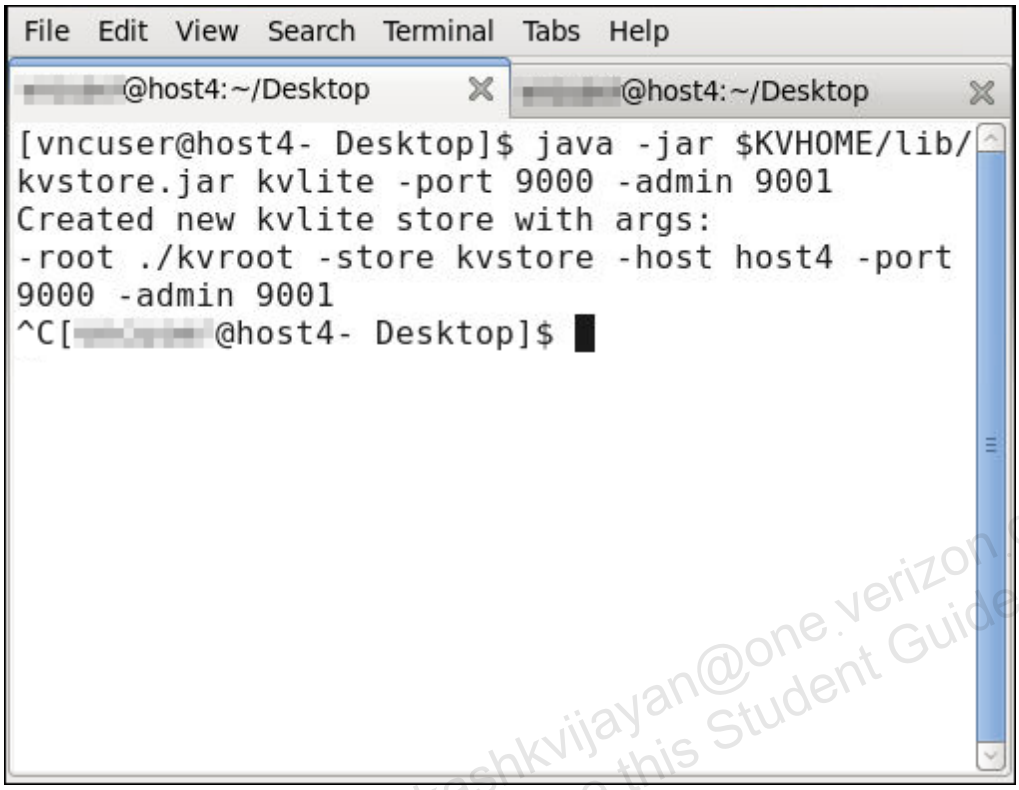



e. Stop KVLite.

1. Switch to the terminal window where KVLite is running. Note that the command prompt is not displayed.



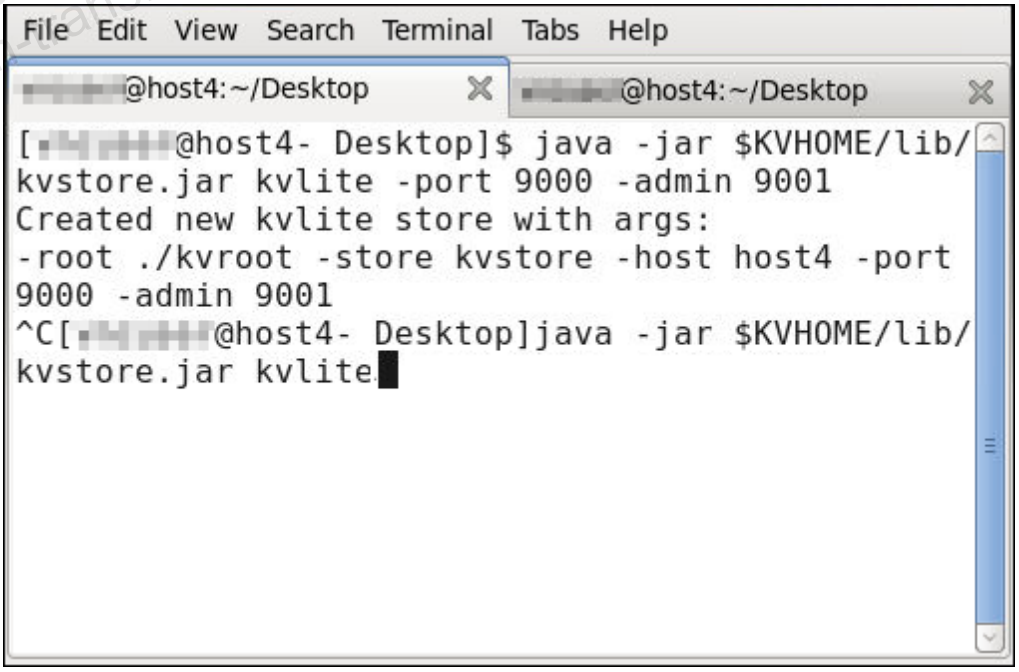
2. Press Ctrl + C. Note that the command prompt is now displayed.



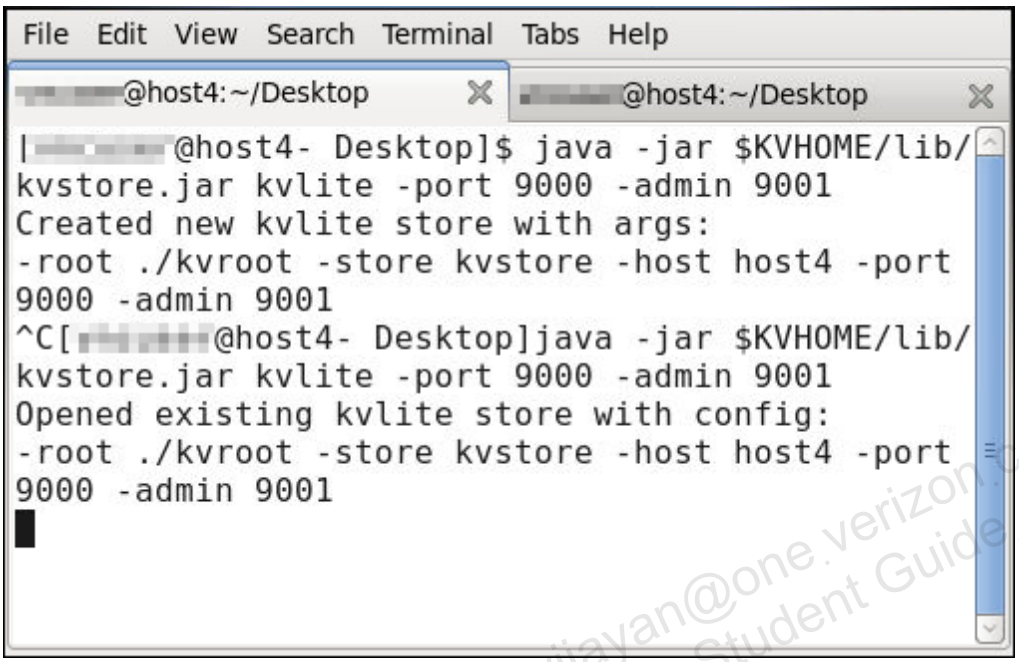

f. Start KVLite again without specifying any parameters. What message is displayed in the terminal console?

1. Run the following command:

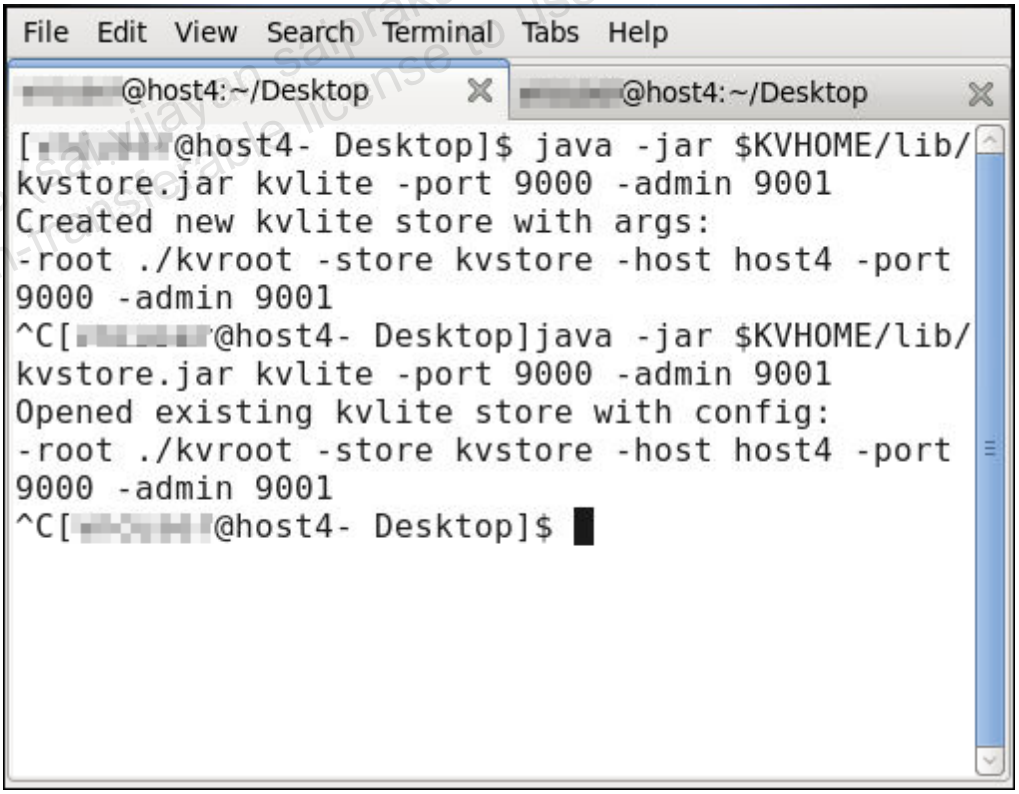
```
java -jar $KVHOME/lib/kvstore.jar kvlite
```



2. Note the message that is displayed. Because you started KVLite previously, the same store is opened with the same parameter values.




3. Press Ctrl + C to stop KVLite.



# Solution 3-2: Understanding Oracle NoSQL Database

## Overview

In this practice, you answer questions to check your understanding of the concepts taught in the lesson.

## Assumptions

None

## Questions

a. Review the following notes written by Doris and check if her understanding is correct.

| Notes | True? |
|---|---|
| Java proficiency is required to be able to develop applications using Oracle NoSQL Database. | **Yes** |
| Oracle NoSQL Database guarantees high data availability by preventing any single point of failure. | **Yes** |
| ONDB is only suited for applications with high read performance requirements and not recommended for applications requiring high write performance. | **No** |
| Oracle NoSQL Database does not allow any schema definition for big data. | **No** |
| Oracle NoSQL Database can be installed to provide read and write performances according to the application requirements. | **Yes** |
| Oracle NoSQL Database installation is very complicated and requires expert administrators for setting up a testing environment. | **No** |

b. Match the given definitions to the correct terms.

| Definition | Terms |
|---|---|
| A collection of distributed systems communicating with each other and providing voluminous data storage | **KVStore** |
| A physical machine with local storage space | **Storage Nodes** |
| A location where ONDB data is stored | **Replication Nodes** |
| A group of data nodes that contain a set of similar data in the KVStore | **Shards** |
| A collection of data nodes physically separate from the rest of the data nodes and allowing read as well as write operations | **Primary Zone** |
| A collection of nodes serving only read requests | **Secondary Zone** |

# Practices for Lesson 4: Schema Design

**Chapter 4**

# Practice 4-1: Designing a Schema

## Overview

In this practice, you answer questions to check your understanding of the concepts taught in the lesson.

## Assumptions

None

## Questions

a. Which of the following statements are true about keys in Oracle NoSQL Database's key-value model? (Choose all that are correct.)
   - Every key must consist of major and minor components.
   - A key must be of the `String` data type only.
   - A key must have at least one major component.
   - A key must have at least one minor component.

b. Records are stored in the KVStore depending on the:
   - Major-key component
   - Minor-key component
   - Primary key
   - Shard key
   - Record fields
   - Data value

c. Consider the following information that needs to be stored in a KVStore. How would you design the record structure for this information using the key-value model?
   - Data to be stored: first name, last name, date of birth, image, voice, complete home address, gender, and areas of interest
   - Details such as date of birth, address, gender, and so on that will always be accessed and updated together
   - Image and voice data that are accessed independently and rarely

d. The Earnback application has many modules that need to be developed in order to implement the new feature requirements. One of the modules is called “User Profiles”. In this module, all aspects of user information and web interaction are captured and stored to enable personalization of a user’s web experience.

Consider the following requirements for this feature and design the schema using a table data model.

1) Users can log in to the system by registering.
2) User information needs to be stored for each customer using the Earnback online points system.
3) Every user should be uniquely identified. During creation of a user, the application must ensure that a similar user does not already exist in the database,
4) For each user, the system should store their first and last names, gender, nationality, country of residence, and birthday.
5) When a user registers, a virtual or real card is created for each user.
6) An Earnback card has validity, status, and a unique 16-digit number.
7) Only one Earnback card can be active at a time for a user.
8) For each user, the system should also store the various bank accounts and credit card accounts from which the points are collected.
9) The various web commerce sites and retail shops that are linked to Earnback points should also be stored.
10) The system should be able to track the total points available for a customer at any point in time. Points collected from individual accounts should also be maintained.
11) Details of each redemption transaction should be maintained. Redemption can be done by purchasing a product from the Earnback site.
12) The total customers of the system, their Earnback card details, card active status, etc. should be maintained.

There are many interactions required in this module. The first few requirements are listed below.

