



Big Data Hands on Workshop

Lab Guide

ORACLE®
BIG DATA APPLIANCE

Version 2.1.9/2014

Exercises 1.3

TABLE OF CONTENTS

Big Data Hands on Workshop Lab Guide	i
Revision History	5
1. Introduction to Big Data	7
2. Introduction to the workshop.....	10
2.1 Workshop Environment.....	10
2.2 Workshop Tips and Tricks.....	10
2.3 Workshop Story.....	13
3. Hadoop Word Count	15
3.1 Introduction to Hadoop.....	15
3.2 Overview of Hands on Exercise	16
3.3 Word Count	16
3.4 Summary	31
4. Oracle NoSQL Database	32
4.1 Introduction to NoSQL.....	32
4.2 Overview of Hands on Exercise	32
4.3 Insert and retrieve data from the NoSQL Database.....	33
4.4 noSQL KV Client.....	47
4.5 Oracle External Tables pointing to NoSQL data	48
4.6 NoSQL and our Transactional Data	63
4.7 Summary	83
5. Pig Exercise	84
5.1 Introduction to Pig	84
5.2 Overview of Hands on Exercise	84
5.3 Working with PIG.....	85
5.4 Summary	105
6. Hive and Impala.....	106
6.1 Introduction to Hive	106
6.2 Overview of Hands On Exercise	106
6.3 Queries with Hive	107
6.4 Hive Summary.....	134
6.5 Impala Concepts and Architecture	135
6.6 Impala Exercises	138
6.7 Impala Summary	144
7. Working with the Oracle Loader for Hadoop.....	145
8.1 Introduction to the Oracle Loader for Hadoop.....	145

8.2 Overview of Hands on Exercise	146
8.3 Loading HDFS data into the Oracle Database.....	146
8.4 Summary	160
8. Working with the Oracle SQL Connector for HDFS.....	161
9.1 Introduction to the Oracle SQL Connector for HDFS.....	161
9.2 Overview of Hands on Exercise	161
9.3 Configuring External Tables stored in HDFS	162
9.4 Summary	174
9. Oracle ODI and Hadoop	176
9.1 Introduction to ODI Application Adapter for Hadoop	176
9.2 Overview of Hands on Exercise	177
9.3 Setup and Reverse Engineering in ODI	178
9.4 Using ODI to move data from Hive to Hive	186
9.5 Summary	208
10. Oracle Big Data SQL	209
10.1 Introduction to Oracle Big Data SQL.....	209
10.2 Big Data SQL Exercises.....	210
10.3 Summary	215
11. XQuery for Hadoop (OXH).....	216
11.1 Introduction to Oracle OXH XQuery for Hadoop	216
11.2 XML and XQuery primer	218
11.3 Overview OXH XQuery hands on exercise	220
11.4 Exercises.....	221
11.5 Summary	233
12. Programming with R	234
12.1 Introduction to Enterprise R	234
12.2 Overview of Hands on Exercise	234
12.3 Working with R	235
12.4 Summary	283
13. Working with Cloudera Search (Solr)	284
13.1 Introducing Cloudera Search	284
13.2 Cloudera Search Features	285
13.3 Cloudera Search Tasks and Processes.....	286
13.4 Exercises.....	288
13.5 Summary	294
14. Working with Spark (word count revisited)	296
14.1 Introducing Spark	296
Apache Spark	296
Fast Analytics and Stream Processing	296

14.2 Exercises.....	297
14.3 Summary	301
A. Skip of ODI Exercise	304
A.1 Running the cheat script for the ODI exercise	304

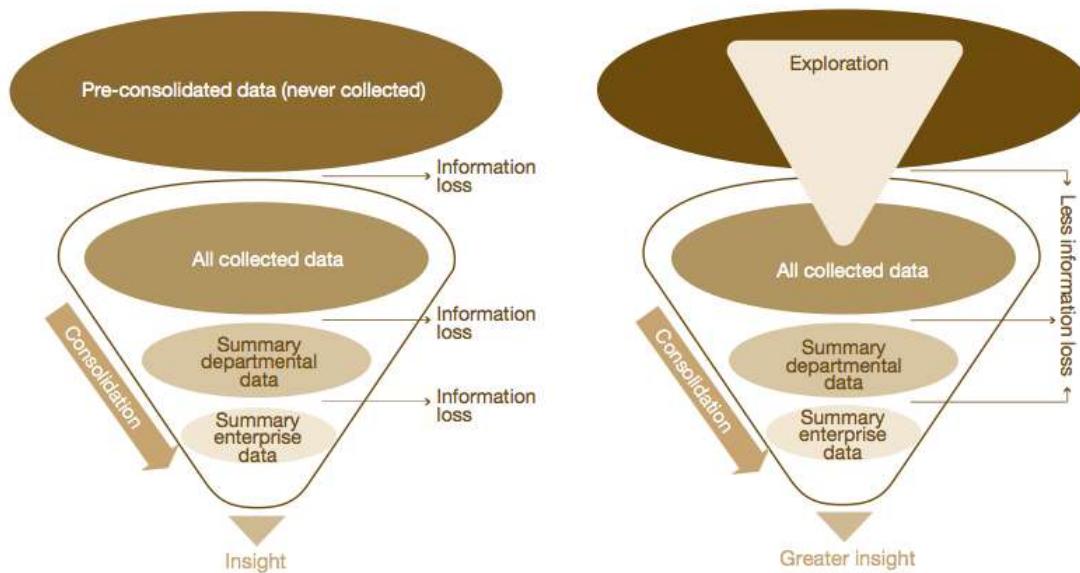
Revision History

1. 2-6-2014 BN
 - a. Added a correction for a missing string in the OLH exercises.
2. 2-9-2014 BN
 - a. Removed orch.connect from R exercise – was extraneous and causing sporadic issues.
3. 5-15-2014 BN
 - a. Made generic for BDA 2.x and 3.x BigDataLite and BDA environments.

1. INTRODUCTION TO BIG DATA

Big data is not just about managing petabytes of data. It is also about managing large numbers of complex unstructured data streams which contain valuable data points. However, which data points are the most valuable depends on who is doing the analysis and when they are doing the analysis. Typical big data applications include: smart grid meters that monitor electricity usage in homes, sensors that track and manage the progress of goods in transit, analysis of medical treatments and drugs that are used, analysis of CT scans etc. What links these big data applications is the need to track millions of events per second, and to respond in real time. Utility companies will need to detect an uptick in consumption as soon as possible, so they can bring supplementary energy sources online quickly. Probably the fastest growing area relates to location data being collected from mobile always-on devices. If retailers are to capitalize on their customers' location data, they must be able to respond as soon as they step through the door.

In the conventional model of business intelligence and analytics, data is cleansed, cross-checked and processed before it is analyzed, and often only a sample of the data is used in the actual analysis. This is possible because the kind of data that is being analyzed - sales figures or stock counts, for example – can easily be arranged in a pre-ordained database schema, and because BI tools are often used simply to create periodic reports.

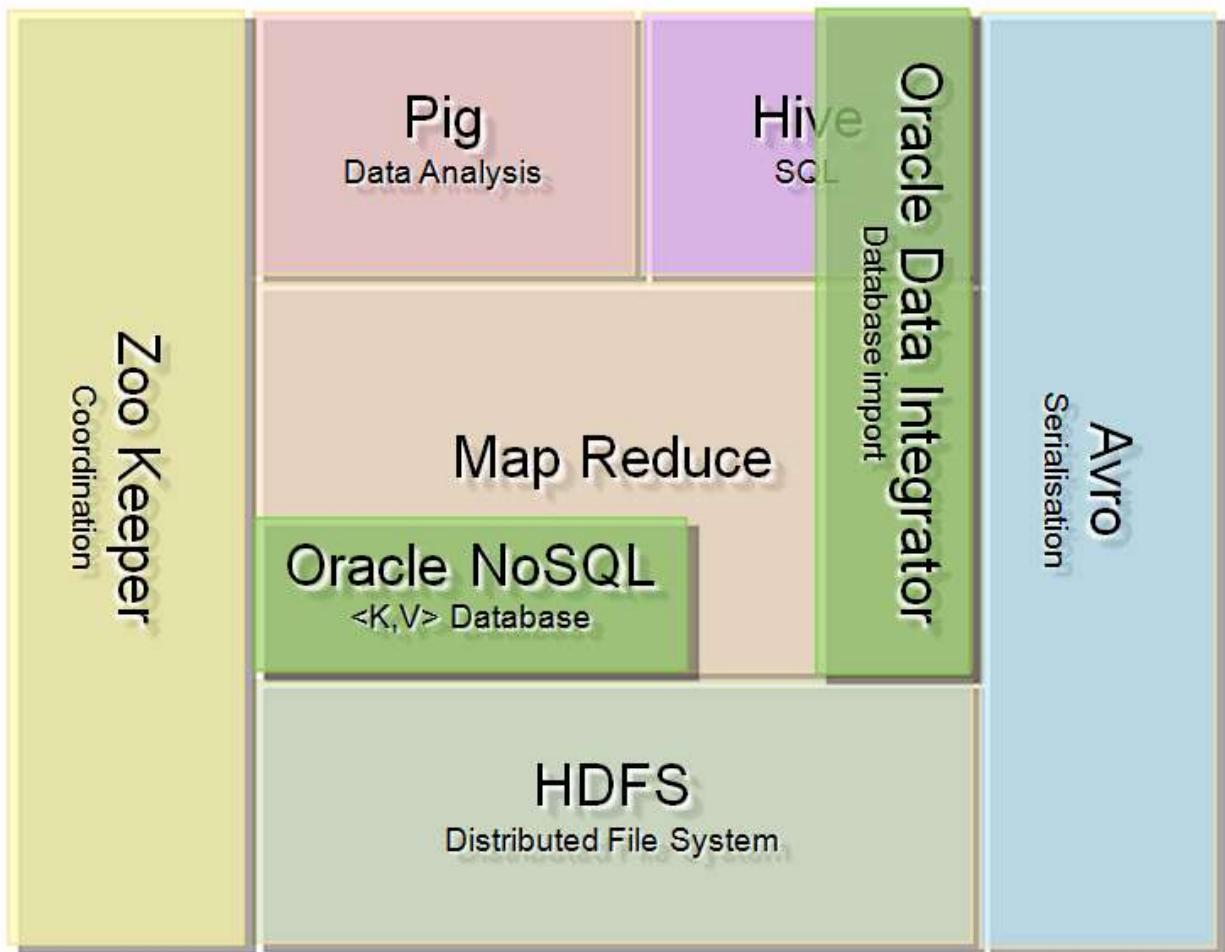


At the center of the big data movement is an open source software framework called Hadoop. Hadoop has become the technology of choice to support applications that in turn support petabyte-sized analytics utilizing large numbers of computing nodes. The Hadoop system consists of three projects: Hadoop Common, a utility layer that provides access to the Hadoop Distributed File System and Hadoop subprojects. HDFS acts as the data storage platform for the Hadoop framework and can scale to a massive size when distributed over numerous computing nodes.

Hadoop Map/Reduce is a framework for processing data sets across clusters of Hadoop nodes. The Map and Reduce process splits the work by first mapping the input across the control nodes of the cluster, then splitting the workload into even smaller data sets and distributing it further throughout the computing cluster. This allows it to leverage massively parallel processing, a computing advantage that technology has introduced to modern system architectures. With MPP, Hadoop can run on inexpensive commodity servers, dramatically reducing the upfront capital costs traditionally required to build out a massive system.

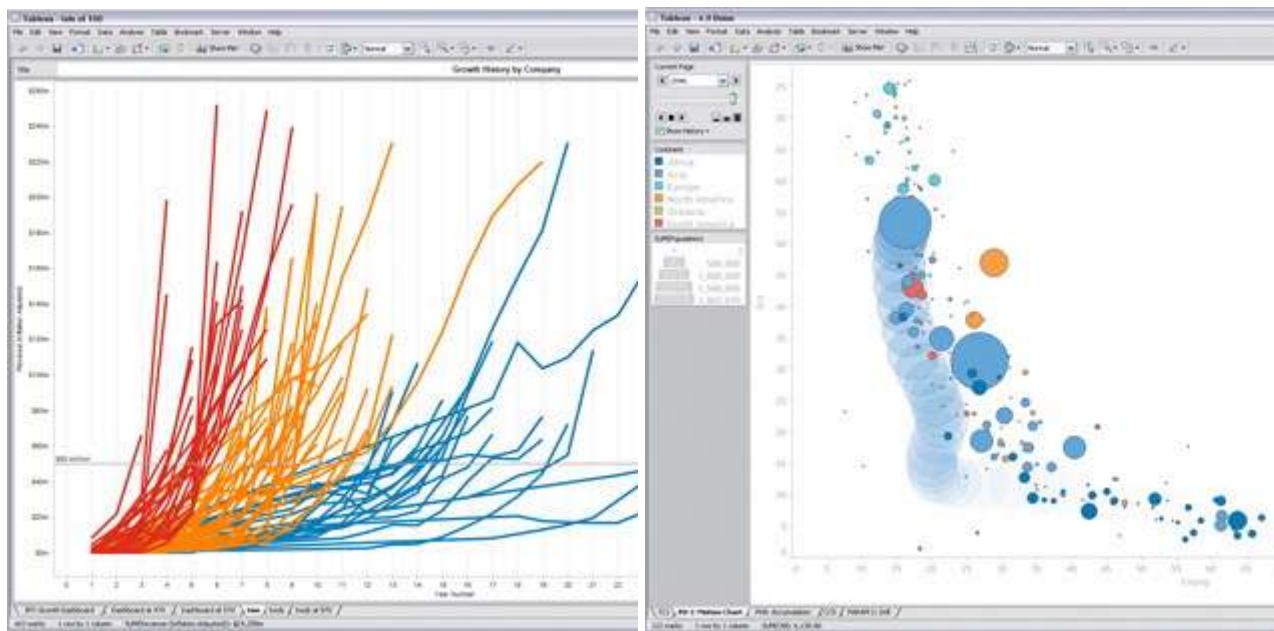
As the nodes "return" their answers, the Reduce function collects and combines the information to deliver a final result.

To extend the basic Hadoop ecosystem capabilities a number of new open source projects have added functionality to the environment. Oracle's Hadoop ecosystem will look something like this:



- Avro is a data serialization system that converts data into a fast, compact binary data format. When Avro data is stored in a file, its schema is stored with it
- Oracle NoSQL is a scalable, Key Value oriented distributed database extended for Oracle's Berkeley Database distributed storage system. Oracle NoSQL is well-suited for real-time data analysis
- Hive is a data warehouse infrastructure that provides ad hoc query and data summarization for Hadoop-supported data. Hive utilizes a SQL-like query language called HiveQL. HiveQL can also be used by programmers to execute custom Map/Reduce jobs
- Pig is a high-level programming language and execution framework for parallel computation. Pig works within the Hadoop and Map/Reduce frameworks
- ZooKeeper provides coordination, configuration and group services for distributed applications working over the Hadoop stack
- Oracle Data Integrator is an enterprise class ETL tool which is Oracle's static tool for moving data into Hadoop and from there into the Oracle Database

Data exploration of Big Data result sets requires displaying millions or billions of data points to uncover hidden patterns or records of interest as shown below:



Many vendors are talking about Big Data in terms of managing petabytes of data. In reality the issue of big data is much bigger and Oracle's aim is to focus on providing a big data platform which provides the following:

- **Deep Analytics** – a fully parallel, extensive and extensible toolbox full of advanced and novel statistical and data mining capabilities
- **High Agility** – the ability to create temporary analytics environments in an end-user driven, yet secure and scalable environment to deliver new and novel insights to the operational business
- **Massive Scalability** – the ability to scale analytics and sandboxes to previously unknown scales while leveraging previously untapped data potential
- **Low Latency** – the ability to instantly act based on these advanced analytics in your operational, production environment

2. INTRODUCTION TO THE WORKSHOP

This workshop is designed with two distinct purposes in mind. We would like through the course of this workshop to introduce you to the ecosystem that is Big Data, as well as to show how these tools can be used together to solve real business problems. Most of the sections in this workshop will have two different types of exercises. The first set of exercises show the basic functionality of the tool, while the last exercise will show how the tool can be used to drive our business case forward.

2.1 Workshop Environment

All of the exercises in this workshop can be executed in the BigDataLite virtual environment.

Important : Before starting on the Workshop exercises you will need to have a running Hadoop Cluster with Hiveserver2, SOLR, Impala enabled and running, also you will need the Oracle database started as well. Please refer to the VM "Start Here" page for instructions on starting DB and services for this workshop.

The following table has the users and passwords for the applications above. Whenever a password is required you will be told explicitly during the exercise.

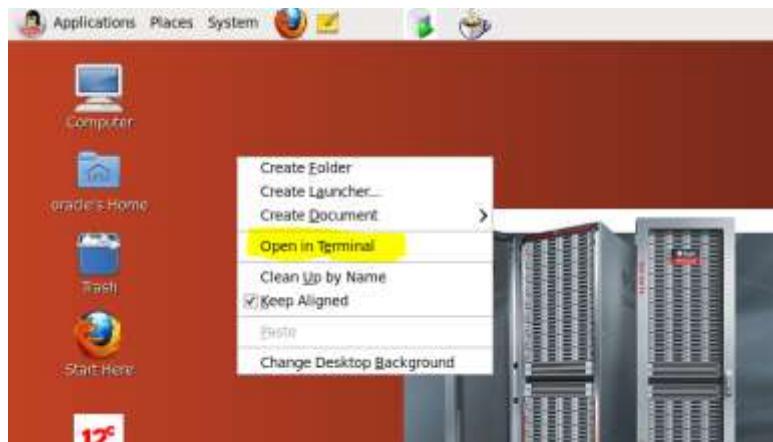
Application	User	Password
Operating System	oracle	welcome1
Operating System	root	welcome1
HDFS	hdfs	hdfs
Oracle Database	sys	welcome1
Oracle Database	BDA	welcome1
MySQL Database	root	welcome1
Cloudera Manager	admin	welcome1
RStudio	oracle	welcome1

2.2 Workshop Tips and Tricks

There are several features built into this workshop which will make getting through the workshop easier detailed below. This section is completely optional and does not affect the run-through of the workshop.

1. You can use the <TAB> button for auto complete in the Linux terminal.

Open a terminal – Right mouse click on the desktop and click on the “Open in Terminal” menu item.

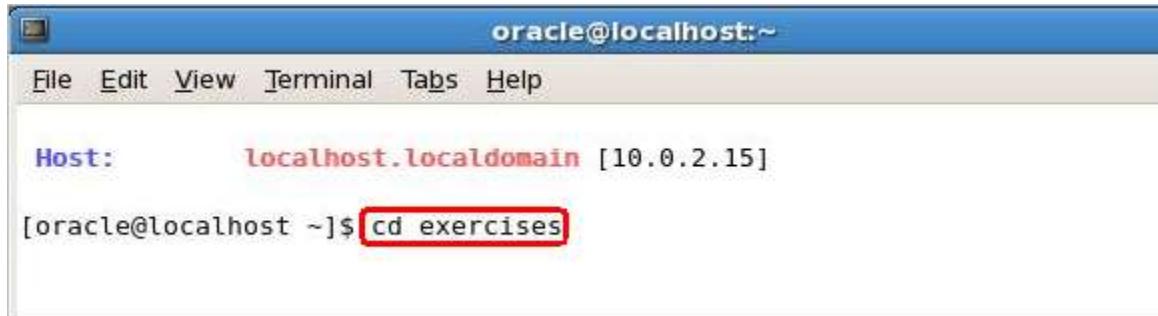


A new terminal window appears.

A screenshot of a terminal window titled "oracle@localhost:~". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The host information is shown as "Host: localhost.localdomain [10.0.2.15]". The prompt "[oracle@localhost ~]\$" is visible at the bottom of the window.

Type `cd exe <TAB>`

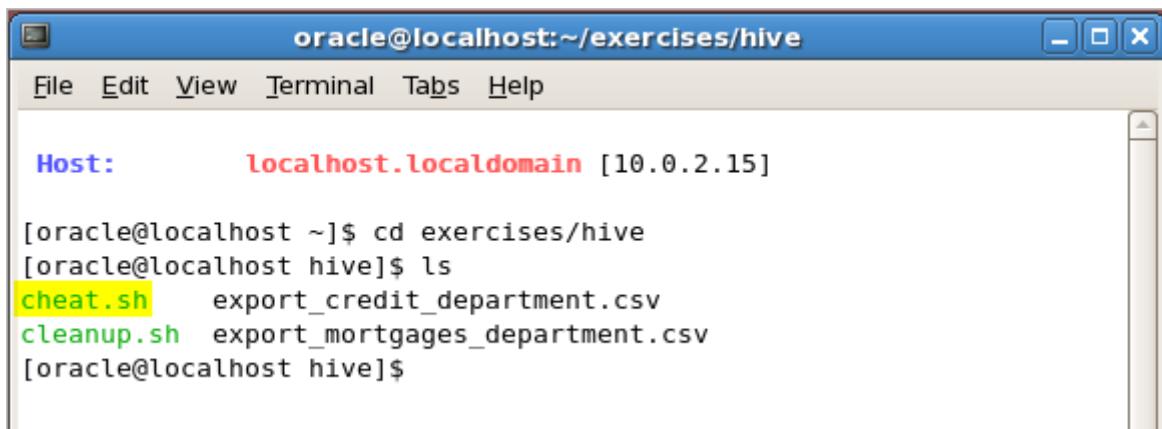
In the terminal, the word “exercises” should be auto-completed.



A screenshot of a terminal window titled "oracle@localhost:~". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". Below the menu is a status line showing "Host: localhost.localdomain [10.0.2.15]". The main terminal area shows the command "[oracle@localhost ~]\$ cd exercises" where "exercises" is highlighted with a red box.

This auto complete can be used any time you are typing in file or directory names, making the process easier and reducing the chance of errors.

2. In each exercises folder there is a file called cheat.sh.

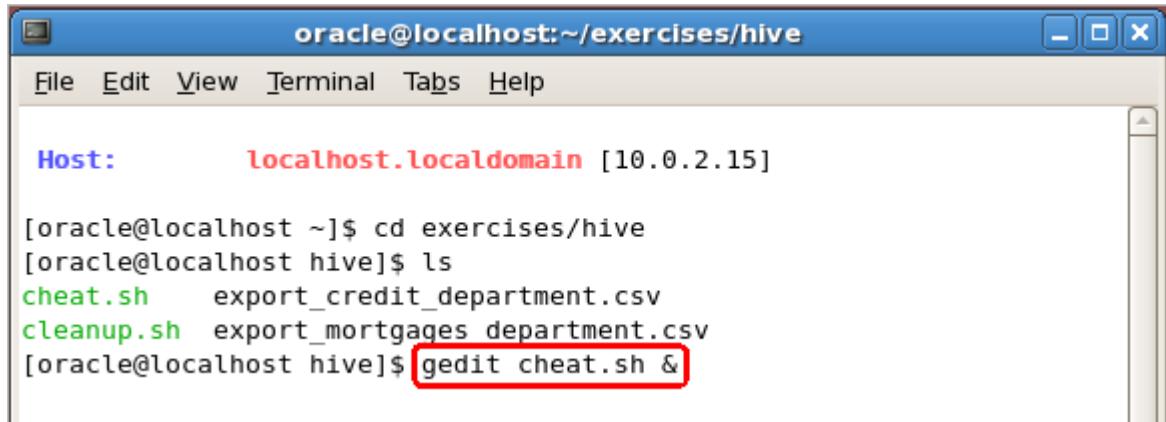


A screenshot of a terminal window titled "oracle@localhost:~/exercises/hive". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". Below the menu is a status line showing "Host: localhost.localdomain [10.0.2.15]". The main terminal area shows the command "[oracle@localhost ~]\$ cd exercises/hive" followed by "[oracle@localhost hive]\$ ls". The output of the "ls" command shows files: "cheat.sh", "export_credit_department.csv", and "cleanup.sh" (with "cleanup.sh" highlighted in yellow). The command "[oracle@localhost hive]\$" is also visible at the bottom.

There are two different ways in which this script can be used.

- a. All of the exercises are required to be completed sequentially. If you would like to skip an exercise you can just run this script and all of the actions required by the exercise will be performed by the script.
- b. In exercises where a lot of typing is required you can open the cheat.sh script and use it as a source for copy and paste into the terminal. This will avoid typos and help get through the exercises quicker if you would like.

To open the script type gedit <script name> & and a new window will pop up with the script.



```
oracle@localhost:~/exercises/hive
File Edit View Terminal Tabs Help

Host:      localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd exercises/hive
[oracle@localhost hive]$ ls
cheat.sh    export_credit_department.csv
cleanup.sh   export_mortgages_department.csv
[oracle@localhost hive]$ gedit cheat.sh &
```

Note: The R exercise also contains a file called cheat.r. Use that file to copy and paste R code.

3. In each exercises folder there is a file called cleanup.sh. You can run this script to clean up the results of the exercise in order to run it again if you need to.

2.3 Workshop Story

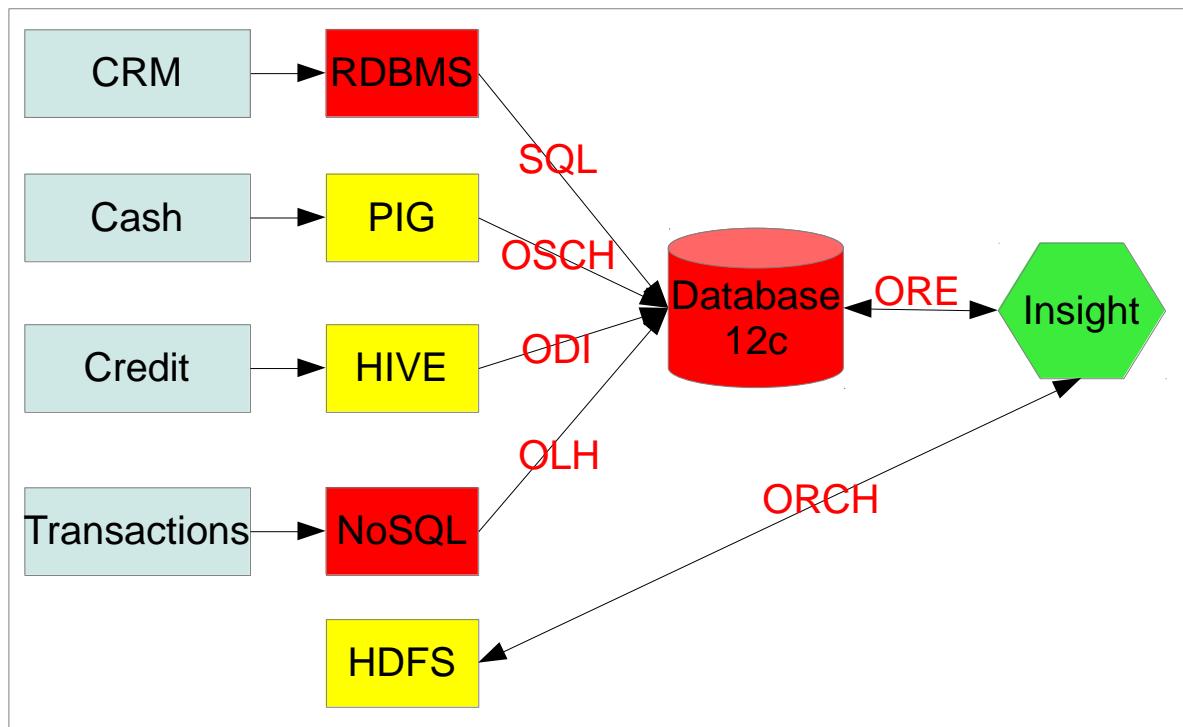
In this workshop we are going to explore the Big Data Process as it pertains to an insurance business case. Though this is a particular story, the process flow, exercises, methodologies and tools are universal. We will see how the flow of Acquire -> Organize -> Analyze -> Decide can apply to our use case and how the Oracle set of products can assist in achieving our goal.

Problem Statement: We are a banking & insurance company and we need to target a marketing campaign for car insurance. We will not use a scatter-gun approach and will be targeting existing customers who already have a history with our company. We need to find those customers who we are sure will purchase insurance and focus our sales effort on them. We will therefore study a sample of customers who have already decided whether to buy car insurance from our company or not and we will try to create a specific profile for the customers who bought insurance. We will then target the rest of the customers, who haven't yet decided whether or not to acquire our insurance services, based on the customer profiles we identified through our analysis.

The Methodology: The key to insight is data, as such our first step will be to find what data we can aggregate, and from where, in order to enable the analysis of our customers.

In this workshop we will:

- 1) Do a sentiment analysis
- 2) Find data related to banking transactions in our production NoSQL database and aggregate it
- 3) Find data in our cash accounts system and aggregate it
- 4) Find data from our internal credit department and aggregate it
- 5) Centralize and merge all aggregated data into our Oracle 12c Database, to create a 360 degree view of the customer
- 6) Analyze the merged data to figure out what our target audience for our marketing campaign is

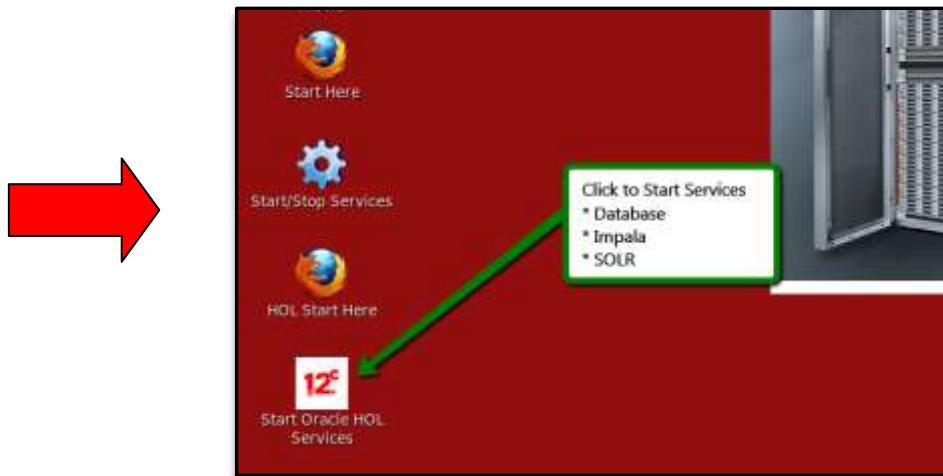


So without further ado let's get started.

3. HADOOP WORD COUNT

3.1 Introduction to Hadoop

Important: Before starting on the Workshop exercises you will need to have a running Hadoop Cluster with HiveServer2, SOLR, Impala enabled and running, also you will need the Oracle database started as well. Please see below for instructions on starting DB and services for this workshop.



Map/Reduce

Map/Reduce is a programming paradigm that expresses a large distributed computation as a sequence of distributed operations on data sets of key/value pairs. The Hadoop Map/Reduce framework harnesses a cluster of machines and executes user defined Map/Reduce jobs across the nodes in the cluster. A Map/Reduce computation has two phases, a map phase and a reduce phase. The input to the computation is a data set of key/value pairs.

In the map phase, the framework splits the input data set into a large number of fragments and assigns each fragment to a map task. The framework also distributes the many map tasks across the cluster of nodes on which it operates. Each map task consumes key/value pairs from its assigned fragment and produces a set of intermediate key/value pairs. For each input key/value pair (K,V), the map task invokes a user defined map function that transmutes the input into a different key/value pair (K',V').

Following the map phase the framework sorts the intermediate data set by key and produces a set of (K',V'*) tuples so that all the values associated with a particular key appear together. It also partitions the set of tuples into a number of fragments equal to the number of reduce tasks.

In the reduce phase, each reduce task consumes the fragment of (K',V'*) tuples assigned to it. For each such tuple it invokes a user-defined reduce function that transmutes the tuple into an output key/value pair (K,V). Once again, the framework distributes the many reduce tasks across the cluster of nodes and deals with shipping the appropriate fragment of intermediate data to each reduce task.

Tasks in each phase are executed in a fault-tolerant manner, if node(s) fail in the middle of a computation the tasks assigned to them are re-distributed among the remaining nodes. Having many map and reduce tasks enables good load balancing and allows failed tasks to be re-run with small runtime overhead.

Architecture

The Hadoop Map/Reduce framework has a master/slave architecture. It has a single master server or jobtracker and several slave servers or tasktrackers, one per node in the cluster. The jobtracker is the point of interaction between users and the framework. Users submit map/reduce jobs to the jobtracker, which puts them in a queue of pending jobs and executes them on a first-come/first-served basis. The jobtracker manages the assignment of map and reduce tasks to the tasktrackers. The tasktrackers execute tasks upon instruction from the jobtracker and also handle data motion between the map and reduce phases.

Hadoop DFS

Hadoop's Distributed File System is designed to reliably store very large files across machines in a large cluster. It is inspired by the Google File System. Hadoop DFS stores each file as a sequence of blocks, all blocks in a file except the last block are the same size. Blocks belonging to a file are replicated for fault tolerance. The block size and replication factor are configurable per file. Files in HDFS are "write once" and have strictly one writer at any time.

Architecture

Like Hadoop Map/Reduce, HDFS follows a master/slave architecture. An HDFS installation consists of a single Namenode, a master server that manages the filesystem namespace and regulates access to files by clients. In addition, there are a number of Datanodes, one per node in the cluster, which manage storage attached to the nodes that they run on. The Namenode makes filesystem namespace operations like opening, closing, renaming etc. of files and directories available via an RPC interface. It also determines the mapping of blocks to Datanodes. The Datanodes are responsible for serving read and write requests from filesystem clients, they also perform block creation, deletion, and replication upon instruction from the Namenode.

3.2 Overview of Hands on Exercise

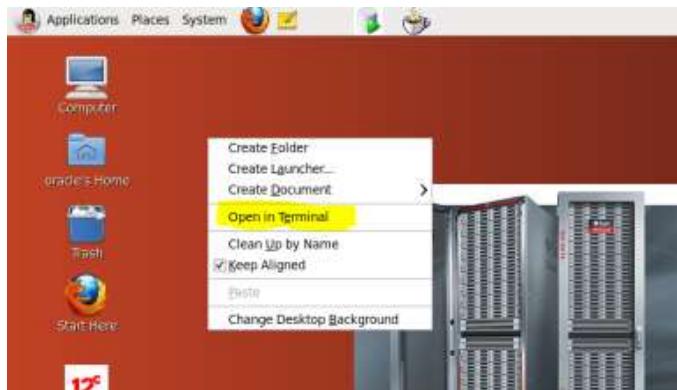
To get an understanding of what is involved in running a Hadoop Job and what are all of the steps one must undertake we will embark on setting up and running a "Hello World" type exercise on our Hadoop Cluster.

In this exercise you will:

- 1) Compile a Java Word Count written to run on a Hadoop Cluster
- 2) Upload the files on which to run the word count into HDFS
- 3) Run Word Count
- 4) View the Results

3.3 Word Count

1. All of the setup and execution for the Work Count exercise can be done from the terminal, hence to start out this first exercise please open the terminal by right clicking on the desktop and click on "Open in Terminal" menu item.



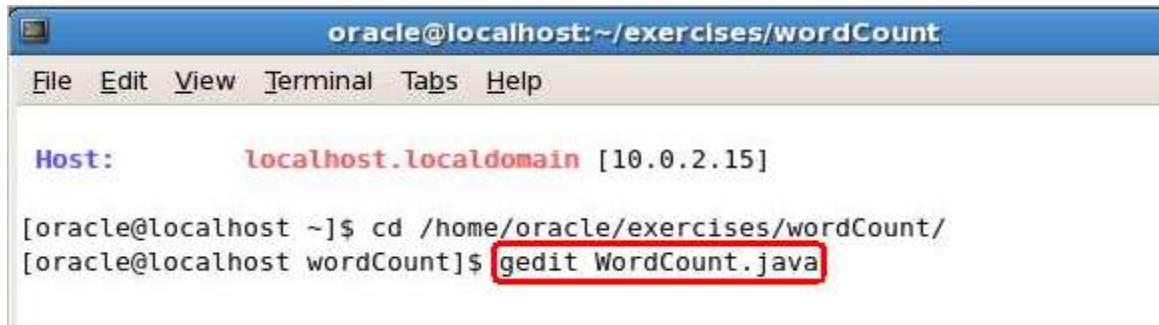
2. To get into the folder where the scripts for the first exercise are, type in the terminal:

```
cd /home/oracle/exercises/wordCount  
Then press Enter
```

A screenshot of a terminal window titled "oracle@localhost:~". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". Below the menu bar, it says "Host: localhost.localdomain [10.0.2.15]". In the terminal window, the command "[oracle@localhost ~]\$ cd /home/oracle/exercises/wordCount/" is displayed, with the entire command highlighted by a red box.

3. Let's look at the Java code which will run word count on a Hadoop cluster. Type in the terminal:

```
gedit WordCount.java  
Then press Enter
```

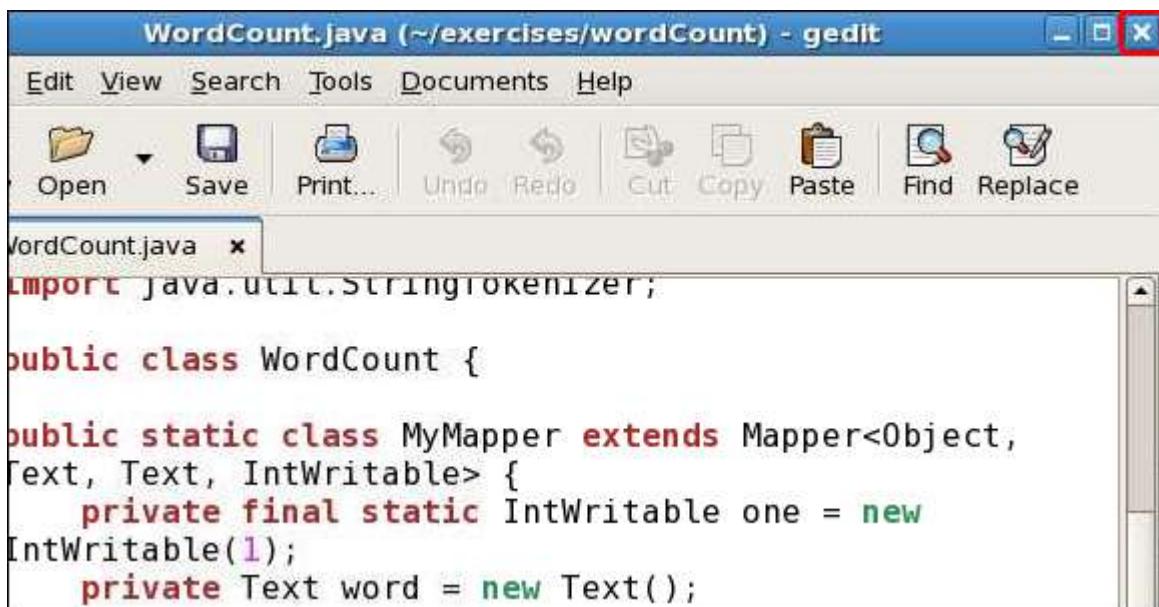


```
oracle@localhost:~/exercises/wordCount  
File Edit View Terminal Tabs Help  
  
Host: localhost.localdomain [10.0.2.15]  
  
[oracle@localhost ~]$ cd /home/oracle/exercises/wordCount/  
[oracle@localhost wordCount]$ gedit WordCount.java
```

A new window will open with the Java code for word count. We would like you to look at line 14 and 28 of the code. You can see there how the Mapper and Reducer Interfaces are being implemented.

```
12 public class WordCount {  
13  
14     public static class MyMapper extends Mapper<Object,  
           Text, Text, IntWritable> {  
15         private final static IntWritable one = new  
           IntWritable(1);  
16         private Text word = new Text();  
17  
18         public void map(Object key, Text value, Context  
           context)  
19             throws IOException, InterruptedException {  
20             StringTokenizer itr = new StringTokenizer  
               (value.toString());  
21             while (itr.hasMoreTokens()) {  
22                 word.set(itr.nextToken());  
23                 context.write(word, one);  
24             }  
25         }  
26     }  
27  
28     public static class MyReducer extends Reducer<Text,  
           IntWritable, Text, IntWritable> {  
29         public void reduce(Text key, Iterable<IntWritable>  
           values, Context context)
```

- When you are done evaluating the code you can click on the X in the right upper corner of the screen to close the window.



A screenshot of the gedit text editor window. The title bar says "WordCount.java (~/exercises/wordCount) - gedit". The menu bar includes "Edit", "View", "Search", "Tools", "Documents", and "Help". The toolbar has icons for "Open", "Save", "Print...", "Undo", "Redo", "Cut", "Copy", "Paste", "Find", and "Replace". A tab labeled "WordCount.java" is selected. The code in the editor is:

```
WordCount.java (~/exercises/wordCount) - gedit
Edit View Search Tools Documents Help
Open Save Print... Undo Redo Cut Copy Paste Find Replace
WordCount.java x
import java.util.StringTokenizer;

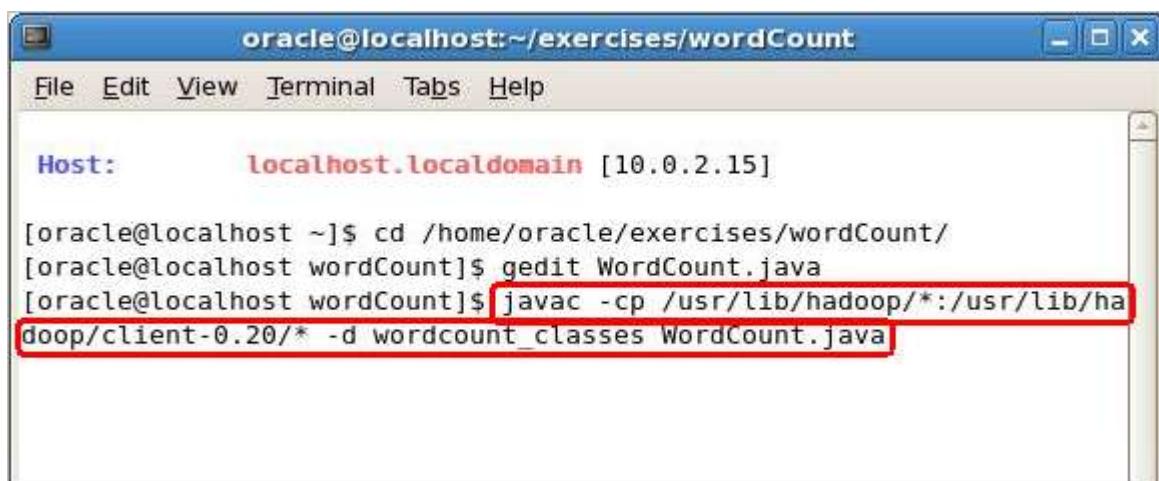
public class WordCount {

    public static class MyMapper extends Mapper<Object,
    Text, Text, IntWritable> {
        private final static IntWritable one = new
        IntWritable(1);
        private Text word = new Text();
```

- We can now go ahead and compile the Word Count code. We need to run a command which will set the correct classpath and output directory while compiling WordCount.java. Type in the terminal:

```
javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop/client-0.20/* -d
wordcount_classes WordCount.java
```

Then press **Enter**



A screenshot of a terminal window titled "oracle@localhost:~/exercises/wordCount". The window has tabs for "File", "Edit", "View", "Terminal", "Tabs", and "Help". The host information shows "Host: localhost.localdomain [10.0.2.15]". The terminal window displays the following command:

```
Host: localhost.localdomain [10.0.2.15]
[oracle@localhost ~]$ cd /home/oracle/exercises/wordCount/
[oracle@localhost wordCount]$ gedit WordCount.java
[oracle@localhost wordCount]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop/client-0.20/* -d wordcount_classes WordCount.java
```

6. We can now create a jar file from the compile directory of Word Count. This jar file is required as the code for word count will be sent to all of the nodes in the cluster and the code will be run simultaneous on all nodes that have appropriate data. To create the jar file, go to the terminal and type:

```
jar -cvf WordCount.jar -C wordcount_classes/ .
```

Then press **Enter**

```
Host: localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/wordCount/
[oracle@localhost wordCount]$ gedit WordCount.java
[oracle@localhost wordCount]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop/client-0.20/* -d wordcount_classes WordCount.java
[oracle@localhost wordCount]$ jar -cvf WordCount.jar -C wordcount_classes/ .
```

7. Let's proceed to view the files on which we will perform the word-count. They contain customer comments that will be the base of our sentiment analysis. Our goal is to see the opinions of the customers when it comes to the car insurance services that our company has to offer. To view the content of the two files, go to the terminal and type:

```
cat file01 file02
```

Then press **Enter**

```
Host: localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/wordCount/
[oracle@localhost wordCount]$ gedit WordCount.java
[oracle@localhost wordCount]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop/client-0.20/* -d wordcount_classes WordCount.java
[oracle@localhost wordCount]$ jar -cvf WordCount.jar -C wordcount_classes/ .
added manifest
adding: WordCount$Map.class(in = 1918) (out= 795)(deflated 58%)
adding: WordCount.class(in = 1516) (out= 740)(deflated 51%)
adding: WordCount$Reduce.class(in = 1591) (out= 642)(deflated 59%)
[oracle@localhost wordCount]$ cat file01 file02
```

In the terminal window you will see the contents of the two files. Each file has some customer comments. Although these are quite small files the code would run identical with more than 2 files and with files that are several Gigabytes or Terabytes in size.

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/wordCount". The window contains the following text:

```
adding: WordCount$Reduce.class(in = 1591) (out= 642)(deflated 59%)
[oracle@localhost wordCount]$ cat file01 file02
very disappointed and very expensive
expensive and unreliable insurance
worthless insurance and expensive
worst customer service
worst insurance company
worst professional staff and unreliable insurance company
insurance is very expensive
worst insurance cover
terrible service
disappointed with the expensive insurance service
worthless and expensive
awful customer service
terrible worst service
worst bank and worst customer service
worst insurance
disappointed with protocols
unreliable insurance
best service I recommend it
good professionals and efficient insurance
I will recommend it
good customer service
best insurance I found I recommend it[oracle@localhost wordCount]$
```

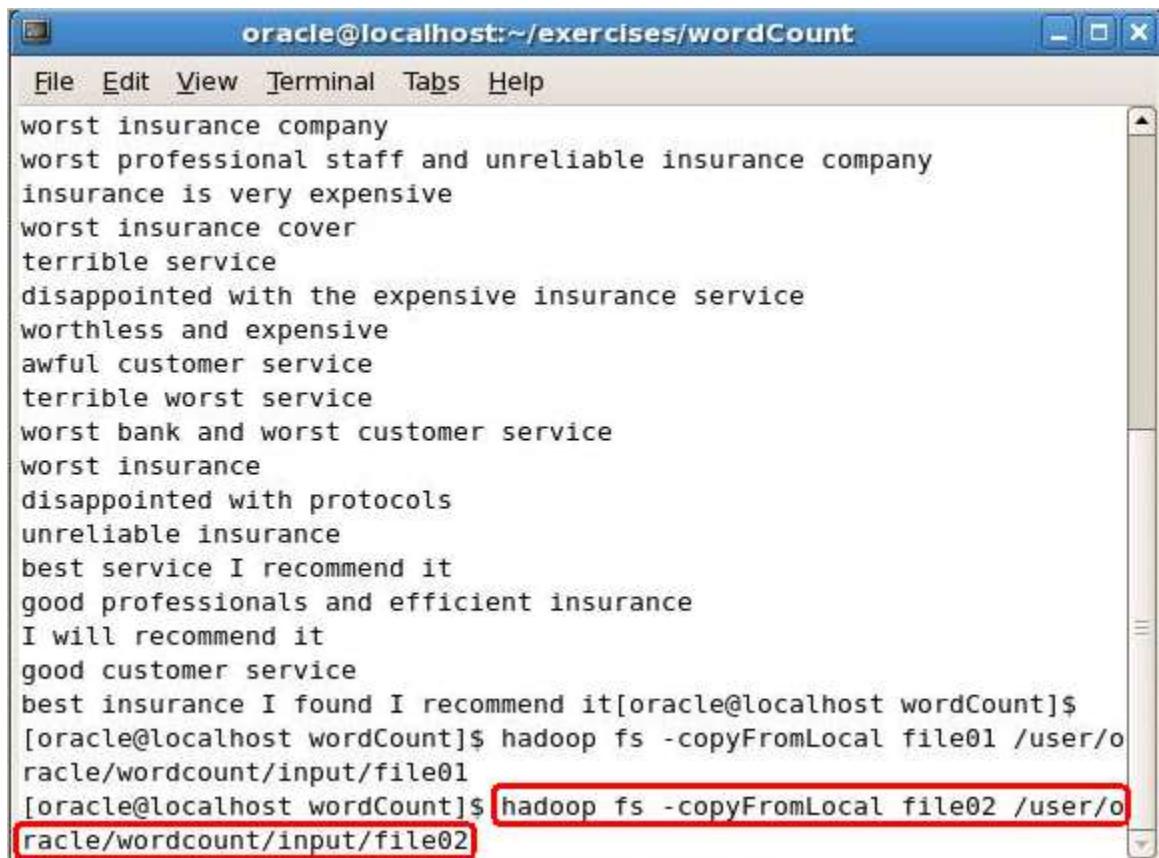
8. Now that we have the files ready we must move them into the Hadoop Distributed File System (HDFS). It is important to note that files which are within HDFS are split into multiple chunks and stored on separate nodes for parallel parsing. To upload our two files into the HDFS you need to use the copyFromLocal command in Hadoop. Run the commands by typing at the terminal:

```
hadoop fs -mkdir /user/oracle/wordcount  
hadoop fs -mkdir /user/oracle/wordcount/input  
hadoop fs -copyFromLocal file01 /user/oracle/wordcount/input/file01
```

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/wordCount". The window contains a list of words, likely from a file, including "worthless insurance and expensive", "worst customer service", "worst insurance company", etc. At the bottom of the list, there is a command being typed:
[oracle@localhost wordCount]\$ hadoop fs -copyFromLocal file01 /user/o
racle/wordcount/input/file01

9. We should now upload the second file. Go to the terminal and type:

```
hadoop fs -copyFromLocal file02 /user/oracle/wordcount/input/file02  
Then press Enter
```

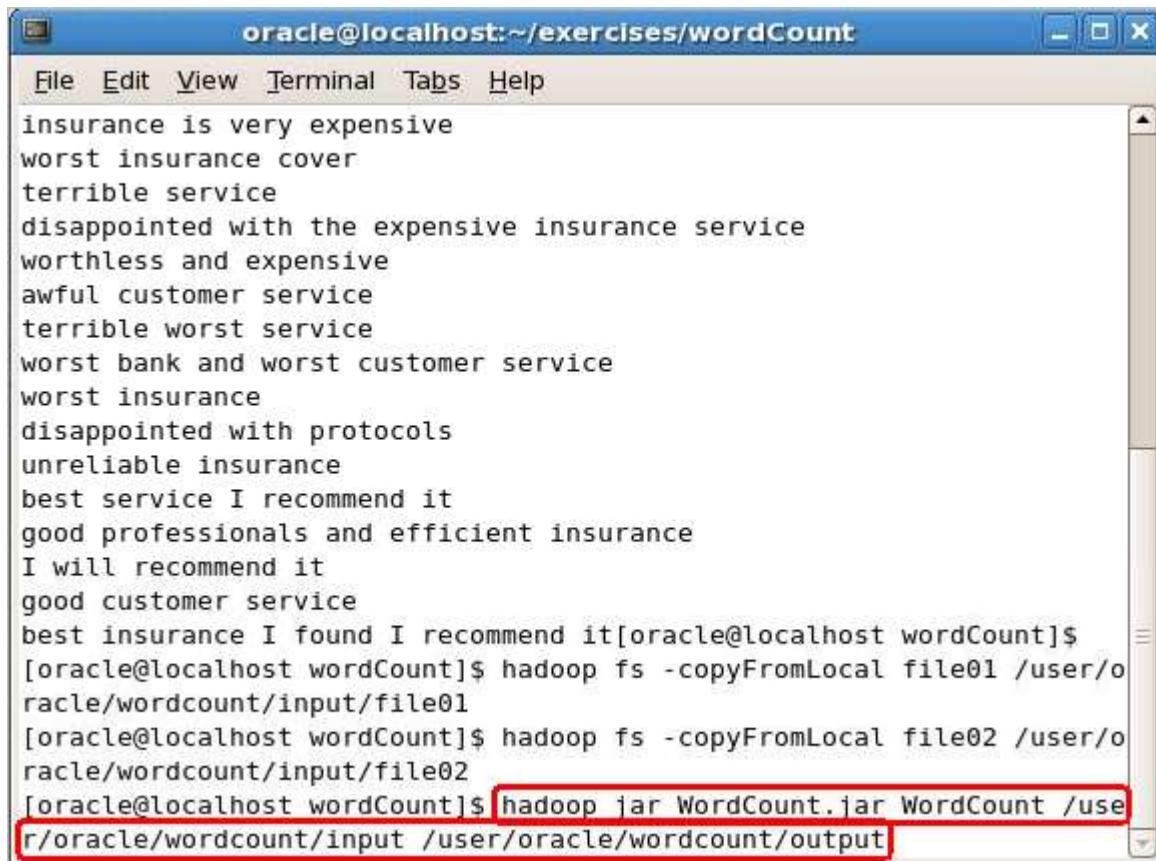


The screenshot shows a terminal window titled "oracle@localhost:~/exercises/wordCount". The window has a menu bar with File, Edit, View, Terminal, Tabs, and Help. The main area of the terminal displays a list of words, likely from a file, including "worst insurance company", "worst professional staff and unreliable insurance company", "insurance is very expensive", "worst insurance cover", "terrible service", "disappointed with the expensive insurance service", "worthless and expensive", "awful customer service", "terrible worst service", "worst bank and worst customer service", "worst insurance", "disappointed with protocols", "unreliable insurance", "best service I recommend it", "good professionals and efficient insurance", "I will recommend it", "good customer service", and "best insurance I found I recommend it". At the bottom of the terminal window, the command "hadoop fs -copyFromLocal file02 /user/oracle/wordcount/input/file02" is visible, with the entire line highlighted by a red rectangle.

10. We can now run our Map/Reduce job to do a word count on the files we just uploaded. Go to the terminal and type:

```
hadoop jar WordCount.jar WordCount /user/oracle/wordcount/input  
/user/oracle/wordcount/output
```

Then press **Enter**



```
insurance is very expensive
worst insurance cover
terrible service
disappointed with the expensive insurance service
worthless and expensive
awful customer service
terrible worst service
worst bank and worst customer service
worst insurance
disappointed with protocols
unreliable insurance
best service I recommend it
good professionals and efficient insurance
I will recommend it
good customer service
best insurance I found I recommend it[oracle@localhost wordCount]$
[oracle@localhost wordCount]$ hadoop fs -copyFromLocal file01 /user/o
racle/wordcount/input/file01
[oracle@localhost wordCount]$ hadoop fs -copyFromLocal file02 /user/o
racle/wordcount/input/file02
[oracle@localhost wordCount]$ hadoop jar WordCount.jar WordCount /use
r/oracle/wordcount/input /user/oracle/wordcount/output
```

A lot of text will role by in the terminal window. This is informational data coming from the Hadoop infrastructure to help track the status of the job. Wait for the job to finish, this is signaled by the command prompt coming back.

11. Once you get the command prompt back, your Map/Reduce task is complete. It is now time to look at the results. We can display the contents of the results file right from HDFS by using the cat command from Hadoop. Go to the terminal and type the following command:

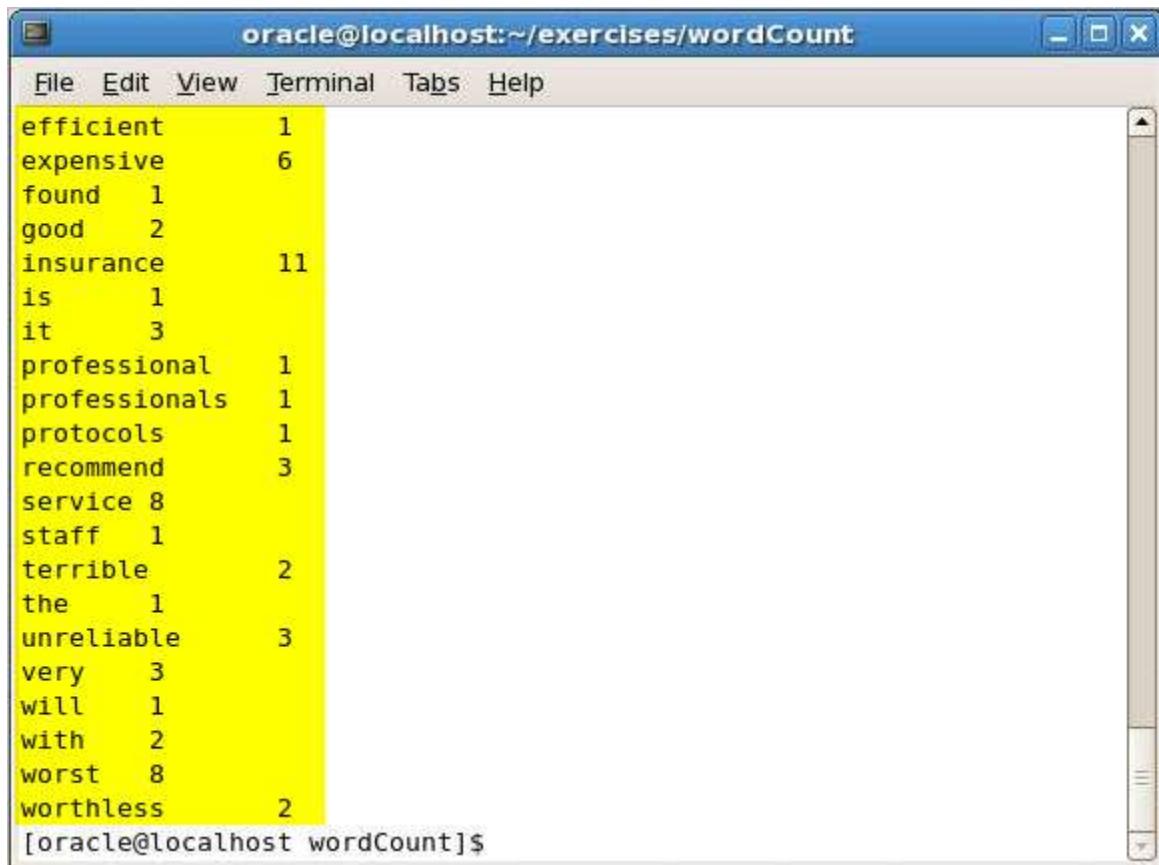
```
hadoop fs -cat /user/oracle/wordcount/output/part-r-00000  
Then press Enter
```

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/wordCount". The window contains several lines of log output from a MapReduce job, followed by the command to display the results.

```
13/08/13 05:15:17 INFO mapred.JobClient: Map output bytes=1000  
13/08/13 05:15:17 INFO mapred.JobClient: Input split bytes=266  
13/08/13 05:15:17 INFO mapred.JobClient: Combine input records=0  
13/08/13 05:15:17 INFO mapred.JobClient: Combine output records=0  
13/08/13 05:15:17 INFO mapred.JobClient: Reduce input groups=30  
13/08/13 05:15:17 INFO mapred.JobClient: Reduce shuffle bytes=504  
13/08/13 05:15:17 INFO mapred.JobClient: Reduce input records=87  
13/08/13 05:15:17 INFO mapred.JobClient: Reduce output records=30  
13/08/13 05:15:17 INFO mapred.JobClient: Spilled Records=174  
13/08/13 05:15:17 INFO mapred.JobClient: CPU time spent (ms)=2440  
13/08/13 05:15:17 INFO mapred.JobClient: Physical memory (bytes)  
snapshot=423661568  
13/08/13 05:15:17 INFO mapred.JobClient: Virtual memory (bytes) s  
napshot=1919705088  
13/08/13 05:15:17 INFO mapred.JobClient: Total committed heap usa  
ge (bytes)=354287616  
[oracle@localhost wordCount]$ hadoop fs -cat /user/oracle/wordcount/o  
utput/part-r-00000
```

In the terminal the word count results are displayed. You will see the job performed a count of the number of times each word appeared in the text files.

We can see that most of the customer comments are negative, as we encounter the word “worst” 8 times, the word “unreliable” 3 times, the word “expensive” 6 times, the word “worthless” 2 times and the word “good” only 2 times.



A screenshot of a terminal window titled "oracle@localhost:~/exercises/wordCount". The window has a blue header bar with the title and standard menu options: File, Edit, View, Terminal, Tabs, Help. The main area of the terminal displays a list of words and their counts, sorted by count. The output is as follows:

Word	Count
efficient	1
expensive	6
found	1
good	2
insurance	11
is	1
it	3
professional	1
professionals	1
protocols	1
recommend	3
service	8
staff	1
terrible	2
the	1
unreliable	3
very	3
will	1
with	2
worst	8
worthless	2

[oracle@localhost wordCount]\$

It is important to note this is a very crude sentiment analysis and this exercise was designed more to give an introduction to map reduce than enable a comprehensive sentiment analysis. Most tools which do sentiment analysis perform context dependent word comparison where the entire sentence is taken into consideration rather than each word in isolation. A Support Vector Machine would be implemented or a specialized tool such as Endeca would give a very well designed sentiment analysis.

12. As an experiment let's try to run the Hadoop job again. Go to the terminal and type:

```
hadoop jar WordCount.jar WordCount /user/oracle/wordcount/input  
/user/oracle/wordcount/output
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/wordCount". The window contains the following text:

```
expensive      6
found         1
good          2
insurance     11
is            1
it            3
professional   1
professionals 1
protocols     1
recommend     3
service       8
staff         1
terrible      2
the           1
unreliable    3
very          3
will          1
with          2
worst         8
worthless     2
```

At the bottom of the window, the command to run the Hadoop job again is shown:

```
[oracle@localhost wordCount]$ hadoop jar WordCount.jar WordCount /use
r/oracle/wordcount/input /user/oracle/wordcount/output
```

The last two lines of the command are highlighted with a red rectangle.

13. You will notice that an error message appears and no map reduce task is executed. This is easily explained by the immutability of data. Since Hadoop does not allow an update of data files (just read and write) you cannot update the data in the results directory hence the execution has nowhere to place the output. For you to re-run the Map-Reduce job you must either point it to another output directory or clean out the current output directory. Let's go ahead and clean out the previous output directory. Go to the terminal and type:

```
hadoop fs -rm -r /user/oracle/wordcount/output  
Then press Enter
```

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/wordCount". The window contains a stack trace of a Java exception, likely a permissions issue, followed by a command being typed at the prompt. The command is "hadoop fs -rm -r /user/oracle/wordcount/output". The last two words of this command are highlighted with a red rectangle.

```
5)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:396)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserG
roupInformation.java:1408)
        at org.apache.hadoop.mapred.JobClient.submitJobInternal(JobCl
ient.java:945)
        at org.apache.hadoop.mapred.JobClient.submitJob(JobClient.jav
a:919)
        at org.apache.hadoop.mapred.JobClient.runJob(JobClient.java:1
368)
        at WordCount.main(WordCount.java:53)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAc
cessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegating
MethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:208)
[oracle@localhost wordCount]$ hadoop fs -rm -r /user/oracle/wordcount
/output
```

14. Now we have cleared the output directory and can re-run the map reduce task. Let's just go ahead and make sure it works again. Go to the terminal and type:

```
hadoop jar WordCount.jar WordCount /user/oracle/wordcount/input  
/user/oracle/wordcount/output
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/wordCount". The window contains the following text:

```
at org.apache.hadoop.mapred.JobClient.submitJobInternal(JobCl  
ient.java:945)  
at org.apache.hadoop.mapred.JobClient.submitJob(JobClient.jav  
a:919)  
at org.apache.hadoop.mapred.JobClient.runJob(JobClient.java:1  
368)  
at WordCount.main(WordCount.java:53)  
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method  
)  
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAc  
cessorImpl.java:39)  
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegating  
MethodAccessorImpl.java:25)  
at java.lang.reflect.Method.invoke(Method.java:597)  
at org.apache.hadoop.util.RunJar.main(RunJar.java:208)  
[oracle@localhost wordCount]$ hadoop fs -rm -r /user/oracle/wordcount  
/output  
Moved: 'hdfs://localhost.localdomain:8020/user/oracle/wordcount/outpu  
t' to trash at: hdfs://localhost.localdomain:8020/user/oracle/.Trash/  
Current  
[oracle@localhost wordCount]$ [hadoop jar WordCount.jar WordCount /use  
r/oracle/wordcount/input /user/oracle/wordcount/output]
```

The command being typed is highlighted with a red rectangle.

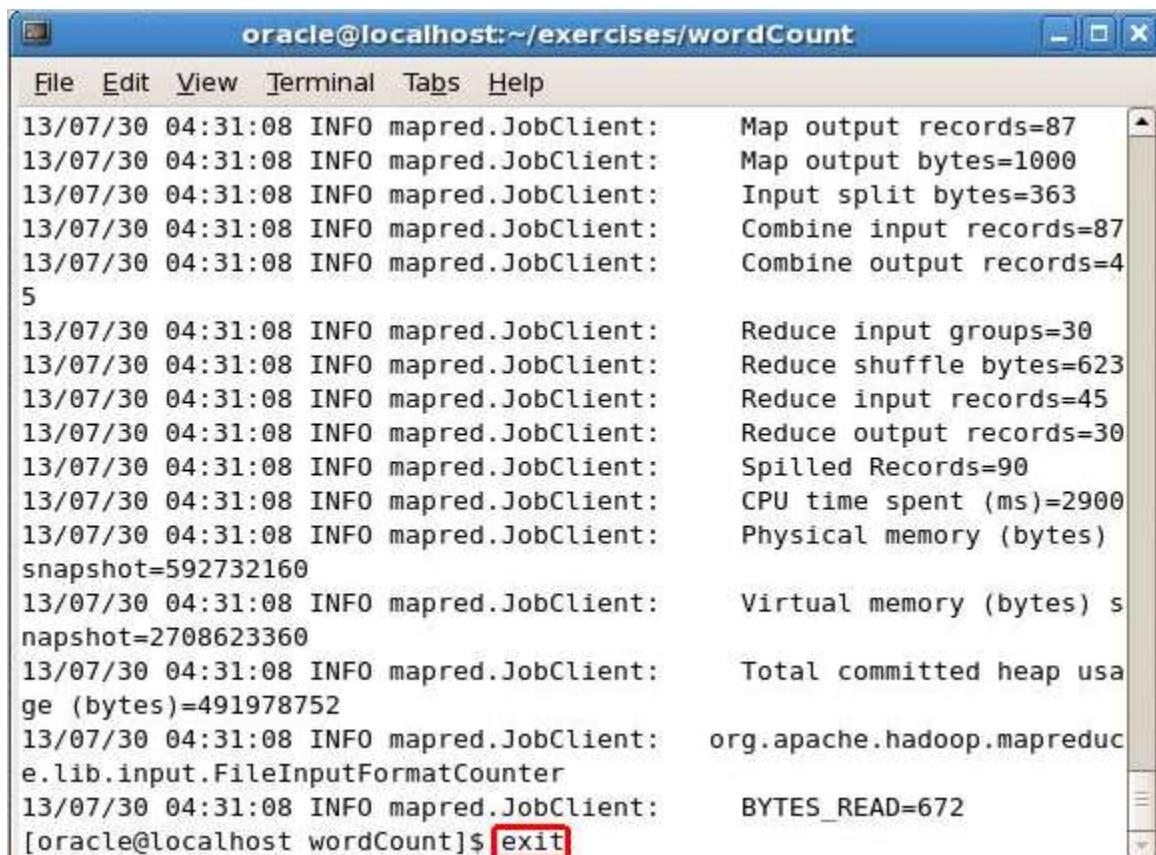
Now the Map Reduce job ran fine again as per the output on the screen.

```
oracle@localhost:~/exercises/wordCount
File Edit View Terminal Tabs Help
13/07/30 04:31:08 INFO mapred.JobClient:      Map output records=87
13/07/30 04:31:08 INFO mapred.JobClient:      Map output bytes=1000
13/07/30 04:31:08 INFO mapred.JobClient:      Input split bytes=363
13/07/30 04:31:08 INFO mapred.JobClient:      Combine input records=87
13/07/30 04:31:08 INFO mapred.JobClient:      Combine output records=4
5
13/07/30 04:31:08 INFO mapred.JobClient:      Reduce input groups=30
13/07/30 04:31:08 INFO mapred.JobClient:      Reduce shuffle bytes=623
13/07/30 04:31:08 INFO mapred.JobClient:      Reduce input records=45
13/07/30 04:31:08 INFO mapred.JobClient:      Reduce output records=30
13/07/30 04:31:08 INFO mapred.JobClient:      Spilled Records=90
13/07/30 04:31:08 INFO mapred.JobClient:      CPU time spent (ms)=2900
13/07/30 04:31:08 INFO mapred.JobClient:      Physical memory (bytes)
snapshot=592732160
13/07/30 04:31:08 INFO mapred.JobClient:      Virtual memory (bytes) s
napshot=2708623360
13/07/30 04:31:08 INFO mapred.JobClient:      Total committed heap usa
ge (bytes)=491978752
13/07/30 04:31:08 INFO mapred.JobClient:      org.apache.hadoop.mapreduc
e.lib.input.FileInputFormatCounter
13/07/30 04:31:08 INFO mapred.JobClient:      BYTES_READ=672
[oracle@localhost wordCount]$
```

15. This completes the word count example. You can now close the terminal window; go to the terminal window and type:

```
exit
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/wordCount". The window contains a list of log entries from a Hadoop job. The entries are as follows:

```
13/07/30 04:31:08 INFO mapred.JobClient: Map output records=87
13/07/30 04:31:08 INFO mapred.JobClient: Map output bytes=1000
13/07/30 04:31:08 INFO mapred.JobClient: Input split bytes=363
13/07/30 04:31:08 INFO mapred.JobClient: Combine input records=87
13/07/30 04:31:08 INFO mapred.JobClient: Combine output records=4
5
13/07/30 04:31:08 INFO mapred.JobClient: Reduce input groups=30
13/07/30 04:31:08 INFO mapred.JobClient: Reduce shuffle bytes=623
13/07/30 04:31:08 INFO mapred.JobClient: Reduce input records=45
13/07/30 04:31:08 INFO mapred.JobClient: Reduce output records=30
13/07/30 04:31:08 INFO mapred.JobClient: Spilled Records=90
13/07/30 04:31:08 INFO mapred.JobClient: CPU time spent (ms)=2900
13/07/30 04:31:08 INFO mapred.JobClient: Physical memory (bytes)
snapshot=592732160
13/07/30 04:31:08 INFO mapred.JobClient: Virtual memory (bytes) s
napshot=2708623360
13/07/30 04:31:08 INFO mapred.JobClient: Total committed heap usa
ge (bytes)=491978752
13/07/30 04:31:08 INFO mapred.JobClient: org.apache.hadoop.mapreduc
e.lib.input.FileInputFormatCounter
13/07/30 04:31:08 INFO mapred.JobClient: BYTES_READ=672
[oracle@localhost wordCount]$ exit
```

3.4 Summary

In this exercise you were able to see the basic steps required in setting up and running a very simple Map Reduce Job. You saw what interfaces must be implemented when creating a Map/Reduce task, you saw how to upload data into HDFS and how to run the map reduce task. It is important to talk about execution time for the exercise, as the amount of time required to count a few words is quite high in absolute terms. It is important to understand that Hadoop needs to start a separate Java Virtual Machine to process each file or chunk of a file on each node of the cluster. As such even a trivial job has some processing time which limits the possible application of Hadoop as it can only handle batch jobs. Real time applications where answers are required immediately can't be run on a Hadoop cluster. At the same time as the data volumes increase processing time does not increase that much as long as there are enough processing nodes. A recent benchmark of a Hadoop cluster saw the complete sorting of 1 terabyte of data in just over 3 minutes on 910 nodes.

4. ORACLE NoSQL DATABASE

4.1 Introduction to NoSQL

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

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

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

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

4.2 Overview of Hands on Exercise

In this exercise you will be experimenting with the Oracle NoSQL Database. We need to process and retrieve our banking transactions-related data from our production NoSQL database, in order to prepare it for import into the Oracle Database. The import will be performed in a later exercise. Before we proceed to our actual use-case, however, you will get to experiment with some simple examples, so that you can get an understanding of what Oracle NoSQL Database is and how it can be used. Most of the exercises will have you look at pre-written Java code, then compile and run that code. Ensure that you understand the code and all of its nuances as it is what makes up the NoSQL Database interface. If you would like to understand all of the functions that are available, see the Javadoc available on the Oracle Website here: <http://docs.oracle.com/cd/NOSQL/html/javadoc/>

In this exercise you will:

1. Insert and retrieve a simple key value pair from the NoSQL Database
2. Experiment with the storeIterator functionality to retrieve multiple values at the same time
3. Use Oracle Database external tables to read data from a NoSQL Database
4. Use AVRO schemas to store complex data inside the value field
5. Aggregate complex data using Map-Reduce and store the results in HDFS

4.3 Insert and retrieve data from the NoSQL Database

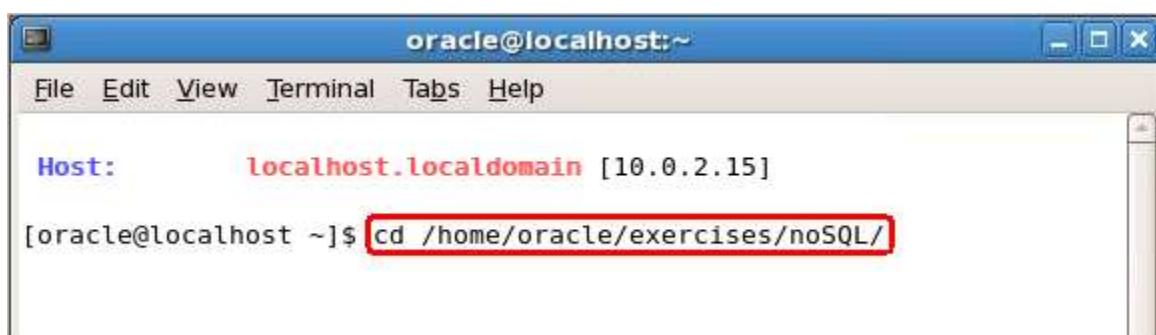
1. All of the setup and execution for this exercise can be done from the terminal, hence open a terminal by double clicking on the **Terminal icon** on the desktop.



2. To get into the folder where the scripts for the NoSQL exercise are, type in the terminal:

```
cd /home/oracle/exercises/noSQL
```

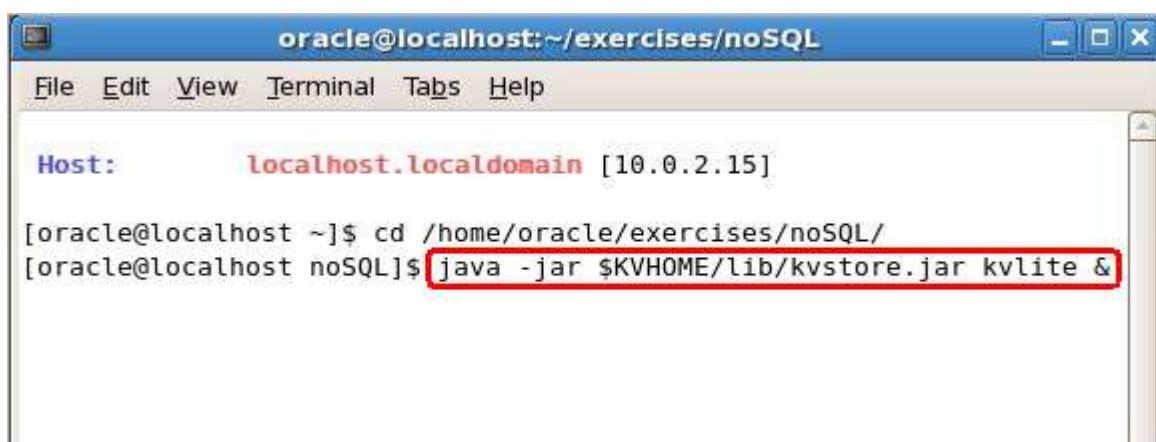
Then press **Enter**



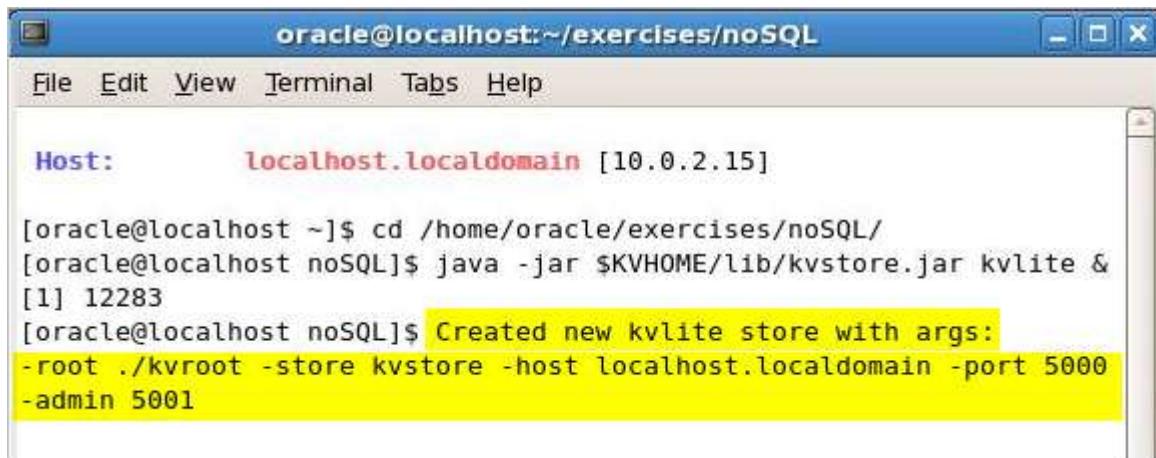
3. Before we do anything with the NoSQL Database we must first start it. So let's go ahead and do that. Go to the terminal and type:

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

Then press **Enter**



Please wait until the following message appears in the terminal, before proceeding.



A screenshot of a terminal window titled "oracle@localhost:~/exercises/noSQL". The window has a standard Linux-style title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main terminal area displays the following text:

```
Host: localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &
[1] 12283
[oracle@localhost noSQL]$ Created new kvlite store with args:
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000
-admin 5001
```

- Once the message appears, press **Enter** to regain control of the terminal prompt.



A screenshot of a terminal window titled "oracle@localhost:~/exercises/noSQL". The window has a standard Linux-style title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main terminal area displays the same text as the previous screenshot, but with a cursor at the end of the last line:

```
Host: localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &
[1] 12283
[oracle@localhost noSQL]$ Created new kvlite store with args:
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000
-admin 5001

[oracle@localhost noSQL]$ █
```

5. To check if the database is up and running we can do a ping on the database. Go to the terminal and type:

```
java -jar $KVHOME/lib/kvstore.jar ping -port 5000 -host `hostname`  
Then press Enter
```



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window has a standard OS X-style title bar with icons for close, minimize, and maximize. Below the title bar is a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main terminal area displays the following text:

```
Host:      localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &
[1] 12283
[oracle@localhost noSQL]$ Created new kvlite store with args:
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000
-admin 5001

[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -por
t 5000 -host `hostname`
```

The last line of the command, "java -jar \$KVHOME/lib/kvstore.jar ping -port 5000 -host `hostname`", is highlighted with a red rectangular box.

You will see Status: RUNNING displayed within the text. This shows the database is up and running.

```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help

Host: localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &
[1] 12283
[oracle@localhost noSQL]$ Created new kvlite store with args:
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000
-admin 5001

[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -por
t 5000 -host `hostname`
Pinging components of store kvstore based upon topology sequence #14
kvstore comprises 10 partitions and 1 Storage Nodes
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLit
e [dcl] Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC
Build id: a48c0a29cbe4
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence nu
mber: 31 haPort: 5006
[oracle@localhost noSQL]$
```

6. Before we start processing our transactional data that is needed to create the 360 degree view of the customers, let's take some time to discover how the Oracle NoSQL Database can be used through some simple "Hello World"-type exercises.

Oracle NoSQL Database uses a Java interface to interact with the data. This is a dedicated Java API which will let you insert, update, delete and query data in the Key/Value store that is the NoSQL Database. Let's look at a very simple example of Java code where we insert a Key/Value pair into the database and then retrieve it. Go to the terminal and type:

```
gedit Hello.java
```

Then press **Enter**

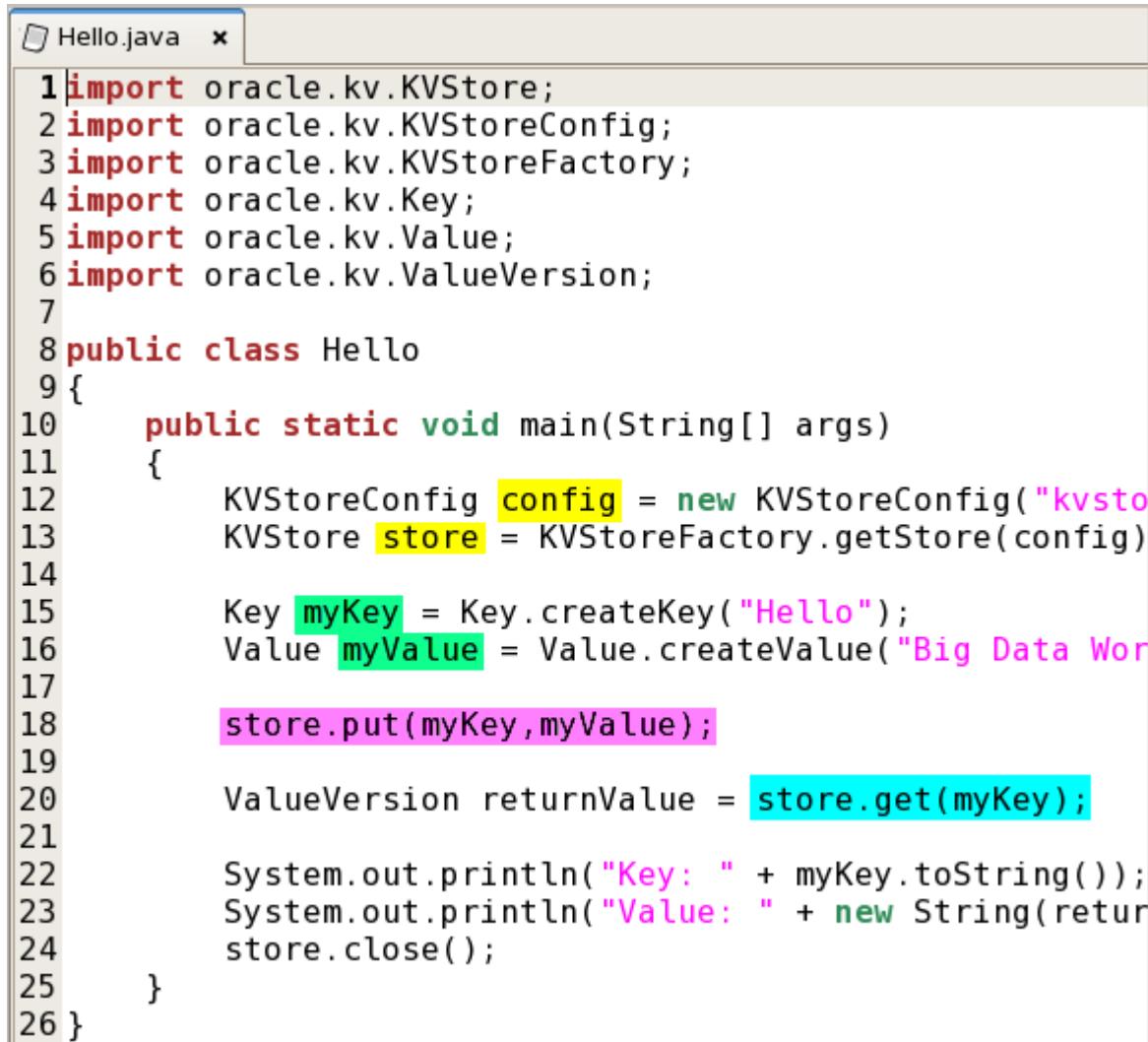
The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains the following text:

```
Host: localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &
[1] 12283
[oracle@localhost noSQL]$ Created new kvlite store with args:
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000
-admin 5001

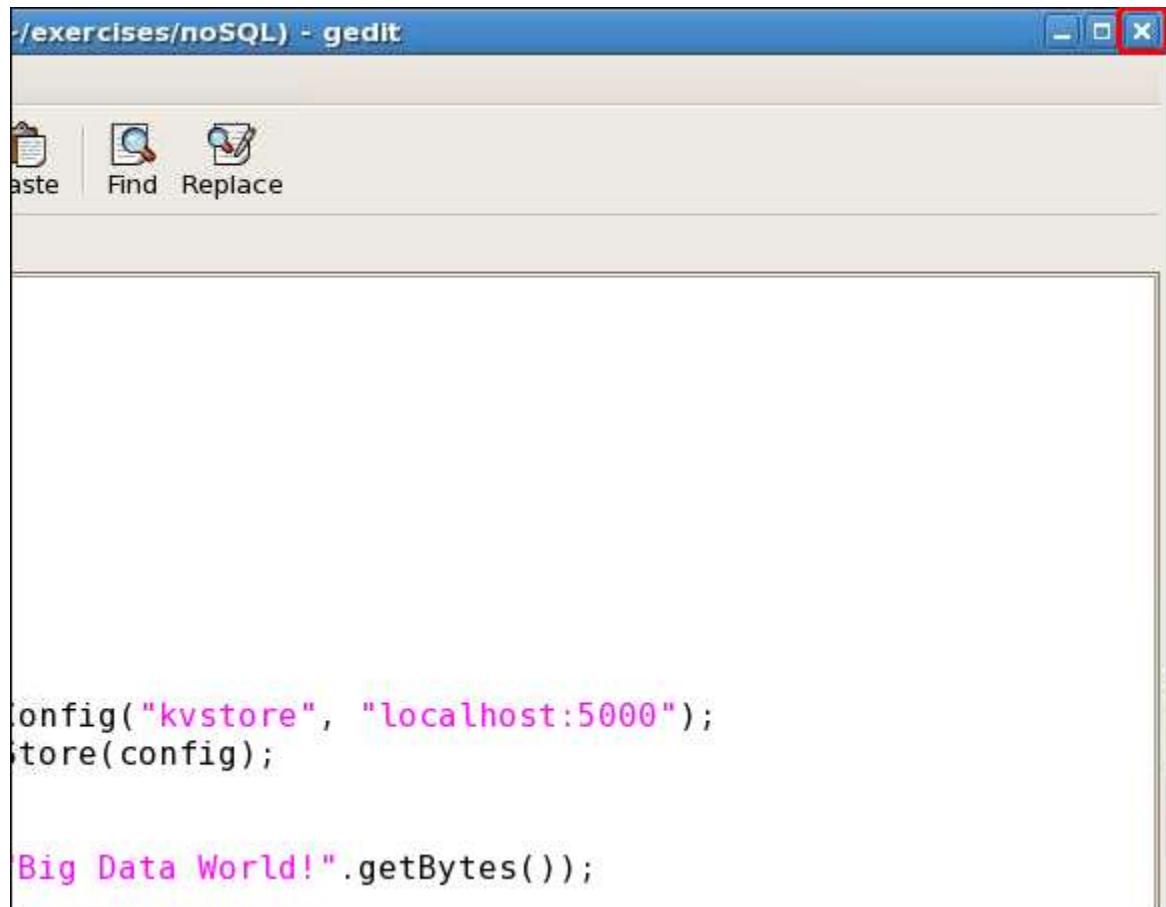
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -por
t 5000 -host `hostname`
Pinging components of store kvstore based upon topology sequence #14
kvstore comprises 10 partitions and 1 Storage Nodes
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLit
e [dcl] Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC
Build id: a48c0a29cbe4
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence nu
mber: 31 haPort: 5006
[oracle@localhost noSQL]$ gedit Hello.java
```

A new window will pop up with the code. In this code there are a couple of things to be noted. We see the *config* variable which holds our connection string and the *store* variable which is our connection factory to the database. They are the initialization variables for the Key/Value Store and are highlighted in yellow. Next we see the definition of 2 variables, of type Key and Value, they will serve as our payload to be inserted. These are highlighted in green. Next we have highlighted in purple the actual insert command. Highlighted in blue is the retrieve command for getting data out of the database.



```
1 import oracle.kv.KVStore;
2 import oracle.kv.KVStoreConfig;
3 import oracle.kv.KVStoreFactory;
4 import oracle.kv.Key;
5 import oracle.kv.Value;
6 import oracle.kv.ValueVersion;
7
8 public class Hello
9 {
10     public static void main(String[] args)
11     {
12         KVStoreConfig config = new KVStoreConfig("kvsto
13         KVStore store = KVStoreFactory.getStore(config)
14
15         Key myKey = Key.createKey("Hello");
16         Value myValue = Value.createValue("Big Data Wor
17
18         store.put(myKey,myValue);
19
20         ValueVersion returnValue = store.get(myKey);
21
22         System.out.println("Key: " + myKey.toString());
23         System.out.println("Value: " + new String(return
24         store.close();
25     }
26 }
```

- When you are done evaluating the code click on the **X** in the right upper corner of the window to close it.



The screenshot shows a window titled "/exercises/noSQL) - gedit". The window has a toolbar with icons for Paste, Find, and Replace. The main text area contains the following Java code:

```
config("kvstore", "localhost:5000");
store(config);

Big Data World!".getBytes());
```

8. Let's go ahead and compile that code. Go to the terminal and type:

```
javac Hello.java  
Then press Enter
```

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The host is "localhost.localdomain [10.0.2.15]". The terminal output shows the following commands and their results:

```
Host: localhost.localdomain [10.0.2.15]

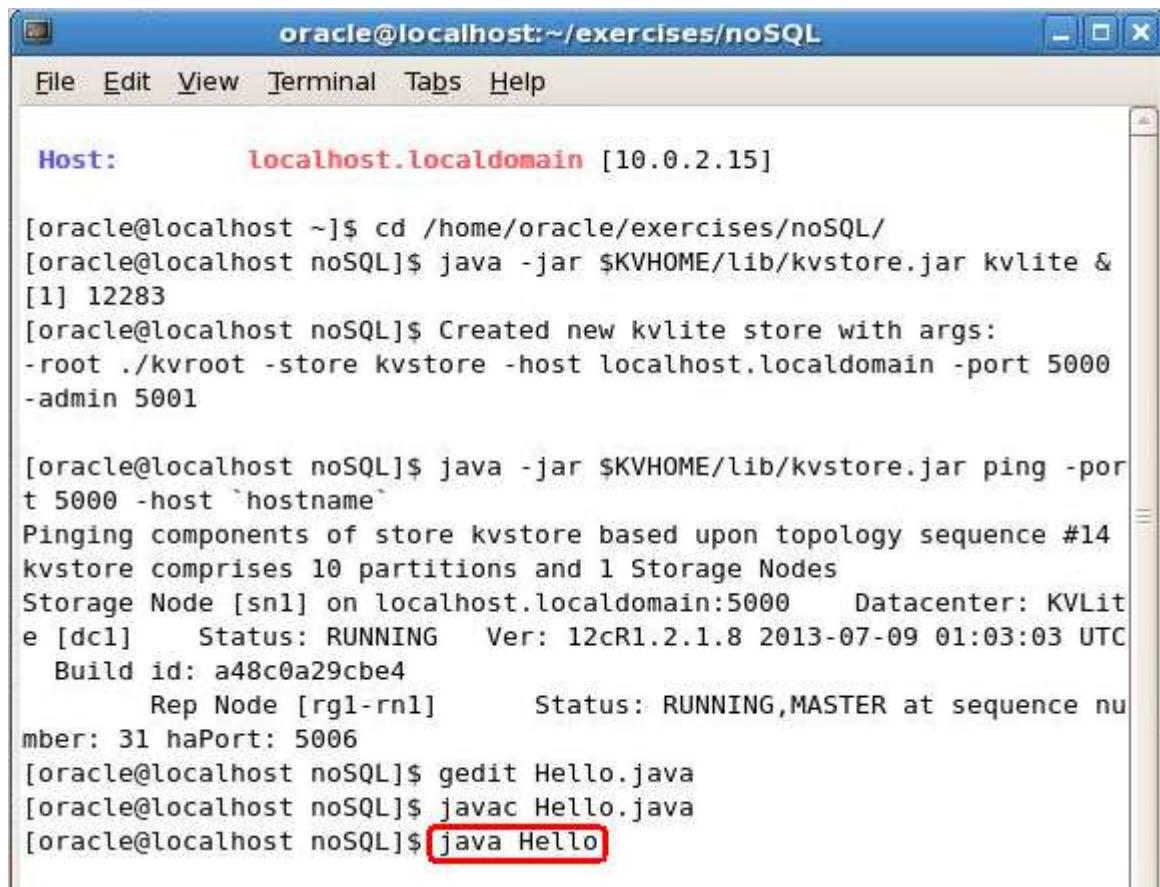
[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &
[1] 12283
[oracle@localhost noSQL]$ Created new kvlite store with args:
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000
-admin 5001

[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -por
t 5000 -host `hostname`
Pinging components of store kvstore based upon topology sequence #14
kvstore comprises 10 partitions and 1 Storage Nodes
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLit
e [dcl] Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC
Build id: a48c0a29cbe4
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence nu
mber: 31 haPort: 5006
[oracle@localhost noSQL]$ gedit Hello.java
[oracle@localhost noSQL]$ javac Hello.java
```

9. Now that the code is complied let's run it. Go to the terminal and type:

```
java Hello
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains the following text:

```
Host: localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &
[1] 12283
[oracle@localhost noSQL]$ Created new kvlite store with args:
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000
-admin 5001

[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -por
t 5000 -host `hostname`
Pinging components of store kvstore based upon topology sequence #14
kvstore comprises 10 partitions and 1 Storage Nodes
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLite
[dcl] Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC
Build id: a48c0a29cbe4
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence nu
mber: 31 haPort: 5006
[oracle@localhost noSQL]$ gedit Hello.java
[oracle@localhost noSQL]$ javac Hello.java
[oracle@localhost noSQL]$ java Hello
```

You will see the Key we inserted in the Database ("Hello"), together with the Value ("Big Data World!") printed on the screen.