13) Users should be able to register into the system
14) Provision to change password
15) When the users log in to the online Earnback application system, they should be able to view the following details:
    - Personal Profile
    - Points Summary
    - Redemption History

e. A leading credit card company wants to use the Oracle NoSQL Database to generate fraudulent scores while a transaction occurs in interactive time. Each time a credit card user swipes the credit card for a purchase, the transaction has to be assigned a fraud score based on a large number of rules as well as details specific to the transaction. In order to achieve this goal, the following details need to be stored and maintained in the database.

- Personal information for each credit card customer including firstname, lastname, date of birth, age, gender, signature image, nationality, and country of residence
- Credit cards used by each user and the issuing banks
- A master copy of all the credit cards issued by the various banks to its customers

- The rules or policies against which each transaction needs to be validated. This will depend on the bank issuing the credit card and the nature of the transaction.
- Details of each credit card transaction including geographical location and merchant name
- During each credit card transaction, the system should be able to retrieve the details of the most recent transaction on the card in order to validate the transaction.

Based on this information that needs to be stored in the database, design the schema using the table model.

## Solution 4-1: Designing a Schema

### Overview

The answers to Practice 4-1 are formatted in **bold**.

### Answers

### Overview

In this practice, you answer questions to check your understanding of the concepts taught in the lesson.

### Assumptions

None

### Questions

a. Which of the following statements are true about keys in Oracle NoSQL Database's key-value model? (Choose all that are correct.)

- Every key must consist of major and minor components.
- **A key must be of the `String` data type only.**
- **A key must have at least one major component.**
- A key must have at least one minor component.

b. Records are stored in the KVStore depending on the:

- **Major-key component**
- Minor-key component
- Primary key
- **Shard key**
- Record fields
- Data value

c. Consider the following information that needs to be stored in a KVStore. How would you design the record structure for this information using the key-value model?

- Data to be stored: first name, last name, date of birth, image, voice, complete home address, gender, and areas of interest
- Details such as date of birth, address, gender, and so on that will always be accessed and updated together
- Image and voice data that are accessed independently and rarely

**You can design two major-key components and one minor-key component for this information.**
**Major keys: `firstname` and `lastname`**
**Minor key values: `info`, `image`, and `voice`**

```
firstname/lastname.info
firstname/lastname.image
firstname/lastname.voice
```

**The data value for the record with the `info` minor-key component contains information such as date of birth, home address, areas of interest, and gender.**

d. The Earnback application has many modules that need to be developed in order to implement the new feature requirements. One of the modules is called “User Profiles”. In this module, all aspects of user information and web interaction are captured and stored to enable personalization of a user’s web experience.

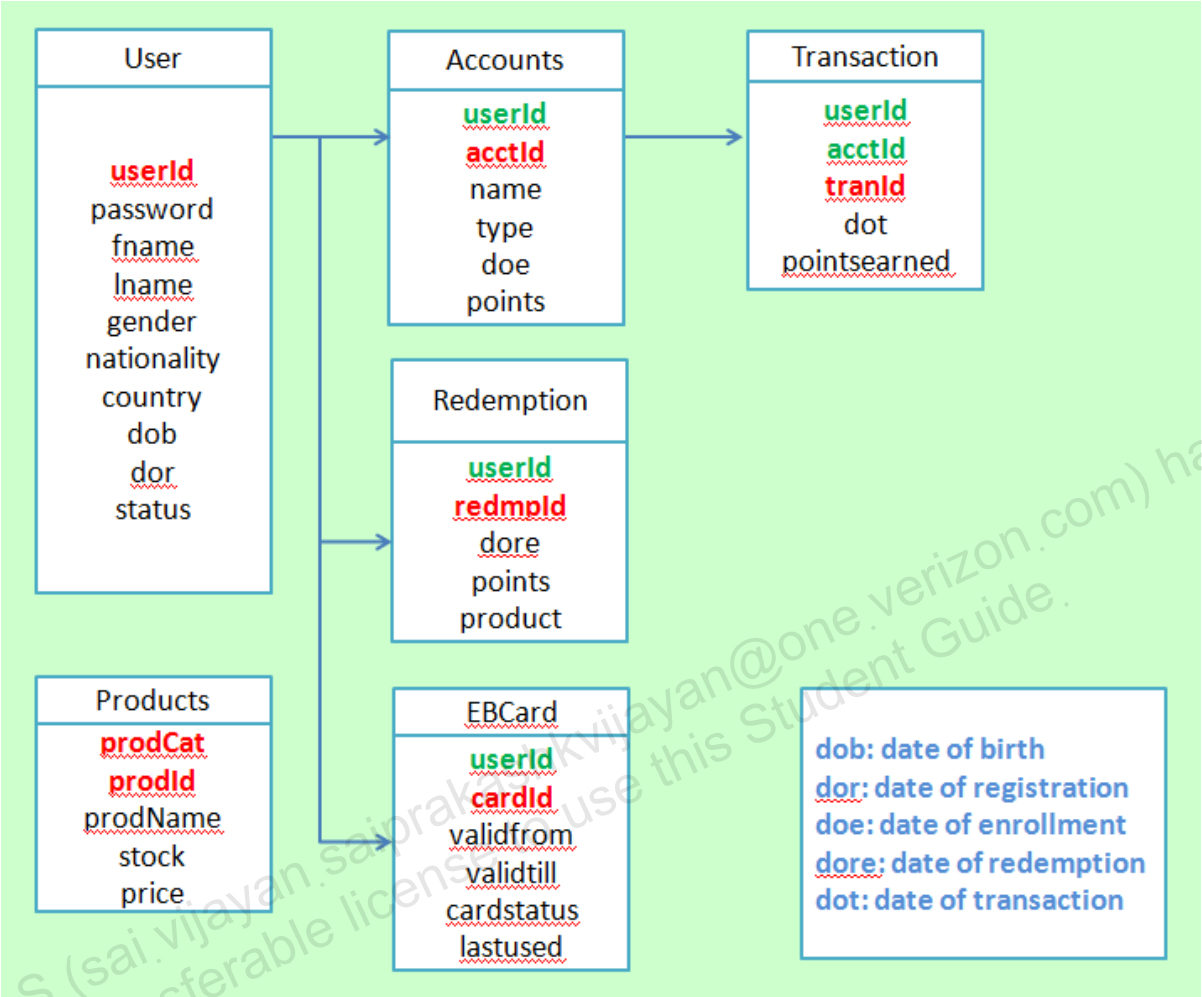
Consider the following requirements for this feature and design the schema using a table data model.

1) Users can log in to the system by registering.
2) User information needs to be stored for each customer using the Earnback online points system.
3) Every user should be uniquely identified. During the creation of a user, the application must ensure that a similar user does not already exist in the database,
4) For each user, the system should store their first and last names, gender, nationality, country of residence, and birthday.
5) When a user registers, a virtual or real card is created for each user.
6) An Earnback card has validity, status, and a unique 16-digit number.
7) Only one Earnback card can be active at a time for a user.
8) For each user, the system should also store the various bank accounts and credit card accounts from which the points are collected.
9) The various web commerce sites and retail shops that are linked to Earnback points should also be stored.
10) The system should be able to track the total points available for a customer at any point in time. Points collected from individual accounts should also be maintained.
11) Details of each redemption transaction should be maintained. Redemption can be done by purchasing a product from the Earnback site.
12) The total customers of the system, their Earnback card details, card active status, etc. should be maintained.

There are many interactions required in this module. The first few requirements are listed below.

13) Users should be able to register into the system
14) Provision to change password
15) When the users log in to the online Earnback application system, they should be able to view the following details:
    - Personal Profile
    - Points Summary

- Redemption History



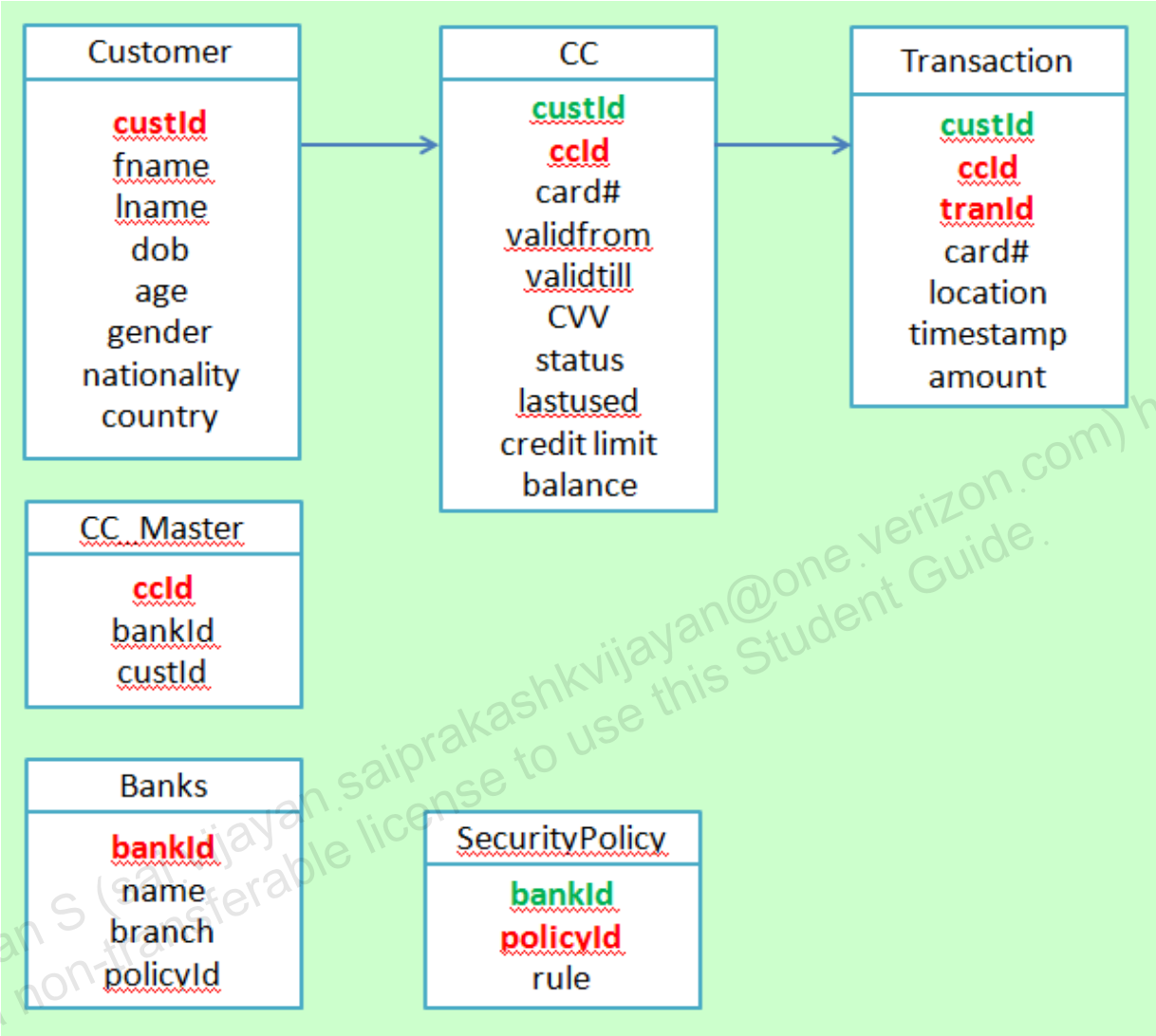

| Name | Parent | Description |
|---|---|---|
| User | - | To store user details on registration and to retrieve while displaying user profile |
| Accounts | User | To store the various accounts from which points are earned by the users. Points can be earned from credit cards, debit cards, online retail stores, and from shopping malls and stores. It also stores the date users enrolled into the Earnback system from a particular account.<br>This data is used to view the various accounts from which users earn points. |
| Redemption | User | To store the redemption details of each user. When was the redemption done, how many points were used, and what was purchased.<br>This data is used to show the users their redemption history. |
| EBCard | User | To store a unique 16-digit card issued to each user. This card may be a hard copy received from stores or virtual |

| | | |
|---|---|---|
| | | cards issued by online stores. Each card has a validity period which is used to validate transactions. If a card is lost or invalid, a new card can be issued to the user. But, at any given point in time, only one card can be active for the user. |
| Transactions | User.Accounts | To store individual transactions from which points were earned. This data is used to find out from where the points were earned by the users. |
| Products | - | To store and list products that can be purchased while redeeming points or general online shopping.<br>When a product is purchased, quantity in stock will decrease accordingly and points from the user's total points will also decrease. |

e. A leading credit card company wants to use the Oracle NoSQL Database to generate fraudulent scores while a transaction occurs in interactive time. Each time a credit card user swipes the credit card for a purchase, the transaction has to be assigned a fraud score based on a large number of rules as well as details specific to the transaction. In order to achieve this goal, the following details need to be stored and maintained in the database.

- Personal information for each credit card customer including firstname, lastname, date of birth, age, gender, signature image, nationality, and country of residence
- Credit cards used by each user and the issuing banks
- A master copy of all the credit cards issued by the various banks to its customers
- The rules or policies against which each transaction needs to be validated. This will depend on the bank issuing the credit card and the nature of the transaction.
- Details of each credit card transaction including geographical location and merchant name
- During each credit card transaction, the system should be able to retrieve the details of the most recent transaction on the card in order to validate the transaction.

Based on this information that needs to be stored in the database, design the schema using the table model.




| Name | Parent | Description |
|---|---|---|
| Customer | - | To store the details of all the customers using credit cards. |
| CC | User | To store details of the credit cards used by customers. The card details, limit and balance, and last used details. |
| Transaction | User.CC | To store each individual transaction occurring in the credit cards. |
| CC_Master | - | To store the details of all the credit cards held by each customer. |
| Banks | - | To store the details of the banks issuing the credit cards. |
| SecurityPolicy | - | To store the security policy rules that needs to be validated for transactions. |

# Practices for Lesson 5: Understanding Consistency

**Chapter 5**

# Practice 5-1: Understanding Consistency and Durability

## Overview

In this practice, you answer questions to check your understanding of the concepts taught in the lesson.

## Assumptions

None

## Questions

a. If a record stored in a replica node is guaranteed to be identical to the same record stored in the master node, the application is said to have a _____.

b. Which of the following statements is true regarding the `Consistency.NONE_REQUIRED` policy?
   - This consistency guarantee should be used sparingly.
   - This is the most relaxed consistency guarantee.
   - There is a high possibility that your application is operating with out-of-date information.
   - The operation should be serviced at the master node.

c. A version-based consistency policy ensures that a read performed on a replica is at least as current as some previous write performed on the master.
   - True
   - False

d. In which node is a write operation first performed?
   - Master node
   - Replica node

e. Which node waits for acknowledgment before completing a write operation?
   - Master node
   - Replica node

f. Which of the following is the slowest but most durable synchronization policy?
   - NO_SYNC
   - WRITE_NO_SYNC
   - SYNC

g. Match the following appropriately:

| | |
|---|---|
| SYNC | a. The write operation is placed in the in-memory cache and then placed in the file system's data buffers, but it is not necessarily transferred to stable storage. |
| NO_SYNC | b. The write operation is written to the in-memory cache, transferred to the file system's data buffers, and then synchronized to stable storage. |
| WRITE_NO_SYNC | c. The write operation is placed in the host's in-memory cache, but the master node does not wait for that operation to be written to the file system's data buffers. |

h. An acknowledgment-based durability policy specifies that a master node should wait for acknowledgments from the replica nodes before considering the write operation to have completed successfully.

- True
- False

## Practice 5-2: Identifying Consistency and Durability Requirements

### Overview

In this practice, you identify the consistency and durability requirements for the Earnback application.

### Assumptions

None

### Questions

a. The Earnback application requires high write performance. The data consistency requirements when the data is being read are not stringent. The data can be read from any node. What should be the default consistency for the KVStore supporting the Earnback application?

b. Identify the synchronization and acknowledgment policies for the master and replica nodes to configure a medium level of durability as the default durability for the Earnback application.

c. One of the transactions in the Earnback application is to be able to change the password required for login. This transaction must be made on the most recent data for the user and the password change should not be lost due to any failure. What consistency and durability policy should you apply for this particular transaction?

## Solution 5-1: Understanding Consistency and Durability

### Overview

The answers to Practice 5-1 are formatted in **bold**.

### Answers

a. If a record stored in a replica node is guaranteed to be identical to the same record stored in the master node, the application is said to have a **high consistency guarantee**.

b. Which of the following statements is true regarding the `Consistency.NONE_REQUIRED` policy?
- This consistency guarantee should be used sparingly.
- **This is the most relaxed consistency guarantee.**
- **There is a high possibility that your application is operating with out-of-date information.**
- The operation should be serviced at the master node.

c. A version-based consistency policy ensures that a read performed on a replica is at least as current as some previous write performed on the master.
- **True**
- False

d. In which node is a write operation first performed?
- **Master node**
- Replica node

e. Which node waits for acknowledgment before completing a write operation?
- **Master node**
- Replica node

f. Which of the following is the slowest but most durable synchronization policy?
- NO_SYNC
- WRITE_NO_SYNC
- **SYNC**

g. Match the following appropriately:

| | |
|---|---|
| SYNC | **b.** The write operation is written to the in-memory cache, transferred to the file system's data buffers, and then synchronized to stable storage. |
| NO_SYNC | **c.** The write operation is placed in the host's in-memory cache, but the master node does not wait for that operation to be written to the file system's data buffers. |
| WRITE_NO_SYNC | **a.** The write operation is placed in the in-memory cache and then placed in the file system's data buffers, but it is not necessarily transferred to stable storage. |

h. An acknowledgment-based durability policy specifies that a master node should wait for acknowledgments from the replica nodes before considering the write operation to have completed successfully.

- **True**
- False

## Solution 5-2: Identifying Consistency and Durability Requirements

### Overview

In this practice, you identify the consistency and durability requirements for the Earnback application.

### Assumptions

None

### Solutions

a. The Earnback application requires high write performance. The data consistency requirements when the data is being read are not stringent. The data can be read from any node. What should be the default consistency for the KVStore supporting the Earnback application?

   **`Consistency.NONE`**

b. Identify the synchronization and acknowledgment policies for the master and replica nodes to configure a medium level of durability as the default durability for the Earnback application.

   **`SYNC` at master, `WRITE_NO_SYNC` at replicas, and `SIMPLE_MAJORITY` acknowledgement**

c. One of the transactions in the Earnback application is to be able to change the password required for login. This transaction must be made on the most recent data for the user and the password change should not be lost due to any failure. What is the highest level of consistency and durability policy you can apply for this particular transaction?

   Consistency – Consistency.ABSOLUTE

   Durability - **`SYNC` at master, `SYNC` at replicas, and `ALL` acknowledgement**

# Practices for Lesson 6: Creating Tables

**Chapter 6**

## Practice 6-1: Accessing the KVStore Tables

### Overview

In this practice, you create code required to access the KVStore and the Tables APIs.

### Tasks

a. Start NetBeans and open the `/home/oracle/labs/dev/practices/Earnback-Practices Unit II` Java Application project. You perform all the lab exercise for Unit II of this course in this project. Add the `kvclient.jar` file from the `$KVHOME/lib` directory to this project.

b. To perform any operation on a KVStore, you first need to obtain a KVStore handle. Complete the code in the `fetchKVStore` method of the `BaseDAO` class to obtain and return a KVStore handle.

c. To perform any operation on KVStore tables, you need to obtain a `TableAPI` handle. Complete the code in the `fetchTableAPI` method of the `BaseDAO` class to obtain and return a `TableAPI` handle.

# Practice 6-2: Creating Tables from a Java Application

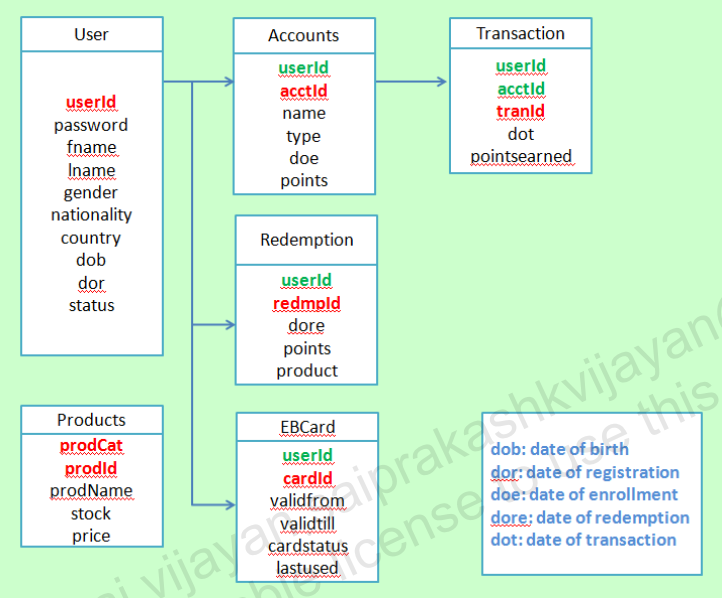
## Overview

In this practice, you create the `products` table identified for the Earnback application.

## Tasks

a. Create the `Products` table identified for the Earnback application from the Earnback application itself. Select appropriate data types for the identified fields.

Create the code in the `ProductDAO.java` class. Complete the `createTable()` method. Verify the created table using data CLI.



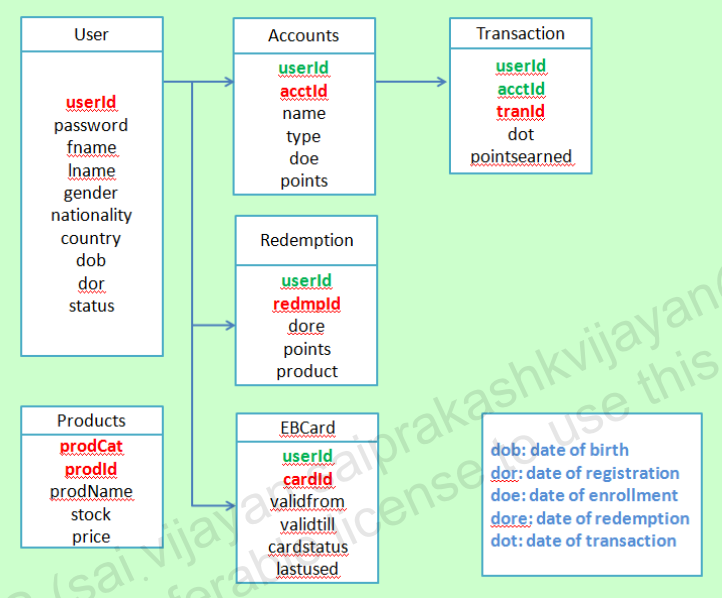
# Practice 6-3: Creating Tables from CLI

## Overview

In this practice, you create the remaining tables identified for the Earnback application.

## Tasks

a. Create the `User` table for the Earnback application by executing the DDL command at the `kv` prompt. Select appropriate data types for the identified fields. The `fname` and `lname` fields should be stored together as a record. The `gender` and `status` fields should be stored as `ENUM`s having values [F,M] and [ACTIVE,BLOCKED,CANCELLED] respectively. Use a `String` data type for all fields.




b. Create the DLL command for creating the `Redemption` and `EBCard` tables and save the command in a script file. Use the `load` command at the `kv` prompt to run the script file. For the `Redemption` table, use `Integer` data type for the `points` field and `String` for all other fields. For the `EBCard` table, use `ENUM` for `cardstatus` having values [active, blocked, cancelled] and `String` for all other fields. The `lastused` field should be a record storing Merchant, Location, and Time details.

c. Exit from the `kv` prompt if running in a terminal. Create the DLL command for creating the `Accounts` and `Transactions` table and save the command in a script file. Run the command to invoke the CLI and pass the script file to execute. For both the tables, use `Integer` for the points fields and `String` for all the other fields.

## Practice 6-4: Creating DDL to Alter Tables

### Overview

In this practice, you modify the `products` table for the Earnback application.

### Tasks

a. Previously, you created a `products` table for the Earnback application. Review the schema for this table. Modify this table and create another column called `AllowRedeem`. This column will hold one of the Boolean values `true` or `false`. Confirm that the field was added. Modify the table again and remove this field.

# Solution 6-1: Accessing the KVStore Tables

## Overview

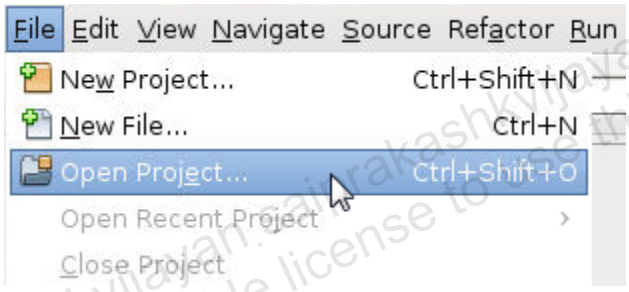
In this solution, the code to create required access to the KVStore and the Tables APIs are given.

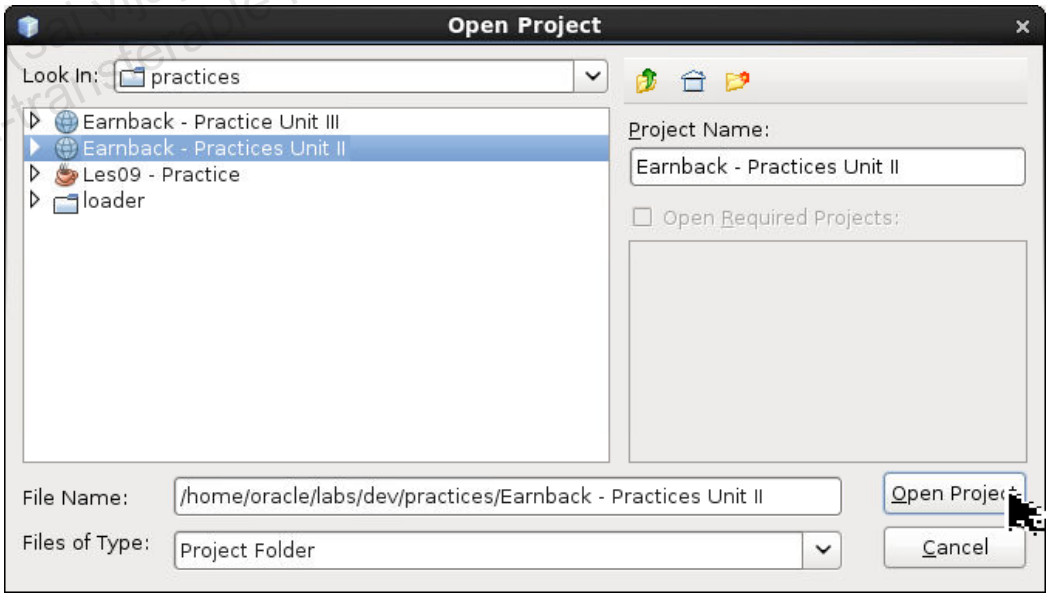
## Steps

a. Start NetBeans and open the `/home/oracle/labs/dev/practices/Earnback-Practices Unit II` Java Application project. You perform all the lab exercise for Unit II of this course in this project. Add the `kvclient.jar` file from the `$KVHOME/lib` directory to this project.