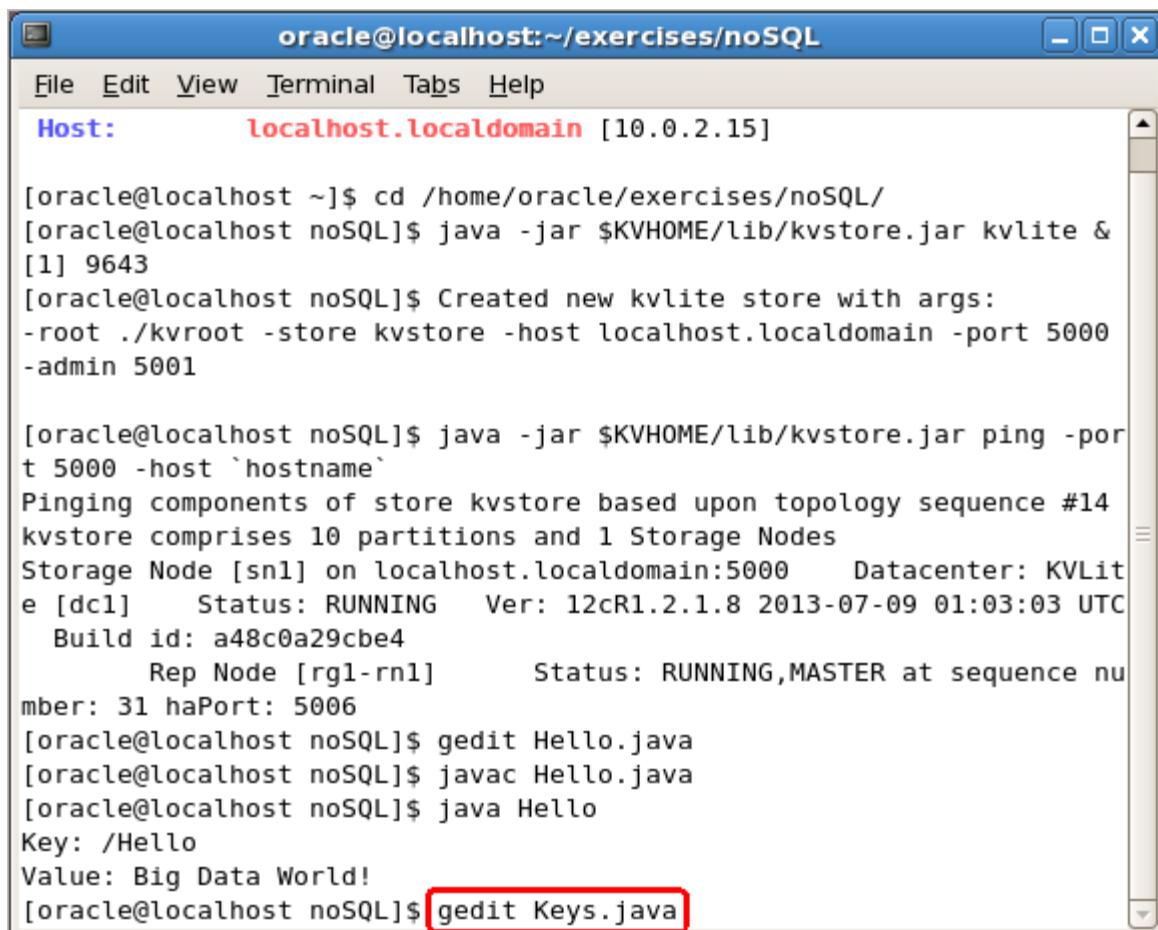
```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
Host: localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &
[1] 9643
[oracle@localhost noSQL]$ Created new kvlite store with args:
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000
-admin 5001

[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -por
t 5000 -host `hostname`
Pinging components of store kvstore based upon topology sequence #14
kvstore comprises 10 partitions and 1 Storage Nodes
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLit
e [dcl] Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC
Build id: a48c0a29cbe4
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence nu
mber: 31 haPort: 5006
[oracle@localhost noSQL]$ gedit Hello.java
[oracle@localhost noSQL]$ javac Hello.java
[oracle@localhost noSQL]$ java Hello
Key: /Hello
Value: Big Data World!
[oracle@localhost noSQL]$
```

10. Oracle NoSQL Database has the possibility of having a major and a minor component to the Key. This feature can be very useful when trying to group and retrieve multiple items at the same time from the database. Additionally, the major component of the Key may be comprised of more than one item. In the next code we have 2 major components to the Key, each major component being represented by a pair of items ((Participant,Mike) and (Participant,Dave)); each major component has a minor component (Question and Answer). We will insert a Value for each Key and we will use the storeIterator function to retrieve all of the Values regardless of the minor component of the Key for Mike and completely ignore Dave. Let's see what that code looks like. Go to the terminal and type:

```
gedit Keys.java  
Then press Enter
```



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The host is listed as "localhost.localdomain [10.0.2.15]". The terminal output shows the following sequence of commands and responses:

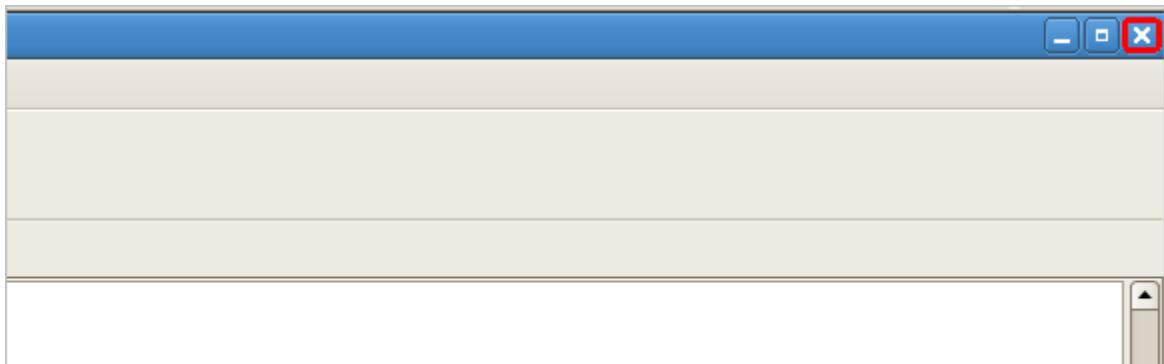
```
[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/  
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &  
[1] 9643  
[oracle@localhost noSQL]$ Created new kvlite store with args:  
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000  
-admin 5001  
  
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -por  
t 5000 -host `hostname`  
Pinging components of store kvstore based upon topology sequence #14  
kvstore comprises 10 partitions and 1 Storage Nodes  
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLit  
e [dcl] Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC  
Build id: a48c0a29cbe4  
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence nu  
mber: 31 haPort: 5006  
[oracle@localhost noSQL]$ gedit Hello.java  
[oracle@localhost noSQL]$ javac Hello.java  
[oracle@localhost noSQL]$ java Hello  
Key: /Hello  
Value: Big Data World!  
[oracle@localhost noSQL]$ gedit Keys.java
```

A new window will pop up with the code. If you scroll to the bottom you will remark the code in the screenshot below. Highlighted in purple are the insertion calls which will add the data to the database. The retrieval of multiple records is highlighted in blue, and the green shows the display of the retrieved data. Do note that there were 4 Key-Value pairs inserted into the database.

```
Value MikeQuestion = Value.createValue(questionMike.getBytes());
Value MikeAnswer = Value.createValue(answerMike.getBytes());
Value DaveQuestion = Value.createValue(questionDave.getBytes());
Value DaveAnswer = Value.createValue(answerDave.getBytes());
store.put(MikeKey1, MikeQuestion);
store.put(MikeKey2, MikeAnswer);
store.put(DaveKey1, DaveQuestion);
store.put(DaveKey2, DaveAnswer);

Iterator<KeyValueVersion> MyStoreIterator = store.storeIterator();
while (MyStoreIterator.hasNext())
{
    KeyValueVersion entry = MyStoreIterator.next();
    Key k = entry.getKey();
    Value v = entry.getValue();
    System.out.println(k.toString() + " - " + new String(v));
}
```

11. When you are done evaluating the code click on the **X** in the right upper corner of the window to close it.



12. Let's go ahead and compile that code. Go to the terminal and type:

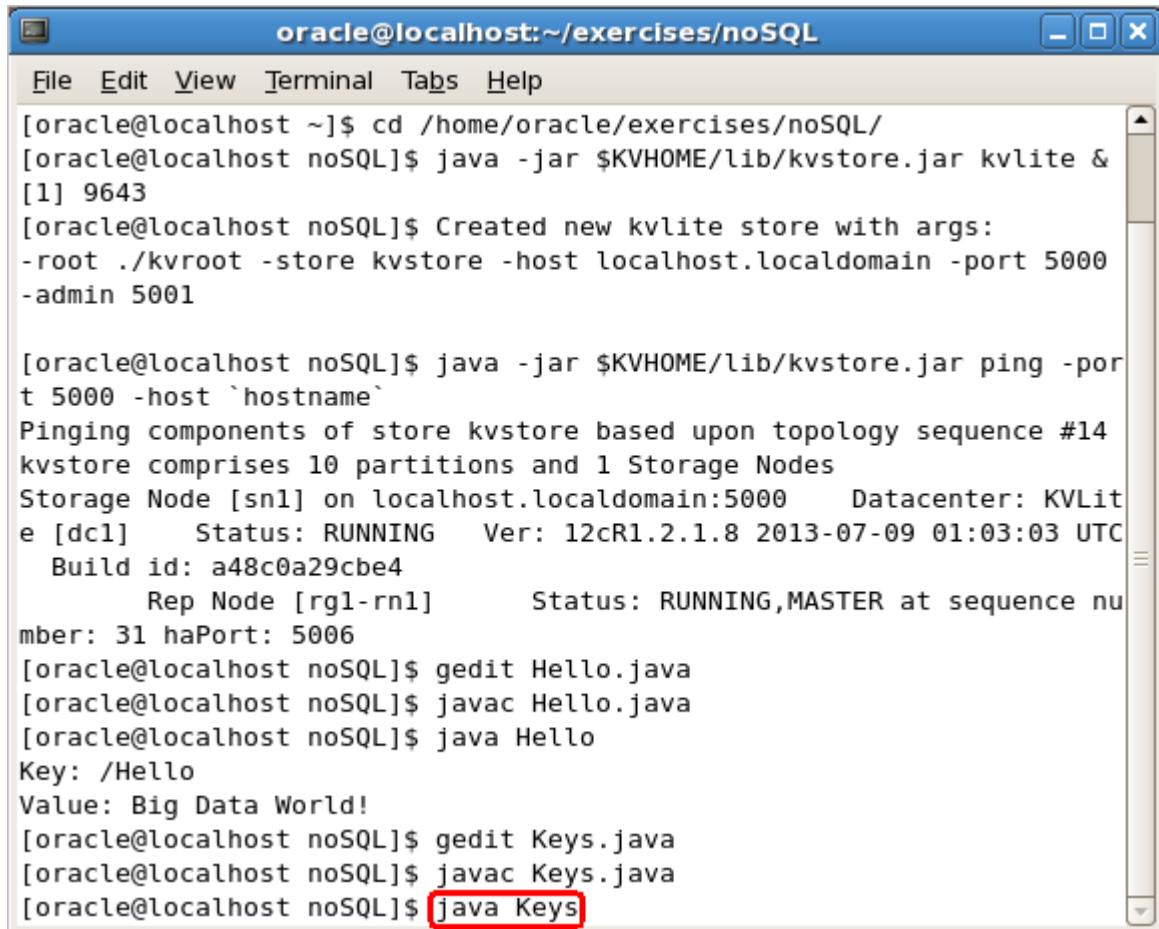
```
javac Keys.java  
Then press Enter
```

```
[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/  
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &  
[1] 9643  
[oracle@localhost noSQL]$ Created new kvlite store with args:  
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000  
-admin 5001  
  
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -por  
t 5000 -host `hostname`  
Pinging components of store kvstore based upon topology sequence #14  
kvstore comprises 10 partitions and 1 Storage Nodes  
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLit  
e [dc1] Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC  
Build id: a48c0a29cbe4  
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence nu  
mber: 31 haPort: 5006  
[oracle@localhost noSQL]$ gedit Hello.java  
[oracle@localhost noSQL]$ javac Hello.java  
[oracle@localhost noSQL]$ java Hello  
Key: /Hello  
Value: Big Data World!  
[oracle@localhost noSQL]$ gedit Keys.java  
[oracle@localhost noSQL]$ javac Keys.java
```

13. Now that the code is complied let's run it. Go to the terminal and type:

java Keys

Then press **Enter**

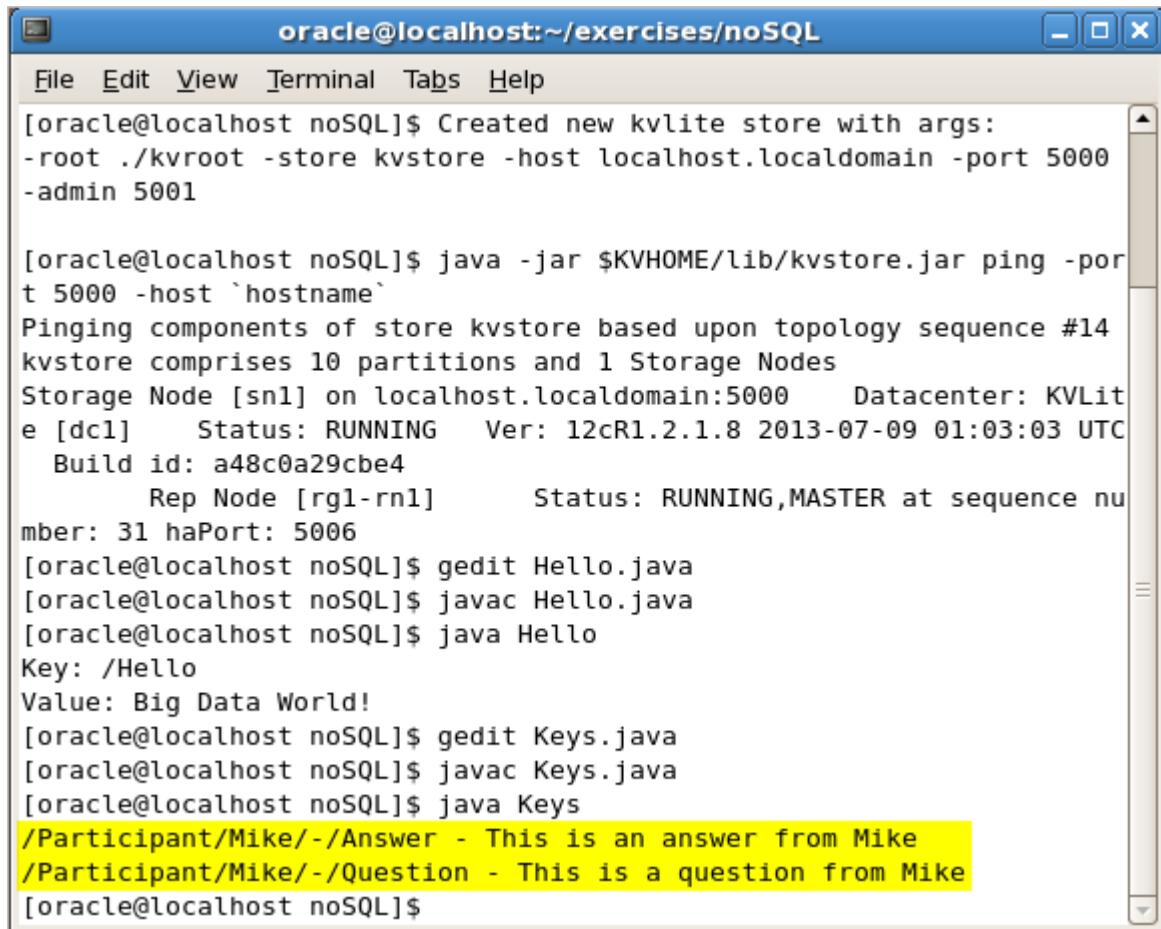


The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains a command-line session. The user has navigated to the directory "/home/oracle/exercises/noSQL" and run the command "java -jar \$KVHOME/lib/kvstore.jar kvlite &". This command creates a new kvlite store with specific arguments: -root ./kvroot -store kvstore -host localhost.localdomain -port 5000 -admin 5001. The user then runs a ping command: "java -jar \$KVHOME/lib/kvstore.jar ping -port 5000 -host `hostname`", which pings components of the store based on topology sequence #14. The kvstore comprises 10 partitions and 1 Storage Nodes. A Storage Node [sn1] is listed with details: Datacenter: KVLite [dcl], Status: RUNNING, Version: 12cR1.2.1.8 2013-07-09 01:03:03 UTC, Build id: a48c0a29cbe4, Rep Node [rg1-rn1], Status: RUNNING,MASTER at sequence number: 31, haPort: 5006. The user then edits and compiles a Java file: "gedit Hello.java", "javac Hello.java", and "java Hello". The output shows a key-value pair: Key: /Hello and Value: Big Data World!. Finally, the user runs the "java Keys" command, which is highlighted with a red rectangle.

```
[oracle@localhost ~]$ cd /home/oracle/exercises/noSQL/
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar kvlite &
[1] 9643
[oracle@localhost noSQL]$ Created new kvlite store with args:
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000
-admin 5001

[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -port 5000 -host `hostname`
Pinging components of store kvstore based upon topology sequence #14
kvstore comprises 10 partitions and 1 Storage Nodes
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLite [dcl]
Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC
Build id: a48c0a29cbe4
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence number: 31 haPort: 5006
[oracle@localhost noSQL]$ gedit Hello.java
[oracle@localhost noSQL]$ javac Hello.java
[oracle@localhost noSQL]$ java Hello
Key: /Hello
Value: Big Data World!
[oracle@localhost noSQL]$ gedit Keys.java
[oracle@localhost noSQL]$ javac Keys.java
[oracle@localhost noSQL]$ java Keys
```

You will see the 2 values that are stored under the (Participant, Mike) major key displayed on the screen, and no values for the (Participant, Dave) major key. Major and minor parts of the key can be composed of multiple strings and further filtering can be done. This is left up to the participants to experiment with.



```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
[oracle@localhost noSQL]$ Created new kvlite store with args:
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000
-admin 5001

[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -port 5000 -host `hostname`
Pinging components of store kvstore based upon topology sequence #14
kvstore comprises 10 partitions and 1 Storage Nodes
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLite [dc1]
Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC
Build id: a48c0a29cbe4
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence number: 31 haPort: 5006
[oracle@localhost noSQL]$ gedit Hello.java
[oracle@localhost noSQL]$ javac Hello.java
[oracle@localhost noSQL]$ java Hello
Key: /Hello
Value: Big Data World!
[oracle@localhost noSQL]$ gedit Keys.java
[oracle@localhost noSQL]$ javac Keys.java
[oracle@localhost noSQL]$ java Keys
/Participant/Mike/-/Answer - This is an answer from Mike
/Participant/Mike/-/Question - This is a question from Mike
[oracle@localhost noSQL]$
```

4.4 noSQL KV Client

1. We have now added some keys and data to our noSQL DB – the Oracle NoSQL DB comes with a command line utility to view and edit data in the DB.

Run the command :

```
java -jar $KVHOME/lib/kvcli.jar -host localhost -port 5000 -store kvstore
```

```
[oracle@bigdatalite noSQL]$ java -jar $KVHOME/lib/kvcli.jar -host localhost -port 5000 -store kvstore
Connected to kvstore
kv-> █
```

2. Enter “help” at the kv-> prompt to see the available commands

```
kv-> help
Oracle NoSQL Database Administrative Commands:
    aggregate
    change-policy
    configure
    connect
    ddl
    delete
    exit
    get
    help
    history
    load
    logtail
    ping
    plan
    pool
    put
    show
    snapshot
    table
    time
    topology
    verbose
    verify

kv-> █
```

3. Enter "get kv -key /Hello" at the kv-> prompt to display the first key we entered in this exercise.
4. Enter "put kv -key /Hello -value "goodbye yall" at the kv-> prompt, then enter "get kv -key /Hello" at the kv-> prompt. You have just successfully updated a value in the noSQL DB.

```
kv-> get kv -key /Hello
Big Data World!
kv-> put kv -key /Hello -value "goodbye yall"
Operation successful, record updated.
kv-> get kv -key /Hello
goodbye yall
█
```

5. Enter exit at the kv-> prompt.

4.5 Oracle External Tables pointing to NoSQL data

1. We will now proceed to show how to create external tables in the Oracle Database, pointing to data stored in a NoSQL Database. We will create the Oracle external table that will allow Oracle users to query the NoSQL data. For this exercise we will use the NoSQL data that we just inserted and create a table with three columns: the *name* ("Mike" or "Dave"), the *type* ("Question" or "Answer") and the *text* ("This is a(n) question/answer from Mike/Dave"). This table will contain all the NoSQL records for which the first item in the Key's major component is "Participant"; basically, it will contain all four records that were inserted.

First, let's review the command used to create the external table. Go to the terminal and type:

gedit createTable.sh

Then press **Enter**

```
[oracle@localhost noSQL]$ Created new kvlite store with args:  
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000  
-admin 5001  
  
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -por  
t 5000 -host `hostname`  
Pinging components of store kvstore based upon topology sequence #14  
kvstore comprises 10 partitions and 1 Storage Nodes  
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLit  
e [dcl] Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC  
Build id: a48c0a29cbe4  
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence nu  
mber: 31 haPort: 5006  
[oracle@localhost noSQL]$ gedit Hello.java  
[oracle@localhost noSQL]$ javac Hello.java  
[oracle@localhost noSQL]$ java Hello  
Key: /Hello  
Value: Big Data World!  
[oracle@localhost noSQL]$ gedit Keys.java  
[oracle@localhost noSQL]$ javac Keys.java  
[oracle@localhost noSQL]$ java Keys  
/Participant/Mike/-/Answer - This is an answer from Mike  
/Participant/Mike/-/Question - This is a question from Mike  
[oracle@localhost noSQL]$ gedit createTable.sh
```

First we create an operating system directory (highlighted in yellow). Then, we create the necessary Oracle Database directory objects and grant appropriate privileges to the BDA user, which will hold our external table (highlighted in purple). Finally, we issue the command that creates the external table called EXTERNAL_NOSQL (highlighted in green). Please notice the type (*oracle_loader*) and preprocessor used in the CREATE TABLE command; a specific NoSQL preprocessor is used.

```
#!/bin/sh

mkdir -p /u01/nosql/test/exttab/data/

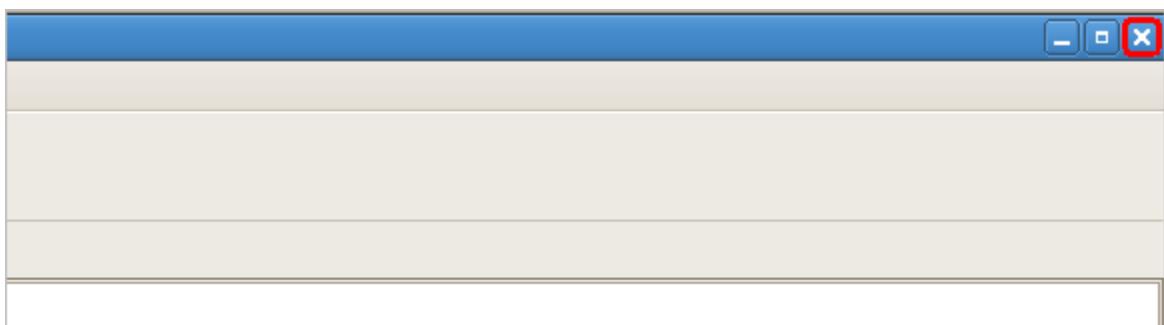
. oraenv <<EOF
orcl
EOF

sqlplus sys/welcome1 as sysdba<<EOF
create or replace directory NOSQL_BIN_DIR as '/u01/nosq'
create or replace directory EXT_TAB as '/u01/nosql/test'
grant read, write, execute on directory NOSQL_BIN_DIR to bda;
grant read, write, execute on directory EXT_TAB to bda;
GRANT EXECUTE ON SYS.UTL_FILE TO BDA;
exit;
EOF

sqlplus bda/welcome1 <<EOF

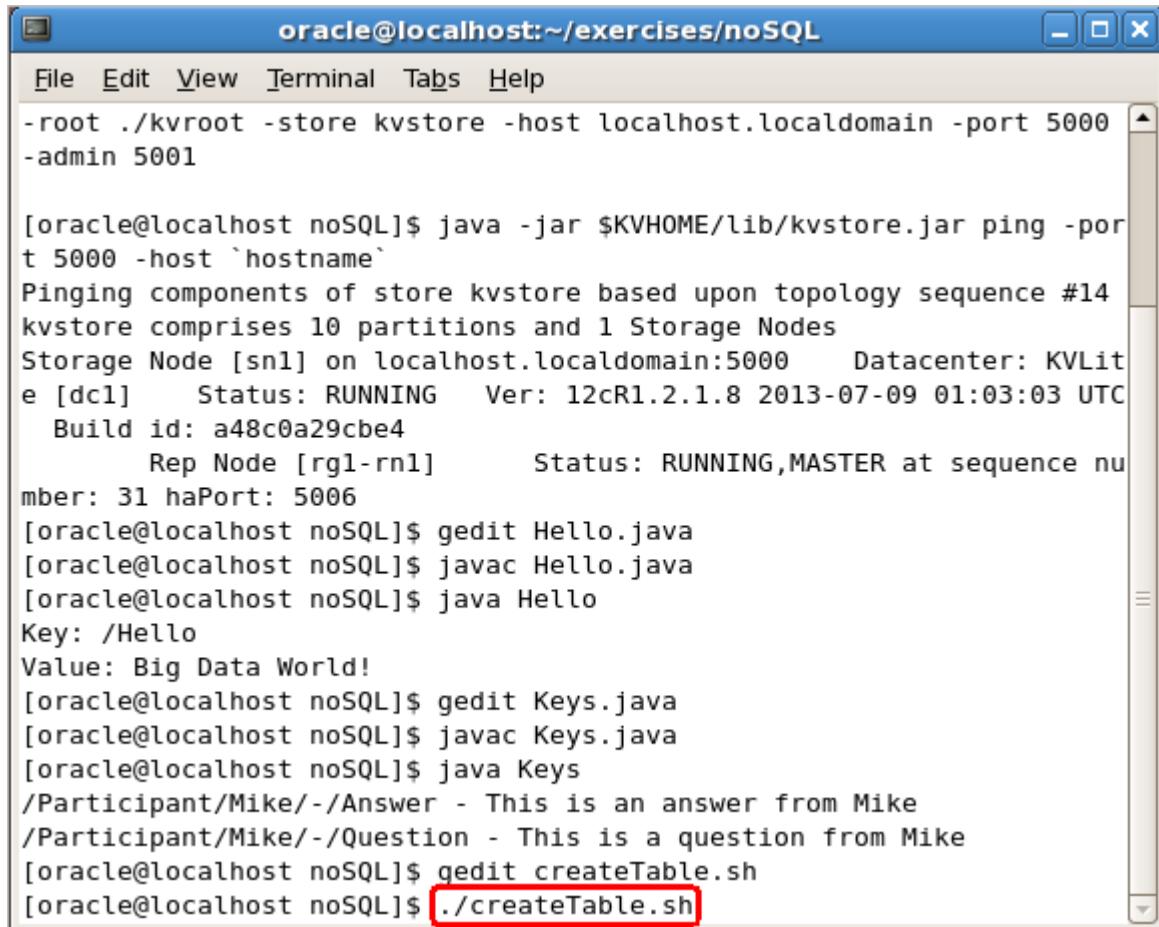
CREATE TABLE external_nosql (name VARCHAR2(30), type VARCHAR2(10))
ORGANIZATION EXTERNAL
  (type oracle_loader
   default directory EXT_TAB
   access parameters (records delimited by newline preprocessor
LOCATION ('nosql.dat'))
   PARALLEL;
exit;
EOF
```

- When you are done evaluating the script, click on the X in the right upper corner of the window to close it.



3. We are now ready to run the script that creates the external table. Go to the terminal and type:

```
./createTable.sh  
Then press Enter
```



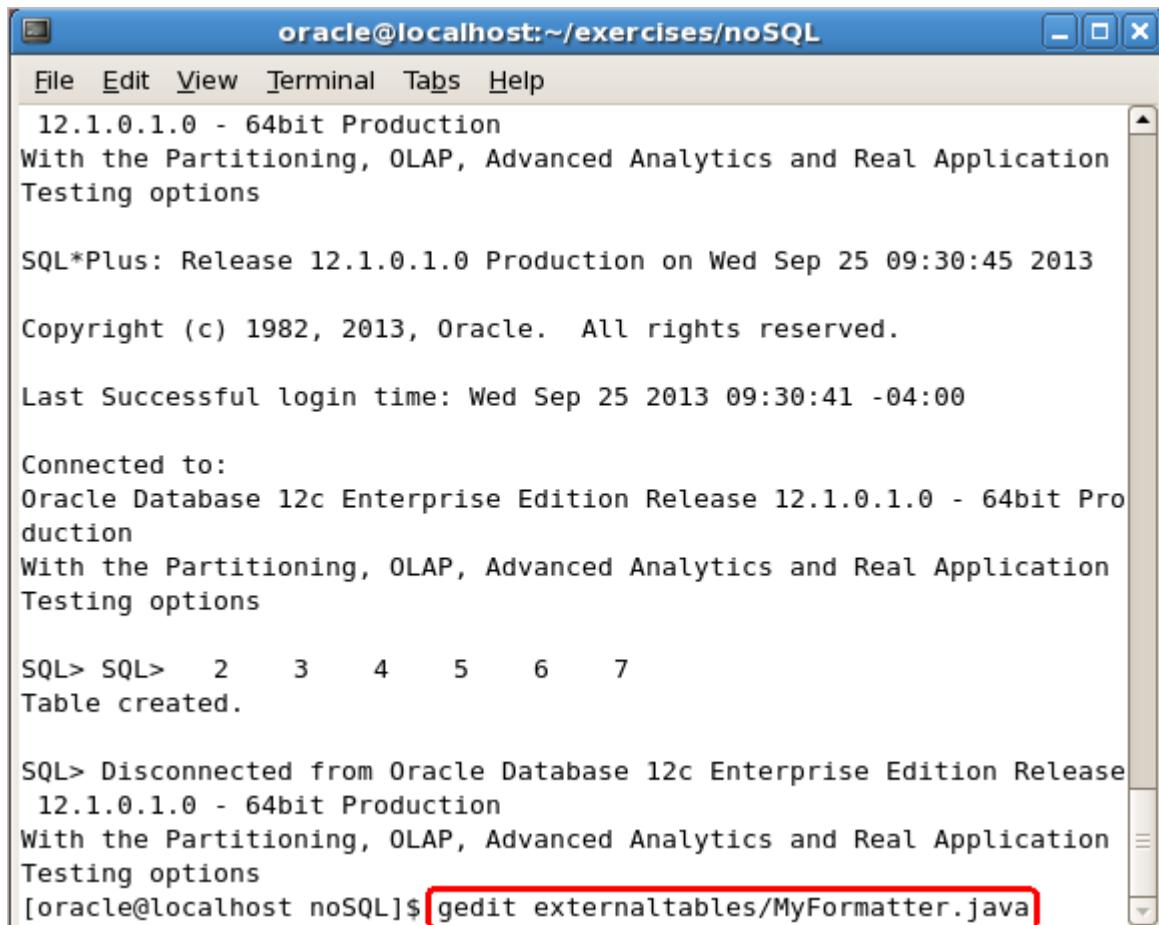
The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains a command-line session. The user has entered the command "java -jar \$KVHOME/lib/kvstore.jar ping -port 5000 -host `hostname`" and received output regarding a storage node's status. The user then enters "java Hello" and sees the output "Key: /Hello" and "Value: Big Data World!". The user edits a Java file "Keys.java", compiles it, and runs it, displaying the output "Keys /Participant/Mike/-/Answer - This is an answer from Mike /Participant/Mike/-/Question - This is a question from Mike". Finally, the user runs the script "createTable.sh", which is highlighted with a red rectangle.

```
-root ./kvroot -store kvstore -host localhost.localdomain -port 5000  
-admin 5001  
  
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar ping -port 5000 -host `hostname`  
Pinging components of store kvstore based upon topology sequence #14  
kvstore comprises 10 partitions and 1 Storage Nodes  
Storage Node [sn1] on localhost.localdomain:5000 Datacenter: KVLite [dcl]  
Status: RUNNING Ver: 12cR1.2.1.8 2013-07-09 01:03:03 UTC  
Build id: a48c0a29cbe4  
Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence number: 31 haPort: 5006  
[oracle@localhost noSQL]$ gedit Hello.java  
[oracle@localhost noSQL]$ javac Hello.java  
[oracle@localhost noSQL]$ java Hello  
Key: /Hello  
Value: Big Data World!  
[oracle@localhost noSQL]$ gedit Keys.java  
[oracle@localhost noSQL]$ javac Keys.java  
[oracle@localhost noSQL]$ java Keys  
/Participant/Mike/-/Answer - This is an answer from Mike  
/Participant/Mike/-/Question - This is a question from Mike  
[oracle@localhost noSQL]$ gedit createTable.sh  
[oracle@localhost noSQL]$ ./createTable.sh
```

4. For this particular exercise we also need a Java Formatter class to be used for converting the K/V pairs from the NoSQL record format to the External Table format. Let's go ahead and review that Java code. Go to the terminal and type:

```
gedit externaltables/MyFormatter.java
```

Then press **Enter**



```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options

SQL*Plus: Release 12.1.0.1.0 Production on Wed Sep 25 09:30:45 2013

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Last Successful login time: Wed Sep 25 2013 09:30:41 -04:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Pro
duction
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options

SQL> SQL> 2 3 4 5 6 7
Table created.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit externaltables/MyFormatter.java
```

Highlighted in yellow are the commands used to get the second item of the Key's major component (majorPath.get(1)) in the *userName* variable, the Key's minor component in the *type* variable and the Value in the *text* variable. The concatenation of these three variables, separated by the pipe sign ("|") is highlighted in purple.

```
public class MyFormatter implements Formatter {

    public MyFormatter() {}

    @Override
    public String toOracleLoaderFormat(final KeyValueVersioned kvv,
                                       final KVStore kvStore) {
        final Key key = kvv.getKey();
        final Value value = kvv.getValue();

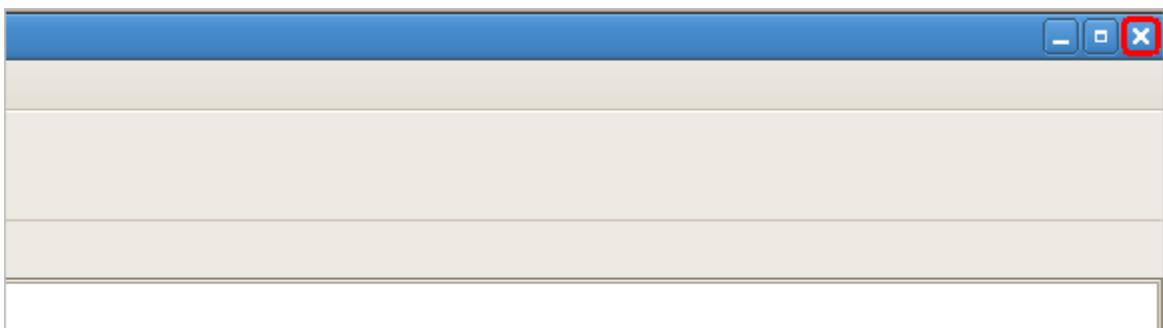
        final List<String> majorPath = key.getMajorPath();
        final List<String> minorPath = key.getMinorPath();

        final String userName = majorPath.get(1);
        final String type = (minorPath.size() > 0) ? minorPath.get(0) : null;
        final String text = new String(value.getValue());

        //try {
        final StringBuilder sb = new StringBuilder();
        sb.append(userName).append('|');
        sb.append(type).append('|');
        sb.append(text);
        return sb.toString();
        //}

        //catch (IOException e) {
        //    throw new RuntimeException(e);
        //}
    }
}
```

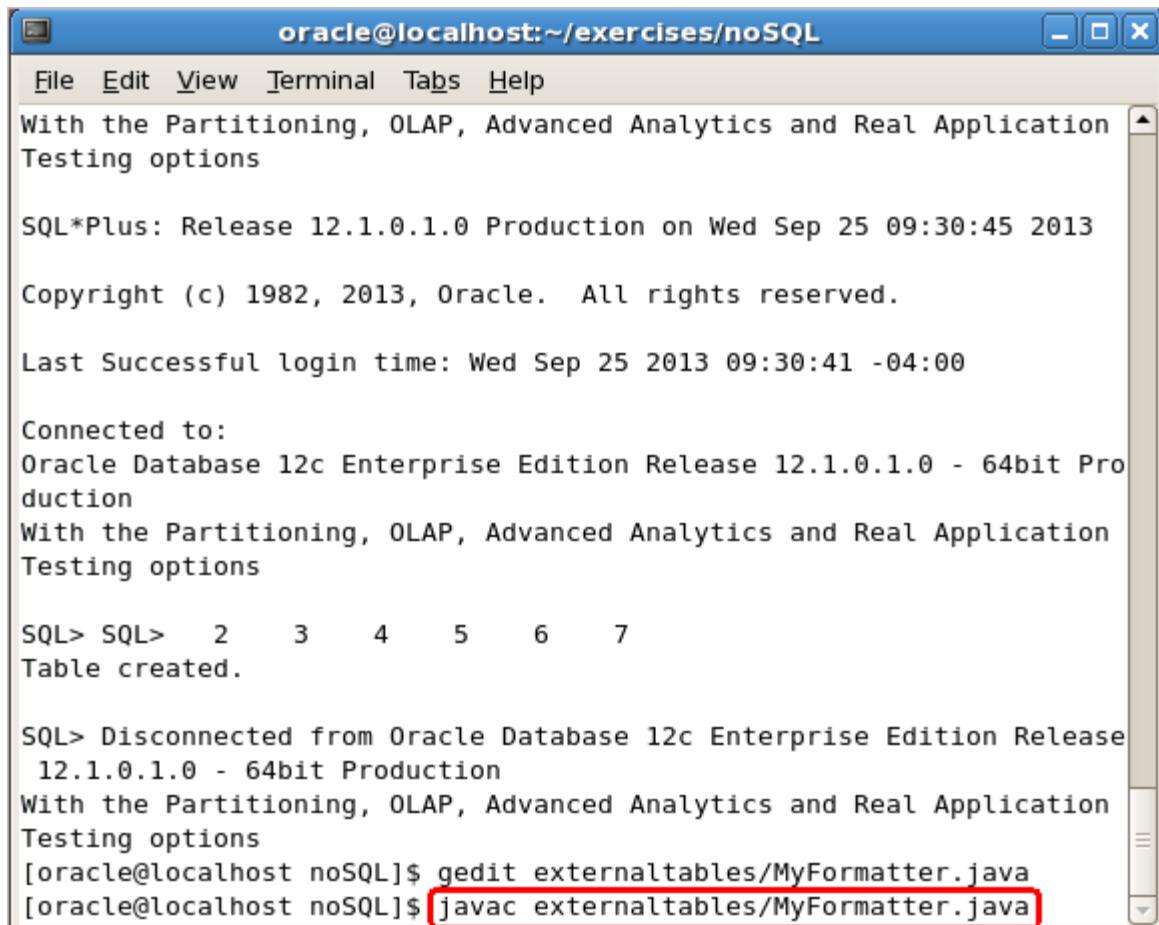
5. When you are done evaluating the code, click on the X in the right upper corner of the window to close it.



6. Let's go ahead and compile that code. Go to the terminal and type:

```
javac externaltables/MyFormatter.java
```

Then press **Enter**



```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options

SQL*Plus: Release 12.1.0.1.0 Production on Wed Sep 25 09:30:45 2013

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Last Successful login time: Wed Sep 25 2013 09:30:41 -04:00

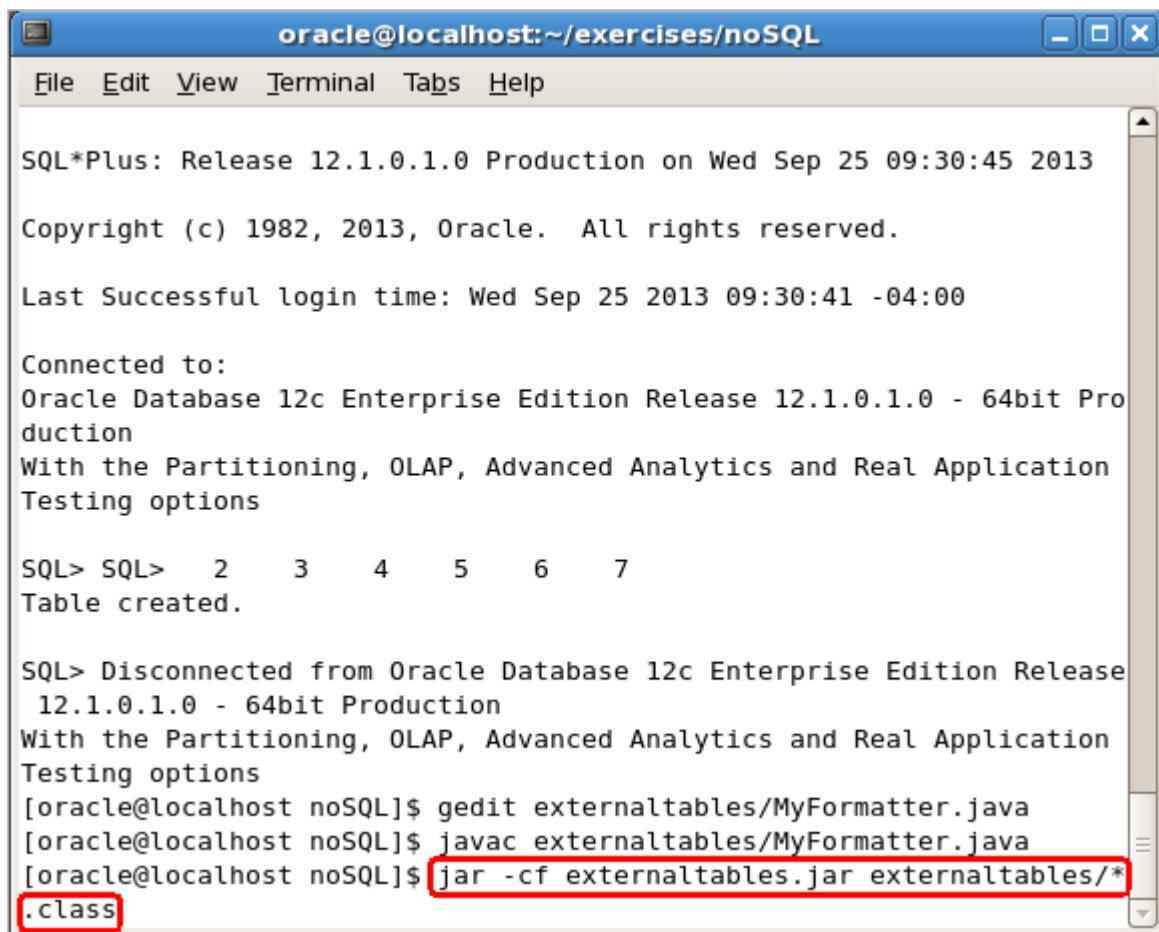
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Pro
duction
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options

SQL> SQL> 2 3 4 5 6 7
Table created.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit externaltables/MyFormatter.java
[oracle@localhost noSQL]$ javac externaltables/MyFormatter.java
```

7. Now that the code is compiled, let's place it in a JAR file. Go to the terminal and type:

```
jar -cf externaltables.jar externaltables/*.class  
Then press Enter
```

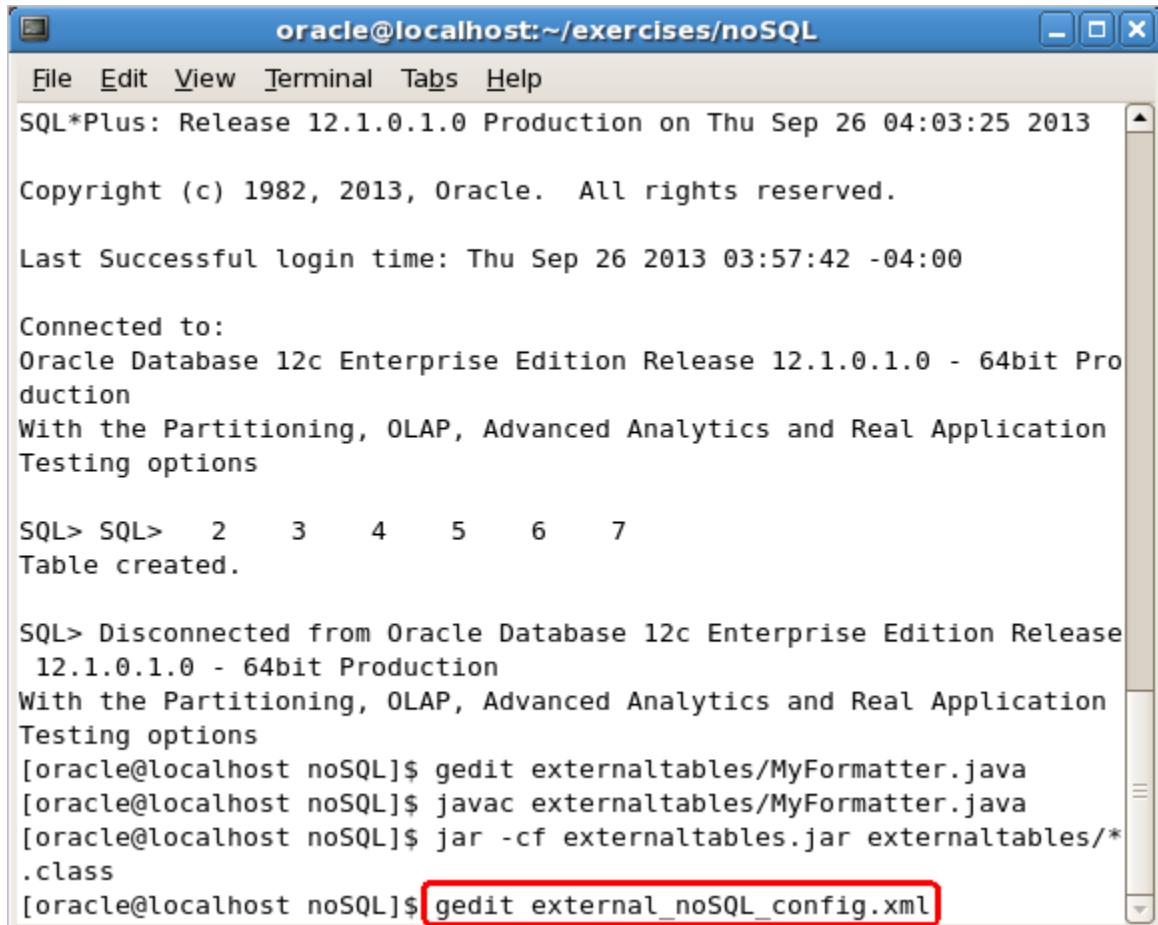


```
oracle@localhost:~/exercises/noSQL  
File Edit View Terminal Tabs Help  
  
SQL*Plus: Release 12.1.0.1.0 Production on Wed Sep 25 09:30:45 2013  
  
Copyright (c) 1982, 2013, Oracle. All rights reserved.  
  
Last Successful login time: Wed Sep 25 2013 09:30:41 -04:00  
  
Connected to:  
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application  
Testing options  
  
SQL> SQL> 2 3 4 5 6 7  
Table created.  
  
SQL> Disconnected from Oracle Database 12c Enterprise Edition Release  
12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application  
Testing options  
[oracle@localhost noSQL]$ gedit externaltables/MyFormatter.java  
[oracle@localhost noSQL]$ javac externaltables/MyFormatter.java  
[oracle@localhost noSQL]$ jar -cf externaltables.jar externaltables/*  
.class
```

8. To connect the external table to the NoSQL Database, we will have to run a publish command. This command makes use of an XML configuration file. Let's review that file first. Go to the terminal and type:

```
gedit external_noSQL_config.xml
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains the following text:

```
File Edit View Terminal Tabs Help
SQL*Plus: Release 12.1.0.1.0 Production on Thu Sep 26 04:03:25 2013
Copyright (c) 1982, 2013, Oracle. All rights reserved.

Last Successful login time: Thu Sep 26 2013 03:57:42 -04:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options

SQL> SQL> 2 3 4 5 6 7
Table created.

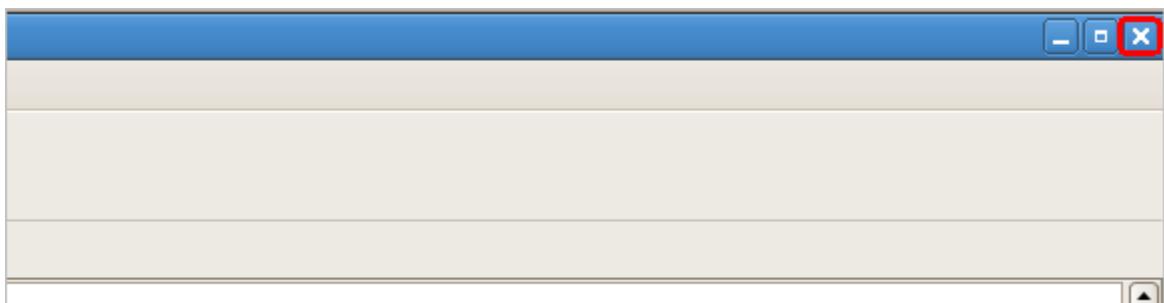
SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit externaltables/MyFormatter.java
[oracle@localhost noSQL]$ javac externaltables/MyFormatter.java
[oracle@localhost noSQL]$ jar -cf externaltables.jar externaltables/*
.class
[oracle@localhost noSQL]$ gedit external_noSQL_config.xml
```

The last command, "gedit external_noSQL_config.xml", is highlighted with a red rectangle.

This file contains the name of the Oracle external table (EXTERNAL_NOSQL), the Parent Key of the NoSQL records (the first item in the Key's major component) and the name of the formatter class used to format the NoSQL records (as highlighted in yellow).

```
<property name="oracle.kv.exttab.connection.url"
          value="jdbc:oracle:thin:@//localhost:1521/orcl"
          type="STRING"/>
<property name="oracle.kv.exttab.connection.user"
          value="BDA" type="STRING"/>
<property name="oracle.kv.exttab.tableName" value="external"
          type="STRING"/>
</component>
<component name="nosql_stream" type="params">
    <!-- Fill in appropriate values for
        oracle.kv.kvstore and oracle.kv.hosts
    -->
    <property name="oracle.kv.kvstore"
              value="kvstore"
              type="STRING"/>
    <property name="oracle.kv.hosts"
              value="localhost:5000"
              type="STRING"/>
    <property name="oracle.kv.batchSize"
              value="100"
              type="INT"/>
    <property name="oracle.kv.depth"
              value="PARENT_AND_DESCENDANTS"
              type="STRING"/>
    <property name="oracle.kv.parentKey"
              value="/Participant"
              type="STRING"/>
    <property name="oracle.kv.formatterClass"
              value="externaltables.MyFormatter"
              type="STRING"/>
```

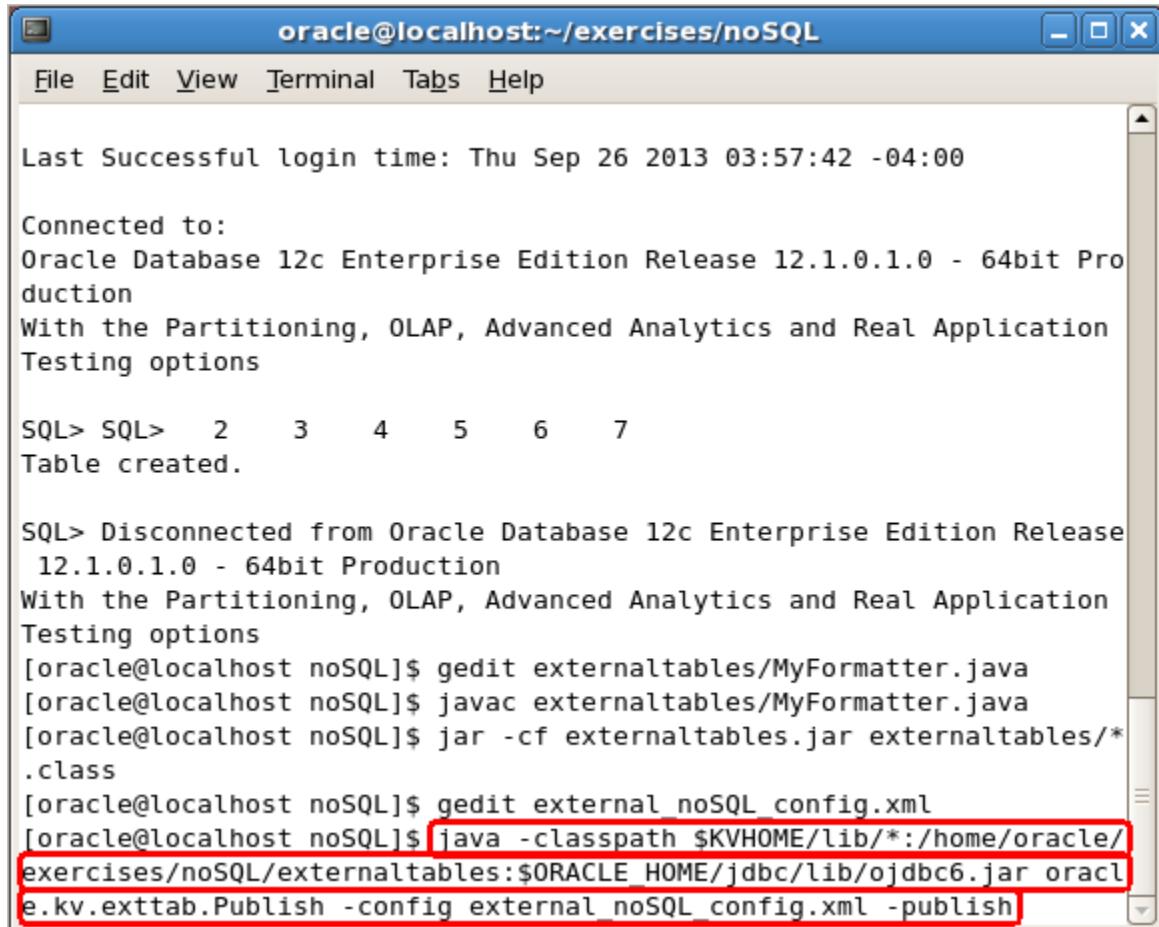
- When you're done reviewing the file click the X in the top right corner of the window to close it.



10. Let's now run the publish command that connects the Oracle external table to the NoSQL data. Go to the terminal and type:

```
java -classpath  
$KVHOME/lib/*:/home/oracle/exercises/noSQL/externaltables:$ORACLE_HOME/jd  
bc/lib/ojdbc6.jar oracle.kv.exttab.Publish -config  
external_noSQL_config.xml -publish
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains the following text:

```
Last Successful login time: Thu Sep 26 2013 03:57:42 -04:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options

SQL> SQL> 2 3 4 5 6 7
Table created.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit externaltables/MyFormatter.java
[oracle@localhost noSQL]$ javac externaltables/MyFormatter.java
[oracle@localhost noSQL]$ jar -cf externaltables.jar externaltables/*
.class
[oracle@localhost noSQL]$ gedit external_noSQL_config.xml
[oracle@localhost noSQL]$ java -classpath $KVHOME/lib/*:/home/oracle/
exercises/noSQL/externaltables:$ORACLE HOME/jdbc/lib/ojdbc6.jar oracle.
kv.exttab.Publish -config external_noSQL config.xml -publish
```

The last line of the command is highlighted with a red rectangle.

11. You will be asked to enter a password for the code to be able to login to the database user. Enter the following information

[Enter Database Password:] welcome1
Then Press **Enter**

NOTE: No text will appear while you type

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains the following text:

```
File Edit View Terminal Tabs Help
Last Successful login time: Thu Sep 26 2013 03:57:42 -04:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options

SQL> SQL> 2 3 4 5 6 7
Table created.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit externaltables/MyFormatter.java
[oracle@localhost noSQL]$ javac externaltables/MyFormatter.java
[oracle@localhost noSQL]$ jar -cf externaltables.jar externaltables/*
.class
[oracle@localhost noSQL]$ gedit external_noSQL_config.xml
[oracle@localhost noSQL]$ java -classpath $KVHOME/lib/*:/home/oracle/
exercises/noSQL/externaltables:$ORACLE_HOME/jdbc/lib/ojdbc6.jar oracle.
kv.exttab.Publish -config external_noSQL_config.xml -publish
[Enter Database Password:] welcome1
```

12. We will now be able to query the external table. We have a script that does just that. Let's review it first, so go to the terminal and type:

```
gedit viewData.sh
```

Then press **Enter**

```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> SQL> 2 3 4 5 6 7
Table created.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
[oracle@localhost noSQL]$ gedit externaltables/MyFormatter.java
[oracle@localhost noSQL]$ javac externaltables/MyFormatter.java
[oracle@localhost noSQL]$ jar -cf externaltables.jar externaltables/*
.class
[oracle@localhost noSQL]$ gedit external_noSQL_config.xml
[oracle@localhost noSQL]$ java -classpath $KVHOME/lib/*:/home/oracle/exercises/noSQL/externaltables:$ORACLE_HOME/jdbc/lib/ojdbc6.jar oracle.kv.extbl.Publish -config external_noSQL_config.xml -publish
[Enter Database Password:]
Publish was successful.
[oracle@localhost noSQL]$ gedit viewData.sh
```

The script performs a simple select on the BDA.EXTERNAL_NOSQL table that we've just created.

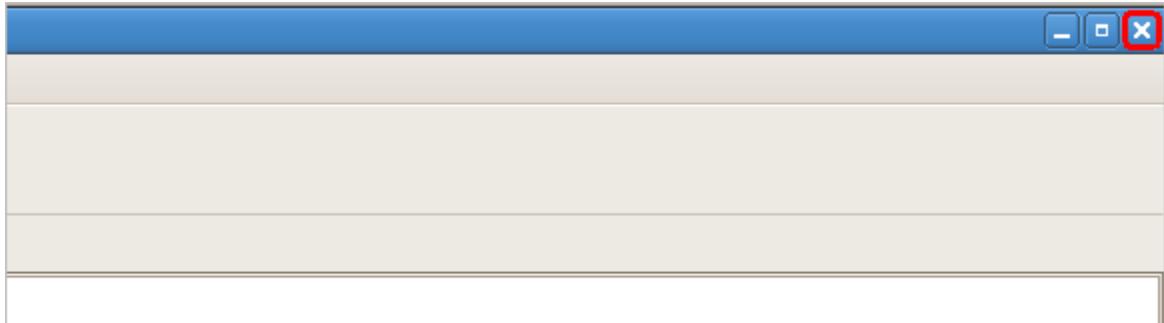
```
#!/bin/sh

. oraenv <<EOF
orcl
EOF

sqlplus sys/welcomel as sysdba<<EOF
set linesize 100
SELECT * FROM BDA.EXTERNAL_NOSQL;
exit;

EOF
```

13. When you're done reviewing the script click the **X** in the top right corner of the window to close it.



14. Let's run the script that queries the table. Go to the terminal and type:

```
./viewData.sh
```

Then press **Enter**

```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> SQL> 2 3 4 5 6 7
Table created.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
[oracle@localhost noSQL]$ gedit externaltables/MyFormatter.java
[oracle@localhost noSQL]$ javac externaltables/MyFormatter.java
[oracle@localhost noSQL]$ jar -cf externaltables.jar externaltables/*
.class
[oracle@localhost noSQL]$ gedit external_noSQL_config.xml
[oracle@localhost noSQL]$ java -classpath $KVHOME/lib/*:/home/oracle/exercises/noSQL/externaltables:$ORACLE_HOME/jdbc/lib/ojdbc6.jar oracle.kv.extab.Publish -config external_noSQL_config.xml -publish
[Enter Database Password:]
Publish was successful.
[oracle@localhost noSQL]$ gedit viewData.sh
[oracle@localhost noSQL]$ ./viewData.sh
```

The records generated by the Java program that was run a few steps ago to insert some data into the NoSQL Database are visible from the Oracle database, as highlighted below.

```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> SQL>
NAME          TYPE          TEXT
-----
-----
Mike          Answer        This is an answer
from Mike
Mike          Question      This is a question
n from Mike
Dave          Answer        This is an answer
from Dave
Dave          Question      This is a question
n from Dave

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$
```

4.6 NoSQL and our Transactional Data

1. Now that we have an idea about how we can interface with the NoSQL database, let's proceed to process the data that we need for our analysis. First, we will load some data into the NoSQL Database. Normally, the transaction-related data that we are about to process should be loaded in a real-time manner from the production environment. In our case, we will perform the load ourselves, as our data is stored in a CSV file.

Before loading the data, however, let's have a look at its structure. With version 2.0 of the Oracle NoSQL Database we have integrated AVRO serialization into our database. This has greatly increased the flexibility of what you can store as the Value of a Key/Value pair. With AVRO you can now define an entire schema which will be stored in the value field. To view the AVRO schema that will hold our transactional data, go to the terminal and type:

```
gedit trans.avsc  
Then press Enter
```

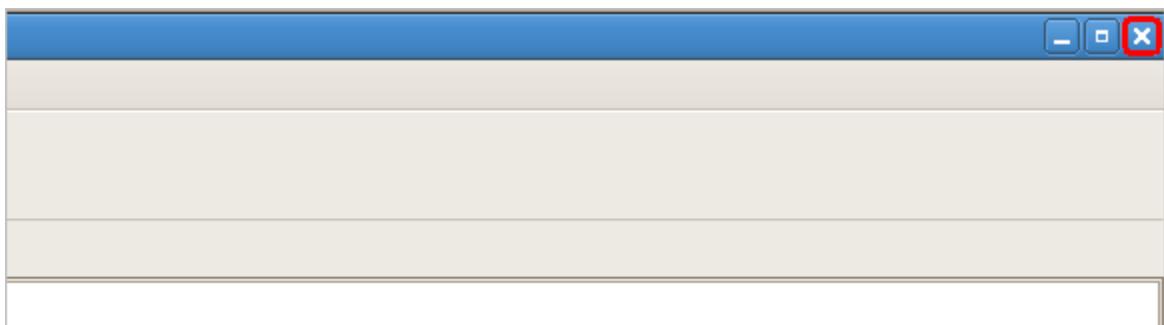
NAME	TYPE	TEXT
Mike	Answer	This is an answer
from Mike		
Mike	Question	This is a question
n from Mike		
Dave	Answer	This is an answer
from Dave		
Dave	Question	This is a question
n from Dave		

```
SQL> Disconnected from Oracle Database 12c Enterprise Edition Release  
12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application  
Testing options  
[oracle@localhost noSQL]$ gedit trans.avsc
```

Our schema's name is *trans*, highlighted in yellow in the screenshot below. The entire declaration of a field is highlighted in green. We have an integer-type field called *teller*, which holds the value 1 if the transaction is of type teller or 0 if not. This field is followed by three other integer-type fields called *kiosk*, *atm* and *webbank*. The same rule as the above also applies to these three fields. Finally, we have the *sum* field, of type float, that holds the sum of the transaction.

```
{  
  "type" : "record",  
  "name" : "trans",  
  "fields" :  
    [ {"name" : "teller", "type" : "int", "default" :0 },  
      {"name" : "kiosk", "type" : "int", "default" :0 },  
      {"name" : "atm", "type" : "int", "default" :0 },  
      {"name" : "webbank", "type" : "int", "default" :0 }  
      {"name" : "sum", "type" : "float", "default" :0.0 }  
    ]  
}
```

2. When you are done evaluating the file click on the X in the right upper corner of the window to close it.



3. Now that we've reviewed the schema, let's attach it to the NoSQL Database. First we have to open the NoSQL Database administration console. Go to the terminal and type:

```
java -jar $KVHOME/lib/kvstore.jar runadmin -port 5000 -host `hostname`
```

Then press **Enter**

```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
duction
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options

SQL> SQL>
NAME          TYPE          TEXT
-----
Mike          Answer        This is an answer
 from Mike
Mike          Question      This is a question
n from Mike
Dave          Answer        This is an answer
 from Dave
Dave          Question      This is a question
n from Dave

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit trans.avsc
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin
-port 5000 -host `hostname`
```

4. Now let's run the command that adds the schema to the NoSQL Database. Go to the terminal and type:

```
ddl add-schema -file trans.avsc
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window displays a table of data with columns "NAME", "TYPE", and "TEXT". Below the table, it shows the SQL prompt "SQL> SQL>" followed by disconnected information and command history.

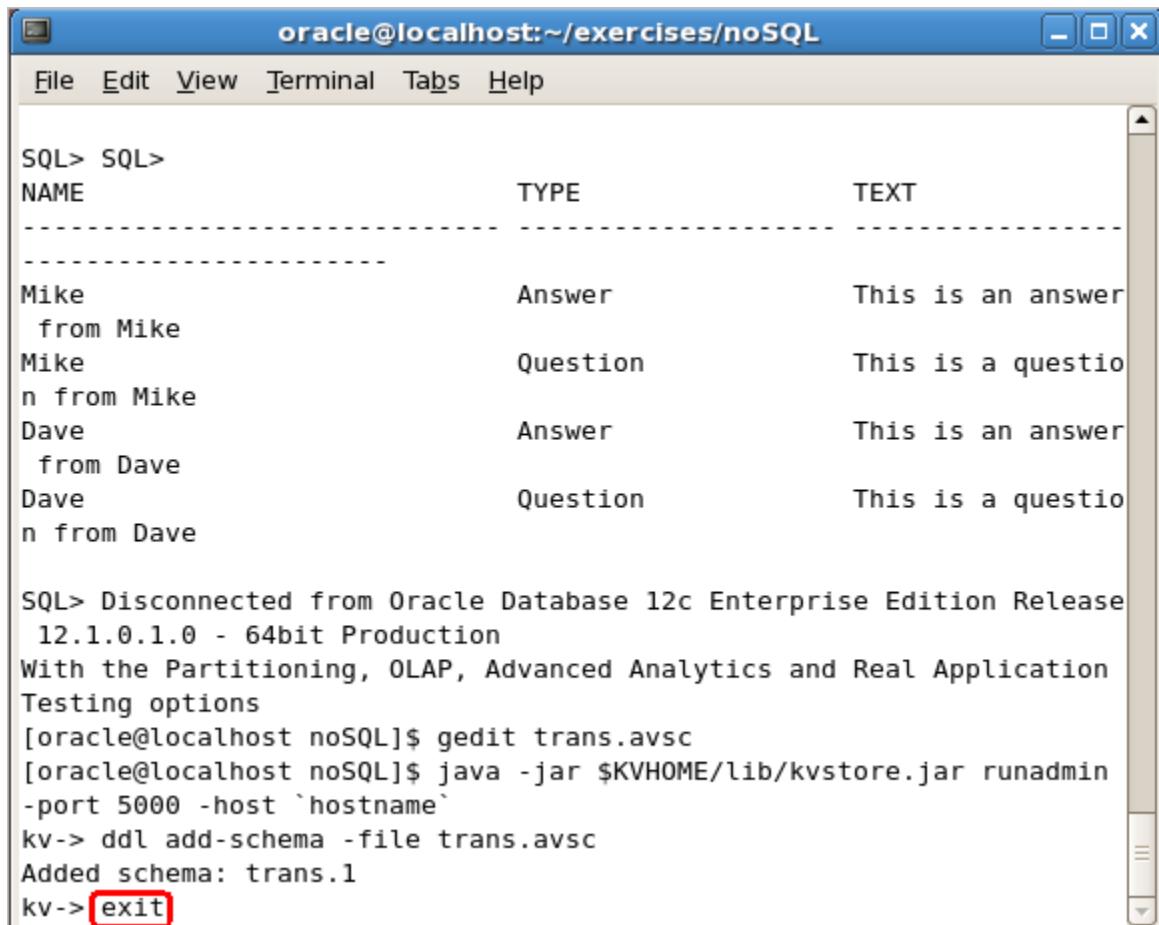
NAME	TYPE	TEXT
Mike	Answer	This is an answer
from Mike		
Mike	Question	This is a question
n from Mike		
Dave	Answer	This is an answer
from Dave		
Dave	Question	This is a question
n from Dave		

```
SQL> Disconnected from Oracle Database 12c Enterprise Edition Release  
12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application  
Testing options  
[oracle@localhost noSQL]$ gedit trans.avsc  
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin  
-port 5000 -host `hostname`  
kv-> ddl add-schema -file trans.avsc
```

5. Now that we've added the schema, we can exit the NoSQL administration console. Go to the terminal and type:

```
exit
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains the following text:

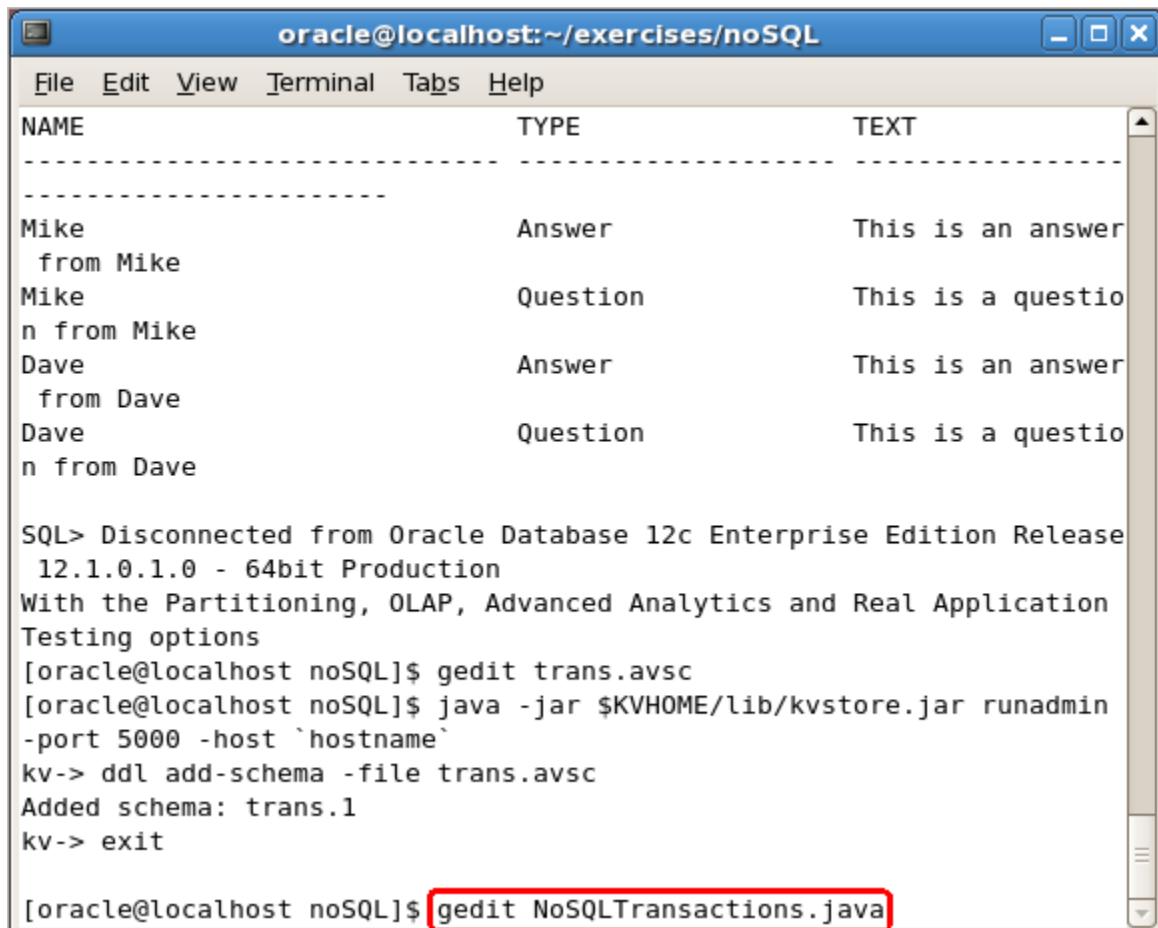
```
SQL> SQL>
NAME          TYPE      TEXT
-----
-----
Mike          Answer    This is an answer
from Mike
Mike          Question   This is a question
n from Mike
Dave          Answer    This is an answer
from Dave
Dave          Question   This is a question
n from Dave

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit trans.avsc
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin
-port 5000 -host `hostname`
kv-> ddl add-schema -file trans.avsc
Added schema: trans.1
kv-> exit
```

6. The schema is now added, we can proceed to load the data into the NoSQL Database. Before actually doing that, let's review the code we will use to perform this action. Go to the terminal and type:

```
gedit NoSQLTransactions.java
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains a table with three columns: NAME, TYPE, and TEXT. Below the table, there is a series of database commands and their output.

NAME	TYPE	TEXT
Mike	Answer	This is an answer
from Mike		
Mike	Question	This is a question
n from Mike		
Dave	Answer	This is an answer
from Dave		
Dave	Question	This is a question
n from Dave		

```
SQL> Disconnected from Oracle Database 12c Enterprise Edition Release  
12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application  
Testing options  
[oracle@localhost noSQL]$ gedit trans.avsc  
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin  
-port 5000 -host `hostname`  
kv-> ddl add-schema -file trans.avsc  
Added schema: trans.1  
kv-> exit  
  
[oracle@localhost noSQL]$ gedit NoSQLTransactions.java
```

In the code you will see how to interact with this AVRO schema. In yellow we highlighted how to configure the interface to the schema from the code. In green is the code on how to fill data into the data structure. In purple there is the typical code for inserting the data into the database and finally in blue is the code for retrieving and displaying the data inside the database. Notice that the key has a major and a minor component. The major component is comprised of two items: the string "Load1" and the first column in the csv file (the customer id). The minor component is an automatically generated string like trans1, trans2, etc.

```

File f = new File("trans.avsc");
Schema.Parser parser = new Schema.Parser();
parser.parse(f);
Schema infoSchema = parser.getTypes().get("trans");
GenericAvroBinding binding = catalog.getGenericBinding(infoSchema);
//load Data
GenericRecord infoRecord = new GenericData.Record(infoSchema);
BufferedReader br = new BufferedReader(new FileReader("exp1.csv"));
int i = 0;
String line = null;
while ((line = br.readLine()) != null){
    String[] tokens = line.split(",");
    infoRecord.put("teller", Integer.parseInt(tokens[0]));
    infoRecord.put("kiosk", Integer.parseInt(tokens[1]));
    infoRecord.put("atm", Integer.parseInt(tokens[2]));
    infoRecord.put("webbank", Integer.parseInt(tokens[3]));
    infoRecord.put("sum", Float.parseFloat(tokens[5]));
}
//store in database
Value value = binding.toValue(infoRecord);
List<String> majorComponent = Arrays.asList("Load1", "CU9988");
store.put(Key.createKey(majorComponent, "trans"+(i+1)));
}

```

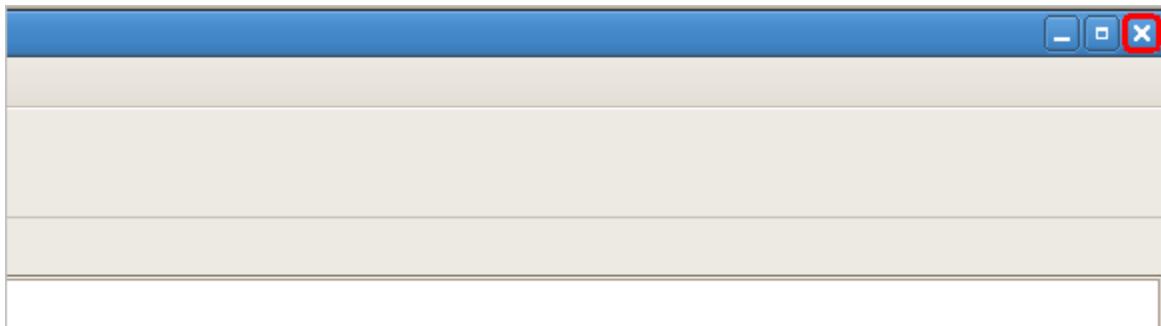
Only the transactions corresponding to customer id CU9988 are retrieved from the database and will be displayed in the terminal when running this Java program.

```

//retrieve from database
Iterator<KeyValueVersion> MyStoreIterator = store.storeIterator();
while (MyStoreIterator.hasNext())
{
    KeyValueVersion entry = MyStoreIterator.next();
    Key k = entry.getKey();
    GenericRecord outputRecord = new GenericData.Record(infoSchema);
    outputRecord = binding.toObject(entry.getValue());
    System.out.println(k.toString() + " - " + outputRecord);
}
store.close();

```

7. When you are done evaluating the code click on the **X** in the right upper corner of the window to close it.



8. First, we have to compile the code. Go to the terminal and type:

```
javac NoSQLTransactions.java
```

Then press **Enter**

```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
-----
-----
Mike           Answer      This is an answer
 from Mike
Mike           Question    This is a question
n from Mike
Dave           Answer      This is an answer
 from Dave
Dave           Question    This is a question
n from Dave

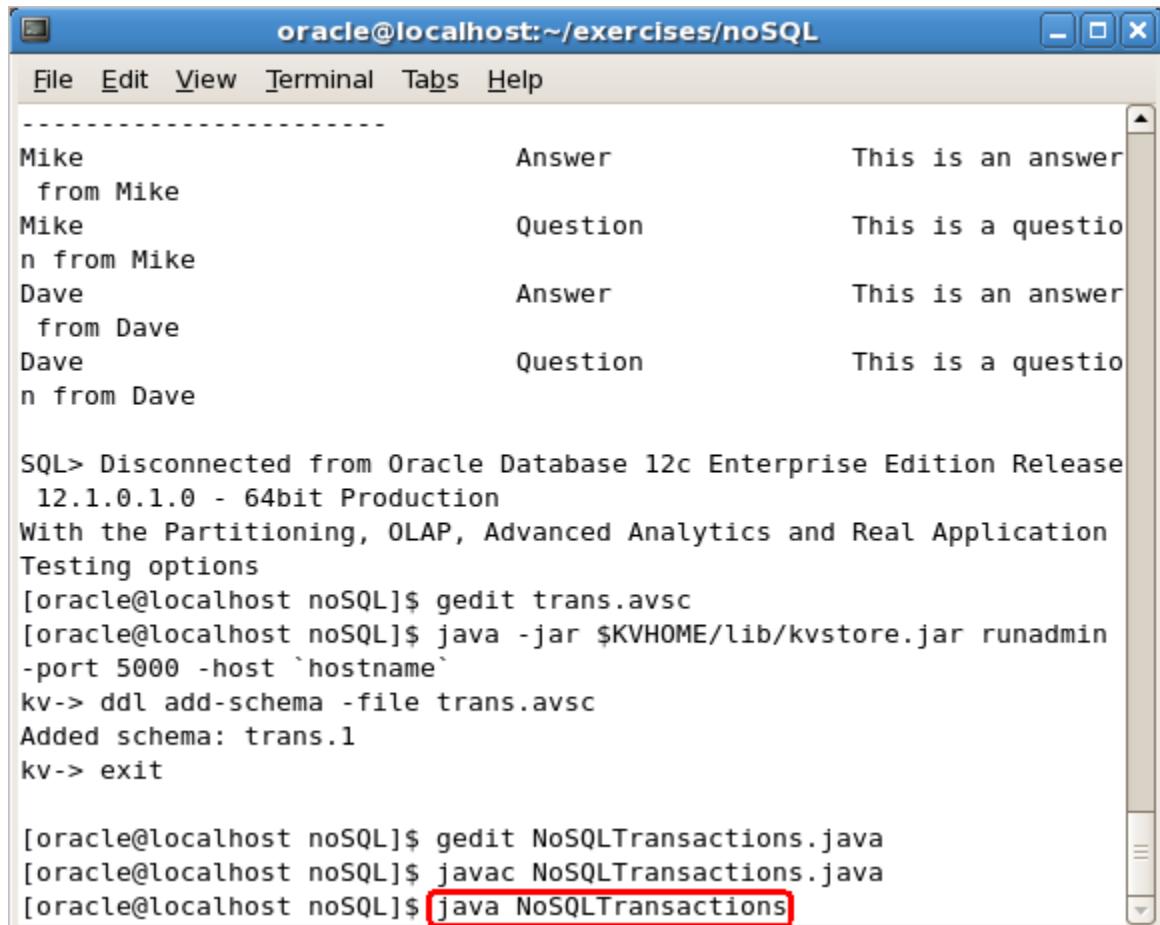
SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit trans.avsc
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin
-port 5000 -host `hostname`
kv-> ddl add-schema -file trans.avsc
Added schema: trans.1
kv-> exit

[oracle@localhost noSQL]$ gedit NoSQLTransactions.java
[oracle@localhost noSQL]$ javac NoSQLTransactions.java
```

9. We are now ready to run the Java program. Go to the terminal and type:

```
java NoSQLTransactions
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains the following text:

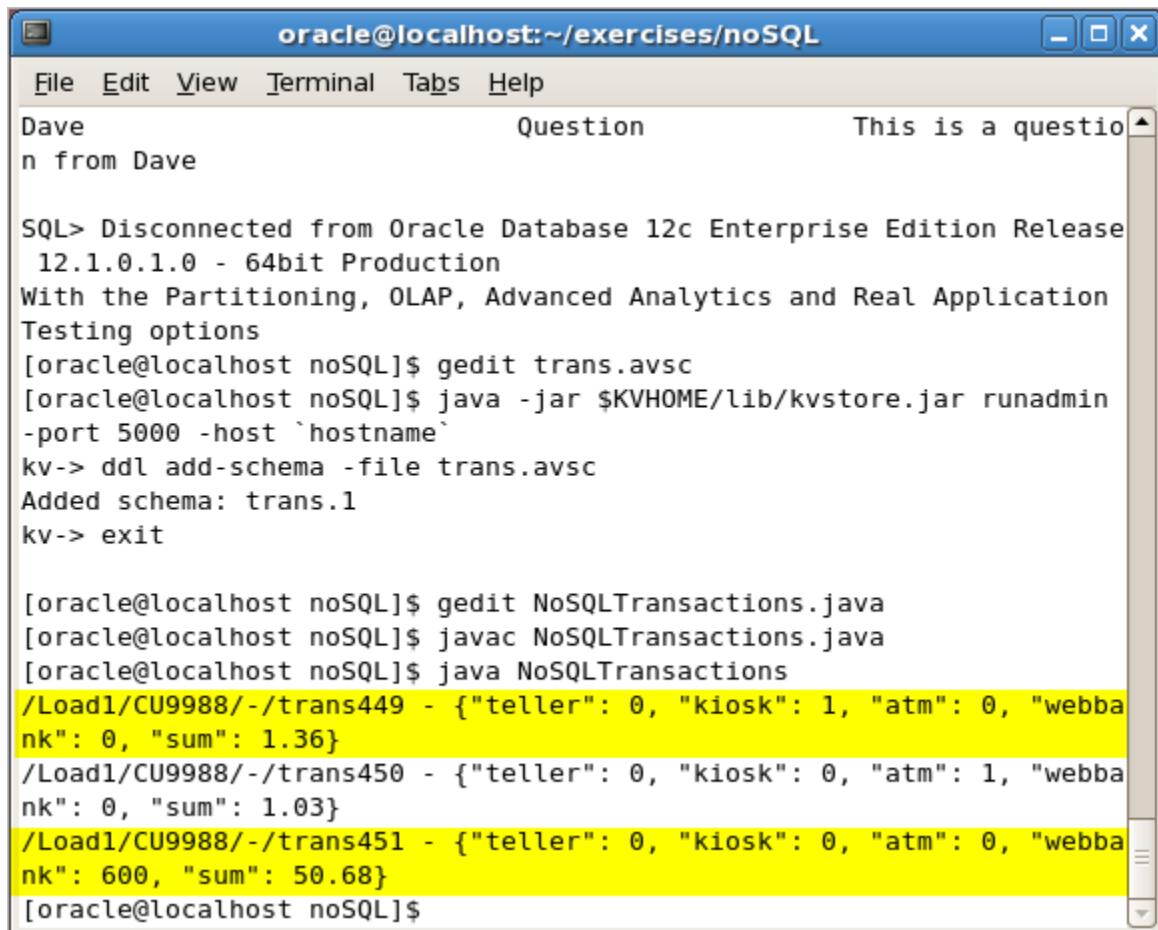
```
File Edit View Terminal Tabs Help
-----
Mike Answer This is an answer
 from Mike
Mike Question This is a question
n from Mike
Dave Answer This is an answer
 from Dave
Dave Question This is a question
n from Dave

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit trans.avsc
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin
-port 5000 -host `hostname`
kv-> ddl add-schema -file trans.avsc
Added schema: trans.1
kv-> exit

[oracle@localhost noSQL]$ gedit NoSQLTransactions.java
[oracle@localhost noSQL]$ javac NoSQLTransactions.java
[oracle@localhost noSQL]$ java NoSQLTransactions
```

The command "java NoSQLTransactions" is highlighted with a red rectangle.

The data is inserted into the NoSQL Database, and then the transactions corresponding to customer id CU9988 are retrieved from the database and displayed in the terminal, as highlighted in the image below.



A screenshot of a terminal window titled "oracle@localhost:~/exercises/noSQL". The window has a menu bar with File, Edit, View, Terminal, Tabs, and Help. There are three tabs visible: "Dave", "Question", and "This is a question". The "Dave" tab contains the following text:

```
Dave
n from Dave
```

The "Question" tab contains the following text:

```
SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit trans.avsc
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin
-port 5000 -host `hostname`
kv-> ddl add-schema -file trans.avsc
Added schema: trans.1
kv-> exit
```

The "This is a question" tab contains the following text:

```
[oracle@localhost noSQL]$ gedit NoSQLTransactions.java
[oracle@localhost noSQL]$ javac NoSQLTransactions.java
[oracle@localhost noSQL]$ java NoSQLTransactions
/Load1/CU9988/-/trans449 - {"teller": 0, "kiosk": 1, "atm": 0, "webba
nk": 0, "sum": 1.36}
/Load1/CU9988/-/trans450 - {"teller": 0, "kiosk": 0, "atm": 1, "webba
nk": 0, "sum": 1.03}
/Load1/CU9988/-/trans451 - {"teller": 0, "kiosk": 0, "atm": 0, "webba
nk": 600, "sum": 50.68}
[oracle@localhost noSQL]$
```

The lines starting with "/Load1/CU9988/-/" are highlighted in yellow.

10. For our analysis we need an aggregated view of the transactions for each customer. Our final goal is to load the aggregated data into the Oracle Database using the Oracle Loader for Hadoop (this will be done in a later exercise). To aggregate the transactions, we will use a Hadoop Map-Reduce job that interacts with the NoSQL data, performs the processing and stores the results into an HDFS file. Let's review the Java code that does this. Go to the terminal and type:

```
gedit Hadoop.java
```

Then press **Enter**

```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
Dave Question This is a question from Dave

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit trans.avsc
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin
-port 5000 -host `hostname`
kv-> ddl add-schema -file trans.avsc
Added schema: trans.1
kv-> exit

[oracle@localhost noSQL]$ gedit NoSQLTransactions.java
[oracle@localhost noSQL]$ javac NoSQLTransactions.java
[oracle@localhost noSQL]$ java NoSQLTransactions
/Load1/CU9988/-/trans449 - {"teller": 0, "kiosk": 1, "atm": 0, "webba
nk": 0, "sum": 1.36}
/Load1/CU9988/-/trans450 - {"teller": 0, "kiosk": 0, "atm": 1, "webba
nk": 0, "sum": 1.03}
/Load1/CU9988/-/trans451 - {"teller": 0, "kiosk": 0, "atm": 0, "webba
nk": 600, "sum": 50.68}
[oracle@localhost noSQL]$ gedit Hadoop.java
```

The Mapper class includes a filter in order to select only the Keys in the NoSQL Database corresponding to our customer IDs (CU*). This is highlighted in yellow. It then stores the 5 fields from the Value part of the NoSQL records into an array of floats. This is highlighted in blue. Finally, it sends this information grouped by customer id to the Reducer through the context.write command. This is highlighted in purple.

```
public static class MyMapper extends Mapper<Key, IndexedRecord, Text, FloatWritable>

    public void map(Key key, IndexedRecord value, Context context
                    throws IOException, InterruptedException {
        String customerId = new String(key.getMajorKey());
        if (customerId.matches("(\\D{2})(\\d*)"))
        {
            FloatArrayWritable data;
            data = new FloatArrayWritable();
            FloatWritable[] buff = new FloatWritable[5];
            for(int i=0;i<numField;i++)
            {
                buff[i]= new FloatWritable();
            }
            data.set(buff);
            context.write(new Text(customerId), data);
        }
    }
}
```

The reducer performs a sum of every component field of the Value (previously stored in the array of floats), for each customer id. This is highlighted in yellow. Then, it creates the output records to be stored in the HDFS file. The teller, kiosk, atm and webbank values are actually integers, not float values, so a conversion is performed for these four values before writing them to the output. Only the sum (the last field) is a float value. This is highlighted in blue.