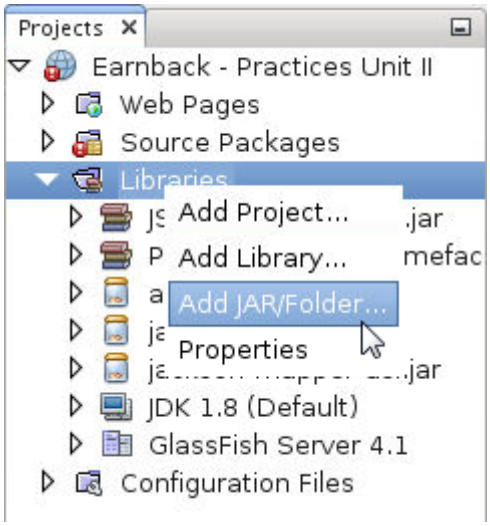
1. Use the icon on the desktop of `host4` to start Netbeans.

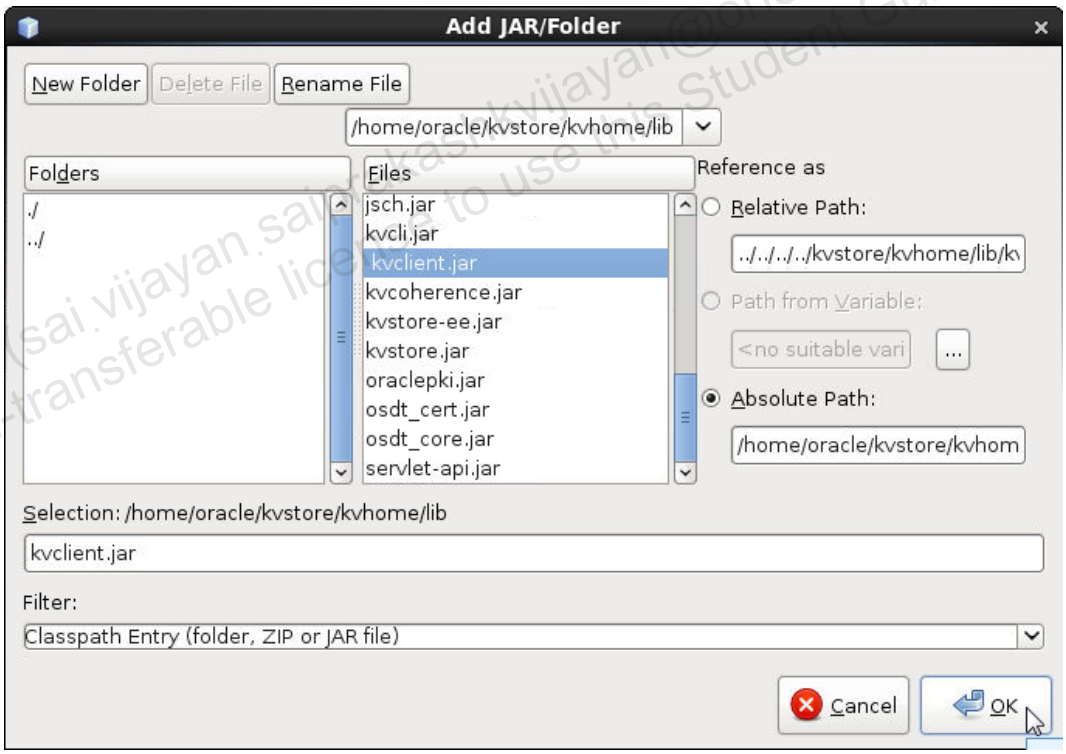

2. Open the ~/`labs/dev/practices/Earnback - Practice Unit II` project.



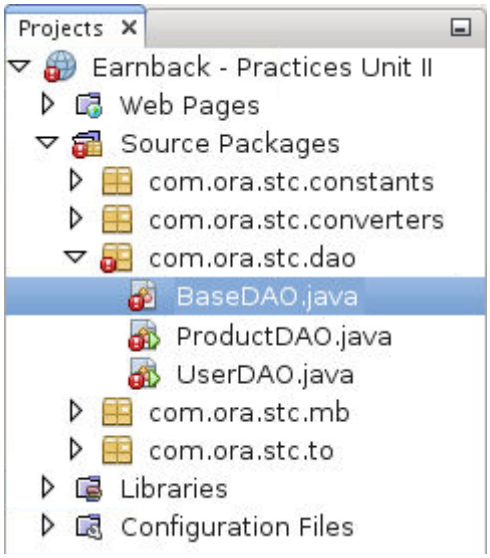
3. Expand libraries. Here, you will add the `kvclient.jar` file. Right-click and select **Add JAR/Folder…** .Browse and select the `kvclient.jar` file from the `$KVHOME/lib` directory.



b. To perform any operation on a KVStore, you first need to obtain a KVStore handle. Complete the code in the `fetchKVStore` method of the `BaseDAO` class to obtain and return a KVStore handle.

1. Under Source Packages, expand `com.ora.stc.dao` and open `BaseDAO.java`.




2. You need to create a `KVStoreConfig` object with the configuration details and then use the `getStore()` method of the `KVStoreFactory` class to obtain the KVStore handle. To do this, add the following lines of code after the TODO comment in the `fetchKVStore()` method.

```
KVStoreConfig myKVStoreConfig = new KVStoreConfig(storeName,
   hostPort);
kvStore = KVStoreFactory.getStore(myKVStoreConfig);
```

3. Add the following import statements at the beginning of the file.

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

4. Ensure that there are no errors in the `fetchKVStore()` method and save `BaseDAO.java`.

c. To perform any operation on KVStore tables, you need to obtain a `TableAPI` handle. Complete the code in the `fetchTableAPI` method of the `BaseDAO` class to obtain and return a `TableAPI` handle.

1. You need to use the `getTableAPI()` method of the KVStore class to obtain a `TableAPI` handle. To obtain a KVStore handle you can use the method you completed in the previous task. Add the following lines of code after the TODO comment in the `fetchTableAPI()` method.

```
tableAPI = fetchKVStore().getTableAPI();
```

2. Add the following import statements at the beginning of the file.

```
import oracle.kv.table.TableAPI;
```

3. Ensure that there are no errors in the `fetchTableAPI()` method and save `BaseDAO.java`.

# Solution 6-2: Creating Tables from a Java Application

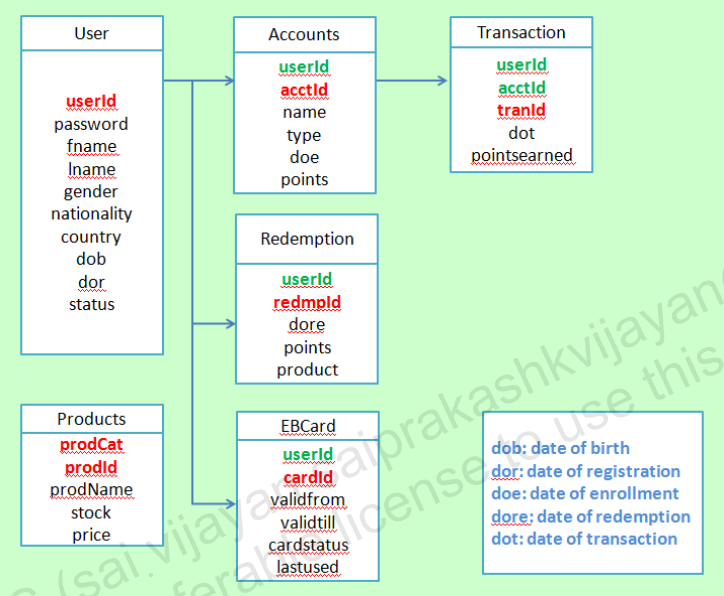
## Overview

In this solution, the steps to create the `products` table identified for the Earnback application using a Java application are given.

## Steps

a. Create the `Products` table identified for the Earnback application from the Earnback application itself. Select appropriate data types for the identified fields.

Create the code in the `ProductDAO.java` class. Complete the `createTable()` method. Verify the created table using data CLI.




1. Open the `ProductDAO.java` class.
2. You need to create the DDL statement and assign it to a variable of type STRING. In the `createTable()` method, add the following line before the `try` statement.

```
String tableStr = "CREATE TABLE products(prodCat STRING,
prodId String, prodName STRING, stock INTEGER, price INTEGER,
PRIMARY KEY (prodCat, prodId))";
```

b. You now need to obtain a `TableAPI` handle. You can do this by using the `fetchTableAPI` method of the BaseDAO class, which you completed in the previous practice. Then, you use the `execute` or `executeSync` method to run the DDL statement. Add the following lines of code after the TODO comment.

Note: Due to a typo in the lab file, the TODO statement reads **6-2, a** instead of **6-2, b.**

```
TableAPI myTableAPI = BaseDAO.fetchTableAPI();
StatementResult sr = myTableAPI.executeSync(tableStr);
```

1. Add the following import statements.

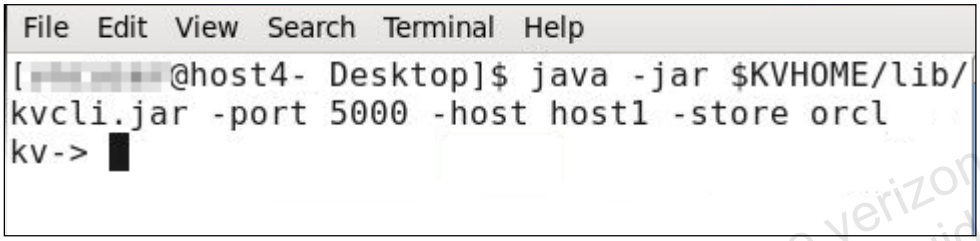
```
import oracle.kv.table.TableAPI;
import oracle.kv.table.StatementResult;
```

c. Add the following lines of code in the `main()` method to create an instance of the `ProductDAO` class and invoke the `createTable` method.

```
ProductDAO myProductDAO = new ProductDAO();
myProductDAO.createTable();
```

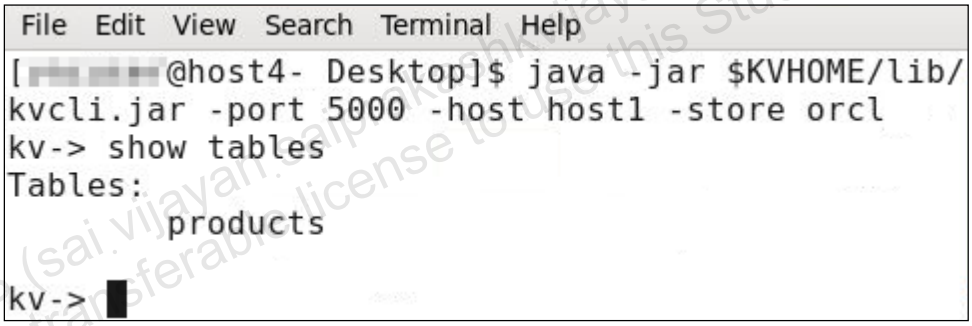
1. Ensure that there are no errors. Right-click `ProductDAO.java` and click Run.
2. After the table has been created, comment the lines of code you added to the `main()` method.
3. In a terminal window, invoke the data CLI using the following command.

```
java -jar $KVHOME/lib/kvcli.jar -port 5000 -host host1 -store
orcl
```




4. Verify the table created by using the following command.

```
show tables
```



d. View the schema for the `products` table.

```
show tables -name products
```

```
kv-> show tables -name products
{
  "type" : "table",
  "name" : "products",
  "comment" : null,
  "shardKey" : [ "prodCat", "prodId" ],
  "primaryKey" : [ "prodCat", "prodId" ],
  "fields" : [ {
    "name" : "prodCat",
    "type" : "STRING",
    "nullable" : true,
    "default" : null
  }, {
    "name" : "prodId",
    "type" : "STRING",
    "nullable" : true,
    "default" : null
  }, {
    "name" : "prodName",
    "type" : "STRING",
    "nullable" : true,
    "default" : null
  }, {
    "name" : "stock",
    "type" : "INTEGER",
    "nullable" : true,
    "default" : null
  }, {
    "name" : "price",
    "type" : "INTEGER",
    "nullable" : true,
    "default" : null
  } ]
}
```

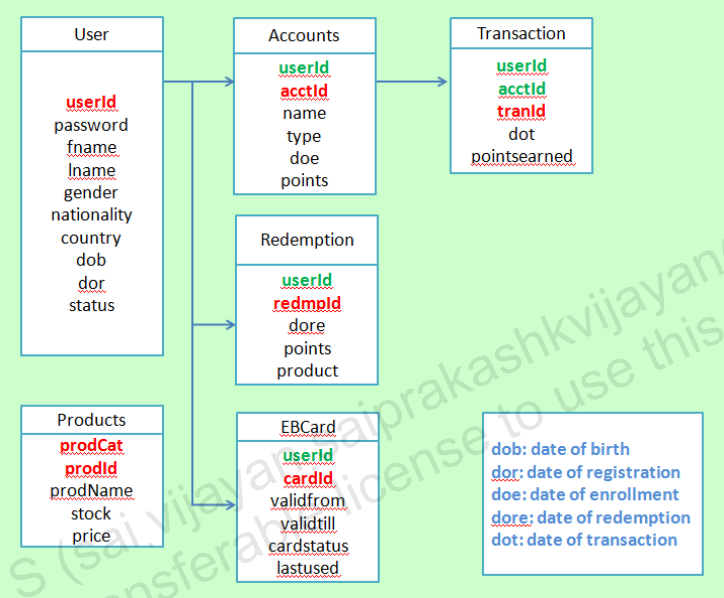
# Solution 6-3: Creating Tables from CLI

## Overview

In this solution, the steps to create the remaining tables identified for the Earnback application from a CLI are given.

## Steps

a. Create the `User` table for the Earnback application by executing the DDL command at the `kv` prompt. Select appropriate data types for the identified fields. The `fname` and `lname` fields should be stored together as a record. The `gender` and `status` fields should be stored as `ENUM`s having values [F,M] and [ACTIVE,BLOCKED,CANCELLED] respectively. Use a `String` data type for all fields.




1. Focus on the terminal where data CLI is running.

b. Create the DDL statement to create the user table and use the `execute` command to run the statement. Enter the following command to execute the DLL statement at the `kv` prompt.

```
execute "CREATE TABLE user(userId STRING, password STRING, name
RECORD (first STRING, last STRING), gender ENUM(M,F),
nationality STRING, country STRING, dob STRING, dor STRING,
status ENUM (ACTIVE,BLOCKED, CANCELLED), PRIMARY KEY (userId))"
```

1. Confirm that the table was created.

```
kv-> execute "CREATE TABLE user(userId STRING, p
assword STRING, name RECORD (first STRING, last
STRING), gender ENUM(M,F), nationality STRING, c
ountry STRING, dob STRING, dor STRING, status EN
UM (ACTIVE,BLOCKED, CANCELLED), PRIMARY KEY (use
rId))"
Statement completed successfully
kv-> show tables
Tables:
        products
        user

kv->
```

c. Create the DLL command for creating the `Redemption` and `EBCard` tables and save the command in a script file. Use the `load` command at the `kv` prompt to run the script file. For the `Redemption` table, use an `Integer` data type for the `points` field and `String` for all the other fields. For the `EBCard` table, use `ENUM` for `cardstatus` having values [active,blocked,cancelled] and `String` for all the other fields. The `lastused` field should be a record storing Merchant, Location, and Time details.

1. Add the following commands to create the tables in a text file and save the file in the ~/`labs/dev/practices` directory as `createTables1.kvs`.

```
execute "CREATE TABLE user.redemption(redmpId String, dore
STRING, points INTEGER, product STRING, PRIMARY KEY
(redmpId))"
execute "CREATE TABLE user.ebcard(cardId String, validfrom
STRING, validtill STRING, cardstatus ENUM(active, blocked,
cancelled), lastused RECORD (Merchant STRING, Location
STRING, TIME STRING), PRIMARY KEY (cardId))"
```

2. Enter the following command at the `kv` prompt in the data CLI.

```
load -file /home/oracle/labs/dev/practices/createTables1.kvs
```

3. View the tables created so far.

```
show tables
```

```
kv-> load -file /home/oracle/labs/dev/practices/
createTables1.kvs
Statement completed successfully
Statement completed successfully

kv-> show tables
Tables:
        products
        user
          user.ebcard
          user.redemption

kv->
```

d. Exit from the `kv` prompt if running in a terminal. Create the DLL command for creating the `Accounts` and `Transactions` table and save the command in a script file. Run the command to invoke the CLI and pass the script file to execute. For both the tables, use `Integer` for the points fields and `String` for all the other fields.

1. At the data CLI `kv` prompt, enter `exit`.
2. Add the following commands to create the tables in a text file and save the file in the ~/`labs/dev/practices` directory as `createTables2.kvs`.

```
connect store –name orcl
execute "CREATE TABLE user.accounts(acctId STRING, name
STRING, type STRING, doe STRING, points INTEGER, PRIMARY KEY
(acctId))"
execute "CREATE TABLE user.accounts.transaction(tranId
String, dot STRING, points INTEGER, PRIMARY KEY (tranId))"
```

3. Enter the following command at the terminal prompt.

```
java -jar $KVHOME/lib/kvstore.jar runadmin -port 5000 -host
host1 load –file
/home/oracle/labs/dev/practices/createTables2.kvs
```


4. View the tables created so far.

```
show tables
```

## Solution 6-4: Creating DDL to Alter Tables

### Overview

In this solution, the steps to modify the `products` table for the Earnback application are given.

### Steps

a. Previously, you created a `products` table for the Earnback Application. Review the schema for this table. Modify this table and create another column called `AllowRedeem`. This column will hold one of the Boolean values `true` or `false`. Confirm that the field was added. Modify the table again and remove this field.

1. Invoke the data CLI, if it is not already running in a terminal.

```
java -jar $KVHOME/lib/kvcli.jar -port 5000 -host host1 -store
orcl
```

2. View the `products` table schema.

```
show tables –name products
```

3. Enter the DDL command to modify the table at the `kv` prompt.

```
execute "ALTER TABLE products (ADD AllowRedeem BOOLEAN)"
```

4. Confirm the change by viewing the table schema again.

```
show tables –name products
```

```
kv-> execute "ALTER TABLE products (ADD AllowRed
eem BOOLEAN)"
Statement completed successfully
kv-> show tables -name products
```

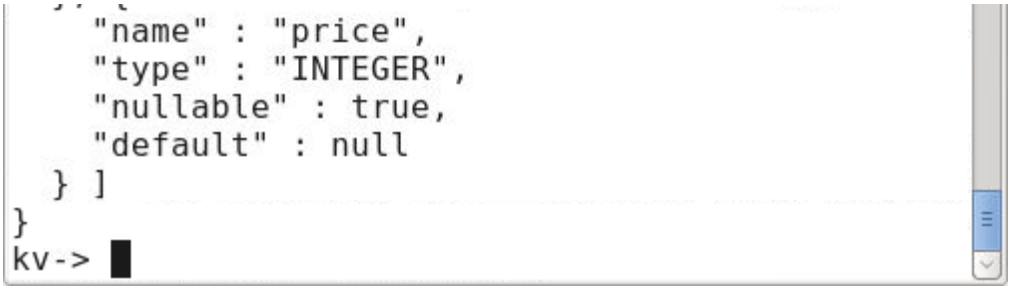
```
    "name" : "AllowRedeem",
    "type" : "BOOLEAN",
    "nullable" : true,
    "default" : null
  } ]
}
kv->
```

5. Enter the DDL command to modify the table again at the `kv` prompt.

```
execute "ALTER TABLE products (DROP AllowRedeem)"
```

```
kv-> execute "ALTER TABLE products (DROP AllowRe
deem)"
Statement completed successfully
kv->
```

6. Confirm that the table was modified and the `AllowRedeem` field was removed.



# Practices for Lesson 7: Creating Table Data

**Chapter 7**

# Practice 7-1: Writing Data to Tables

## Overview

In this practice, you use the appropriate Java API to write data into the KVStore tables created for the Earnback application.

## Tasks

a. To perform any operation on a specific table, you need to obtain a handle to that table. Complete the code in the `fetchTable` method of the `BaseDAO` class to obtain and return a handle to a table.

b. In the Earnback application, you need to allow the users to register themselves. Each successful registration should result in a record creation in the `user` table. Part of the Java code to receive the user information from the registration page is completed for you. You need to write the code to insert these values into the `user` table. Review the schema for the `user` table and then review the `UserTO.java` class. Complete the code in the `registerUser()` method of the `UserDAO` class.

   **Notes:**

   - Retrieve input values from `UserTO` as:
     - `myUserTO.getUserID()`, `myUserTO.getPassword()`, `myUserTO.getGender()`, `myUserTO.getNationality()`, `myUserTO.getCountry()`, `myUserTO.getFirstName()`, `myUserTO.getLastName()`
   - To retrieve the date of birth in a string format, use:
     - `ConvertCalendar.getAsString(myUserTO.getDateOfBirth())`
   - To get the current date to use as date of registration, use:
     - `ConvertCalendar.getAsString(Calendar.getInstance())`
   - For the status field, use:
     - `UserStatus.ACTIVE`
   - Use the `version` variable to store the version value returned by the write API

c. The Earnback application uses a `products` table to store the product information for the application. Review the schema for the `products` table. Product details are received from partners in various formats. One such partner maintains its product information in text files. The data is formatted with respect to the product table columns. The code to read through the file and obtain the values in each row of the text file is completed for you. Review this code in the `EarnbackFileReader.java` class in the ~/`labs/dev/practices/loader` directory. Complete the code in the `LoadProducts.java` class. If the product exists in the table, then you need to update the row. Otherwise, you need to create a new row for the product. Compile both the Java classes and run `LoadProducts`. Confirm that the rows are inserted in the `Products` table.

## Solution 7-1: Writing Data to Tables

### Overview

This solution provides code to write data into the KVStore tables created for the Earnback application.

### Steps

a. To perform any operation on a specific table, you need to obtain a handle to that table. Complete the code in the `fetchTable` method of the `BaseDAO` class to obtain and return a handle to a table.

1. You can use the `fetchTableAPI()` method you created in the previous lesson's practice to retrieve a handle to the `TableAPI`. You then use the `getTable()` method to fetch the handle to a specific table. Add the following lines of code after the TODO comment in the `fetchTable()` method.

```
Table table = fetchTableAPI().getTable(tableName);
```

2. Add the following import statements at the beginning of the file.

```
import oracle.kv.table.Table;
```

3. Ensure that there are no errors in the `fetchTable()` method and save `BaseDAO.java`. `BaseDAO.java` should now be free of all errors.

b. In the Earnback application, you need to allow the users to register themselves. Each successful registration should result in a record creation in the `user` table. Part of the Java code to receive the user information from the registration page is completed for you. You need to write the code to insert these values into the `user` table. Review the schema for the `user` table and then review the `UserTO.java` class. Complete the code in the `registerUser()` method of the `UserDAO` class.

**Notes:**

- Retrieve input values from `UserTO` as:
  - `myUserTO.getUserID()`, `myUserTO.getPassword()`, `myUserTO.getGender()`, `myUserTO.getNationality()`, `myUserTO.getCountry()`, `myUserTO.getFirstName()`, `myUserTO.getLastName()`
- To retrieve the date of birth in a string format, use:
  - `ConvertCalendar.getAsString(myUserTO.getDateOfBirth())`
- To get the current date to use as date of registration, use:
  - `ConvertCalendar.getAsString(Calendar.getInstance())`
- For the status field, use:
  - `UserStatus.ACTIVE`
- Use the `version` variable to store the version value returned by the write API.

1. In the data CLI, review the schema for the `user` table.

```
show tables -name user
```

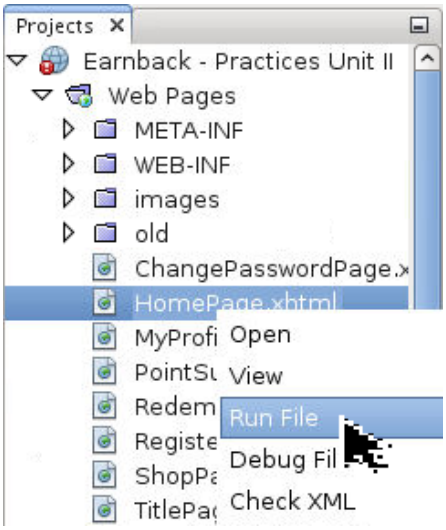
2. Switch to NetBeans and review the `UserTO.java` class.
3. Open the `UserDAO.java` class and remove the comment tags before (/*) and after (*/) the `registerUser` method. Review the existing code in the `registerUser()` method.
4. You need to fetch a handle for the `user` table. You can do this by using the `fetchTable()` method you created in the BaseDAO class. You then create a row object for the user table. Use the row.put, row.putEnum, and row.putRecord methods to insert the field values to the row object. Finally, you use the putIfAbsent() method to write the row to the table as you want to ensure a unique user is created for each registration. Review and add the following code after the TODO comment.

```
Table myTable = BaseDAO.fetchTable("user");
Row row = myTable.createRow();
row.put("userid", myUserTO.getUserId());
row.put("password", myUserTO.getPassword());
row.putEnum("gender", myUserTO.getGender().toUpperCase());
row.put("nationality",
myUserTO.getNationality().toLowerCase());
row.put("country", myUserTO.getCountry().toLowerCase());
row.put("dob",
ConvertCalendar.getAsString(myUserTO.getDateOfBirth()));
row.put("dor",
ConvertCalendar.getAsString(Calendar.getInstance()));
row.putEnum("status", UserStatus.ACTIVE);
RecordValue name = row.putRecord("name");
name.put("first", myUserTO.getFirstName());
name.put("last", myUserTO.getLastName());
Version version = BaseDAO.fetchTableAPI().putIfAbsent(row,
null, null);
```

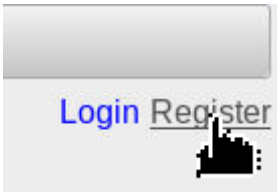
5. Add the following import statements.

```
import oracle.kv.table.Table;
import oracle.kv.table.Row;
import oracle.kv.table.RecordValue;
import oracle.kv.Version;
```

6. Under Web Pages, right-click and run the `HomePage.xhtml` page under the old folder.

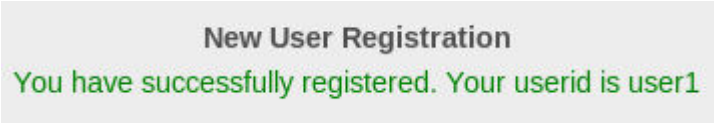



7. Click the Register link.




8. Enter the details in the register form to create a user and click Register. Create two users with userid user1 and user2.

9. You should see a message that the registration was successful.




10. Confirm that the data is present in the `user` table.

```
get table -name user
```

```
kv-> get table -name user
{"userId":"user2","password":"user2","name":{"first":"u
ser2","last":"user2"},"gender":"F","nationality":"us","
country":"germany","dob":"04-Jan-1997","dor":"05-Feb-20
15","status":"ACTIVE"}
{"userId":"user1","password":"user1","name":{"first":"u
ser1","last":"user1"},"gender":"M","nationality":"india
n","country":"canada","dob":"01-Feb-1996","dor":"05-Feb
-2015","status":"ACTIVE"}

2 rows returned
```

c. The Earnback application uses a `products` table to store the product information for the application. Review the schema for the `products` table. Product details are received from partners in various formats. One such partner maintains its product information in text files. The data is formatted with respect to the product table columns. The code to read through the file and obtain the values in each row of the text file is completed for you. Review this code in the `EarnbackFileReader.java` class in the ~/`labs/dev/practices/loader` directory. Complete the code in the `LoadProducts.java` class. If the product exists in the table, then you need to update the row. Else you need to create a new row for the product. Compile both the Java classes and run `LoadProducts`. Confirm that the rows are inserted in the `Products` table.