Finally, the records are passed forward to be written into the HDFS output file. This is highlighted in purple.

The resulting records will contain an aggregated view of the customers' transactions. For each customer, we will see the number of teller, kiosk, atm and webbank transactions, along with the total sum of all transactions. We've selected a sample of 1015 customers, so our resulting HDFS file will contain 1015 lines, each line corresponding to a customer.

```
public static class MyReducer extends Reducer<Text, FloatArrayWritable, Text, Text>
    public void reduce(Text key, Iterable<FloatArrayWritable> values, Context context)
        throws IOException, InterruptedException {
        float sums[] = new float[numField];
        for(FloatArrayWritable aValue: values) {
            for(int i=0;i<numField;i++) {
                sums[i]+=(float)aValue.get(i);
            }
        }
        String output="";
        for(int i=0;i<numField;i++) {
            if (i<numField-1)
                output+=(int)sums[i]+",";
            else
                output+=sums[i];
        }
        context.write(key, new Text(output));
    }
}
```

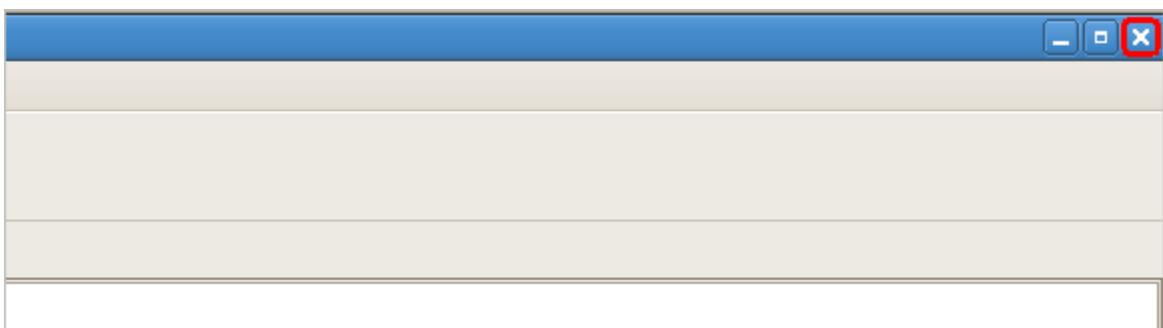
The main function sets the parameters for the Map/Reduce job. We've highlighted a few in the screenshot from below. The line highlighted in yellow shows the separator of the fields in the output HDFS file (","). The lines highlighted in blue show the connection to the NoSQL Database and the line highlighted in green shows the Parent Key of the NoSQL records to be processed: "Load1". We chose this string as the first item of the Key's major component for our transactional data, when we loaded the records into the NoSQL Database.

```
public static int main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set("mapred.textoutputformat.separator", ",");
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    Job job = new Job(conf, "Customer Sums");
    job.setJarByClass(Hadoop.class);
    job.setMapperClass(MyMapper.class);
    job.setReducerClass(MyReducer.class);

    job.setInputFormatClass(KVAvroInputFormat.class);
    KVAvroInputFormat.setKVHelperHosts(new String[] {"localhost"});
    KVAvroInputFormat.setKVStoreName("kvstore");
    KVAvroInputFormat.setParentKey(Key.createKey("Load1"));
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(FloatArrayWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    FileOutputFormat.setOutputPath(job, new Path(otherArgs[0]));
    boolean success = job.waitForCompletion(true);
    return success ? 0 : 1;
}
```

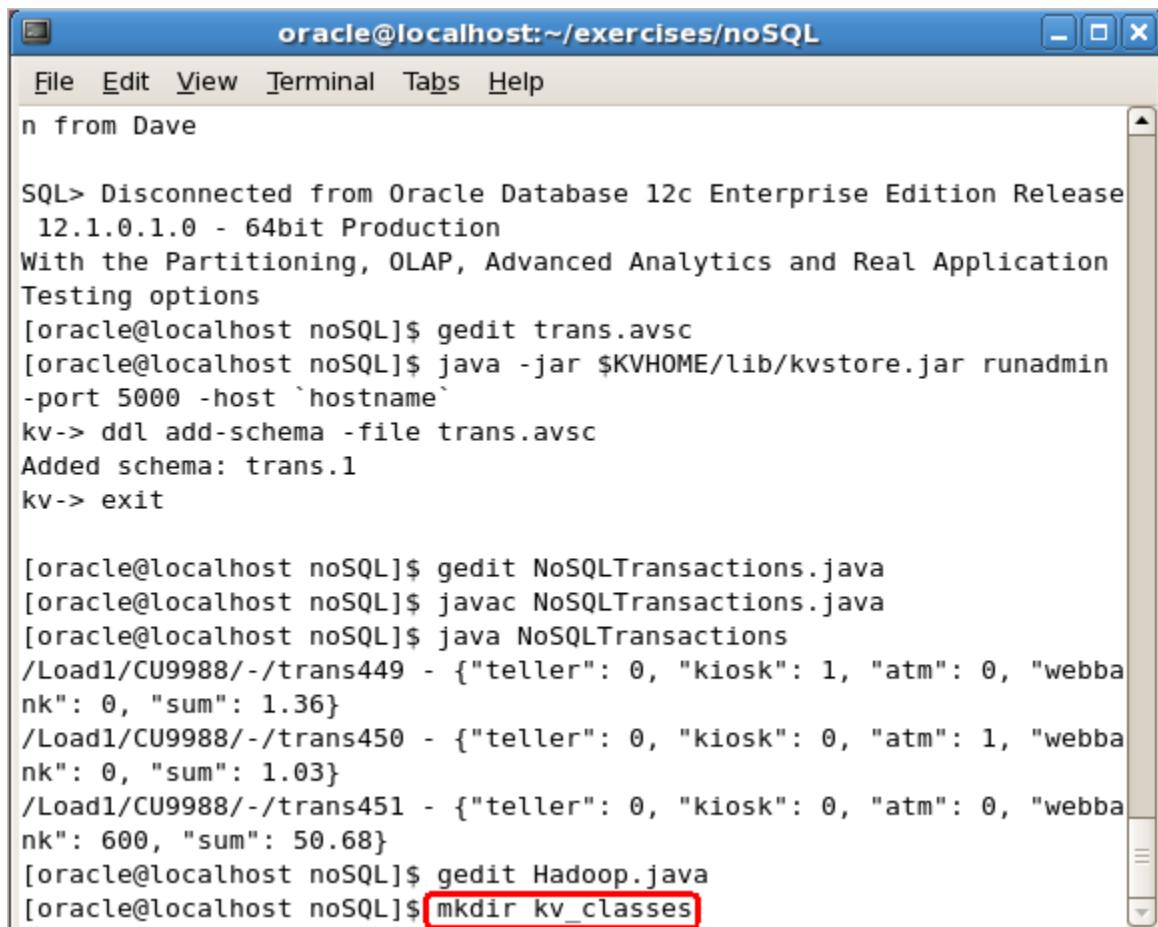
- When you are done evaluating the code click on the X in the right upper corner of the window to close it.



12. Let's create a new directory where we will store the class file resulting from the compilation of the Java code. Go to the terminal and type:

```
mkdir kv_classes
```

Then press **Enter**



```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
n from Dave

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit trans.avsc
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin
-port 5000 -host `hostname`
kv-> ddl add-schema -file trans.avsc
Added schema: trans.1
kv-> exit

[oracle@localhost noSQL]$ gedit NoSQLTransactions.java
[oracle@localhost noSQL]$ javac NoSQLTransactions.java
[oracle@localhost noSQL]$ java NoSQLTransactions
/Load1/CU9988/-/trans449 - {"teller": 0, "kiosk": 1, "atm": 0, "webba
nk": 0, "sum": 1.36}
/Load1/CU9988/-/trans450 - {"teller": 0, "kiosk": 0, "atm": 1, "webba
nk": 0, "sum": 1.03}
/Load1/CU9988/-/trans451 - {"teller": 0, "kiosk": 0, "atm": 0, "webba
nk": 600, "sum": 50.68}
[oracle@localhost noSQL]$ gedit Hadoop.java
[oracle@localhost noSQL]$ mkdir kv_classes
```

13. The following statement compiles the Java code and stores the resulting .class file in the previously created directory. Go to the terminal and type:

```
javac -d kv_classes Hadoop.java
```

Then press **Enter**

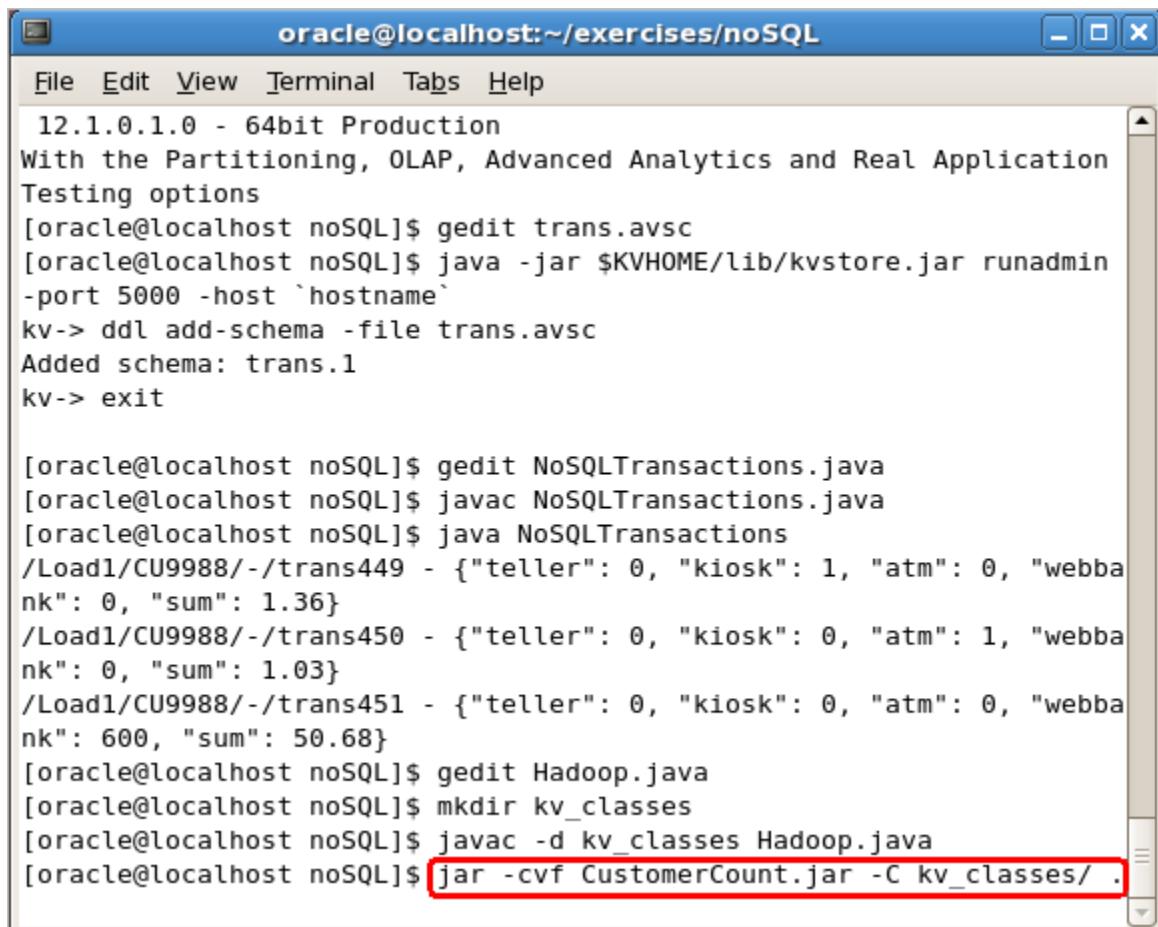
```
oracle@localhost:~/exercises/noSQL
File Edit View Terminal Tabs Help
SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit trans.avsc
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin
-port 5000 -host `hostname`
kv-> ddl add-schema -file trans.avsc
Added schema: trans.1
kv-> exit

[oracle@localhost noSQL]$ gedit NoSQLTransactions.java
[oracle@localhost noSQL]$ javac NoSQLTransactions.java
[oracle@localhost noSQL]$ java NoSQLTransactions
/Load1/CU9988/-/trans449 - {"teller": 0, "kiosk": 1, "atm": 0, "webba
nk": 0, "sum": 1.36}
/Load1/CU9988/-/trans450 - {"teller": 0, "kiosk": 0, "atm": 1, "webba
nk": 0, "sum": 1.03}
/Load1/CU9988/-/trans451 - {"teller": 0, "kiosk": 0, "atm": 0, "webba
nk": 600, "sum": 50.68}
[oracle@localhost noSQL]$ gedit Hadoop.java
[oracle@localhost noSQL]$ mkdir kv_classes
[oracle@localhost noSQL]$ javac -d kv_classes Hadoop.java
```

14. The following statement creates a JAR file containing the compiled Java code. Go to the terminal and type:

```
jar -cvf CustomerCount.jar -C kv_classes/ .
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains a command-line session. The user has already run several commands to set up a database schema and transaction files. The final command shown is "jar -cvf CustomerCount.jar -C kv_classes/ .", which is highlighted with a red rectangle. This command is used to create a JAR file named "CustomerCount.jar" that contains all the classes in the "kv_classes" directory.

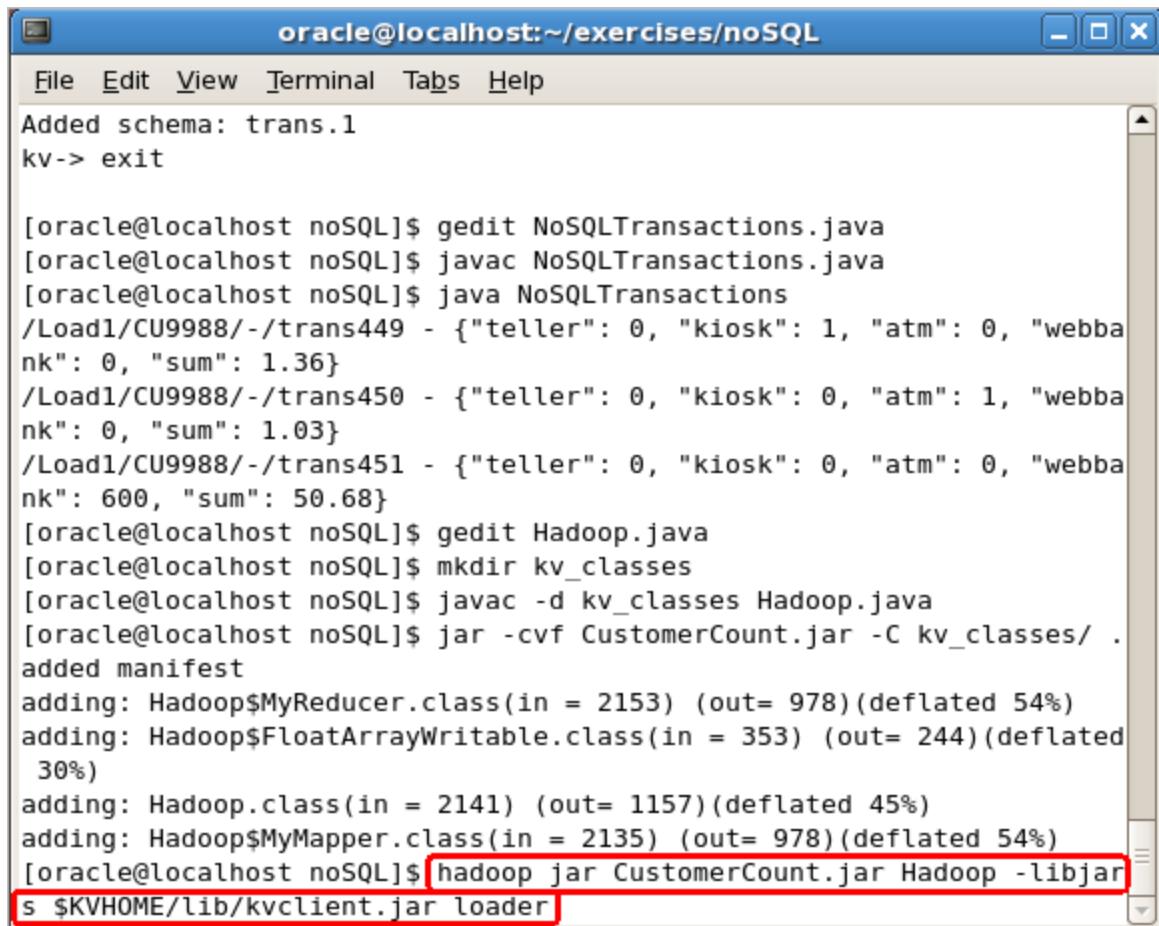
```
File Edit View Terminal Tabs Help
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost noSQL]$ gedit trans.avsc
[oracle@localhost noSQL]$ java -jar $KVHOME/lib/kvstore.jar runadmin
-port 5000 -host `hostname`
kv-> ddl add-schema -file trans.avsc
Added schema: trans.1
kv-> exit

[oracle@localhost noSQL]$ gedit NoSQLTransactions.java
[oracle@localhost noSQL]$ javac NoSQLTransactions.java
[oracle@localhost noSQL]$ java NoSQLTransactions
/Load1/CU9988/-/trans449 - {"teller": 0, "kiosk": 1, "atm": 0, "webba
nk": 0, "sum": 1.36}
/Load1/CU9988/-/trans450 - {"teller": 0, "kiosk": 0, "atm": 1, "webba
nk": 0, "sum": 1.03}
/Load1/CU9988/-/trans451 - {"teller": 0, "kiosk": 0, "atm": 0, "webba
nk": 600, "sum": 50.68}
[oracle@localhost noSQL]$ gedit Hadoop.java
[oracle@localhost noSQL]$ mkdir kv_classes
[oracle@localhost noSQL]$ javac -d kv_classes Hadoop.java
[oracle@localhost noSQL]$ jar -cvf CustomerCount.jar -C kv_classes/ .
```

15. We are now ready to execute the JAR. The output of the Map-Reduce job will be a new directory in HDFS called *loader*, as evidenced by the last parameter of the following command. Go to the terminal and type:

```
hadoop jar CustomerCount.jar Hadoop -libjars $KVHOME/lib/kvclient.jar loader
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains the following text:

```
Added schema: trans.1
kv-> exit

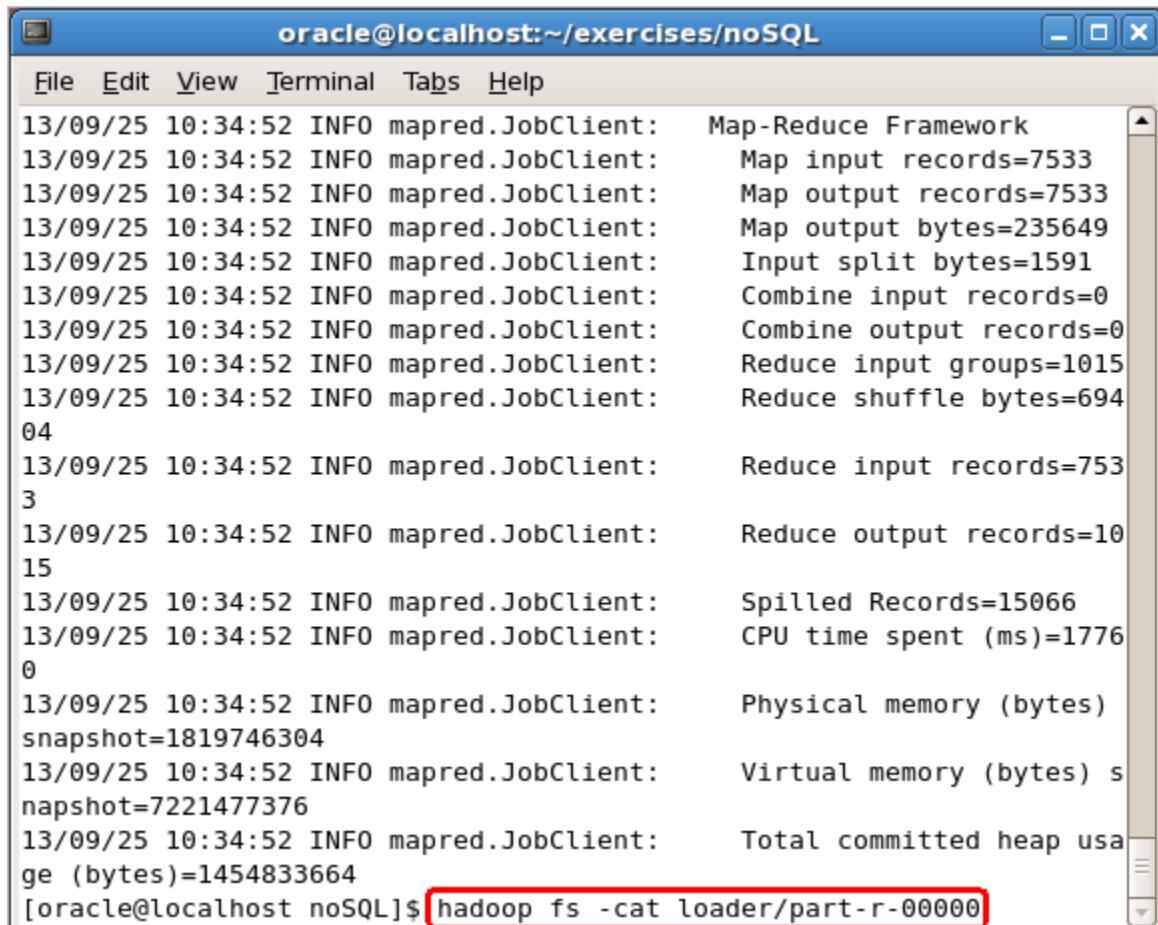
[oracle@localhost noSQL]$ gedit NoSQLTransactions.java
[oracle@localhost noSQL]$ javac NoSQLTransactions.java
[oracle@localhost noSQL]$ java NoSQLTransactions
/Load1/CU9988/-/trans449 - {"teller": 0, "kiosk": 1, "atm": 0, "webbank": 0, "sum": 1.36}
/Load1/CU9988/-/trans450 - {"teller": 0, "kiosk": 0, "atm": 1, "webbank": 0, "sum": 1.03}
/Load1/CU9988/-/trans451 - {"teller": 0, "kiosk": 0, "atm": 0, "webbank": 600, "sum": 50.68}
[oracle@localhost noSQL]$ gedit Hadoop.java
[oracle@localhost noSQL]$ mkdir kv_classes
[oracle@localhost noSQL]$ javac -d kv_classes Hadoop.java
[oracle@localhost noSQL]$ jar -cvf CustomerCount.jar -C kv_classes/ .
added manifest
adding: Hadoop$MyReducer.class(in = 2153) (out= 978)(deflated 54%)
adding: Hadoop$FloatArrayWritable.class(in = 353) (out= 244)(deflated 30%)
adding: Hadoop.class(in = 2141) (out= 1157)(deflated 45%)
adding: Hadoop$MyMapper.class(in = 2135) (out= 978)(deflated 54%)
[oracle@localhost noSQL]$ hadoop jar CustomerCount.jar Hadoop -libjars $KVHOME/lib/kvclient.jar loader
```

The command at the bottom of the terminal window is highlighted with a red rectangle.

16. Once the job is finished, we can view the results. Let's see the content of the part-r-00000 from the loader HDFS directory. Go to the terminal and type:

```
hadoop fs -cat loader/part-r-00000
```

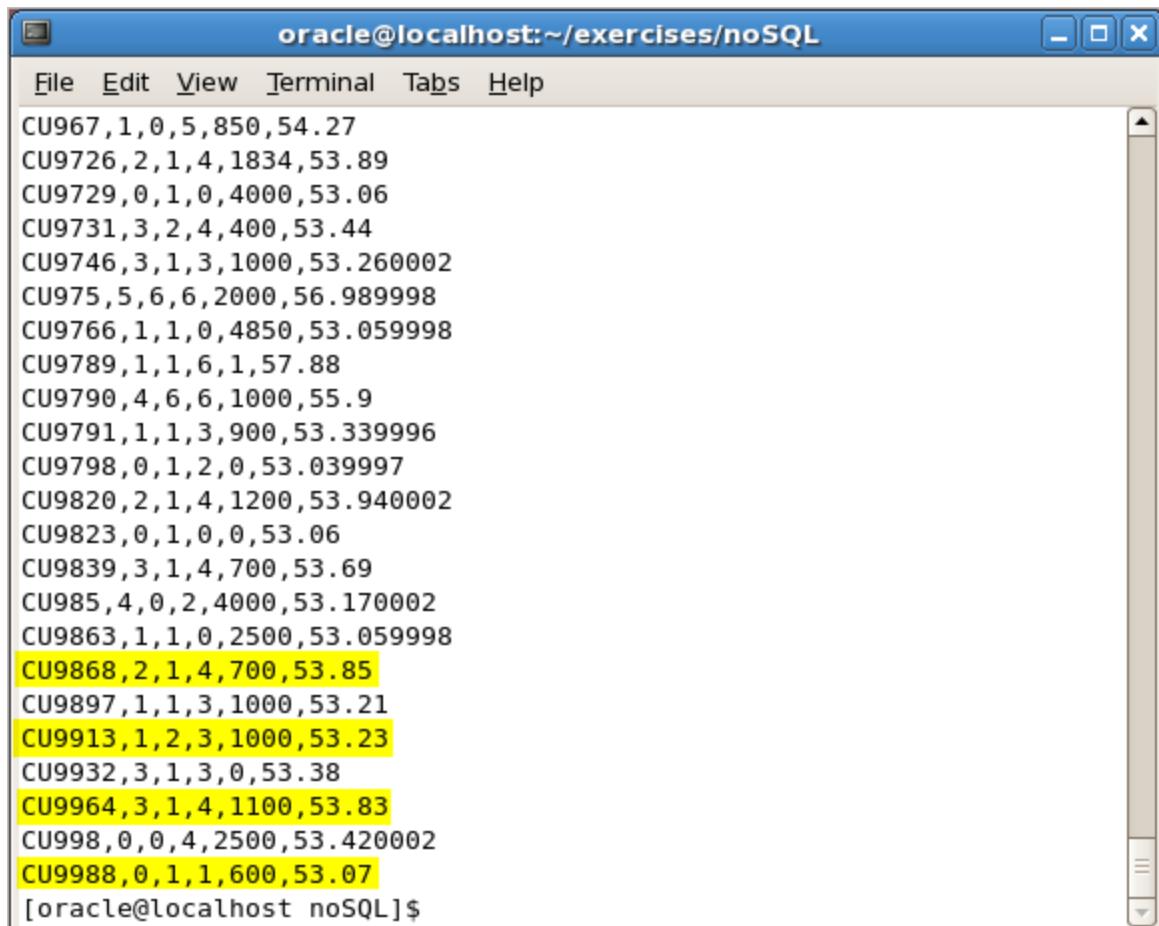
Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window displays log output from a MapReduce job. The log includes various metrics such as map input records, map output records, map output bytes, input split bytes, combine input records, combine output records, reduce input groups, reduce shuffle bytes, reduce input records, reduce output records, spilled records, CPU time spent, physical memory usage, virtual memory usage, and total committed heap usage. At the bottom of the terminal, the command "hadoop fs -cat loader/part-r-00000" is visible, with the entire command line highlighted by a red rectangle.

```
13/09/25 10:34:52 INFO mapred.JobClient: Map-Reduce Framework
13/09/25 10:34:52 INFO mapred.JobClient: Map input records=7533
13/09/25 10:34:52 INFO mapred.JobClient: Map output records=7533
13/09/25 10:34:52 INFO mapred.JobClient: Map output bytes=235649
13/09/25 10:34:52 INFO mapred.JobClient: Input split bytes=1591
13/09/25 10:34:52 INFO mapred.JobClient: Combine input records=0
13/09/25 10:34:52 INFO mapred.JobClient: Combine output records=0
13/09/25 10:34:52 INFO mapred.JobClient: Reduce input groups=1015
13/09/25 10:34:52 INFO mapred.JobClient: Reduce shuffle bytes=694
04
13/09/25 10:34:52 INFO mapred.JobClient: Reduce input records=753
3
13/09/25 10:34:52 INFO mapred.JobClient: Reduce output records=10
15
13/09/25 10:34:52 INFO mapred.JobClient: Spilled Records=15066
13/09/25 10:34:52 INFO mapred.JobClient: CPU time spent (ms)=1776
0
13/09/25 10:34:52 INFO mapred.JobClient: Physical memory (bytes)
snapshot=1819746304
13/09/25 10:34:52 INFO mapred.JobClient: Virtual memory (bytes) s
napshot=7221477376
13/09/25 10:34:52 INFO mapred.JobClient: Total committed heap usa
ge (bytes)=1454833664
[oracle@localhost noSQL]$ hadoop fs -cat loader/part-r-00000
```

We now have an aggregated view of the customers' transactions stored in an HDFS file, which we will later use to load the results into the Oracle Database.



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/noSQL". The window contains a list of 20 aggregated customer transaction records, each consisting of five fields separated by commas. The last four records are highlighted with yellow backgrounds:

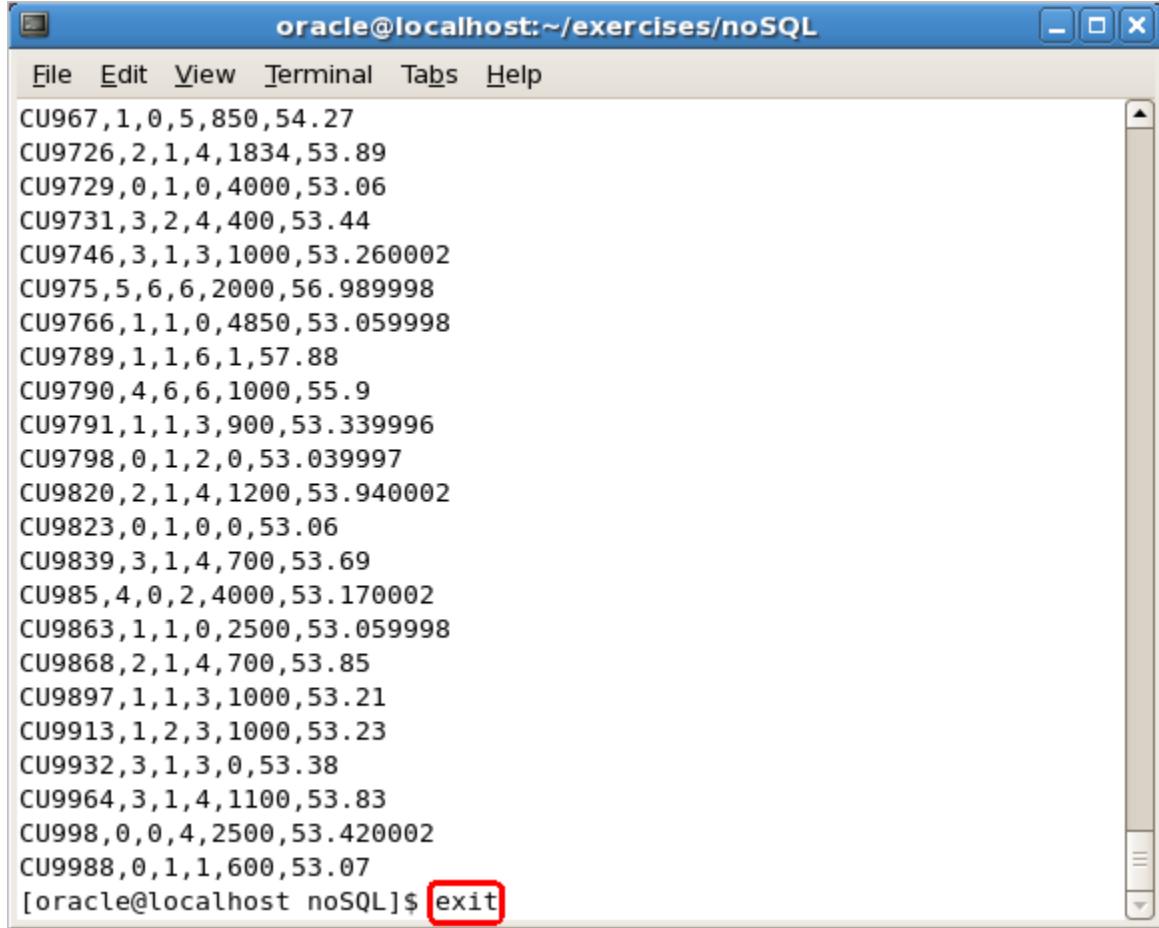
CU967,1,0,5,850,54.27	CU9726,2,1,4,1834,53.89	CU9729,0,1,0,4000,53.06	CU9731,3,2,4,400,53.44	CU9746,3,1,3,1000,53.260002
CU975,5,6,6,2000,56.989998	CU9766,1,1,0,4850,53.059998	CU9789,1,1,6,1,57.88	CU9790,4,6,6,1000,55.9	CU9791,1,1,3,900,53.339996
CU9798,0,1,2,0,53.039997	CU9820,2,1,4,1200,53.940002	CU9823,0,1,0,0,53.06	CU9839,3,1,4,700,53.69	CU985,4,0,2,4000,53.170002
CU9863,1,1,0,2500,53.059998	CU9868,2,1,4,700,53.85	CU9897,1,1,3,1000,53.21	CU9913,1,2,3,1000,53.23	CU9932,3,1,3,0,53.38
CU9964,3,1,4,1100,53.83	CU998,0,0,4,2500,53.420002	CU9988,0,1,1,600,53.07		

[oracle@localhost noSQL]\$

17. This exercise is now finished. Go to the terminal and type:

```
exit
```

Then press **Enter**



A screenshot of a terminal window titled "oracle@localhost:~/exercises/noSQL". The window has a menu bar with File, Edit, View, Terminal, Tabs, and Help. The main area displays a list of approximately 30 data entries, each consisting of five comma-separated values. At the bottom of the list, the command "[oracle@localhost noSQL]\$ exit" is visible, with the word "exit" highlighted by a red rectangle.

CU967,1,0,5,850,54.27
CU9726,2,1,4,1834,53.89
CU9729,0,1,0,4000,53.06
CU9731,3,2,4,400,53.44
CU9746,3,1,3,1000,53.260002
CU975,5,6,6,2000,56.989998
CU9766,1,1,0,4850,53.059998
CU9789,1,1,6,1,57.88
CU9790,4,6,6,1000,55.9
CU9791,1,1,3,900,53.339996
CU9798,0,1,2,0,53.039997
CU9820,2,1,4,1200,53.940002
CU9823,0,1,0,0,53.06
CU9839,3,1,4,700,53.69
CU985,4,0,2,4000,53.170002
CU9863,1,1,0,2500,53.059998
CU9868,2,1,4,700,53.85
CU9897,1,1,3,1000,53.21
CU9913,1,2,3,1000,53.23
CU9932,3,1,3,0,53.38
CU9964,3,1,4,1100,53.83
CU998,0,0,4,2500,53.420002
CU9988,0,1,1,600,53.07
[oracle@localhost noSQL]\$ exit

4.7 Summary

In this exercise you were introduced to Oracle's NoSQL Database. You saw how to insert and retrieve key/value pairs as well as the storeIterator function where multiple values could be retrieved under the same major component of a key. You also saw how NoSQL data can be accessed through external tables from the Oracle Database, how to interact with data stored in AVRO schemas and how NoSQL data can be processed by using Hadoop Map/Reduce jobs.

It is important to note here the differences between the NoSQL Database and a traditional RDBMS. With relational data the queries performed are much more powerful and more complex while NoSQL simply stores and retrieves values for a specific key. Given that simplicity in NoSQL storage type, it has a significant performance and scaling advantage. A NoSQL Database can store petabytes worth of information in a distributed cluster and still maintain very good performance on data interaction at a much lower cost per megabyte of data. NoSQL has many uses and has been implemented successfully in many different circumstances but at the same time it does not mimic or replace the use of a traditional RDBMS.

5. PIG EXERCISE

5.1 Introduction to Pig

Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with an infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turn enables them to handle very large data sets.

At the present time, Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject). Pig's language layer currently consists of a textual language called Pig Latin, which has the following key properties:

Ease of programming: It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain.

Optimization opportunities: The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.

Extensibility: Users can create their own functions to do special-purpose processing.

5.2 Overview of Hands on Exercise

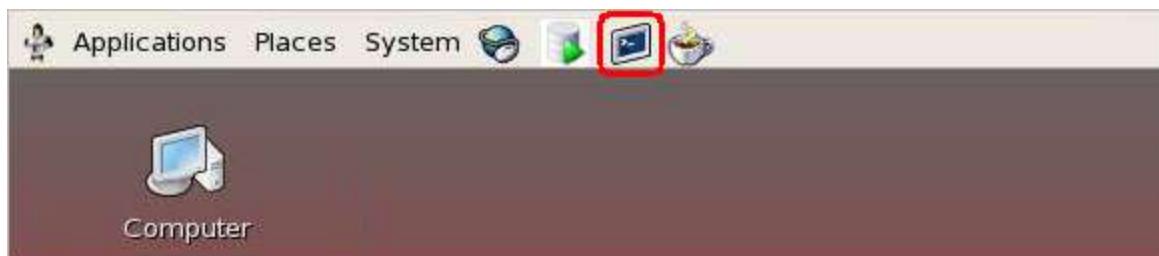
In this exercise we will be processing monthly cash accounts data for the customers in our system. We have a comma delimited file with five columns: customer id, checking amount, bank funds, number of monthly checks written and number of automatic payments performed. This data is retrieved from one of our internal systems on a monthly basis and contains the monthly cash situation for each customer. The file that will be processed contains data for three months, therefore we will see three rows for each customer, each row corresponding to one month. Once we will have loaded the data in Pig, we will proceed to aggregate it so that we obtain monthly average values per customer, for each measured dimension. These values will then be imported in a later exercise into the Oracle Database along with data from other systems, in order to create a 360 degree view of our customers.

In this exercise we will:

1. Load our monthly cash account data into our HDFS
2. Run a PIG script which aggregate the data to determine the average values of the accounts for each customer
3. View the results and save them into an HDFS file that will be imported into the Oracle Database in a later exercise.

5.3 Working with PIG

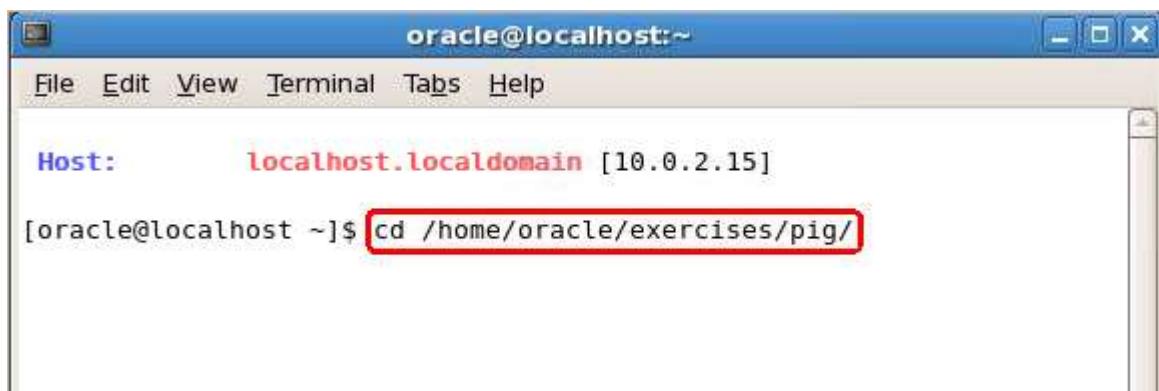
1. All of the setup and execution for this exercise can be done from the terminal, hence open the terminal by clicking on the **Terminal icon** on the desktop.



2. To get into the folder where the scripts for this exercise are, type in the terminal:

```
cd /home/oracle/exercises/pig
```

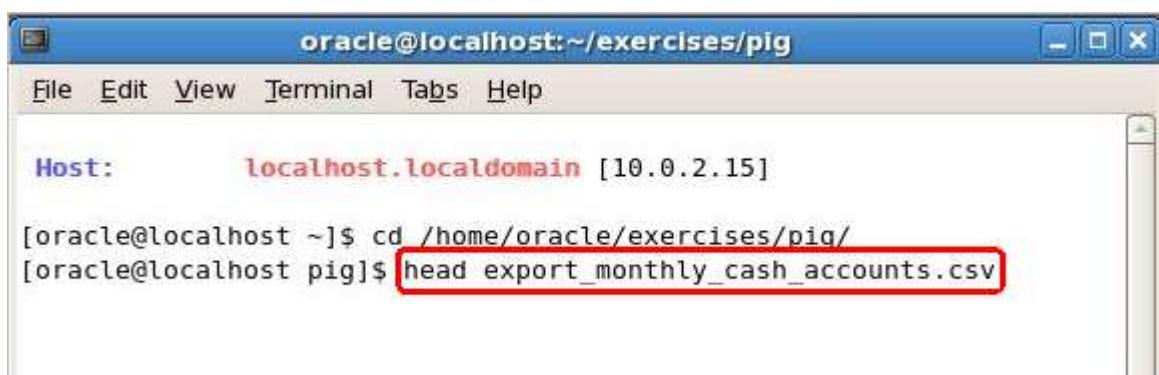
Then press **Enter**



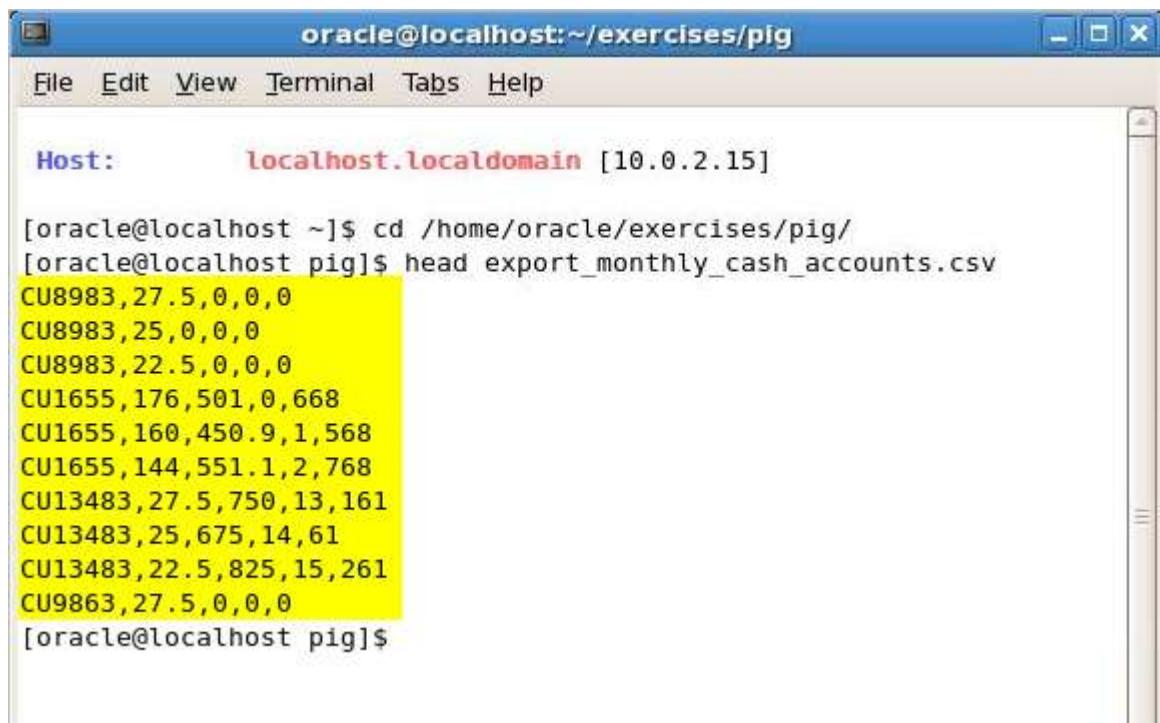
3. To get an idea of what our monthly cash accounts file looks like let's look at the first couple of rows. In the terminal type:

```
head export_monthly_cash_accounts.csv
```

Then press **Enter**



The first 10 rows of the data file will be displayed on the screen. The first column represents the customer id, followed by the monthly checking amount, bank funds, number of checks written in a month and number of automatic payments performed.



A screenshot of a terminal window titled "oracle@localhost:~/exercises/pig". The window has a menu bar with File, Edit, View, Terminal, Tabs, and Help. Below the menu is a status line showing "Host: localhost.localdomain [10.0.2.15]". The terminal window contains the following text:

```
[oracle@localhost ~]$ cd /home/oracle/exercises/pig/  
[oracle@localhost pig]$ head export_monthly_cash_accounts.csv  
CU8983,27.5,0,0,0  
CU8983,25,0,0,0  
CU8983,22.5,0,0,0  
CU1655,176,501,0,668  
CU1655,160,450.9,1,568  
CU1655,144,551.1,2,768  
CU13483,27.5,750,13,161  
CU13483,25,675,14,61  
CU13483,22.5,825,15,261  
CU9863,27.5,0,0,0  
[oracle@localhost pig]$
```

4. Now that we have an idea of what our data file looks like, let's load it into the HDFS for processing. To load the data we use the copyFromLocal function of Hadoop, go to the terminal and type:

```
hadoop fs -copyFromLocal export_monthly_cash_accounts.csv  
Then press Enter
```

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window has a blue header bar with the title and standard menu options: File, Edit, View, Terminal, Tabs, Help. Below the title bar is a toolbar with icons for copy, paste, and close. The main area of the terminal shows the following text:

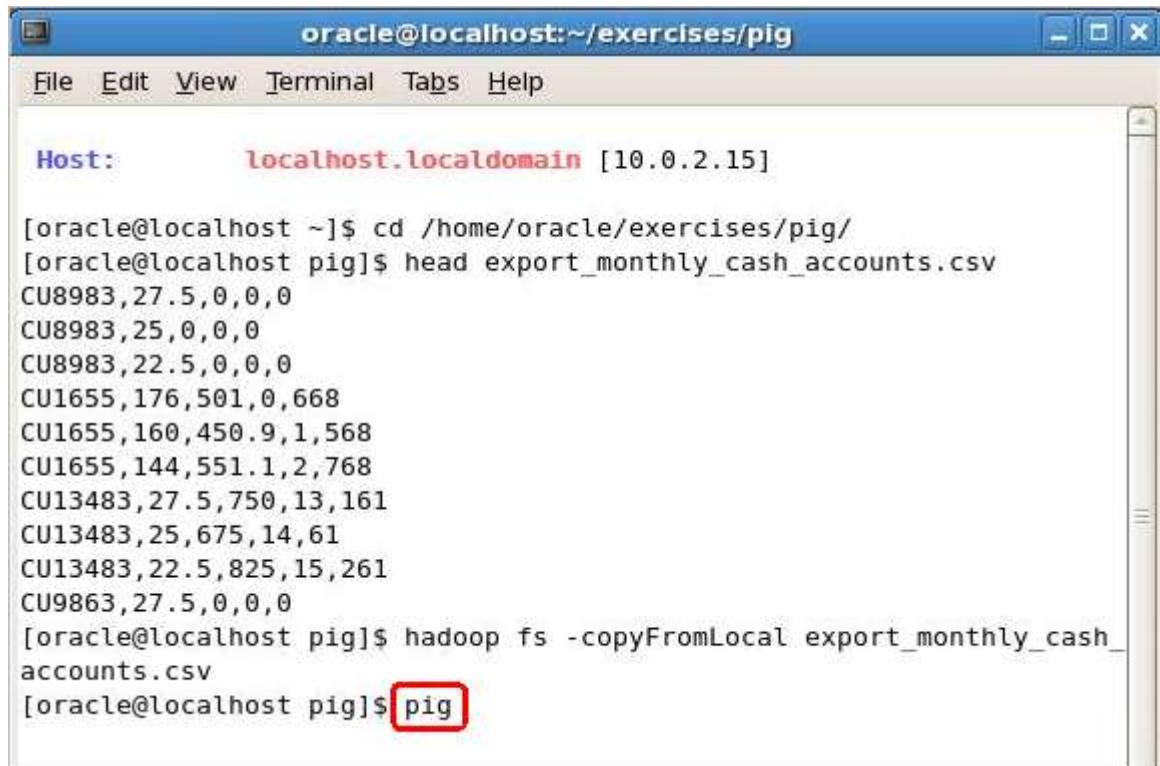
```
Host: localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/pig/
[oracle@localhost pig]$ head export_monthly_cash_accounts.csv
CU8983,27.5,0,0,0
CU8983,25,0,0,0
CU8983,22.5,0,0,0
CU1655,176,501,0,668
CU1655,160,450.9,1,568
CU1655,144,551.1,2,768
CU13483,27.5,750,13,161
CU13483,25,675,14,61
CU13483,22.5,825,15,261
CU9863,27.5,0,0,0
[oracle@localhost pig]$ hadoop fs -copyFromLocal export_monthly_cash_
accounts.csv
```

The command `hadoop fs -copyFromLocal export_monthly_cash_accounts.csv` is highlighted with a red rectangular box.

5. We will be running our PIG script in interactive mode so that we can see each step of the process. For this we will need to open the PIG interpreter called grunt. To do this, go to the terminal and type:

pig
Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window has a menu bar with File, Edit, View, Terminal, Tabs, and Help. The host information is displayed as "Host: localhost.localdomain [10.0.2.15]". The terminal output shows the following sequence of commands and data:

```
Host: localhost.localdomain [10.0.2.15]

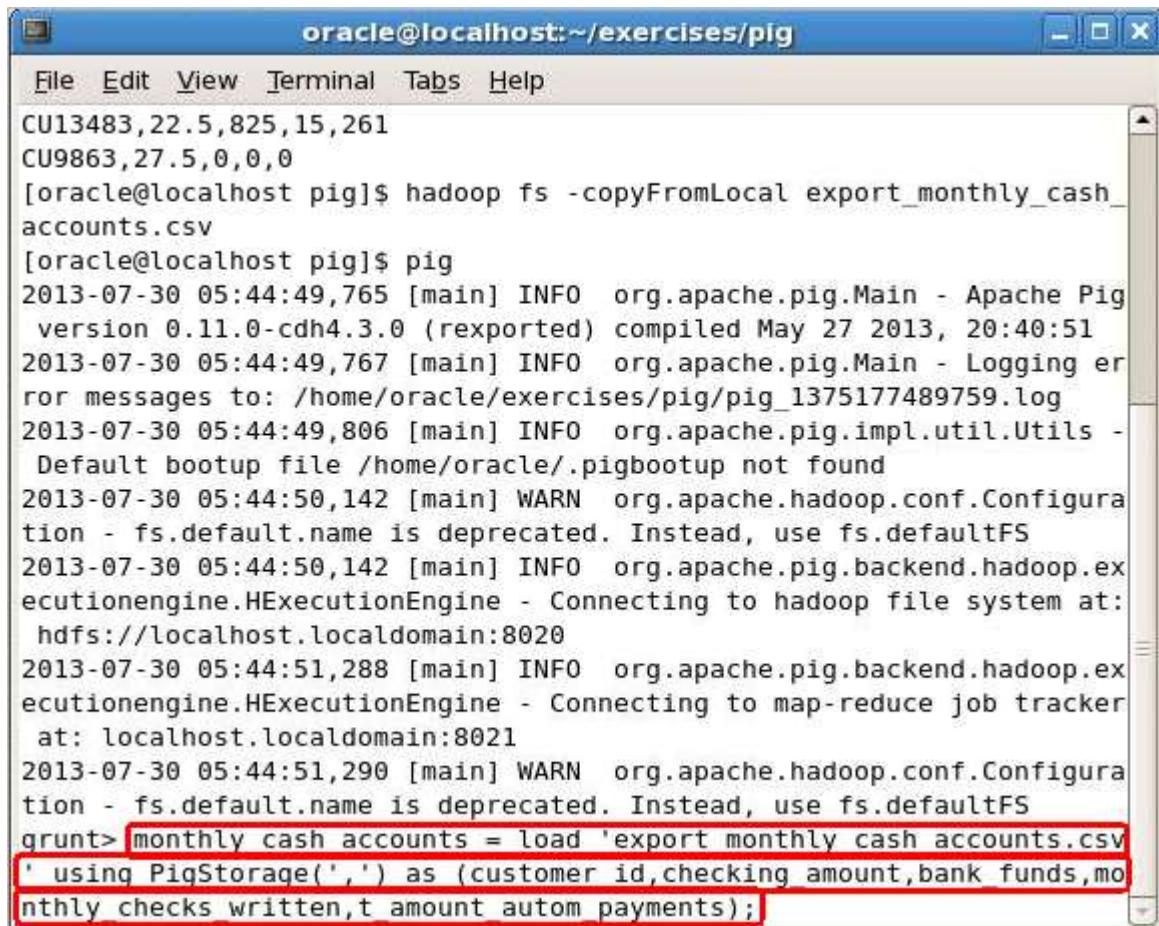
[oracle@localhost ~]$ cd /home/oracle/exercises/pig/
[oracle@localhost pig]$ head export_monthly_cash_accounts.csv
CU8983,27.5,0,0,0
CU8983,25,0,0,0
CU8983,22.5,0,0,0
CU1655,176,501,0,668
CU1655,160,450.9,1,568
CU1655,144,551.1,2,768
CU13483,27.5,750,13,161
CU13483,25,675,14,61
CU13483,22.5,825,15,261
CU9863,27.5,0,0,0
[oracle@localhost pig]$ hadoop fs -copyFromLocal export_monthly_cash_
accounts.csv
[oracle@localhost pig]$ pig
```

The command "pig" at the end of the session is highlighted with a red box.

6. Once we are in the grunt shell we can start typing Pig script. The first thing we need to do is load the datafile from HDFS into Pig for processing. The data is not actually copied but a handler is created for the file so Pig knows how to interpret the data. Go to the grunt shell and type:

```
monthly_cash_accounts = load 'export_monthly_cash_accounts.csv' using  
PigStorage(',') as  
(customer_id,checking_amount,bank_funds,monthly_checks_written,t_amount_a  
utom_payments);
```

Then Press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window contains the following text:

```
File Edit View Terminal Tabs Help  
CU13483,22.5,825,15,261  
CU9863,27.5,0,0,0  
[oracle@localhost pig]$ hadoop fs -copyFromLocal export_monthly_cash_accounts.csv  
[oracle@localhost pig]$ pig  
2013-07-30 05:44:49,765 [main] INFO org.apache.pig.Main - Apache Pig  
version 0.11.0-cdh4.3.0 (rexported) compiled May 27 2013, 20:40:51  
2013-07-30 05:44:49,767 [main] INFO org.apache.pig.Main - Logging er  
ror messages to: /home/oracle/exercises/pig/pig_1375177489759.log  
2013-07-30 05:44:49,806 [main] INFO org.apache.pig.impl.util.Utils -  
Default bootup file /home/oracle/.pigbootup not found  
2013-07-30 05:44:50,142 [main] WARN org.apache.hadoop.conf.Configura  
tion - fs.default.name is deprecated. Instead, use fs.defaultFS  
2013-07-30 05:44:50,142 [main] INFO org.apache.pig.backend.hadoop.ex  
ecutionengine.HExecutionEngine - Connecting to hadoop file system at:  
hdfs://localhost.localdomain:8020  
2013-07-30 05:44:51,288 [main] INFO org.apache.pig.backend.hadoop.ex  
ecutionengine.HExecutionEngine - Connecting to map-reduce job tracker  
at: localhost.localdomain:8021  
2013-07-30 05:44:51,290 [main] WARN org.apache.hadoop.conf.Configura  
tion - fs.default.name is deprecated. Instead, use fs.defaultFS  
grunt> monthly cash accounts = load 'export monthly cash accounts.csv'  
' using PigStorage(',') as (customer id,checking amount,bank funds,mo  
nthly checks written,t amount autom payments);
```

The last two lines of the script are highlighted with a red rectangle.

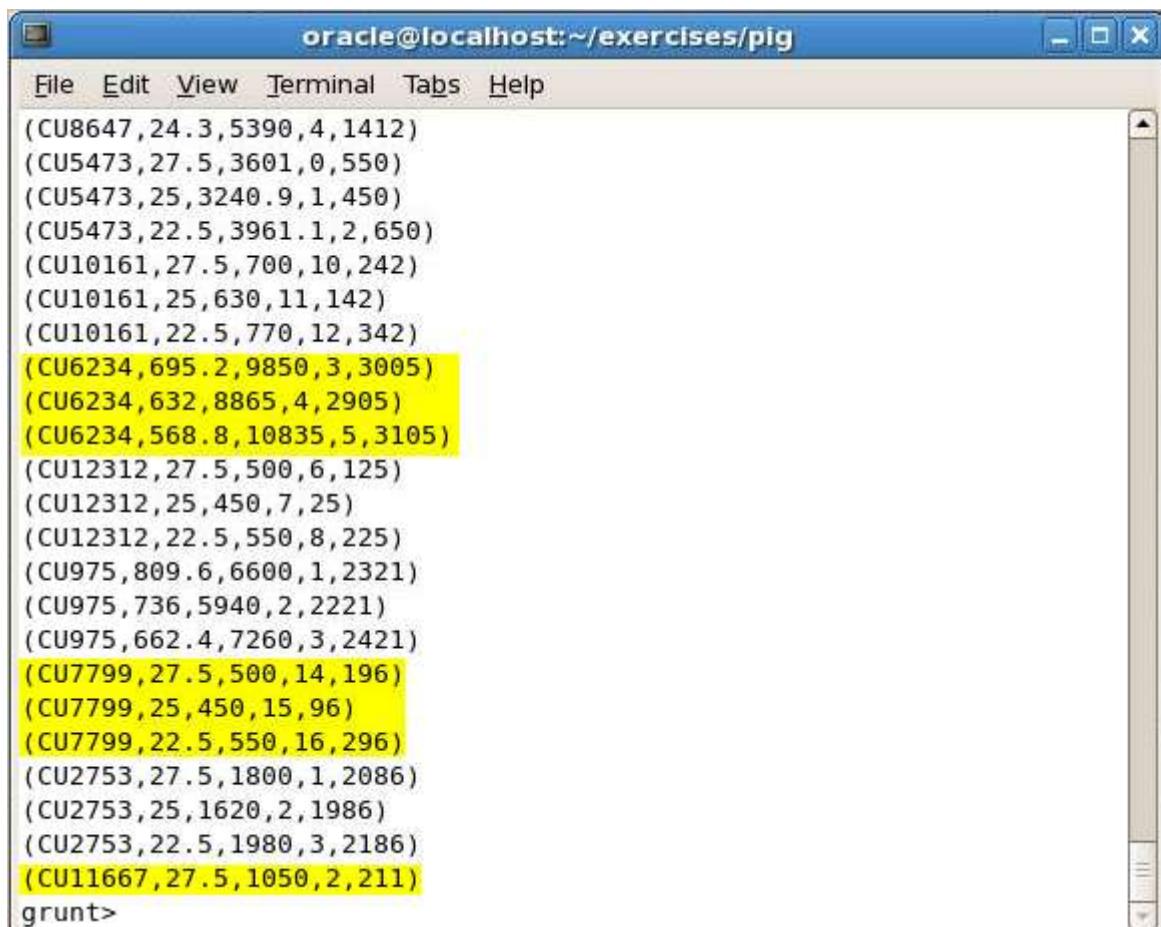
7. Now the data is loaded as a five column table lets see what the data looks like in PIG. Go to the grunt shell and type:

```
dump monthly_cash_accounts;  
Then press Enter
```

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window contains several lines of Hadoop configuration warnings from July 30, 2013, at 05:46:19,248. These warnings are related to deprecated configurations such as "dfs.backup.address", "topology.script.number.args", "dfs.balance.bandwidthPerSec", "dfs.name.edits.dir", "dfs.safemode.threshold.pct", "dfs.name.dir", and "fs.checkpoint.period". Below these warnings, the PIG command "dump monthly_cash_accounts;" is typed into the terminal.

```
File Edit View Terminal Tabs Help  
2013-07-30 05:46:19,248 [main] WARN org.apache.hadoop.conf.Configuration - dfs.backup.address is deprecated. Instead, use dfs.namenode.backup.address  
2013-07-30 05:46:19,248 [main] WARN org.apache.hadoop.conf.Configuration - topology.script.number.args is deprecated. Instead, use net.topology.script.number.args  
2013-07-30 05:46:19,248 [main] WARN org.apache.hadoop.conf.Configuration - dfs.balance.bandwidthPerSec is deprecated. Instead, use dfs.datanode.balance.bandwidthPerSec  
2013-07-30 05:46:19,249 [main] WARN org.apache.hadoop.conf.Configuration - dfs.name.edits.dir is deprecated. Instead, use dfs.namenode.edits.dir  
2013-07-30 05:46:19,249 [main] WARN org.apache.hadoop.conf.Configuration - dfs.safemode.threshold.pct is deprecated. Instead, use dfs.namenode.safemode.threshold-pct  
2013-07-30 05:46:19,249 [main] WARN org.apache.hadoop.conf.Configuration - dfs.name.dir is deprecated. Instead, use dfs.namenode.name.dir  
2013-07-30 05:46:19,249 [main] WARN org.apache.hadoop.conf.Configuration - fs.checkpoint.period is deprecated. Instead, use dfs.namenode.checkpoint.period  
2013-07-30 05:46:19,249 [main] WARN org.apache.hadoop.conf.Configuration - hadoop.native.lib is deprecated. Instead, use io.native.lib.available  
grunt> dump monthly_cash_accounts;
```

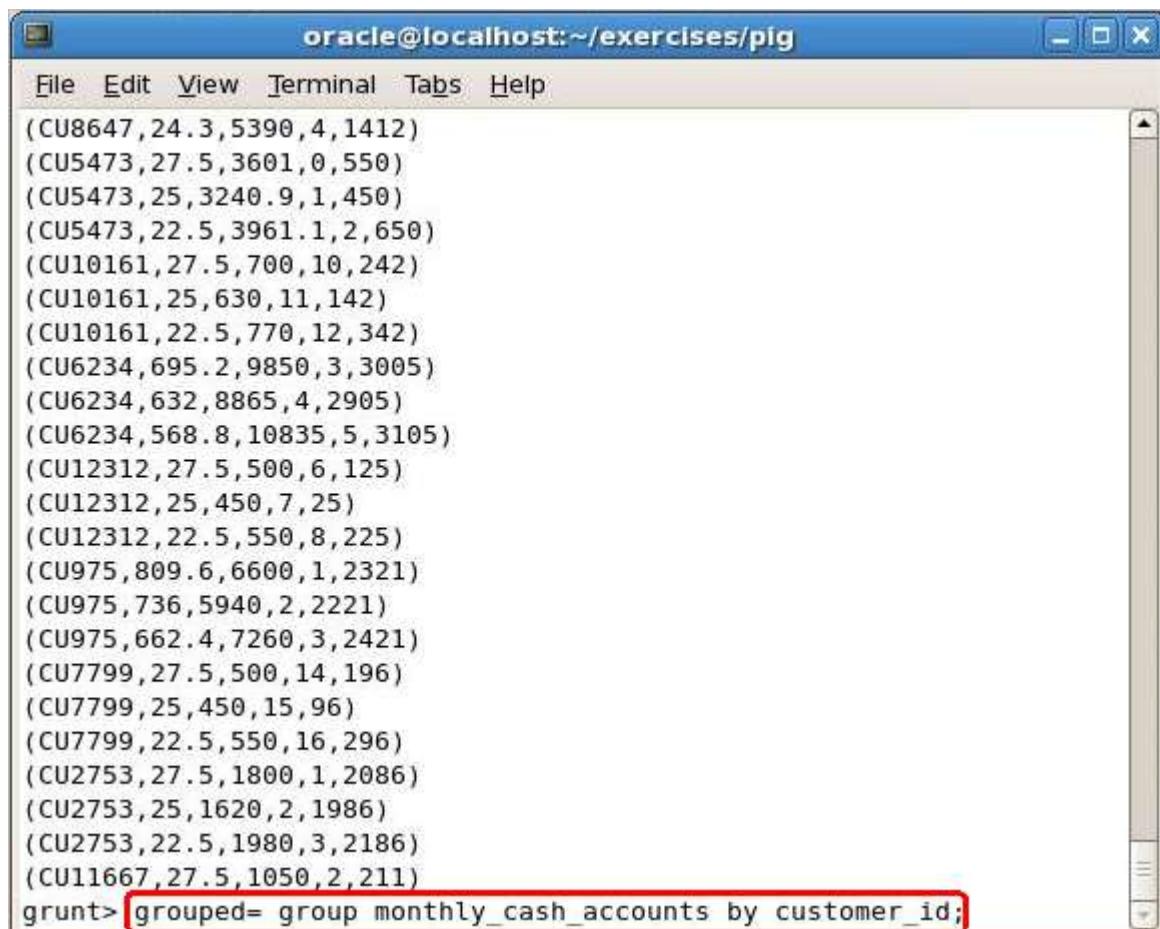
You will see output similar to the Word-Count exercise on the screen. This is normal as Pig is merely a high level language. All commands which process data simply run Map Reduce tasks in the background, so the dump command simply becomes a map reduce job that is run. This will apply to all of the commands you will run in Pig. The output on the screen will show you all of the rows of the file in tuple form.



```
oracle@localhost:~/exercises/pig
File Edit View Terminal Tabs Help
(CU8647,24.3,5390,4,1412)
(CU5473,27.5,3601,0,550)
(CU5473,25,3240.9,1,450)
(CU5473,22.5,3961.1,2,650)
(CU10161,27.5,700,10,242)
(CU10161,25,630,11,142)
(CU10161,22.5,770,12,342)
(CU6234,695.2,9850,3,3005)
(CU6234,632,8865,4,2905)
(CU6234,568.8,10835,5,3105)
(CU12312,27.5,500,6,125)
(CU12312,25,450,7,25)
(CU12312,22.5,550,8,225)
(CU975,809.6,6600,1,2321)
(CU975,736,5940,2,2221)
(CU975,662.4,7260,3,2421)
(CU7799,27.5,500,14,196)
(CU7799,25,450,15,96)
(CU7799,22.5,550,16,296)
(CU2753,27.5,1800,1,2086)
(CU2753,25,1620,2,1986)
(CU2753,22.5,1980,3,2186)
(CU11667,27.5,1050,2,211)
grunt>
```

8. The first step in analyzing the data will be grouping the data by customer id so that we have all of the account details of one customer grouped together. Go to the grunt shell and type:

```
grouped= group monthly_cash_accounts by customer_id;  
Then press Enter
```

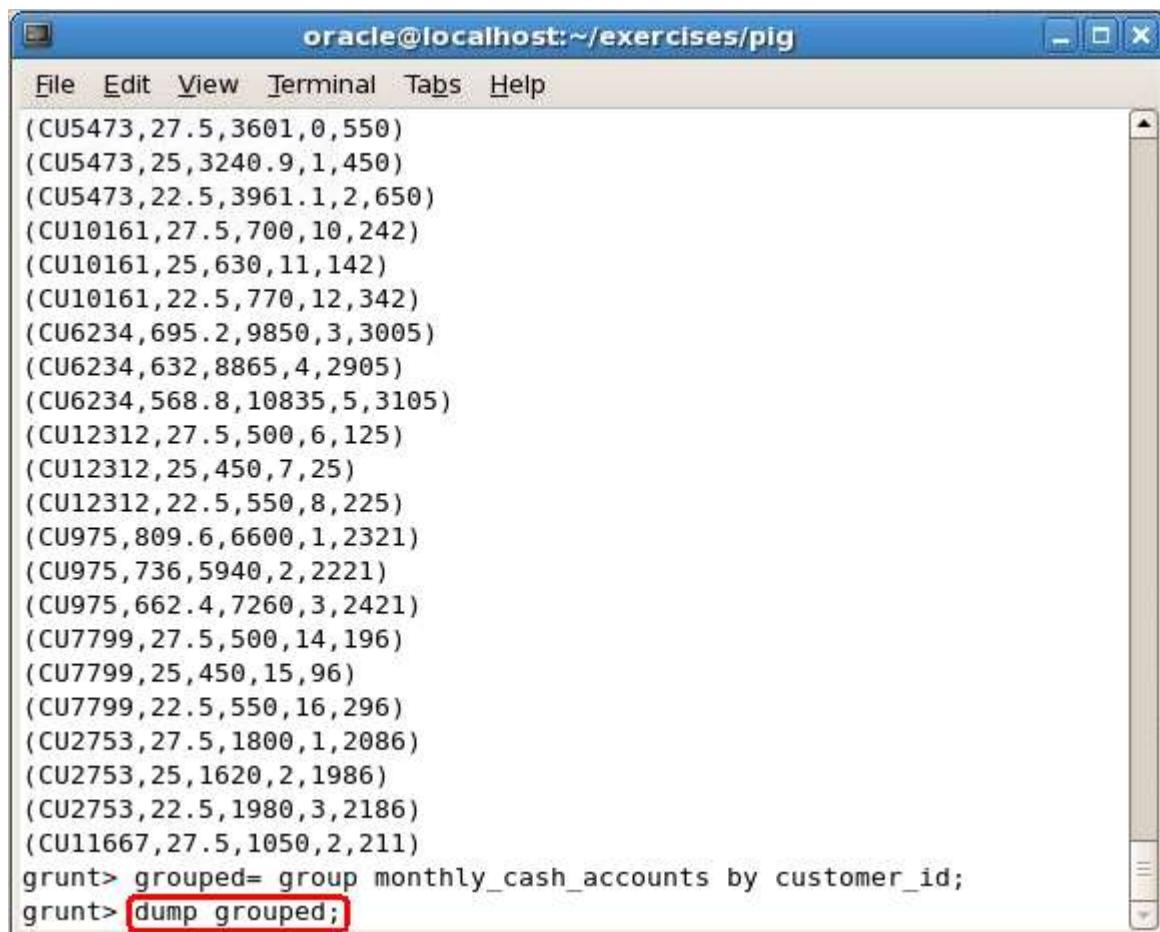


A screenshot of a terminal window titled "oracle@localhost:~/exercises/pig". The window shows a list of tuples representing monthly cash accounts. The last line of the terminal, which contains the command "grouped= group monthly_cash_accounts by customer_id;", is highlighted with a red rectangle.

```
(CU8647,24.3,5390,4,1412)  
(CU5473,27.5,3601,0,550)  
(CU5473,25,3240.9,1,450)  
(CU5473,22.5,3961.1,2,650)  
(CU10161,27.5,700,10,242)  
(CU10161,25,630,11,142)  
(CU10161,22.5,770,12,342)  
(CU6234,695.2,9850,3,3005)  
(CU6234,632,8865,4,2905)  
(CU6234,568.8,10835,5,3105)  
(CU12312,27.5,500,6,125)  
(CU12312,25,450,7,25)  
(CU12312,22.5,550,8,225)  
(CU975,809.6,6600,1,2321)  
(CU975,736,5940,2,2221)  
(CU975,662.4,7260,3,2421)  
(CU7799,27.5,500,14,196)  
(CU7799,25,450,15,96)  
(CU7799,22.5,550,16,296)  
(CU2753,27.5,1800,1,2086)  
(CU2753,25,1620,2,1986)  
(CU2753,22.5,1980,3,2186)  
(CU11667,27.5,1050,2,211)  
grunt> grouped= group monthly_cash_accounts by customer_id;
```

9. Let's go ahead and dump this grouped variable to the screen to see what its contents look like. Go to the grunt shell and type:

```
dump grouped;  
Then press Enter
```

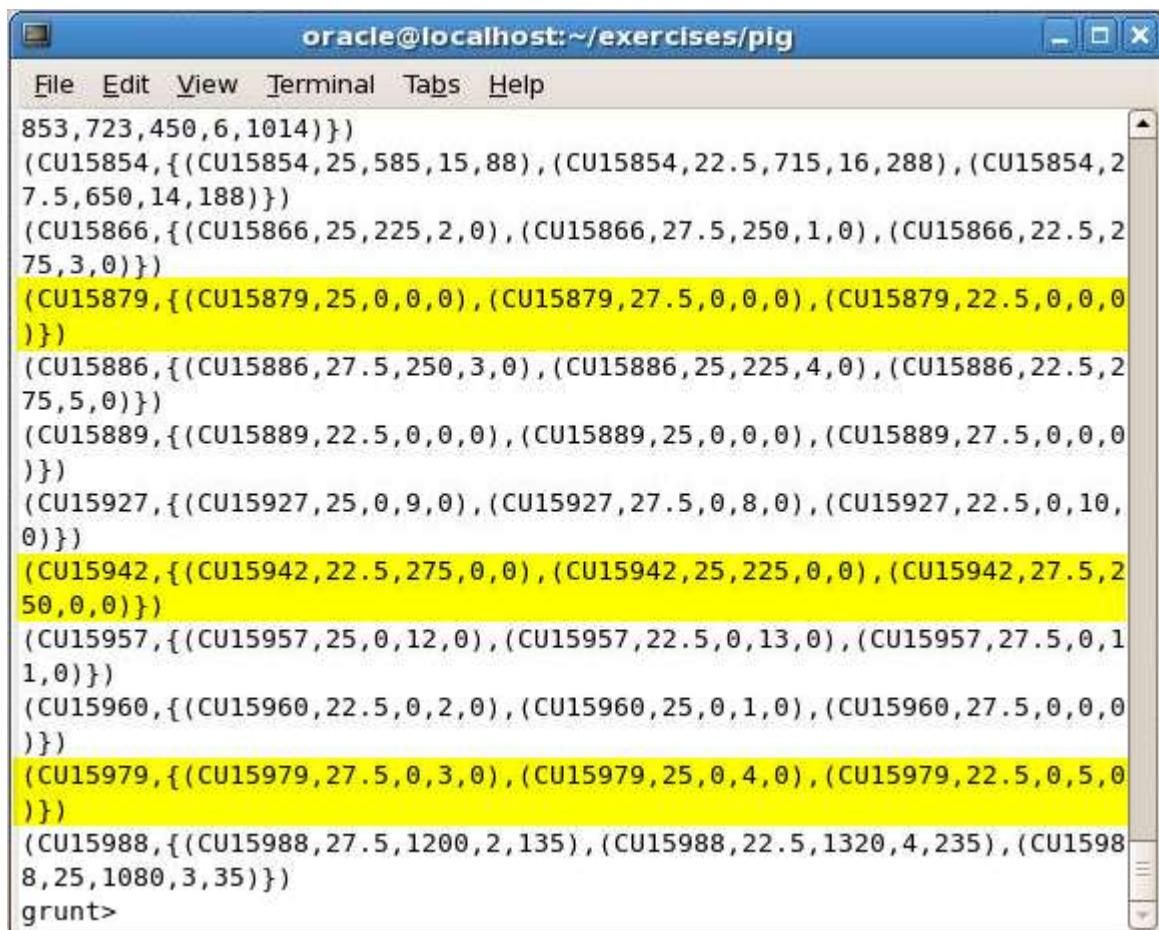


The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window contains a list of tuples representing grouped data. The tuples are as follows:

- (CU5473, 27.5, 3601, 0, 550)
- (CU5473, 25, 3240.9, 1, 450)
- (CU5473, 22.5, 3961.1, 2, 650)
- (CU10161, 27.5, 700, 10, 242)
- (CU10161, 25, 630, 11, 142)
- (CU10161, 22.5, 770, 12, 342)
- (CU6234, 695.2, 9850, 3, 3005)
- (CU6234, 632, 8865, 4, 2905)
- (CU6234, 568.8, 10835, 5, 3105)
- (CU12312, 27.5, 500, 6, 125)
- (CU12312, 25, 450, 7, 25)
- (CU12312, 22.5, 550, 8, 225)
- (CU975, 809.6, 6600, 1, 2321)
- (CU975, 736, 5940, 2, 2221)
- (CU975, 662.4, 7260, 3, 2421)
- (CU7799, 27.5, 500, 14, 196)
- (CU7799, 25, 450, 15, 96)
- (CU7799, 22.5, 550, 16, 296)
- (CU2753, 27.5, 1800, 1, 2086)
- (CU2753, 25, 1620, 2, 1986)
- (CU2753, 22.5, 1980, 3, 2186)
- (CU11667, 27.5, 1050, 2, 211)

At the bottom of the terminal window, the Grunt prompt shows the command "dump grouped;" with the word "grouped;" highlighted in red.

On the screen you will see all of the groups displayed in tuples of tuples form. As the output might look a bit confusing only some tuples are highlighted in the screenshot below to help clarity.



```
oracle@localhost:~/exercises/pig
File Edit View Terminal Tabs Help
853,723,450,6,1014)})
(CU15854,{(CU15854,25,585,15,88),(CU15854,22.5,715,16,288),(CU15854,2
7.5,650,14,188)})
(CU15866,{(CU15866,25,225,2,0),(CU15866,27.5,250,1,0),(CU15866,22.5,2
75,3,0)})
(CU15879,{(CU15879,25,0,0,0),(CU15879,27.5,0,0,0),(CU15879,22.5,0,0,0
)})
(CU15886,{(CU15886,27.5,250,3,0),(CU15886,25,225,4,0),(CU15886,22.5,2
75,5,0)})
(CU15889,{(CU15889,22.5,0,0,0),(CU15889,25,0,0,0),(CU15889,27.5,0,0,0
)})
(CU15927,{(CU15927,25,0,9,0),(CU15927,27.5,0,8,0),(CU15927,22.5,0,10,
0)})
(CU15942,{(CU15942,22.5,275,0,0),(CU15942,25,225,0,0),(CU15942,27.5,2
50,0,0)})
(CU15957,{(CU15957,25,0,12,0),(CU15957,22.5,0,13,0),(CU15957,27.5,0,1
1,0)})
(CU15960,{(CU15960,22.5,0,2,0),(CU15960,25,0,1,0),(CU15960,27.5,0,0,0
)})
(CU15979,{(CU15979,27.5,0,3,0),(CU15979,25,0,4,0),(CU15979,22.5,0,5,0
)})
(CU15988,{(CU15988,27.5,1200,2,135),(CU15988,22.5,1320,4,235),(CU1598
8,25,1080,3,35)})
grunt>
```

10. In the next step we will go through each group tuple and get the customer id and the average values for the checking amount, bank funds, number of checks written monthly and number of automatic payments performed monthly. Go to the grunt shell and type:

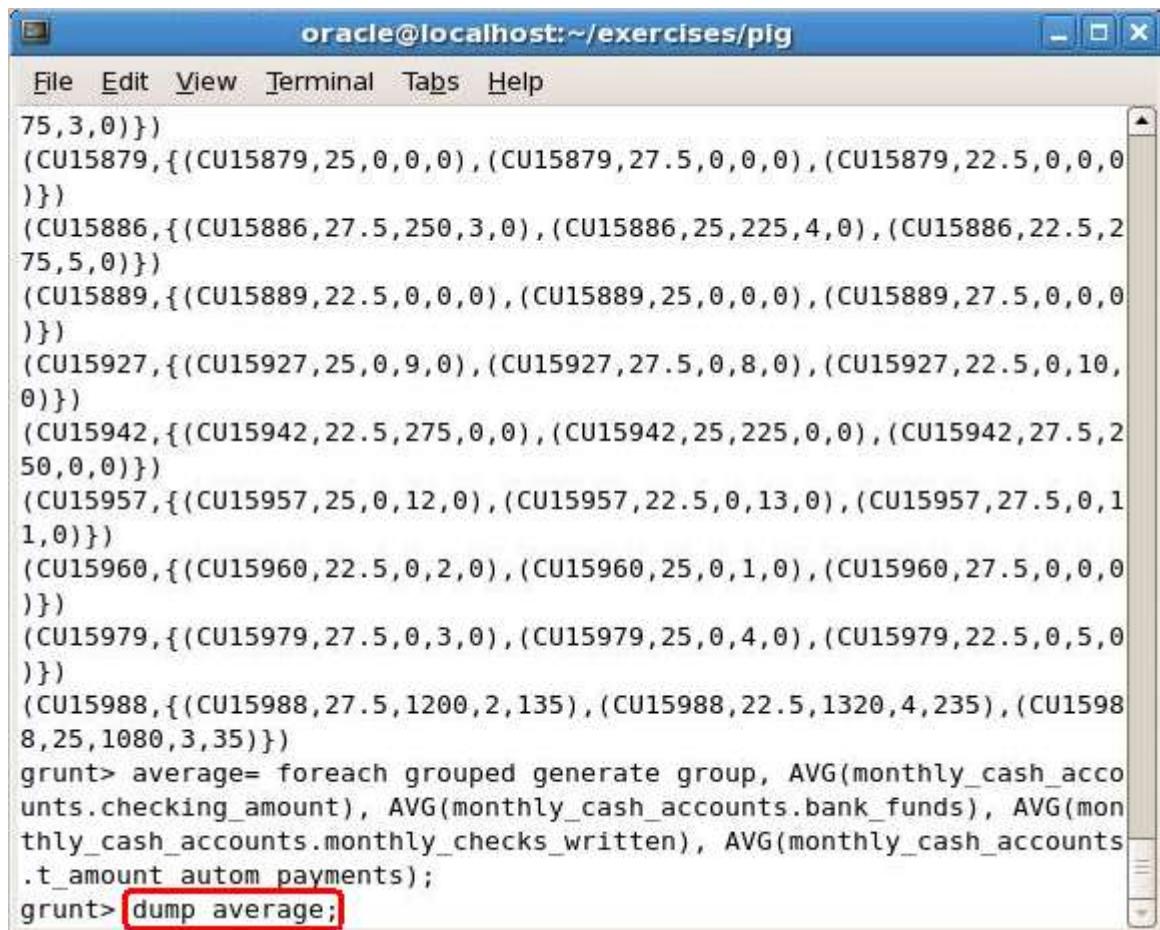
```
average= foreach grouped generate group,
AVG(monthly_cash_accounts.checking_amount),
AVG(monthly_cash_accounts.bank_funds),
AVG(monthly_cash_accounts.monthly_checks_written),
AVG(monthly_cash_accounts.t_amount_autom_payments);
```

Then press **Enter**

```
(CU15866,{{(CU15866,25,225,2,0),(CU15866,27.5,250,1,0),(CU15866,22.5,275,3,0)}})
(CU15879,{{(CU15879,25,0,0,0),(CU15879,27.5,0,0,0),(CU15879,22.5,0,0,0)}})
(CU15886,{{(CU15886,27.5,250,3,0),(CU15886,25,225,4,0),(CU15886,22.5,275,5,0)}})
(CU15889,{{(CU15889,22.5,0,0,0),(CU15889,25,0,0,0),(CU15889,27.5,0,0,0)}})
(CU15927,{{(CU15927,25,0,9,0),(CU15927,27.5,0,8,0),(CU15927,22.5,0,10,0)}})
(CU15942,{{(CU15942,22.5,275,0,0),(CU15942,25,225,0,0),(CU15942,27.5,250,0,0)}})
(CU15957,{{(CU15957,25,0,12,0),(CU15957,22.5,0,13,0),(CU15957,27.5,0,11,0)}})
(CU15960,{{(CU15960,22.5,0,2,0),(CU15960,25,0,1,0),(CU15960,27.5,0,0,0)}})
(CU15979,{{(CU15979,27.5,0,3,0),(CU15979,25,0,4,0),(CU15979,22.5,0,5,0)}})
(CU15988,{{(CU15988,27.5,1200,2,135),(CU15988,22.5,1320,4,235),(CU15988,25,1080,3,35)}})
grunt> average= foreach grouped generate group, AVG(monthly_cash_accounts.checking_amount), AVG(monthly_cash_accounts.bank_funds), AVG(monthly_cash_accounts.monthly_checks_written), AVG(monthly_cash_accounts.t_amount_autom_payments);
```

11. Let's go ahead and see what this output looks like. Go to the grunt shell and type:

```
dump average;  
Then press Enter
```

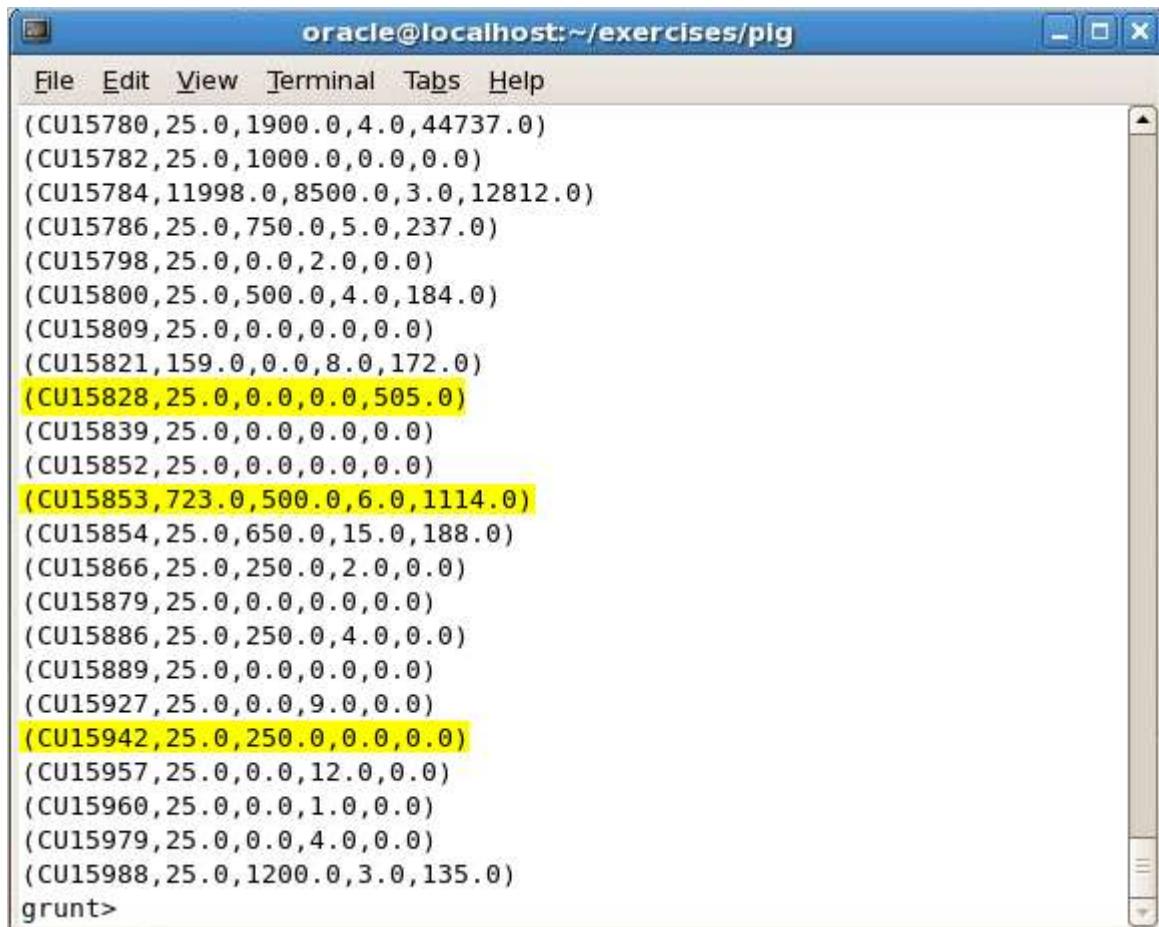


The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window contains the following text:

```
File Edit View Terminal Tabs Help  
75,3,0)})  
(CU15879,{(CU15879,25,0,0,0),(CU15879,27.5,0,0,0),(CU15879,22.5,0,0,0  
)})  
(CU15886,{(CU15886,27.5,250,3,0),(CU15886,25,225,4,0),(CU15886,22.5,2  
75.5,0)})  
(CU15889,{(CU15889,22.5,0,0,0),(CU15889,25,0,0,0),(CU15889,27.5,0,0,0  
)})  
(CU15927,{(CU15927,25,0,9,0),(CU15927,27.5,0,8,0),(CU15927,22.5,0,10,  
0)})  
(CU15942,{(CU15942,22.5,275,0,0),(CU15942,25,225,0,0),(CU15942,27.5,2  
50,0,0)})  
(CU15957,{(CU15957,25,0,12,0),(CU15957,22.5,0,13,0),(CU15957,27.5,0,1  
1,0)})  
(CU15960,{(CU15960,22.5,0,2,0),(CU15960,25,0,1,0),(CU15960,27.5,0,0,0  
)})  
(CU15979,{(CU15979,27.5,0,3,0),(CU15979,25,0,4,0),(CU15979,22.5,0,5,0  
)})  
(CU15988,{(CU15988,27.5,1200,2,135),(CU15988,22.5,1320,4,235),(CU1598  
8,25,1080,3,35)})  
grunt> average= foreach grouped generate group, AVG(monthly_cash_acco  
unts.checking_amount), AVG(monthly_cash_accounts.bank_funds), AVG(mon  
thly_cash_accounts.monthly_checks_written), AVG(monthly_cash_accounts  
.t_amount autom payments);  
grunt> dump average;
```

The command "dump average;" is highlighted with a red rectangle.

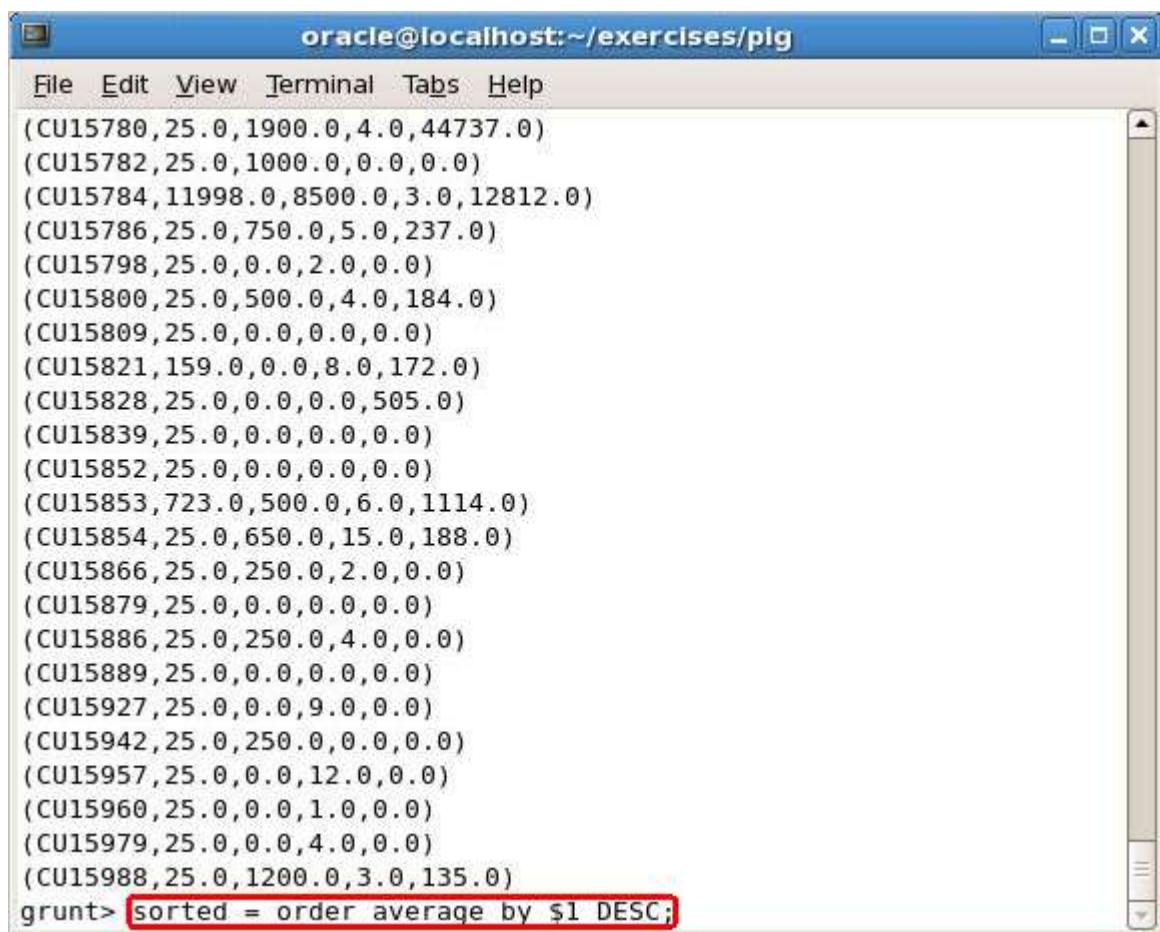
Now you can see on the screen a dump of all customer ids with their respective average account values. A couple of them are highlighted in the screen shot below.



```
oracle@localhost:~/exercises/pig
File Edit View Terminal Tabs Help
(CU15780,25.0,1900.0,4.0,44737.0)
(CU15782,25.0,1000.0,0.0,0.0)
(CU15784,11998.0,8500.0,3.0,12812.0)
(CU15786,25.0,750.0,5.0,237.0)
(CU15798,25.0,0.0,2.0,0.0)
(CU15800,25.0,500.0,4.0,184.0)
(CU15809,25.0,0.0,0.0,0.0)
(CU15821,159.0,0.0,8.0,172.0)
(CU15828,25.0,0.0,0.0,505.0)
(CU15839,25.0,0.0,0.0,0.0)
(CU15852,25.0,0.0,0.0,0.0)
(CU15853,723.0,500.0,6.0,1114.0)
(CU15854,25.0,650.0,15.0,188.0)
(CU15866,25.0,250.0,2.0,0.0)
(CU15879,25.0,0.0,0.0,0.0)
(CU15886,25.0,250.0,4.0,0.0)
(CU15889,25.0,0.0,0.0,0.0)
(CU15927,25.0,0.0,9.0,0.0)
(CU15942,25.0,250.0,0.0,0.0)
(CU15957,25.0,0.0,12.0,0.0)
(CU15960,25.0,0.0,1.0,0.0)
(CU15979,25.0,0.0,4.0,0.0)
(CU15988,25.0,1200.0,3.0,135.0)
grunt>
```

12. Now that we have calculated the average monthly values for each customer id it would be nice if we had them in order from highest to lowest checking_amount. Let's get that list, so go to the grunt shell and type:

```
sorted = order average by $1 DESC;  
Then press Enter
```

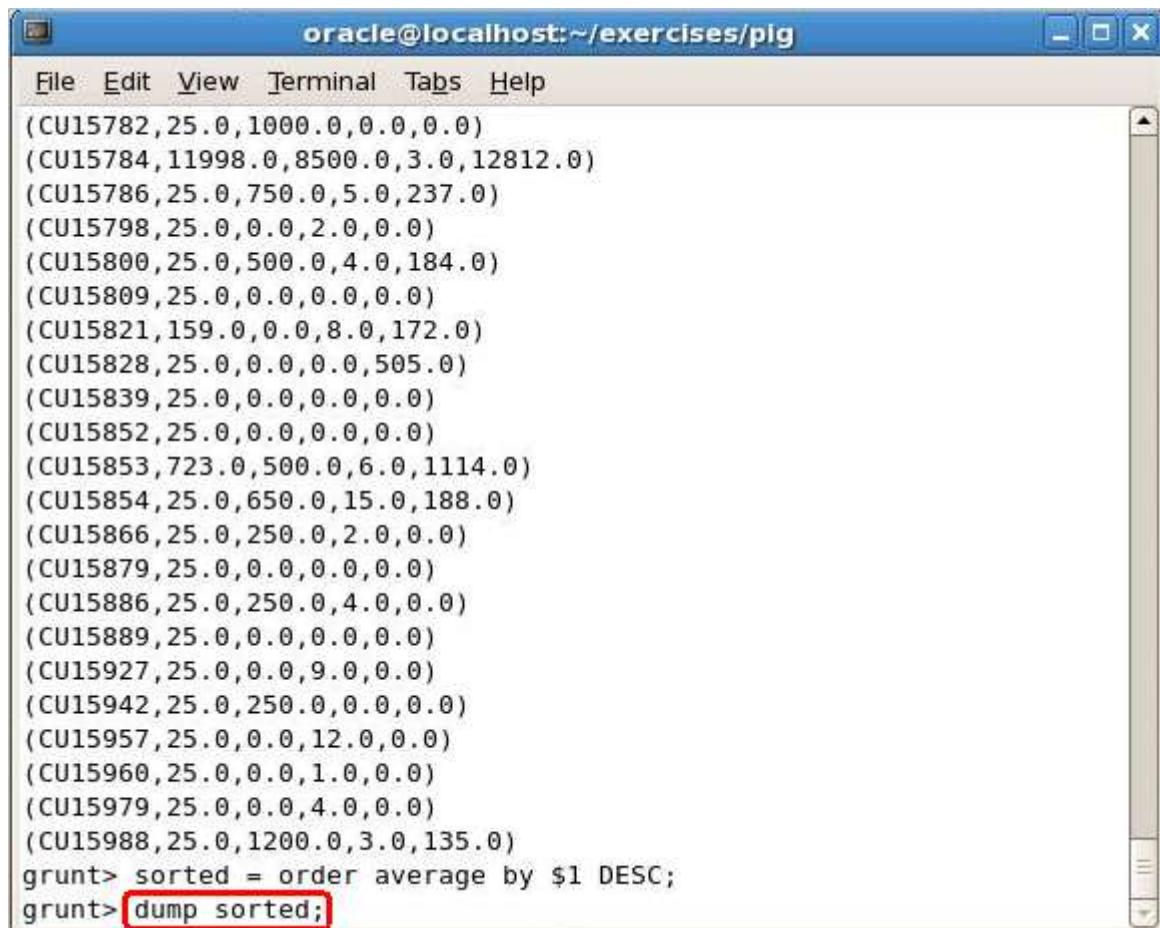


The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window contains a list of tuples representing customer data. Each tuple consists of a customer ID followed by five numerical values. The data is sorted in descending order of the fifth value (average). The last line of the terminal shows the command "sorted = order average by \$1 DESC;" which was entered by the user.

Customer ID	Value 1	Value 2	Value 3	Value 4	Average
CU15780	25.0	1900.0	4.0	44737.0	
CU15782	25.0	1000.0	0.0	0.0	
CU15784	11998.0	8500.0	3.0	12812.0	
CU15786	25.0	750.0	5.0	237.0	
CU15798	25.0	0.0	2.0	0.0	
CU15800	25.0	500.0	4.0	184.0	
CU15809	25.0	0.0	0.0	0.0	
CU15821	159.0	0.0	8.0	172.0	
CU15828	25.0	0.0	0.0	505.0	
CU15839	25.0	0.0	0.0	0.0	
CU15852	25.0	0.0	0.0	0.0	
CU15853	723.0	500.0	6.0	1114.0	
CU15854	25.0	650.0	15.0	188.0	
CU15866	25.0	250.0	2.0	0.0	
CU15879	25.0	0.0	0.0	0.0	
CU15886	25.0	250.0	4.0	0.0	
CU15889	25.0	0.0	0.0	0.0	
CU15927	25.0	0.0	9.0	0.0	
CU15942	25.0	250.0	0.0	0.0	
CU15957	25.0	0.0	12.0	0.0	
CU15960	25.0	0.0	1.0	0.0	
CU15979	25.0	0.0	4.0	0.0	
CU15988	25.0	1200.0	3.0	135.0	

We can now see what the sorted list looks like. Go to the grunt terminal and type:

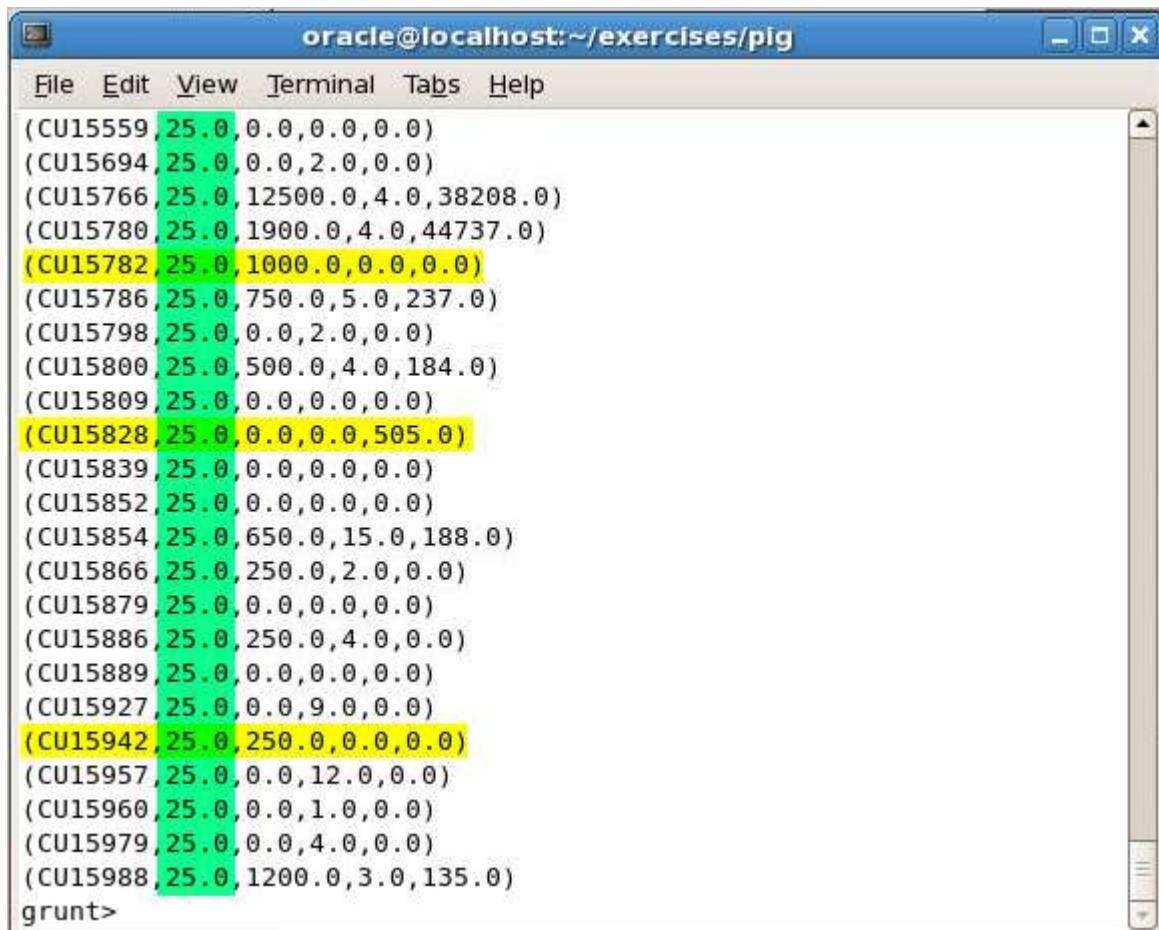
```
dump sorted;  
Then press Enter
```



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window contains a list of tuples sorted by the second column in descending order. The tuples represent CU numbers and their average values. The command "dump sorted;" is highlighted with a red rectangle at the bottom of the list.

```
(CU15782,25.0,1000.0,0.0,0.0)  
(CU15784,11998.0,8500.0,3.0,12812.0)  
(CU15786,25.0,750.0,5.0,237.0)  
(CU15798,25.0,0.0,2.0,0.0)  
(CU15800,25.0,500.0,4.0,184.0)  
(CU15809,25.0,0.0,0.0,0.0)  
(CU15821,159.0,0.0,8.0,172.0)  
(CU15828,25.0,0.0,0.0,505.0)  
(CU15839,25.0,0.0,0.0,0.0)  
(CU15852,25.0,0.0,0.0,0.0)  
(CU15853,723.0,500.0,6.0,1114.0)  
(CU15854,25.0,650.0,15.0,188.0)  
(CU15866,25.0,250.0,2.0,0.0)  
(CU15879,25.0,0.0,0.0,0.0)  
(CU15886,25.0,250.0,4.0,0.0)  
(CU15889,25.0,0.0,0.0,0.0)  
(CU15927,25.0,0.0,9.0,0.0)  
(CU15942,25.0,250.0,0.0,0.0)  
(CU15957,25.0,0.0,12.0,0.0)  
(CU15960,25.0,0.0,1.0,0.0)  
(CU15979,25.0,0.0,4.0,0.0)  
(CU15988,25.0,1200.0,3.0,135.0)  
grunt> sorted = order average by $1 DESC;  
grunt> dump sorted;
```

On the screen you now see the list sorted in descending order. Displayed on the screen are accounts with the lowest checking amounts (column highlighted in green) but you can scroll up the see the rest of the values.

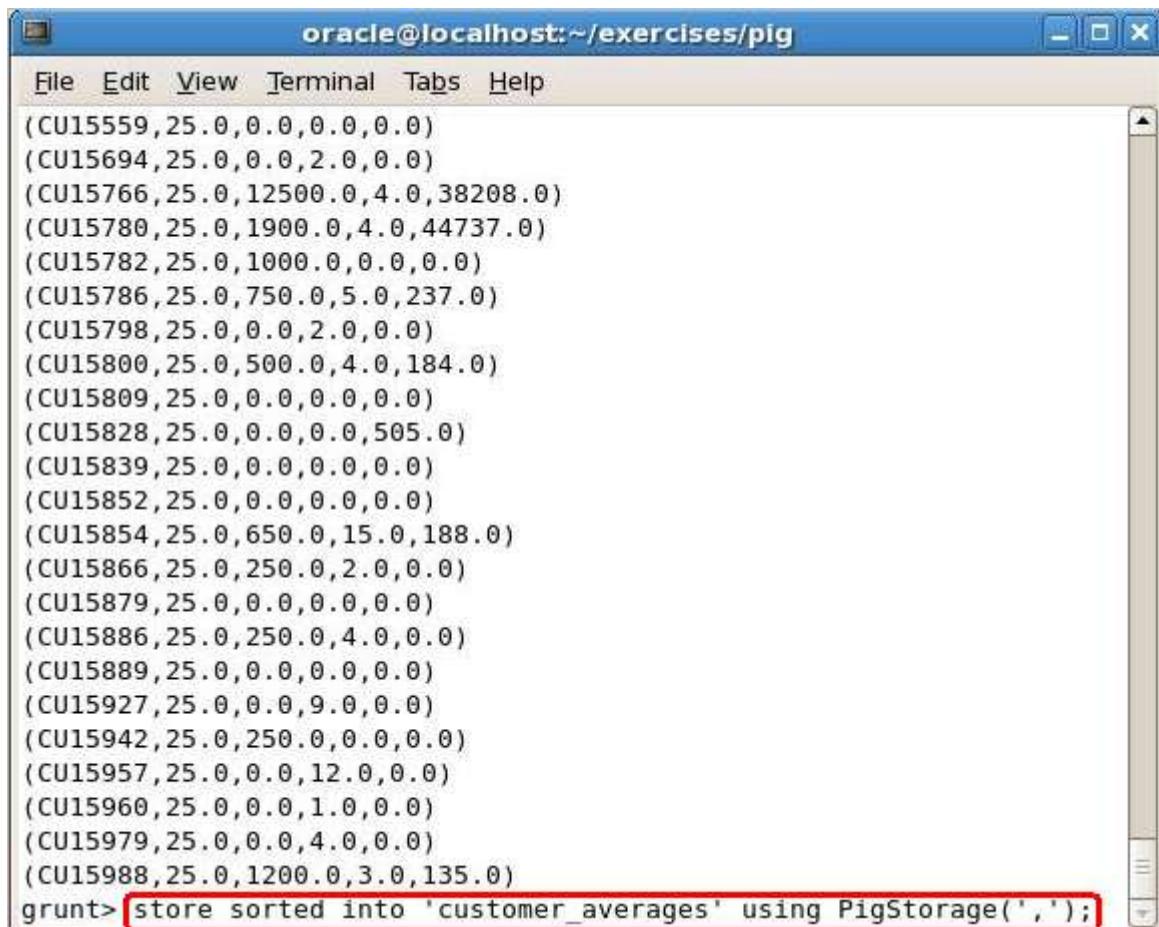


```
oracle@localhost:~/exercises/pig
File Edit View Terminal Tabs Help
(CU15559,25.0,0.0,0.0,0.0)
(CU15694,25.0,0.0,2.0,0.0)
(CU15766,25.0,12500.0,4.0,38208.0)
(CU15780,25.0,1900.0,4.0,44737.0)
(CU15782,25.0,1000.0,0.0,0.0)
(CU15786,25.0,750.0,5.0,237.0)
(CU15798,25.0,0.0,2.0,0.0)
(CU15800,25.0,500.0,4.0,184.0)
(CU15809,25.0,0.0,0.0,0.0)
(CU15828,25.0,0.0,0.0,505.0)
(CU15839,25.0,0.0,0.0,0.0)
(CU15852,25.0,0.0,0.0,0.0)
(CU15854,25.0,650.0,15.0,188.0)
(CU15866,25.0,250.0,2.0,0.0)
(CU15879,25.0,0.0,0.0,0.0)
(CU15886,25.0,250.0,4.0,0.0)
(CU15889,25.0,0.0,0.0,0.0)
(CU15927,25.0,0.0,9.0,0.0)
(CU15942,25.0,250.0,0.0,0.0)
(CU15957,25.0,0.0,12.0,0.0)
(CU15960,25.0,0.0,1.0,0.0)
(CU15979,25.0,0.0,4.0,0.0)
(CU15988,25.0,1200.0,3.0,135.0)
grunt>
```

13. We now have the final results that we want. We will proceed to write these results out to HDFS, as we will need them in a file that will be processed by the Oracle SQL Connector for HDFS (OSCH) in a later exercise. The OSCH will take the data out of the HDFS file and load it into the Oracle Database, where we want to create a 360 degree view of our customers. Go to the grunt shell and type:

```
store sorted into 'customer_averages' using PigStorage(',');
```

Then press **Enter**



```
(CU15559,25.0,0.0,0.0,0.0)
(CU15694,25.0,0.0,2.0,0.0)
(CU15766,25.0,12500.0,4.0,38208.0)
(CU15780,25.0,1900.0,4.0,44737.0)
(CU15782,25.0,1000.0,0.0,0.0)
(CU15786,25.0,750.0,5.0,237.0)
(CU15798,25.0,0.0,2.0,0.0)
(CU15800,25.0,500.0,4.0,184.0)
(CU15809,25.0,0.0,0.0,0.0)
(CU15828,25.0,0.0,0.0,505.0)
(CU15839,25.0,0.0,0.0,0.0)
(CU15852,25.0,0.0,0.0,0.0)
(CU15854,25.0,650.0,15.0,188.0)
(CU15866,25.0,250.0,2.0,0.0)
(CU15879,25.0,0.0,0.0,0.0)
(CU15886,25.0,250.0,4.0,0.0)
(CU15889,25.0,0.0,0.0,0.0)
(CU15927,25.0,0.0,9.0,0.0)
(CU15942,25.0,250.0,0.0,0.0)
(CU15957,25.0,0.0,12.0,0.0)
(CU15960,25.0,0.0,1.0,0.0)
(CU15979,25.0,0.0,4.0,0.0)
(CU15988,25.0,1200.0,3.0,135.0)
grunt> store sorted into 'customer_averages' using PigStorage(',');
```

14. The new calculated data is now permanently stored in HDFS. We can now exit the grunt shell. Go to the grunt shell and type:

```
quit;  
Then press Enter
```

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window displays the following output from a Pig Latin session:

```
Input(s):  
Successfully read 3045 records (67919 bytes) from: "hdfs://localhost.  
localdomain:8020/user/oracle/export_monthly_cash_accounts.csv"  
  
Output(s):  
Successfully stored 1015 records (29193 bytes) in: "hdfs://localhost.  
localdomain:8020/user/oracle/customer_averages"  
  
Counters:  
Total records written : 1015  
Total bytes written : 29193  
Spillable Memory Manager spill count : 0  
Total bags proactively spilled: 0  
Total records proactively spilled: 0  
  
Job DAG:  
job_201307290906_0055 -> job_201307290906_0056,  
job_201307290906_0056 -> job_201307290906_0057,  
job_201307290906_0057  
  
2013-07-30 06:11:48,510 [main] INFO org.apache.pig.backend.hadoop.ex  
ecutionengine.mapReduceLayer.MapReduceLauncher - Success!  
grunt> quit;
```

15. Now back at the terminal let's view some of the content of the HDFS file. Go to the terminal and type:

```
hadoop fs -cat /user/oracle/customer_averages/part-r-00000 | head
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window displays the output of a Pig job. It includes sections for Output(s), Counters, Job DAG, and a timestamped log message. At the bottom, a command is entered: "hadoop fs -cat /user/oracle/customer_averages/part-r-00000 | head". This command is highlighted with a red rectangle.

```
oracle@localhost:~/exercises/pig
File Edit View Terminal Tabs Help
localdomain:8020/user/oracle/export_monthly_cash_accounts.csv"

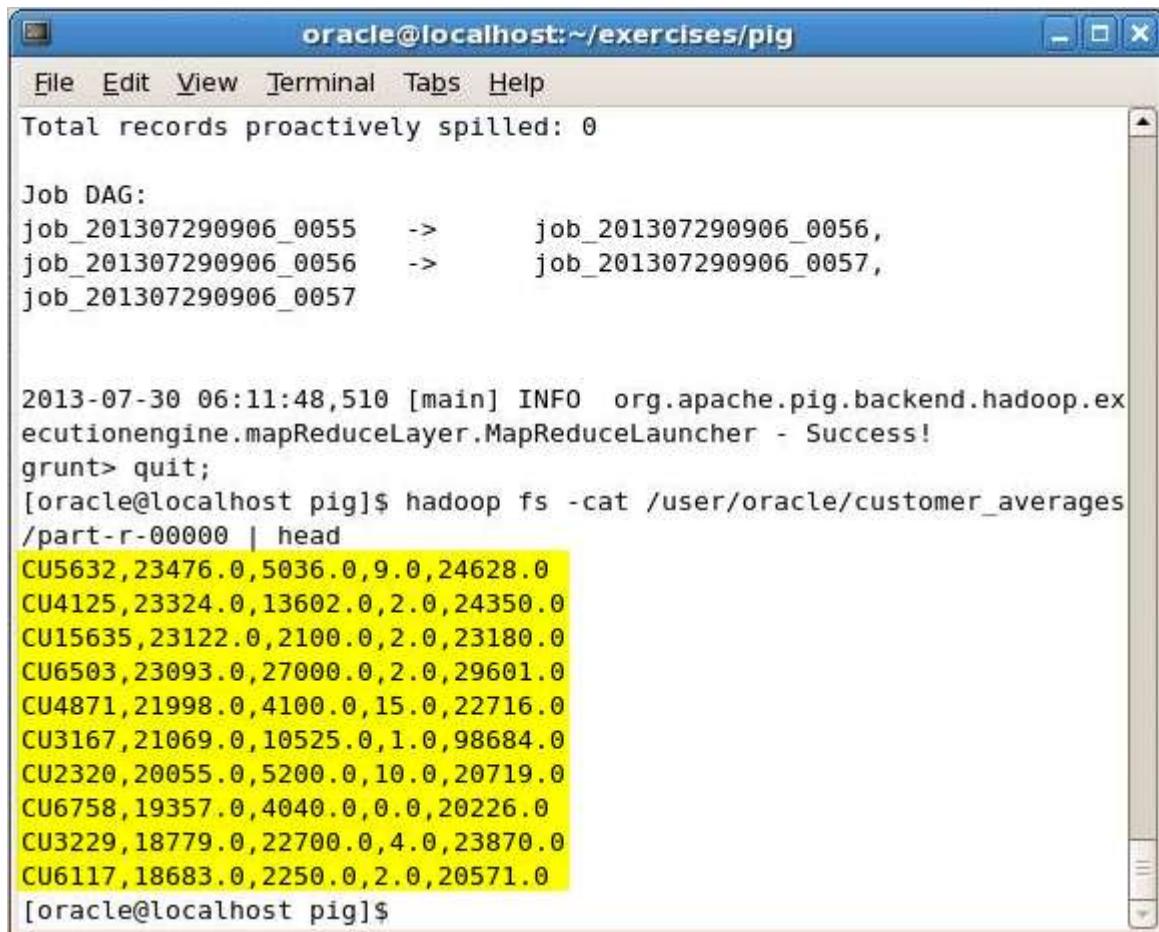
Output(s):
Successfully stored 1015 records (29193 bytes) in: "hdfs://localhost.
localdomain:8020/user/oracle/customer_averages"

Counters:
Total records written : 1015
Total bytes written : 29193
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_201307290906_0055    ->      job_201307290906_0056,
job_201307290906_0056    ->      job_201307290906_0057,
job_201307290906_0057

2013-07-30 06:11:48,510 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
grunt> quit;
[oracle@localhost pig]$ hadoop fs -cat /user/oracle/customer_averages
/part-r-00000 | head
```

This command simply did cat on the results file available in the HDFS. The results are seen on the screen.



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window contains the following text:

```
Total records proactively spilled: 0

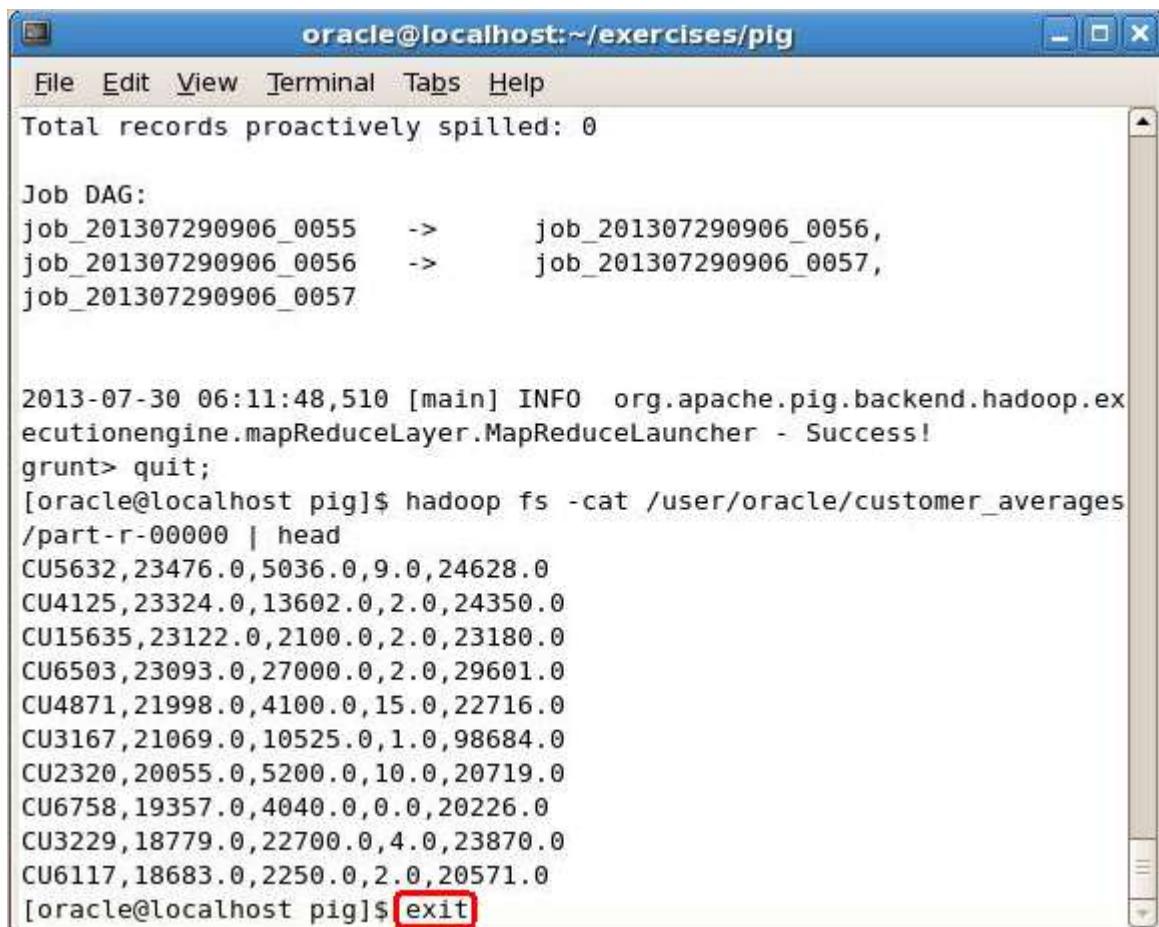
Job DAG:
job_201307290906_0055    ->      job_201307290906_0056,
job_201307290906_0056    ->      job_201307290906_0057,
job_201307290906_0057

2013-07-30 06:11:48,510 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
grunt> quit;
[oracle@localhost pig]$ hadoop fs -cat /user/oracle/customer_averages/part-r-00000 | head
CU5632,23476.0,5036.0,9.0,24628.0
CU4125,23324.0,13602.0,2.0,24350.0
CU15635,23122.0,2100.0,2.0,23180.0
CU6503,23093.0,27000.0,2.0,29601.0
CU4871,21998.0,4100.0,15.0,22716.0
CU3167,21069.0,10525.0,1.0,98684.0
CU2320,20055.0,5200.0,10.0,20719.0
CU6758,19357.0,4040.0,0.0,20226.0
CU3229,18779.0,22700.0,4.0,23870.0
CU6117,18683.0,2250.0,2.0,20571.0
[oracle@localhost pig]$
```

16. That concludes the Pig exercise. You can now close the terminal window. Go to the terminal and type:

```
exit
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/pig". The window contains the following text output from a Pig session:

```
Total records proactively spilled: 0

Job DAG:
job_201307290906_0055 -> job_201307290906_0056,
job_201307290906_0056 -> job_201307290906_0057,
job_201307290906_0057

2013-07-30 06:11:48,510 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
grunt> quit;
[oracle@localhost pig]$ hadoop fs -cat /user/oracle/customer_averages/part-r-00000 | head
CU5632,23476.0,5036.0,9.0,24628.0
CU4125,23324.0,13602.0,2.0,24350.0
CU15635,23122.0,2100.0,2.0,23180.0
CU6503,23093.0,27000.0,2.0,29601.0
CU4871,21998.0,4100.0,15.0,22716.0
CU3167,21069.0,10525.0,1.0,98684.0
CU2320,20055.0,5200.0,10.0,20719.0
CU6758,19357.0,4040.0,0.0,20226.0
CU3229,18779.0,22700.0,4.0,23870.0
CU6117,18683.0,2250.0,2.0,20571.0
[oracle@localhost pig]$ exit
```

5.4 Summary

In this exercise you saw what a pig script looks like and how to run it. It is important to understand that Pig is a scripting language which ultimately runs Map/Reduce jobs on a Hadoop cluster hence all of the power of a distributed system and the high data volume / size which HDFS can accommodate are exploitable through Pig. Pig provides an easier interface to the Map/Reduce infrastructure allowing for a scripting paradigm to be used rather than direct Java coding.

6. HIVE AND IMPALA

6.1 Introduction to Hive

Hive is a data warehouse system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time this language also allows traditional map/reduce programmers to plug in their custom Mappers and Reducers when it is inconvenient or inefficient to express this logic in HiveQL.

Hive is not a database, but is another map-reduce generator like Pig that uses SQL like language and metadata instead of a 4GL language.

The main components of Hive are:

- UI - The user interface for users to submit queries and other operations to the system. Currently the system has a command line interface and a web based GUI is being developed.
- Driver - The component which receives the queries. This component implements the notion of session handles and provides execute and fetch APIs modeled on JDBC/ODBC interfaces.
- Compiler - The component that parses the query does semantic analysis on the different query blocks and query expressions and eventually generates an execution plan with the help of the table and partition metadata looked up from the metastore.
- Metastore - The component that stores all the structure information of the various table and partitions in the warehouse including column and column type information, the serializers and deserializers necessary to read and write data and the corresponding hdfs files where the data is stored.
- Execution Engine - The component which executes the execution plan created by the compiler. Typically running stages of map reduce code to satisfy the execution plan.

6.2 Overview of Hands On Exercise

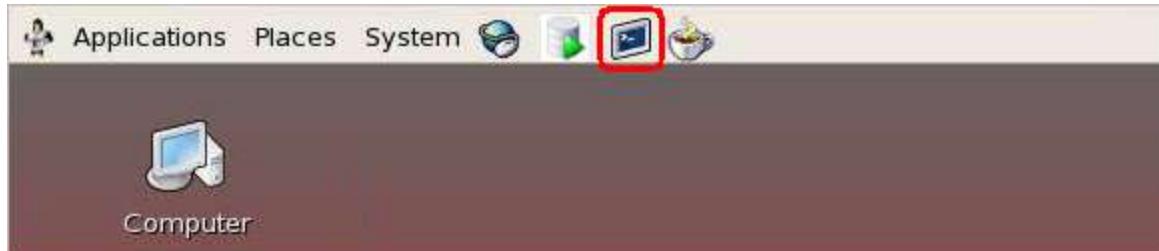
In this exercise you will use Hive Query Language to create tables, insert data into those tables and run queries on that data. For this exercise we will use two datafiles that contain details related to customer credits and mortgages. The purpose is to perform some aggregations on the data and store them into a Hive table. The content of this table will be then loaded into the Oracle Database via Oracle Data Integrator in a later exercise.

In this exercise you will:

1. Upload files into HDFS
2. Create tables in Hive
3. Load the data into the Hive tables
4. Run queries on the Hive tables and create tables that will hold our final aggregated data

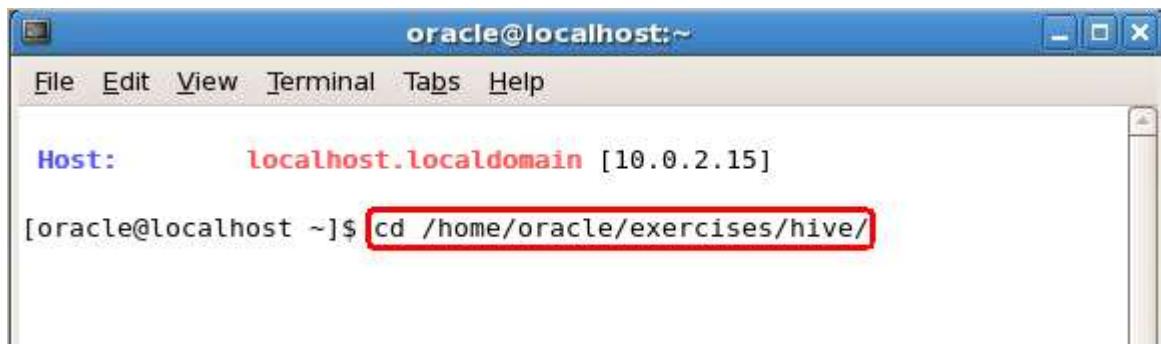
6.3 Queries with Hive

1. All of the setup and execution for this exercise can be done from the terminal, hence open a terminal by clicking on the **Terminal icon** on the desktop.



2. To get into the folder where the scripts for the Hive exercise are, type in the terminal:

```
cd /home/oracle/exercises/hive  
Then press Enter
```

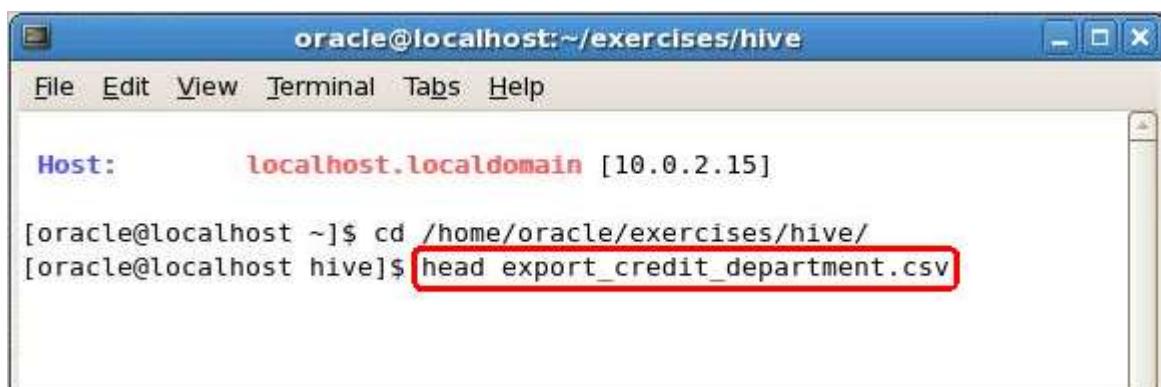


```
oracle@localhost:~  
File Edit View Terminal Tabs Help  
  
Host: localhost.localdomain [10.0.2.15]  
  
[oracle@localhost ~]$ cd /home/oracle/exercises/hive/
```

3. Let's take a look at the first file that holds the credits of customers. Go to the terminal and type

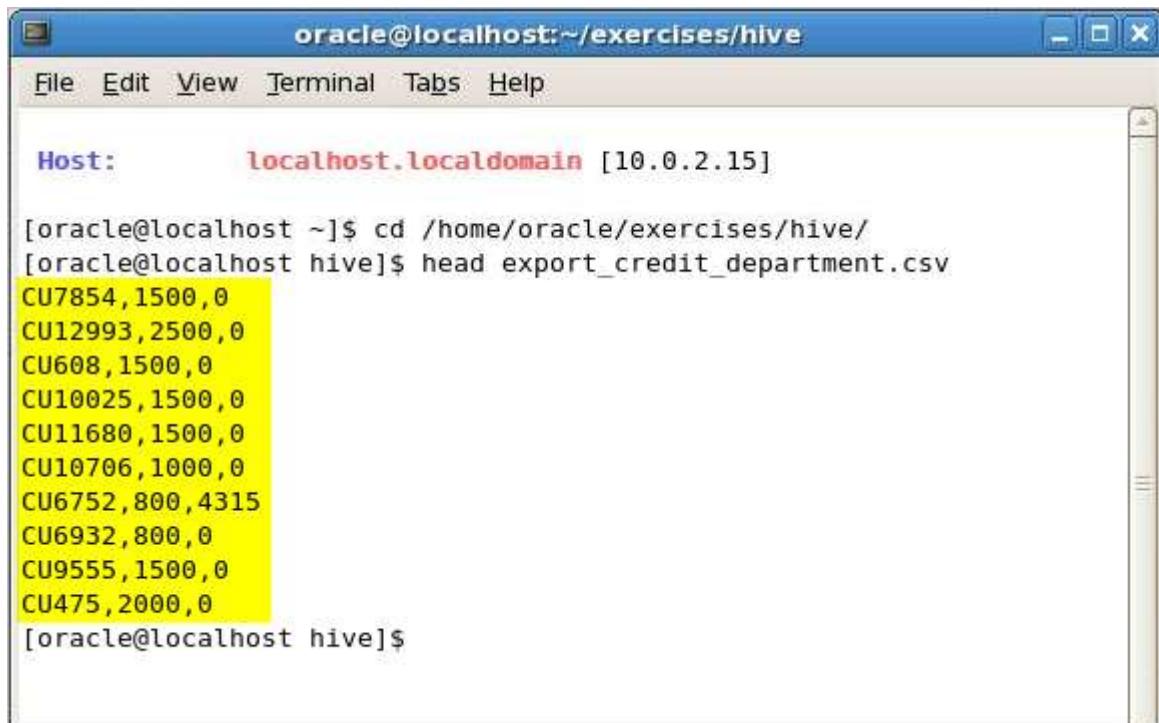
```
head export_credit_department.csv
```

Then press **Enter**



```
oracle@localhost:~/exercises/hive  
File Edit View Terminal Tabs Help  
  
Host: localhost.localdomain [10.0.2.15]  
  
[oracle@localhost ~]$ cd /home/oracle/exercises/hive/  
[oracle@localhost hive]$ head export_credit_department.csv
```

The comma-delimited file contains the customer id, followed by the credit card limit and the credit balance, for each customer.



A screenshot of a terminal window titled "oracle@localhost:~/exercises/hive". The window has a menu bar with File, Edit, View, Terminal, Tabs, and Help. The host information is shown as "Host: localhost.localdomain [10.0.2.15]". The terminal session shows the user navigating to the directory "/home/oracle/exercises/hive/" and running the command "head export_credit_department.csv". The output displays the first ten lines of the CSV file, which contain customer IDs and their corresponding credit limits and balances. The entire output is highlighted with a yellow box.

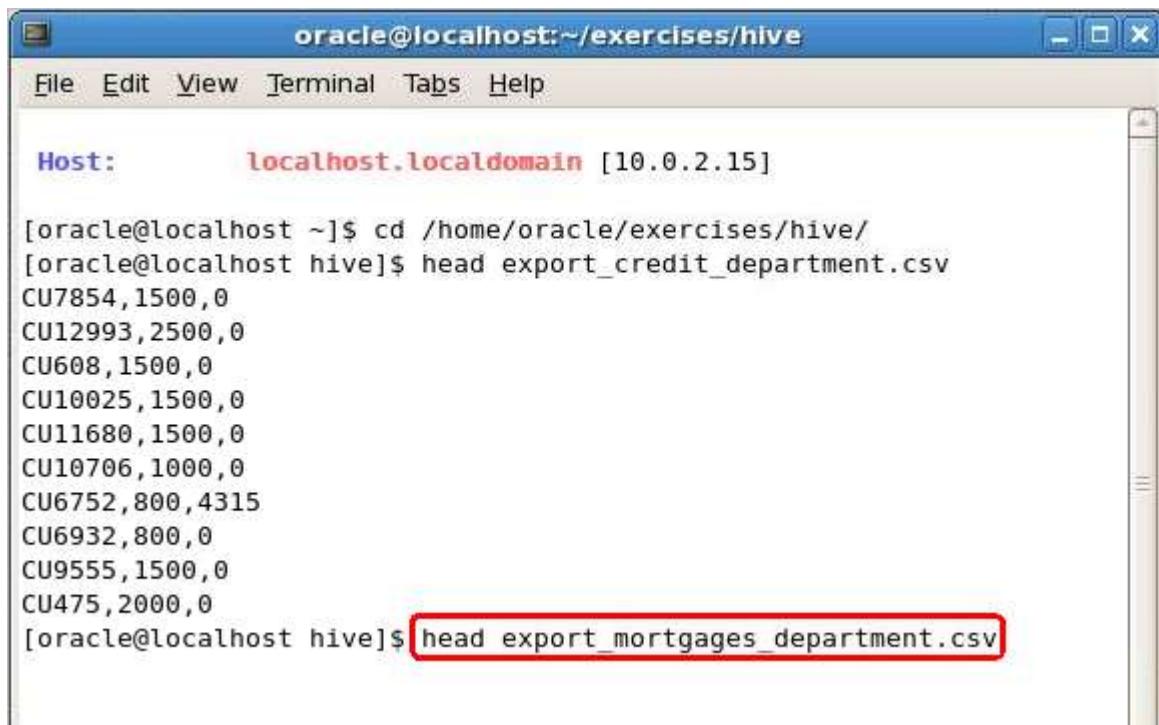
```
Host: localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/hive/
[oracle@localhost hive]$ head export_credit_department.csv
CU7854,1500,0
CU12993,2500,0
CU608,1500,0
CU10025,1500,0
CU11680,1500,0
CU10706,1000,0
CU6752,800,4315
CU6932,800,0
CU9555,1500,0
CU475,2000,0
[oracle@localhost hive]$
```

4. Let's take a look at the mortgages file that holds the customer mortgages. Go to the terminal and type

```
head export_mortgages_department.csv
```

Then press **Enter**

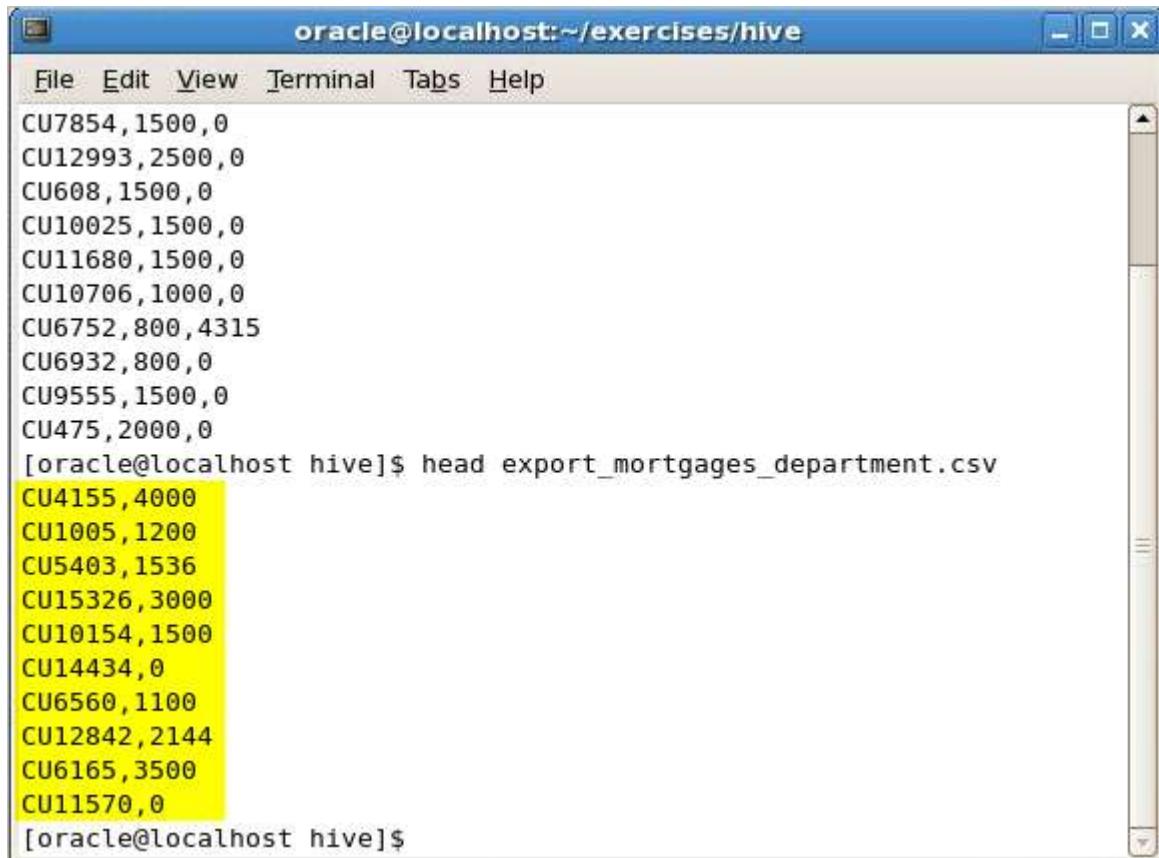


The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window has a menu bar with File, Edit, View, Terminal, Tabs, and Help. Below the menu is a status line showing "Host: localhost.localdomain [10.0.2.15]". The terminal window contains the following text:

```
[oracle@localhost ~]$ cd /home/oracle/exercises/hive/  
[oracle@localhost hive]$ head export_credit_department.csv  
CU7854,1500,0  
CU12993,2500,0  
CU608,1500,0  
CU10025,1500,0  
CU11680,1500,0  
CU10706,1000,0  
CU6752,800,4315  
CU6932,800,0  
CU9555,1500,0  
CU475,2000,0  
[oracle@localhost hive]$ head export_mortgages_department.csv
```

The last line of the command, "head export_mortgages_department.csv", is highlighted with a red rectangle.

This file holds two columns, delimited by a comma. The first column is the customer id, and the second column is the mortgage amount. In case a customer has two mortgages there will be two rows for the same customer in the file. In case a customer has no mortgages the mortgage amount will be 0.



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

```
File Edit View Terminal Tabs Help
CU7854,1500,0
CU12993,2500,0
CU608,1500,0
CU10025,1500,0
CU11680,1500,0
CU10706,1000,0
CU6752,800,4315
CU6932,800,0
CU9555,1500,0
CU475,2000,0
[oracle@localhost hive]$ head export_mortgages_department.csv
CU4155,4000
CU1005,1200
CU5403,1536
CU15326,3000
CU10154,1500
CU14434,0
CU6560,1100
CU12842,2144
CU6165,3500
CU11570,0
[oracle@localhost hive]$
```

The lines from "CU4155,4000" down to "CU11570,0" are highlighted with a yellow background.

5. Let's proceed to copy the two files in HDFS. We will first copy the credits file into HDFS. Go to the terminal and type:

```
hadoop fs -copyFromLocal export_credit_department.csv  
/user/oracle/export_credit_department
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window displays the contents of the "export_mortgages_department.csv" file, followed by the command to copy the "export_credit_department.csv" file to HDFS. The command is highlighted with a red rectangle.

```
oracle@localhost:~/exercises/hive  
File Edit View Terminal Tabs Help  
CU12993,2500,0  
CU608,1500,0  
CU10025,1500,0  
CU11680,1500,0  
CU10706,1000,0  
CU6752,800,4315  
CU6932,800,0  
CU9555,1500,0  
CU475,2000,0  
[oracle@localhost hive]$ head export_mortgages_department.csv  
CU4155,4000  
CU1005,1200  
CU5403,1536  
CU15326,3000  
CU10154,1500  
CU14434,0  
CU6560,1100  
CU12842,2144  
CU6165,3500  
CU11570,0  
[oracle@localhost hive]$ hadoop fs -copyFromLocal export credit department.csv /user/oracle/export credit department
```

6. We will now copy the mortgages file into HDFS. Go to the terminal and type:

```
hadoop fs -copyFromLocal export_mortgages_department.csv  
/user/oracle/export_mortgages_department
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

```
File Edit View Terminal Tabs Help  
CU10025,1500,0  
CU11680,1500,0  
CU10706,1000,0  
CU6752,800,4315  
CU6932,800,0  
CU9555,1500,0  
CU475,2000,0  
[oracle@localhost hive]$ head export_mortgages_department.csv  
CU4155,4000  
CU1005,1200  
CU5403,1536  
CU15326,3000  
CU10154,1500  
CU14434,0  
CU6560,1100  
CU12842,2144  
CU6165,3500  
CU11570,0  
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_credit_department.csv /user/oracle/export_credit_department  
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_mortgages_department.csv /user/oracle/export_mortgages_department
```

The last two lines of the command are highlighted with a red rectangular box.

7. Let's now enter the Hive interactive shell environment to create tables and run queries against those tables. To give an analogy, this is similar to SQL*Plus but it's specifically designed for the HiveQL language. To enter the environment go to the terminal and type:

```
hive
```

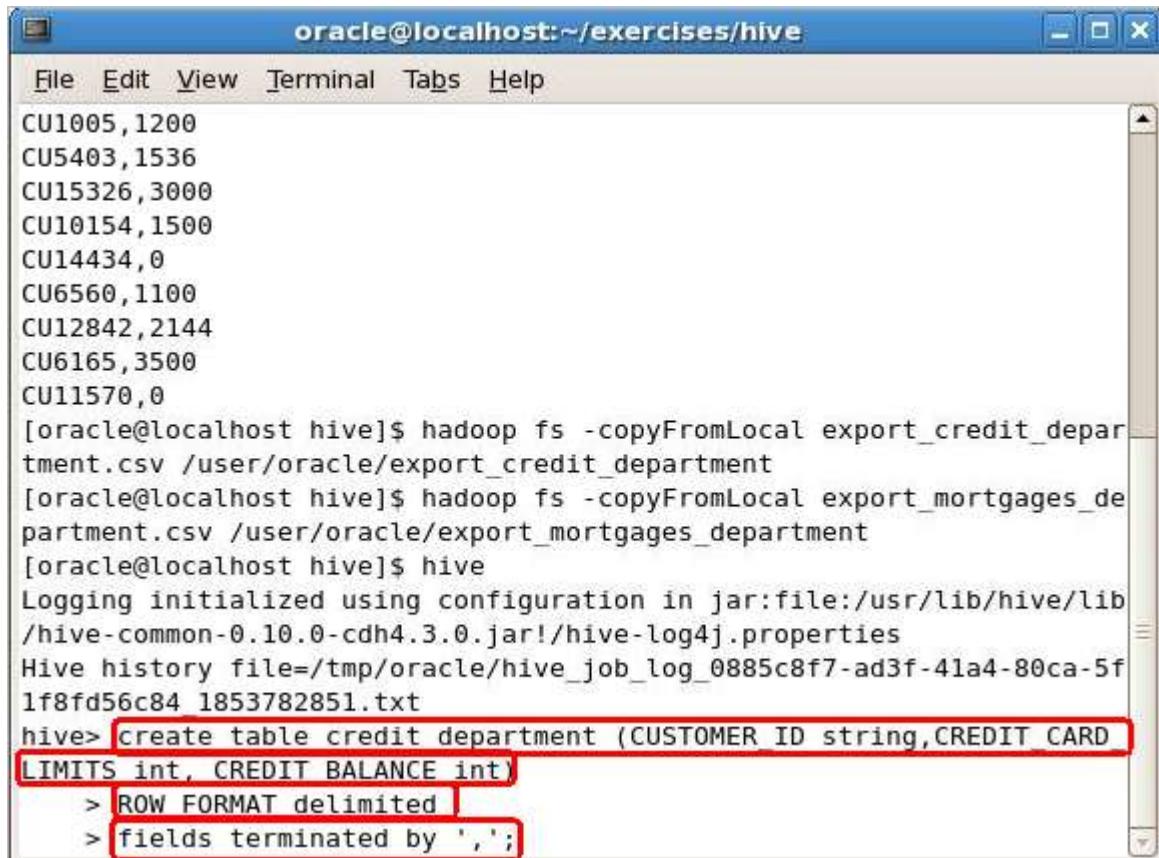
Then press **Enter**

```
oracle@localhost:~/exercises/hive
File Edit View Terminal Tabs Help
CU11680,1500,0
CU10706,1000,0
CU6752,800,4315
CU6932,800,0
CU9555,1500,0
CU475,2000,0
[oracle@localhost hive]$ head export_mortgages_department.csv
CU4155,4000
CU1005,1200
CU5403,1536
CU15326,3000
CU10154,1500
CU14434,0
CU6560,1100
CU12842,2144
CU6165,3500
CU11570,0
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_credit_department.csv /user/oracle/export_credit_department
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_mortgages_department.csv /user/oracle/export_mortgages_department
[oracle@localhost hive]$ hive
```

8. The first thing we need to do in Hive is to create the tables that will hold our credit and mortgages data. We will create a table named credit_department with three fields called customer_id, credit_card_limits and credit_balance, something that looks very natural based on the data set. In the create table command we will specify that the fields in the HDFS file are delimited by commas. Go the terminal and type:

```
create table credit_department (CUSTOMER_ID string,CREDIT_CARD_LIMITS int, CREDIT_BALANCE int) ROW FORMAT delimited fields terminated by ',';
```

Then press **Enter**

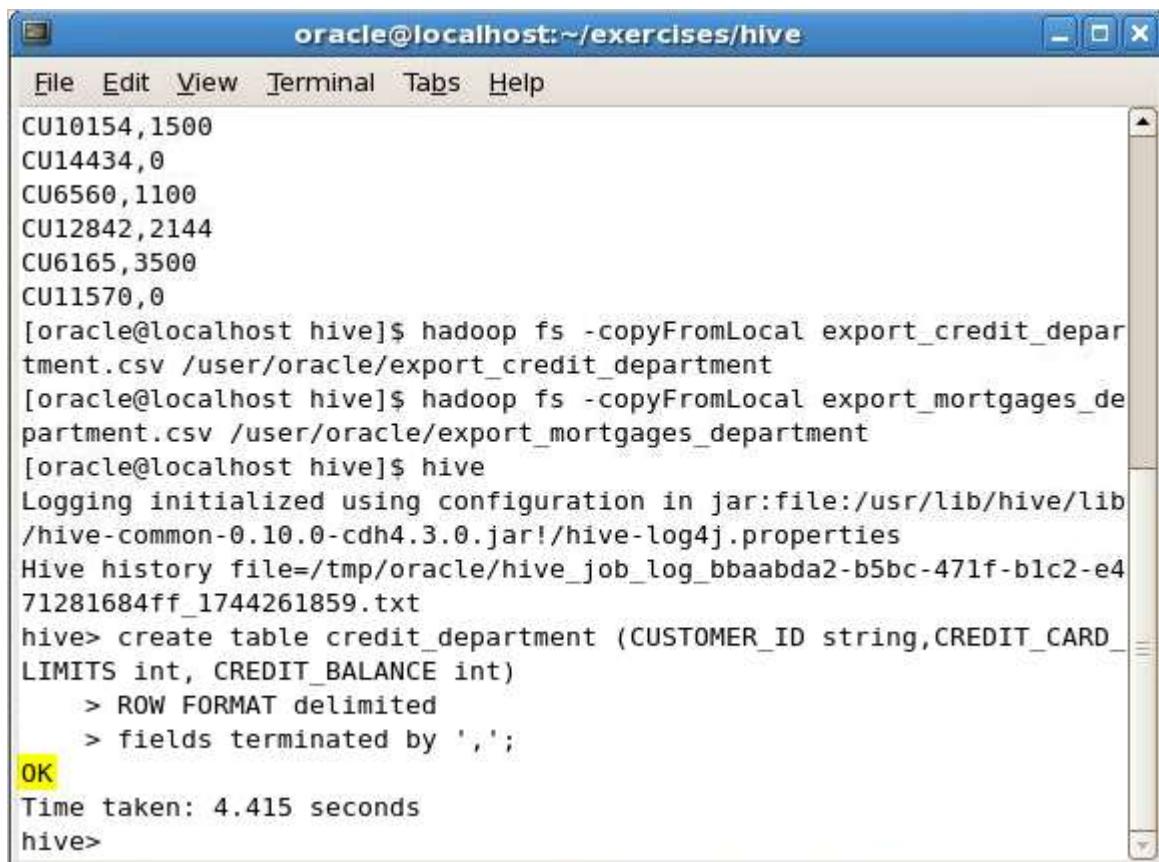


The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

```
CU1005,1200
CU5403,1536
CU15326,3000
CU10154,1500
CU14434,0
CU6560,1100
CU12842,2144
CU6165,3500
CU11570,0
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_credit_department.csv /user/oracle/export_credit_department
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_mortgages_department.csv /user/oracle/export_mortgages_department
[oracle@localhost hive]$ hive
Logging initialized using configuration in jar:file:/usr/lib/hive/lib/hive-common-0.10.0-cdh4.3.0.jar!/hive-log4j.properties
Hive history file=/tmp/oracle/hive_job_log_0885c8f7-ad3f-41a4-80ca-5f1f8fd56c84 1853782851.txt
hive> create table credit_department (CUSTOMER_ID string,CREDIT_CARD_LIMITS int, CREDIT_BALANCE int)
      > ROW FORMAT delimited
      > fields terminated by ',';
```

The last three lines of the command are highlighted with red boxes.

An OK should be printed on the screen indicating the success of the operation. This OK message will be printed for all operations but we will only mention it this time. It is left up to the user to check for this message for future HiveQL commands.



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

```
CU10154,1500
CU14434,0
CU6560,1100
CU12842,2144
CU6165,3500
CU11570,0
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_credit_department.csv /user/oracle/export_credit_department
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_mortgages_department.csv /user/oracle/export_mortgages_department
[oracle@localhost hive]$ hive
Logging initialized using configuration in jar:file:/usr/lib/hive/lib/hive-common-0.10.0-cdh4.3.0.jar!/hive-log4j.properties
Hive history file=/tmp/oracle/hive_job_log_bbaabda2-b5bc-471f-b1c2-e471281684ff_1744261859.txt
hive> create table credit_department (CUSTOMER_ID string,CREDIT_CARD_LIMITS int, CREDIT_BALANCE int)
      > ROW FORMAT delimited
      > fields terminated by ',';
OK
Time taken: 4.415 seconds
hive>
```

9. We will now create the second table called mortgages_department, with two columns, the customer_id and the mortgage_sum. Go to the terminal and type:

```
create table mortgages_department (CUSTOMER_ID string, MORTGAGE_SUM float)
ROW FORMAT delimited fields terminated by ',';
```

Then press **Enter**

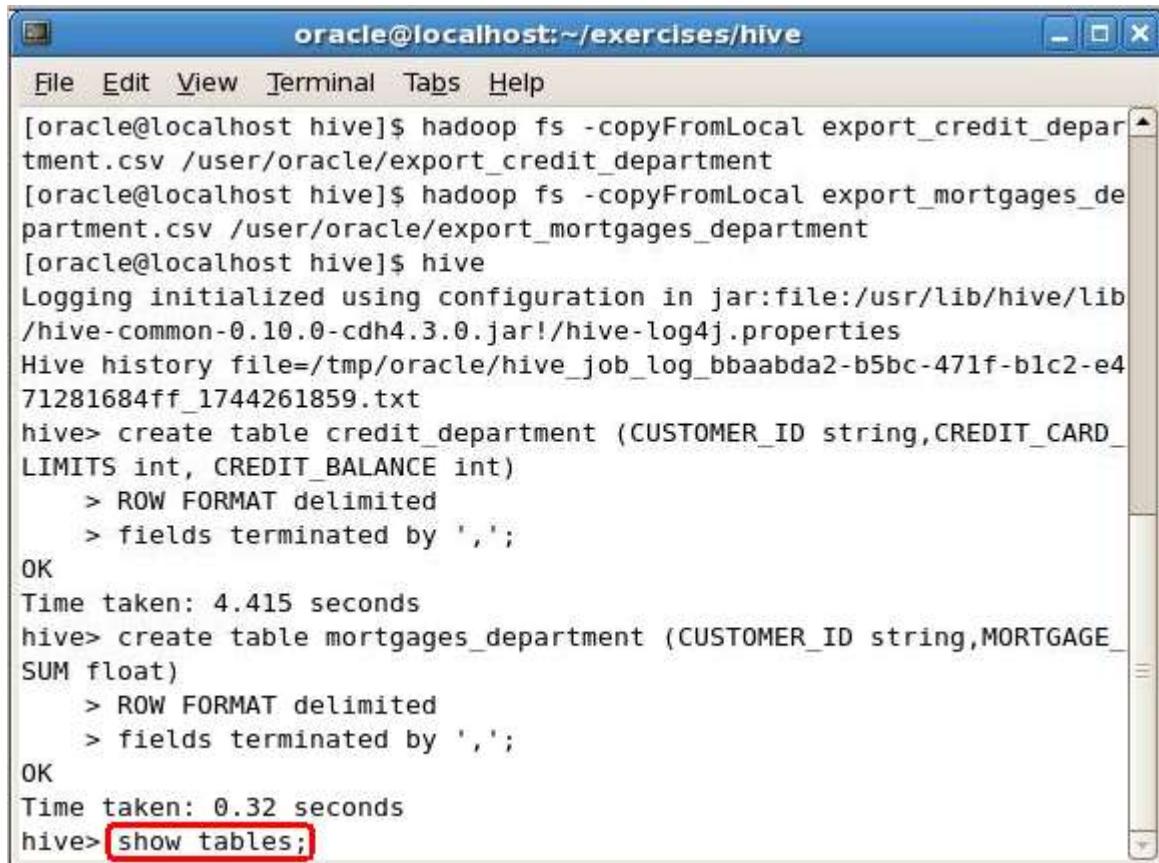
The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

```
CU12842,2144
CU6165,3500
CU11570,0
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_credit_department.csv /user/oracle/export_credit_department
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_mortgages_department.csv /user/oracle/export_mortgages_department
[oracle@localhost hive]$ hive
Logging initialized using configuration in jar:file:/usr/lib/hive/lib/hive-common-0.10.0-cdh4.3.0.jar!/hive-log4j.properties
Hive history file=/tmp/oracle/hive_job_log_bbaabda2-b5bc-471f-b1c2-e471281684ff_1744261859.txt
hive> create table credit_department (CUSTOMER_ID string, CREDIT_CARD_LIMITS int, CREDIT_BALANCE int)
    > ROW FORMAT delimited
    > fields terminated by ',';
OK
Time taken: 4.415 seconds
hive> create table mortgages_department (CUSTOMER_ID string, MORTGAGE_SUM float)
    > ROW FORMAT delimited
    > fields terminated by ',';
```

The last three lines of the command are highlighted with red boxes.

10. We can now run a command to see all of the tables available to this OS user. Go to the Hive terminal and type:

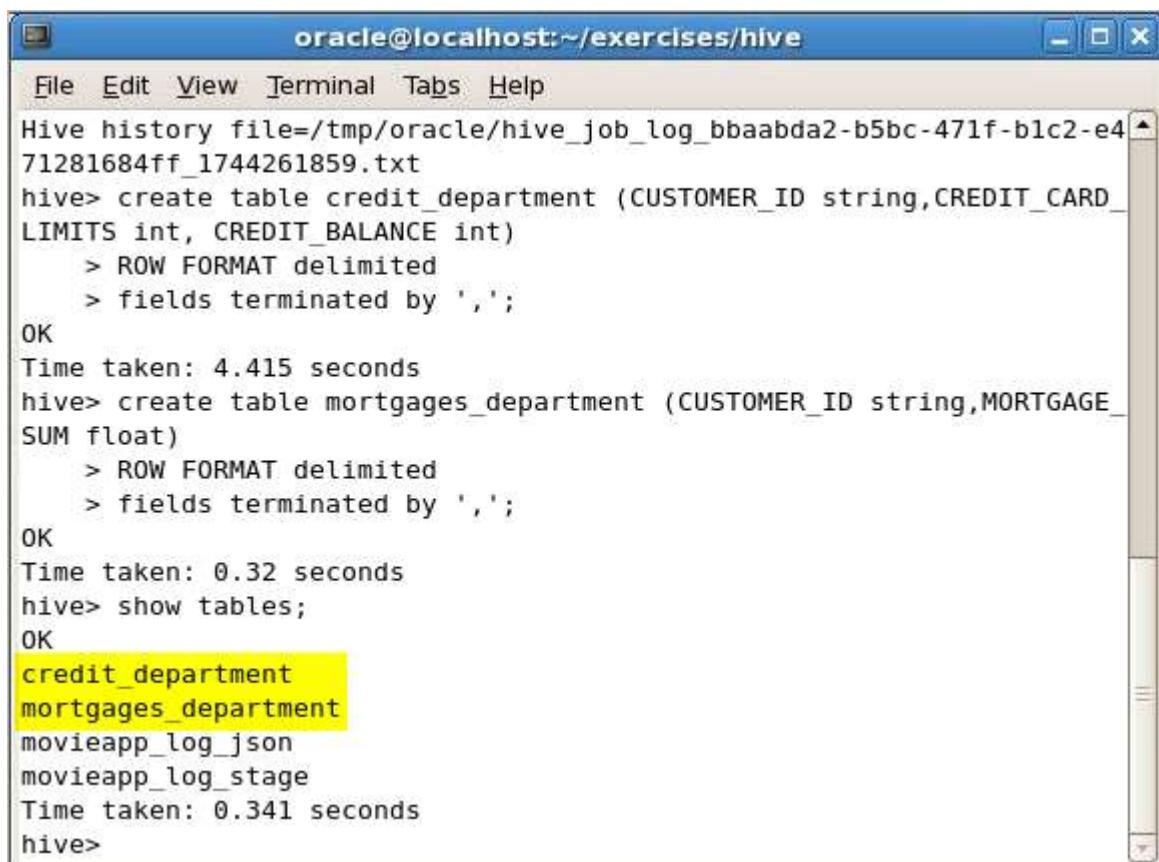
```
show tables;  
Then press Enter
```



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

```
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_credit_department.csv /user/oracle/export_credit_department  
[oracle@localhost hive]$ hadoop fs -copyFromLocal export_mortgages_department.csv /user/oracle/export_mortgages_department  
[oracle@localhost hive]$ hive  
Logging initialized using configuration in jar:file:/usr/lib/hive/lib/hive-common-0.10.0-cdh4.3.0.jar!/hive-log4j.properties  
Hive history file=/tmp/oracle/hive_job_log_bbaabda2-b5bc-471f-b1c2-e471281684ff_1744261859.txt  
hive> create table credit_department (CUSTOMER_ID string,CREDIT_CARD_LIMITS int, CREDIT_BALANCE int)  
    > ROW FORMAT delimited  
    > fields terminated by ',';  
OK  
Time taken: 4.415 seconds  
hive> create table mortgages_department (CUSTOMER_ID string,MORTGAGE_SUM float)  
    > ROW FORMAT delimited  
    > fields terminated by ',';  
OK  
Time taken: 0.32 seconds  
hive> show tables;
```

The two previously created tables are visible in the output of the `show tables` command. There are two other tables already created but we can ignore those for this exercise.



A screenshot of a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

```
Hive history file=/tmp/oracle/hive_job_log_bbaabda2-b5bc-471f-b1c2-e471281684ff_1744261859.txt
hive> create table credit_department (CUSTOMER_ID string,CREDIT_CARD_
LIMITS int, CREDIT_BALANCE int)
    > ROW FORMAT delimited
    > fields terminated by ',';
OK
Time taken: 4.415 seconds
hive> create table mortgages_department (CUSTOMER_ID string,MORTGAGE_
SUM float)
    > ROW FORMAT delimited
    > fields terminated by ',';
OK
Time taken: 0.32 seconds
hive> show tables;
OK
credit_department
mortgages_department
movieapp_log_json
movieapp_log_stage
Time taken: 0.341 seconds
hive>
```

The lines "credit_department" and "mortgages_department" are highlighted with a yellow background.

11. Let's go ahead and load some data into the two tables. Data is loaded into Hive tables from flat files available in the HDFS file system. In our case, these files were just loaded into HDFS during the previous steps of the exercise. We will first load data into the credit_department table. Go the terminal and type:

```
load data inpath '/user/oracle/export_credit_department' into table  
credit_department;
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

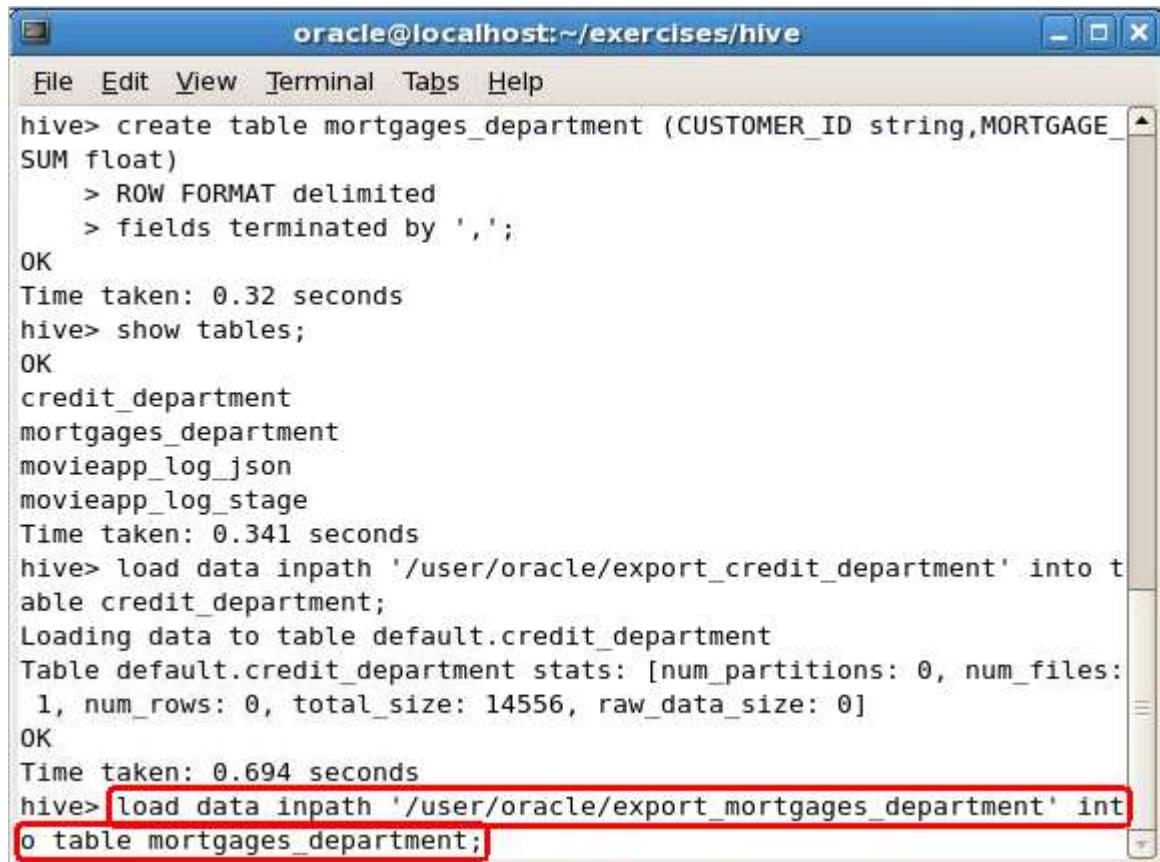
```
File Edit View Terminal Tabs Help  
71281684ff_1744261859.txt  
hive> create table credit_department (CUSTOMER_ID string,CREDIT_CARD_  
LIMITS int, CREDIT_BALANCE int)  
    > ROW FORMAT delimited  
    > fields terminated by ',';  
OK  
Time taken: 4.415 seconds  
hive> create table mortgages_department (CUSTOMER_ID string,MORTGAGE_  
SUM float)  
    > ROW FORMAT delimited  
    > fields terminated by ',';  
OK  
Time taken: 0.32 seconds  
hive> show tables;  
OK  
credit_department  
mortgages_department  
movieapp_log_json  
movieapp_log_stage  
Time taken: 0.341 seconds  
hive> load data inpath '/user/oracle/export_credit_department' into t  
able credit_department;
```

The last command in the session is highlighted with a red rectangle.

12. Next, we will load the data into the mortgages_department table. Go the terminal and type:

```
load data inpath '/user/oracle/export_mortgages_department' into table mortgages_department;
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following Hive session:

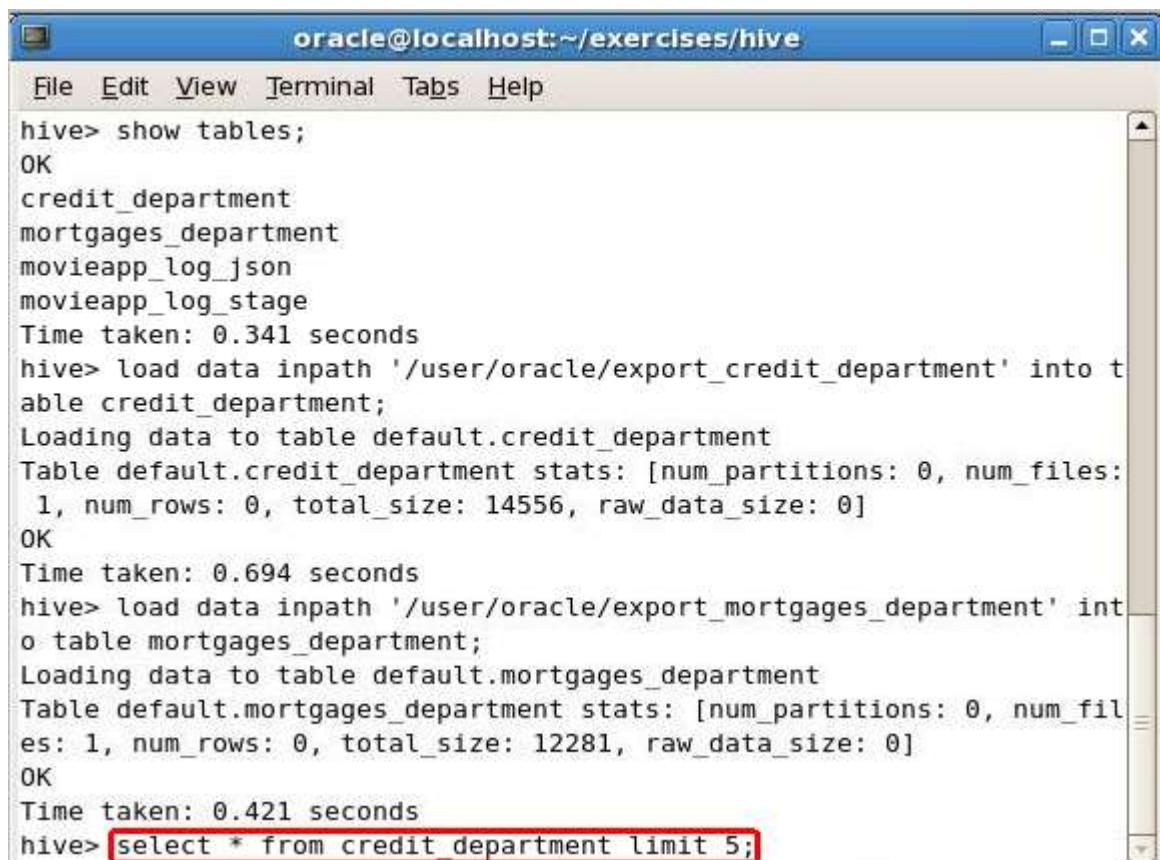
```
hive> create table mortgages_department (CUSTOMER_ID string, MORTGAGE_SUM float)
    > ROW FORMAT delimited
    > fields terminated by ',';
OK
Time taken: 0.32 seconds
hive> show tables;
OK
credit_department
mortgages_department
movieapp_log_json
movieapp_log_stage
Time taken: 0.341 seconds
hive> load data inpath '/user/oracle/export_credit_department' into table credit_department;
Loading data to table default.credit_department
Table default.credit_department stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 14556, raw_data_size: 0]
OK
Time taken: 0.694 seconds
hive> load data inpath '/user/oracle/export_mortgages_department' into table mortgages_department;
```

The last command in the session is highlighted with a red rectangle.

The data is now loaded into the two tables.

13. We can now see the data that has been loaded into the credit_department table. Go to the terminal and type:

```
select * from credit_department limit 5;  
Then press Enter
```

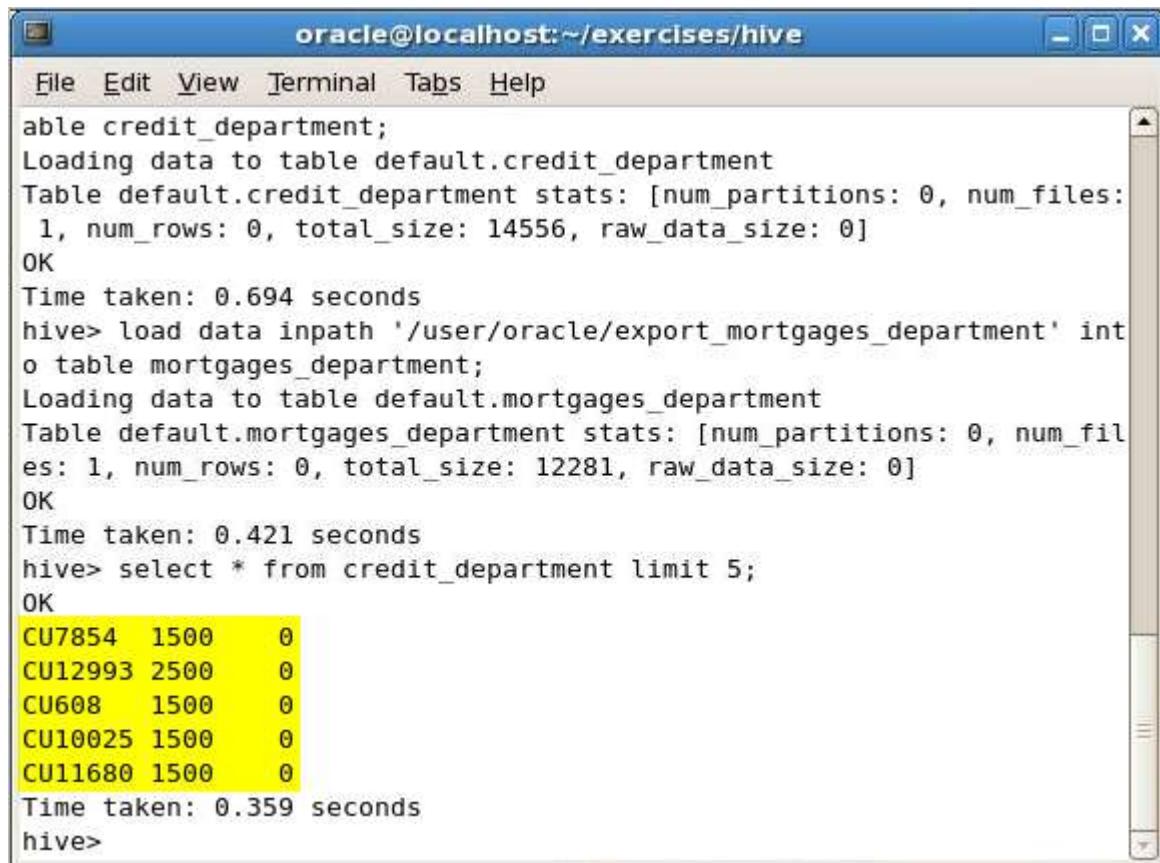


The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following Hive session:

```
hive> show tables;
OK
credit_department
mortgages_department
movieapp_log_json
movieapp_log_stage
Time taken: 0.341 seconds
hive> load data inpath '/user/oracle/export_credit_department' into table credit_department;
Loading data to table default.credit_department
Table default.credit_department stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 14556, raw_data_size: 0]
OK
Time taken: 0.694 seconds
hive> load data inpath '/user/oracle/export_mortgages_department' into table mortgages_department;
Loading data to table default.mortgages_department
Table default.mortgages_department stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 12281, raw_data_size: 0]
OK
Time taken: 0.421 seconds
hive> select * from credit_department limit 5;
```

The command `select * from credit_department limit 5;` is highlighted with a red rectangle.

The first five rows in the credit_department table have been displayed in the terminal.



A screenshot of a terminal window titled "oracle@localhost:~/exercises/hive". The window shows the following Hive session:

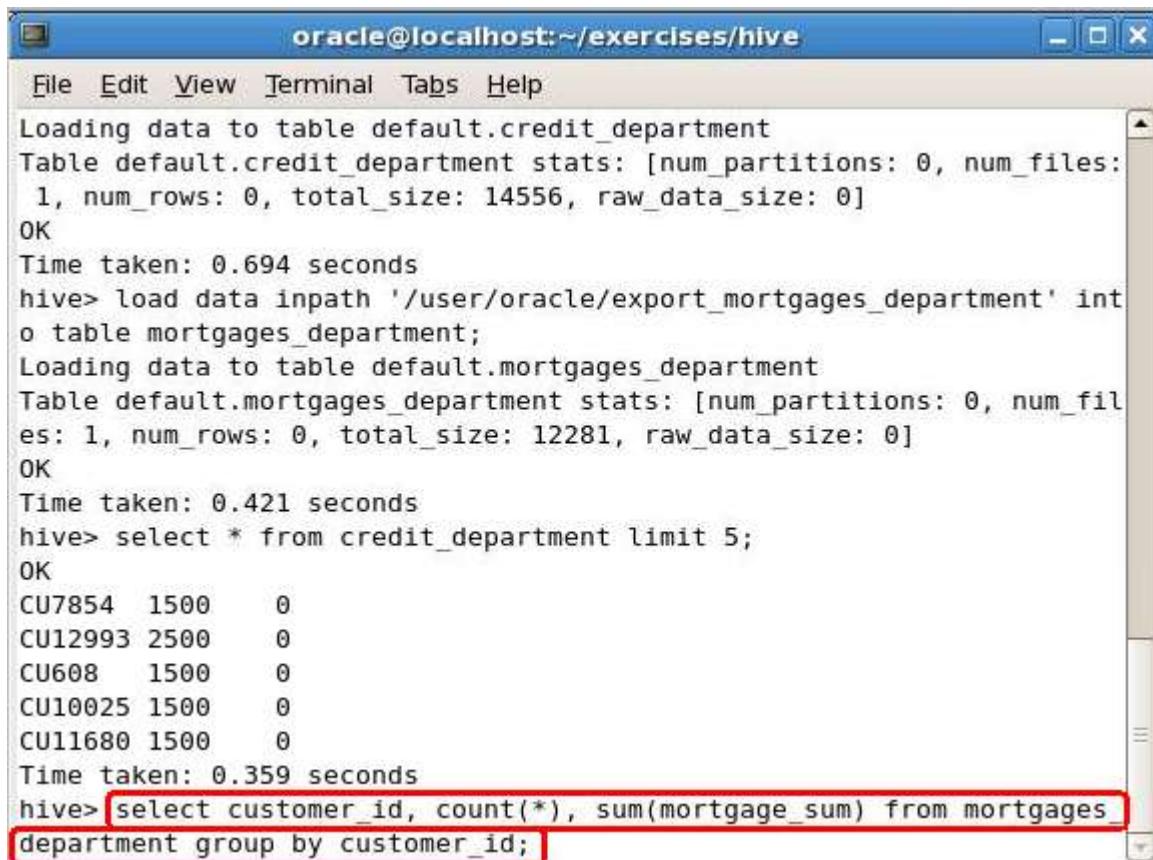
```
File Edit View Terminal Tabs Help
able credit_department;
Loading data to table default.credit_department
Table default.credit_department stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 14556, raw_data_size: 0]
OK
Time taken: 0.694 seconds
hive> load data inpath '/user/oracle/export_mortgages_department' into table mortgages_department;
Loading data to table default.mortgages_department
Table default.mortgages_department stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 12281, raw_data_size: 0]
OK
Time taken: 0.421 seconds
hive> select * from credit_department limit 5;
OK
CU7854 1500    0
CU12993 2500    0
CU608   1500    0
CU10025 1500    0
CU11680 1500    0
Time taken: 0.359 seconds
hive>
```

The output of the final "select" command is highlighted in yellow, showing the first five rows of the credit_department table.

14. Next, we will perform an aggregation of the mortgage data to see, for each customer, how many mortgages they have and what the total amount of their mortgages is. Go to the terminal and type:

```
select customer_id, count(*), sum(mortgage_sum) from mortgages_department  
group by customer_id;
```

Then press **Enter**

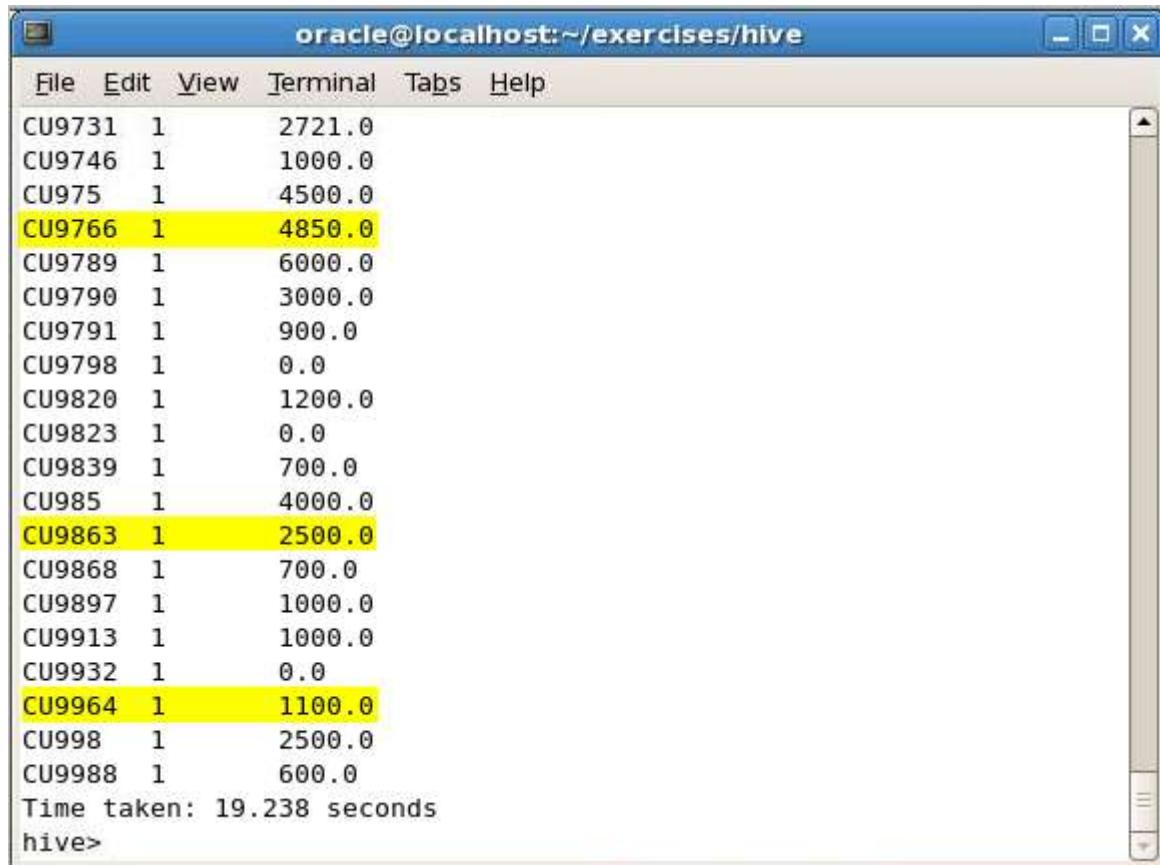


The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following Hive session:

```
File Edit View Terminal Tabs Help  
Loading data to table default.credit_department  
Table default.credit_department stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 14556, raw_data_size: 0]  
OK  
Time taken: 0.694 seconds  
hive> load data inpath '/user/oracle/export_mortgages_department' into table mortgages_department;  
Loading data to table default.mortgages_department  
Table default.mortgages_department stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 12281, raw_data_size: 0]  
OK  
Time taken: 0.421 seconds  
hive> select * from credit_department limit 5;  
OK  
CU7854 1500 0  
CU12993 2500 0  
CU608 1500 0  
CU10025 1500 0  
CU11680 1500 0  
Time taken: 0.359 seconds  
hive> select customer_id, count(*), sum(mortgage_sum) from mortgages_department group by customer_id;
```

The last command entered by the user is highlighted with a red rectangle.

The customer data is displayed in the terminal. We can see the customer_id, followed by the number of mortgages and the total amount for these mortgages.



A screenshot of a terminal window titled "oracle@localhost:~/exercises/hive". The window contains a list of customer data. The columns are customer_id, number of mortgages (labeled as 1), and total amount. Several rows are highlighted with yellow background: CU9766, CU9863, and CU9964. The terminal also shows the execution time and the hive prompt at the bottom.