1. Open the ~/`labs/dev/practices/loader/LoadProducts.java` class using an editor.
2. You can use the put method to insert the product details into the products table. Add the following statement after the TODO comment for lesson 7. Save the file.

```
myStore.getTableAPI().put(myrow,null,null);
```

3. Open a terminal and ensure that the current working directory is `/home/oracle/labs/dev/practices` and compile the `EarnbackFileReader.java` and `LoadProducts.java`.

```
javac -cp ./:/home/oracle/kvstore/kvhome/lib/kvclient.jar
loader/EarnbackFileReader.java
javac -cp ./:/home/oracle/kvstore/kvhome/lib/kvclient.jar
loader/LoadProducts.java
```

```
[oracle@host4- Desktop]$ cd /home/oracle/labs/dev/practices
[oracle@host4- practices]$ pwd
/home/oracle/labs/dev/practices
[oracle@host4- practices]$ javac -cp ./:/home/oracle/kvstore/kvhome/lib/kvclient.jar loader/EarnbackFileReader.java
[oracle@host4- practices]$ javac -cp ./:/home/oracle/kvstore/kvhome/lib/kvclient.jar loader/LoadProducts.java
[oracle@host4- practices]$
```

4. Run `LoadProducts.java`.

```
java -cp ./:/home/oracle/kvstore/kvhome/lib/kvclient.jar
loader/LoadProducts
```

5. In the data CLI, verify that the rows were inserted into the `Products` table.

```
get table –name products
```

```
kv-> get table -name products
{"prodCat":"camera","prodId":"1","prodName":"nikon","stock":12,"price":145}
{"prodCat":"electronics","prodId":"3","prodName":"bose speaker","stock":3,"price":150}
{"prodCat":"electronics","prodId":"2","prodName":"dry iron","stock":2,"price":55}
{"prodCat":"electronics","prodId":"1","prodName":"philips mp3 player","stock":15,"price":79}
{"prodCat":"mobiles","prodId":"1","prodName":"nokia 1100","stock":4,"price":67}
{"prodCat":"kitchenware","prodId":"1","prodName":"tava","stock":5,"price":15}
{"prodCat":"electronics","prodId":"4","prodName":"toaster","stock":6,"price":34}

7 rows returned
```

# Practices for Lesson 8: Retrieving Table Data

**Chapter 8**

# Practice 8-1: Fetching Data from Tables

## Overview

In this practice, you use the appropriate Java API to write data into the KVStore tables created for the Earnback application.

## Tasks

a. In the Earnback application, you need to allow the users to log in to the application. A login page is created for the application. The code to receive the username and password entered by the user is completed for you. Complete the `validateUser()` method in the `UserDAO` class to verify the entered details with the existing details in the user table and return a Boolean value indicating if the login details are accurate or not. If the method returns a true, the code to display the home page for the user is completed for you.

b. The Earnback application has a tabbed page called **My Account**. One of the regions on this page is named **My Profile**. You need to display the following user details in this region:

- First name
- Last name
- DOB
- Status

Complete the code to retrieve this information in the `fetchProfileDetails()` method of the `UserDAO` class.

**Notes:**

- To store the values retrieved from the KVStore in the `UserTO` object use:
  - `myUserTO = new UserTO()`
- Use the following methods to set the field values:
  - `myUserTO.setFirstName()`
  - `myUserTO.setLastName()`
  - `myUserTO.setDateOfBirth()`
  - `myUserTO.setUserStatus()`
- To convert the date value into date format, use:
  - `ConvertCalendar.getAsCalendar()`

c. On the **My Account** page, there is another link called **Redemption History**. In this region you want to display the redemption history of the user. Write the code to retrieve the date of redemption, points used, and product bought by the user. Complete the code in the `fetchRedemptionHistory()` method of the `UserDAO` class. Run the `LoadRHData.java` file to insert some transactions for the userID you created. You will have to manually change the userID in this Java file.

d. On the **Shop** page of the Earnback application, when the **All** link is clicked, you want to display all the products available. Complete the `getAllProducts()` method in the `ProductDAO` class.

e. On the **Shop** page of the Earnback Application, when the **Electronics** link is clicked, you want to display only the electronics available. Complete the `getElectronics()` method in the `ProductDAO` class.

# Solution 8-1: Fetching Data from Tables

## Overview

This solution provides code to write data into the KVStore tables created for the Earnback application.

## Steps

a. In the Earnback application, you need to allow the users to log in to the application. A login page is created for the application. The code to receive the username and password entered by the user is completed for you. Complete the `validateUser()` method in the `UserDAO` class to verify the entered details with the existing details in the user table and return a Boolean value indicating if the login details are accurate or not. If the method returns a true the code to display the home page for the user is completed for you.

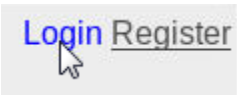
1. In NetBeans, view `validateUser()` method in the `UserDAO` class. Remove the comment tags before and after this method.
2. In order to retrieve details from the user table, you need to fetch a handle to the table and create the primary key for the record you want to fetch. Since you want to fetch a single record, you can use the `get()` method. This method returns data in a `row` object. You then retrieve the password field from the `row` object and compare it to the password entered by the user. You then return a Boolean value to indicate if the comparison was successful or not. Add the following lines of code after the TODO comment.

```
Table myTable = BaseDAO.fetchTable("user");
PrimaryKey primaryKey = myTable.createPrimaryKey();
primaryKey.put("userId", userId);

Row myRow = BaseDAO.fetchTableAPI().get(primaryKey, null);

   if (myRow != null) {
   String dbPassword = myRow.get("password").toString();
   if (dbPassword.equals(password)) {
           status = true;
    }
}
```

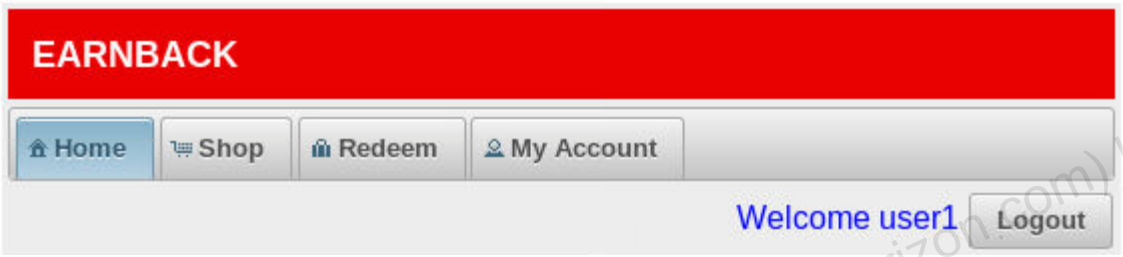
3. Add the following import statement.

```
import oracle.kv.table.PrimaryKey;
```

4. Save the `UserDAO.java` file and switch to the Earnback application web page.
5. Click login and enter the userID and password you registered in the previous practice.




6. You should be able to see the Redeem and My Account tabs now.




b. The Earnback application has a tabbed page called **My Account**. One of the regions in this page is named **My Profile**. You need to display the following user details in this region:
   - First name
   - Last name
   - DOB
   - Status

   Complete the code to retrieve this information in the `fetchProfileDetails()` method of the `UserDAO` class.

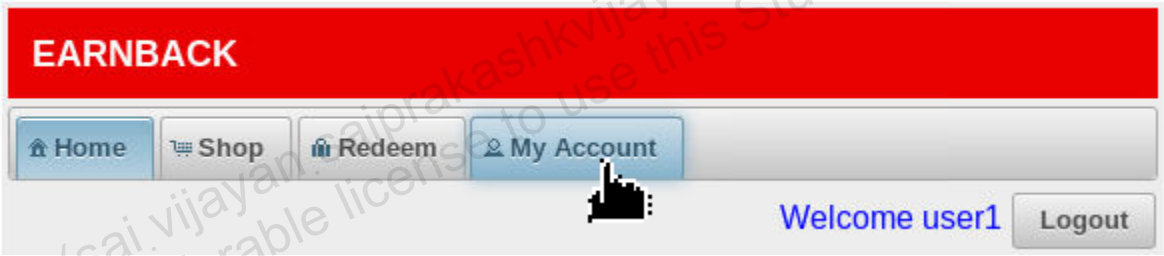
   **Notes:**
   - To store the values retrieved from the KVStore in the `UserTO` object use:
     - `myUserTO = new UserTO()`
   - Use the following methods to set the field values:
     - `myUserTO.setFirstName()`
     - `myUserTO.setLastName()`
     - `myUserTO.setDateOfBirth()`
     - `myUserTO.setUserStatus()`
   - To convert the date value into date format, use:
     - `ConvertCalendar.getAsCalendar()`

1. View the `fetchProfileDetails()` method in the `UserDAO` class. Remove the comment tags before and after the method.

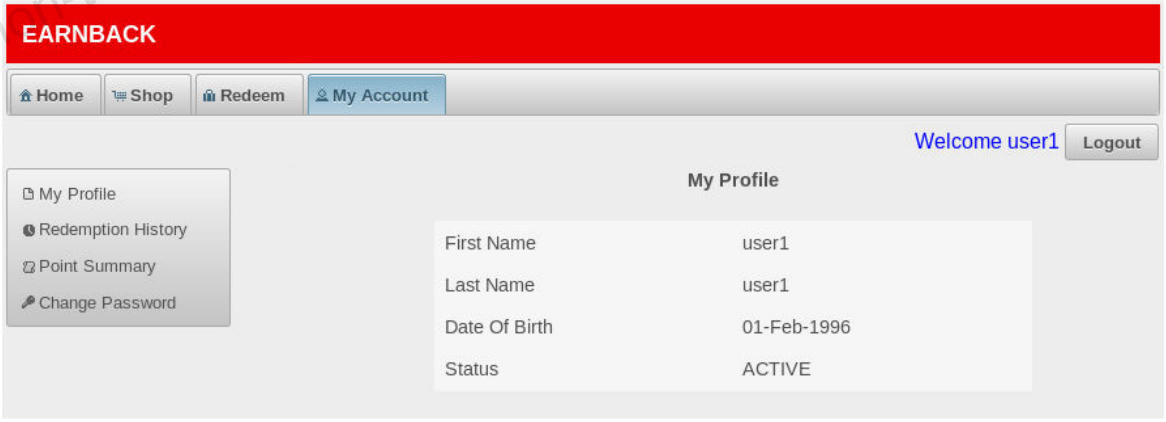
2. To retrieve details from the user table, obtain the table handle and create the primary key for the record you want to fetch. Since you want to fetch a single record, you can use the `get()` method. This method returns data in a `row` object. You then retrieve the required fields from the `row` object and set the values in `UserTO`. Add the following lines of code after the TODO comment.

```
Table myTable = BaseDAO.fetchTable("user");
PrimaryKey primaryKey = myTable.createPrimaryKey();
primaryKey.put("userId", userId);
Row myRow = BaseDAO.fetchTableAPI().get(primaryKey, null);
if (myRow != null) {
     myUserTO = new UserTO();
     RecordValue myName = myRow.get("name").asRecord();
 myUserTO.setFirstName(myName.get("first").asString().get());
 myUserTO.setLastName(myName.get("last").asString().get());
 myUserTO.setDateOfBirth(ConvertCalendar.getAsCalendar(myRow.ge
 t("dob").toString()));
 myUserTO.setUserStatus(myRow.get("status").toString());
 }
```

3. Save and switch to the Earnback application and click the My Profile link on the My Account page.




4. You should be able to see the profile details. If you do not see the data, ensure that you are logged in.




c. On the **My Account** page, there is another link called **Redemption History**. In this region you want to display the redemption history of the user. Write the code to retrieve the date of redemption, points used, and product bought by the user. Complete the code in the `fetchRedemptionHistory()` method of the `UserDAO` class. Run the

`LoadRHData.java` file to insert some transactions for the userID you created. You will have to manually change the userID in this Java file.

1. Open the `LoadRHData.java` file from the `labs/dev/practices/loader` directory and change the userID to the userID you created previously (user1 or user2, if you are following this solution guide).
2. In a terminal, compile and run `LoadRHData.java`. Note: Ensure that you are in the `/home/oracle/labs/dev/practices` folder.

```
javac -cp ./:/home/oracle/kvstore/kvhome/lib/kvclient.jar
loader/LoadRHData.java
java -cp ./:/home/oracle/kvstore/kvhome/lib/kvclient.jar
loader/LoadRHData
```

3. In NetBeans, view the `fetchRedemptionHistory()` method in the `UserDAO` class. Remove the comment tags before and after the method.
4. Obtain a handle to the child table and create the primary key using the parent key alone (as you want to retrieve all the child table rows for the user). Use the `multiGet()` method to fetch the rows from the table. Add the following lines of code after the first TODO comment.

```
Table myTable = BaseDAO.fetchTable("user.redemption");
PrimaryKey primaryKey = myTable.createPrimaryKey();
primaryKey.put("userId", userId);
List<Row> myRowSet =
 BaseDAO.fetchTableAPI().multiGet(primaryKey, null, null);
```

5. Now you need to set the fetched values into a TO object. Add the following lines of code after the TODO comment in the for loop.

```
myTO.setRedemptionId(aRow.get("redmpId").toString());

 myTO.setDateOfRedemption(ConvertCalendar.getAsCalendar(aRow.ge
 t("dore").toString()));

 myTO.setPoints(Integer.parseInt(aRow.get("points").toString())
 );
myTO.setProduct(aRow.get("product").toString());
```

6. Save `UserDAO.java`.
7. Click the Redemption History link on the My Account page of the Earnback application.

8. You should be able to see some data. If you do not see the data, ensure that you are logged in.

Redemption History

| Id | date | points | product |
|---|---|---|---|
| 1 | 10-May-2014 | 50 | charity |
| 2 | 15-May-2014 | 200 | recharger |
| 3 | 30-May-2014 | 200 | recharger |
| 4 | 10-Dec-2014 | 200 | charity |
| 5 | 30-Dec-2014 | 200 | charity |
| 6 | 01-Jan-2015 | 1500 | mp3player |
| 7 | 11-Jan-2015 | 2500 | mp3player |

d. On the **Shop** page of the Earnback Application, when the **All** link is clicked, you want to display all the products available. Complete the `getAllProducts()` method in the `ProductDAO` class.