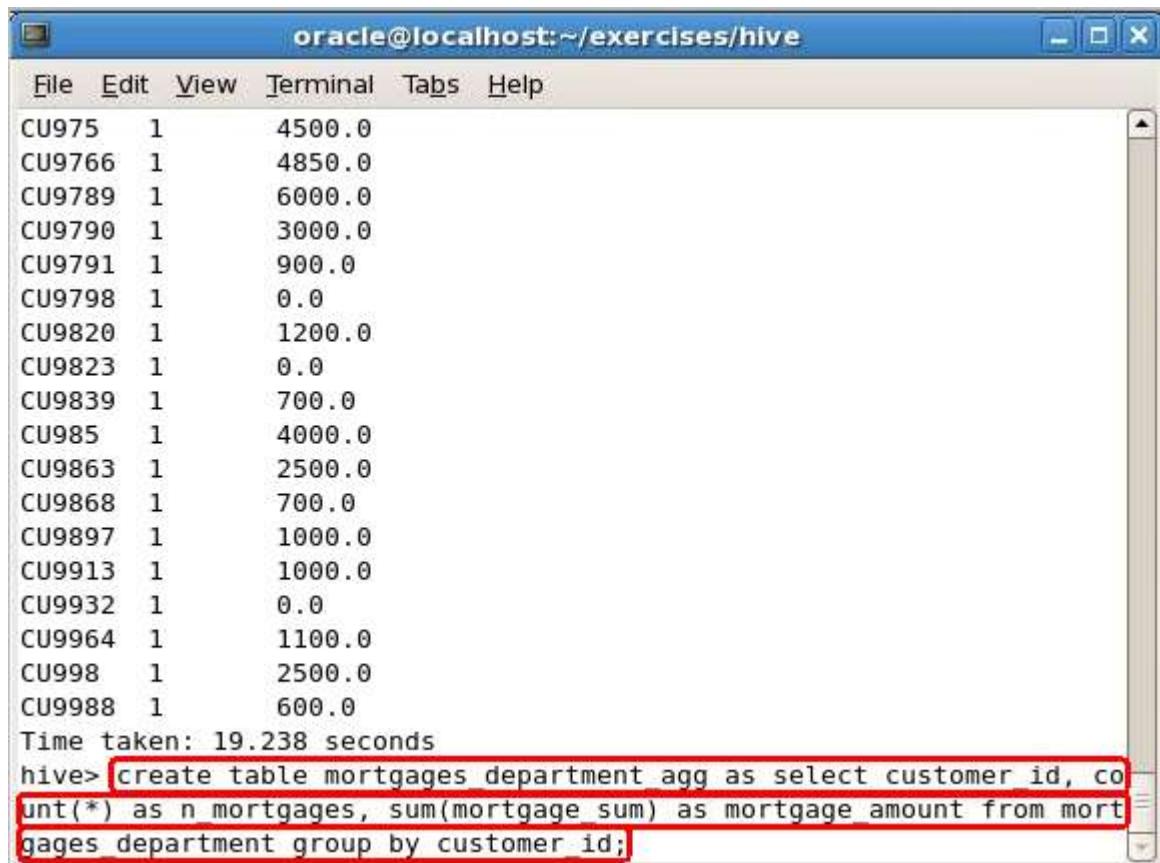
customer_id	Number of Mortgages	Total Amount
CU9731	1	2721.0
CU9746	1	1000.0
CU975	1	4500.0
CU9766	1	4850.0
CU9789	1	6000.0
CU9790	1	3000.0
CU9791	1	900.0
CU9798	1	0.0
CU9820	1	1200.0
CU9823	1	0.0
CU9839	1	700.0
CU985	1	4000.0
CU9863	1	2500.0
CU9868	1	700.0
CU9897	1	1000.0
CU9913	1	1000.0
CU9932	1	0.0
CU9964	1	1100.0
CU998	1	2500.0
CU9988	1	600.0

Time taken: 19.238 seconds
hive>

15. It is now useful to create a table with these results, using the SQL-like command “CREATE TABLE...AS SELECT”. Go to the terminal and type:

```
create table mortgages_department_agg as select customer_id, count(*) as n_mortgages, sum(mortgage_sum) as mortgage_amount from mortgages_department group by customer_id;
```

Then press **Enter**



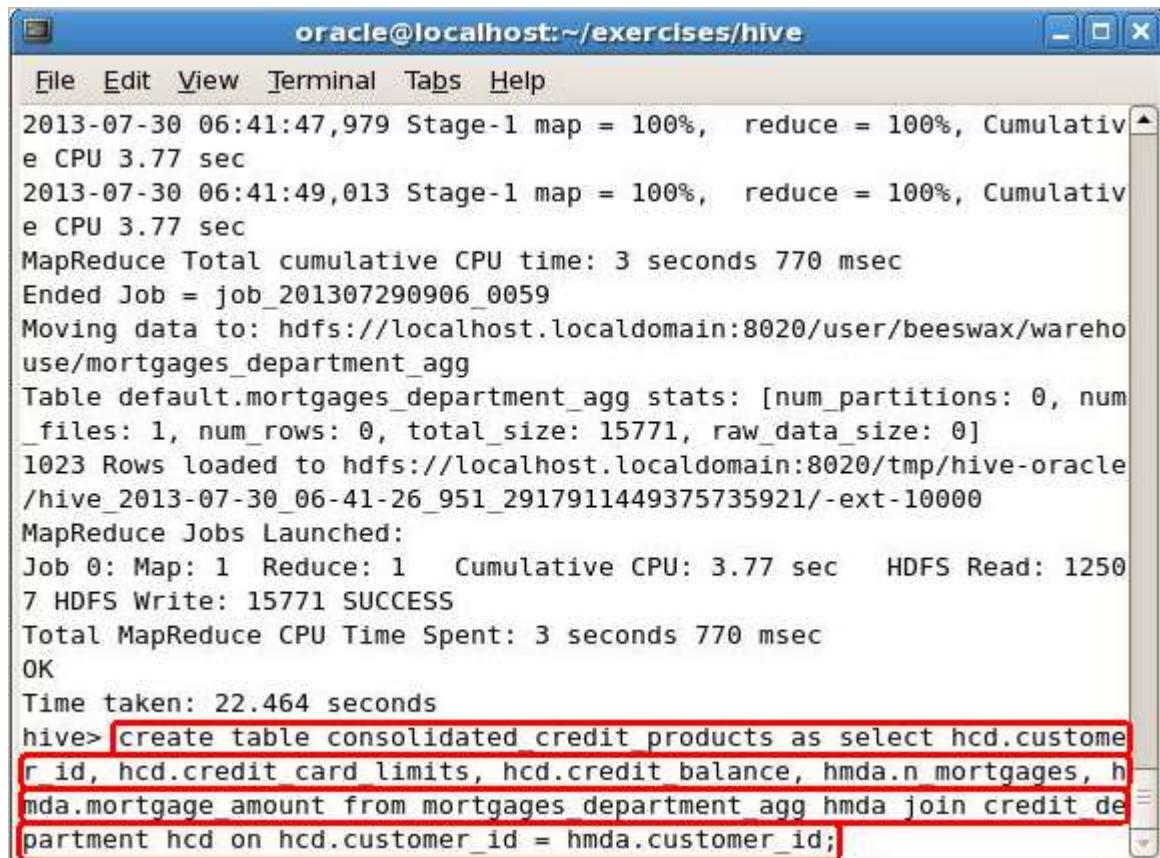
The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window has a menu bar with File, Edit, View, Terminal, Tabs, and Help. The main area displays a list of 24 rows of data, each consisting of a string identifier (e.g., CU975, CU9766, CU9789, etc.) followed by a single digit (e.g., 1, 1, 1, etc.) and a floating-point number (e.g., 4500.0, 4850.0, 6000.0, etc.). Below this data, the text "Time taken: 19.238 seconds" is displayed. At the bottom of the window, there is a red box highlighting the following SQL command:

```
hive> create table mortgages_department_agg as select customer_id, count(*) as n_mortgages, sum(mortgage_sum) as mortgage_amount from mortgages_department group by customer_id;
```

16. We are now ready to create our final table, called *consolidated_credit_products*, by joining the credits table with the aggregated mortgages table, again using a “CREATE TABLE ... AS SELECT” command. Go to the terminal and type:

```
create table consolidated_credit_products as select hcd.customer_id,
hcd.credit_card_limits, hcd.credit_balance, hmda.n_mortgages,
hmda.mortgage_amount from mortgages_department_agg hmda join
credit_department hcd on hcd.customer_id = hmda.customer_id;
```

Then press **Enter**



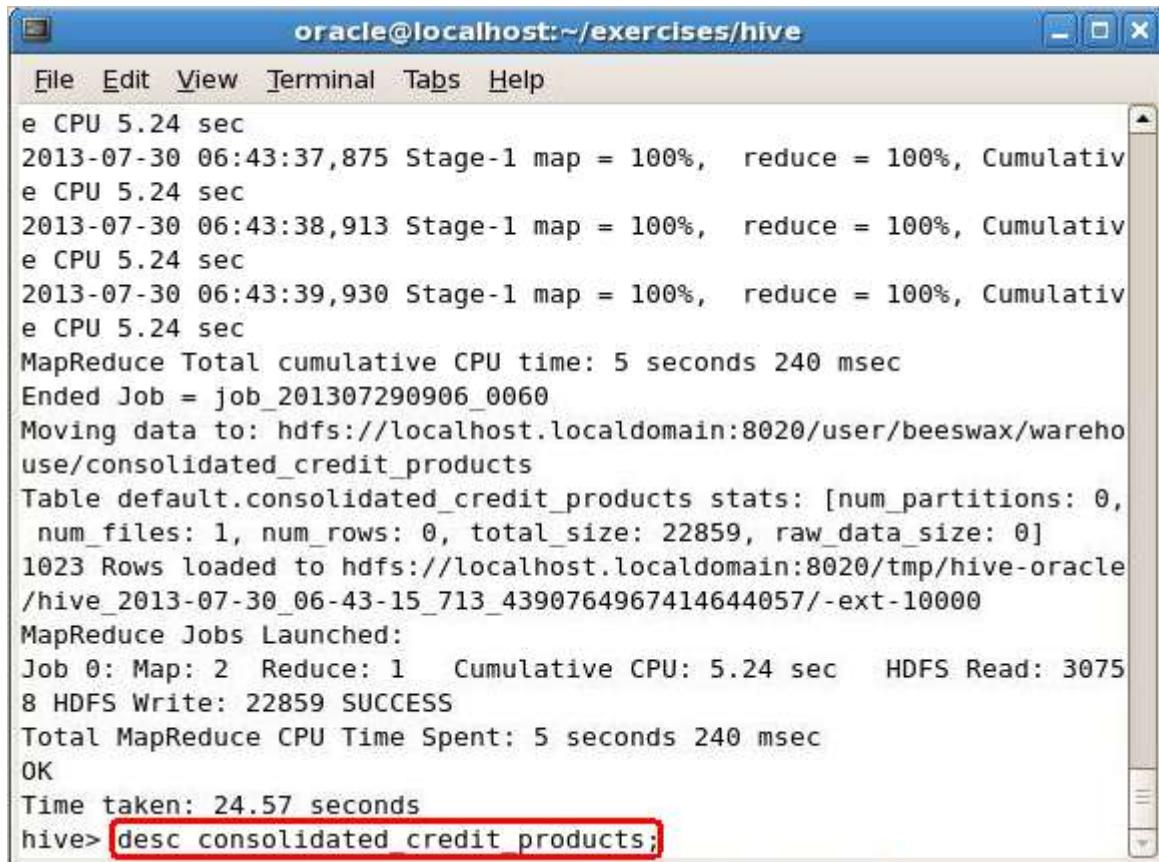
The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window displays the following text:

```
2013-07-30 06:41:47,979 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 3.77 sec
2013-07-30 06:41:49,013 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 3.77 sec
MapReduce Total cumulative CPU time: 3 seconds 770 msec
Ended Job = job_201307290906_0059
Moving data to: hdfs://localhost.localdomain:8020/user/beeswax/warehouse/mortgages_department_agg
Table default.mortgages_department_agg stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 15771, raw_data_size: 0]
1023 Rows loaded to hdfs://localhost.localdomain:8020/tmp/hive-oracle/hive_2013-07-30_06-41-26_951_2917911449375735921/-ext-10000
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Cumulative CPU: 3.77 sec HDFS Read: 1250
7 HDFS Write: 15771 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 770 msec
OK
Time taken: 22.464 seconds
hive> create table consolidated_credit_products as select hcd.customer_id, hcd.credit_card_limits, hcd.credit_balance, hmda.n_mortgages, hmda.mortgage_amount from mortgages_department_agg hmda join credit_department hcd on hcd.customer_id = hmda.customer_id;
```

17. As with normal SQL you also have a describe command available to see the columns in the table.
Go the terminal and type:

```
desc consolidated_credit_products;
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text output from a Hive session:

```
e CPU 5.24 sec
2013-07-30 06:43:37,875 Stage-1 map = 100%, reduce = 100%, Cumulative
e CPU 5.24 sec
2013-07-30 06:43:38,913 Stage-1 map = 100%, reduce = 100%, Cumulative
e CPU 5.24 sec
2013-07-30 06:43:39,930 Stage-1 map = 100%, reduce = 100%, Cumulative
e CPU 5.24 sec
MapReduce Total cumulative CPU time: 5 seconds 240 msec
Ended Job = job_201307290906_0060
Moving data to: hdfs://localhost.localdomain:8020/user/beeswax/warehouse/consolidated_credit_products
Table default.consolidated_credit_products stats: [num_partitions: 0,
  num_files: 1, num_rows: 0, total_size: 22859, raw_data_size: 0]
1023 Rows loaded to hdfs://localhost.localdomain:8020/tmp/hive-oracle
/hive_2013-07-30_06-43-15_713_4390764967414644057/-ext-10000
MapReduce Jobs Launched:
Job 0: Map: 2 Reduce: 1 Cumulative CPU: 5.24 sec HDFS Read: 3075
8 HDFS Write: 22859 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 240 msec
OK
Time taken: 24.57 seconds
hive> desc consolidated credit products;
```

The command "desc consolidated credit products;" is highlighted with a red box.

The structure of the table is displayed in the terminal, so that we can see the column names and the data type for each column.

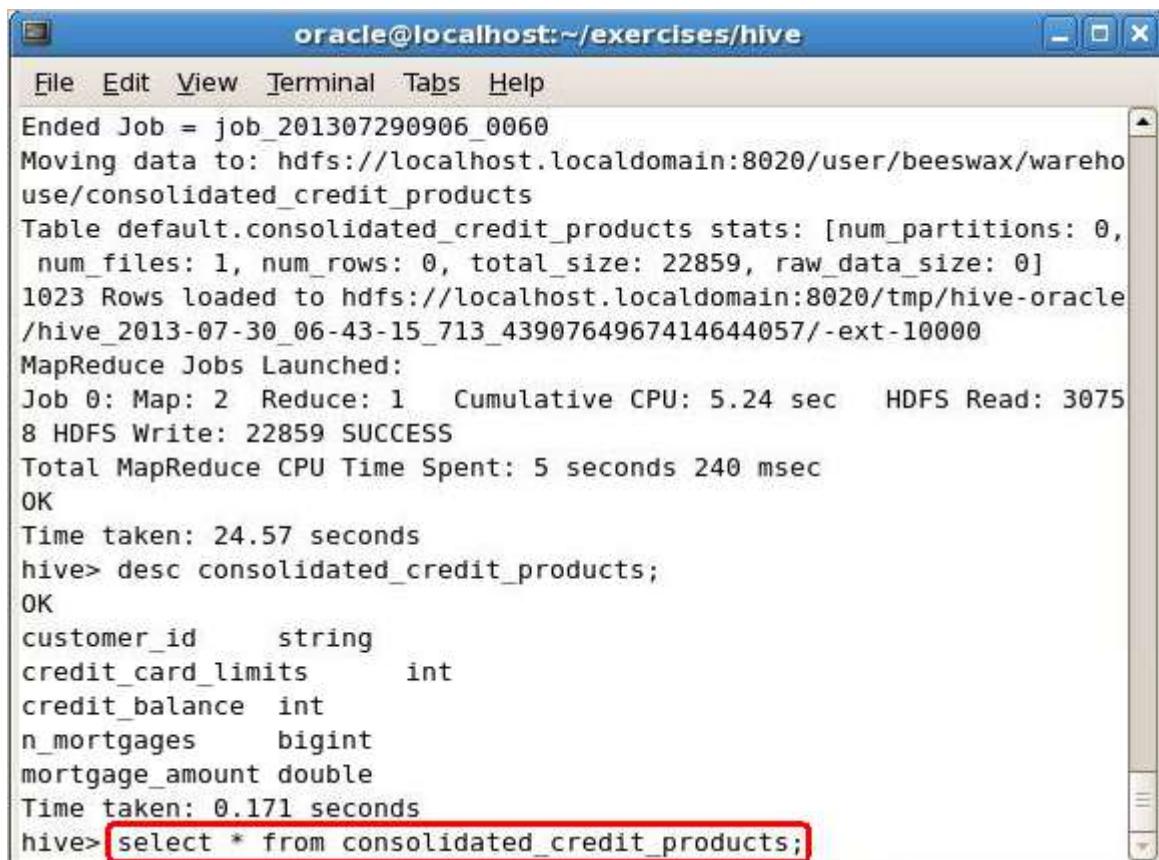
A screenshot of a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

```
Ended Job = job_201307290906_0060
Moving data to: hdfs://localhost.localdomain:8020/user/beeswax/warehouse/consolidated_credit_products
Table default.consolidated_credit_products stats: [num_partitions: 0,
  num_files: 1, num_rows: 0, total_size: 22859, raw_data_size: 0]
1023 Rows loaded to hdfs://localhost.localdomain:8020/tmp/hive-oracle
/hive_2013-07-30_06-43-15_713_4390764967414644057/-ext-10000
MapReduce Jobs Launched:
Job 0: Map: 2 Reduce: 1 Cumulative CPU: 5.24 sec HDFS Read: 3075
8 HDFS Write: 22859 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 240 msec
OK
Time taken: 24.57 seconds
hive> desc consolidated_credit_products;
OK
customer_id      string
credit_card_limits      int
credit_balance    int
n_mortgages      bigint
mortgage_amount double
Time taken: 0.171 seconds
hive>
```

18. We will now proceed to display the data from the newly created Hive table. Go to the terminal and type:

```
select * from consolidated_credit_products;
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window displays the output of a Hive query. The output includes information about a job completion, table statistics, and the results of a SELECT statement. The last line of the output, "hive> select * from consolidated_credit_products;", is highlighted with a red rectangle.

```
oracle@localhost:~/exercises/hive
File Edit View Terminal Tabs Help
Ended Job = job_201307290906_0060
Moving data to: hdfs://localhost.localdomain:8020/user/beeswax/warehouse/consolidated_credit_products
Table default.consolidated_credit_products stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 22859, raw_data_size: 0]
1023 Rows loaded to hdfs://localhost.localdomain:8020/tmp/hive-oracle/hive_2013-07-30_06-43-15_713_4390764967414644057/-ext-10000
MapReduce Jobs Launched:
Job 0: Map: 2 Reduce: 1 Cumulative CPU: 5.24 sec HDFS Read: 3075
8 HDFS Write: 22859 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 240 msec
OK
Time taken: 24.57 seconds
hive> desc consolidated_credit_products;
OK
customer_id      string
credit_card_limits      int
credit_balance      int
n_mortgages      bigint
mortgage_amount double
Time taken: 0.171 seconds
hive> select * from consolidated_credit_products;
```

The data is visible in Hive. The customer_id is followed by the credit limit and the credit balance. Then, the number of mortgages is displayed, followed by the total amount of mortgages.

<img alt="Screenshot of a terminal window showing Hive query results. The results are a list of tuples with columns: customer_id, credit_limit, n_mortgages, and total_mortgage_amount. Rows 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 5510, 5511, 5512, 5513, 5514, 5515, 5516, 5517, 5518, 5519, 5520, 5521, 5522, 5523, 5524, 5525, 5526, 5527, 5528, 5529, 55210, 55211, 55212, 55213, 55214, 55215, 55216, 55217, 55218, 55219, 55220, 55221, 55222, 55223, 55224, 55225, 55226, 55227, 55228, 55229, 552210, 552211, 552212, 552213, 552214, 552215, 552216, 552217, 552218, 552219, 552220, 552221, 552222, 552223, 552224, 552225, 552226, 552227, 552228, 552229, 5522210, 5522211, 5522212, 5522213, 5522214, 5522215, 5522216, 5522217, 5522218, 5522219, 5522220, 5522221, 5522222, 5522223, 5522224, 5522225, 5522226, 5522227, 5522228, 5522229, 55222210, 55222211, 55222212, 55222213, 55222214, 55222215, 55222216, 55222217, 55222218, 55222219, 55222220, 55222221, 55222222, 55222223, 55222224, 55222225, 55222226, 55222227, 55222228, 55222229, 552222210, 552222211, 552222212, 552222213, 552222214, 552222215, 552222216, 552222217, 552222218, 552222219, 552222220, 552222221, 552222222, 552222223, 552222224, 552222225, 552222226, 552222227, 552222228, 552222229, 5522222210, 5522222211, 5522222212, 5522222213, 5522222214, 5522222215, 5522222216, 5522222217, 5522222218, 5522222219, 5522222220, 5522222221, 5522222222, 5522222223, 5522222224, 5522222225, 5522222226, 5522222227, 5522222228, 5522222229, 55222222210, 55222222211, 55222222212, 55222222213, 55222222214, 55222222215, 55222222216, 55222222217, 55222222218, 55222222219, 55222222220, 55222222221, 55222222222, 55222222223, 55222222224, 55222222225, 55222222226, 55222222227, 55222222228, 55222222229, 552222222210, 552222222211, 552222222212, 552222222213, 552222222214, 552222222215, 552222222216, 552222222217, 552222222218, 552222222219, 552222222220, 552222222221, 552222222222, 552222222223, 552222222224, 552222222225, 552222222226, 552222222227, 552222222228, 552222222229, 5522222222210, 5522222222211, 5522222222212, 5522222222213, 5522222222214, 5522222222215, 5522222222216, 5522222222217, 5522222222218, 5522222222219, 5522222222220, 5522222222221, 5522222222222, 5522222222223, 5522222222224, 5522222222225, 5522222222226, 5522222222227, 5522222222228, 5522222222229, 55222222222210, 55222222222211, 55222222222212, 55222222222213, 55222222222214, 55222222222215, 55222222222216, 55222222222217, 55222222222218, 55222222222219, 55222222222220, 55222222222221, 55222222222222, 55222222222223, 55222222222224, 55222222222225, 55222222222226, 55222222222227, 55222222222228, 55222222222229, 552222222222210, 552222222222211, 552222222222212, 552222222222213, 552222222222214, 552222222222215, 552222222222216, 552222222222217, 552222222222218, 552222222222219, 552222222222220, 552222222222221, 552222222222222, 552222222222223, 552222222222224, 552222222222225, 552222222222226, 552222222222227, 552222222222228, 552222222222229, 5522222222222210, 5522222222222211, 5522222222222212, 5522222222222213, 5522222222222214, 5522222222222215, 5522222222222216, 5522222222222217, 5522222222222218, 5522222222222219, 5522222222222220, 5522222222222221, 5522222222222222, 5522222222222223, 5522222222222224, 5522222222222225, 5522222222222226, 5522222222222227, 5522222222222228, 5522222222222229, 55222222222222210, 55222222222222211, 55222222222222212, 55222222222222213, 55222222222222214, 55222222222222215, 55222222222222216, 55222222222222217, 55222222222222218, 55222222222222219, 55222222222222220, 55222222222222221, 55222222222222222, 55222222222222223, 55222222222222224, 55222222222222225, 55222222222222226, 55222222222222227, 55222222222222228, 55222222222222229, 552222222222222210, 552222222222222211, 552222222222222212, 552222222222222213, 552222222222222214, 552222222222222215, 552222222222222216, 552222222222222217, 552222222222222218, 552222222222222219, 552222222222222220, 552222222222222221, 552222222222222222, 552222222222222223, 552222222222222224, 552222222222222225, 552222222222222226, 552222222222222227, 552222222222222228, 552222222222222229, 5522222222222222210, 5522222222222222211, 5522222222222222212, 5522222222222222213, 5522222222222222214, 5522222222222222215, 5522222222222222216, 5522222222222222217, 5522222222222222218, 5522222222222222219, 5522222222222222220, 5522222222222222221, 5522222222222222222, 5522222222222222223, 5522222222222222224, 5522222222222222225, 5522222222222222226, 5522222222222222227, 5522222222222222228, 5522222222222222229, 55222222222222222210, 55222222222222222211, 55222222222222222212, 55222222222222222213, 55222222222222222214, 55222222222222222215, 55222222222222222216, 55222222222222222217, 55222222222222222218, 55222222222222222219, 55222222222222222220, 55222222222222222221, 55222222222222222222, 55222222222222222223, 55222222222222222224, 55222222222222222225, 55222222222222222226, 55222222222222222227, 55222222222222222228, 55222222222222222229, 552222222222222222210, 552222222222222222211, 552222222222222222212, 552222222222222222213, 552222222222222222214, 552222222222222222215, 552222222222222222216, 552222222222222222217, 552222222222222222218, 552222222222222222219, 552222222222222222220, 552222222222222222221, 552222222222222222222, 552222222222222222223, 552222222222222222224, 552222222222222222225, 552222222222222222226, 552222222222222222227, 552222222222222222228, 552222222222222222229, 5522222222222222222210, 5522222222222222222211, 5522222222222222222212, 5522222222222222222213, 5522222222222222222214, 5522222222222222222215, 5522222222222222222216, 5522222222222222222217, 5522222222222222222218, 5522222222222222222219, 5522222222222222222220, 5522222222222222222221, 5522222222222222222222, 5522222222222222222223, 5522222222222222222224, 5522222222222222222225, 5522222222222222222226, 5522222222222222222227, 5522222222222222222228, 5522222222222222222229, 55222222222222222222210, 55222222222222222222211, 55222222222222222222212, 55222222222222222222213, 55222222222222222222214, 55222222222222222222215, 55222222222222222222216, 55222222222222222222217, 55222222222222222222218, 55222222222222222222219, 55222222222222222222220, 55222222222222222222221, 55222222222222222222222, 55222222222222222222223, 55222222222222222222224, 55222222222222222222225, 55222222222222222222226, 55222222222222222222227, 55222222222222222222228, 55222222222222222222229, 552222222222222222222210, 552222222222222222222211, 552222222222222222222212, 552222222222222222222213, 552222222222222222222214, 552222222222222222222215, 552222222222222222222216, 552222222222222222222217, 552222222222222222222218, 552222222222222222222219, 552222222222222222222220, 552222222222222222222221, 552222222222222222222222, 552222222222222222222223, 552222222222222222222224, 552222222222222222222225, 552222222222222222222226, 552222222222222222222227, 552222222222222222222228, 552222222222222222222229, 5522222222222222222222210, 5522222222222222222222211, 5522222222222222222222212, 5522222222222222222222213, 5522222222222222222222214, 5522222222222222222222215, 5522222222222222222222216, 5522222222222222222222217, 5522222222222222222222218, 5522222222222222222222219, 5522222222222222222222220, 5522222222222222222222221, 5522222222222222222222222, 5522222222222222222222223, 5522222222222222222222224, 5522222222222222222222225, 5522222222222222222222226, 5522222222222222222222227, 5522222222222222222222228, 5522222222222222222222229, 55222222222222222222222210, 55222222222222222222222211, 55222222222222222222222212, 55222222222222222222222213, 55222222222222222222222214, 55222222222222222222222215, 55222222222222222222222216, 55222222222222222222222217, 55222222222222222222222218, 55222222222222222222222219, 55222222222222222222222220, 55222222222222222222222221, 55222222222222222222222222, 55222222222222222222222223, 55222222222222222222222224, 55222222222222222222222225, 55222222222222222222222226, 55222222222222222222222227, 55222222222222222222222228, 55222222222222222222222229, 552222222222222222222222210, 552222222222222222222222211, 552222222222222222222222212, 552222222222222222222222213, 552222222222222222222222214, 552222222222222222222222215, 552222222222222222222222216, 552222222222222222222222217, 552222222222222222222222218, 552222222222222222222222219, 552222222222222222222222220, 552222222222222222222222221, 552222222222222222222222222, 552222222222222222222222223, 552222222222222222222222224, 552222222222222222222222225, 552222222222222222222222226,

20. Same as the Hive table the describe command is available to see the columns in the view. Go the terminal and type:

```
desc v_consolidated_credit_products;
```

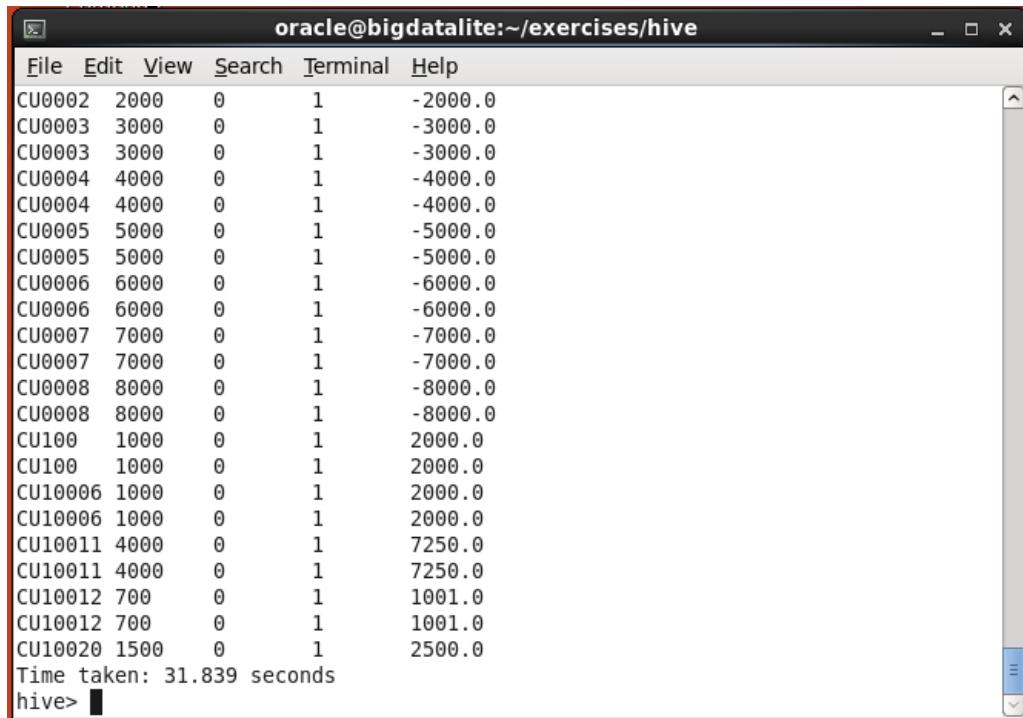


A screenshot of a terminal window titled "oracle@bigdatalite:~/exercises/hive". The window shows a series of greater-than signs (>) followed by the command "desc v_consolidated_credit_products;". The output shows the schema of the view, including columns for customer_id (string), credit_card_limits (int), credit_balance (int), n_mortgages (bigint), and mortgage_amount (double). The command was run in 0.227 seconds. The terminal prompt "hive>" is visible at the bottom.

```
>
>
>
>
>
>
>
>
>
>
>
> desc v_consolidated_credit_products;
OK
customer_id      string
credit_card_limits    int
credit_balance    int
n_mortgages      bigint
mortgage_amount  double
Time taken: 0.227 seconds
hive> ■
```

21. Execute the query :

```
select * from v_consolidated_credit_products;
```



A screenshot of a terminal window titled "oracle@bigdatalite:~/exercises/hive". The window displays the results of a SELECT query on the v_consolidated_credit_products view. The output consists of multiple rows of data, each containing five columns: customer_id, credit_card_limits, credit_balance, n_mortgages, and mortgage_amount. The data includes various values such as CU0002, CU0003, CU0004, CU0005, CU0006, CU0007, CU0008, CU100, CU1000, CU10006, CU10007, CU10011, CU10012, and CU10020, with corresponding numerical values for the other columns. The command was run in 31.839 seconds. The terminal prompt "hive>" is visible at the bottom.

```
CU0002 2000 0 1 -2000.0
CU0003 3000 0 1 -3000.0
CU0003 3000 0 1 -3000.0
CU0004 4000 0 1 -4000.0
CU0004 4000 0 1 -4000.0
CU0005 5000 0 1 -5000.0
CU0005 5000 0 1 -5000.0
CU0006 6000 0 1 -6000.0
CU0006 6000 0 1 -6000.0
CU0007 7000 0 1 -7000.0
CU0007 7000 0 1 -7000.0
CU0008 8000 0 1 -8000.0
CU0008 8000 0 1 -8000.0
CU100 1000 0 1 2000.0
CU100 1000 0 1 2000.0
CU10006 1000 0 1 2000.0
CU10006 1000 0 1 2000.0
CU10011 4000 0 1 7250.0
CU10011 4000 0 1 7250.0
CU10012 700 0 1 1001.0
CU10012 700 0 1 1001.0
CU10020 1500 0 1 2500.0
Time taken: 31.839 seconds
hive> ■
```

21. This exercise is now finished, so we will first exit Hive.

Go to the terminal and type:

```
exit;
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

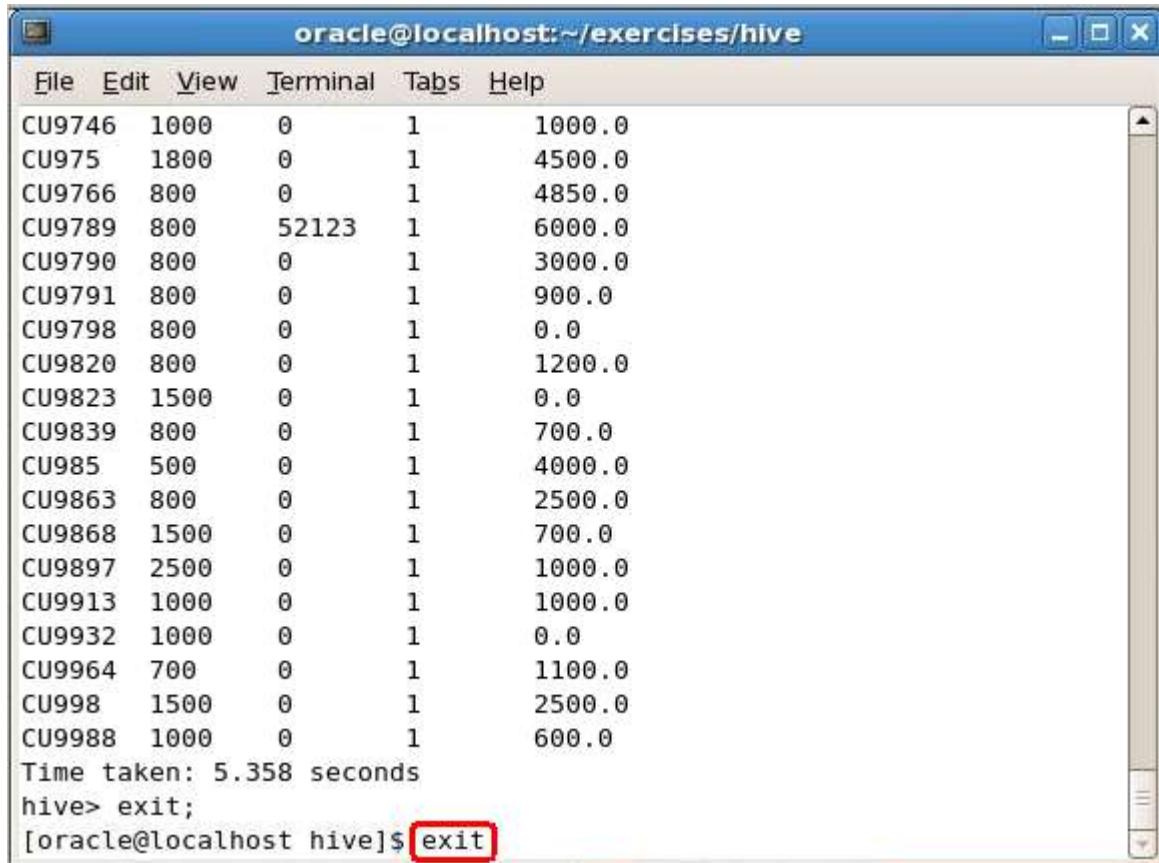
CU9731	1000	0	1	2721.0
CU9746	1000	0	1	1000.0
CU975	1800	0	1	4500.0
CU9766	800	0	1	4850.0
CU9789	800	52123	1	6000.0
CU9790	800	0	1	3000.0
CU9791	800	0	1	900.0
CU9798	800	0	1	0.0
CU9820	800	0	1	1200.0
CU9823	1500	0	1	0.0
CU9839	800	0	1	700.0
CU985	500	0	1	4000.0
CU9863	800	0	1	2500.0
CU9868	1500	0	1	700.0
CU9897	2500	0	1	1000.0
CU9913	1000	0	1	1000.0
CU9932	1000	0	1	0.0
CU9964	700	0	1	1100.0
CU998	1500	0	1	2500.0
CU9988	1000	0	1	600.0

Time taken: 5.358 seconds
hive> **exit;**

22. We can now close the terminal. Go to the terminal and type:

```
exit
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/hive". The window contains the following text:

```
File Edit View Terminal Tabs Help
CU9746 1000 0 1 1000.0
CU975 1800 0 1 4500.0
CU9766 800 0 1 4850.0
CU9789 800 52123 1 6000.0
CU9790 800 0 1 3000.0
CU9791 800 0 1 900.0
CU9798 800 0 1 0.0
CU9820 800 0 1 1200.0
CU9823 1500 0 1 0.0
CU9839 800 0 1 700.0
CU985 500 0 1 4000.0
CU9863 800 0 1 2500.0
CU9868 1500 0 1 700.0
CU9897 2500 0 1 1000.0
CU9913 1000 0 1 1000.0
CU9932 1000 0 1 0.0
CU9964 700 0 1 1100.0
CU998 1500 0 1 2500.0
CU9988 1000 0 1 600.0
Time taken: 5.358 seconds
hive> exit;
[oracle@localhost hive]$ exit
```

6.4 Hive Summary

In this exercise you were introduced to the Hive Query Language. You saw how to create and view tables and Hive views using HiveQL. Once tables were created you were introduced to some standard SQL constructs which HiveQL has available. It is important to understand that Hive is an abstraction layer for Hadoop and Map/Reduce jobs. All queries written in HiveQL get transformed into a DAG (Directed Acyclic Graph) of Map/Reduce tasks which are then run on the Hadoop cluster, hence taking advantage of all performance, scalability capabilities, but also maintain all of the limitations of Hadoop.

HiveQL has most of the functionality available with standard SQL, including a series of DDL and DML functions, but at the same time it does not strictly adhere to the SQL-92 standard. HiveQL offers extensions to SQL, including multitable inserts and "CREATE TABLE... AS SELECT", but only offers basic support for indexing. Also, HiveQL lacks support for transactions and materialized views, and only limited subquery support. It is intended for long running queries of a Data Warehousing type rather than a transactional OLTP type of data load.

6.5 Impala Concepts and Architecture

Components of the Impala Server

The Impala server is a distributed, massively parallel processing (MPP) database engine. It consists of different daemon processes that run on specific hosts within your CDH cluster.

The Impala Daemon

The core Impala component is a daemon process that runs on each node of the cluster, physically represented by the impalad process. It reads and writes to data files; accepts queries transmitted from the impala-shell command, Hue, JDBC, or ODBC; parallelizes the queries and distributes work to other nodes in the Impala cluster; and transmits intermediate query results back to the central coordinator node.

You can submit a query to the Impala daemon running on any node, and that node serves as the coordinator node for that query. The other nodes transmit partial results back to the coordinator, which constructs the final result set for a query. When running experiments with functionality through the impala-shell command, you might always connect to the same Impala daemon for convenience. For clusters running production workloads, you might load-balance between the nodes by submitting each query to a different Impala daemon in round-robin style, using the JDBC or ODBC interfaces.

The Impala daemons are in constant communication with the statestore, to confirm which nodes are healthy and can accept new work.

They also receive broadcast messages from the catalogd daemon (introduced in Impala 1.2) whenever any Impala node in the cluster creates, alters, or drops any type of object, or when an INSERT or LOAD DATA statement is processed through Impala.

The Impala Statestore

The Impala component known as the statestore checks on the health of Impala daemons on all the nodes in a cluster, and continuously relays its findings to each of those daemons. It is physically represented by a daemon process named statetstored; you only need such a process on one node in the cluster. If an Impala node goes offline due to hardware failure, network error, software issue, or other reason, the statestore informs all the other nodes so that future queries can avoid making requests to the unreachable node.

Because the statestore's purpose is to help when things go wrong, it is not critical to the normal operation of an Impala cluster. If the statestore is not running or becomes unreachable, the other nodes continue running and distributing work among themselves as usual; the cluster just becomes less robust if other nodes fail while the statestore is offline. When the statestore comes back online, it re-establishes communication with the other nodes and resumes its monitoring function.

The Impala Catalog Service

The Impala component known as the catalog service relays the metadata changes from Impala SQL statements to all the nodes in a cluster. It is physically represented by a daemon process named catalogd; you only need such a process on one node in the cluster. Because the requests are passed through the statestore daemon, it makes sense to run the statetstored and catalogd services on the same node.

Programming Impala Applications

The core development language with Impala is SQL. You can also use Java or other languages to interact with Impala through the standard JDBC and ODBC interfaces used by many business intelligence tools.

For specialized kinds of analysis, you can supplement the SQL built-in functions by writing user-defined functions (UDFs) in C++ or Java.

Overview of the Impala SQL Dialect

The Impala SQL dialect is descended from the SQL syntax used in the Apache Hive component (HiveQL). As such, it is familiar to users who are already familiar with running SQL queries on the Hadoop infrastructure. Currently, Impala SQL supports a subset of HiveQL statements, data types, and built-in functions.

For users coming to Impala from traditional database backgrounds, the following aspects of the SQL dialect might seem familiar or unusual:

Impala SQL is focused on queries and includes relatively little DML. There is no UPDATE or DELETE statement. Stale data is typically discarded (by DROP TABLE or ALTER TABLE ... DROP PARTITION statements) or replaced (by INSERT OVERWRITE statements).

All data loading is done by INSERT statements, which typically insert data in bulk by querying from other tables. There are two variations, INSERT INTO which appends to the existing data, and INSERT OVERWRITE which replaces the entire contents of a table or partition (similar to TRUNCATE TABLE followed by a new INSERT). There is no INSERT ... VALUES syntax to insert a single row.

You often construct Impala table definitions and data files in some other environment, and then attach Impala so that it can run real-time queries. The same data files and table metadata are shared with other components of the Hadoop ecosystem.

Because Hadoop and Impala are focused on data warehouse-style operations on large data sets, Impala SQL includes some idioms that you might find in the import utilities for traditional database systems. For example, you can create a table that reads comma-separated or tab-separated text files, specifying the separator in the CREATE TABLE statement. You can create external tables that read existing data files but do not move or transform them.

Because Impala reads large quantities of data that might not be perfectly tidy and predictable, it does not impose length constraints on string data types. For example, you define a database column as STRING rather than CHAR(1) or VARCHAR(64).

For query-intensive applications, you will find familiar notions such as joins, built-in functions for processing strings, numbers, and dates, aggregate functions, subqueries, and comparison operators such as IN() and BETWEEN.

Overview of Impala Programming Interfaces

You can connect and submit requests to the Impala daemons through:

- The impala-shell interactive command interpreter.
- The Apache Hue web-based user interface.
- JDBC and ODBC.

With these options, you can use Impala in heterogeneous environments, with JDBC or ODBC applications running on non-Linux platforms. You can also use Impala in combination with various Business Intelligence tools that use the JDBC and ODBC interfaces.

Each impalad daemon process, running on separate nodes in a cluster, listens to several ports for incoming requests. Requests from impala-shell and Hue are routed to the impalad daemons through the same port. The impalad daemons listen on separate ports for JDBC and ODBC requests.

How Impala Fits Into the Hadoop Ecosystem

Impala makes use of many familiar components within the Hadoop ecosystem. Impala can interchange data with other Hadoop components, as both a consumer and a producer, so it can fit in flexible ways into your ETL and ELT pipelines.

How Impala Works with Hive

A major Impala goal is to make SQL-on-Hadoop operations fast and efficient enough to appeal to new categories of users and open up Hadoop to new types of use cases. Where practical, it makes use of existing Apache Hive infrastructure that many Hadoop users already have in place to perform long-running, batch-oriented SQL queries.

In particular, Impala keeps its table definitions in a traditional MySQL or PostgreSQL database known as the metastore, the same database where Hive keeps this type of data. Thus, Impala can access tables defined or loaded by Hive, as long as all columns use Impala-supported data types, file formats, and compression codecs.

The initial focus on query features and performance means that Impala can read more types of data with the SELECT statement than it can write with the INSERT statement. To query data using the Avro, RCFile, or SequenceFile file formats, you load the data using Hive.

The Impala query optimizer can also make use of table statistics and column statistics. Originally, you gathered this information with the ANALYZE TABLE statement in Hive; in Impala 1.2.2 and higher, use the Impala COMPUTE STATS statement instead. COMPUTE STATS requires less setup, is more reliable and faster, and does not require switching back and forth between impala-shell and the Hive shell.

Overview of Impala Metadata and the Metastore

As discussed in How Impala Works with Hive, Impala maintains information about table definitions in a central database known as the metastore. Impala also tracks other metadata for the low-level characteristics of data files.

For tables with a large volume of data and/or many partitions, retrieving all the metadata for a table can be time-consuming, taking minutes in some cases. Thus, each Impala node caches all of this metadata to reuse for future queries against the same table.

If the table definition or the data in the table is updated, all other Impala daemons in the cluster must receive the latest metadata, replacing the obsolete cached metadata, before issuing a query against that table. The metadata update is automatic, coordinated through the catalogd daemon, for all DDL and DML statements issued through Impala.

For DDL and DML issued through Hive, or changes made manually to files in HDFS, you still use the REFRESH statement (when new data files are added to existing tables) or the INVALIDATE METADATA statement (for entirely new tables, or after dropping a table, performing an HDFS rebalance operation, or deleting data files). Issuing INVALIDATE METADATA by itself retrieves metadata for all the tables tracked by the metastore. If you know that only specific tables have been changed outside of Impala, you can issue REFRESH table_name for each affected table to only retrieve the latest metadata for those tables.

How Impala Uses HDFS

Impala uses the distributed filesystem HDFS as its primary data storage medium. Impala relies on the redundancy provided by HDFS to guard against hardware or network outages on individual nodes. Impala table data is physically represented as data files in HDFS, using familiar HDFS file formats and compression codecs. When data files are present in the directory for a new table, Impala reads them all, regardless of file name. New data is added in files with names controlled by Impala.

How Impala Uses HBase

HBase is an alternative to HDFS as a storage medium for Impala data. It is a database storage system built on top of HDFS, without built-in SQL support. Many Hadoop users already have it configured and store large (often sparse) data sets in it. By defining tables in Impala and mapping them to equivalent tables in HBase, you can query the contents of the HBase tables through Impala, and even perform join queries including both Impala and HBase tables.

6.6 Impala Exercises

The Exercises for Impala are extending the Hive exercises by first using the Hive tables created in exercise 6 and then recreating the exercise 6 tables in Impala to demonstrate the difference in performance and capabilities. CD into the `/home/oracle/exercises/impala` directory to begin these exercises.

Accessing Hive tables in Impala :

1. The first task we will need to accomplish is to make sure that Impala syncs its metadata from the metastore. Run `impala-shell`, connect to the Impala server “`connect bigdatalite.localdomain:21000`” and execute “`show tables`”. If you do not see the hive tables created, you would need to run the command “`invalidate metadata`” to clear and reload the local metadata, quit `impala-shell` and restart the `impala-shell` at the command line and run the command “`show tables`”

```
[oracle@bigdatalite setup]$ impala-shell
Starting Impala Shell without Kerberos authentication
Connected to bigdatalite.localdomain:21000
Server version: impalad version cdh5-1.3.0 RELEASE (build
40e1b62cf0b97f666d084d9509bf9639c575068c)
Welcome to the Impala shell. Press TAB twice to see a list of available commands.

Copyright (c) 2012 Cloudera, Inc. All rights reserved.

(Shell build version: Impala Shell vcdh5-1.3.0 (40e1b62) built on Tue Mar 25
13:46:44 PDT 2014)
[bigdatalite.localdomain:21000] > connect bigdatalite.localdomain:21000;
Connected to bigdatalite.localdomain:21000
Server version: impalad version cdh5-1.3.0 RELEASE (build
40e1b62cf0b97f666d084d9509bf9639c575068c)
[bigdatalite.localdomain:21000] > show tables;
Query: show tables
+-----+
| name      |
+-----+
```

```

| bookstore |
| sample_07 |
| sample_08 |
+-----+
Returned 3 row(s) in 0.03s
[bigdatalite.localdomain:21000] > invalidate metadata;
Query: invalidate metadata

Returned 0 row(s) in 6.82s
[bigdatalite.localdomain:21000] > quit;
Goodbye
[oracle@bigdatalite impala]$ impala-shell
Starting Impala Shell without Kerberos authentication
Connected to bigdatalite.localdomain:21000
Server version: impalad version 1.2.3 RELEASE (build
1cab04cdb88968a963a8ad6121a2e72a3a623eca)
Welcome to the Impala shell. Press TAB twice to see a list of available commands.

Copyright (c) 2012 Cloudera, Inc. All rights reserved.

(Shell build version: Impala Shell v1.2.3 (1cab04c) built on Fri Dec 20 19:39:39
PST 2013)
[bigdatalite.localdomain:21000] > show tables;
Query: show tables
+-----+
| name           |
+-----+
| bookstore      |
| consolidated_credit_products |
| credit_department |
| mortgages_department |
| mortgages_department_agg |
| sample_07       |
| sample_08       |
+-----+
Returned 7 row(s) in 0.02s
[bigdatalite.localdomain:21000] >
```

2. Run the 2 queries from exercise 6:

a. select * from credit_department limit 5;

```

[bigdatalite.localdomain:21000] > select * from credit_department
limit 5;
Query: select * from credit_department limit 5
+-----+-----+-----+
| customer_id | credit_card_limits | credit_balance |
+-----+-----+-----+
```

```

| CU7854      | 1500          | 0          |
| CU12993     | 2500          | 0          |
| CU608       | 1500          | 0          |
| CU10025     | 1500          | 0          |
| CU11680     | 1500          | 0          |
+-----+-----+-----+
Returned 5 row(s) in 0.38s
[bigdatalite.localdomain:21000] >

```

```

b. select customer_id, count(*), sum(mortgage_sum) from
mortgages_department group by customer_id;

[bigdatalite.localdomain:21000] > select customer_id, count(*),
sum(mortgage_sum) from mortgages_department group by customer_id
limit 25;
Query: select customer_id, count(*), sum(mortgage_sum) from
mortgages_department group by customer_id limit 25
+-----+-----+-----+
| customer_id | count(*) | sum(mortgage_sum) |
+-----+-----+-----+
| CU15049     | 1         | 4000        |
| CU3835      | 1         | 800         |
| CU5859      | 1         | 200         |
| CU7949      | 1         | 0           |
| CU3592      | 1         | 5000        |
| CU6227      | 1         | 4528        |
| CU0002      | 1         | -2000       |
| CU6466      | 2         | 3250        |
| CU3434      | 2         | 5000        |
| CU2620      | 1         | 1590        |
| CU4877      | 1         | 1000        |
| CU790       | 1         | 0           |
| CU14912     | 1         | 250         |
| CU12283     | 1         | 1055        |
| CU14991     | 1         | 300         |
| CU12577     | 1         | 300         |
| CU11032     | 1         | 0           |
. .
| CU3039      | 1         | 6000        |
| CU11006     | 1         | 2500        |
| CU10180     | 1         | 176         |
| CU14493     | 1         | 0           |
| CU4663      | 1         | 2200        |
| CU6293      | 1         | 0           |
| CU1829      | 1         | 2800        |
+-----+-----+-----+
Returned 1023 row(s) in 1.07s
[bigdatalite.localdomain:21000] >

```

Observe the difference in query speed in the second query and also notice no MapReduce was run.

Redo Exercise 6 (Hive) with Impala :

1. We move the export_* files into our /user/oracle directory on HDFS. Because Impala unlike Hive has a running distributed kernel (impalad) owned by the impala user, we have to make change our permissions to allow the impala user to move, not copy, the export_* files into the /hive/warehouse/*.

```

hadoop fs -put export_*
hadoop fs -chmod 777 /user/oracle
hadoop fs -chmod 666 export_*

```

2. Run the impala-shell command to create and load our initial tables :

```

create table imp_credit_department (CUSTOMER_ID string,CREDIT_CARD_LIMITS int,
CREDIT_BALANCE int)
ROW FORMAT delimited
fields terminated by ',';

create table imp_mortgages_department (CUSTOMER_ID string,MORTGAGE_SUM float)
ROW FORMAT delimited
fields terminated by ',';

load data inpath '/user/oracle/export_credit_department.csv' into table
imp_credit_department;

load data inpath '/user/oracle/export_mortgages_department.csv' into table
imp_mortgages_department;

```

3. Now that our tables are loaded we can now open a new terminal window and execute the following to set our permissions back to original.

```

hadoop fs -chmod 755 /user/oracle

```

4. Lets test our new tables : .

```

select * from imp_credit_department limit 5;

```

```

[bigdatalite.localdomain:21000] > select * from imp_credit_department limit 5;
Query: select * from imp_credit_department limit 5
+-----+-----+-----+
| customer_id | credit_card_limits | credit_balance |
+-----+-----+-----+
| CU7854     | 1500           | 0             |
| CU12993    | 2500           | 0             |
| CU608      | 1500           | 0             |
| CU10025    | 1500           | 0             |
| CU11680    | 1500           | 0             |
+-----+-----+-----+
Returned 5 row(s) in 0.48s
[bigdatalite.localdomain:21000] >

```

```

select customer_id, count(*), sum(mortgage_sum) from imp_mortgages_department group by
customer_id limit 25;

```

```

[bigdatalite.localdomain:21000] > select customer_id, count(*), sum(mortgage_sum)
from imp_mortgages_department group by customer_id limit 25;
Query: select customer_id, count(*), sum(mortgage_sum) from
imp_mortgages_department group by customer_id limit 25
+-----+-----+-----+
| customer_id | count(*) | sum(mortgage_sum) |
+-----+-----+-----+

```

CU15049 1 4000			
CU3835 1 800			
CU5859 1 200			
CU7949 1 0			
CU3592 1 5000			
CU6227 1 4528			
CU0002 1 -2000			
CU6466 2 3250			
CU3434 2 5000			
CU2620 1 1590			
CU4877 1 1000			
CU790 1 0			
CU14912 1 250			
CU12283 1 1055			
CU14991 1 300			
CU12577 1 300			
CU11032 1 0			
CU3715 1 1216			
CU14309 1 1200			
CU8122 1 3000			
CU4983 1 1600			
CU135 1 3000			
CU878 1 350			
CU13543 2 2728			
CU2943 1 0			
+-----+-----+-----+			
Returned 25 row(s) in 0.72s			
[bigdatalite.localdomain:21000] >			

Finally lets create a table and a view from our previous tables : _

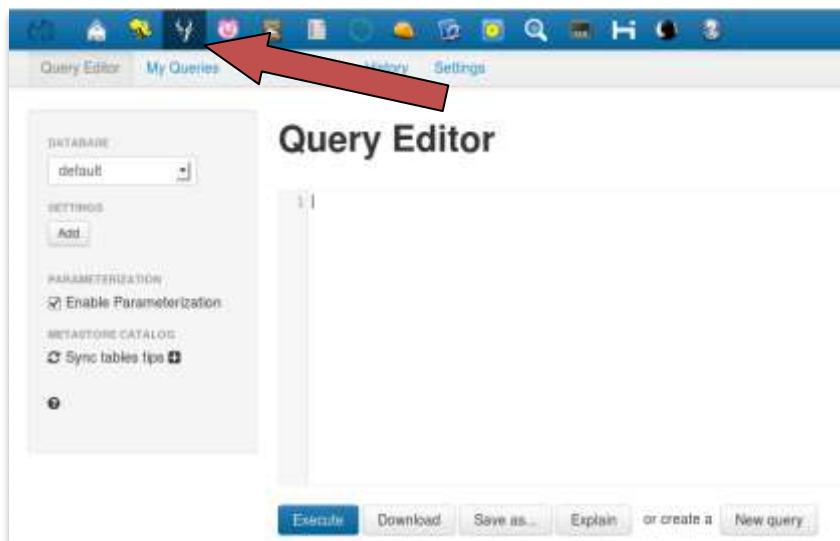
```
create table imp_mortgages_department_agg as select customer_id, count(*) as n_mortgages,
sum(mortgage_sum) as mortgage_amount from imp_mortgages_department group by customer_id;

create view v_imp_consolidated_credit_products as select hcd.customer_id,
hcd.credit_card_limits, hcd.credit_balance, hmda.n_mortgages, hmda.mortgage_amount from
imp_mortgages_department_agg hmda join credit_department hcd on hcd.customer_id =
hmda.customer_id;
```

```
select * from v_imp_consolidated_credit_products limit 25;
```

```
[bigdatalite.localdomain:21000] > select * from v_imp_consolidated_credit_products
limit 25;
Query: select * from v_imp_consolidated_credit_products limit 25
+-----+-----+-----+-----+-----+
| customer_id | credit_card_limits | credit_balance | n_mortgages | mortgage_amount |
+-----+-----+-----+-----+-----+
| CU15049 | 1000 | 0 | 1 | 4000 |
| CU3835 | 900 | 0 | 1 | 800 |
| CU5859 | 900 | 0 | 1 | 200 |
| CU7949 | 700 | 0 | 1 | 0 |
| CU3592 | 1400 | 0 | 1 | 5000 |
| CU6227 | 800 | 0 | 1 | 4528 |
| CU0002 | 2000 | 0 | 1 | -2000 |
| CU6466 | 1500 | 0 | 2 | 3250 |
| CU3434 | 3000 | 0 | 2 | 5000 |
| CU2620 | 1500 | 0 | 1 | 1590 |
| CU4877 | 900 | 0 | 1 | 1000 |
| CU790 | 900 | 0 | 1 | 0 |
| CU14912 | 1000 | 0 | 1 | 250 |
| CU12283 | 2500 | 0 | 1 | 1055 |
| CU14991 | 1000 | 0 | 1 | 300 |
| CU12577 | 1000 | 0 | 1 | 300 |
| CU11032 | 500 | 0 | 1 | 0 |
| CU3715 | 500 | 0 | 1 | 1216 |
| CU14309 | 2500 | 0 | 1 | 1200 |
| CU8122 | 1500 | 0 | 1 | 3000 |
| CU4983 | 900 | 0 | 1 | 1600 |
| CU135 | 1300 | 0 | 1 | 3000 |
| CU878 | 900 | 0 | 1 | 350 |
| CU13543 | 1500 | 0 | 2 | 2728 |
| CU2943 | 700 | 0 | 1 | 0 |
+-----+-----+-----+-----+
Returned 25 row(s) in 0.52s
[bigdatalite.localdomain:21000] >
```

Introducing HUE Impala GUI interface : Try out some of the queries using the HUE GUI.



6.7 Impala Summary

From the Impala exercises you can see that Impala shares a lot with Hive as far as being effective at interacting with data in the Hadoop cluster and using HiveQL. Where Impala is different is achieving real database speeds but at a price. Impala cannot scale to the same data volumes as hive and also cannot do SerDe transformations like hive. But you can see that running the same queries against the same tables as Hive Impala is dramatically faster.

7. WORKING WITH THE ORACLE LOADER FOR HADOOP

8.1 Introduction to the Oracle Loader for Hadoop

Oracle Loader for Hadoop is an efficient and high-performance loader for fast movement of data from a Hadoop cluster into a table in an Oracle database. It pre-partitions the data if necessary and transforms it into a database-ready format. It can also sort records by primary key or user-specified columns before loading the data or creating output files. Oracle Loader for Hadoop uses the parallel processing framework of Hadoop to perform these preprocessing operations, which other loaders typically perform on the database server as part of the load process. Offloading these operations to Hadoop reduces the CPU requirements on the database server, thereby reducing the performance impact on other database tasks.

Oracle Loader for Hadoop is a Java Map/Reduce application that balances the data across reducers to help maximize performance. It works with a range of input data formats that present the data as records with fields. It can read from sources that have the data already in a record format (such as Avro files or Hive tables), or it can split the lines of a text file into fields.

You can run Oracle Loader for Hadoop simply by using the hadoop command-line utility or attach it as last step of an existing MapReduce job, hence automating the loading task. In the command line, you provide configuration settings with the details of the job. You typically provide these settings in a job configuration file. You can optionally identify the name of a file that maps input fields to the columns of the target table in an Oracle Database.

If you have Java programming skills, you can extend the types of data that the loader can handle by defining custom input formats. Then Oracle Loader for Hadoop uses your code to extract the fields and records.

Oracle Loader for Hadoop works with a range of input data formats. It handles skew in the input data to help maximize performance.

There are two modes for loading the data into an Oracle database from a Hadoop cluster:

- Online database mode: Oracle Loader for Hadoop can connect to the target database using the credentials provided in the job configuration file or in an Oracle wallet (for secure authentication). The loader obtains the table metadata from the database. It can insert new records directly into the target table or write them to a file in the Hadoop cluster. You can load records from an output file when the data is needed in the database, or when the database system is less busy.
- Offline database mode: This mode enables you to use Oracle Loader for Hadoop when the Oracle Database system is on a separate network from the Hadoop cluster, or is otherwise inaccessible. In this mode, Oracle Loader for Hadoop uses the information supplied in a table metadata file, which you generate using a separate utility. The loader job stores the output data in binary or text format output files on the Hadoop cluster. Loading the data into Oracle Database is a separate procedure using

another utility, such as Oracle SQL Connector for Hadoop Distributed File System (HDFS) or SQL*Loader.

8.2 Overview of Hands on Exercise

This Exercise will involve loading our transactional data, initially stored in NoSQL and now residing in an aggregate format within an HDFS file into the Oracle Database, working with the Oracle Loader for Hadoop.

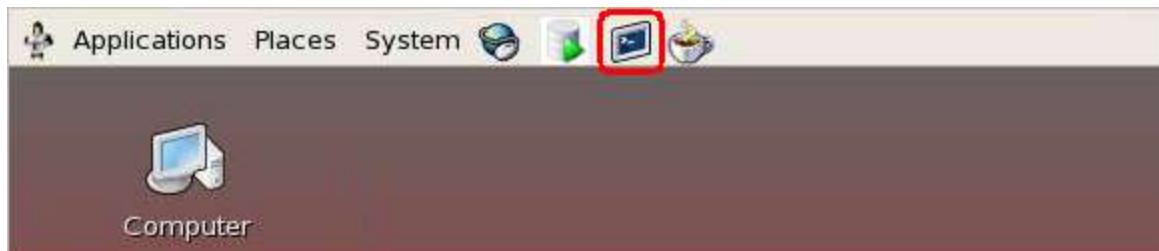
In this exercise you will:

1. Create an Oracle Database table
2. Load the data from an HDFS file into an Oracle Database table
3. View the results

Note: To successfully go through this exercise, you have to go through the NoSQL Exercise first (see Chapter 4).

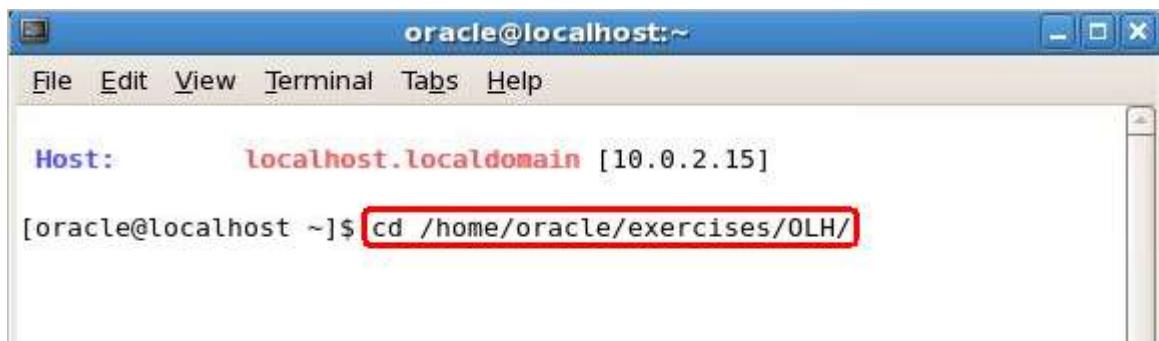
8.3 Loading HDFS data into the Oracle Database

1. All of the setup and execution for this exercise can be done from the terminal, hence open a terminal by double clicking on the **Terminal icon** on the desktop.



2. To get into the folder where the scripts for this exercise are, type in the terminal:

```
cd /home/oracle/exercises/OLH/  
Then press Enter
```



3. First we will create the table that will hold our transactional data. Let's review the script that creates the table. Go to the terminal and type:

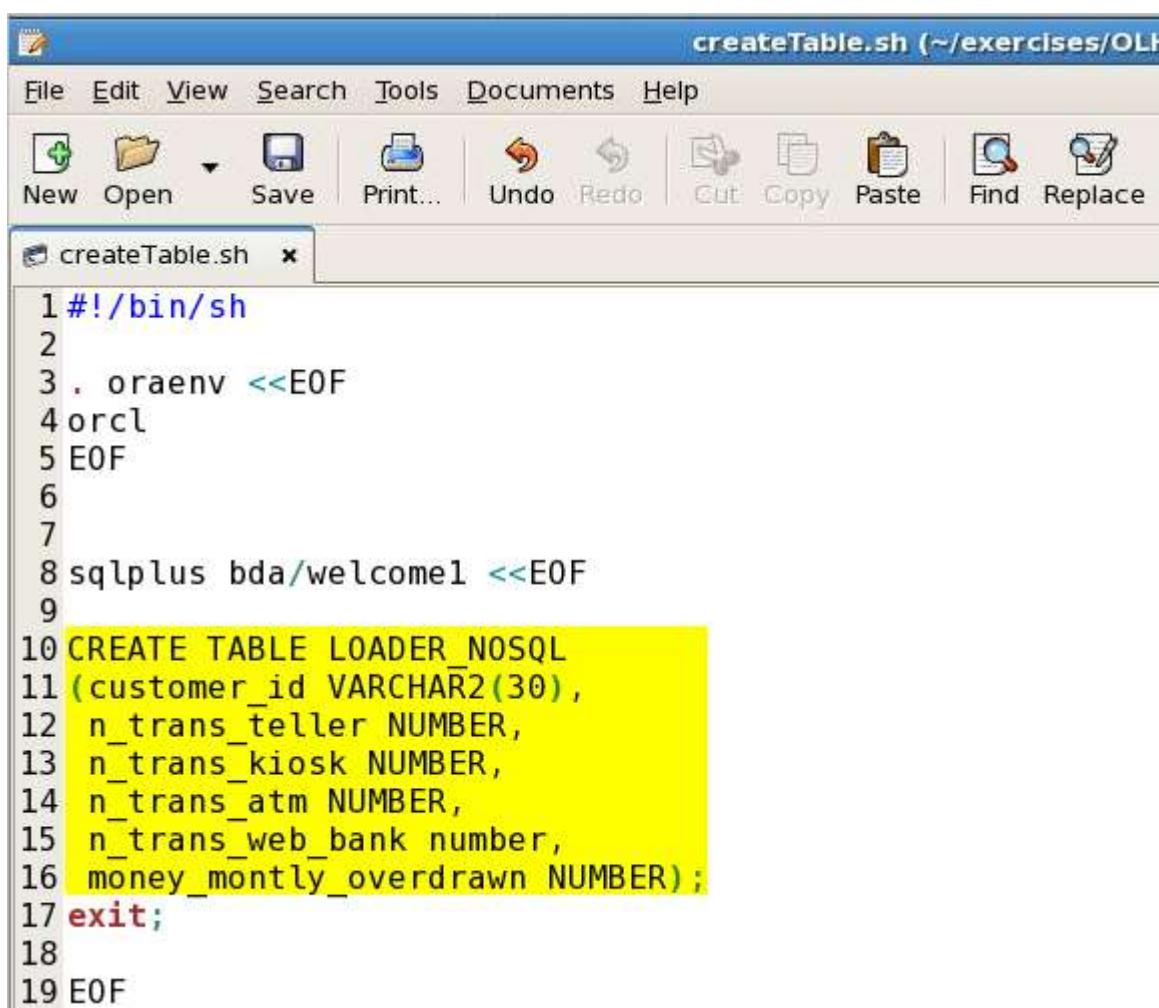
```
gedit createTable.sh
```

Then press **Enter**



```
oracle@localhost:~/exercises/OLH
File Edit View Terminal Tabs Help
Host: localhost.localdomain [10.0.2.15]
[oracle@localhost ~]$ cd /home/oracle/exercises/OLH/
[oracle@localhost OLH]$ gedit createTable.sh
```

A new table called LOADER_NOSQL is created in the BDA schema. It will hold the customer_id, the number of transactions by type (teller, kiosk, atm and web bank) and the sums of the transactions. Please notice how the structure of the table closely resembles the structure of the HDFS file that we created when running the Map-Reduce job on NoSQL data.



```
createTable.sh (~/exercises/OLH)
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
createTable.sh x
1#!/bin/sh
2
3 . oraenv <<EOF
4 orcl
5 EOF
6
7
8 sqlplus bda/welcome1 <<EOF
9
10 CREATE TABLE LOADER_NOSQL
11 (customer_id VARCHAR2(30),
12 n_trans_teller NUMBER,
13 n_trans_kiosk NUMBER,
14 n_trans_atm NUMBER,
15 n_trans_web_bank number,
16 money_montly_overdrawn NUMBER);
17 exit;
18
19 EOF
```

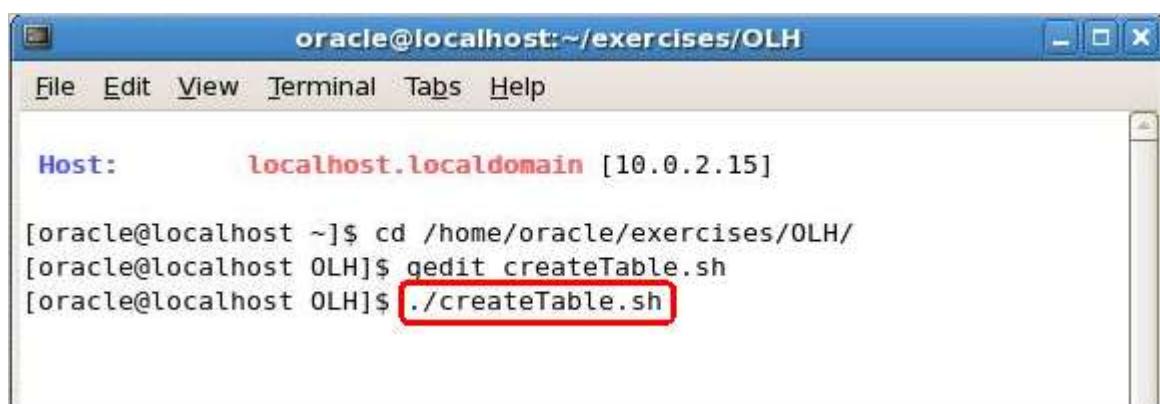
4. When you are done evaluating the table creation script, click on the X in the right upper corner of the window to close it.



5. Let's proceed to run the script. Go to the terminal and type:

```
./createTable.sh
```

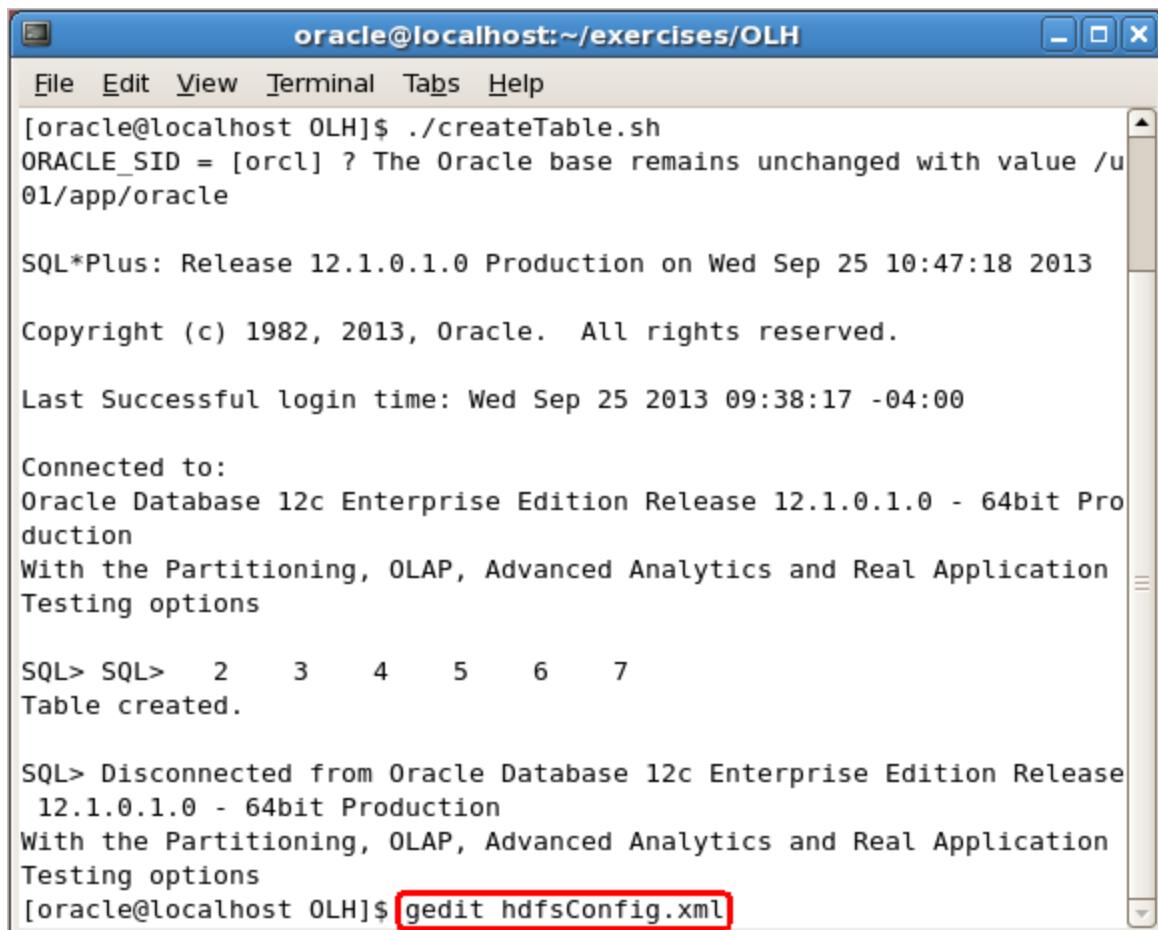
Then press **Enter**



6. The Oracle Loader for Hadoop makes use of two configuration files that have been pre-built for this exercise, hdfsConfig.xml and hdfsMap.xml, that contain useful information necessary to load the data from the HDFS file into the Oracle Database. Let's review hdfsConfig.xml. Go to the terminal and type:

```
gedit hdfsConfig.xml
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/OLH". The window contains the following text:

```
[oracle@localhost OLH]$ ./createTable.sh
ORACLE_SID = [orcl] ? The Oracle base remains unchanged with value /u
01/app/oracle

SQL*Plus: Release 12.1.0.1.0 Production on Wed Sep 25 10:47:18 2013

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Last Successful login time: Wed Sep 25 2013 09:38:17 -04:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Pro
duction
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options

SQL> SQL> 2 3 4 5 6 7
Table created.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost OLH]$ gedit hdfsConfig.xml
```

The command `gedit hdfsConfig.xml` is highlighted with a red box.

Notice the InputFormat class used by the Oracle Loader for Hadoop: DelimitedTextInputFormat, as the input for the Oracle Loader for Hadoop is a comma delimited text file. The *input.dir* property specifies the directory where the input file resides in HDFS, while the *output.dir* directory specifies the path in HDFS where the Oracle Loader for Hadoop will store its output. Please notice that the input directory is /user/oracle/loader/, which is the output directory of the Hadoop Map/Reduce job that aggregated the NoSQL records in a previous exercise. The path to the previously mentioned hdfsMap.xml file is also specified. We will review this file in a later step.

```
<property>
  <name>mapreduce.inputformat.class</name>
  <value>oracle.hadoop.loader.lib.input.DelimitedTextInputFormat</value>
</property>
<property>
  <name>mapreduce.outputformat.class</name>
  <value>oracle.hadoop.loader.lib.output.JDBCOutputFormat</value>
</property>
<property>
  <name>mapred.job.name</name>
  <value>OraLoader</value>
</property>
<property>
  <name>mapred.input.dir</name>
  <value>/user/oracle/loader/</value>
</property>
<property>
  <name>mapred.output.dir</name>
  <value>/user/oracle/output3/</value>
</property>
<property>
  <name>oracle.hadoop.loader.loaderMapFile</name>
  <value>file:///home/oracle/exercises/OLH/hdfsMap.xml</value>
</property>
<property>
  <name>oracle.hadoop.loader.extTabDirectoryName</name>
  <value>EXTERNAL_DIR</value>
```

If you scroll down you can find other useful information, like the character used to delimit the fields in the text file (comma), the Oracle Database connection URL, the user schema name and password. Again, the password could be stored within Oracle Wallet to provide secure authentication which is more likely in production environments! The last property shown in the screenshot from below specifies the name of the fields in the input text file. We need these names to perform a mapping between them and the Oracle Database columns, through the hdfsMap.xml file.

```
<property>
  <name>oracle.hadoop.loader.input.fieldTerminator</name>
  <value>,</value>
</property>
<property>
  <name>oracle.hadoop.loader.connection.url</name>
  <value>jdbc:oracle:thin:@//localhost:1521/orcl</value>
</property>
<property>
  <name>oracle.hadoop.loader.connection.user</name>
  <value>BDA</value>
</property>
<property>
  <name>oracle.hadoop.loader.connection.password</name>
  <value>welcomel</value>
</property>
<property>
  <name>oracle.hadoop.loader.logBadRecords</name>
  <value>true</value>
</property>
<property>
  <name>oracle.hadoop.loader.input.fieldNames</name>
  <value>customerid,teller,kiosk,atm,webbank,sum</value>
</property>
```

- When you are done evaluating the configuration file, click on the X in the right upper corner of the window to close it.



8. Let's review hdfsMap.xml file. Go to the terminal and type:

```
gedit hdfsMap.xml
Then press Enter
```

```
oracle@localhost:~/exercises/OLH
File Edit View Terminal Tabs Help
ORACLE_SID = [orcl] ? The Oracle base remains unchanged with value /u01/app/oracle

SQL*Plus: Release 12.1.0.1.0 Production on Wed Sep 25 10:47:18 2013

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Last Successful login time: Wed Sep 25 2013 09:38:17 -04:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> SQL> 2 3 4 5 6 7
Table created.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
[oracle@localhost OLH]$ gedit hdfsConfig.xml
[oracle@localhost OLH]$ gedit hdfsMap.xml
```

The file specifies the mapping between the fields in the text file and the Oracle Database table columns.

```
<?xml version="1.0"?>
<LOADER_MAP>
<SCHEMA>BDA</SCHEMA>
<TABLE>LOADER_NOSQL</TABLE>
<COLUMN field="customerid">CUSTOMER_ID</COLUMN>
<COLUMN field="teller">N_TRANS_TELLER</COLUMN>
<COLUMN field="kiosk">N_TRANS_KIOSK</COLUMN>
<COLUMN field="atm">N_TRANS_ATM</COLUMN>
<COLUMN field="webbank">N_TRANS_WEB_BANK</COLUMN>
<COLUMN field="sum">MONEY_MONTLY_OVERDRAWN</COLUMN>
</LOADER_MAP>
```

9. When you are done evaluating the file, click on the **X** in the right upper corner of the window to close it.



10. Before running OLH, let's make sure the output HDFS directory doesn't exist from a previous run of the exercise. To do that, we will issue a command that deletes it. Go to the terminal and type:

```
hadoop fs -rm -r /user/oracle/output3
```

Then press **Enter**

```
oracle@localhost:~/exercises/OLH
File Edit View Terminal Tabs Help
01/app/oracle

SQL*Plus: Release 12.1.0.1.0 Production on Wed Sep 25 10:47:18 2013

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Last Successful login time: Wed Sep 25 2013 09:38:17 -04:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options

SQL> SQL> 2 3 4 5 6 7
Table created.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost OLH]$ gedit hdfsConfig.xml
[oracle@localhost OLH]$ gedit hdfsMap.xml
[oracle@localhost OLH]$ hadoop fs -rm -r /user/oracle/output3
```

Note: You may receive an error stating that the directory doesn't exist. You can ignore it.

11. We are now ready to run the Oracle Loader for Hadoop. Go to the terminal and type:

```
hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader -  
conf hdfsConfig.xml
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/OLH". The window contains the following text:

```
Copyright (c) 1982, 2013, Oracle. All rights reserved.  
Last Successful login time: Wed Sep 25 2013 09:38:17 -04:00  
Connected to:  
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application  
Testing options  
SQL> SQL> 2 3 4 5 6 7  
Table created.  
SQL> Disconnected from Oracle Database 12c Enterprise Edition Release  
12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application  
Testing options  
[oracle@localhost OLH]$ gedit hdfsConfig.xml  
[oracle@localhost OLH]$ gedit hdfsMap.xml  
[oracle@localhost OLH]$ hadoop fs -rm -r /user/oracle/output3  
rm: '/user/oracle/output3': No such file or directory  
[oracle@localhost OLH]$ hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader -conf hdfsConfig.xml
```

The command at the bottom is highlighted with a red rectangle.

In the output of the Map-Reduce job launched by the Oracle Loader for Hadoop we can see that 1015 rows were processed.

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/OLH". The window displays the output of a Map-Reduce job. Key statistics shown include:

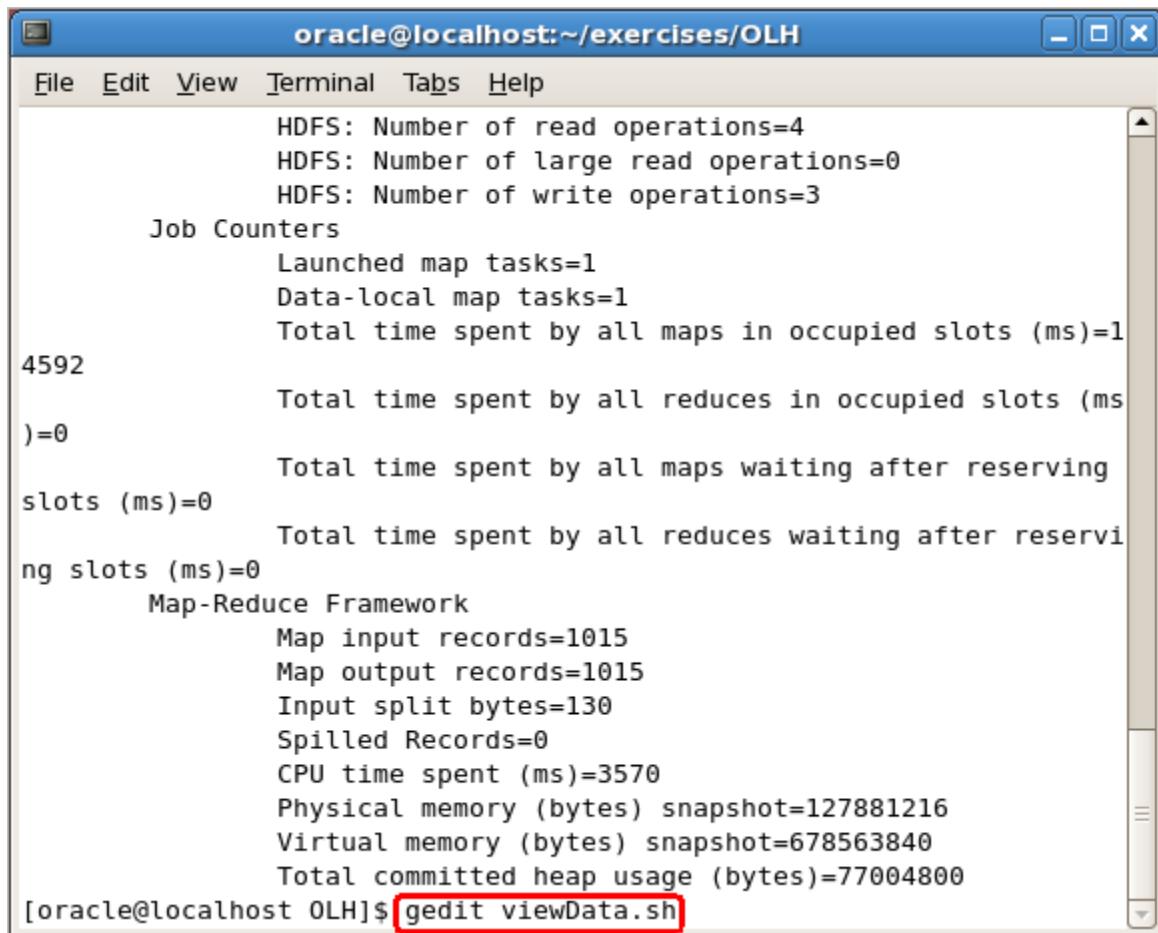
- HDFS metrics: Number of read operations=4, large read operations=0, write operations=3.
- Job Counters: Launched map tasks=1, Data-local map tasks=1, Total time spent by all maps in occupied slots (ms)=14592.
- Total time spent by all reduces in occupied slots (ms)=0.
- Total time spent by all maps waiting after reserving slots (ms)=0.
- Total time spent by all reduces waiting after reserving slots (ms)=0.
- Map-Reduce Framework metrics:
 - Map input records=1015 (highlighted)
 - Map output records=1015 (highlighted)
 - Input split bytes=130
 - Spilled Records=0
 - CPU time spent (ms)=3570 (highlighted)
 - Physical memory (bytes) snapshot=127881216
 - Virtual memory (bytes) snapshot=678563840
 - Total committed heap usage (bytes)=77004800

[oracle@localhost OLH]\$

12. Let's view the results. We will query the previously created Oracle Database table to see if the rows were successfully imported from the HDFS file. We have a script that queries the database table. Let's review it. Go to the terminal and type:

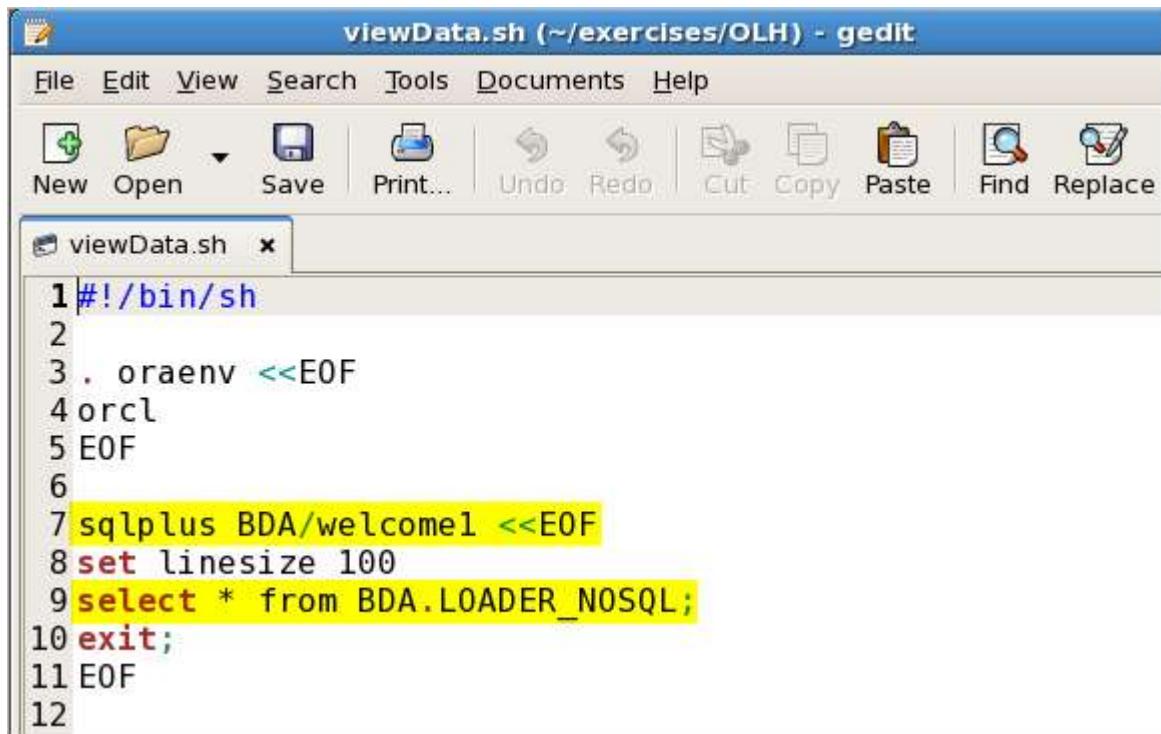
```
gedit viewData.sh
```

Then press **Enter**



```
oracle@localhost:~/exercises/OLH
File Edit View Terminal Tabs Help
HDFS: Number of read operations=4
HDFS: Number of large read operations=0
HDFS: Number of write operations=3
Job Counters
    Launched map tasks=1
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=1
4592
    Total time spent by all reduces in occupied slots (ms)
)=0
    Total time spent by all maps waiting after reserving
slots (ms)=0
    Total time spent by all reduces waiting after reservi
ng slots (ms)=0
Map-Reduce Framework
    Map input records=1015
    Map output records=1015
    Input split bytes=130
    Spilled Records=0
    CPU time spent (ms)=3570
    Physical memory (bytes) snapshot=127881216
    Virtual memory (bytes) snapshot=678563840
    Total committed heap usage (bytes)=77004800
[oracle@localhost OLH]$ gedit viewData.sh
```

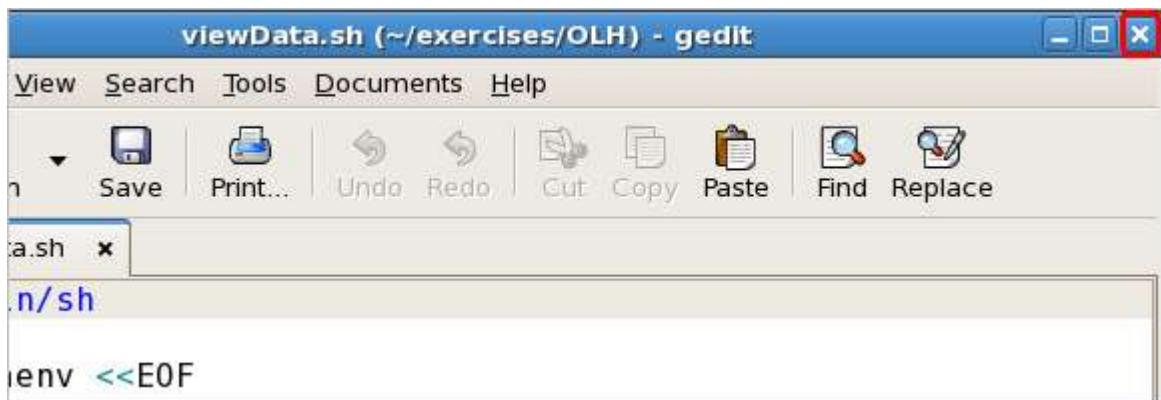
The script connects to sqlplus with the BDA user and retrieves every column from the LOADER_NOSQL table of the BDA schema.



A screenshot of the gedit text editor window. The title bar says "viewData.sh (~/exercises/OLH) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar below has icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main text area contains the following script:

```
1#!/bin/sh
2
3 . oraenv <<EOF
4 orcl
5 EOF
6
7 sqlplus BDA/welcome1 <<EOF
8 set linesize 100
9 select * from BDA.LOADER_NOSQL;
10 exit;
11 EOF
12
```

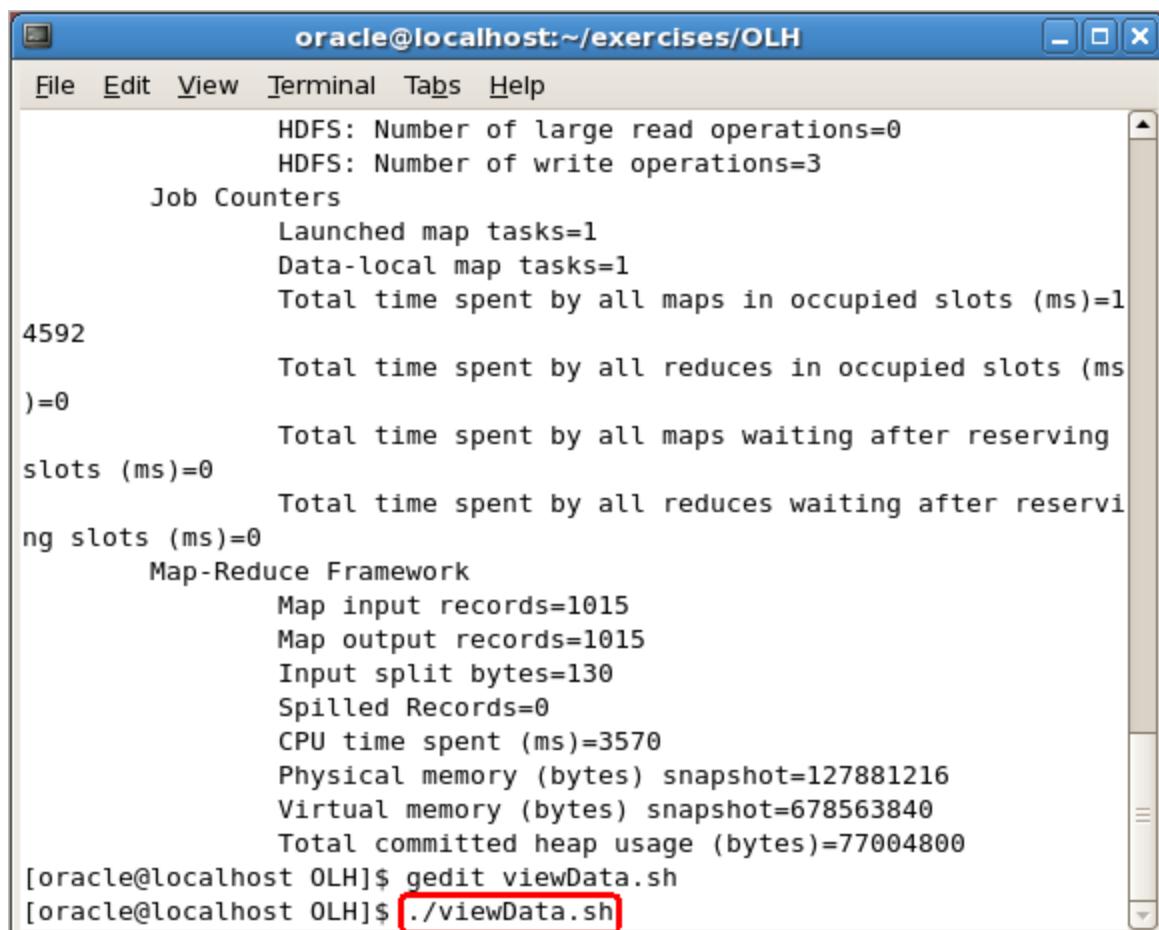
13. When you are done evaluating the script, click on the X in the right upper corner of the window to close it.



14. Time to run the script that queries the LOADER_NOSQL table. Go to the terminal and type:

```
./viewData.sh
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/OLH". The window contains the following text output from the script:

```
HDFS: Number of large read operations=0
HDFS: Number of write operations=3
Job Counters
    Launched map tasks=1
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=1
4592
    Total time spent by all reduces in occupied slots (ms)
)=0
    Total time spent by all maps waiting after reserving
slots (ms)=0
    Total time spent by all reduces waiting after reservi
ng slots (ms)=0
Map-Reduce Framework
    Map input records=1015
    Map output records=1015
    Input split bytes=130
    Spilled Records=0
    CPU time spent (ms)=3570
    Physical memory (bytes) snapshot=127881216
    Virtual memory (bytes) snapshot=678563840
    Total committed heap usage (bytes)=77004800
[oracle@localhost OLH]$ gedit viewData.sh
[oracle@localhost OLH]$ ./viewData.sh
```

The command `./viewData.sh` is highlighted with a red rectangle.

The 1015 customer-related rows originating in the NoSQL Database are now present in the Oracle Database.

```
oracle@localhost:~/exercises/OLH
File Edit View Terminal Tabs Help

CU998          0          0
4      2500
      53.420002

CUSTOMER_ID      N_TRANS_TELLER N_TRANS_KIOSK N_TRANS_A
TM N_TRANS_WEB_BANK
-----
-- 
-- 
MONEY_MONTLY_OVERDRAWN

CU9988          0          1
1      600
      53.07

1015 rows selected.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost OLH]$
```

15. This exercise is now finished, so you can close the terminal window. Go to the terminal and type:

```
exit
```

Then press **Enter**

```

oracle@localhost:~/exercises/OLH
File Edit View Terminal Tabs Help

CU998          0          0
4      2500
      53.420002

CUSTOMER_ID      N_TRANS_TELLER N_TRANS_KIOSK N_TRANS_A
TM N_TRANS_WEB_BANK
-----
-- -----
MONEY_MONTLY_OVERDRAWN
-----
CU9988         0          1
1      600
      53.07

1015 rows selected.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost OLH]$ exit

```

8.4 Summary

In this chapter we've shown how to load data from delimited text files residing in HDFS into Oracle Database tables, using the Oracle Loader for Hadoop. Please note that the Oracle Loader for Hadoop can also be used to read data from Oracle NoSQL Databases directly.

For using the loader usually one would go through the following steps:

1. Selecting one of the predefined InputFormats, or build your own
2. Creating the configuration files
3. Invoking the Oracle Loader for Hadoop

This way, Oracle users can have easy access to HDFS/Hive/NoSQL data, using the tools they know and taking advantage of the powerful capabilities provided by the Oracle Database.

8. WORKING WITH THE ORACLE SQL CONNECTOR FOR HDFS

9.1 Introduction to the Oracle SQL Connector for HDFS

Using Oracle SQL Connector for HDFS, you can use Oracle Database to access and analyze data residing in Hadoop in these formats:

- Data Pump files in HDFS
- Delimited text files in HDFS
- Non-partitioned Hive tables on top of delimited text files

For other file formats, such as JSON files, you can stage the input in Hive tables before using Oracle SQL Connector for HDFS.

Oracle SQL Connector for HDFS leverages Oracle Database external tables to provide Oracle Database with read access to Hive tables as well as delimited text and Data Pump files in HDFS. An **external table** is an Oracle Database object that identifies the location of data outside of a database. Oracle Database accesses the data by using the metadata provided when the external table was created. By querying the external tables, you can access data stored in HDFS as if that data were stored in tables in an Oracle Database.

To create an external table for this purpose, you use the ExternalTable command-line tool provided with Oracle SQL Connector for HDFS. You provide ExternalTable with information about the data source in Hadoop and about your schema in an Oracle Database. You can provide this information either as parameters to the ExternalTable command directly or in an XML file.

When the external table is ready, you can query the data the same way as any other database table. You can query and join data in HDFS or a Hive table with other database-resident data.

You may also perform bulk loads of data into Oracle database tables using SQL. However, for large bulk loads the Oracle Loader for Hadoop should be considered.

9.2 Overview of Hands on Exercise

This exercise will involve working with Oracle external tables. We need the data related to monthly cash accounts, previously processed in Pig, to be available inside the Oracle Database. Therefore, the HDFS file that holds this data will be connected to the Oracle Database using external tables. To do this, we will use the Oracle SQL Connector for HDFS.

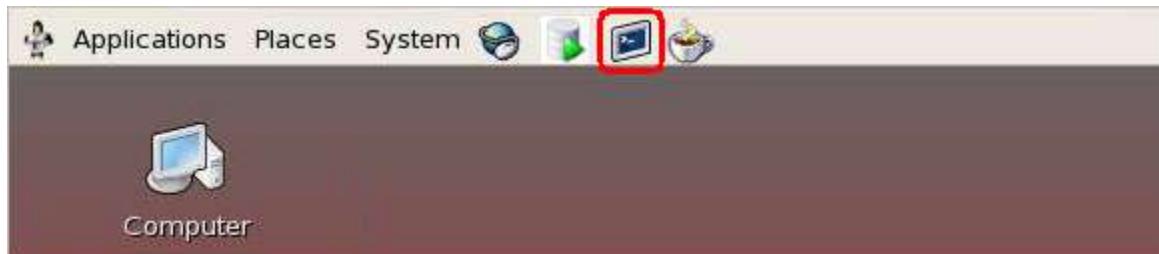
In this exercise you will:

1. Create an external table in the Oracle Database
2. Query the external table which stores data in HDFS

Note: To successfully go through this exercise, you have to go through the Pig Exercise first (see Chapter 5).

9.3 Configuring External Tables stored in HDFS

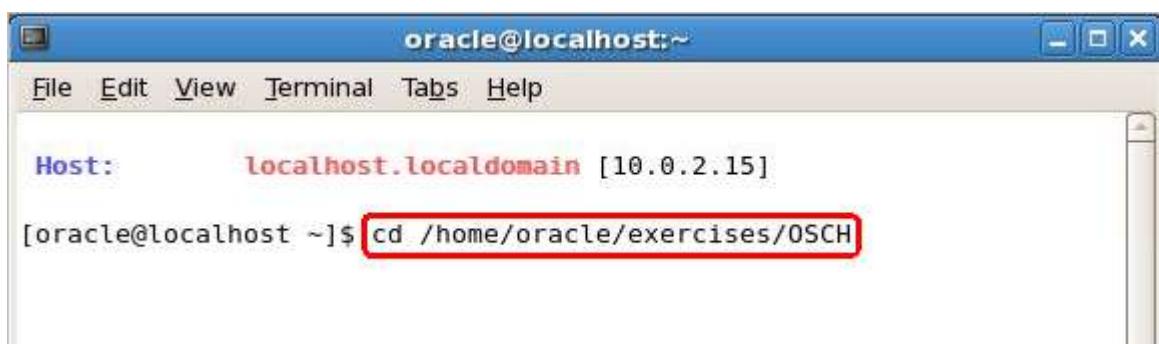
1. All of the setup and execution for this exercise can be done from the terminal, hence open a terminal by double clicking on the **Terminal icon** on the desktop.



2. To get into the folder where the scripts for the external table exercise are, type in the terminal:

```
cd /home/oracle/exercises/OSCH
```

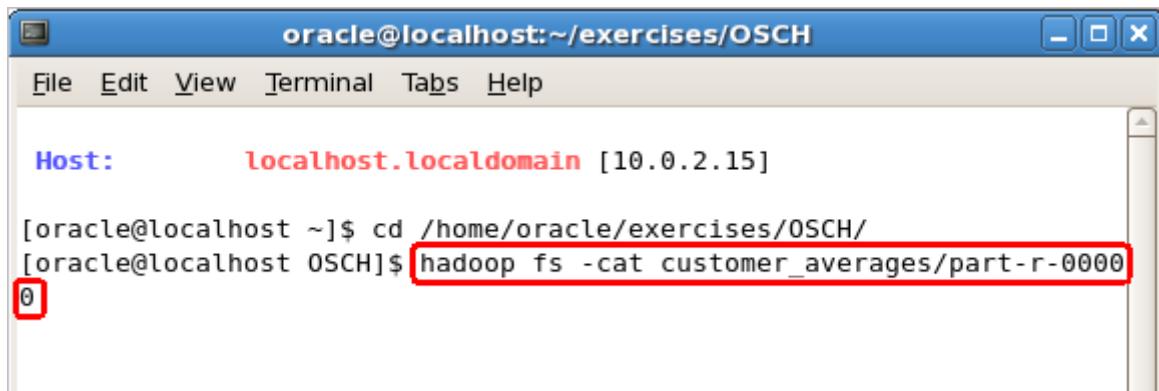
Then press **Enter**



3. Let's look at the HDFS file which contains the monthly cash account situations for our sample customers. Go to the terminal and type:

```
hadoop fs -cat customer_averages/part-r-00000
```

Then press **Enter**

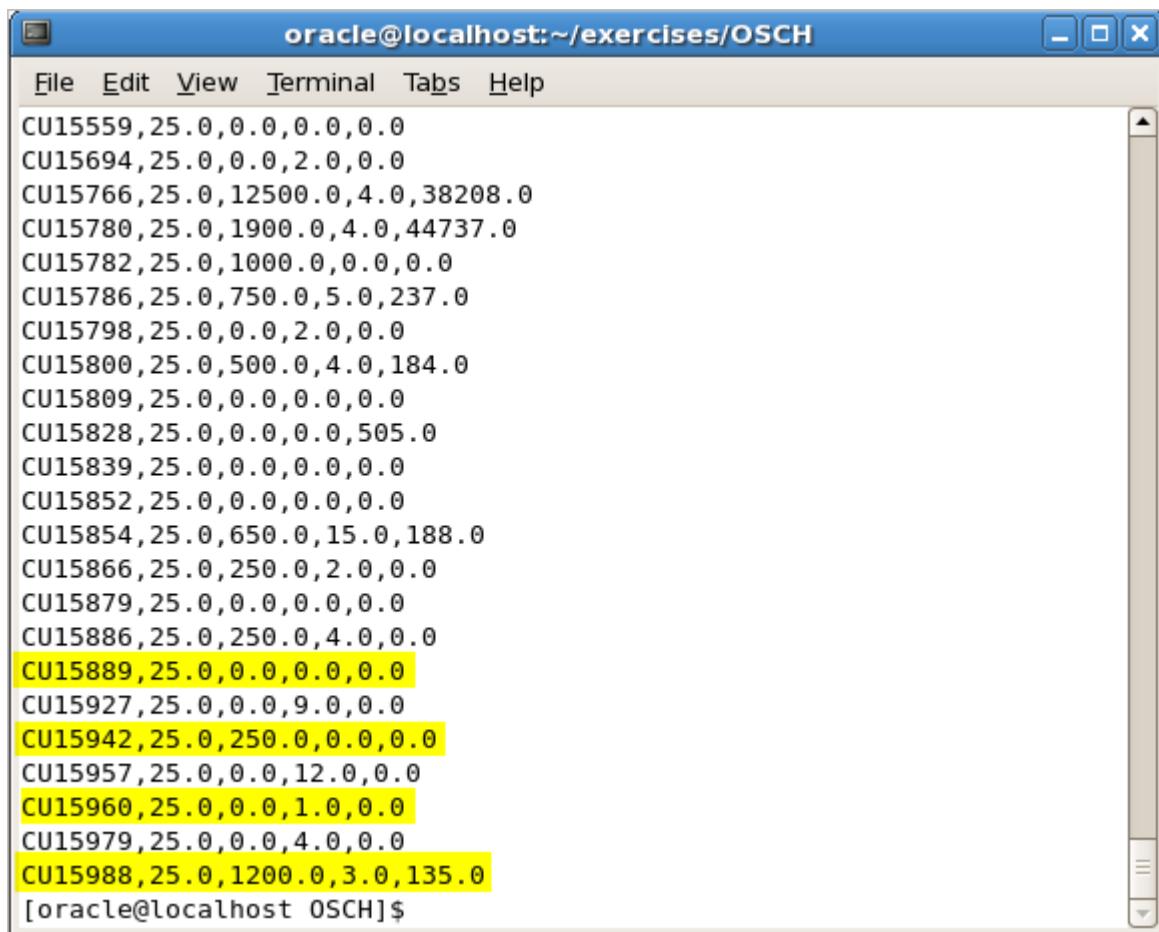


```
oracle@localhost:~/exercises/OSCH
File Edit View Terminal Tabs Help

Host:      localhost.localdomain [10.0.2.15]

[oracle@localhost ~]$ cd /home/oracle/exercises/OSCH/
[oracle@localhost OSCH]$ hadoop fs -cat customer_averages/part-r-00000
0
```

The data in this file has been extracted from our internal systems and processed and aggregated in Pig, in order to provide a view of the monthly cash accounts for our sample customers. This text file is comprised of 5 columns, delimited by commas. The first column represents the customer id, followed by the monthly checking amount, bank funds, number of checks written and the total amount corresponding to automatic payments.



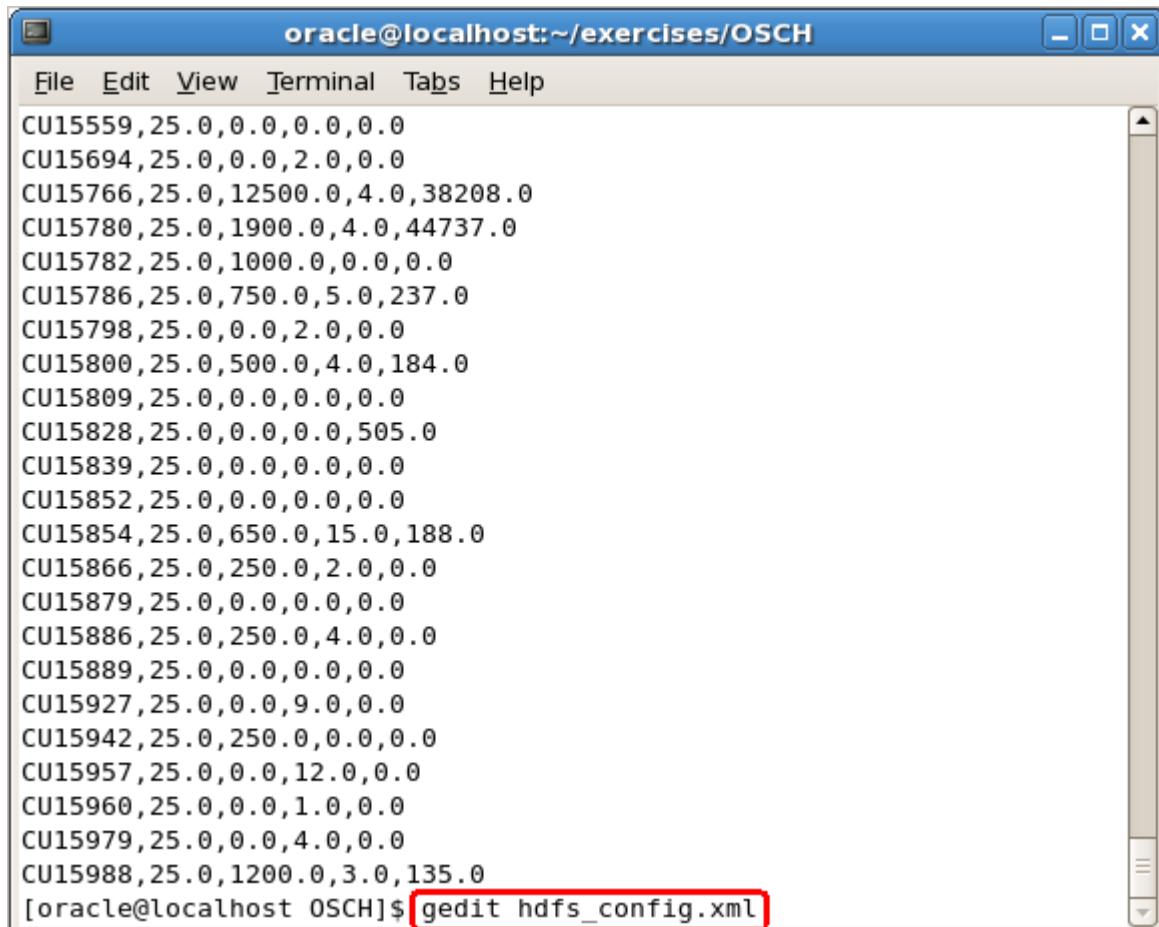
```
oracle@localhost:~/exercises/OSCH
File Edit View Terminal Tabs Help

CU15559,25.0,0.0,0.0,0.0
CU15694,25.0,0.0,2.0,0.0
CU15766,25.0,12500.0,4.0,38208.0
CU15780,25.0,1900.0,4.0,44737.0
CU15782,25.0,1000.0,0.0,0.0
CU15786,25.0,750.0,5.0,237.0
CU15798,25.0,0.0,2.0,0.0
CU15800,25.0,500.0,4.0,184.0
CU15809,25.0,0.0,0.0,0.0
CU15828,25.0,0.0,0.0,505.0
CU15839,25.0,0.0,0.0,0.0
CU15852,25.0,0.0,0.0,0.0
CU15854,25.0,650.0,15.0,188.0
CU15866,25.0,250.0,2.0,0.0
CU15879,25.0,0.0,0.0,0.0
CU15886,25.0,250.0,4.0,0.0
CU15889,25.0,0.0,0.0,0.0
CU15927,25.0,0.0,9.0,0.0
CU15942,25.0,250.0,0.0,0.0
CU15957,25.0,0.0,12.0,0.0
CU15960,25.0,0.0,1.0,0.0
CU15979,25.0,0.0,4.0,0.0
CU15988,25.0,1200.0,3.0,135.0
[oracle@localhost OSCH]$
```

4. The Oracle SQL Connector for HDFS will create an external table in the Oracle Database that points to data in the HDFS file. OSCH requires a XML file for the configuration, a file that has been prebuilt for this exercise (hdfs_config.xml). Let's review the configuration file. Go to the terminal and type:

```
gedit hdfs_config.xml
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/OSCH". The window contains a list of approximately 30 data rows, each consisting of a unique identifier followed by six numerical values separated by commas. Below this list, the terminal prompt "[oracle@localhost OSCH]\$" is visible, followed by the command "gedit hdfs_config.xml" which is highlighted with a red rectangle.

Row ID	Value 1	Value 2	Value 3	Value 4	Value 5	Value 6
CU15559	25.0	0.0	0.0	0.0	0.0	0.0
CU15694	25.0	0.0	2.0	0.0		
CU15766	25.0	12500.0	4.0	38208.0		
CU15780	25.0	1900.0	4.0	44737.0		
CU15782	25.0	1000.0	0.0	0.0		
CU15786	25.0	750.0	5.0	237.0		
CU15798	25.0	0.0	2.0	0.0		
CU15800	25.0	500.0	4.0	184.0		
CU15809	25.0	0.0	0.0	0.0		
CU15828	25.0	0.0	0.0	505.0		
CU15839	25.0	0.0	0.0	0.0		
CU15852	25.0	0.0	0.0	0.0		
CU15854	25.0	650.0	15.0	188.0		
CU15866	25.0	250.0	2.0	0.0		
CU15879	25.0	0.0	0.0	0.0		
CU15886	25.0	250.0	4.0	0.0		
CU15889	25.0	0.0	0.0	0.0		
CU15927	25.0	0.0	9.0	0.0		
CU15942	25.0	250.0	0.0	0.0		
CU15957	25.0	0.0	12.0	0.0		
CU15960	25.0	0.0	1.0	0.0		
CU15979	25.0	0.0	4.0	0.0		
CU15988	25.0	1200.0	3.0	135.0		

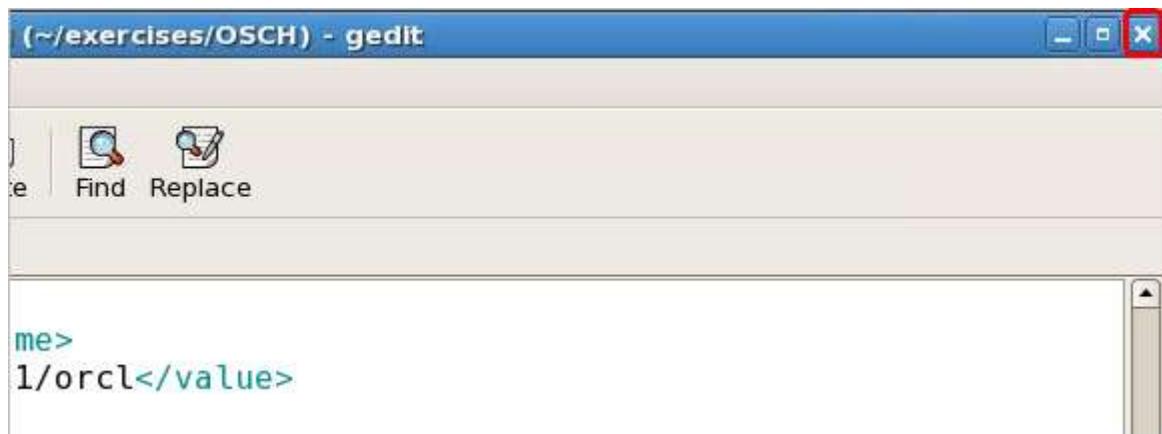
Notice the external table name, the path to the files in HDFS where the actual data is, the column separator used in the files to delimit the fields and the Oracle Database directory that the Oracle SQL Connector for HDFS uses.

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>oracle.hadoop.exttab.tableName</name>
    <value>BDA.EXTERNAL_HDFS</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.dataPaths</name>
    <value>/user/oracle/customer_averages/part-r-00000</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.fieldDelimiter</name>
    <value>,</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.defaultDirectory</name>
    <value>EXTERNAL_DIR</value>
  </property>
```

If you scroll down you can see details like the Oracle Database connection URL, the user schema name and the names of the external table columns. The Oracle SQL Connector for HDFS automatically generates the SQL command that creates the external table using this configuration file. We will see the generated SQL command later in the exercise.

```
<property>
  <name>oracle.hadoop.connection.url</name>
  <value>jdbc:oracle:thin:@localhost:1521/orcl</value>
</property>
<property>
  <name>oracle.hadoop.connection.user</name>
  <value>BDA</value>
</property>
<property>
  <name>oracle.hadoop.exttab.printStackTrace</name>
  <value>true</value>
</property>
<property>
  <name>oracle.hadoop.exttab.columnNames</name>
  <value>customer_id,checking_amount,bank_funds,monthly_
value</value>
```

5. When you are done evaluating the configuration file, click on the X in the right upper corner of the window to close it.



6. We are now ready to run the Oracle SQL Connector for HDFS, to connect the Oracle Database external table to the HDFS file. Go to the terminal and type:

```
hadoop jar $OSCH_HOME/jlib/orahdfs.jar oracle.hadoop.extbl.ExternalTable  
-conf hdfs_config.xml -createTable
```

Then press **Enter**

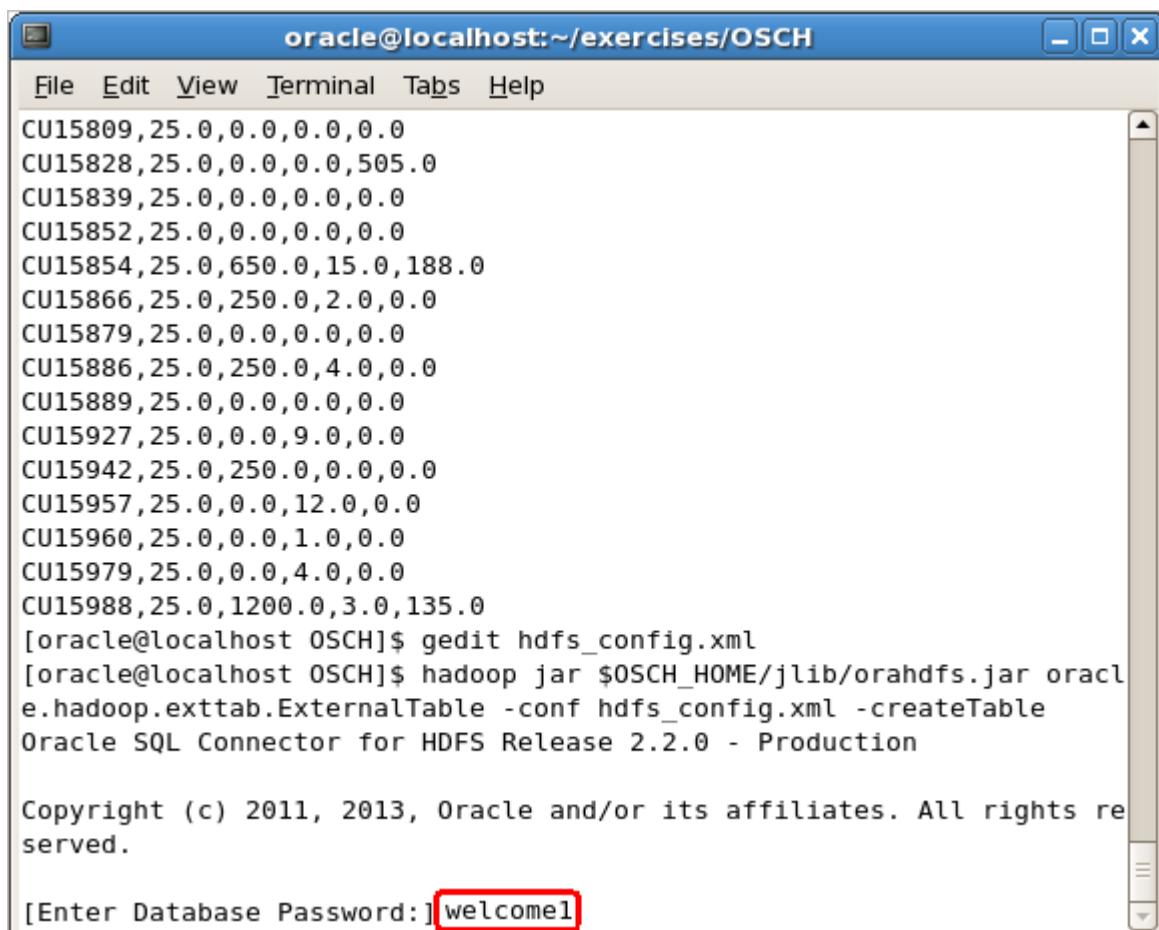
The screenshot shows a terminal window titled "oracle@localhost:~/exercises/OSCH". The window contains a list of data rows from a CSV file and a command prompt at the bottom. The command prompt shows the user running the Hadoop jar command to create an external table, with the final part of the command ("oracle.hadoop.extbl.ExternalTable -conf hdfs_config.xml -createTable") highlighted with a red rectangle.

```
CU15766,25.0,12500.0,4.0,38208.0  
CU15780,25.0,1900.0,4.0,44737.0  
CU15782,25.0,1000.0,0.0,0.0  
CU15786,25.0,750.0,5.0,237.0  
CU15798,25.0,0.0,2.0,0.0  
CU15800,25.0,500.0,4.0,184.0  
CU15809,25.0,0.0,0.0,0.0  
CU15828,25.0,0.0,0.0,505.0  
CU15839,25.0,0.0,0.0,0.0  
CU15852,25.0,0.0,0.0,0.0  
CU15854,25.0,650.0,15.0,188.0  
CU15866,25.0,250.0,2.0,0.0  
CU15879,25.0,0.0,0.0,0.0  
CU15886,25.0,250.0,4.0,0.0  
CU15889,25.0,0.0,0.0,0.0  
CU15927,25.0,0.0,9.0,0.0  
CU15942,25.0,250.0,0.0,0.0  
CU15957,25.0,0.0,12.0,0.0  
CU15960,25.0,0.0,1.0,0.0  
CU15979,25.0,0.0,4.0,0.0  
CU15988,25.0,1200.0,3.0,135.0  
[oracle@localhost OSCH]$ gedit hdfs_config.xml  
[oracle@localhost OSCH]$ hadoop jar $OSCH_HOME/jlib/orahdfs.jar oracle.hadoop.extbl.ExternalTable -conf hdfs_config.xml -createTable
```

7. You will be asked to enter a password for the code to be able to login to the database user. Enter the following information

[Enter Database Password]: welcome1
Then Press **Enter**

NOTE: No text will appear while you type



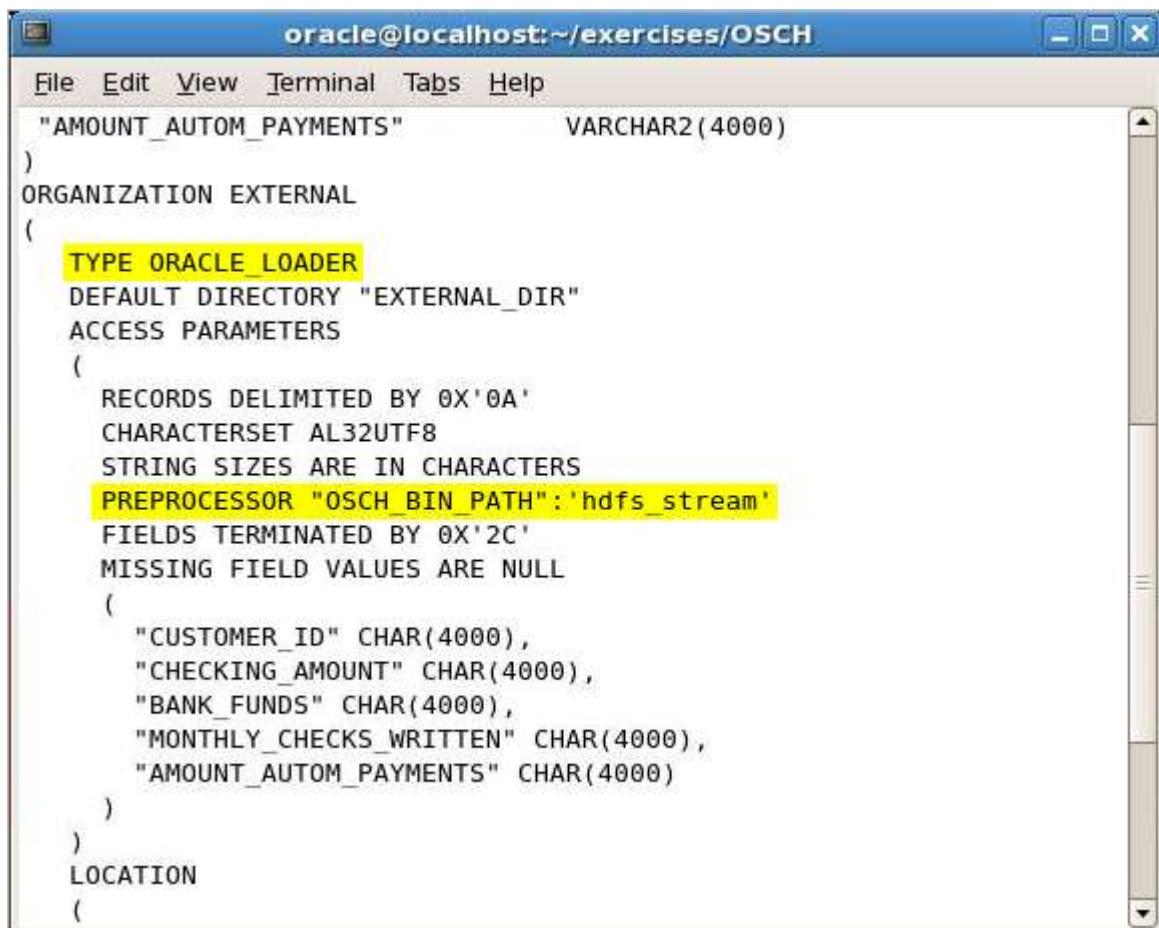
The screenshot shows a terminal window titled "oracle@localhost:~/exercises/OSCH". The window contains the following text:

```
File Edit View Terminal Tabs Help
CU15809,25.0,0.0,0.0,0.0
CU15828,25.0,0.0,0.0,505.0
CU15839,25.0,0.0,0.0,0.0
CU15852,25.0,0.0,0.0,0.0
CU15854,25.0,650.0,15.0,188.0
CU15866,25.0,250.0,2.0,0.0
CU15879,25.0,0.0,0.0,0.0
CU15886,25.0,250.0,4.0,0.0
CU15889,25.0,0.0,0.0,0.0
CU15927,25.0,0.0,9.0,0.0
CU15942,25.0,250.0,0.0,0.0
CU15957,25.0,0.0,12.0,0.0
CU15960,25.0,0.0,1.0,0.0
CU15979,25.0,0.0,4.0,0.0
CU15988,25.0,1200.0,3.0,135.0
[oracle@localhost OSCH]$ gedit hdfs_config.xml
[oracle@localhost OSCH]$ hadoop jar $OSCH_HOME/jlib/orahdfs.jar oracle.hadoop.extbl.ExternalTable -conf hdfs_config.xml -createTable
Oracle SQL Connector for HDFS Release 2.2.0 - Production

Copyright (c) 2011, 2013, Oracle and/or its affiliates. All rights reserved.

[Enter Database Password:] welcome1
```

As the execution of the Oracle SQL Connector for HDFS finishes, the external table creation code is displayed in the terminal. Looking at the code for creating the table you will notice very similar syntax for other types of external tables except for 2 lines: the preprocessor and type highlighted in the image below. The table creation code is automatically generated by the Oracle SQL Connector for HDFS, based on the XML configuration file.



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/OSCH". The window contains Oracle SQL code for creating an external table. The code includes several lines highlighted in yellow, specifically:

- "PREPROCESSOR "OSCH_BIN_PATH":'hdfs_stream'"
- "CHARACTERSET AL32UTF8"

The rest of the code is standard Oracle SQL for defining columns, table organization, access parameters, and location.

```
File Edit View Terminal Tabs Help
"AMOUNT_AUTOM_PAYMENTS"          VARCHAR2(4000)
)
ORGANIZATION EXTERNAL
(
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY "EXTERNAL_DIR"
    ACCESS PARAMETERS
    (
        RECORDS DELIMITED BY 0X'0A'
        CHARACTERSET AL32UTF8
        STRING SIZES ARE IN CHARACTERS
        PREPROCESSOR "OSCH_BIN_PATH":'hdfs_stream'
        FIELDS TERMINATED BY 0X'2C'
        MISSING FIELD VALUES ARE NULL
        (
            "CUSTOMER_ID" CHAR(4000),
            "CHECKING_AMOUNT" CHAR(4000),
            "BANK_FUNDS" CHAR(4000),
            "MONTHLY_CHECKS_WRITTEN" CHAR(4000),
            "AMOUNT_AUTOM_PAYMENTS" CHAR(4000)
        )
    )
    LOCATION
    (
```

8. We can now use SQL from the Oracle Database to read the HDFS file containing the monthly cash accounts data. We have a script that queries the external table. Let's review it. Go to the terminal and type:

```
gedit viewData.sh
```

Then press **Enter**

```
oracle@localhost:~/exercises/OSCH
File Edit View Terminal Tabs Help
FIELDS TERMINATED BY 0X'2C'
MISSING FIELD VALUES ARE NULL
(
    "CUSTOMER_ID" CHAR(4000),
    "CHECKING_AMOUNT" CHAR(4000),
    "BANK_FUNDS" CHAR(4000),
    "MONTHLY_CHECKS_WRITTEN" CHAR(4000),
    "AMOUNT_AUTOM_PAYMENTS" CHAR(4000)
)
)
LOCATION
(
    'osch-20130902035129-8149-1'
)
) PARALLEL REJECT LIMIT UNLIMITED;

The following location files were created.

osch-20130902035129-8149-1 contains 1 URI, 29193 bytes

29193 hdfs://localhost.localdomain:8020/user/oracle/customer_averages/part-r-00000

[oracle@localhost OSCH]$ gedit viewData.sh
```

The script connects to the Oracle Database with the BDA user using SQL*Plus. Once connected, it will query the external table called EXTERNAL_HDFS that points to the data from the HDFS file.

```
#!/bin/sh

. oraenv <<EOF
orcl
EOF

sqlplus BDA/welcome1 <<EOF
set linesize 100
column customer_id format A15
column checking_amount format A15
column bank_funds format A15
column monthly_checks_written format A15
column amount_autom_payments format A15
select * from BDA.EXTERNAL_HDFS;
exit;
EOF
```

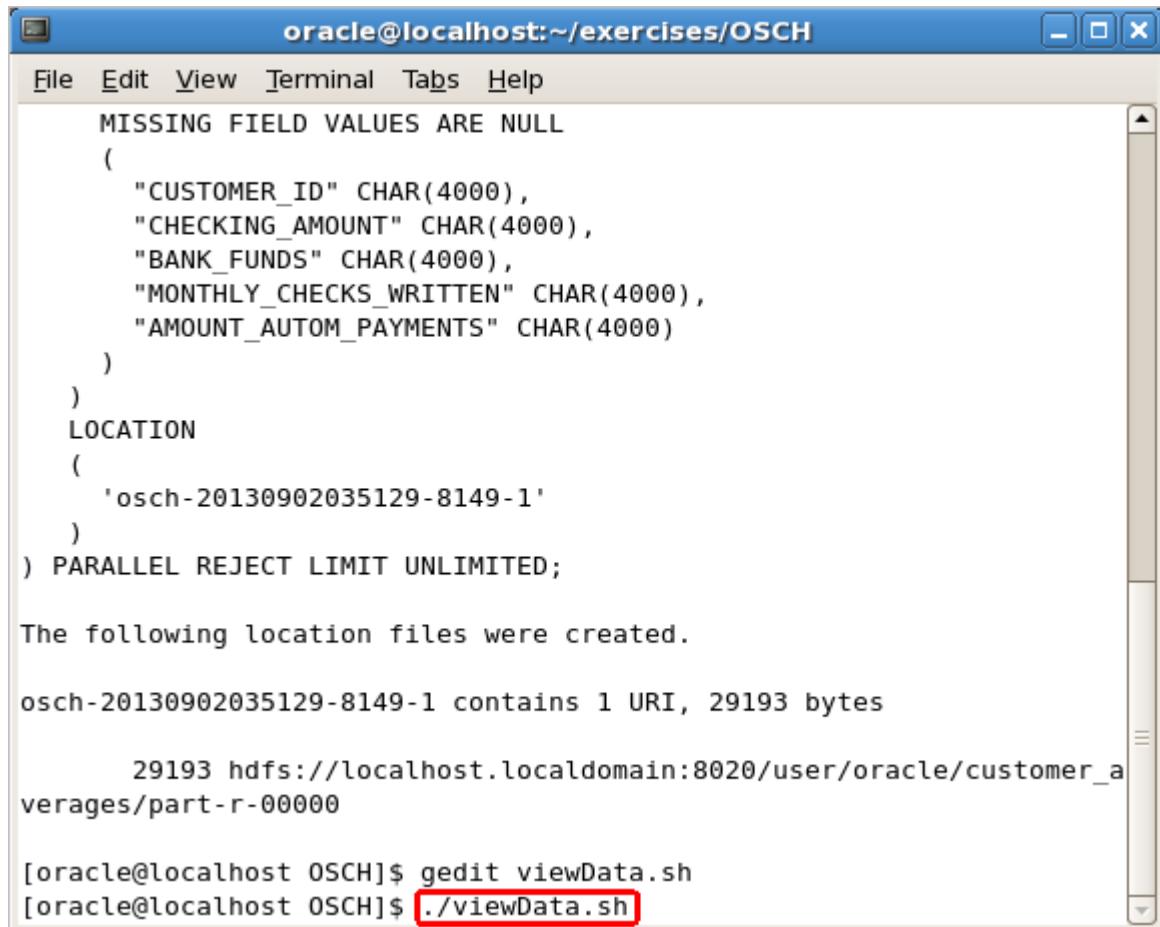
- When you are done evaluating the script, click the X in the right upper corner of the window to close it.



10. Time to run the script that queries the table. Go to the terminal and type:

```
./viewData.sh
```

Then press **Enter**



```
oracle@localhost:~/exercises/OSCH
File Edit View Terminal Tabs Help
MISSING FIELD VALUES ARE NULL
(
    "CUSTOMER_ID" CHAR(4000),
    "CHECKING_AMOUNT" CHAR(4000),
    "BANK_FUNDS" CHAR(4000),
    "MONTHLY_CHECKS_WRITTEN" CHAR(4000),
    "AMOUNT_AUTOM_PAYMENTS" CHAR(4000)
)
)
LOCATION
(
    'osch-20130902035129-8149-1'
)
) PARALLEL REJECT LIMIT UNLIMITED;

The following location files were created.

osch-20130902035129-8149-1 contains 1 URI, 29193 bytes

    29193 hdfs://localhost.localdomain:8020/user/oracle/customer_averages/part-r-00000

[oracle@localhost OSCH]$ gedit viewData.sh
[oracle@localhost OSCH]$ ./viewData.sh
```

Notice how the customer-related rows in HDFS were retrieved from the Oracle Database. The monthly cash accounts data is now available to be queried from within the Oracle Database.

```
oracle@localhost:~/exercises/OSCH
File Edit View Terminal Tabs Help
CU15854      25.0       650.0       15.0       188.0
CU15866      25.0       250.0        2.0        0.0
CU15879      25.0        0.0        0.0        0.0
CU15886      25.0       250.0        4.0        0.0
CU15889      25.0        0.0        0.0        0.0
CU15927      25.0        0.0        9.0        0.0
CU15942      25.0       250.0        0.0        0.0
CU15957      25.0        0.0       12.0        0.0

CUSTOMER_ID    CHECKING_AMOUNT BANK_FUNDS   MONTHLY_CHECKS_AMOUN
T_AUTOM_PA
-----
-----
CU15960      25.0        0.0        1.0        0.0
CU15979      25.0        0.0        4.0        0.0
CU15988      25.0      1200.0        3.0      135.0

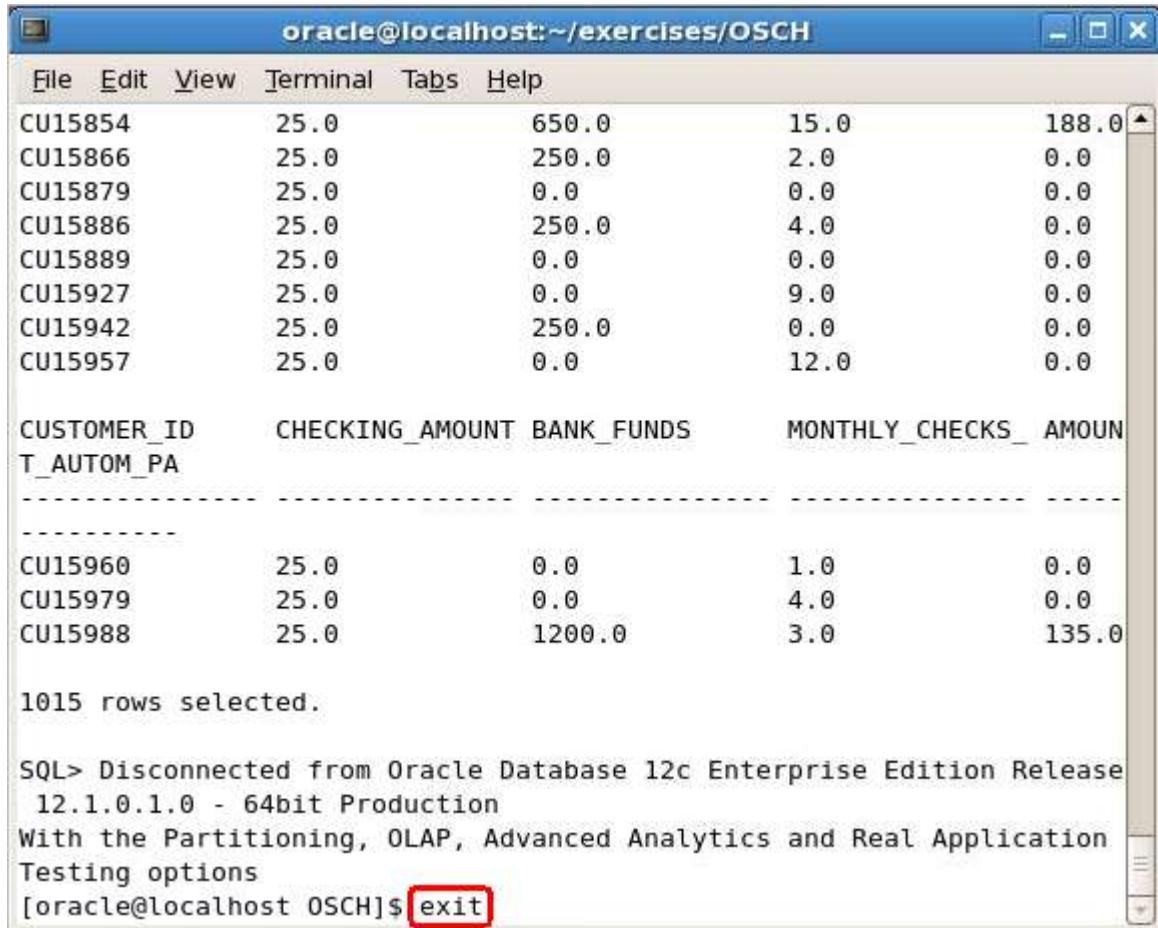
1015 rows selected.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost OSCH]$
```

11. This exercise is now finished, so we can close the terminal window. Go to the terminal and type:

```
exit
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/OSCH". The window contains the following output:

```

File Edit View Terminal Tabs Help
CU15854      25.0        650.0       15.0      188.0
CU15866      25.0        250.0       2.0       0.0
CU15879      25.0        0.0         0.0       0.0
CU15886      25.0        250.0       4.0       0.0
CU15889      25.0        0.0         0.0       0.0
CU15927      25.0        0.0         9.0       0.0
CU15942      25.0        250.0       0.0       0.0
CU15957      25.0        0.0         12.0      0.0
CUSTOMER_ID   CHECKING_AMOUNT BANK_FUNDS  MONTHLY_CHECKS_AMOUN
T_AUTOM_PA
-----
CU15960      25.0        0.0         1.0       0.0
CU15979      25.0        0.0         4.0       0.0
CU15988      25.0        1200.0      3.0      135.0
1015 rows selected.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost OSCH]$ exit

```

9.4 Summary

In this chapter we showed how data in HDFS can be queried using standard SQL right from the Oracle Database. Data from datapump files in HDFS or from non-partitioned Hive tables on top of delimited text files can also be queried through SQL via external tables created by OSCH, right from the Oracle Database. Note: The Oracle NoSQL Database provides similar capabilities for querying data from the Oracle Database. However, those capabilities are built-in in the Oracle NoSQL Database directly and not part of the Oracle SQL Connector for HDFS.

With the data stored in HDFS all of the parallelism and striping that would naturally occur is taken full advantage of while at the same time you can use all of the power and functionality of the Oracle Database. When using this method for interacting with data, parallel processing is extremely important hence when using external tables, consider enabling parallel query with this SQL command:

```
ALTER SESSION ENABLE PARALLEL QUERY;
```

Before loading data into Oracle Database from the external files created by Oracle SQL Connector for HDFS, enable parallel DDL: ALTER SESSION ENABLE PARALLEL DDL;

Before inserting data into an existing database table, enable parallel DML with this SQL command:

ALTER SESSION ENABLE PARALLEL DML;

Hints such as APPEND and PQ_DISTRIBUTE also improve performance when inserting data.

9. ORACLE ODI AND HADOOP

9.1 Introduction to ODI Application Adapter for Hadoop

Apache Hadoop is designed to handle and process data from data sources that are typically non-RDBMS and data volumes that are typically beyond what is handled by relational databases.

The Oracle Data Integrator Application Adapter for Hadoop enables data integration developers to integrate and transform data easily within Hadoop using Oracle Data Integrator. Employing familiar and easy-to-use tools and preconfigured knowledge modules, the adapter provides the following capabilities:

- Loading data into Hadoop from the local file system and HDFS.
- Performing validation and transformation of data within Hadoop.
- Loading processed data from Hadoop to Oracle Database for further processing and generating reports.

Typical processing in Hadoop includes data validation and transformations that are programmed as Map/Reduce jobs. Designing and implementing a Map/Reduce job requires expert programming knowledge. However, using Oracle Data Integrator and the Oracle Data Integrator Application Adapter for Hadoop, you do not need to write Map/Reduce jobs. Oracle Data Integrator uses Hive and the Hive Query Language (HiveQL), a SQL-like language for implementing Map/Reduce jobs. The Oracle Data Integrator graphical user interface enhances the developer's experience and productivity while enabling them to create Hadoop integrations.

When implementing a big data processing scenario, the first step is to load the data into Hadoop. The data source is typically in the local file system, HDFS or Hive tables.

After the data is loaded, you can validate and transform the data using HiveQL like you do in SQL. You can perform data validation such as checking for NULLS and primary keys, and transformations such as filtering, aggregations, set operations, and derived tables. You can also include customized procedural snippets (scripts) for processing the data.

When the data has been aggregated, condensed, or crunched down, you can load it into Oracle Database for further processing and analysis. Oracle Loader for Hadoop is recommended for optimal loading into Oracle Database.

Knowledge Modules:

KM Name	Description	Source	Target
IKM File To Hive (Load Data)	Loads data from local and HDFS files into Hive tables. It provides options for better performance through Hive partitioning and fewer data movements.	File System	Hive
IKM Hive Control Append	Integrates data into a Hive target table in truncate/insert (append) mode. Data can be controlled (validated). Invalid data is isolated in an error table and can be recycled.	Hive	Hive

KM Name	Description	Source	Target
IKM Hive Transform	Integrates data into a Hive target table after the data has been transformed by a customized script such as Perl or Python.	Hive	Hive
IKM File-Hive to Oracle (OLH)	Integrates data from an HDFS file or Hive source into an Oracle Database target using Oracle Loader for Hadoop.	File System or Hive	Oracle Database
CKM Hive	Validates data against constraints.	NA	Hive
RKM Hive	Reverse engineers Hive tables.	Hive Metadata	NA

9.2 Overview of Hands on Exercise

Previously in this workshop we have seen how to load data into HDFS using a cumbersome command line utility one for one. We viewed results from within HDFS also using a command line utility. Although this is fine for smaller jobs, it would be a good idea to integrate the moving of data with an ETL tool such as Oracle Data Integrator (ODI). In this exercise we will see how Oracle Data Integrator integrates seamlessly with Hadoop and more specifically Hive. We will proceed to use ODI in order to cleanse the credit- and mortgages-related customer data stored in Hive during a previous exercise. We will then load the data from Hive into the Oracle Database, also using ODI.

In this exercise you will:

1. Use ODI to reverse engineer a Hive Table
2. Use ODI to move data from a Hive table into the Oracle Database
3. Use ODI to perform transformation of data while it is being transferred
4. Use ODI to perform data cleansing using a check constraints

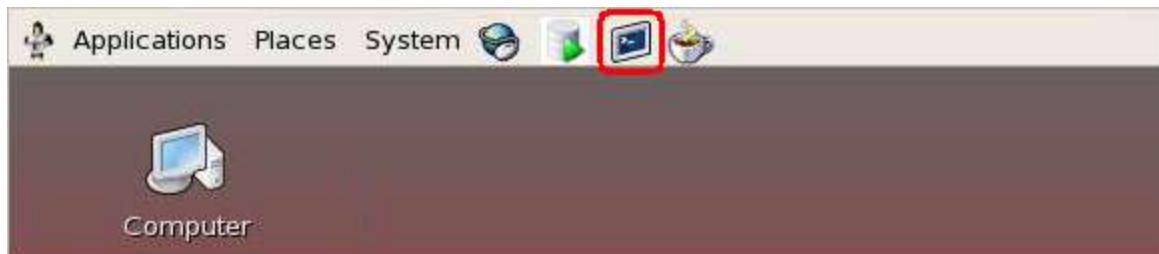
Notes:

To successfully go through this exercise, you have to go through the Hive Exercise first (see Chapter 6).

If you would like to skip this exercise, it is very important to still move the data into the Oracle Database in order for the last exercise of the workshop to work. Please go to Appendix A and run through those instructions.

9.3 Setup and Reverse Engineering in ODI

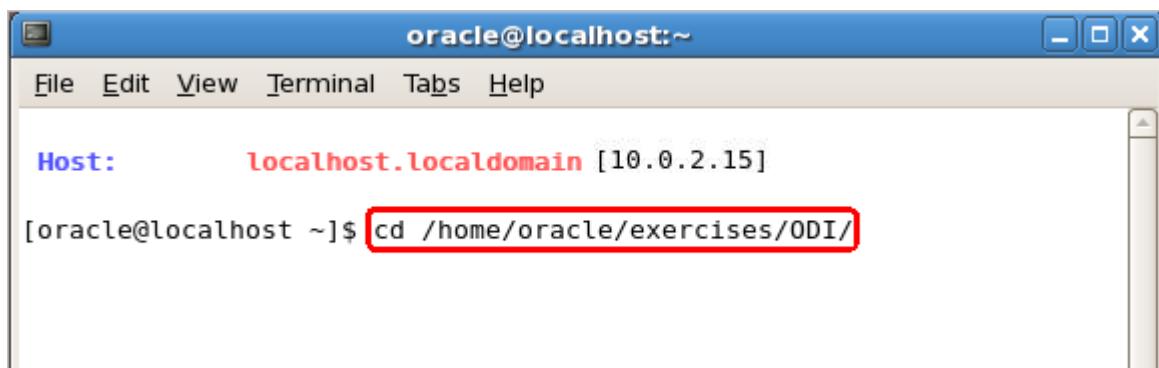
1. All of the setup for this exercise can be done from the terminal, hence open a terminal by double clicking on the **Terminal icon** on the desktop.



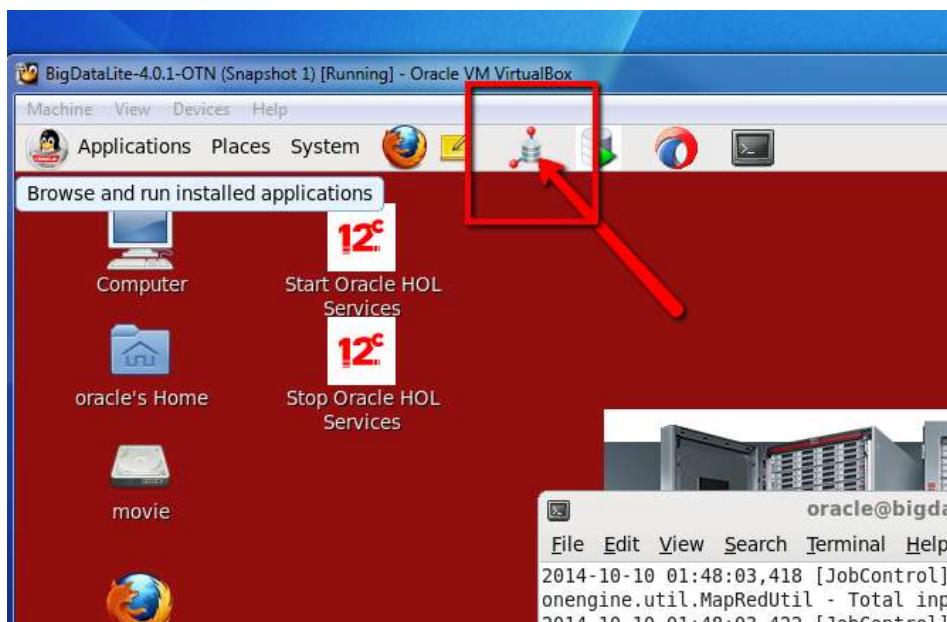
2. To get into the folder where the scripts for the first exercise are, type in the terminal:

```
cd /home/oracle/exercises/ODI
```

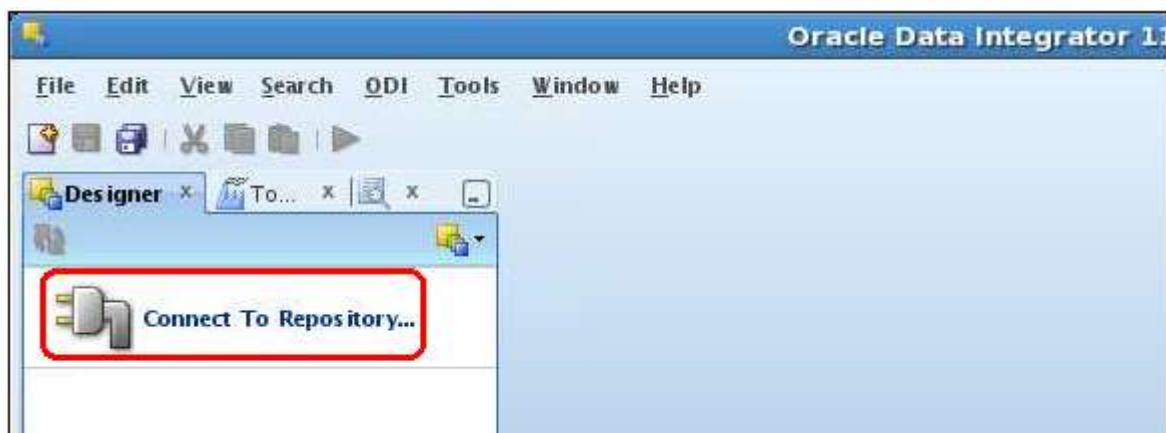
Then press **Enter**



3. Next we need to start Oracle Data Integrator. Click on the ODI icon ..



- Once ODI opens we need to connect to the repository. In the left upper corner of the screen under the Designer tab, click on the **Connect To Repository...**. The Wallet Password is “welcome1”.



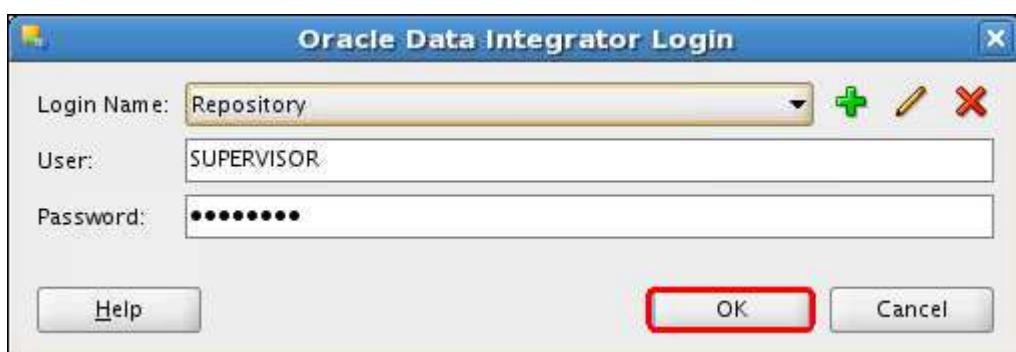
- In the dialog that pops up all of the connection details should already be configured.

Login Name: Repository

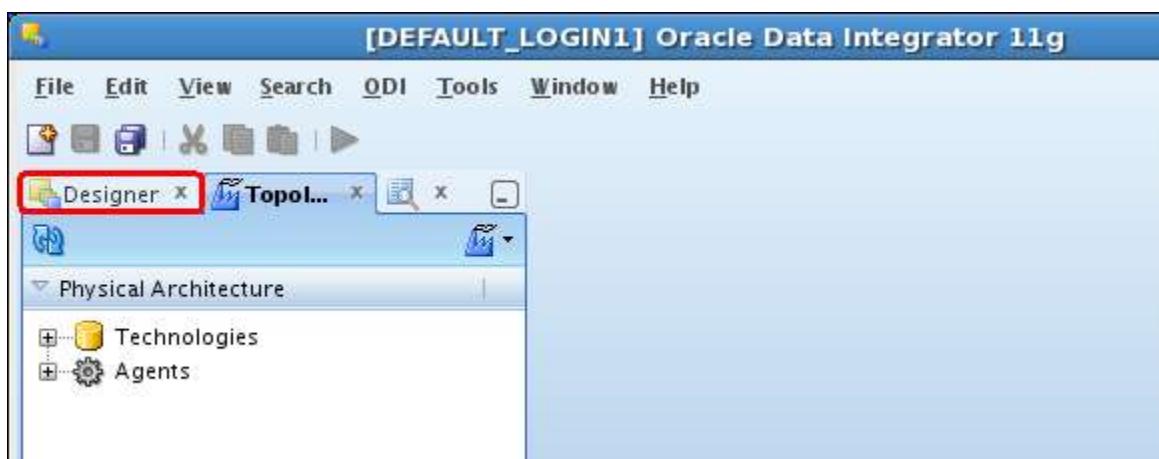
User: SUPERVISOR

Password: welcome1

If all of the data is entered correctly you can simple click **OK**



- Once you login make sure you are on the Designer Tab. Near the top of the screen on the left side click on **Designer**.



7. Near the bottom of the screen on the left side there is a **Models** tab, click on it.



You will notice that we have already created a Hive and Oracle model for you. These were pre-created as to reduce the number of steps in the exercise. For details on how to use Oracle database with ODI please see the excellent ODI tutorials offered by the *Oracle by Example Tutorials* found at <http://www.oracle.com/technetwork/tutorials/index.html>.

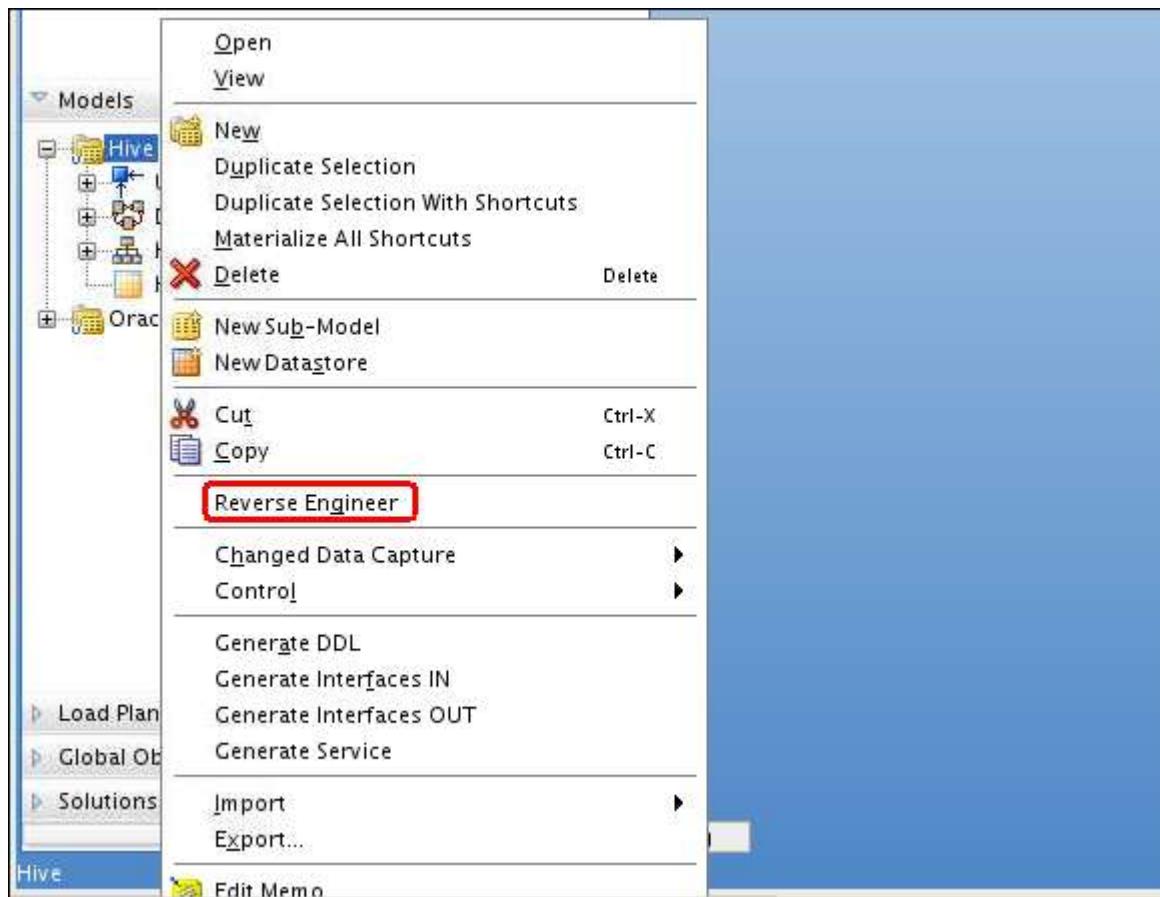
8. We can go ahead and expand out the HiveDefault Model to see what is there. Click on the + beside the Hive Model.



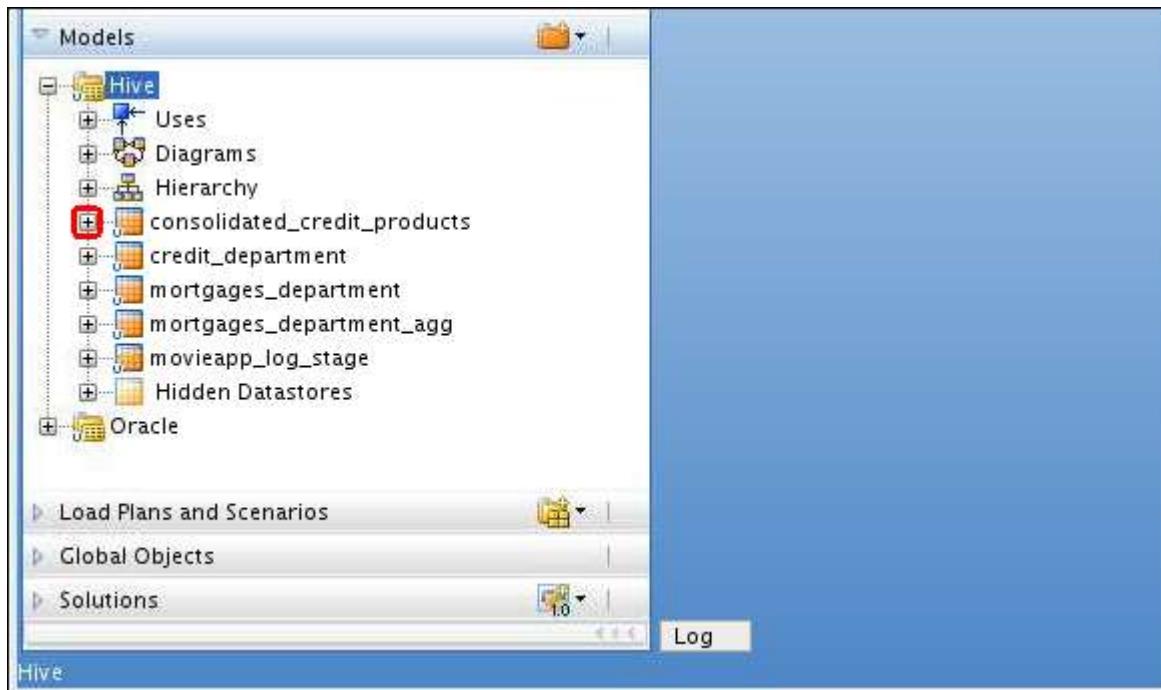
9. Let's use the RKM to discover all of the tables available in Hive.

Right click on the Hive model and select **Reverse Engineer**

Note: The Hive tables may already be displayed from an earlier “Reverse Engineer” action. You can of course run Reverse Engineering as many times as you like, as this action discovers any newly created Hive tables.



10. You will see several stores appear in the Hive Folder, one of which is called *consolidated_credit_products*. It is the table we created in the Hive exercise. You can click on the + beside *consolidated_credit_products* to see some more information.



11. You can also expand the **Attributes** folder to see all of the columns in this particular table, by clicking on the + beside Columns.

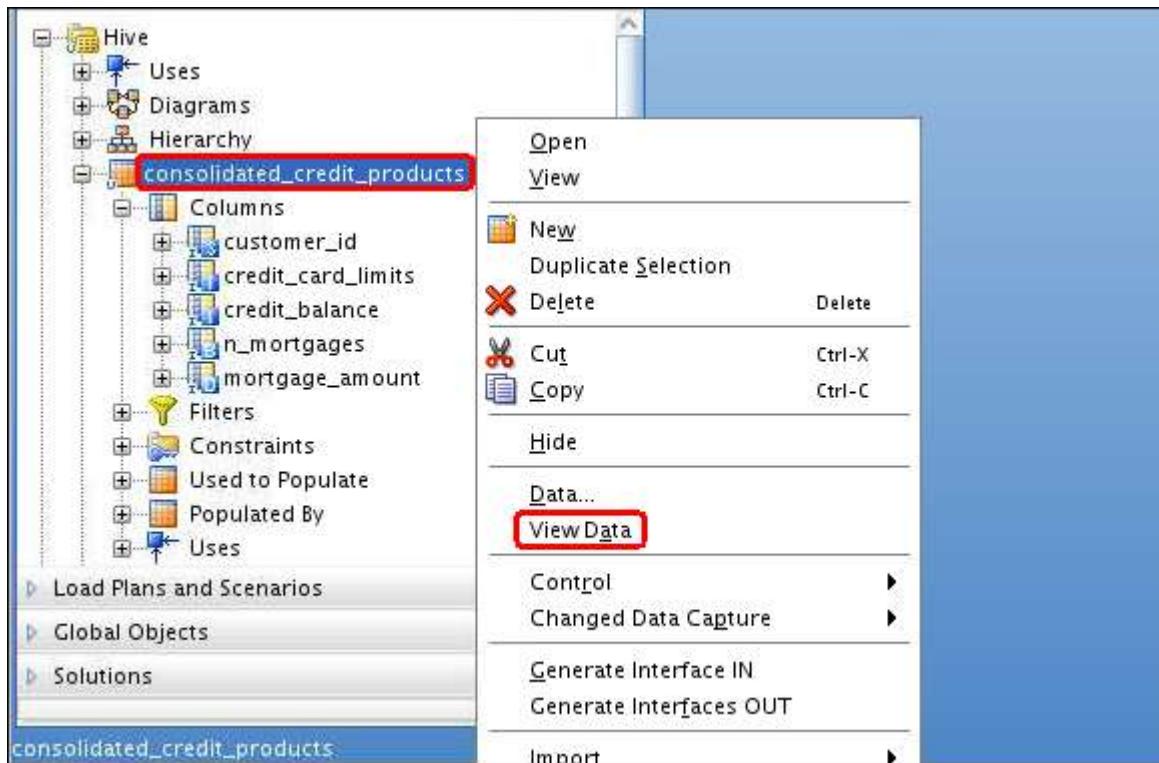


You will see the columns created in the Hive exercise displayed.



The information of all of the tables and their columns was not retrieved manually but auto discovered via the power of the Hive Reverse Engineering Knowledge Module (RKM) integrated in Oracle Data Integrator. Once you define a data store (in our case a hive source), the RKM will automatically discover all tables and their corresponding columns available at that source.

12. Let's go ahead and see the data that is present in our consolidated_credit_products table. Right click on the **consolidated_credit_products** table and select **View Data**



On the right side of the screen you will see a window open which shows the consolidated_credit_products Hive table. If we examine the data closely we will notice that there are some data quality issues. We have negative mortgage amounts which we must eliminate from the data, and we have people with mortgages of 0 dollars (a system glitch as a single mortgage of 0 dollars is actually 0 mortgages).

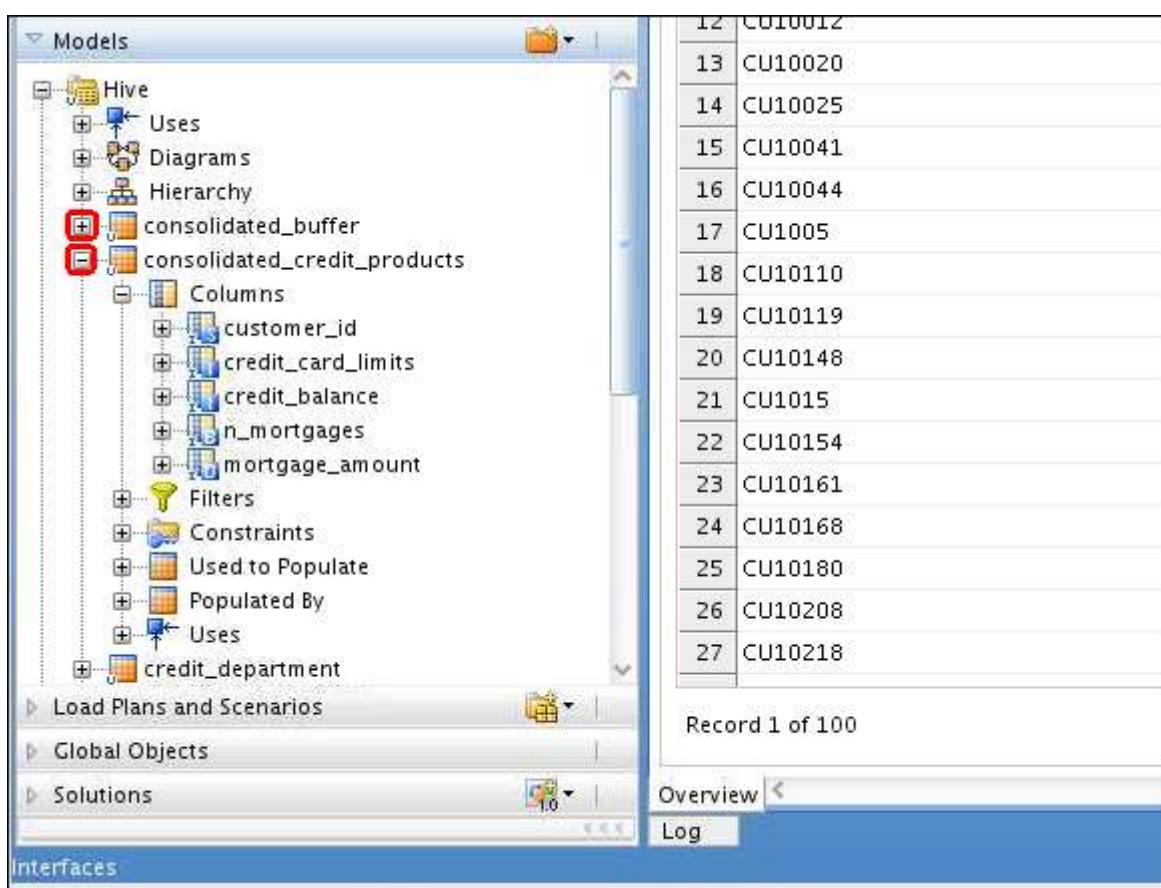
8	CU0008	8000	0	1	-8000.0
9	CU100	1000	0	1	2000.0
10	CU10006	1000	0	1	2000.0
11	CU10011	4000	0	1	7250.0
12	CU10012	700	0	1	1001.0
13	CU10020	1500	0	1	2500.0
14	CU10025	1500	0	1	1840.0
15	CU10041	1500	0	1	700.0
16	CU10044	800	0	1	15000.0
17	CU1005	2500	0	1	1200.0
18	CU10110	1500	0	1	1450.0
19	CU10119	1500	0	1	0.0
20	CU10148	1000	0	1	450.0
21	CU1015	900	0	1	826.0

Let's go ahead and fix these issues while we load the data into the Oracle Database, all done through the ODI interface. As we want to explore more of the features available with ODI we will do

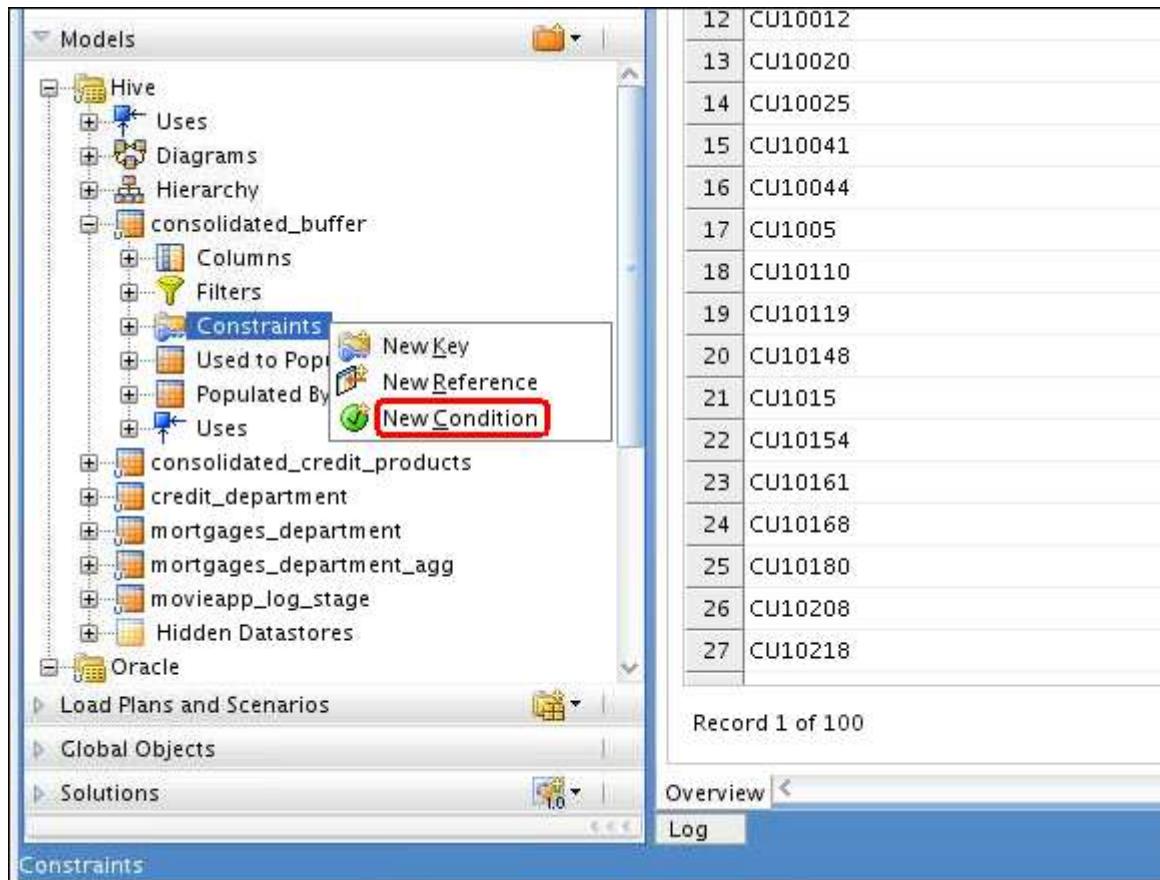
the loading of the data in two stages. First we will move the data into another Hive table, then move that data into the Oracle Database.

9.4 Using ODI to move data from Hive to Hive

1. To start the lesson – run the setup.sh script in the ODI folder to create the empty hive table consolidated_buffer.
2. Before we start moving the data, let's first add some constraints on our target Hive table. Using the feature we can see how data can be cleansed while it is being moved from source to target. To start creating constraints let's minimize the consolidated_credit_products Hive table and maximize the consolidated_buffer table, to give us room to work. Click on the – beside consolidated_credit_products and the + beside consolidated_buffer.



3. Now right click on the Constraints folder (in the consolidated_buffer table) and select **New Condition**.



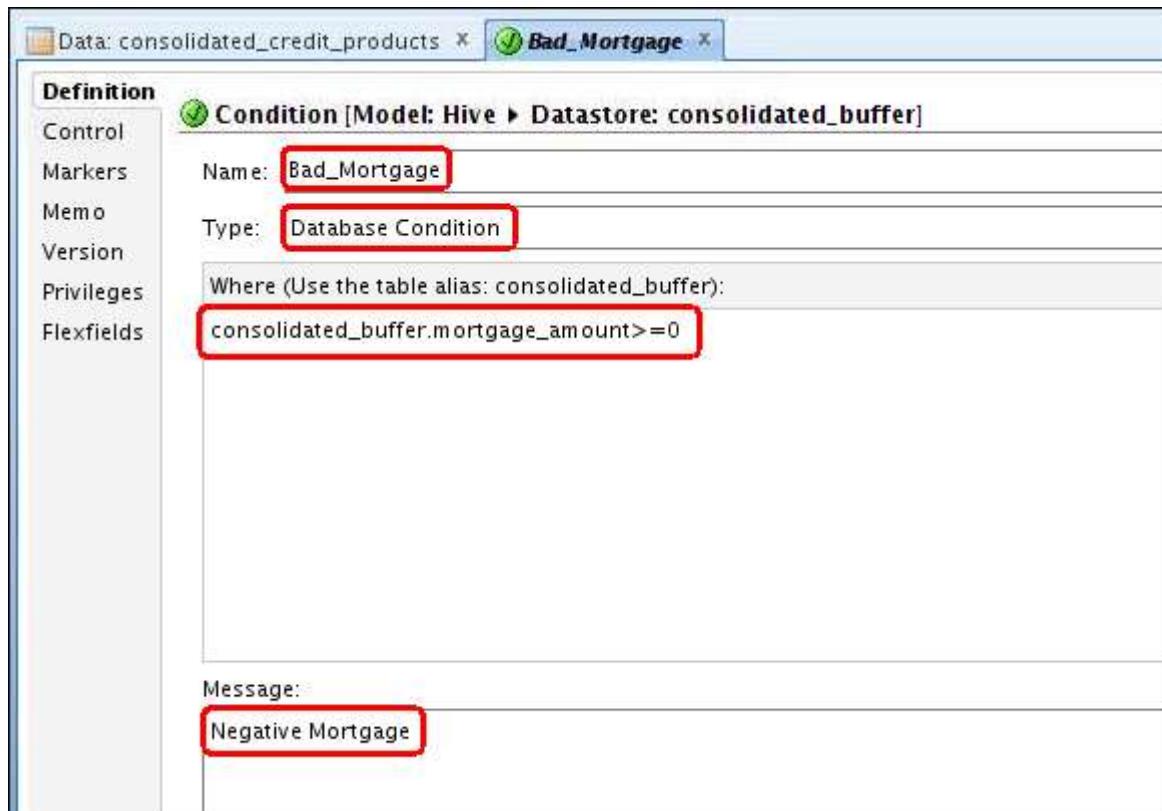
4. In the new window that opens up enter the following information

Name: Bad_Mortgage

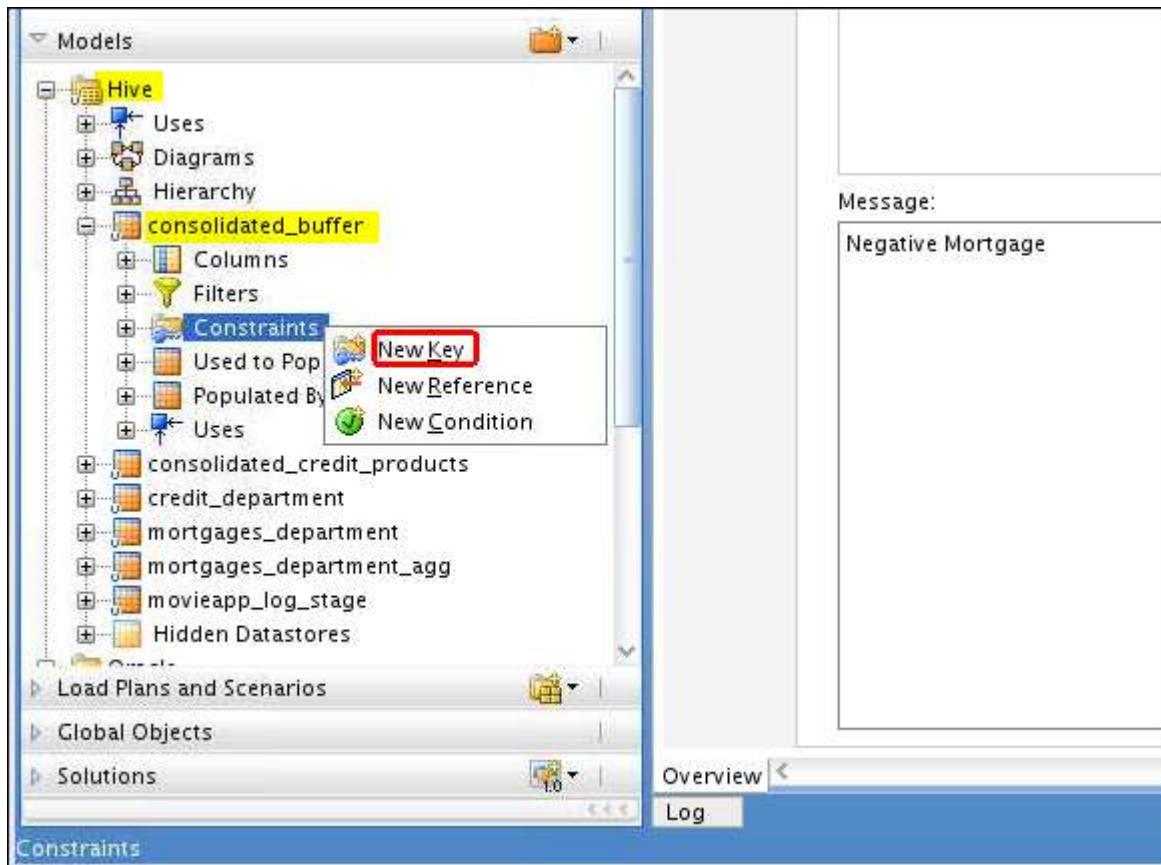
Type: Database Condition

Where: consolidated_buffer.mortgage_amount>=0

Message: Negative Mortgage



5. To enable flow control (the process by which data in movement is cleansed) we will also need a primary key on the target table. Let's add a primary key now. In the left bottom corner in the Models section right click on Constraints and select **New Key**



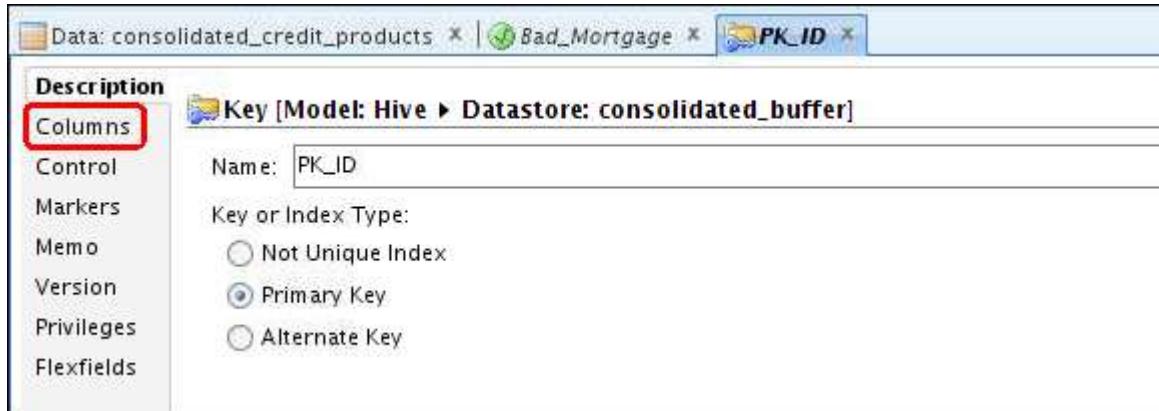
6. In the window that opens up on the right write in the following information

Name: **PK_ID**

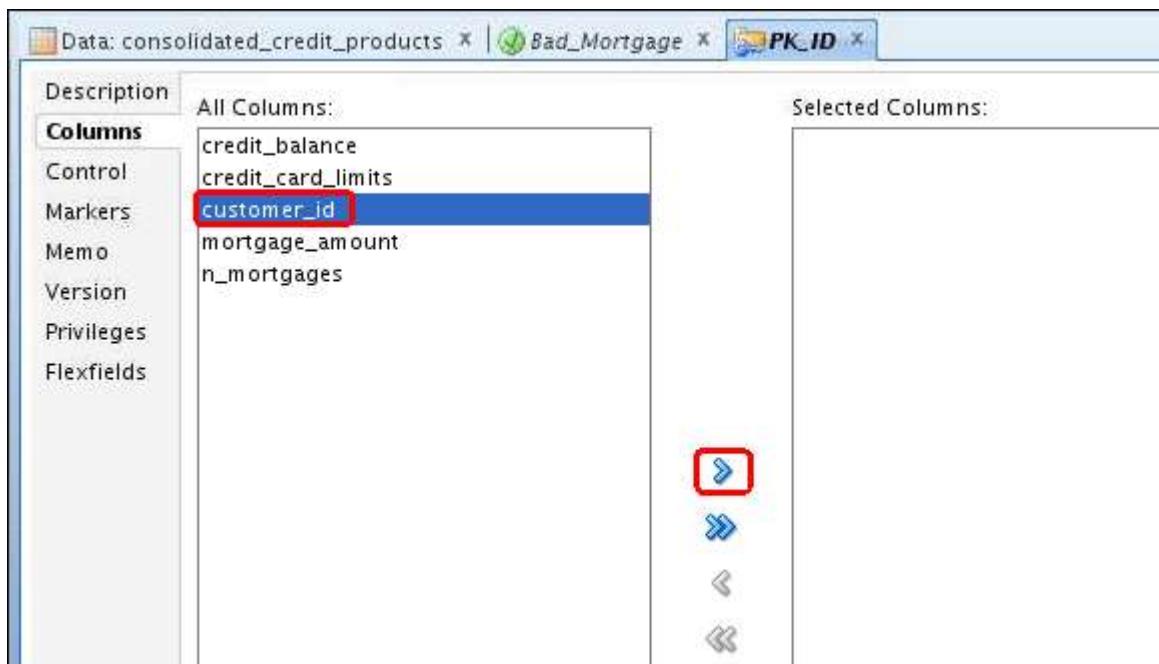
Key or Index Type: **Primary Key**



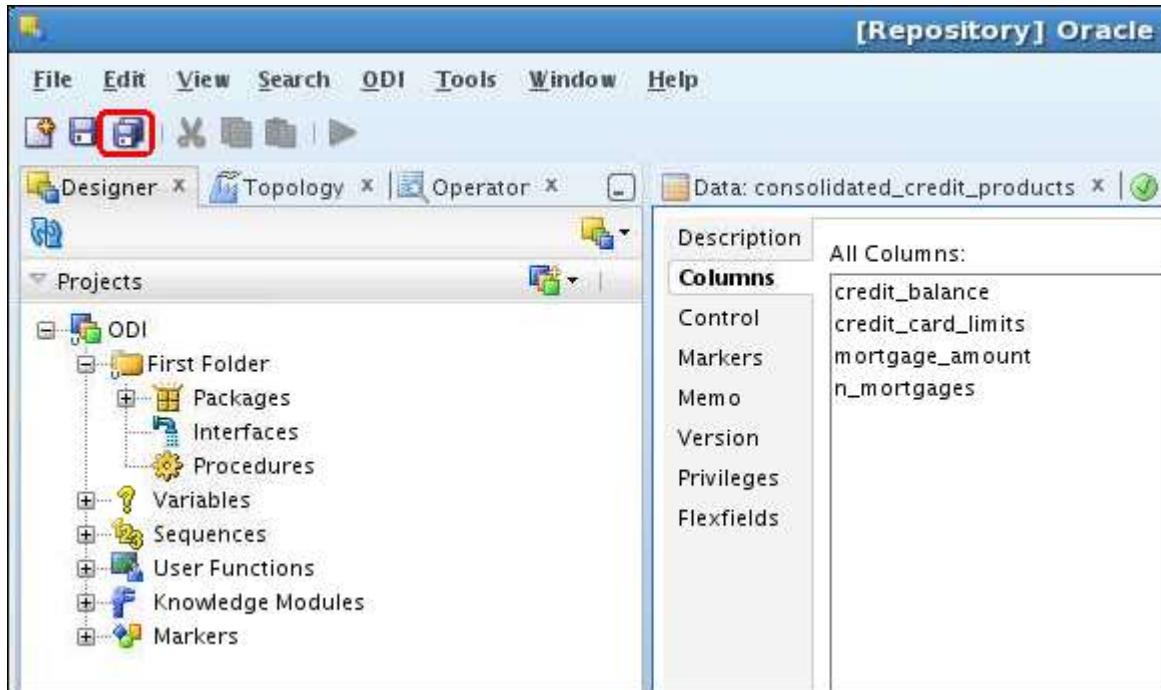
7. Next move to the **Columns** Tab to select the actual primary key column.