1. View `getAllProducts()` method in `ProductDAO`.
2. Remove `/*` from before `getAllProducts()` and place it after the method.
3. Obtain a handle to the `products` table and create an empty primary key (as you want to retrieve all the products). Use the `tableIterator()` method to fetch all the rows from the table. Add the following lines of code after the TODO comment.

```
TableAPI myTableAPI = BaseDAO.fetchTableAPI();
Table myTable = myTableAPI.getTable("products");
PrimaryKey primaryKey = myTable.createPrimaryKey();
TableIterator<Row> iter = myTableAPI.tableIterator(primaryKey,
  null, null);
```

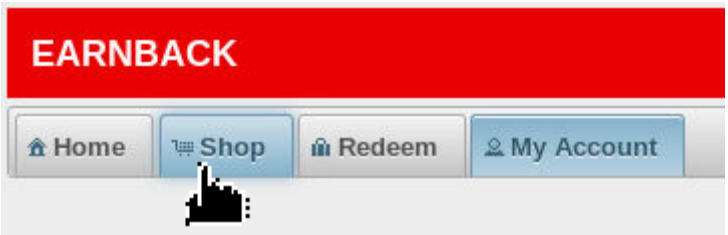
4. Add the following lines of code inside the while statement to set the TO values.

```
myTO.setProductName(aRow.get("prodName").toString());
myTO.setStock(Integer.parseInt(aRow.get("stock").toString()));
myTO.setPrice(Integer.parseInt(aRow.get("price").toString()));
```

5. Add the following import statements.

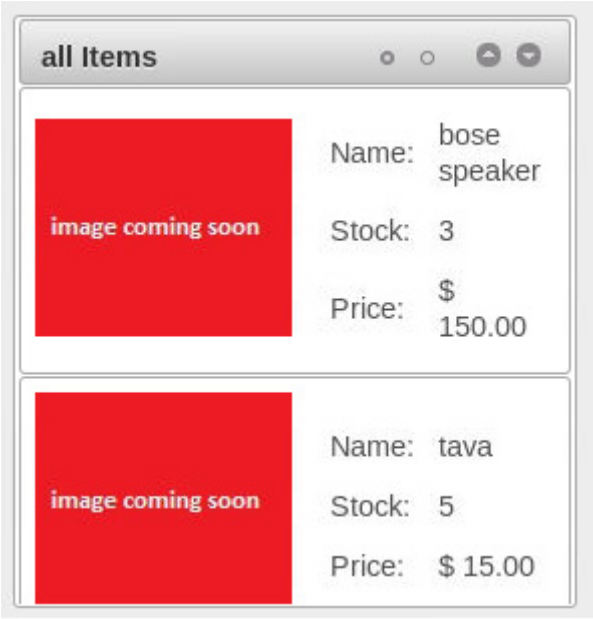
```
import oracle.kv.table.Table;
import oracle.kv.table.TableIterator;
import oracle.kv.table.PrimaryKey;
import oracle.kv.table.Row;
```

6. Save `ProductDAO.java`.
7. View the Shop page in the Earnback application.




8. Click the All Items link, if not already selected.

9. You should see products from all categories listed here (7 products).




e. On the **Shop** page of the Earnback Application, when the **Electronics** link is clicked, you want to display only the electronics available. Complete the `getElectronics()` method in the `ProductDAO` class.

1. View the `getElectronics()` method in `ProductDAO.java`.
2. Remove the comment tags `/*` and `*/` from before and after the `getElectronics()` method.
3. Obtain a handle to the `products` table and create a partial primary key specifying the product category as electronics. Use the `tableIterator()` method to fetch all the matching rows from the products table. Add the following lines of code after the TODO comment.

```
PrimaryKey primaryKey = myTable.createPrimaryKey();
primaryKey.put("prodCat", "electronics");
TableIterator<Row> iter = myTableAPI.tableIterator(primaryKey,
  null, null);
```

4. View the Shop page in the Earnback application.
5. Click the Electronics link.
6. You should see the electronic products listed (4 products).

# Practices for Lesson 9: Using Key-Value APIs

**Chapter 9**

# Practice 9-1: Manipulating Data Stored in Key-Value Model

## Overview

In this practice, you use the Key-Value Model APIs to create, update, and retrieve data.

## Tasks

a. Invoke the data CLI and run the `load.kvs` file from the ~/`labs/dev/scripts` directory. This script loads sample key-value pairs into the `orcl` KVStore. Two types of keys are inserted in this load script: `/emp/<n>` and `/<n>`.

b. Open the `Les09 - Practice` project and open the `KeyValueModel.java` class. Complete the `countEmpRecords()` method to count the number of employee records stored in the KVStore in the key-value model.

c. Complete the `countAllRecords()` method to count the total number of records stored in the KVStore in the key-value model.

d. Create a key with the major-key component as `emp` and minor-key component as `count`. The value for this kv pair will be the result of the `countEmpRecords()` method. Insert this record into the KVStore. Use an API that will create a record that is not already present and will update the record if it exists. Complete the `insertEmpCount()` method.

# Solution 9-1: Manipulating Data Stored in the Key-Value Model

## Overview

In this practice, the code to use the Key-Value Model APIs to create, update, and retrieve data is given.

## Steps

a. Invoke the data CLI and run the `load.kvs` file from the `labs/dev/scripts` directory. This script loads sample key-value pairs into the `orcl` KVStore. Two types of keys are inserted in this load script: `/emp/<n>` and `/<n>`.

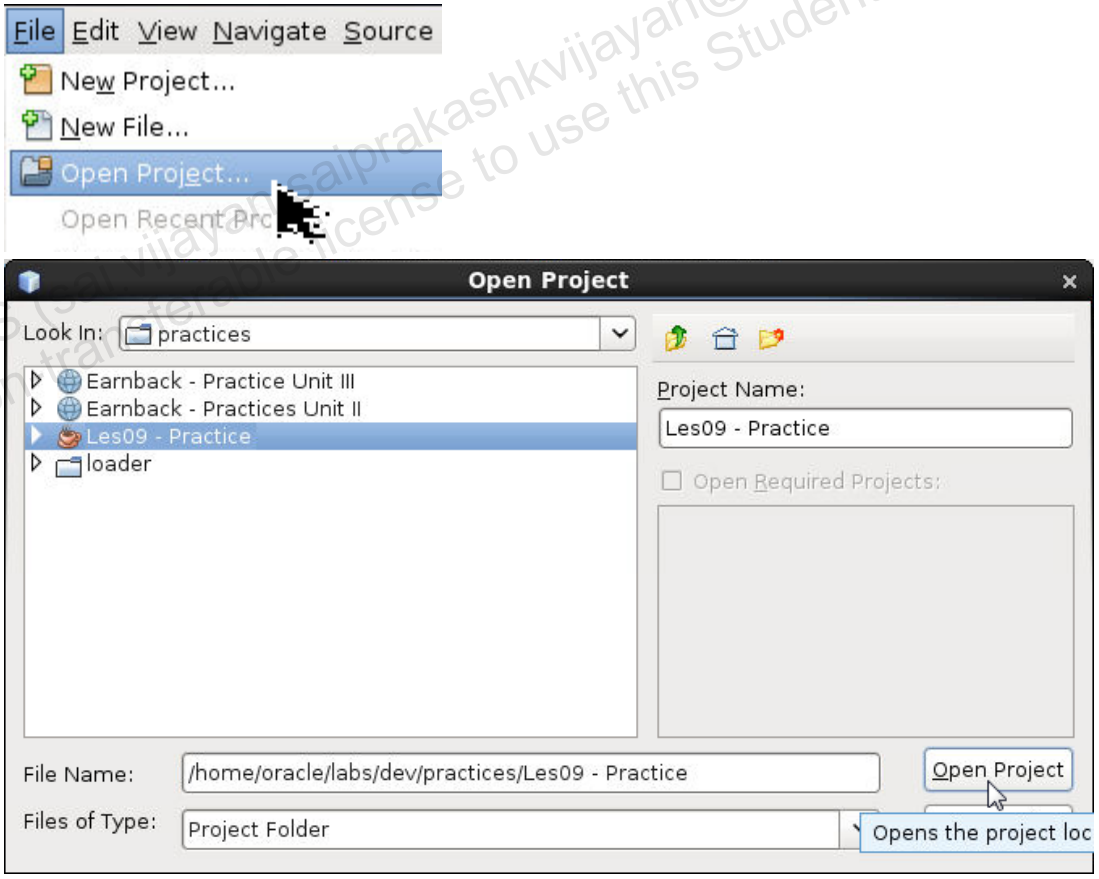
1. In the data CLI, load the ~/`labs/dev/scripts/load.kvs` file.

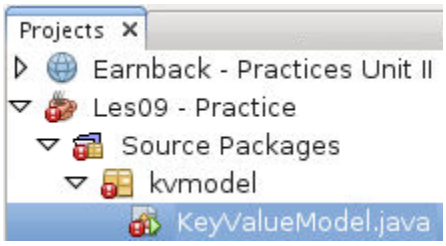
```
load -file /home/oracle/labs/dev/scripts/load.kvs
```

2. The kv pairs are inserted into the `orcl` KVStore.

b. Open the `Les09 - Practice` project and open the `KeyValueModel.java` class. Complete the `countEmpRecords()` method to count the number of employee records stored in the KVStore in the key-value model.

1. View the `countEmpRecords()` method in the `KeyValueModel.java` class of the `Les09 - Practice` project.






2. Create a key with the major key as emp (as you want to retrieve all the employee records). Add the following code after the first TODO comment.

```
Key myKey = Key.createKey("emp");
```

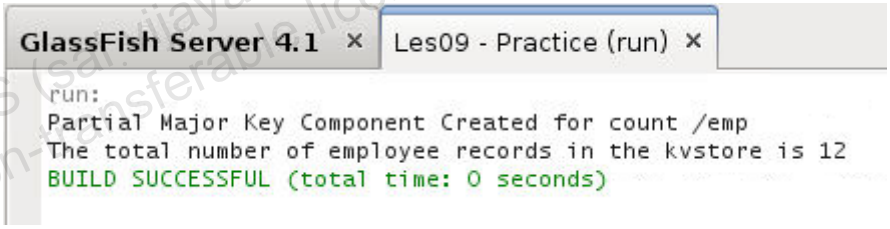
3. Use the `tableIterator` method to fetch all the employee records (as you have created a partial key component). Add the following code after the second TODO comment.

```
Iterator<KeyValueVersion> myrecords =
myStore.storeIterator(Direction.UNORDERED,0, myKey,null,null);
```

4. Enter the following code in the `main()` method to run the count method.

```
KeyValueModel kvm = new KeyValueModel();
KVStore myStore = kvm.orclStore("orcl", "host1", "5000");
Integer countEmp = kvm.countEmpRecords(myStore);
System.out.println("The total number of employee records in the
kvstore is " + countEmp);
```

5. Save `KeyValueModel.java`.
6. Right-click `KeyValueModel.java` and select Run File.
7. View the output in the output pane.




c. Complete the `countAllRecords()` method to count the total number of records stored in the KVStore in the key-value model.

1. View the `countAllRecords()` method in the `KeyValueModel.java` class of the `Les09 - Practice` project.
2. Move the /* comment tag from before the `countAllRecords()` method to after the method.
3. Since you want to count all the records in the store, use the `storeIterator` method and specify the key parameter as null. Add the following code after the TODO comment.

```
Iterator<KeyValueVersion> myrecords =
myStore.storeIterator(Direction.UNORDERED,0, null,null,null);
```

4. Add the following code in the `main()` method to invoke the count method.

```
Integer countAll = kvm.countAllRecords(myStore);
System.out.println("The total number of records in the kvstore is
" + countAll);
```

5. Save `KeyValueModel.java`.
6. Right-click `KeyValueModel.java` and select `Run File`.
7. View the output in the output pane.

```
run:
Partial Major Key Component Created for count /emp
The total number of employee records in the kvstore is 12
The total number of records in the kvstore is 116
BUILD SUCCESSFUL (total time: 2 seconds)
```

d. Create a key with the major-key component as `emp` and minor-key component as `count`. The value for this kv pair will be the result of the `countEmpRecords()` method. Insert this record into the KVStore. Use an API that will create a record that is not already present and will update the record if it exists. Complete the `insertEmpCount()` method.

1. View the insert`EmpCount()` method in the `KeyValueModel.java` class of the `Les09 - Practice` project.
2. Remove the comment tag /* and */ from before and after the method.
3. Create a key with emp as the major-key component and count as the minor-key component. Add the following code after the first TODO comment.

```
Key myKey = Key.createKey("emp", "count");
```

4. Create a value using the count variable. Add the following code after the second TODO comment.

```
String data = count.toString();
Value myValue =Value.createValue(data.getBytes());
```

5. Use the put method to write the key-value pair to the KVStore. Add the following code after the third TODO comment.

```
myStore.put(myKey, myValue);
```

6. Enter the following code in the `main()` method to invoke the insert method.

```
kvm.insertEmpCount(myStore,countEmp);
```

7. Right-click `KeyValueModel.java` and select Run File.
8. View the output in the output pane.

```
run:
Partial Major Key Component Created for count /emp
The total number of employee records in the kvstore is 12
The total number of records in the kvstore is 116
Key Component Created for insert /emp/-/count
Value Created <Value format:NONE bytes: 49 50>
Record inserted
BUILD SUCCESSFUL (total time: 2 seconds)
```

# Practices for Lesson 10: Configuring Consistency

**Chapter 10**

# Practice 10-1: Setting Consistency Policies

## Overview

In this practice, you set consistency parameters.