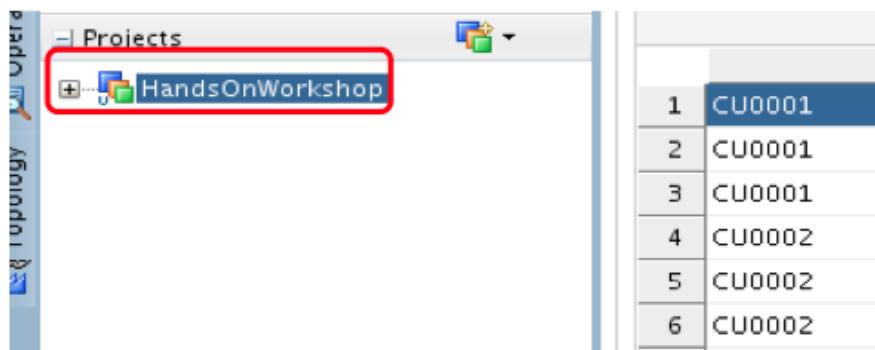
8. In the columns table select the **customer ID** column and click the > symbol to move it into the selected columns dialog.



9. We have now successfully created both the primary key and a conditional constraint on our target Hive table. We can now go ahead and save all of the changes we have made. Go to the upper right hand corner of the screen and click the **Save All** button.



10. Now we can start setting up the movement of the data. To start the process of moving data from a Hive table into another Hive table we must create an interface for this process. In the left upper part of the screen in the projects section there is a project called HandsOnWorkshop. Let's expand it out. Click on the + next to HandsOnWorkshop.

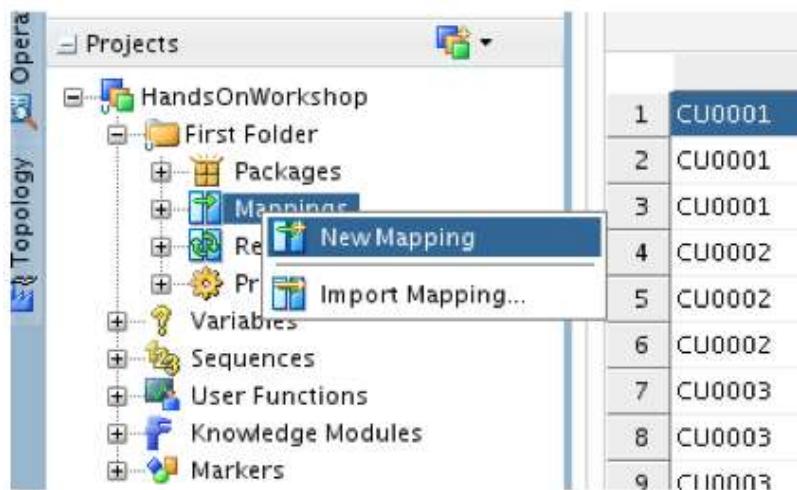


1	CU0001
2	CU0001
3	CU0001
4	CU0002
5	CU0002
6	CU0002

11. We will also need to expand out the folder called First Folder. Click on the + next to it.

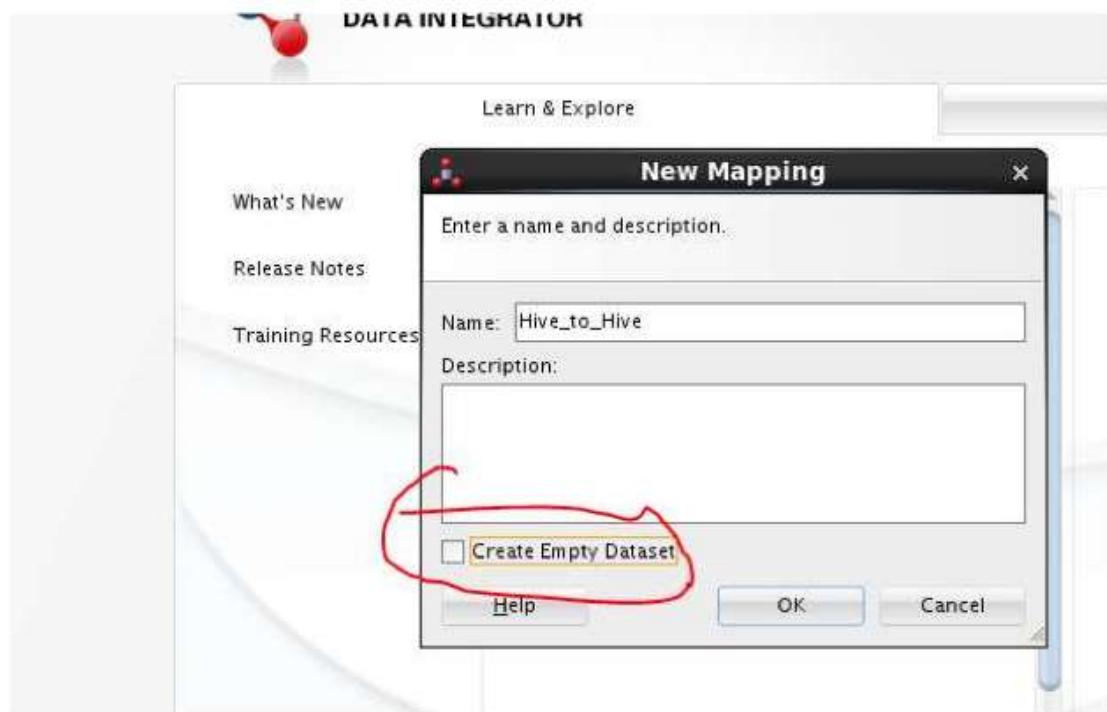


12. In this folder we can create a new mapping. Right click on Mappings and select **New Mapping**.

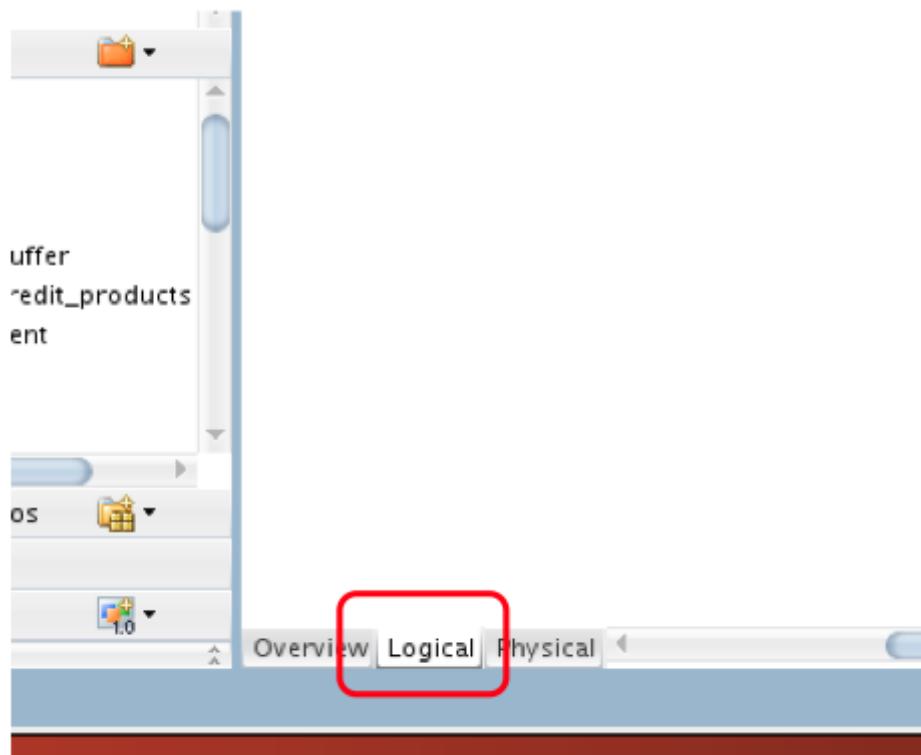


13. In the window that appears on the right we will now create all of the elements that make up the interface. The first thing we need to do is to name the interface, hence you need to type the following information

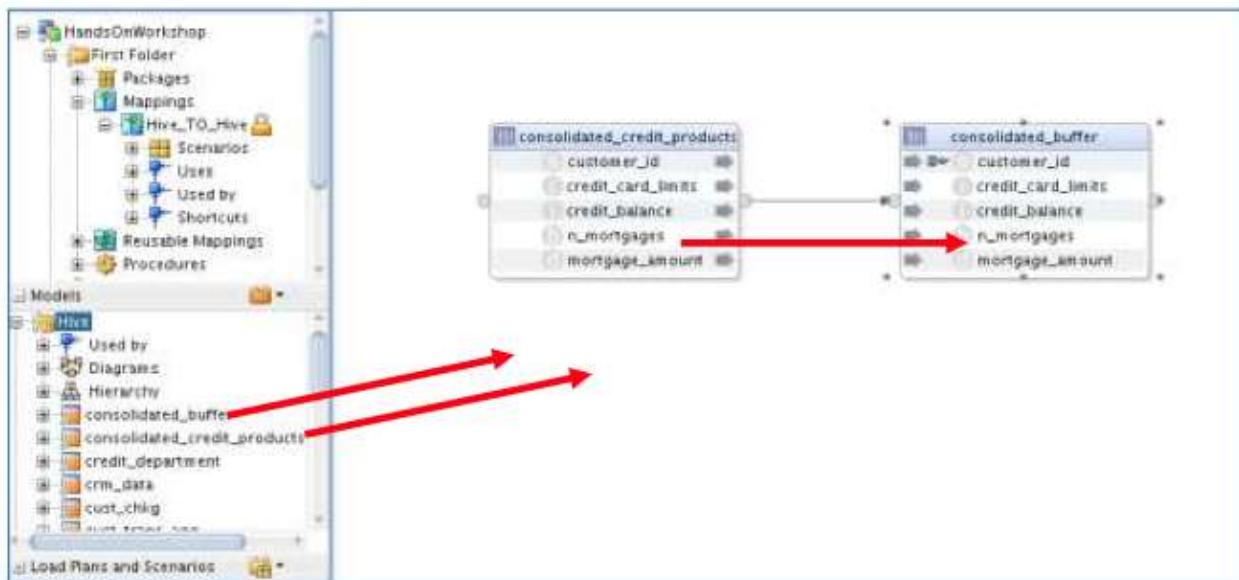
Name: **Hive_To_Hive**



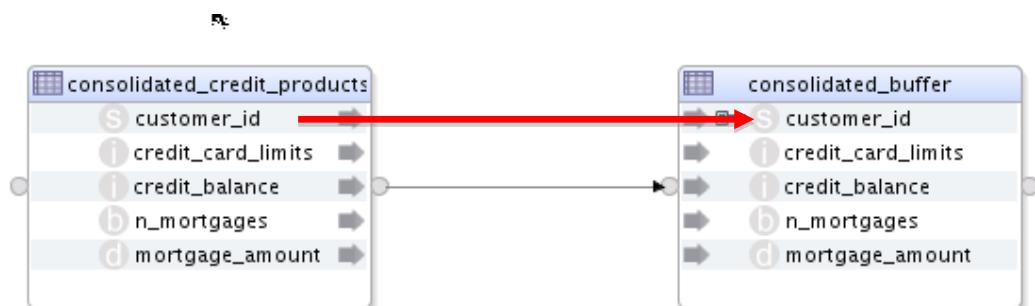
14. Next, we will go to the Logical Tab to create the design. At the bottom of the screen, click the "Logical" Tab.



15. We now need to drag and drop the source and target table into the interface. Drag the consolidated_credit_products into the source tables section in the middle. Also drag the consolidated_buffer table into the target area on the top right.

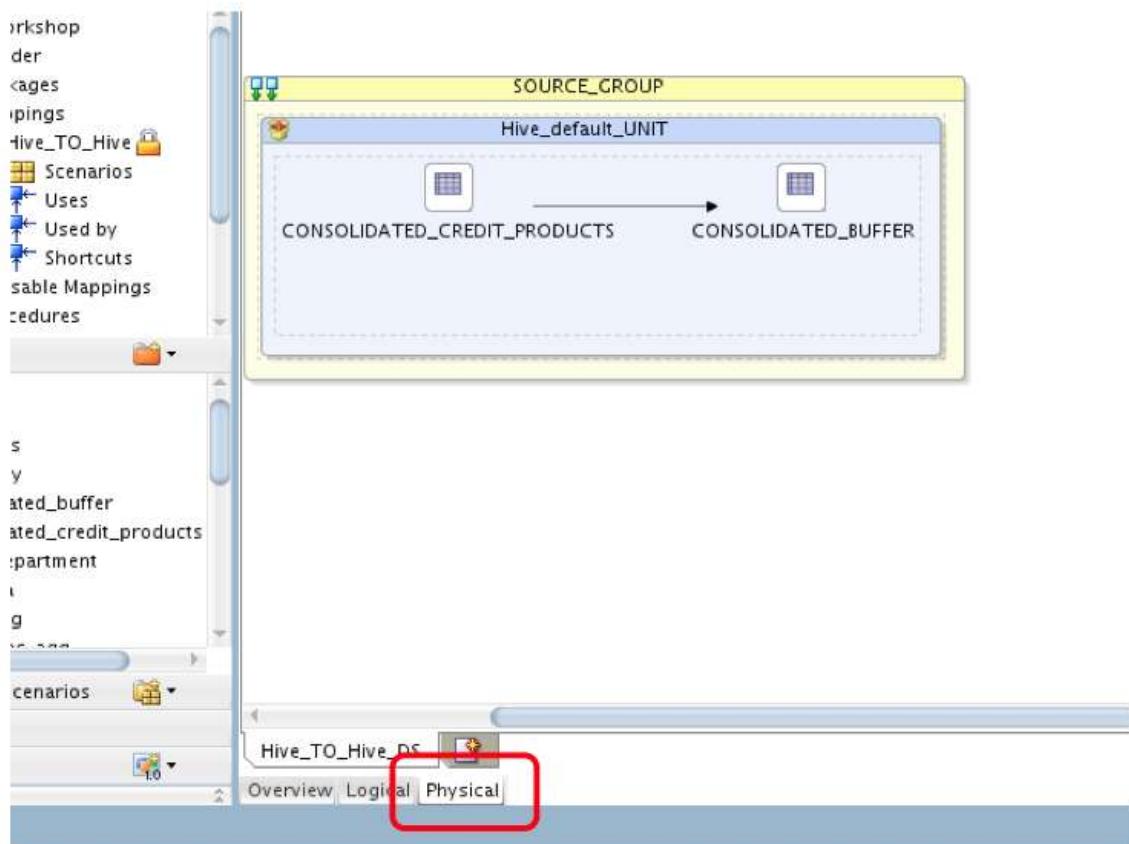


16. Now, you must manually map the fields from the consolidated_credit_products Hive table to the consolidated_buffer table. To do this simply click-and-drag the customer_id column from the consolidated_credit_products table over to the customer_id in the consolidated_buffer table. Repeat this for each of the columns in the consolidated_credit_products table.

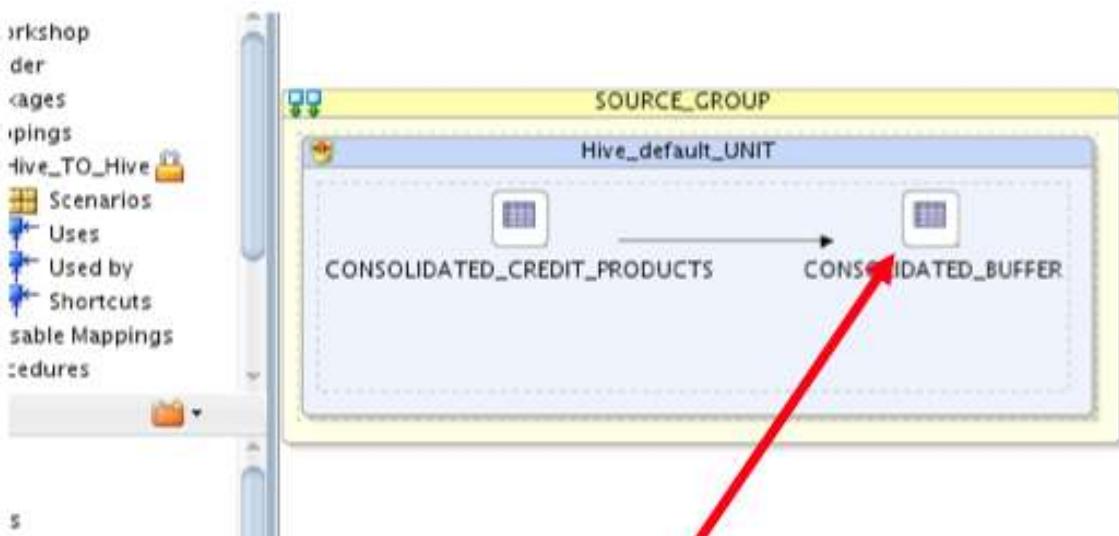


17.

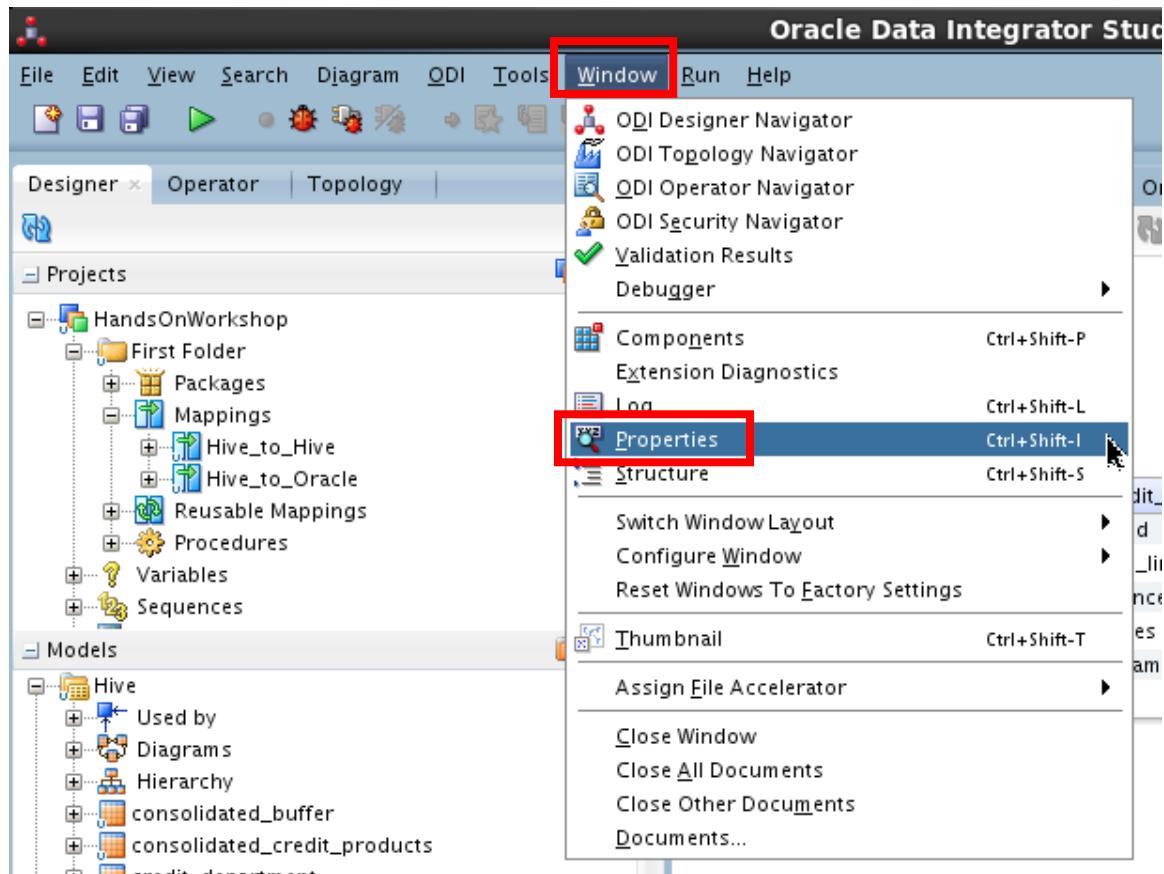
18. Now let's go to the Physical Tab.



19. Click on the target table (consolidated_buffer) to select it.



20. Abc

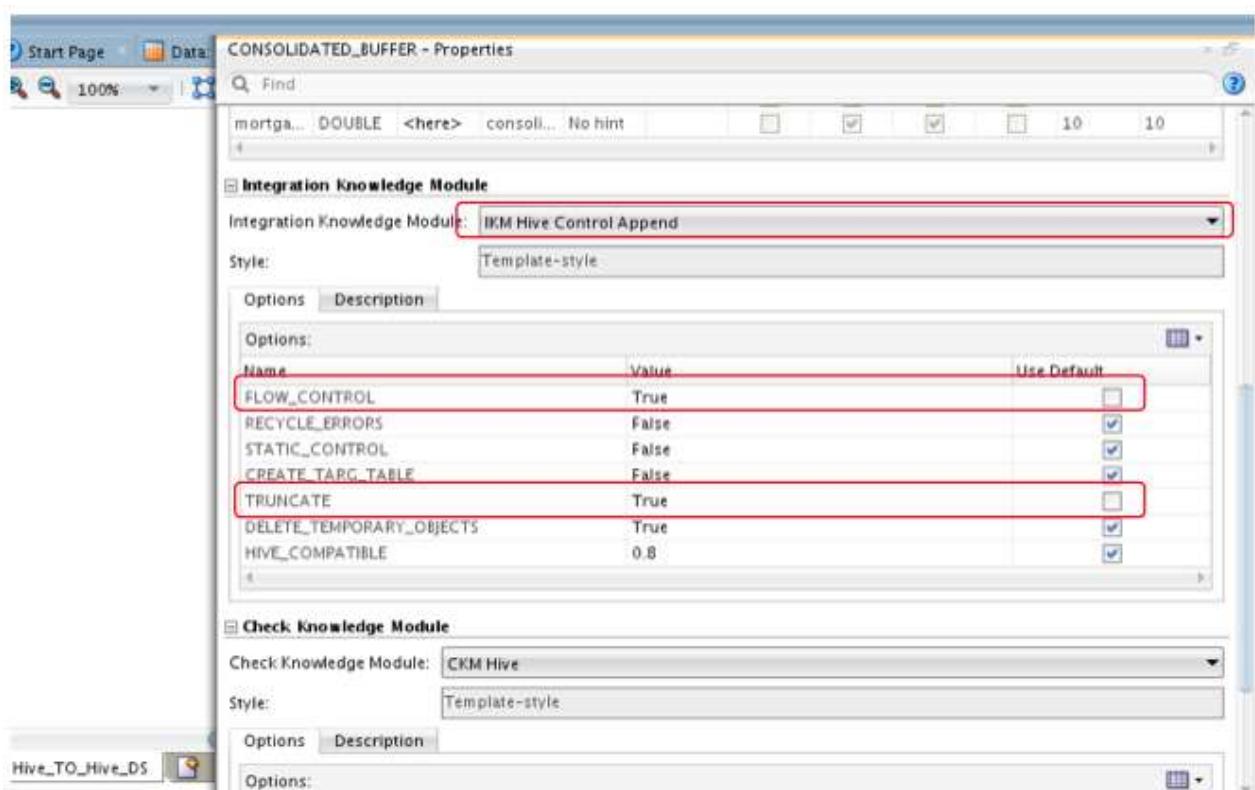


21. In the Properties Window under Integration Knowledge Module, select the following options

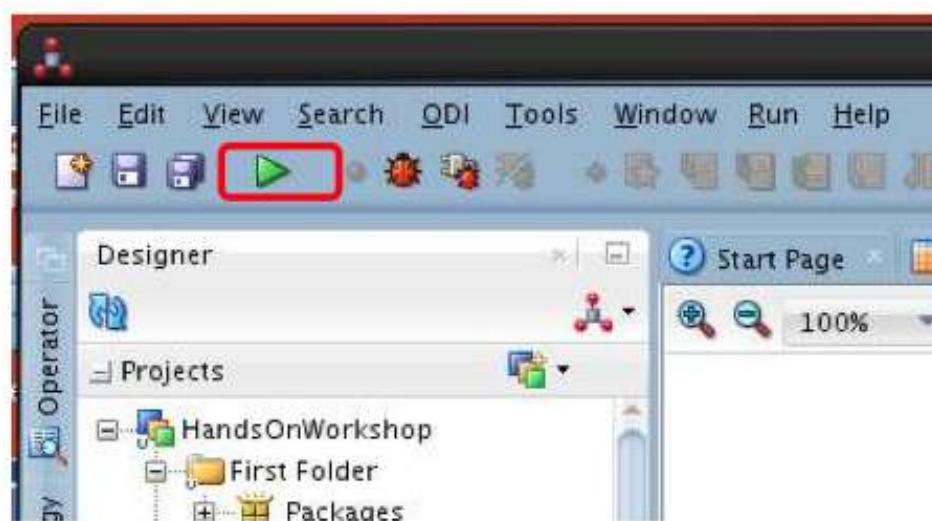
IKM Selector: IKM Hive Control Append

FLOW_CONTROL: True

TRUNCATE: True



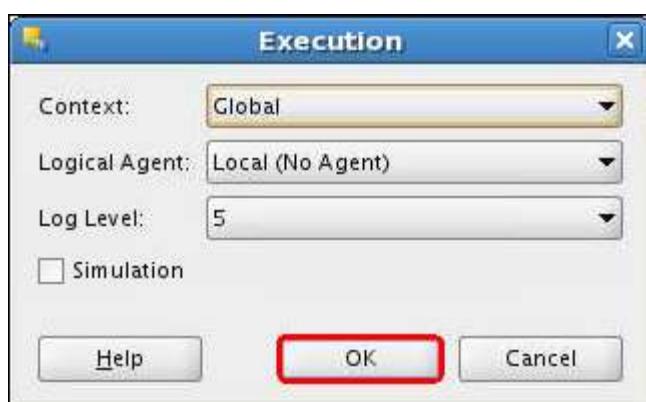
22. Let's go ahead and run this interface. To run the interface go to the left upper corner of the screen and click the **execute** button



23. A window will pop asking you if you want to save your changes. Click Yes



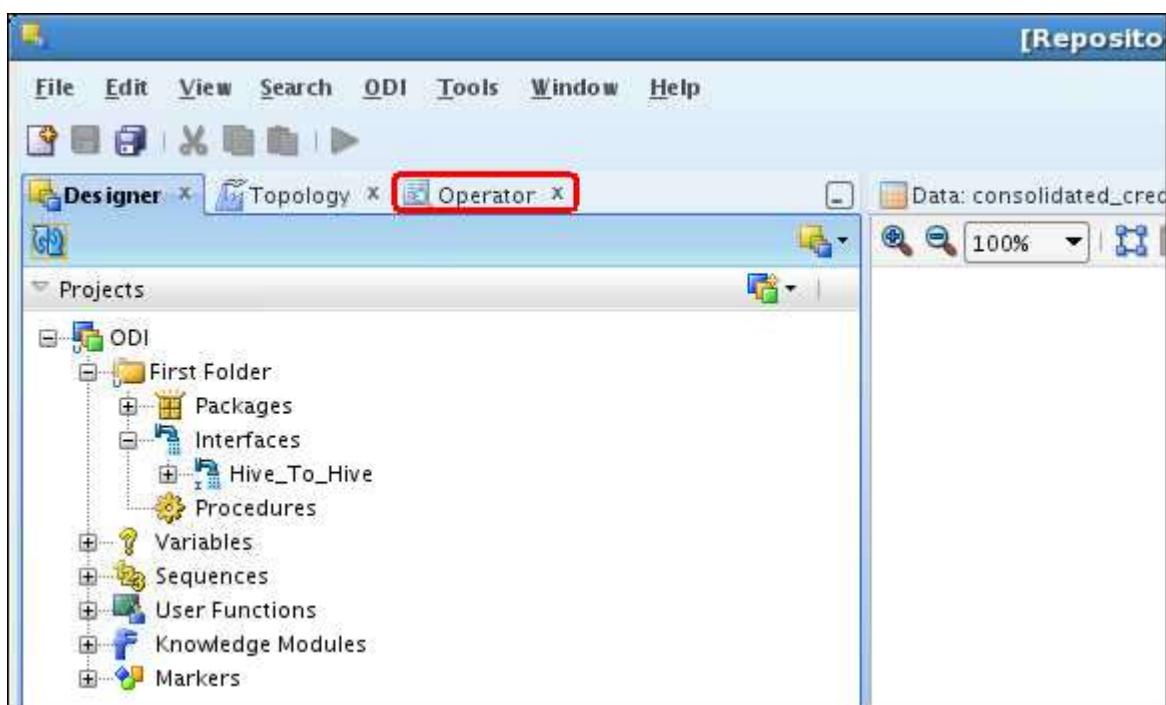
24. A window will pop asking you to configure the Execution Context and Agent. The default values are fine just click OK



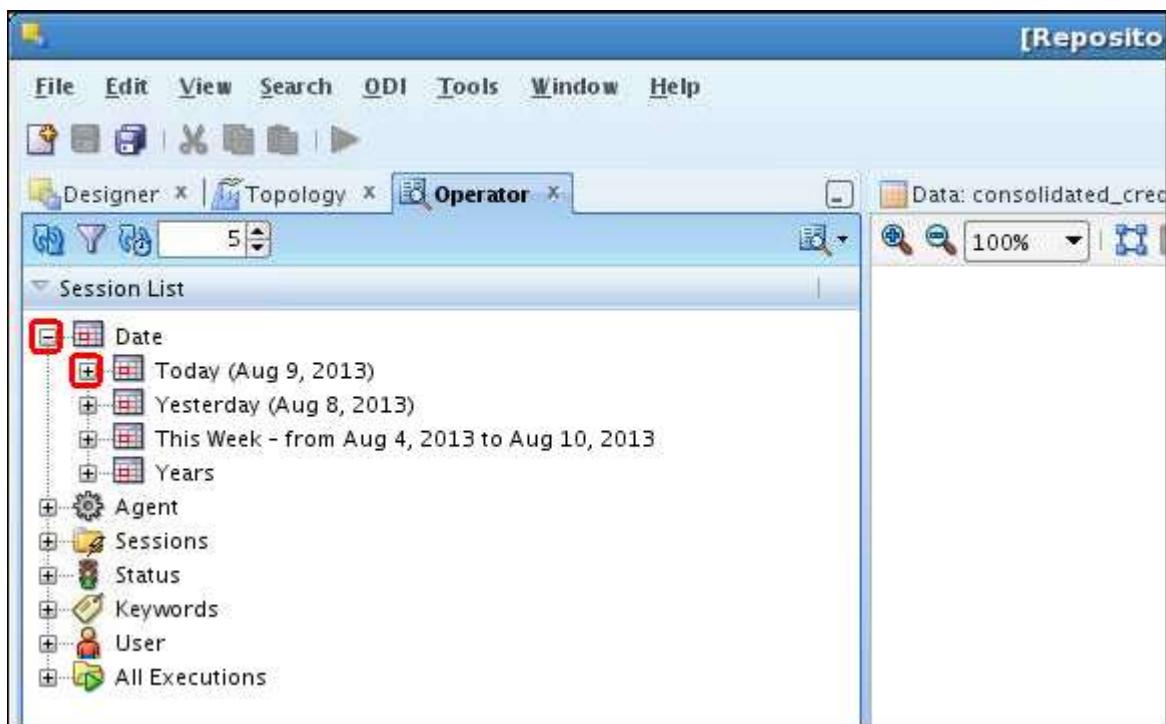
25. A final window will pop up tell you the session has started. Click **OK**



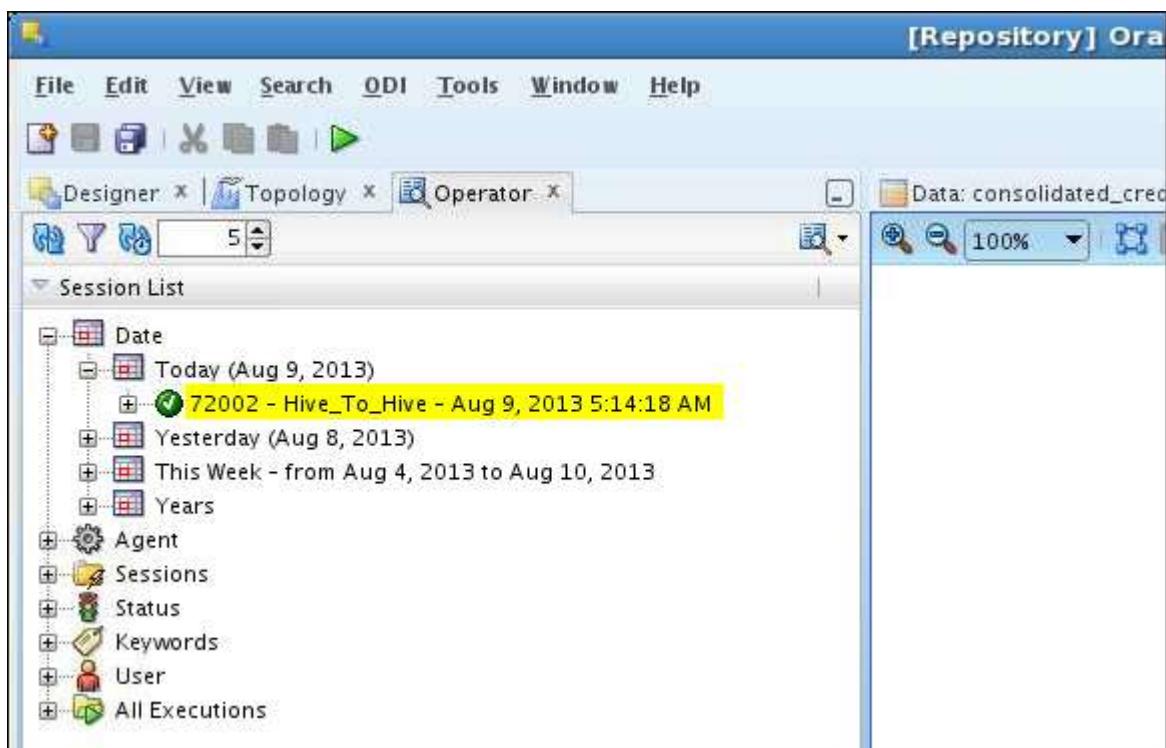
26. Let's now go to the *Operator* tab to check if the execution was successful. In the top left corner of the screen click on the **Operator** tab



27. When you get to the Operator Tab you will need to expand out the Date and Today session List by clicking on the + beside both of them.

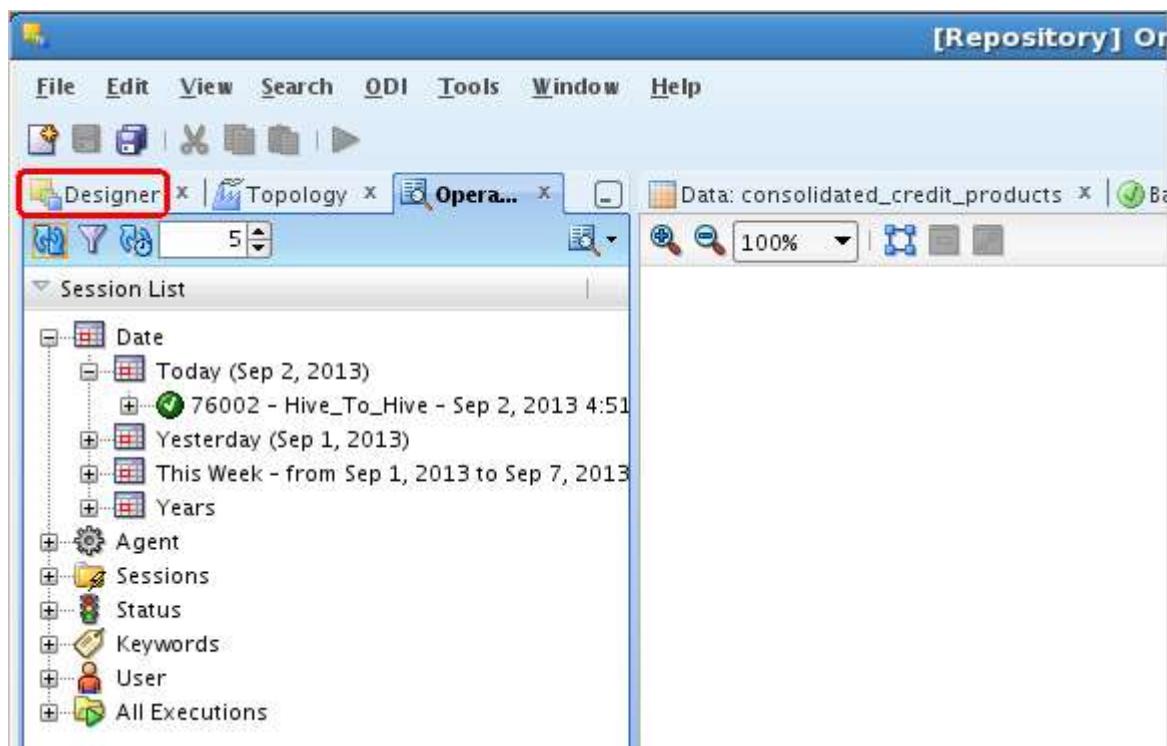


28. You will probably see a lightning bolt beside the Hive_To_Hive execution. This means integration is executing, so wait for a little bit until the checkmark appears.

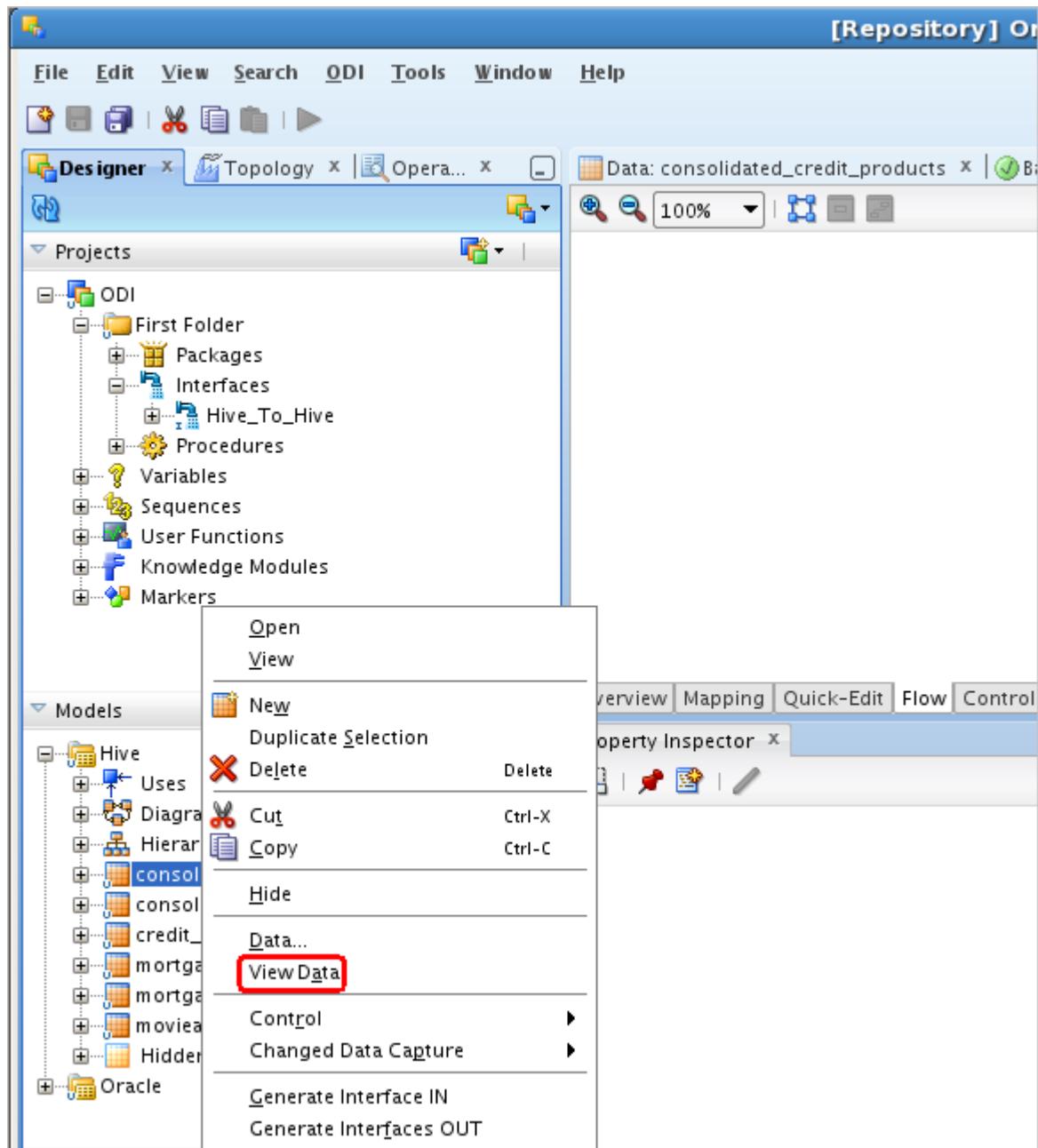


The movement of data from Hive to Hive has completed successfully. Now let's check whether our data cleansing tools did what we told them to.

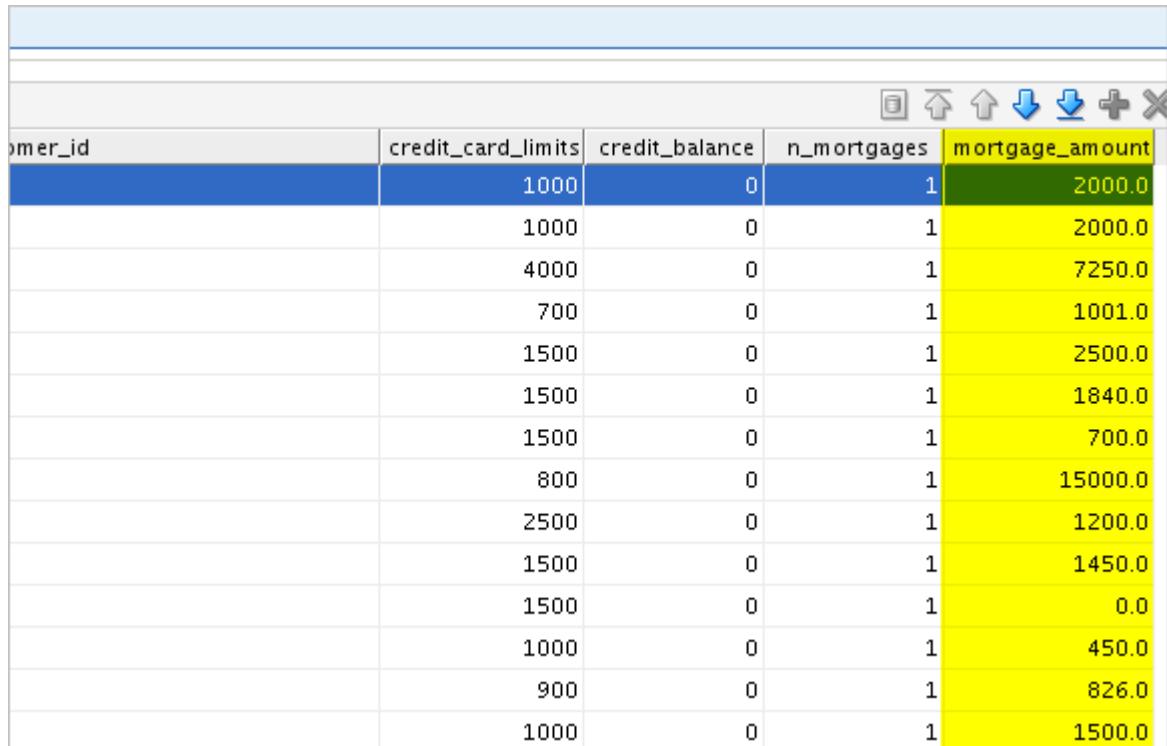
29. First let's see what we've loaded into the consolidated_buffer Hive table. Go back to the **Designer** tab.



30. Under Models, in the left bottom half, right click on consolidated_buffer and click on **View data**

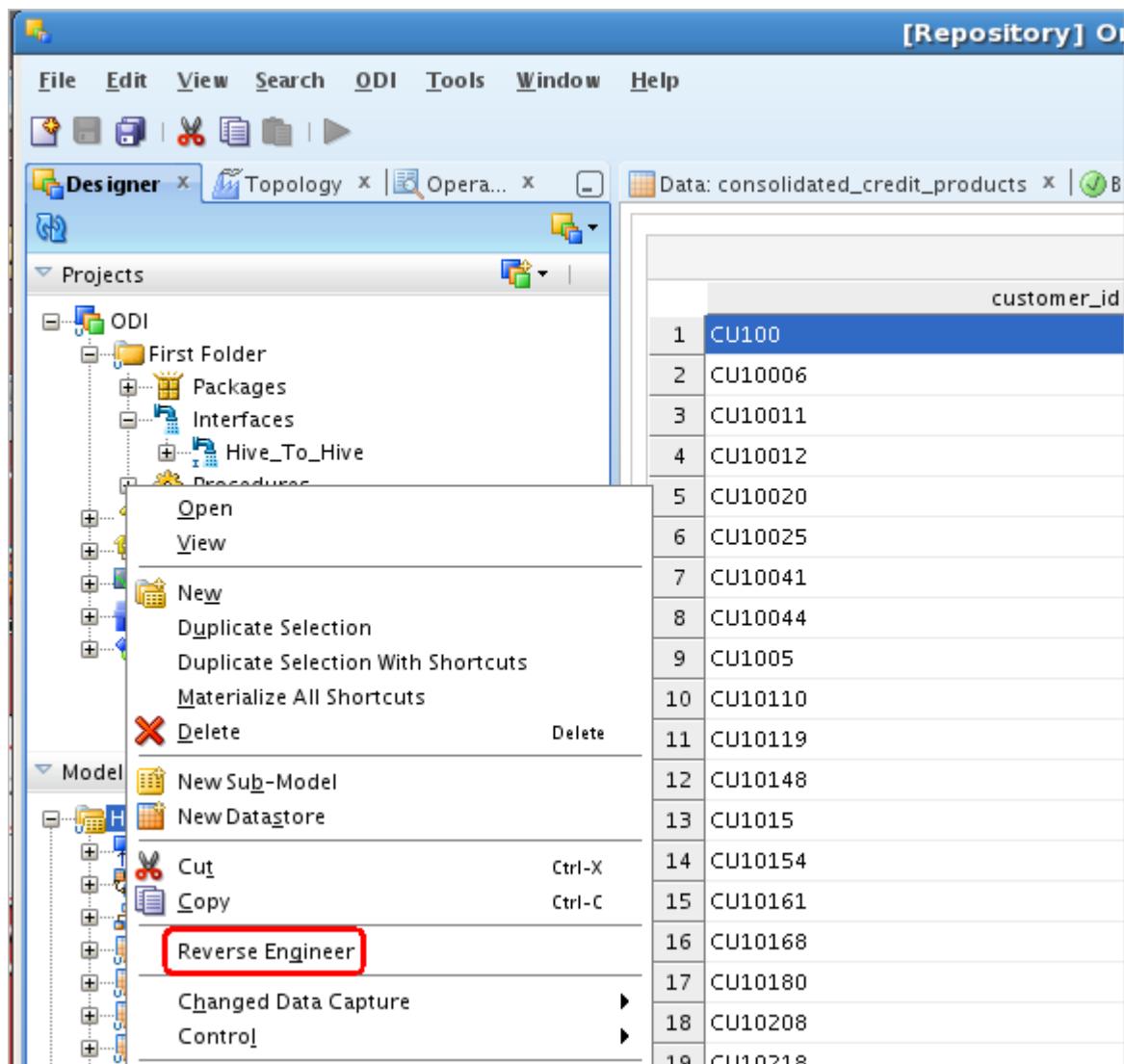


As you can see there are no more mortgages with a negative value in the mortgage_amount column (highlighted in yellow).



customer_id	credit_card_limits	credit_balance	n_mortgages	mortgage_amount
	1000	0	1	2000.0
	1000	0	1	2000.0
	4000	0	1	7250.0
	700	0	1	1001.0
	1500	0	1	2500.0
	1500	0	1	1840.0
	1500	0	1	700.0
	800	0	1	15000.0
	2500	0	1	1200.0
	1500	0	1	1450.0
	1500	0	1	0.0
	1000	0	1	450.0
	900	0	1	826.0
	1000	0	1	1500.0

31. You may now ask, what actually happened to the invalid records that didn't satisfy our previously created condition. We did not just lose them. As mentioned earlier on in the Knowledge Modules explanation, the Hive Control Append KM does store invalid data in an error table. So let's have a look. The Models don't show any more tables, so first let's refresh the view; right click on Hive in the Models view and click **Reverse Engineer**

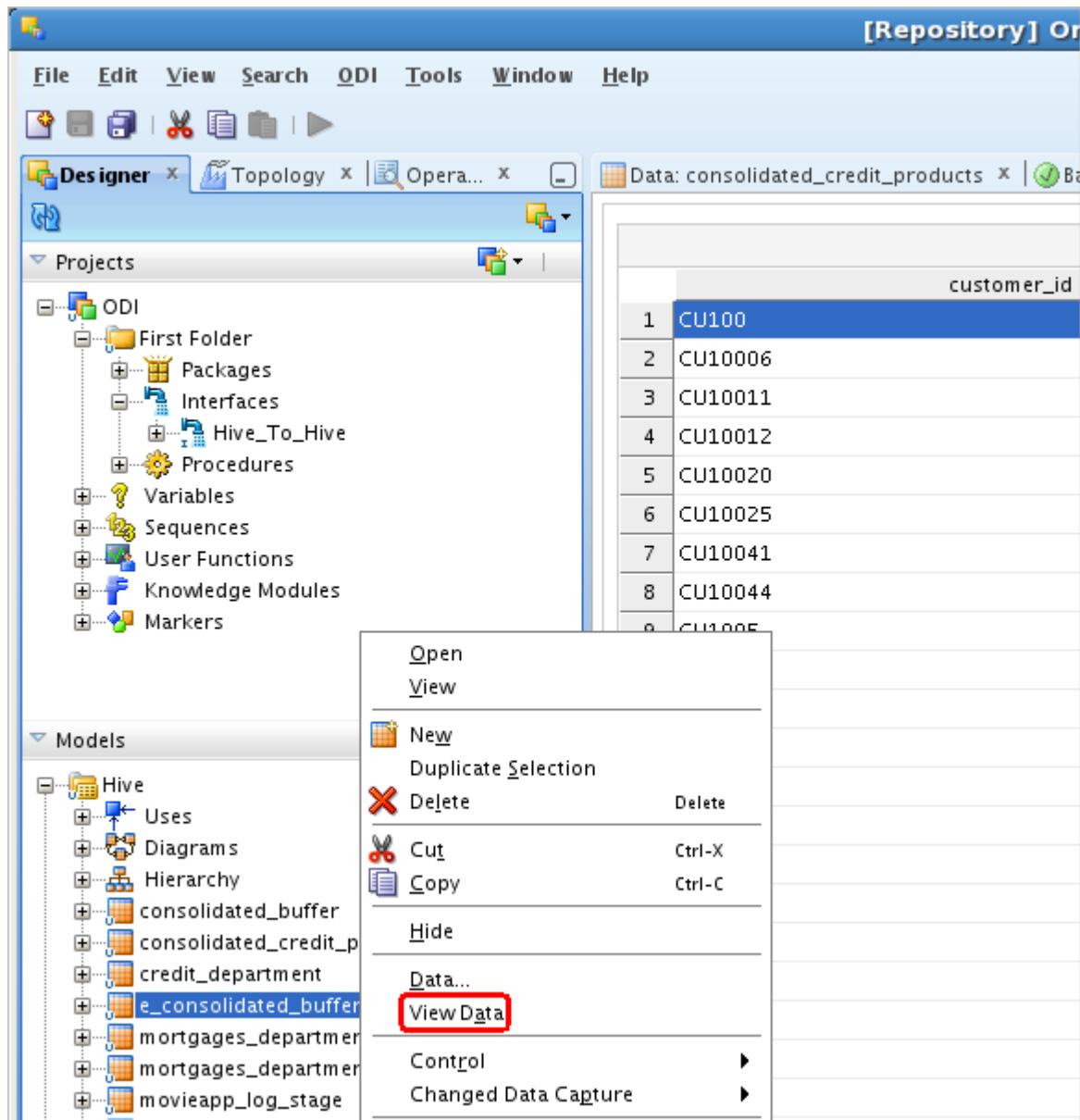


You will now notice another table showing up, called "e_consolidated_buffer".

The screenshot shows the Oracle Big Data Workshop interface. On the left, the 'Models' browser displays a tree structure of objects under the 'Hive' category. The 'e_consolidated_buffer' object is highlighted with a yellow selection bar. Other visible objects include 'consolidated_credit_products', 'credit_department', 'mortgages_department', 'mortgages_department_agg', 'movieapp_log_stage', and 'Hidden Datastores'. Below the browser are three expandable sections: 'Load Plans and Scenarios', 'Global Objects', and 'Solutions'. The main workspace on the right contains a data grid with two columns. The first column is numerical (12 to 23) and the second column contains strings starting with 'CU10...'. A status bar at the bottom indicates 'Record 1 of 100'. Navigation buttons for 'Overview' and 'Log' are also present.

12	CU10148
13	CU1015
14	CU10154
15	CU10161
16	CU10168
17	CU10180
18	CU10208
19	CU10218
20	CU10228
21	CU10236
22	CU10303
23	CU10308

32. Right click on “e Consolidated Buffer” which is our error table for consolidated_buffer and click **View Data**



You can now see the 8 invalid rows that didn't satisfy our condition. Notice not only the "mortgage_amount" column but also the "odi_err_mess" column with our "Negative Mortgage" value previously defined in the Condition Message.

r_type	odi_err_mess	odi_check_date	customer_id	credit_car...	credit...	n_mortg...	mortgage_amount
	Negative Mortgage	1378111978	CU0001	1000	0	1	-1000.0
	Negative Mortgage	1378111978	CU0002	2000	0	1	-2000.0
	Negative Mortgage	1378111978	CU0003	3000	0	1	-3000.0
	Negative Mortgage	1378111978	CU0004	4000	0	1	-4000.0
	Negative Mortgage	1378111978	CU0005	5000	0	1	-5000.0
	Negative Mortgage	1378111978	CU0006	6000	0	1	-6000.0
	Negative Mortgage	1378111978	CU0007	7000	0	1	-7000.0
	Negative Mortgage	1378111978	CU0008	8000	0	1	-8000.0

9.5 Summary

In this exercise you were introduced to Oracle's integration of ODI with Hadoop. It is worthy to note that the integration of an ETL tool with Hadoop and an RDBMS is only available for the Oracle Database and only available from Oracle. It is a custom extension for ODI developed by Oracle to allow users who already have ETL as part of the Data Warehousing methodologies to continue using the same tools and procedures with the new Hadoop technologies.

It is quite important to note that ODI is a very powerful ETL tool which can offer all of the functionality typically found in an enterprise quality ETL. Although the examples given in this exercise are fairly simple, it does not mean that the integration of ODI and Hadoop is. All of the power and functionality of ODI is available when working with Hadoop. Workflow definition, complex transforms, flow control, multi source, just to name a few of the functionalities of ODI and inherently feature that can be used with Hadoop.

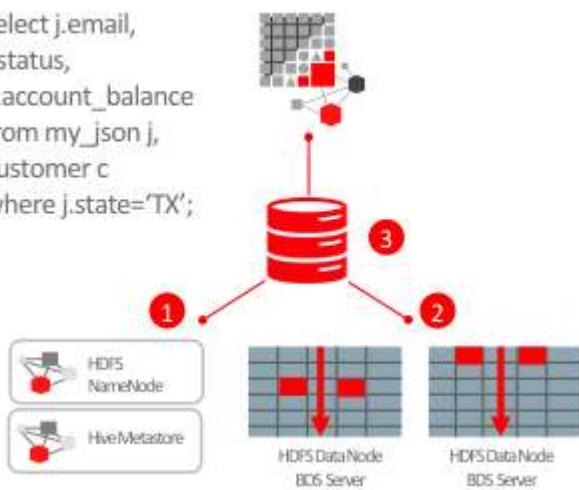
It is now time to move the valid data into the Oracle Database.

10. ORACLE BIG DATA SQL

10.1 Introduction to Oracle Big Data SQL

Oracle Big Data SQL delivers the flexibility of OSCH and the performance of Impala, and it does it specifically through the approach of unifying metadata and optimizing performance by limiting / optimizing data movement.

```
select j.email,
       j.status,
       c.account_balance
  from my_json j,
       customer c
 where j.state='TX';
```



- ① Query compilation determines:
 - Data locations
 - Data structure
 - Parallelism
- ② Parallel reads using Big Data SQL
- ③ Process filtered result
 - Move relevant data to database
 - Join with database tables
 - Apply database security policies

Unifying Metadata

To unify metadata for planning and executing SQL queries, we require a catalog of some sort. What tables do I have? What are their column names and types? Are there special options defined on the tables? Who can see which data in these tables?

Given the richness of the Oracle data dictionary, Oracle Big Data SQL unifies metadata using Oracle Database: specifically as external tables. Tables in Hadoop or NoSQL databases are defined as external tables in Oracle. This makes sense, given that the data is external to the DBMS.

The external tables Big Data SQL presents are different than OSCH. They leverage the Hive metastore or user definitions to determine both parallelism and read semantics. That means that if a file in HDFS is 100 blocks, Oracle database understands there are 100 units which can be read in parallel. If the data was stored in a SequenceFile using a binary SerDe, or as Parquet data, or as Avro, that is how the data is read. Big Data SQL uses the exact same InputFormat, RecordReader, and SerDes defined in the Hive metastore to read the data from HDFS.

Once that data is read, we need only to join it with internal data and provide SQL on Hadoop **and** a relational database.

Optimizing Performance

Being able to join data from Hadoop with Oracle Database is a feat in and of itself. However, given the size of data in Hadoop, it ends up being a lot of data to shift around. In order to optimize performance, we must take advantage of what each system can do.

The effect is striking: minimizing data movement by an order of magnitude often yields performance increases of an order of magnitude.

Big Data SQL takes a page from both the Exadata and Hadoop books to optimize performance: it moves work to the data and radically minimizes data movement. It does this via something we call Smart Scan for Hadoop.

Moving the work to the data is straightforward. Smart Scan for Hadoop introduces a new service into the Hadoop ecosystem, which is co-resident with HDFS DataNodes and YARN NodeManagers. Queries from the new external tables are sent to these services to ensure that reads are direct path and data-local. Reading close to the data speeds up I/O, but minimizing data movement requires that Smart Scan do some things that are, well, smart.

Smart Scan for Hadoop

Most queries don't select all columns, and most queries have some kind of predicate on them. Moving unneeded columns and rows is, by definition, excess data movement and impeding performance. Smart Scan for Hadoop gets rid of this excess movement, which in turn radically improves performance.

For example, suppose we were querying a 100 of TB set of JSON data stored in HDFS, but only cared about a few fields -- email and status -- and only wanted results from the state of Texas.

Once data is read from a DataNode, Smart Scan for Hadoop goes beyond just reading. It applies parsing functions to our JSON data, discards any documents which do not contain 'TX' for the state attribute. Then, for those documents which do match, it projects out only the email and status attributes to merge with the rest of the data. Rather than moving every field, for every document, we're able to cut down 100s of TB to 100s of **GB**.

The approach we take to optimizing performance with Big Data SQL makes Big Data much slimmer.

Summary

So, there you have it: fast queries which join data in Oracle Database with data in Hadoop while preserving the makes each system a valuable part of overall information architectures. Big Data SQL unifies metadata, such that data sources can be queried with the best possible parallelism and the correct read semantics. Big Data SQL optimizes performance using approaches inspired by Exadata: filtering out irrelevant data before it can become a bottleneck.

It's SQL that plays data where it lies, letting you place data where you think it belongs.

10.2 Big Data SQL Exercises

If you look back at the OSCH exercises at the number steps including publishing and external table, location files etc. to link an external table to Hadoop – the BigData SQL lesson will be a very short and simple as a new way to link the Oracle database to hive tables and hdfs files. You will notice there is no publishing and you will see how Big Data SQL can dynamically adapt to changes in the Hadoop data. You will also get a chance to play with Oracle 12c new JSON capabilities as well.

1. Go to ("cd") the /home/oracle/exercises/BigSQL directory
2. In the previous exercise for ODI you should have created a new hive table with transformed data called "consolidated_buffer". Run the hive command to open hive, then execute **show create table consolidated_buffer;**



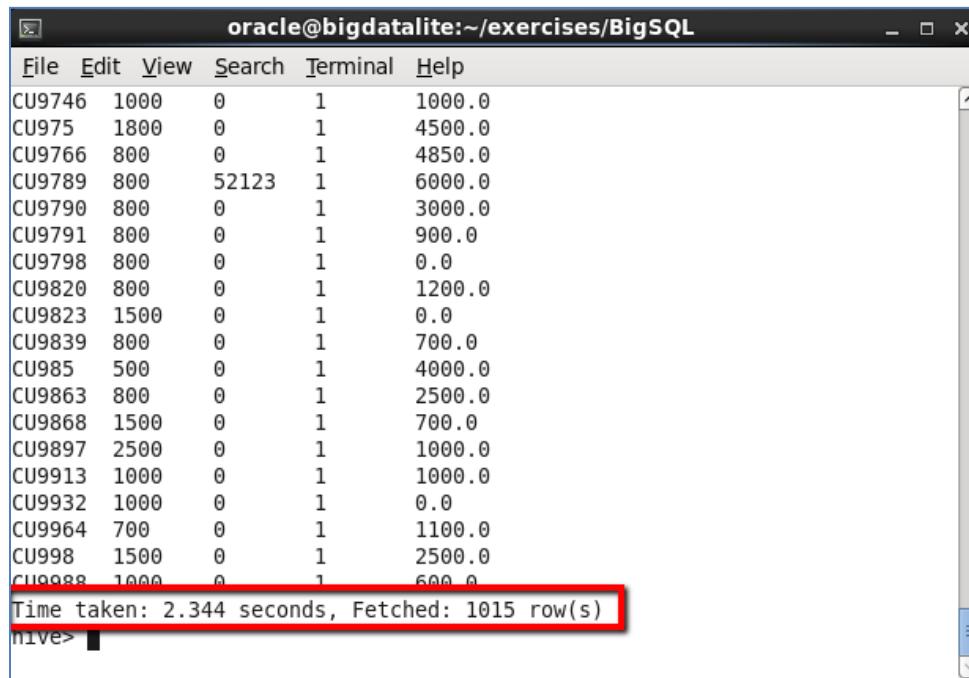
```

oracle@bigdatalite:~/exercises/BigSQL
File Edit View Search Terminal Help

Logging initialized using configuration in jar:/usr/lib/hive/lib/hive-commo
0.12.0 cdh5.1.2.jar!/hive-log4j.properties
hive> show create table consolidated_buffer;
OK
CREATE TABLE `consolidated_buffer`(
  `customer_id` string,
  `credit_card_limits` int,
  `credit_balance` int,
  `n_mortgages` bigint,
  `mortgage_amount` double)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  'hdfs://bigdatalite.localdomain:8020/user/hive/warehouse/consolidated_buffer'
TBLPROPERTIES (
  'transient_lastDdlTime'='1413050816')
Time taken: 2.877 seconds, Fetched: 16 row(s)
hive>

```

3. Lets check that we have data in the table by executing **`select * from consolidated_buffer;`** We should have 1015 rows of data.



customer_id	1000	0	1	1000.0
CU9746	1000	0	1	1000.0
CU975	1800	0	1	4500.0
CU9766	800	0	1	4850.0
CU9789	800	52123	1	6000.0
CU9790	800	0	1	3000.0
CU9791	800	0	1	900.0
CU9798	800	0	1	0.0
CU9820	800	0	1	1200.0
CU9823	1500	0	1	0.0
CU9839	800	0	1	700.0
CU985	500	0	1	4000.0
CU9863	800	0	1	2500.0
CU9868	1500	0	1	700.0
CU9897	2500	0	1	1000.0
CU9913	1000	0	1	1000.0
CU9932	1000	0	1	0.0
CU9964	700	0	1	1100.0
CU998	1500	0	1	2500.0
CU9988	1000	0	1	600.0

Time taken: 2.344 seconds, Fetched: 1015 row(s)

4. Quit hive and open for viewing createtable.sh. This sql script will create an external table in Oracle DB that will point to consolidated_buffer hive table.



```

oracle@bigdatalite:~/exercises/BigSQL
File Edit View Search Terminal Help
#!/bin/sh

sqlplus /nolog <<SQL
connect bda/welcome1@orcl

DROP TABLE BDA.ODI_HIVE;

CREATE TABLE "BDA"."ODI_HIVE"
(
    "CUSTOMER_ID" VARCHAR2(4000 BYTE),
    "CREDIT_CARD_LIMITS" NUMBER,
    "CREDIT_BALANCE" NUMBER,
    "N_MORTGAGES" NUMBER,
    "MORTGAGE_AMOUNT" NUMBER
) ORGANIZATION EXTERNAL
( TYPE ORACLE_HIVE
  DEFAULT DIRECTORY "EXTERNAL_DIR"
  ACCESS PARAMETERS
  (
    com.oracle.bigdata.tablename=default.consolidated_buffer
  )
)
;
SQL
:
```

Notice that compared to OSCH – the format is simple. Also if we were creating an external table with the same name we could even leave out the pointer to the hive table name. But in this case we are changing the table name in Oracle to “ODI_HIVE” to be used with the next exercise in R.

- Run the createtable.sh script and then run the viewdata.sh script to select rows from our newly created external table. We should 1015 rows.

CREDIT_CARD_LIMITS	CREDIT_BALANCE	N_MORTGAGES	MORTGAGE_AMOUNT	
CU9932	1000	0	1	0
CU9964	700	0	1	1100
CU998	1500	0	1	2500

CUSTOMER_ID				
CREDIT_CARD_LIMITS	CREDIT_BALANCE	N_MORTGAGES	MORTGAGE_AMOUNT	
CU9988	1000	0	1	600

1015 rows selected.

SQL> SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0 With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options [oracle@bigdatalite BigSQL]\$

The Next exercise will demonstrate the new JSON features in 12c and dynamic HDFS handling of files.

1. Open the createjsontbl.sh for viewing. You will see that we are adding a single JSON formatted file to the custlog folder in HDFS. Then creating an external table that is point to that folder.

```

oracle@bigdatalite:~/exercises/BigSQL
File Edit View Search Terminal Help
#!/bin/sh

hadoop fs -mkdir custlog
hadoop fs -put data1.json custlog/.

sqlplus /nolog <<SQL
connect bda/welcome1@orcl

CREATE TABLE custlog
(click VARCHAR2(4000))
ORGANIZATION EXTERNAL
(TYPE ORACLE_HDFS
DEFAULT DIRECTORY DEFAULT_DIR
LOCATION ('/user/oracle/custlog/*')
)
REJECT LIMIT UNLIMITED;

select * from custlog;

SQL
-
-
[END]

```

The screenshot shows a terminal window with the title "oracle@bigdatalite:~/exercises/BigSQL". It contains a shell script and SQL code. Red arrows highlight specific parts of the SQL code: one arrow points to the "type ORACLE_HDFS" line, another points to the "LOCATION" clause, and a third points to the "REJECT LIMIT UNLIMITED;" statement. A red box surrounds the "type ORACLE_HDFS" line.

2. Run the createjsontbl.sh script. You should see 4 rows of data return from our new table.

```

oracle@bigdatalite:~/exercises/BigSQL
File Edit View Search Terminal Help

SQL> SQL>
CLICK
-----
{"custid":1185972,"transid":null,"typeid":null,"time":"2012-07-01:00:00:07","teller":null,"activity":8}

{"custid":1354924,"transid":1948,"typeid":9,"time":"2012-07-01:00:00:22","teller":"N","activity":7}

{"custid":1083711,"transid":null,"typeid":null,"time":"2012-07-01:00:00:26","teller":null,"activity":9}

{"custid":1234182,"transid":11547,"typeid":6,"time":"2012-07-01:00:00:32","teller":"Y","activity":7}

CLICK
-----

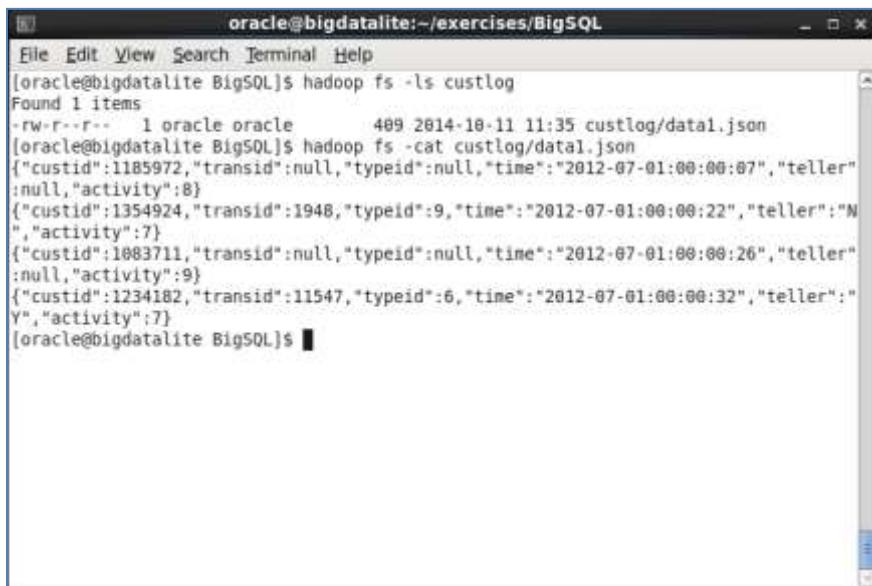
SQL> SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.2.
0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
[oracle@bigdatalite BigSQL]$ 

```

The screenshot shows a terminal window with the title "oracle@bigdatalite:~/exercises/BigSQL". It displays the output of the SQL query, which returns four rows of JSON data. The data is separated by dashed lines and includes fields like custid, transid, typeid, time, teller, and activity.

3. Let check the file and data in the /user/oracle/custlog folder.

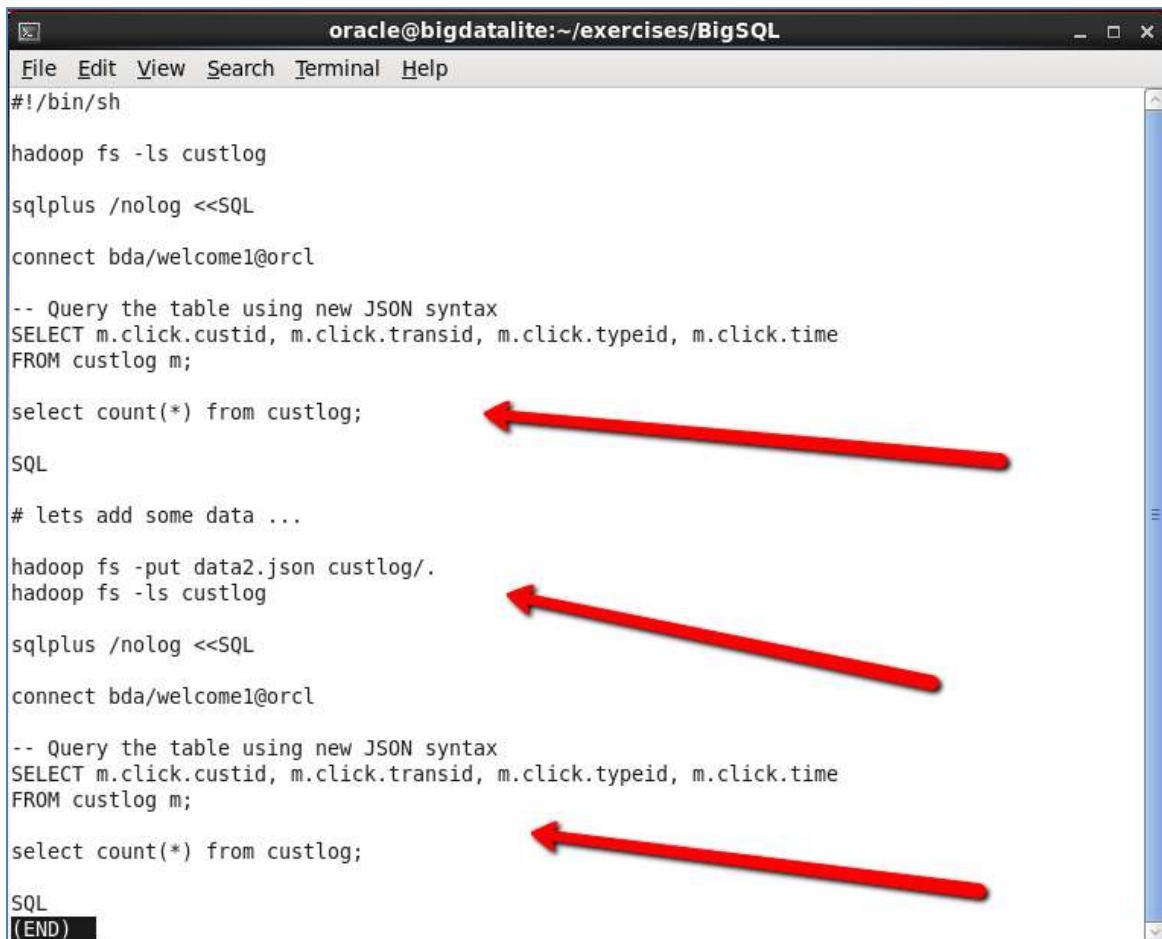
- a. Run **hadoop fs -ls custlog**
- b. Run **hadoop fs -cat custlog/data1.json**



oracle@bigdatalite:~/exercises/BigSQL

```
[oracle@bigdatalite BigSQL]$ hadoop fs -ls custlog
Found 1 items
-rw-r--r-- 1 oracle oracle      409 2014-10-11 11:35 custlog/data1.json
[oracle@bigdatalite BigSQL]$ hadoop fs -cat custlog/data1.json
{"custid":1185972,"transid":null,"typeid":null,"time":"2012-07-01:00:00:07","teller":null,"activity":8}
{"custid":1354924,"transid":1948,"typeid":9,"time":"2012-07-01:00:00:22","teller":N,"activity":7}
{"custid":1083711,"transid":null,"typeid":null,"time":"2012-07-01:00:00:26","teller":null,"activity":9}
{"custid":1234182,"transid":11547,"typeid":6,"time":"2012-07-01:00:00:32","teller":Y,"activity":7}
[oracle@bigdatalite BigSQL]$
```

4. Open for viewing viewjson.sh. You will see that we are using a new dot notation for json data to query our JSON data. We then add a new file to the custlog folder and query it again.



```
oracle@bigdatalite:~/exercises/BigSQL
```

```
#!/bin/sh

hadoop fs -ls custlog

sqlplus /nolog <<SQL
connect bda/welcome1@orcl

-- Query the table using new JSON syntax
SELECT m.click.custid, m.click.transid, m.click.typeid, m.click.time
FROM custlog m;

select count(*) from custlog;
```

←

```
SQL
```

```
# lets add some data ...

hadoop fs -put data2.json custlog/.
hadoop fs -ls custlog
```

←

```
sqlplus /nolog <<SQL
connect bda/welcome1@orcl

-- Query the table using new JSON syntax
SELECT m.click.custid, m.click.transid, m.click.typeid, m.click.time
FROM custlog m;

select count(*) from custlog;
```

←

```
SQL
(END)
```

5. Run viewjson.sh script – the first query should return 4 rows. Notice that no new publishing was needed when we added the second file – you should see that the Oracle external table should now have 9 rows instead of 4.

```
1234182  
11547  
6  
2012-07-01:00:00:32
```

```
SQL> SQL>  
      COUNT(*)  
-----  
        4
```

```
SQL> SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64  
bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options  
Found 2 items
```

```
2012-07-01:00:01:07
```

```
9 rows selected.
```

```
SQL> SQL>  
      COUNT(*)  
-----  
        9
```

```
SQL> SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64  
bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
```

10.3 Summary

You have had a chance to work with Big Data SQL and also as a bonus the new JSON formatting in 12c Oracle DB. Big Data SQL option for BDA is fast and simple to setup and use, the smart scan and dynamic capabilities including using Hive metastore allows you to extend your Exadata database with BDA in ways that truly create a big data ecosystem.,

11. XQUERY FOR HADOOP (OXH)

11.1 Introduction to Oracle OXH XQuery for Hadoop

Oracle XQuery for Hadoop (OXH) enables the use of XQuery to process and transform XML, JSON or Avro content stored on the Hadoop Cluster. OXH is able to take full advantage of the large numbers of CPUs present in a typical cluster, enabling OXH to evaluate XQuery operations in a massively parallel manner.

Oracle XQuery for Hadoop is based on a Hadoop optimized Java implementation of Oracle's proven XQuery engine. This engine automatically evaluates standard W3C XQuery expressions in parallel, leveraging the MapReduce framework to distribute the XQuery expression to all the nodes in the cluster. This enables XQuery expressions to be evaluated by taking the processing to the data, rather than having to bring the data to the XQuery processor, delivering much higher throughput than is available with other XQuery solutions.

Typical use cases for Oracle XQuery for Hadoop include web log analysis, transformation operations on XML, JSON, and Avro content, and processing large volumes of content prior to loading into the database.

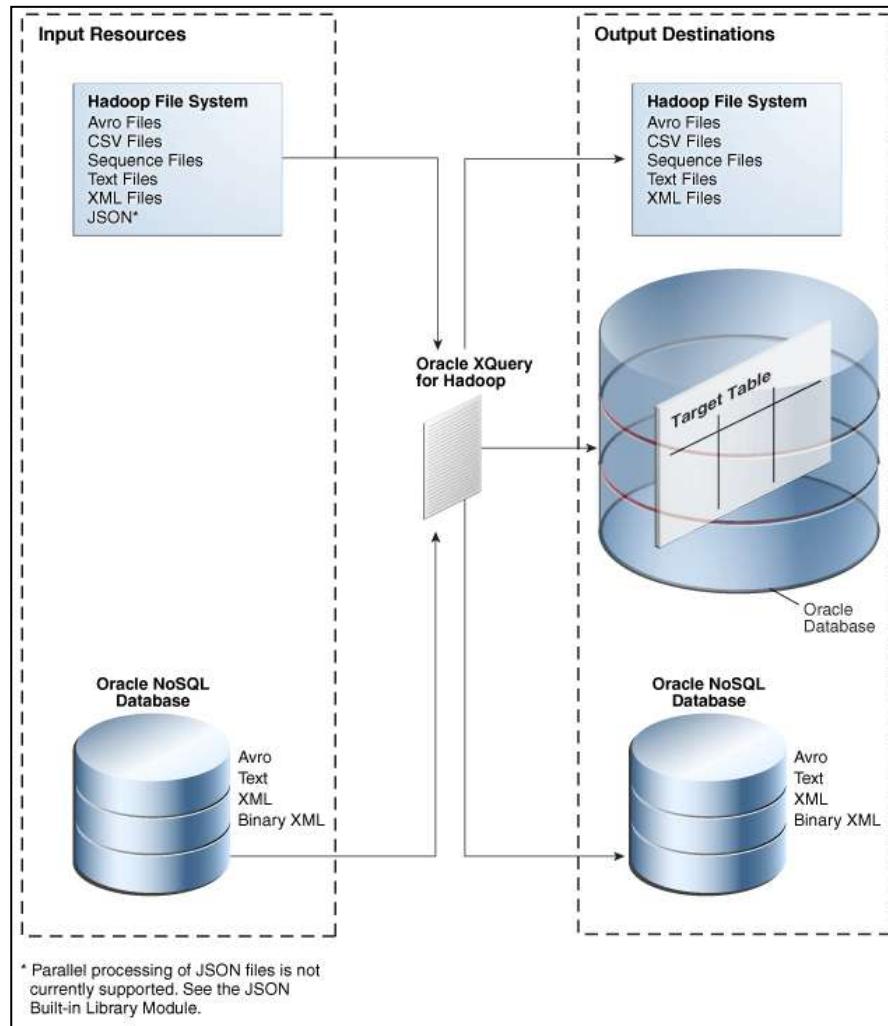
Oracle XQuery for Hadoop	
Features	
Scalable, Native, XQuery Processing	XQuery engines are automatically distributed across the Hadoop cluster, allowing XQueries to be executed where the data is located
Tight Integration with Hadoop Data Stores	Oracle XQuery for Hadoop can operate on content stored in HDFS, Hive or Oracle NoSQL Database
Parallel XML Parsing	Integrated in to the XQuery processor is Oracle's new parallel XML parser. This allows very large XML documents to be processed extremely efficiently by parsing them in parallel on multiple nodes
Fast Load of XQuery Results into Oracle Database	Oracle XQuery for Hadoop allows the results of XQuery operations to be loaded directly into Oracle Database using Oracle Loader for Hadoop

Input data can be located in a file system accessible through the Hadoop File System API, including the Hadoop Distributed File System (HDFS), or stored in Oracle NoSQL Database. Oracle XQuery for Hadoop can write the transformation results to Hadoop files, Oracle NoSQL Database, or Oracle Database.

Oracle XQuery for Hadoop also provides extensions to Apache Hive to support massive XML files. These extensions are available only on Oracle Big Data Appliance.

Oracle XQuery for Hadoop is based on mature industry standards including XPath, XQuery, and XQuery Update Facility. It is fully integrated with other Oracle products, and so it:

- Loads data efficiently into Oracle Database using Oracle Loader for Hadoop.
- Provides read and write support to Oracle NoSQL Database.



Also included is XQuery extensions for Hive. The OXH extensions allow you to create user defined functions (UDF), Input and Output formatters and SerDe for Hive and XML data. Oracle XQuery for Hadoop provides XML processing support that enables you to do the following:

- Query large XML files in HDFS as Hive tables
- Query XML strings in Hive tables
- Query XML file resources in the Hadoop distributed cache
- Efficiently extract atomic values from XML without using expensive DOM parsing
- Retrieve, generate, and transform complex XML elements
- Generate multiple table rows from a single XML value
- Manage missing and dirty data in XML

The OXH extensions for hive also support these W3C modern standards:

- XQuery 1.0
- XQuery Update Facility 1.0 (transform expressions)

- XPath 2.0
- XML Schema 1.0
- XML Namespaces

The OXH extensions have two components:

- XML Input/Output Format and SerDe for creating XML tables

11.2 XML and XQuery primer

XQuery is a W3C Recommendation

XQuery is compatible with several W3C standards, such as XML, Namespaces, XSLT, XPath, and XML Schema.

XQuery 1.0 became a W3C Recommendation January 23, 2007.

- XML stands for eXtensible Markup Language.
- XML is designed to transport and store data.
- XML is important to know, and very easy to learn.

XML Document Example :

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

What is XQuery?

- XQuery is the language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is supported by all major databases
- XQuery is About Querying XML
- XQuery is a language for finding and extracting elements and attributes from XML documents.

Here is an example of a question that XQuery could solve:

"Select all CD records with a price less than \$10 from the CD collection stored in the XML document called cd_catalog.xml"

XQuery - Examples of Use

XQuery can be used to:

- Extract information to use in a Web Service
- Generate summary reports
- Transform XML data to XHTML
- Search Web documents for relevant information

The XML Example Document:

We will use the following XML document in the examples below.

"books.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>

  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>

  <book category="WEB">
    <title lang="en">XQuery Kick Start</title>
    <author>James McGovern</author>
    <author>Per Bothner</author>
    <author>Kurt Cagle</author>
    <author>James Linn</author>
    <author>Vaidyanathan Nagarajan</author>
    <year>2003</year>
    <price>49.99</price>
  </book>

  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

How to Select Nodes from "books.xml"?

Functions

XQuery uses functions to extract data from XML documents. The doc() function in traditional XQuery language is used to open the "books.xml" file from the local filesystem :

```
doc ("books.xml")
```

For Oracle XQuery for Hadoop we are working with the HDFS filesystem and MapReduce so we must import some additional functions to be able to work with data on our cluster :

```

import module namespace xmlf = "oxh:xmlf";
import module namespace text = "oxh:text";
for $x in xmlf:collection("books.xml")/bookstore/book/title
return text:put-xml($x)

```

Path Expressions

XQuery uses path expressions to navigate through elements in an XML document.

The following path expression is used to select all the title elements in the "books.xml" file:

```
xmlf:collection("books.xml")/bookstore/book/title
```

(/bookstore selects the bookstore element, /book selects all the book elements under the bookstore element, and /title selects all the title elements under each book element)

The XQuery above will extract the following:

```

<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>

```

Predicates

XQuery uses predicates to limit the extracted data from XML documents.

The following predicate is used to select all the book elements under the bookstore element that have a price element with a value that is less than 30:

```
xmlf:collection("books.xml")/bookstore/book[price<30]
```

The XQuery above will extract the following:

```

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

```

11.3 Overview OXH XQuery hands on exercise

In this exercise, you will work with the OXH connector to write and execute an XQuery against data in the Hadoop cluster on HDFS and also use OXH to transform an XML file in Hive. The output of one of these queries will load directly into noSQL DB. The exercises are:

1. Simple “XPath” queries.
2. Query an XML file and filter on a single predicate and a single element.
3. Work with Hive UDF and SerDe on XML data.
4. Output results from an XQuery into an Oracle Database.
5. Query noSQL DB Avro schema from exercise.

11.4 Exercises

1. All of the setup and execution for the OXH exercise can be done from the terminal, hence to start out this first exercise please open the terminal by double clicking on the **Terminal icon** on the desktop.



2. To get into the folder where the scripts for the first exercise are, type in the terminal:

```
cd /home/oracle/exercises/oxh  
Then press Enter
```

11.4.1 Simple “XPath” query.

View the file books1a.xq

```
import module namespace xmlf = "oxh:xmlf";  
import module namespace text = "oxh:text";  
for $x in xmlf:collection("books.xml")/bookstore/book/title  
return text:put-xml($x)
```

```
Main Options VT Options VT Fonts  
[oracle@bigdatalite OXH]$ cat books1a.xq  
import module namespace xmlf = "oxh:xmlf";  
import module namespace text = "oxh:text";  
  
for $x in xmlf:collection("books.xml")/bookstore/book/title  
  
return text:put-xml($x)  
  
[oracle@bigdatalite OXH]$ █
```

View the file books1b.xq

```
import module namespace xmlf = "oxh:xmlf";
import module namespace text = "oxh:text";
for $x in xmlf:collection("books.xml")/bookstore/book[price<30]
return text:put-xml($x)
```

```
oracle@bigdatalite:~/exercises/OXH
```

Main Options VT Options VT Fonts

```
[oracle@bigdatalite OXH]$ cat books1b.xq
import module namespace xmlf = "oxh:xmlf";
import module namespace text = "oxh:text";

for $x in xmlf:collection("books.xml")/bookstore/book[price<30]
return text:put-xml($x)

[oracle@bigdatalite OXH]$ █
```

These 2 queries are XPath queries that were shown previously in this section.

View the file books1-local.sh, OXH XQuery is currently set to run locally (-jt local -fs local). This is a great way to test your queries against a small set of sample data before going to the Hadoop cluster and accessing large data sets.