## Tasks

a. View the default consistency policy for the `orcl` KVStore. Complete the `viewConsistency()` method in the `ConfigDAO` class of the `Earnback – Practice Unit III` project.

b. Modify the read API used in the `validateUser` method in the `UserDAO` class and use the consistency policy that ensures that the operation has the same data on both master and replica nodes.

c. Modify the read API in the `fetchRedemptionHistory` method in the `UserDAO` class and apply the consistency policy that lets the replica node lag behind the master node by two seconds. The entire transaction should complete in four seconds.

## Solution 10-1: Setting Consistency Policies

### Overview

This solution lists the steps to complete the practice tasks.

### Steps

a. View the default consistency policy for the `orcl` KVStore. Complete the `viewConsistency()` method in the `ConfigDAO` class of the `Earnback - Practice Unit III` project.

1. Open the `Earnback - Practice Unit III` project.
2. View the `viewConsistency()` method in the `ConfigDAO` class.
3. KVStore policy details can be obtained from the KVStoreConfig class. Add the following code after the TODO comment to create a KVStoreConfig object and retrieve the consistency policy details.

```
KVStoreConfig myKVStoreConfig = new KVStoreConfig(storeName,
hostPort);
System.out.println(myKVStoreConfig.getConsistency());
```

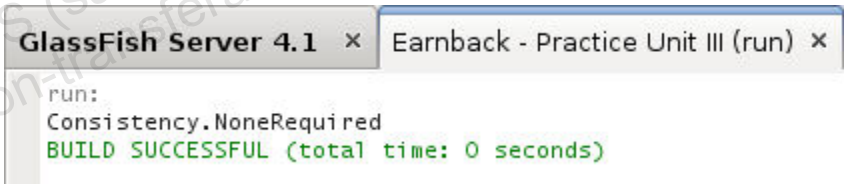
4. Add the following import statements.

```
import oracle.kv.KVStoreConfig;
```

5. Add the following code in the `main()` method to invoke the viewConsistency method.

```
ConfigDAO dao = new ConfigDAO();
dao.viewConsistency();
```

6. Run `ConfigDAO.java`.
7. View the output in the output pane.




b. Modify the read API used in the `validateUser` method in the `UserDAO` class and use the consistency policy that ensures that the operation has the same data on both the master and replica nodes.

1. View the `validateUser()` method in the `UserDAO` class.
2. You need to specify the consistency policy for an operation using the ReadOptions class. Add the following code after the TODO comment.

```
ReadOptions ro = new ReadOptions(Consistency.ABSOLUTE,0,null);
```

3. Add the following import statements.

```
import oracle.kv.Consistency;
import oracle.kv.table.ReadOptions;
```

4. Now, add the `ReadOptions` parameter to the read API.

```
Row myRow = BaseDAO.fetchTableAPI().get(primaryKey, ro);
```

5. Ensure that there are no errors.

c. Modify the read API in the `fetchRedemptionHistory` method in the `UserDAO` class and apply the consistency policy that lets the replica node lag behind the master node by two seconds. The entire transaction should complete in four seconds.

1. View the `fetchRedemptionHistory()` method in the `UserDAO` class.
2. Create the consistency policy and then use ReadOptions to specify the policy for the operation. Add the following code after the TODO comment.

```
Consistency.Time cpolicy = new
  Consistency.Time(2,TimeUnit.SECONDS,4,TimeUnit.SECONDS);
ReadOptions ro = new ReadOptions(cpolicy,0,null);
```

3. Add the `ReadOptions` parameter to the read API.

```
List<Row> myRowSet =
BaseDAO.fetchTableAPI().multiGet(primaryKey, null, ro);
```

4. Ensure that there are no errors.

# Practices for Lesson 11: Configuring Durability

**Chapter 11**

# Practice 11-1: Setting Durability Policies

## Overview

In this practice, you learn how to set a durability policy.

## Tasks

a. View the default durability policy. Complete the `viewDurability()` method in the `ConfigDAO` class.

b. Change the default durability policy to `SYNC` at master, `WRITE_NO_SYNC` at replicas, and `SIMPLE_MAJORITY` acknowledgment.

c. Open the `UserDAO.java` class and modify the `registerUser` method to use the durability policy that ensures that a majority of the replica nodes send acknowledgment to the master node and that the synchronization is performed at the master node as well as replica nodes.

d. Open the `LoadProducts.java` class from the ~/`labs/dev/practices/loader` directory and apply the durability policy that ensures the fastest possible write operation.

# Solution 11-1: Setting Durability Policies

## Overview

This solution lists the steps to complete the practice tasks.

## Steps

a. View the default durability policy. Complete the `viewDurability()` method in the `ConfigDAO` class.

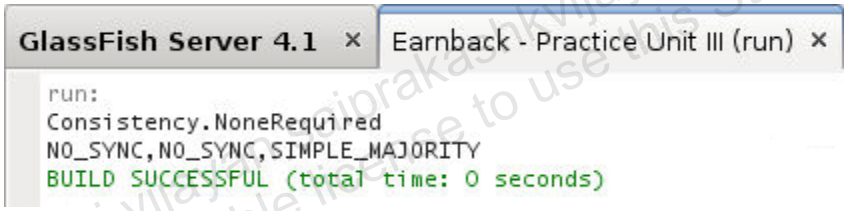
1. View the `viewDurability()` method in the `ConfigDAO` class.
2. Add the following code after the TODO comment to view the KVStore durability policy.

```
KVStoreConfig myKVStoreConfig = new KVStoreConfig(storeName,
hostPort);
System.out.println(myKVStoreConfig.getDurability());
```

3. Add the following code in the `main()` method to invoke the `viewDurability` method.

```
dao.viewDurability();
```

4. Run `ConfigDAO.java`.
5. View the output in the output pane.




b. Change the default durability policy to `SYNC` at master, `WRITE_NO_SYNC` at replicas, and `SIMPLE_MAJORITY` acknowledgment.

1. View the `changeDurability()` method in the `ConfigDAO` class.
2. Create the durability policy and use the setDurability method to specify the new created policy for the entire KVStore. Add the following code after the TODO comment.

```
KVStoreConfig myKVStoreConfig = new KVStoreConfig(storeName,
hostPort);
Durability durability = new
Durability(Durability.SyncPolicy.SYNC,
Durability.SyncPolicy.WRITE_NO_SYNC,Durability.ReplicaAckPolic
y.SIMPLE_MAJORITY);
myKVStoreConfig.setDurability(durability);
```

3. Add the following import statement.

```
import oracle.kv.Durability;
```

4. Add the following code in the `main()` method to invoke the `changeDurability` method.

```
dao.changeDurability();
```

5. Run `ConfigDAO.java`.
6. View the output in the output pane.

```
run:
Consistency.NoneRequired
NO_SYNC,NO_SYNC,SIMPLE_MAJORITY
Default Durability Changed.
NO_SYNC,NO_SYNC,SIMPLE_MAJORITY
BUILD SUCCESSFUL (total time: 7 seconds)
```

c. Open the `UserDAO.java` class and modify the `registerUser` method to use the durability policy that ensures that a majority of the replica nodes send acknowledgment to the master node and that the synchronization is performed at the master node as well as at the replica nodes.

1. View the `registerUser()` method in the `UserDAO` class.
2. Create the durability policy and use the WriteOptions class to specify the policy for an operation. Add the following code after the TODO comment.

```
Durability durability = new
Durability(Durability.SyncPolicy.SYNC,
Durability.SyncPolicy.SYNC,Durability.ReplicaAckPolicy.ALL);
WriteOptions wo = new WriteOptions(durability,0,null);
```

3. Add the following import statements.

```
import oracle.kv.Durability;
import oracle.kv.table.WriteOptions;
```

4. Add the `ReadOptions` parameter to the read API.

```
Version version = BaseDAO.fetchTableAPI().putIfAbsent(row,
null, wo);
```

5. Ensure that there are no errors.

d. Open the `LoadProducts.java` class from the ~/`labs/dev/practices/loader` directory and apply the durability policy that ensures the fastest possible write operation.

1. Open the `LoadProducts.java` file.
2. Create the durability policy and use WriteOptions to specify the policy for an operation. Add the following code after the TODO comment for lesson 11.

```
Durability durability = new
Durability(Durability.SyncPolicy.NO_SYNC,
Durability.SyncPolicy.NO_SYNC,Durability.ReplicaAckPolicy.NONE);
WriteOptions wo = new WriteOptions(durability,0,null);
```

3. Ensure the following import statements are included.

```
import oracle.kv.Durability;
import oracle.kv.table.WriteOptions;
```

4. Add the `ReadOptions` parameter to the read API.

```
myStore.getTableAPI().put(myrow,null,wo);
```

5. Ensure that there are no errors.

# Practices for Lesson 12: Creating Transactions

**Chapter 12**

# Practice 12-1: Creating and Executing Transactions

## Overview

In this practice, you use the available methods to create and execute transactions.

## Tasks

a. In the `AccountsDAO.java` class, the code to insert rows into the accounts table is partially completed. Three row objects are created using the userID user1. Complete the code to execute the insert operations as a transaction.

# Solution 12-1: Creating and Executing Transactions

## Overview

In this practice, the code to create and execute transactions is shown.

## Steps

a. In the `AccountsDAO.java` class, the code to insert rows into the accounts table is partially completed. Three row objects are created using the userID user01. Complete the code to execute the insert operations as a transaction.

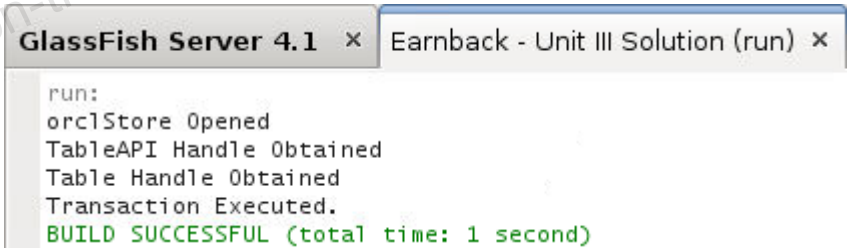
1. Open `AccountsDAO.java` and review the existing code.
2. Obtain a `TableOperationFactory` object as you will need it to invoke the write methods. Create a list of type `TableOperation` and add the operations to this list. Use the `execute` method to run the transaction. Add the following code after the TODO comment.

```
TableOperationFactory tof =
BaseDAO.fetchTableAPI().getTableOperationFactory();
List<TableOperation> opList = new ArrayList<TableOperation>();
opList.add(tof.createPut(row1, null, true));
opList.add(tof.createPut(row2, null, true));
opList.add(tof.createPut(row3, null, true));
BaseDAO.fetchTableAPI().execute(opList,null);
System.out.println("Transaction Executed.");
```

3. Add the following import statements.

```
import oracle.kv.table.TableOperation;
import oracle.kv.table.TableOperationFactory;
```

4. Run `AccountsDAO.java`.
5. View the output.



# Practices for Lesson 13: Handling Large Objects

**Chapter 13**

# Practice 13-1: Creating Large Objects

## Overview

In this practice, you use the available methods to create and retrieve large objects.

## Tasks

a. Open `LOB.java` and review the code.

b. In the `insertLOB()` method, complete the code by creating a key `image` for the LOB. The LOB suffix is the default suffix `lob`. Use the appropriate API to insert the LOB into the KVStore. Call the `insertLOB()` method form the `main()` method.

c. Complete the `getLOB()` method to fetch the LOB from the KVStore. In the `main()` method, create the key to retrieve the LOB image and pass it to the `getLOB()` method.

## Solution 13-1: Creating Large Objects

### Overview

In this solution, the code to create and retrieve large objects is provided.

### Steps

a. Open `LOB.java` and review the code.

b. In the `insertLOB()` method, complete the code by creating a key `image` for the LOB. The LOB suffix is the default suffix `lob`. Use the appropriate API to insert the LOB into the KVStore. Call the `insertLOB()` method form the `main()` method.

1. Create the key for the LOB. Note that the key should end with the word “lob”. Add the following code to create a key.

```
Key key = Key.createKey("image.lob");
```

2. Add the following code to write the LOB into the KVStore.

```
Version vr = BaseDAO.fetchKVStore().putLOB(key, fis, null, 0,
null);
```

3. Run `LOB.java`.
4. View the output message.

c. Complete the `getLOB()` method to fetch the LOB from the KVStore. In the `main()` method, create the key to retrieve the LOB image and pass it to the `getLOB()` method.

1. Review the `getLOB()` method and remove the comment tags before and after the method.
2. Add the following code in the `getLOB()` method to fetch the LOB.

```
InputStreamVersion isv = BaseDAO.fetchKVStore().getLOB(key,
null, 0, null);
```

3. Add the following code in the `main()` method to create the key.

```
Key key = Key.createKey("image.lob");
```

4. Remove the comment tags before and after the code to fetch the LOB and count the number of bytes.
5. Run `LOB.java`.
6. View the output message.

# Practices for Lesson 14: Accessing a Secure Store

**Chapter 14**

# Practices for Lesson 14

This is a hands-on practice that covers accessing a secure store.

# Practices for Lesson 15: Handling Exceptions

**Chapter 15**

# Practices for Lesson 15

There are no practices for this lesson.