Run the 2 queries in books1-local.sh

```
[oracle@bigdatalite ~]$ ./books1-local.sh
13/12/31 19:11:42 WARN FsFilesystem: "local" is a deprecated filesystem name. Use "file://" instead.
13/12/31 19:11:48 INFO hadoop.xquery [OXH: Oracle XQuery for Hadoop 2.4.0 (build 2.4.0-edn-4.5.0), Copyright (c) 2014 Oracle. All rights reserved.
13/12/31 19:11:51 INFO hadoop.xquery: Submitting map-reduce job "oxhbooks1a.xq#0" id="9F230e4b-44dc-4c3a-9990-0706a9cd9aL0", inputs:[file:/home/oracle/exercises/OXH/books.xml], output:books1
13/12/31 19:11:51 WARN conf.Configuration: session_id is deprecated. Instead, use dfs.metrics.session-id
13/12/31 19:11:51 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=hadoop, sessionId=13425441197854547242_1631180845_1175833885
13/12/31 19:11:52 INFO InputFileInputFormat: Total input paths to process : 1
13/12/31 19:11:52 WARN conf.Configuration: fs.default.name is deprecated. Instead, use fs.defaultFS
13/12/31 19:11:52 INFO filecache.TrackerDistributedCacheManager: Creating apache-xlibbeans.jar in /tmp/hadoop-oracle/wrapped/local/archive/3043401197854547242_1631180845_1175833885/file
13/12/31 19:11:52 INFO filecache.TrackerDistributedCacheManager: Cached File:///opt/oracle/oxh-2.4.0-1/lib/apache-xlibbeans.jar as /tmp/hadoop-oracle/wrapped/local/archive/3043401197854547242_1631180845_1175833885/file
13/12/31 19:11:52 INFO filecache.TrackerDistributedCacheManager: Cached File:///opt/oracle/oxh-2.4.0-1/lib/apache-xlibbeans.jar as /tmp/hadoop-oracle/wrapped/local/archive/3043401197854547242_1631180845_1175833885/file
13/12/31 19:11:52 INFO filecache.TrackerDistributedCacheManager: Creating avro-wrapped-1.7.4-hadoop2.jar in /tmp/hadoop-oracle/wrapped/local/archive/2810429764205038270_-1681533795_1175833885/file
13/12/31 19:11:52 INFO filecache.TrackerDistributedCacheManager: Cached File:///opt/oracle/oxh-2.4.0-1/lib/apache-xlibbeans.jar as /tmp/hadoop-oracle/wrapped/local/archive/2810429764205038270_-1681533795_1175833885/file
13/12/31 19:11:52 INFO filecache.TrackerDistributedCacheManager: Creating avro-wrapped-1.7.4-hadoop2.jar in /tmp/hadoop-oracle/wrapped/local/archive/2810429764205038270_-1681533795_1175833885/file
13/12/31 19:11:52 INFO filecache.TrackerDistributedCacheManager: Cached File:///opt/oracle/oxh-2.4.0-1/lib/apache-xlibbeans.jar as /tmp/hadoop-oracle/wrapped/local/archive/2810429764205038270_-1681533795_1175833885/file
```

Output from books1a.xq:

```
[oracle@bigdatalite OXH]$ cat books1/part-m-00000
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
[oracle@bigdatalite OXH]$
```

Output from books1b.xq:

```
[oracle@bigdatalite OXH]$ cat books2/part-m-00000
<book category="CHILDREN">&xA; <title lang="en">Harry Potter</title>&xA; <author>J K. Rowling</author>&xA;
<year>2005</year>&xA; <price>29.99</price>&xA;</book>
[oracle@bigdatalite OXH]$
```

To convert the queries in books1-local.sh to run against the Hadoop cluster instead of locally, remove “-jt local -fs local”. View the books1.sh :

```
hadoop fs rm -r books1 books2
hadoop jar $OXH_HOME/lib/oxh.jar ./books1a.xq -print -output ./books1
hadoop jar $OXH_HOME/lib/oxh.jar ./books1b.xq -print -output ./books2
```

Copy the file books.xml to the Hadoop HDFS filesystem and run the 2 queries in books1.sh again against the Hadoop Cluster.

```
oracle@bigdatalite:~/exercises/OXH
Main Options VT Options VT Fonts
[oracle@bigdatalite OXH]$ hadoop fs -put books.xml .
[oracle@bigdatalite OXH]$ hadoop fs -ls books.xml
Found 1 items
-rw-r--r-- 1 oracle supergroup 875 2013-12-31 19:27 books.xml
[oracle@bigdatalite OXH]$ ./books1.sh
rm: `books1': No such file or directory
rm: `books2': No such file or directory
13/12/31 19:27:41 INFO hadoop.xquery: OXH: Oracle XQuery for Hadoop 2.4.0 (build 2.4.0-cdh-4.5.0). Copyright (c) 2013, Oracle. All rights reserved.
13/12/31 19:27:44 INFO hadoop.xquery: Submitting map-reduce job "oxh:books1a.xq#0" id="dee0ddc5-13cb-47e9-975a-5c62359408c6.0", inputs=[hdfs://bigdatalite.localdomain:8020/user/oracle/books.xml], output=books1
13/12/31 19:27:46 INFO input.FileInputFormat: Total input paths to process : 1
13/12/31 19:27:46 INFO hadoop.xquery: Waiting for map-reduce job oxh:books1a.xq#0
13/12/31 19:27:46 INFO mapred.JobClient: Running job: job_201312311504_0006
13/12/31 19:27:47 INFO mapred.JobClient: map 0% reduce 0%
13/12/31 19:28:00 INFO mapred.JobClient: map 100% reduce 0%
13/12/31 19:28:02 INFO mapred.JobClient: Job complete: job_201312311504_0006
13/12/31 19:28:02 INFO mapred.JobClient: Counters: 24
13/12/31 19:28:02 INFO mapred.JobClient: File System Counters
13/12/31 19:28:02 INFO mapred.JobClient: FILE: Number of bytes read=0
13/12/31 19:28:02 INFO mapred.JobClient: FILE: Number of bytes written=188525
13/12/31 19:28:02 INFO mapred.JobClient: FILE: Number of read operations=0
13/12/31 19:28:02 INFO mapred.JobClient: FILE: Number of large read operations=0
13/12/31 19:28:02 INFO mapred.JobClient: FILE: Number of write operations=0
13/12/31 19:28:02 INFO mapred.JobClient: HDFS: Number of bytes read=997
13/12/31 19:28:02 INFO mapred.JobClient: HDFS: Number of bytes written=161
13/12/31 19:28:02 INFO mapred.JobClient: HDFS: Number of read operations=2
13/12/31 19:28:02 INFO mapred.JobClient: HDFS: Number of large read operations=0
13/12/31 19:28:02 INFO mapred.JobClient: HDFS: Number of write operations=1
```

Output from books1a.xq:

```
[oracle@bigdatalite OXH]$ hadoop fs -cat books1/part-m-00000
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
[oracle@bigdatalite OXH]$
```

Output from books1b.xq:

```
[oracle@bigdatalite OXH]$ hadoop fs -cat books2/part-m-00000
<book category="CHILDREN">&#xA; <title lang="en">Harry
Potter</title>&#xA; <author>J. K. Rowling</author>&#xA;
<year>2005</year>&#xA; <price>29.99</price>&#xA;</book>
[oracle@bigdatalite OXH]$
```

11.4.2 Query an XML file and filter on a single predicate.

In XQuery we can use FLWOR (for let while order-by return) statements to query our data. In XPath we filtered in path “collection(“books.xml”)/bookstore/book[price<30]”. We can also use a while predicate and even do joins ! Lets convert the earlier boo1a.xq for FLWOR. **Copy books1a.xq to books2.xq** and edit the query to add a where clause, and change the XPath in the query:

Filter by price? Change the collection, predicate and return to:

```
import module namespace xmlf = "oxh:xmlf";
import module namespace text = "oxh:text";
for $x in xmlf:collection("books.xml", "book")
where $x/price<30
return text:put-text($x/title || ' ' || $x/price)
```

Run the query in books2.sh

```
[oracle@bigdatalite OXH]$ ./books2.sh
```

Output of books2.xq

```
[oracle@bigdatalite OXH]$ hadoop fs -cat books3/part-m-00000
Harry Potter $29.99
[oracle@bigdatalite OXH]$
```

Add year to the filter? Change the predicate and return to :

```
import module namespace xmlf = "oxh:xmlf";
import module namespace text = "oxh:text";
for $x in xmlf:collection("books.xml", "book")
where $x/price>10 and $x/year eq '2005'
return text:put-text($x/title || ' ' || $x/price)
```

Run the query in books2.sh

```
[oracle@bigdatalite OXH]$ ./books2.sh
```

Output of books2.xq

```
[oracle@bigdatalite OXH]$ hadoop fs -cat books3/part-m-00000
Everyday Italian $30.00
Harry Potter $29.99
[oracle@bigdatalite OXH]$
```

Filter by attribute? Lets filter by the category “COOKING”. Change the predicate and return to :

```
import module namespace xmlf = "oxh:xmlf";
import module namespace text = "oxh:text";
for $x in xmlf:collection("books.xml", "book")
where $x/@category eq 'COOKING'
return text:put-text($x/title || ' Category : ' || $x/@category || ' ' || $x/price)
```

Run the query in books2.sh

```
[oracle@bigdatalite OXH]$ ./books2.sh
```

Output of books2.xq

```
[oracle@bigdatalite OXH]$ hadoop fs -cat books3/part-m-00000
Everyday Italian Category : COOKING $30.00
[oracle@bigdatalite OXH]$
```

11.4.3 Work with Hive UDF and SerDe on XML data.

In order to use UDF (user defined functions) and SerDe (serialize / de-serialize) operations in Hive, we need to add the some additional JAR files to the MapReduce process using **-auxlib** and also function definitions in **\$OXH_HOME/hive/init.sql** to Hive CLI. To make this a bit simpler, a shell script had been created for you in **~/exercises/OXH** to automate this step. Additionally, the hive exercises can be run in Hue Beeswax, the JAR files have been added to the Beeswax configuration and a saved query in Hue will demonstrate adding the UDF to the session. Remember that Beeswax in Hue is stateless, so we have to give it a hint how to set up UDFs on a per query basis.

Contents of init.sql function definitions :

```
CREATE TEMPORARY FUNCTION xml_query AS 'oracle.hadoop.xquery.hive.OXMLQueryUDF';
CREATE TEMPORARY FUNCTION xml_query_as_bigint AS 'oracle.hadoop.xquery.hive.OXMLQueryBigintUDF';
CREATE TEMPORARY FUNCTION xml_query_as_int AS 'oracle.hadoop.xquery.hive.OXMLQueryIntUDF';
CREATE TEMPORARY FUNCTION xml_query_as_smallint AS 'oracle.hadoop.xquery.hive.OXMLQuerySmallintUDF';
CREATE TEMPORARY FUNCTION xml_query_as_tinyint AS 'oracle.hadoop.xquery.hive.OXMLQueryTinyintUDF';
CREATE TEMPORARY FUNCTION xml_query_as_float AS 'oracle.hadoop.xquery.hive.OXMLQueryFloatUDF';
CREATE TEMPORARY FUNCTION xml_query_as_double AS 'oracle.hadoop.xquery.hive.OXMLQueryDoubleUDF';
CREATE TEMPORARY FUNCTION xml_query_as_boolean AS 'oracle.hadoop.xquery.hive.OXMLQueryBooleanUDF';
CREATE TEMPORARY FUNCTION xml_query_as_string AS 'oracle.hadoop.xquery.hive.OXMLQueryStringUDF';
CREATE TEMPORARY FUNCTION xml_exists AS 'oracle.hadoop.xquery.hive.OXMLExists';
CREATE TEMPORARY FUNCTION xml_table AS 'oracle.hadoop.xquery.hive.OXMLTableUDTF';
```

Shell Script for Hive:

```
** For Big Data Lite 3.0 you will need to replace "hive --auxpath $OXH_HOME/hive/lib -i $OXH_HOME/hive/init.sql $" with "hive --auxpath $HIVE_AUX_JARS_PATH -i $OXH_HOME/hive/init.sql $"
```

```
[oracle@bigdatalite OXH]$ cat hive_oxh.sh
export OXH_HOME=/opt/oracle/oxh-3.0.0-1

# hive --auxpath $OXH_HOME/hive/lib -i $OXH_HOME/hive/init.sql $@
hive --auxpath $HIVE_AUX_JARS_PATH -i $OXH_HOME/hive/init.sql $@
```

Using Hue Beeswax for OXH UDFs :

The screenshot shows the Hue Beeswax interface for running Hive queries. The main area is the 'Query Editor : OXH Query' where an XML query is being run against the 'books.xml' file. On the left side, there's a sidebar with various settings and resource management options. In the 'USER-DEFINED FUNCTIONS' section, two fields are visible: 'Name' (containing 'xml_query') and 'Class' (containing 'oracle.hadoop.xquery.hive.OXMLQueryUDF'). A callout bubble highlights the 'Class' field, indicating the specific UDF implementation being used.

Create Table and load data into Hive: We first need to create a table definition in Hive that will transform our books.xml file into a tabular structure that can be queried in Hive. If you look at the create table statement below, you will notice that the array of authors in the books.xml file is serialized in to a its own XML string.

The function `fn:serialize(<AUTHORS>{./author}</AUTHORS>)` converts :

```
<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>
```

Becomes :

```
<AUTHORS><author>James McGovern</author><author>Per Bothner</author><author>Kurt
Cagle</author><author>James Linn</author><author>Vaidyanathan
Nagarajan</author></AUTHORS>
```

Also we are counting the number of author per book with `fn:count(./author)`.

HiveQL script books-create.hql :

```
drop table bookstore;

CREATE TABLE bookstore (title STRING, category STRING,
lang STRING, year INT, authors STRING,
numauthors INT, price FLOAT)
ROW FORMAT
SERDE 'oracle.hadoop.xquery.hive.OXMSerDe'
STORED AS
INPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLInputFormat'
OUTPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLOutputFormat'
TBLPROPERTIES(
"oxh-elements" = "book",
"oxh-column.title" = "./title",
"oxh-column.category" = "./@category",
"oxh-column.lang" = "./title/@lang",
"oxh-column.year" = "./year",
"oxh-column.authors" = "fn:serialize(<AUTHORS>{./author}</AUTHORS>)",
"oxh-column.numauthors" = "fn:count(./author)",
"oxh-column.price" = "./price");

LOAD DATA LOCAL INPATH 'books.xml' OVERWRITE INTO TABLE bookstore;
```

To create the table and load the data run the command :

```
[oracle@bigdatalite OXH]$ ./hive_oxh.sh -f books-create.hql
Logging initialized using configuration in
jar:file:/usr/lib/hive/lib/hive-common-0.10.0-cdh4.5.0.jar!/hive-
log4j.properties
```

```
Hive history file=/tmp/oracle/hive_job_log_dd5a494a-2481-48ee-a697-  
2a34a180475a_1184142861.txt  
OK  
Time taken: 5.498 seconds
```

We are now ready to work with our XML file in Hive or Beeswax. Below are some sample queries to run and their results.

Query 1

```
select title, category, numauthors, lang, year, price from bookstore;
```

Results

Everyday Italian	COOKING	1	en	2005	30.0
Harry Potter	CHILDREN	1	en	2005	29.99
XQuery Kick Start	WEB	5	en	2003	49.99
Learning XML	WEB	1	en	2003	39.95

Query 2

```
select title, auth from bookstore  
lateral view  
explode(xml_query("AUTHORS/author", authors)) authtab as auth
```

Results

Everyday Italian	Giada De Laurentiis
Harry Potter	J K. Rowling
XQuery Kick Start	James McGovern
XQuery Kick Start	Per Bothner
XQuery Kick Start	Kurt Cagle
XQuery Kick Start	James Linn
XQuery Kick Start	Vaidyanathan Nagarajan
Learning XML	Erik T. Ray

Query 3

```
select title, auth from bookstore  
lateral view  
explode(xml_query("AUTHORS/author", authors)) authtab as auth  
where auth like 'James%';
```

Results

XQuery Kick Start	James McGovern
XQuery Kick Start	James Linn

Query 4

```
select title, authors from bookstore
```

Results

Everyday Italian	<AUTHORS><author>Giada De Laurentiis</author></AUTHORS>
Harry Potter	<AUTHORS><author>J K. Rowling</author></AUTHORS>
XQuery Kick Start	<AUTHORS><author>James McGovern</author><author>Per Bothner...
Learning XML	<AUTHORS><author>Erik T. Ray</author></AUTHORS>

11.4.4 Load results from an XQuery into an Oracle Database.

Oracle OXH XQuery for Hadoop can load query results directly into an Oracle database using Oracle Loader for Hadoop (OLH). OXH Stages the results from an XQuery on the HDFS filesystem in Avro format,

and then calls OLH to load the result sets into Oracle DB. For this exercise we will load the results of a transformation of our books.xml file into a tabular format.

DDL of the target table:

```
CREATE TABLE BOOKS
(
  TITLE VARCHAR2(100),
  LANG VARCHAR2(10),
  CATEGORY VARCHAR2(40),
  YEAR INTEGER,
  PRICE FLOAT,
  AUTHOR VARCHAR2(100)
);
```

The file books3olh.sql has the DDL statements needed to create the target table, execute the following command `sqlplus /nolog @books3olh` to create the table books.

```
[oracle@bigdatalite OXH]$ sqlplus /nolog @books3olh

SQL*Plus: Release 12.1.0.1.0 Production on Thu Jan 2 23:09:57 2014

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Connected.

Table dropped.

Table created.

Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 -
64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing
options
[oracle@bigdatalite OXH]$
```

Open the file books3.xq for viewing. OXH includes an Oracle database adapter that uses a custom put command to publish XQuery results to an Oracle table. The oracle adapter will take the data types in the Oracle table and map them to data in the put command.

```
import module namespace xmlf = "oxh:xmlf";
import module namespace text = "oxh:text";

declare
  %oracle:put
  %oracle-property:targetTable('books')
  %oracle-columns('title','lang','category','year','price','author')
  %oracle-property:connection.user('BDA')
  %oracle-property:connection.password('welcome1')
  %oracle-property:connection.url('jdbc:oracle:thin:@//localhost:1521/orcl')
  function local:myPut($c1, $c2, $c3, $c4, $c5, $c6) external;

for $x in xmlf:collection("books.xml", "book")
for $y in $x/author
  return local:myPut($x/title,$x/title/@lang,$x/@category,$x/year,$x/price,$y)
```

Run the book3.sh command and then check the Oracle table “books” using the Viewdata.sh script or be checking the BDA schema in SQL developer.

```
[oracle@bigdatalite OXH]$ ./books3.sh
[oracle@bigdatalite OXH]$ ./Viewdata.sh

SQL*Plus: Release 12.1.0.1.0 Production on Mon Jan 6 03:27:49 2014

Copyright (c) 1982, 2013, Oracle. All rights reserved.

SQL> Connected.
SQL>
TITLE
-----
LANG      CATEGORY          YEAR      PRICE
-----
AUTHOR
-----
Everyday Italian
en          COOKING           2005      30
Giada De Laurentiis

Harry Potter
en          CHILDREN          2005      29.99
J K. Rowling
```

SQLDeveloper output :

TITLE	LANG	CATEGORY	YEAR	PRICE	AUTHOR
1 Everyday Italian	en	COOKING	2005	30	Giada De Laurentiis
2 Harry Potter	en	CHILDREN	2005	29.99	J K. Rowling
3 XQuery Kick Start	en	WEB	2003	49.99	James McGovern
4 XQuery Kick Start	en	WEB	2003	49.99	Per Bothner
5 XQuery Kick Start	en	WEB	2003	49.99	Kurt Cagle
6 XQuery Kick Start	en	WEB	2003	49.99	James Linn
7 XQuery Kick Start	en	WEB	2003	49.99	Vaidyanathan Nagarajan
8 Learning XML	en	WEB	2003	39.95	Erik T. Ray

11.4.5 Query noSQL DB Avro schema from exercise 4.

The Oracle noSQL DB Adapter for OXH can read and write to the Oracle noSQL DB. The noSQL adapter can work with key-value pairs, JSON documents and Avro schemas. In the noSQL exercises (4), we loaded a set of transactions into our noSQL DB. The parent key for these transactions was “Load1/<customer id>” using the Avro schema “trans”. We will use a collection function for Avro data “kv:collection-avroxml” to read the data from noSQL DB.

For the first example we will use OXH to query the data in noSQL DB and group the transaction by teller.
Open nosql-sums.xq for viewing :

```
[oracle@bigdatalite OXH]$ cat nosql-sums.xq
import module namespace text = "oxh:text";
import module namespace kv = "oxh:kv";

for $y in kv:collection-avroxml("/Load1",kv:depth-descendants-only())
let $teller := xs:string($y/teller)
group by $teller

return text:put-text("Teller: " || $teller || " Num Transactions: " ||
fn:count($y) || " Total Amount: " || fn:sum($y/sum))
```

In order to connect to our noSQL DB, we supply connection information

```
[oracle@bigdatalite OXH]$ cat nosql.sumsh
hadoop fs -rm -r nosqlsums-out

hadoop jar $OXH_HOME/lib/oxh.jar -D oracle.kv.hosts=localhost:5000 /
-D oracle.kv_kvstore=kvstore ./nosql-sums.xq -print -output ./nosqlsums-out
```

Execute the nosql.sumsh script.

Results of the nosql-sums.xq query :

```
Teller: 0 Num Transactions: 5776 Total Amount: 51866.65
Teller: 1 Num Transactions: 1757 Total Amount: 2647.17
```

In the second example we will use OXH to query the data in noSQL DB and group the transactions by customer id. We extract the customer id from the parent key by using the kv:key() function to get the parent key and then converting it to a string and separating out the customer id. Also you can uncomment (comments are "(:)") the where clause to select an individual customer. Open nosql-custsums.xq for viewing:

```
[oracle@bigdatalite OXH]$ cat nosql-custsums.xq
import module namespace text = "oxh:text";
import module namespace kv = "oxh:kv";

for $y in kv:collection-avroxml("/Load1",kv:depth-descendants-only())
let $keystr := fn:tokenize(xs:string($y/kv:key()),"\s*/\s*")
let $cust_id := $keystr[3]
(: where $cust_id eq "CU998" :)
group by $cust_id

return text:put-text(" cust: " || $cust_id || " Number of Trans: " ||
fn:count($y) || " Totals : $" || xs:string(fn:sum($y/sum)))
```

Execute the nosql.custsums.sh script , Results of the nosql-custsums.xq query:

```
cust: CU2798 Number of Trans: 9 Totals : $53.15
cust: CU2799 Number of Trans: 6 Totals : $53.04
cust: CU2804 Number of Trans: 4 Totals : $53.07
cust: CU2806 Number of Trans: 5 Totals : $53.12
cust: CU2855 Number of Trans: 12 Totals : $54.26
cust: CU2866 Number of Trans: 5 Totals : $53.07
cust: CU2870 Number of Trans: 9 Totals : $53.16
cust: CU2880 Number of Trans: 10 Totals : $54.78
cust: CU2885 Number of Trans: 19 Totals : $62.12
cust: CU289 Number of Trans: 7 Totals : $53.98
```

The third and last exercise we will take our noSQL query and join our noSQL transactions with an HDFS file that represents our customers from CRM and add first / last name and LTV score to the transactions.

Open nosql-join.xq for viewing :

```
import module namespace text = "oxh:text";
import module namespace kv = "oxh:kv";

for $y in kv:collection-avroxml("/Load1",kv:depth-descendants-only())
for $x in fn:unparsed-text-lines("crm_data.csv")
let $keystr := fn:tokenize(xs:string($y/kv:key()),"\s*/\s*")
let $split := fn:tokenize($x, "\s*,\s*")
let $cust_id := $keystr[3]
(:CUSTOMER_ID,LAST,FIRST,STATE,REGION,SEX,PROFESSION,BUY_INSURANCE,AGE,HAS_CHILDREN,SALARY,N_OF_DEPENDENTS,CAR_OWNERSHIP,HOUSE_OWNERSHIP,TIME_AS_CUSTOMER,MARITAL_STATUS,LTV,LTV_BIN :)
let $crmcust_id := $split[1]
let $first := $split[2]
let $last := $split[3]
let $ltvbin := $split[18]
where $crmcust_id eq $cust_id

return text:put-text(" name: " || $last || " " || $first || " cust: " ||
$cust_id || " Number of Trans: " || fn:count($y) || " Rating: " || $ltvbin || " Totals : $" || xs:string(fn:sum($y/sum)))
```

Open the file `crm_data.csv` for viewing. This is the file we will be joining our nosql data with:

```
# cat crm_data.csv
CU7854,BRENTON,ROJAS,LA,South,M,Childcare
Worker,No,33,1,69838,1,1,1,1,MARRIED,24259.5,HIGH
CU12993,GABRIELE,ATKINSON,NY,NorthEast,F,First-line
Manager,No,23,0,75409,5,1,1,5,SINGLE,23652.25,HIGH
CU608,ALEJANDRO,PAULSON,CA,West,M,Secretary,No,27,1,72953,1,1,2,1,DIVORCED,34438.25,VERY
HIGH
CU10025,DIRK,DYE,CA,West,M,DBA,No,36,0,72196,3,1,1,3,DIVORCED,27149,HIGH
CU11680,JUAN,ROARK,NY,NorthEast,M,PROF-5,No,20,0,54836,0,0,0,2,SINGLE,16709,MEDIUM
CU10706,RANDY,BARRON,NY,NorthEast,M,PROF-20,No,26,0,68436,3,1,1,3,MARRIED,25209,HIGH
CU6752,ELEONOR,COLLIER,MI,Midwest,F,Author,No,68,1,62499,1,1,1,1,WIDOWED,25924.75,HIGH
CU6932,TYRELL,LOONEY,CA,West,M,PROF-30,No,48,1,65042,6,1,1,4,SINGLE,16060.5,MEDIUM
CU9555,ZACK,DANIEL,TX,Southwest,M,Not
specified,No,33,0,69052,1,1,1,1,DIVORCED,29063,HIGH
CU475,SIERRA,VANCE,IL,Midwest,F,PROF-58,No,54,0,68490,5,1,1,5,DIVORCED,25022.5,HIGH
CU4188,TERRANCE,WORKMAN,MI,Midwest,M,PROF-
25,No,32,0,60341,1,1,0,4,SINGLE,18285.25,MEDIUM
```

Open the `nosql.join.sh` for viewing : This query is using Hadoop distributed cache to put the `crm` file `crm_data.csv` into memory since it needs to be scanned every time we loop through a transaction. The `files` parameter accomplishes this in the script. The script is also loading the file into the HDFS file system.

```
hadoop fs -rm crm_data.csv
hadoop fs -rm -r nosql.j-out

hadoop fs -put crm_data.csv

hadoop jar $OXH_HOME/lib/oxh.jar -files "crm_data.csv" -D
oracle.kv.hosts=localhost:5000 -D oracle.kv_kvstore=kvstore ./nosql-join.xq -print -
output ./nosql.j-out
```

Execute `nosql.join.sh` script. Results of the `nosql-join.xq` query:

```
name: FOWLER, RIKKI cust: CU4143 Number of Trans: 1 Rating: HIGH Totals : $1.59
name: FOWLER, RIKKI cust: CU4143 Number of Trans: 1 Rating: HIGH Totals : $43.95
name: ROSALES, STACY cust: CU4153 Number of Trans: 1 Rating: MEDIUM Totals : $1.34
name: ROSALES, STACY cust: CU4153 Number of Trans: 1 Rating: MEDIUM Totals : $1.82
name: ROSALES, STACY cust: CU4153 Number of Trans: 1 Rating: MEDIUM Totals : $1.58
name: ROSALES, STACY cust: CU4153 Number of Trans: 1 Rating: MEDIUM Totals : $1.77
name: ROSALES, STACY cust: CU4153 Number of Trans: 1 Rating: MEDIUM Totals : $1.55
name: ROSALES, STACY cust: CU4153 Number of Trans: 1 Rating: MEDIUM Totals : $1.36
name: ROSALES, STACY cust: CU4153 Number of Trans: 1 Rating: MEDIUM Totals : $43.06
name: LOVETT, IRVING cust: CU4291 Number of Trans: 1 Rating: VERY HIGH Totals : $1.06
name: LOVETT, IRVING cust: CU4291 Number of Trans: 1 Rating: VERY HIGH Totals : $1.15
name: LOVETT, IRVING cust: CU4291 Number of Trans: 1 Rating: VERY HIGH Totals : $1.07
```

11.5 Summary

In the previous exercises you have had a chance to explore the OXH Oracle XQuery for Hadoop connector and have transformed, queried, loaded into Oracle DB and noSQL DB. The OXH connector is a very powerful tool that can help you work with XML files directly and with Hive without having to first convert the XML files to a different format. Oracle XQuery for Hadoop (OXH) uses the power of XQuery language to process and transform XML, JSON or Avro content stored on the Hadoop Cluster.

12. PROGRAMMING WITH R

12.1 Introduction to Enterprise R

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, etc.) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

Oracle R Enterprise integrates the open-source R statistical environment and language with Oracle Database 11g/12c and Exadata; it delivers enterprise-level advanced analytics based on the R environment. The Oracle R Connector for Hadoop integrates R with the Big Data Appliance and Hadoop.

Oracle R Enterprise allows analysts and statisticians to leverage existing R applications and use the R client directly against data stored in Oracle Database 11g/12c—vastly increasing scalability, performance and security. The combination of Oracle Database 11g/12c and R delivers an enterprise-ready, deeply integrated environment for advanced analytics. Data analysts can also take advantage of analytical sandboxes, where they can analyze data and develop R scripts for deployment while results stay managed inside Oracle Database.

As an embedded component of the RDBMS, Oracle R Enterprise eliminates R's memory constraints since it can work on data directly in the database. Oracle R Enterprise leverages Oracle's in-database analytics and scales R for high-performance in Exadata. As well does Oracle R Connector for Hadoop and the Big Data Appliance. Being part of the Oracle ecosystem, ORE enables execution of R scripts in the database to support enterprise production applications and OBIEE dashboards, both for structured results and graphics. Since it's R, we're able to leverage the latest R algorithms and contributed packages.

Oracle R Enterprise users not only can build models using any of the data mining algorithms in the CRAN task view for machine learning, but also leverage in-database implementations for predictions (e.g., stepwise regression, GLM, SVM), attribute selection, clustering, feature extraction via non-negative matrix factorization, association rules, and anomaly detection.

12.2 Overview of Hands on Exercise

In this exercise you will be introduced to the R programming language as well as to the enhancements Oracle has brought to the programming language. Limitations where all data must be kept in system memory are now gone as you can save and load data from and to both the Oracle database and HDFS. Now that we have brought the data processed with NoSQL, Pig and Hive into the Oracle Database we can create a 360 degree view of the customers by merging the imported data sets with the CRM dataset that already exists inside the Oracle Database. We will create the 360 degree view of the customers and perform decision tree algorithms on our data to identify the customer profiles that are most likely to buy insurance from our company. With that information we can target such customers with marketing campaigns in the future. Furthermore, we will merge the data sets in R and we will save the resulting data

set into the Oracle Database using Oracle R Enterprise and into HDFS using the Oracle R Connector for Hadoop.

In this exercise you will:

1. Pull data from Oracle Database tables into R
2. Merge the data in R to obtain a 360 degree view of the customers
3. Mine the data using decision tree algorithms
4. Create histogram plots and decision tree plots
5. Save the data in HDFS and the Oracle Database
6. View the data from HDFS and the Oracle Database

Note: To successfully go through this exercise, you have to go through the NoSQL, Hive, Pig, OLH, OSCH and ODI exercises first (see Chapters 4 to 9).

12.3 Working with R

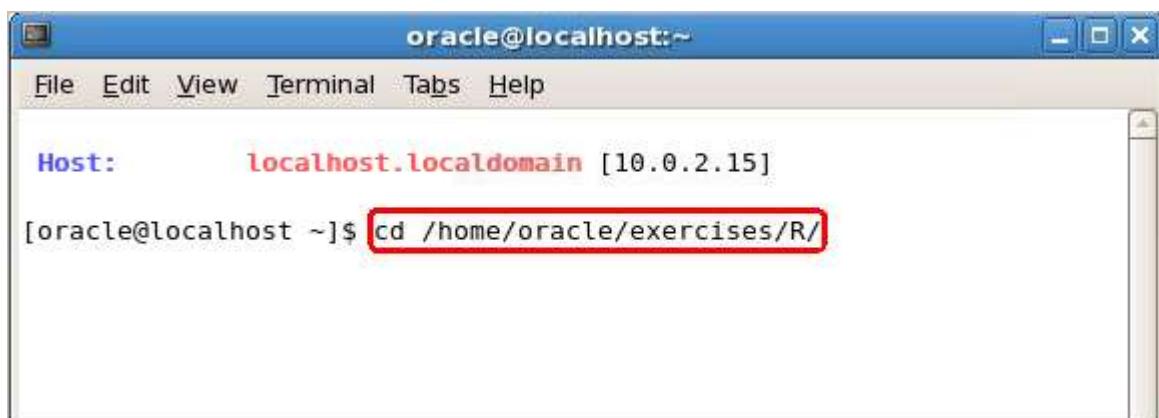
1. All of the setup and execution for this exercise can be done from the terminal, hence open a terminal by double clicking on the **Terminal icon** on the desktop.



2. To get into the folder where the scripts for the R exercise are, type in the terminal:

```
cd /home/oracle/exercises/R
```

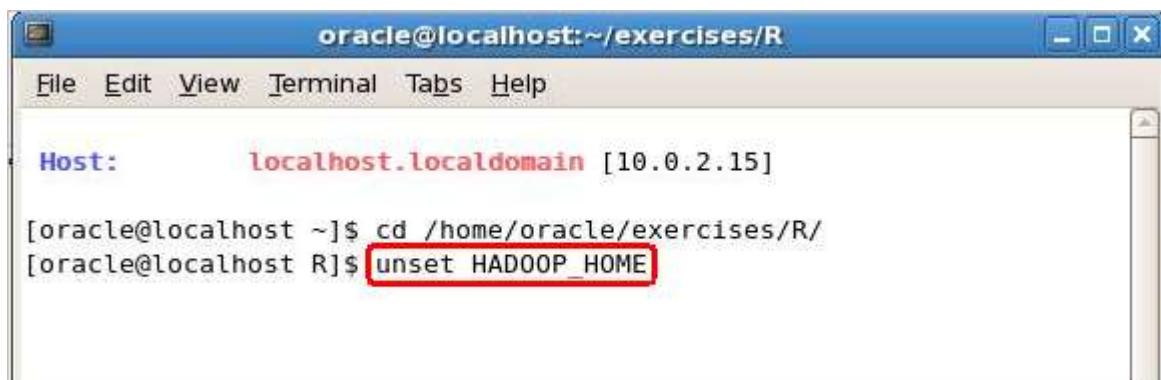
Then press **Enter**



3. Before starting R, we need to correct a couple of environment variables for R.
 - a. We need to correct the value of OLH_HOME environment variable in the file /home/oracle/.Renviron to **OLH_HOME=/opt/oracle/oraloader-2.3.1-h2**
 - b. Un-set the value of the HADOOP_HOME environment variable, as it interferes with the Oracle R Connector for Hadoop. Go to the terminal and type:

```
unset HADOOP_HOME
```

Then press **Enter**

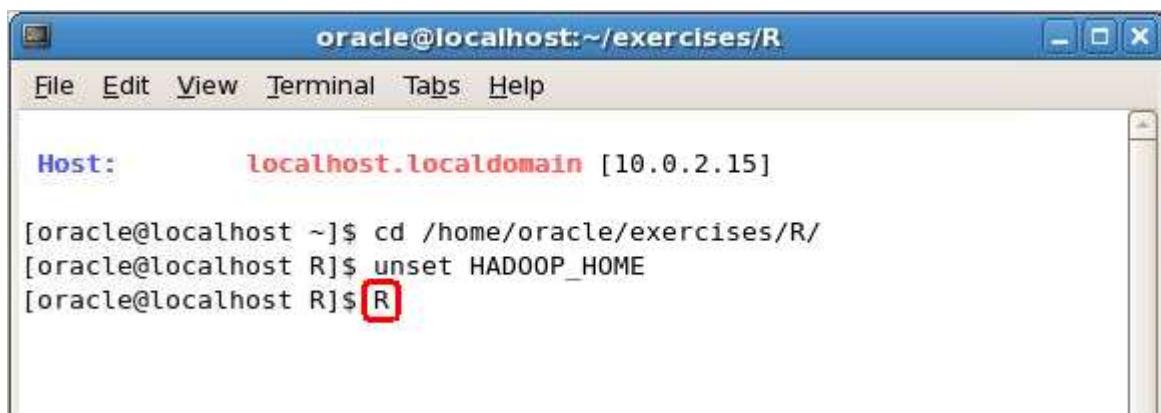


A screenshot of a terminal window titled "oracle@localhost:~/exercises/R". The window has a standard Linux-style interface with a menu bar (File, Edit, View, Terminal, Tabs, Help) and a title bar. The terminal prompt shows the host as "localhost.localdomain [10.0.2.15]". Below the prompt, the user has entered the command "unset HADOOP_HOME", which is highlighted with a red rectangle.

4. We can now start R. Go to the terminal and type:

```
R
```

Then press **Enter**



A screenshot of a terminal window titled "oracle@localhost:~/exercises/R". The window has a standard Linux-style interface with a menu bar (File, Edit, View, Terminal, Tabs, Help) and a title bar. The terminal prompt shows the host as "localhost.localdomain [10.0.2.15]". Below the prompt, the user has entered the command "R", which is highlighted with a red rectangle.

5. We should now load the library for Oracle R Enterprise. This will enable us to connect R with the Oracle Database and move data between the two environments, as well as calling of R functions from the database and vice versa. Go to the terminal and type:

```
library(ORE)
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the standard R startup message, which includes information about the R version, platform, license, and distribution details. At the bottom of the window, the command "R> library(ORE)" is entered, with the entire command highlighted by a red rectangular box.

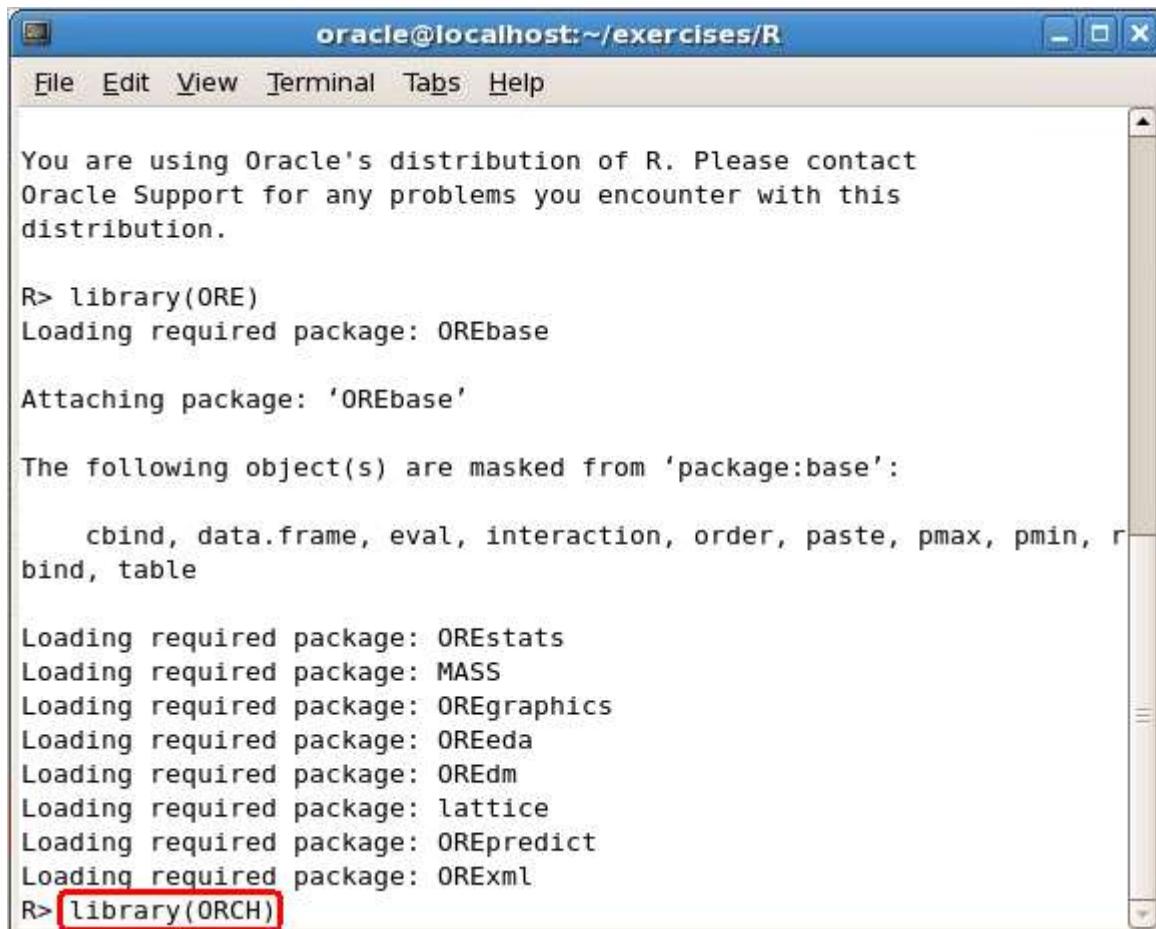
```
s"  
Copyright (C) The R Foundation for Statistical Computing  
ISBN 3-900051-07-0  
Platform: x86_64-unknown-linux-gnu (64-bit)  
  
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
You are using Oracle's distribution of R. Please contact  
Oracle Support for any problems you encounter with this  
distribution.
```

R> **library(ORE)**

6. Next up, let's load the library for Oracle R Connector for Hadoop. The library will enable the movement of data from and to HFDS. This library also has many different functions for combining R with Hadoop. Go to the terminal and type:

```
library(ORCH)
```

Then press **Enter**



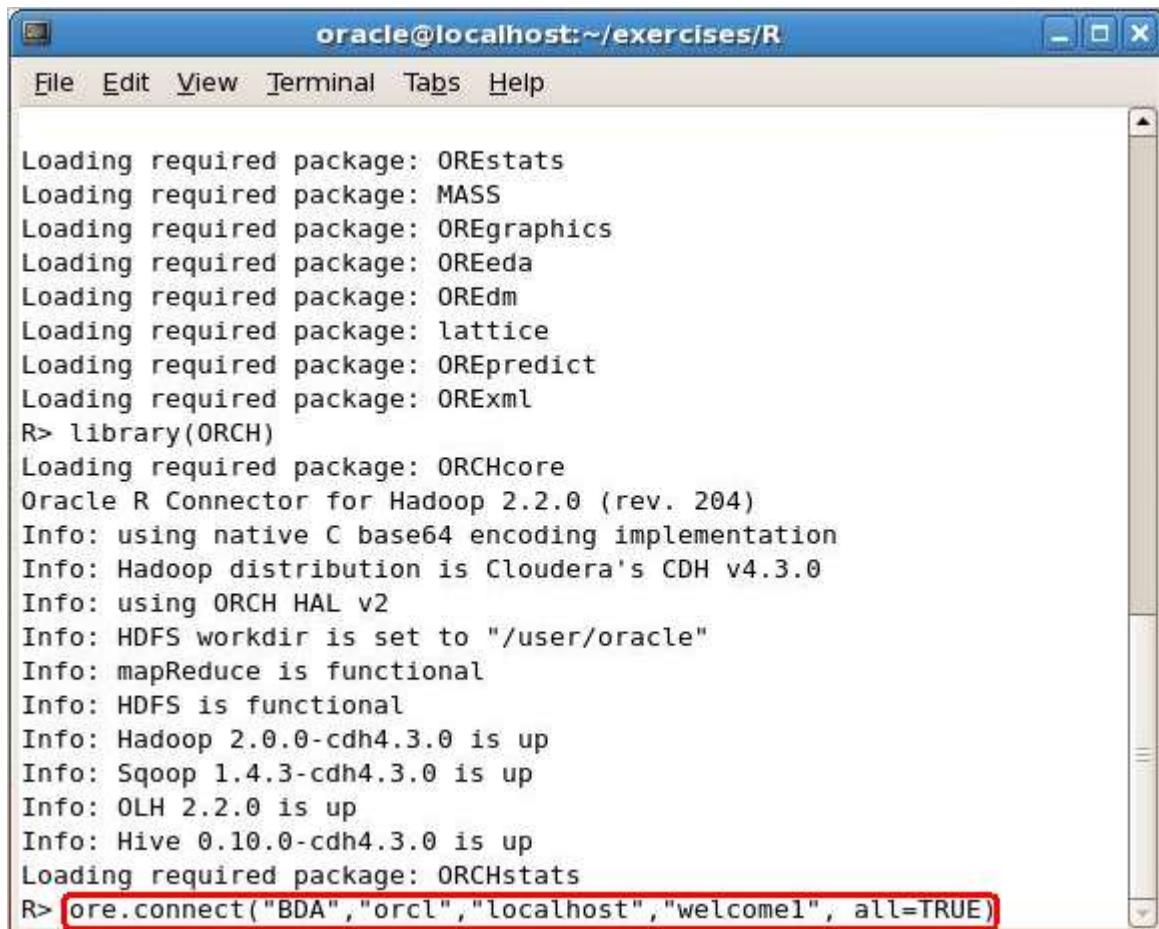
The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following R session output:

```
You are using Oracle's distribution of R. Please contact  
Oracle Support for any problems you encounter with this  
distribution.  
  
R> library(ORE)  
Loading required package: OREbase  
  
Attaching package: 'OREbase'  
  
The following object(s) are masked from 'package:base':  
  
  cbind, data.frame, eval, interaction, order, paste, pmax, pmin, r  
  bind, table  
  
Loading required package: OREstats  
Loading required package: MASS  
Loading required package: OREgraphics  
Loading required package: OREeda  
Loading required package: OREdm  
Loading required package: lattice  
Loading required package: OREPredict  
Loading required package: ORExml  
R> library(ORCH)
```

7. The next command connects Oracle R Enterprise to the Oracle Database user BDA. This will give us the ability to import data from the tables in this schema, as well as the ability to push data from R into tables in this schema. Go to the terminal and type:

```
ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)
```

Then press **Enter**



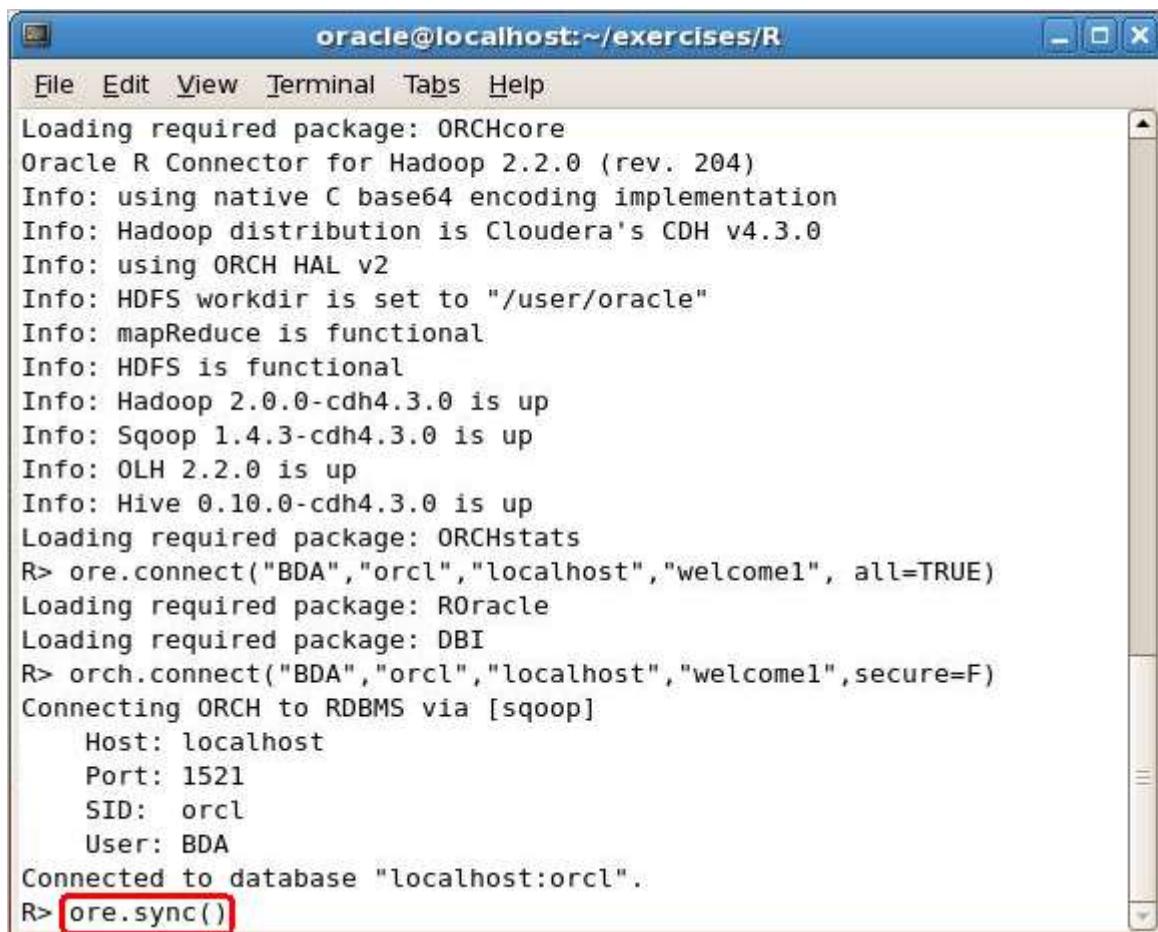
```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help

Loading required package: OREstats
Loading required package: MASS
Loading required package: OREgraphics
Loading required package: OREeda
Loading required package: OREdm
Loading required package: lattice
Loading required package: OREPredict
Loading required package: ORExml
R> library(ORCH)
Loading required package: ORCHcore
Oracle R Connector for Hadoop 2.2.0 (rev. 204)
Info: using native C base64 encoding implementation
Info: Hadoop distribution is Cloudera's CDH v4.3.0
Info: using ORCH HAL v2
Info: HDFS workdir is set to "/user/oracle"
Info: mapReduce is functional
Info: HDFS is functional
Info: Hadoop 2.0.0-cdh4.3.0 is up
Info: Sqoop 1.4.3-cdh4.3.0 is up
Info: OLH 2.2.0 is up
Info: Hive 0.10.0-cdh4.3.0 is up
Loading required package: ORCHstats
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)
```

8. As part of the setup we will also need to run the ore.sync() function which synchronizes the metadata in the database schema with the R environment. Let's run that function, so go to the terminal and type:

```
ore.sync()
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following text output:

```
File Edit View Terminal Tabs Help
Loading required package: ORCHcore
Oracle R Connector for Hadoop 2.2.0 (rev. 204)
Info: using native C base64 encoding implementation
Info: Hadoop distribution is Cloudera's CDH v4.3.0
Info: using ORCH HAL v2
Info: HDFS workdir is set to "/user/oracle"
Info: mapReduce is functional
Info: HDFS is functional
Info: Hadoop 2.0.0-cdh4.3.0 is up
Info: Sqoop 1.4.3-cdh4.3.0 is up
Info: OLH 2.2.0 is up
Info: Hive 0.10.0-cdh4.3.0 is up
Loading required package: ORCHstats
R> ore.connect("BDA","orcl","localhost","welcomel", all=TRUE)
Loading required package: ROracle
Loading required package: DBI
R> orch.connect("BDA","orcl","localhost","welcomel",secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
```

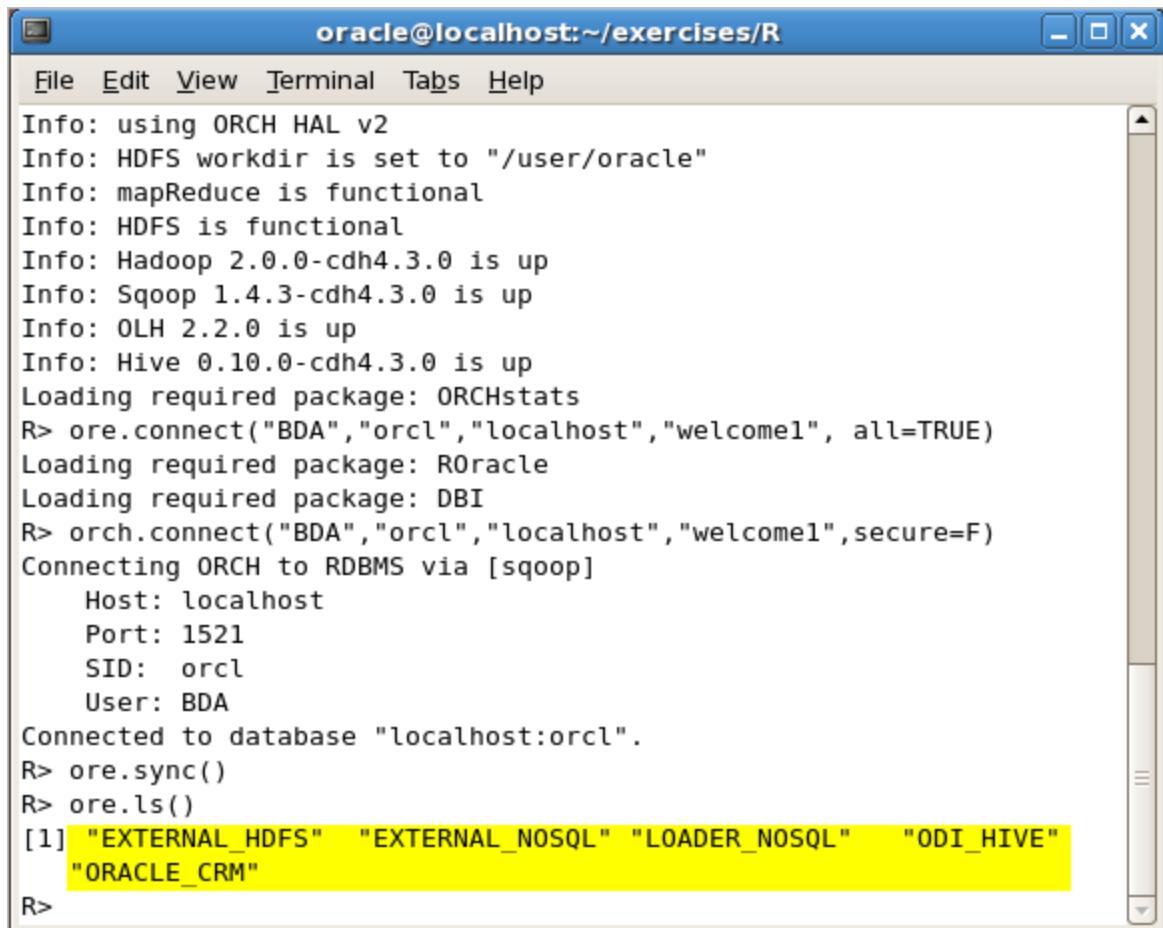
9. We can now list all of the tables (ORE objects) available in our specified schema. The ore.ls() function lists the ORE objects in a specific schema. Let's run that function, so go to the terminal and type:

```
ore.ls()
```

Then press **Enter**

```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
Oracle R Connector for Hadoop 2.2.0 (rev. 204)
Info: using native C base64 encoding implementation
Info: Hadoop distribution is Cloudera's CDH v4.3.0
Info: using ORCH HAL v2
Info: HDFS workdir is set to "/user/oracle"
Info: mapReduce is functional
Info: HDFS is functional
Info: Hadoop 2.0.0-cdh4.3.0 is up
Info: Sqoop 1.4.3-cdh4.3.0 is up
Info: OLH 2.2.0 is up
Info: Hive 0.10.0-cdh4.3.0 is up
Loading required package: ORCHstats
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)
Loading required package: ROracle
Loading required package: DBI
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
```

The five tables from the BDA schema are shown. Notice the ORACLE_CRM table that already exists in the Oracle Database and holds CRM data for our 1015 customers.



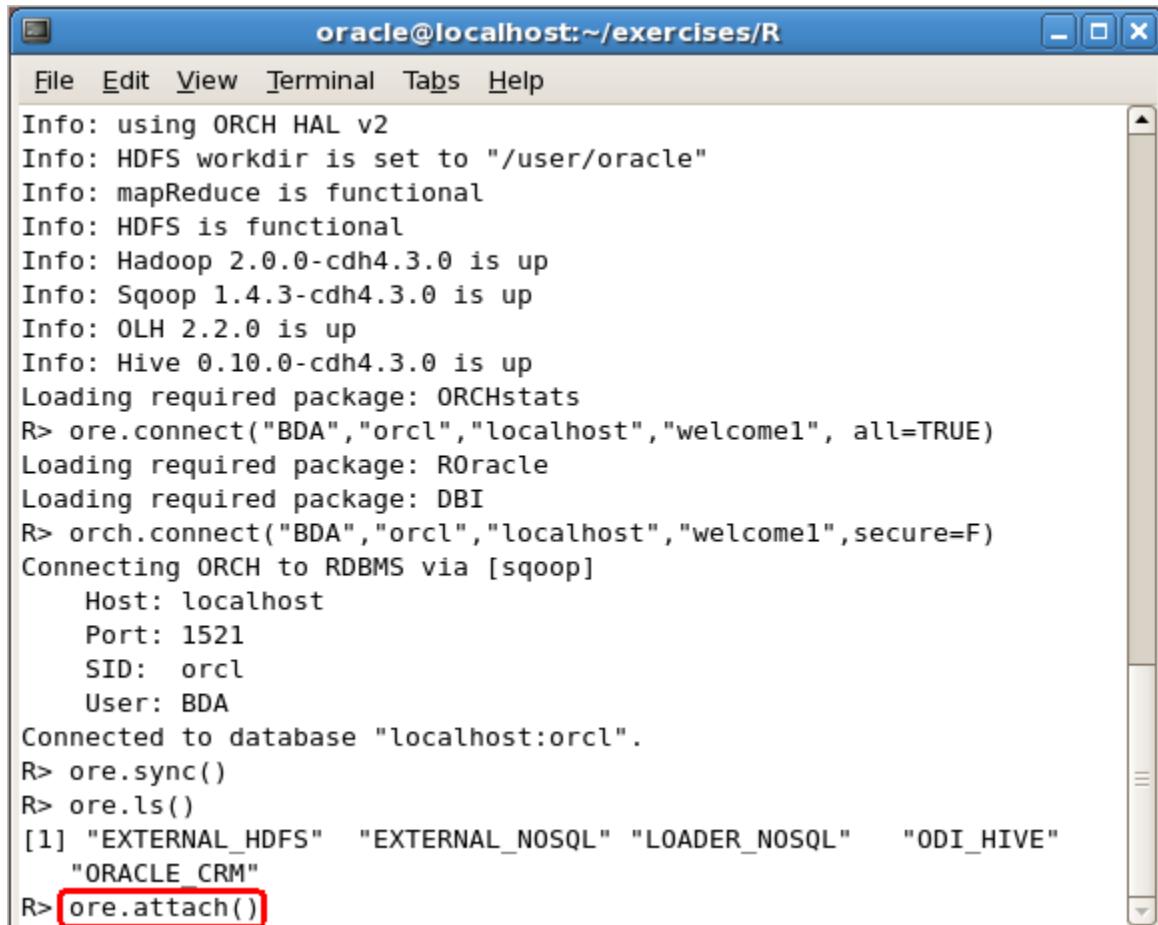
A screenshot of a terminal window titled "oracle@localhost:~/exercises/R". The window shows an R session connecting to an Oracle database named "BDA" on "orcl" at "localhost". The session starts with system information and then connects to the database. It then runs the command "ore.ls()", which lists several tables: "EXTERNAL_HDFS", "EXTERNAL_NOSQL", "LOADER_NOSQL", "ODI_HIVE", and "ORACLE_CRM". The last two items in the list are highlighted with a yellow background.

```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
Info: using ORCH HAL v2
Info: HDFS workdir is set to "/user/oracle"
Info: mapReduce is functional
Info: HDFS is functional
Info: Hadoop 2.0.0-cdh4.3.0 is up
Info: Sqoop 1.4.3-cdh4.3.0 is up
Info: OLH 2.2.0 is up
Info: Hive 0.10.0-cdh4.3.0 is up
Loading required package: ORCHstats
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)
Loading required package: ROracle
Loading required package: DBI
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL"  "LOADER_NOSQL"    "ODI_HIVE"
     "ORACLE_CRM"
R>
```

10. The ore.attach() function attaches the database schema to provide access to views and tables, so that they can be manipulated from a local R session. Let's run that function, go to the terminal and type:

```
ore.attach()
```

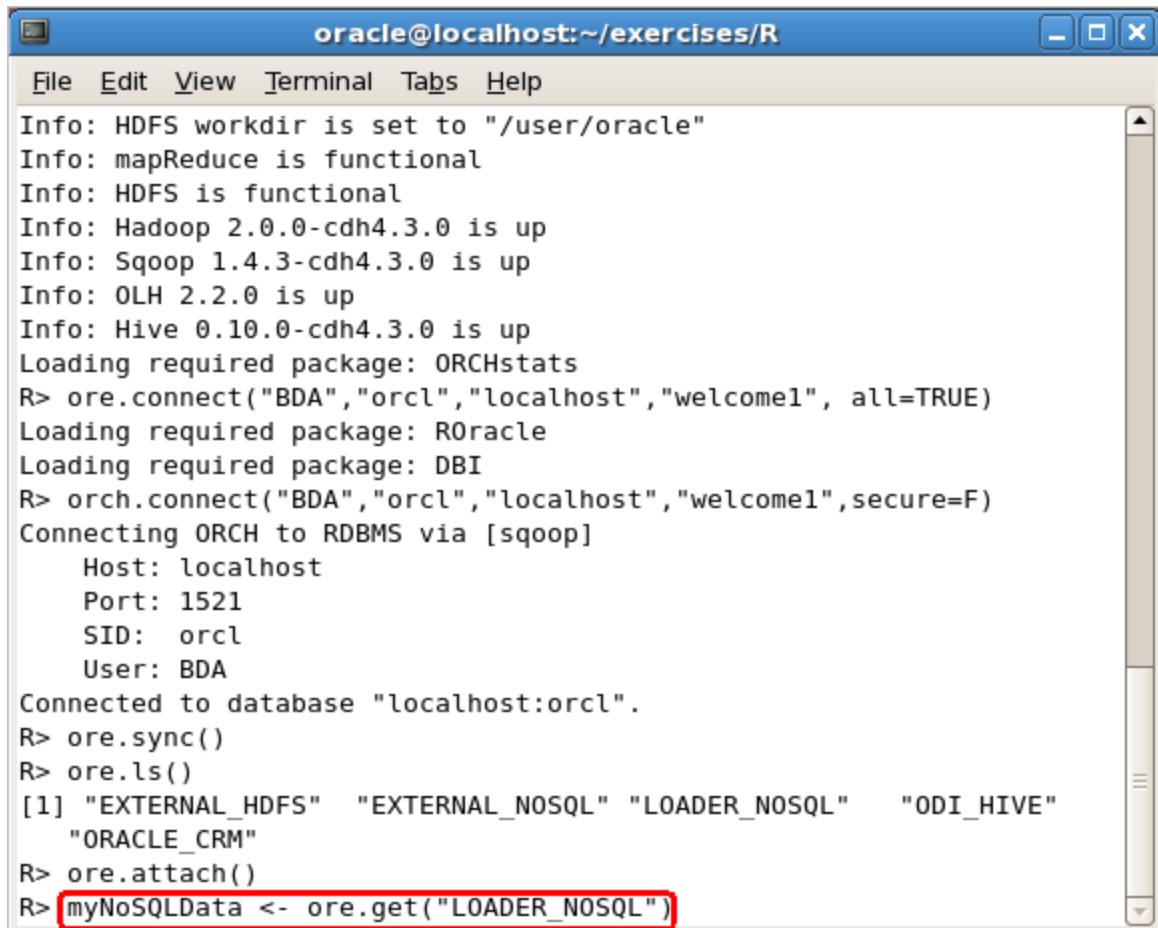
Then press **Enter**



```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
Info: using ORCH HAL v2
Info: HDFS workdir is set to "/user/oracle"
Info: mapReduce is functional
Info: HDFS is functional
Info: Hadoop 2.0.0-cdh4.3.0 is up
Info: Sqoop 1.4.3-cdh4.3.0 is up
Info: OLH 2.2.0 is up
Info: Hive 0.10.0-cdh4.3.0 is up
Loading required package: ORCHstats
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)
Loading required package: ROracle
Loading required package: DBI
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL" "LOADER_NOSQL"      "ODI_HIVE"
     "ORACLE_CRM"
R> ore.attach()
```

11. It's time to get the database table data into R. First, we get the data out of the LOADER_NOSQL database table. Go to the terminal and type:

```
myNoSQLData <- ore.get("LOADER_NOSQL")  
Then press Enter
```



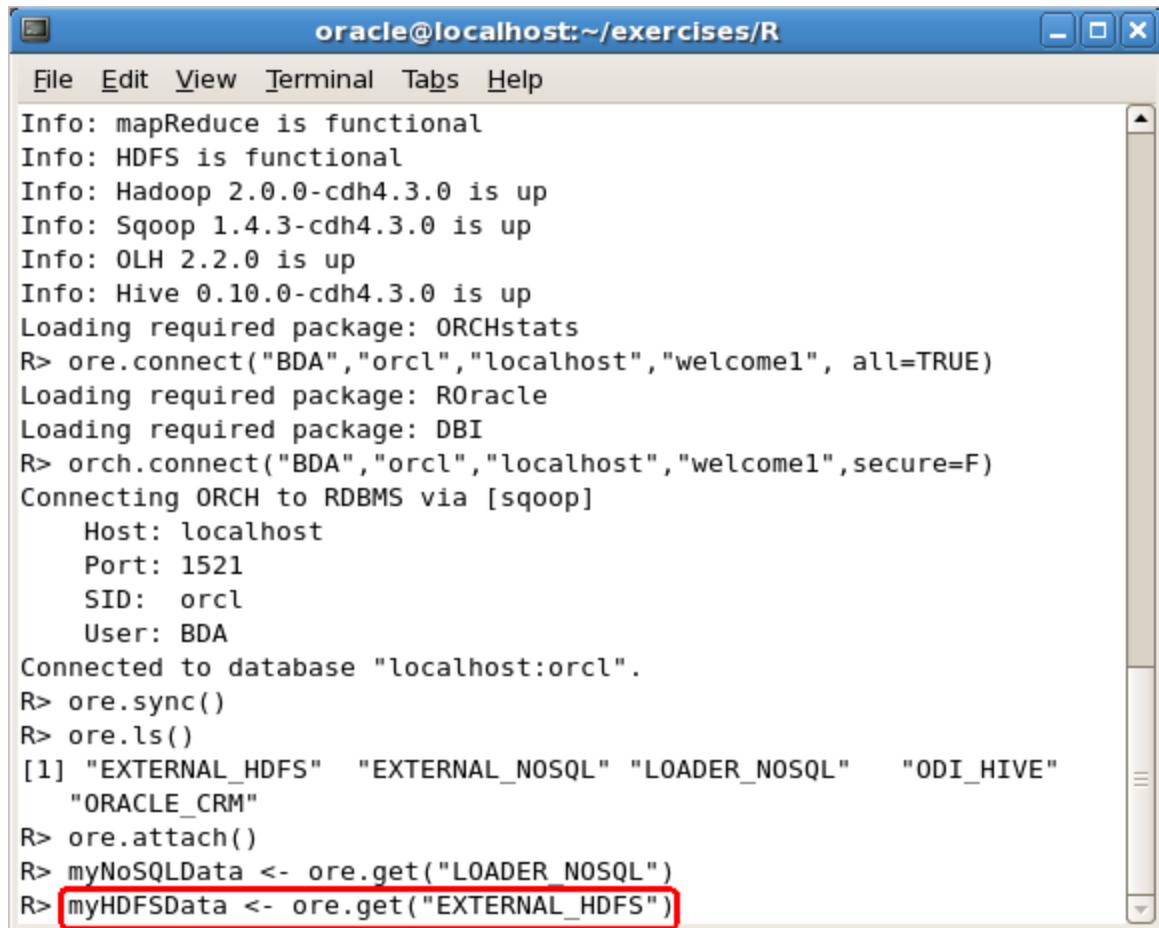
The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following R session output:

```
Info: HDFS workdir is set to "/user/oracle"  
Info: mapReduce is functional  
Info: HDFS is functional  
Info: Hadoop 2.0.0-cdh4.3.0 is up  
Info: Sqoop 1.4.3-cdh4.3.0 is up  
Info: OLH 2.2.0 is up  
Info: Hive 0.10.0-cdh4.3.0 is up  
Loading required package: ORCHstats  
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)  
Loading required package: ROracle  
Loading required package: DBI  
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)  
Connecting ORCH to RDBMS via [sqoop]  
  Host: localhost  
  Port: 1521  
  SID: orcl  
  User: BDA  
Connected to database "localhost:orcl".  
R> ore.sync()  
R> ore.ls()  
[1] "EXTERNAL_HDFS"  "EXTERNAL_NOSQL" "LOADER_NOSQL"    "ODI_HIVE"  
     "ORACLE_CRM"  
R> ore.attach()  
R> myNoSQLData <- ore.get("LOADER_NOSQL")
```

The line "myNoSQLData <- ore.get("LOADER_NOSQL")" is highlighted with a red rectangle.

12. Then, we get the data processed with Pig out of the EXTERNAL_HDFS table. Go to the terminal and type:

```
myHDFSData <- ore.get("EXTERNAL_HDFS")  
Then press Enter
```



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following R session output:

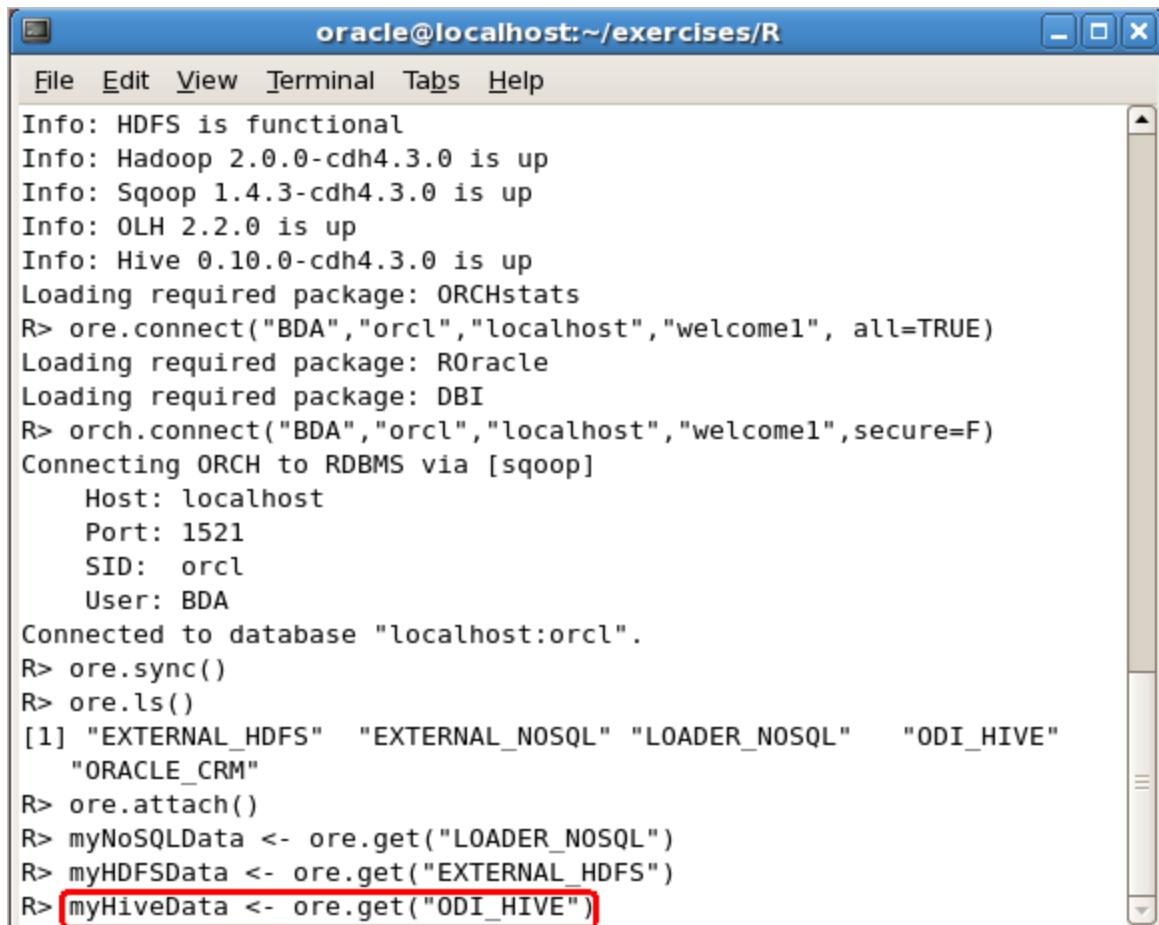
```
Info: mapReduce is functional  
Info: HDFS is functional  
Info: Hadoop 2.0.0-cdh4.3.0 is up  
Info: Sqoop 1.4.3-cdh4.3.0 is up  
Info: OLH 2.2.0 is up  
Info: Hive 0.10.0-cdh4.3.0 is up  
Loading required package: ORCHstats  
R> ore.connect("BDA","orcl","localhost","welcomel", all=TRUE)  
Loading required package: ROracle  
Loading required package: DBI  
R> orch.connect("BDA","orcl","localhost","welcomel",secure=F)  
Connecting ORCH to RDBMS via [sqoop]  
  Host: localhost  
  Port: 1521  
  SID: orcl  
  User: BDA  
Connected to database "localhost:orcl".  
R> ore.sync()  
R> ore.ls()  
[1] "EXTERNAL_HDFS"  "EXTERNAL_NOSQL" "LOADER_NOSQL"    "ODI_HIVE"  
     "ORACLE_CRM"  
R> ore.attach()  
R> myNoSQLData <- ore.get("LOADER_NOSQL")  
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
```

The last two lines of the session, "myHDFSData <- ore.get("EXTERNAL_HDFS")", are highlighted with a red rectangle.

13. Next, we get the data out of the ODI_HIVE table. Go to the terminal and type:

```
myHiveData <- ore.get("ODI_HIVE")
```

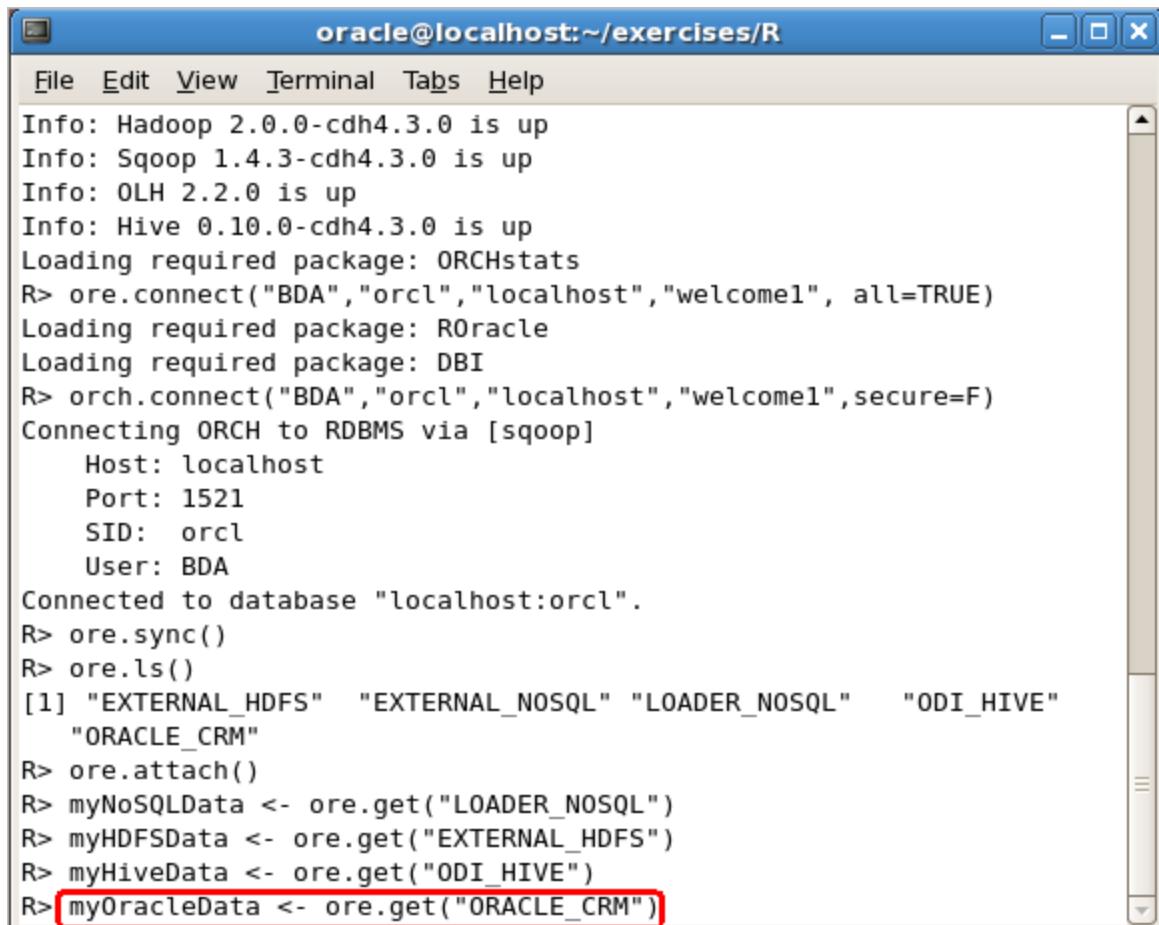
Then press **Enter**



```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
Info: HDFS is functional
Info: Hadoop 2.0.0-cdh4.3.0 is up
Info: Sqoop 1.4.3-cdh4.3.0 is up
Info: OLH 2.2.0 is up
Info: Hive 0.10.0-cdh4.3.0 is up
Loading required package: ORCHstats
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)
Loading required package: ROracle
Loading required package: DBI
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"  "EXTERNAL_NOSQL" "LOADER_NOSQL"      "ODI_HIVE"
  "ORACLE_CRM"
R> ore.attach()
R> myNoSQLData <- ore.get("LOADER_NOSQL")
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
R> myHiveData <- ore.get("ODI_HIVE")
```

14. Finally, we get the data out of the ORACLE_CRM table. Go to the terminal and type:

```
myOracleData <- ore.get("ORACLE_CRM")  
Then press Enter
```



The screenshot shows an R terminal window titled "oracle@localhost:~/exercises/R". The window contains the following R session history:

```
Info: Hadoop 2.0.0-cdh4.3.0 is up  
Info: Sqoop 1.4.3-cdh4.3.0 is up  
Info: OLH 2.2.0 is up  
Info: Hive 0.10.0-cdh4.3.0 is up  
Loading required package: ORCHstats  
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)  
Loading required package: ROracle  
Loading required package: DBI  
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)  
Connecting ORCH to RDBMS via [sqoop]  
  Host: localhost  
  Port: 1521  
  SID: orcl  
  User: BDA  
Connected to database "localhost:orcl".  
R> ore.sync()  
R> ore.ls()  
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL"  "LOADER_NOSQL"    "ODI_HIVE"  
     "ORACLE_CRM"  
R> ore.attach()  
R> myNoSQLData <- ore.get("LOADER_NOSQL")  
R> myHDFSData <- ore.get("EXTERNAL_HDFS")  
R> myHiveData <- ore.get("ODI_HIVE")  
R> myOracleData <- ore.get("ORACLE_CRM")
```

The line "R> myOracleData <- ore.get("ORACLE_CRM")" is highlighted with a red rectangle.

15. We will now proceed to merge the data. First, we merge the NoSQL transactional data with the Hive credit products data. The datasets will be “joined” through the common customer_id column. Go to the terminal and type:

```
myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))
```

Then press **Enter**

The screenshot shows an R terminal window titled "oracle@localhost:~/exercises/R". The window contains the following R session history:

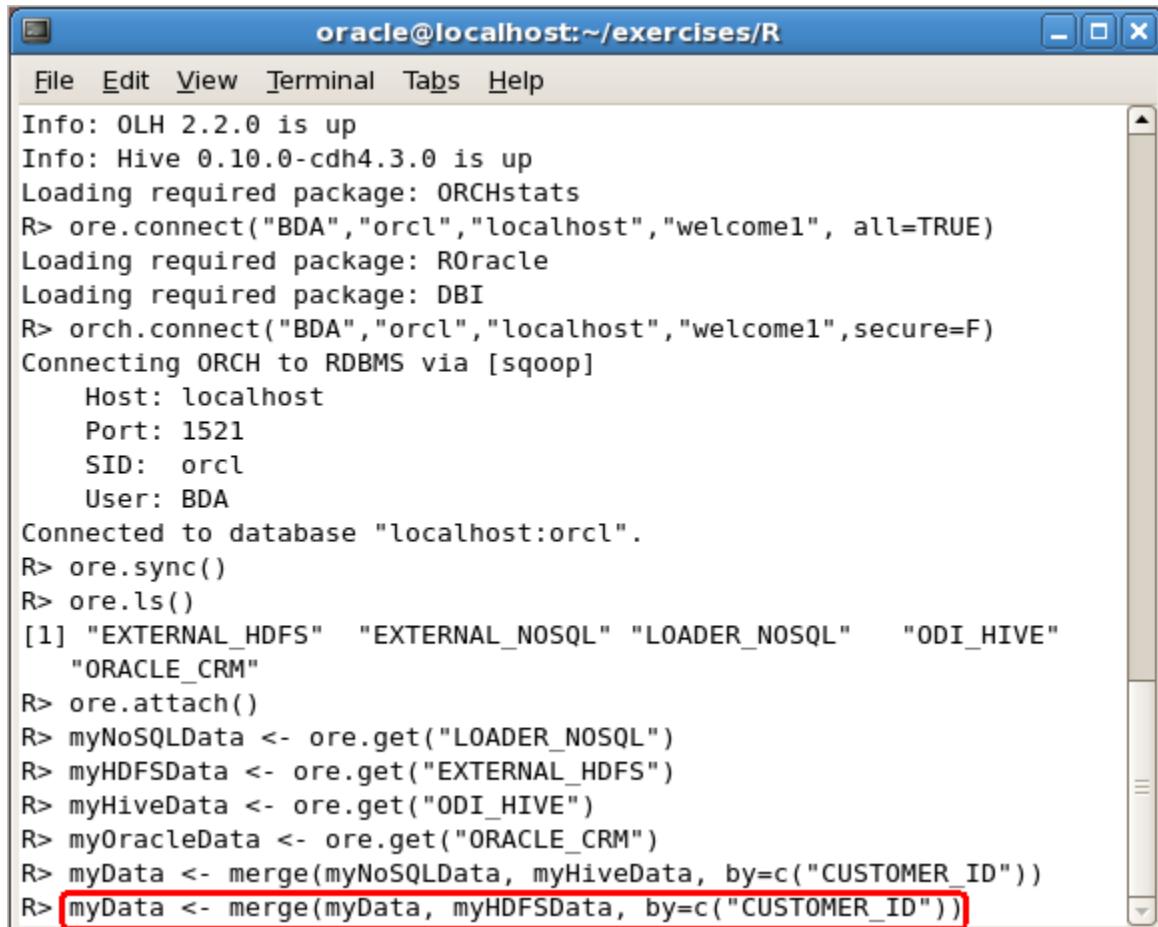
```
Info: Sqoop 1.4.3-cdh4.3.0 is up
Info: OLH 2.2.0 is up
Info: Hive 0.10.0-cdh4.3.0 is up
Loading required package: ORCHstats
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)
Loading required package: ROracle
Loading required package: DBI
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL"  "LOADER_NOSQL"    "ODI_HIVE"
      "ORACLE_CRM"
R> ore.attach()
R> myNoSQLData <- ore.get("LOADER_NOSQL")
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
R> myHiveData <- ore.get("ODI_HIVE")
R> myOracleData <- ore.get("ORACLE_CRM")
R> myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))
```

The last command, `myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))`, is highlighted with a red rectangle.

16. Then, we merge the intermediary dataset obtained from the previous step with the monthly cash accounts data processed with Pig and stored in HDFS. The data sets will be again “joined” through the common customer_id column. Go to the terminal and type:

```
myData <- merge(myData, myHDFSData, by=c("CUSTOMER_ID"))
```

Then press **Enter**

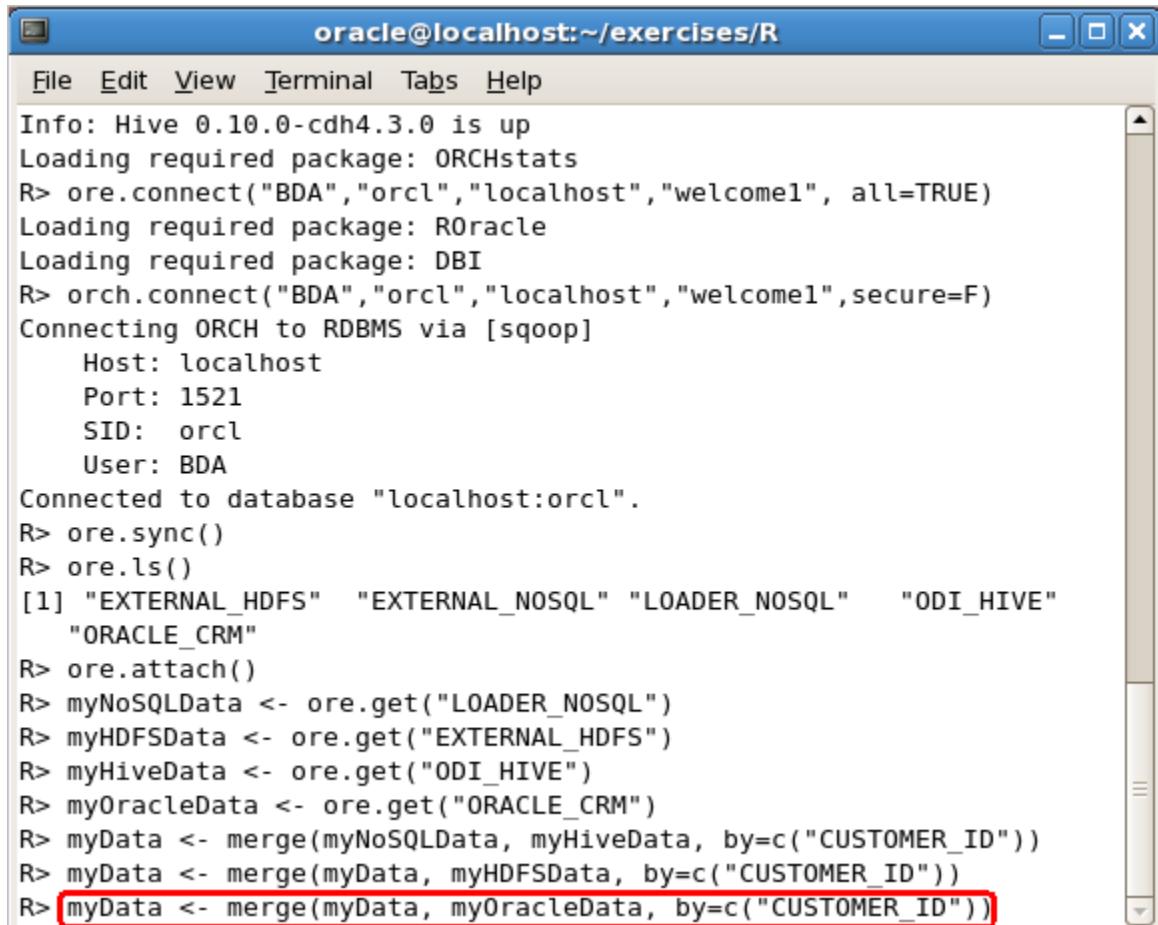


```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
Info: OLH 2.2.0 is up
Info: Hive 0.10.0-cdh4.3.0 is up
Loading required package: ORCHstats
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)
Loading required package: ROracle
Loading required package: DBI
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL"  "LOADER_NOSQL"    "ODI_HIVE"
  "ORACLE_CRM"
R> ore.attach()
R> myNoSQLData <- ore.get("LOADER_NOSQL")
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
R> myHiveData <- ore.get("ODI_HIVE")
R> myOracleData <- ore.get("ORACLE_CRM")
R> myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myHDFSData, by=c("CUSTOMER_ID"))
```

17. Finally, we merge the intermediary dataset obtained from the previous step with the Oracle CRM data. The datasets will be again “joined” through the common customer_id column. Go to the terminal and type:

```
myData <- merge(myData, myOracleData, by=c("CUSTOMER_ID"))
```

Then press **Enter**

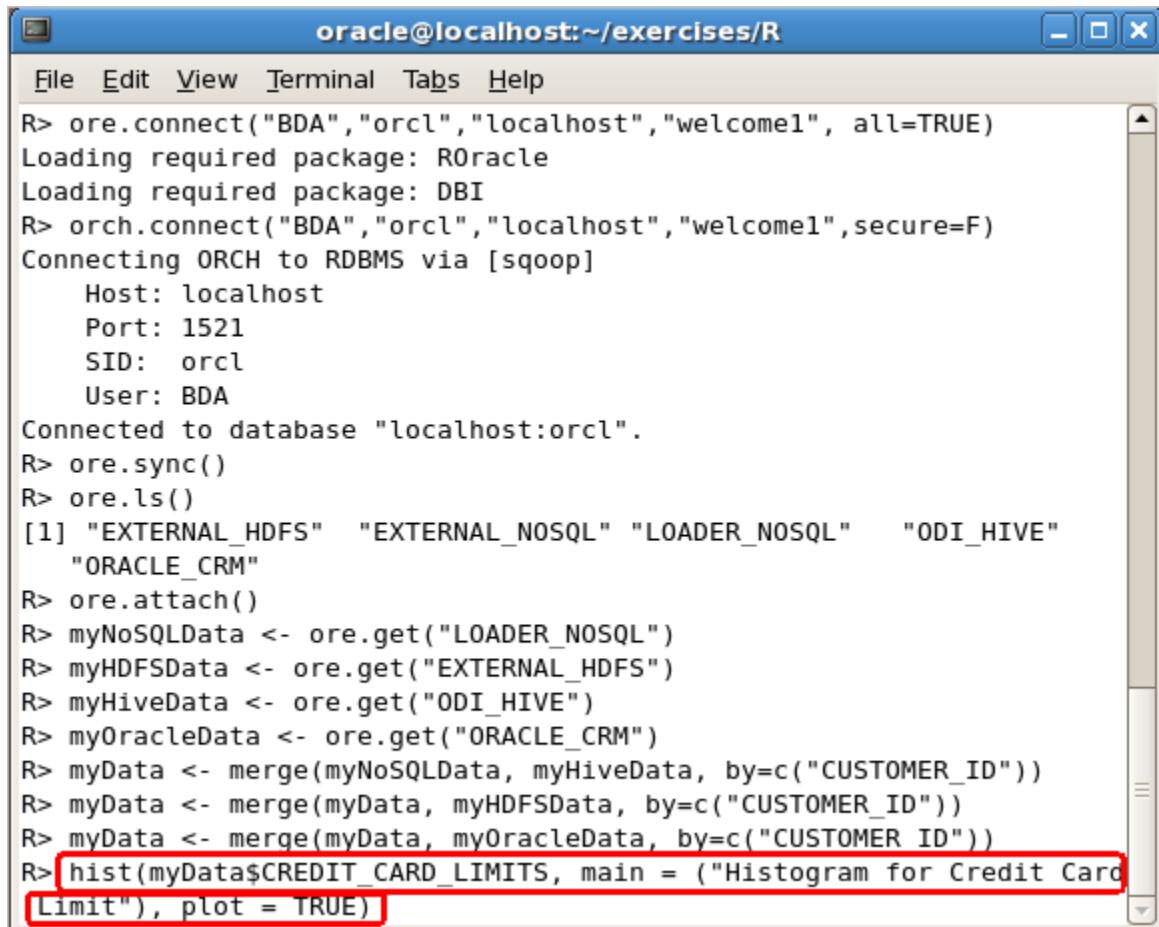


```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
Info: Hive 0.10.0-cdh4.3.0 is up
Loading required package: ORCHstats
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)
Loading required package: ROracle
Loading required package: DBI
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL"  "LOADER_NOSQL"    "ODI_HIVE"
  "ORACLE_CRM"
R> ore.attach()
R> myNoSQLData <- ore.get("LOADER_NOSQL")
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
R> myHiveData <- ore.get("ODI_HIVE")
R> myOracleData <- ore.get("ORACLE_CRM")
R> myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myHDFSData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myOracleData, by=c("CUSTOMER_ID"))
```

18. Let's create a histogram of the CREDIT_CARD_LIMITS column from our data set, to see the distribution of its values. Go to the terminal and type:

```
hist(myData$CREDIT_CARD_LIMITS, main = ("Histogram for Credit Card  
Limit"), plot = TRUE)
```

Then press **Enter**

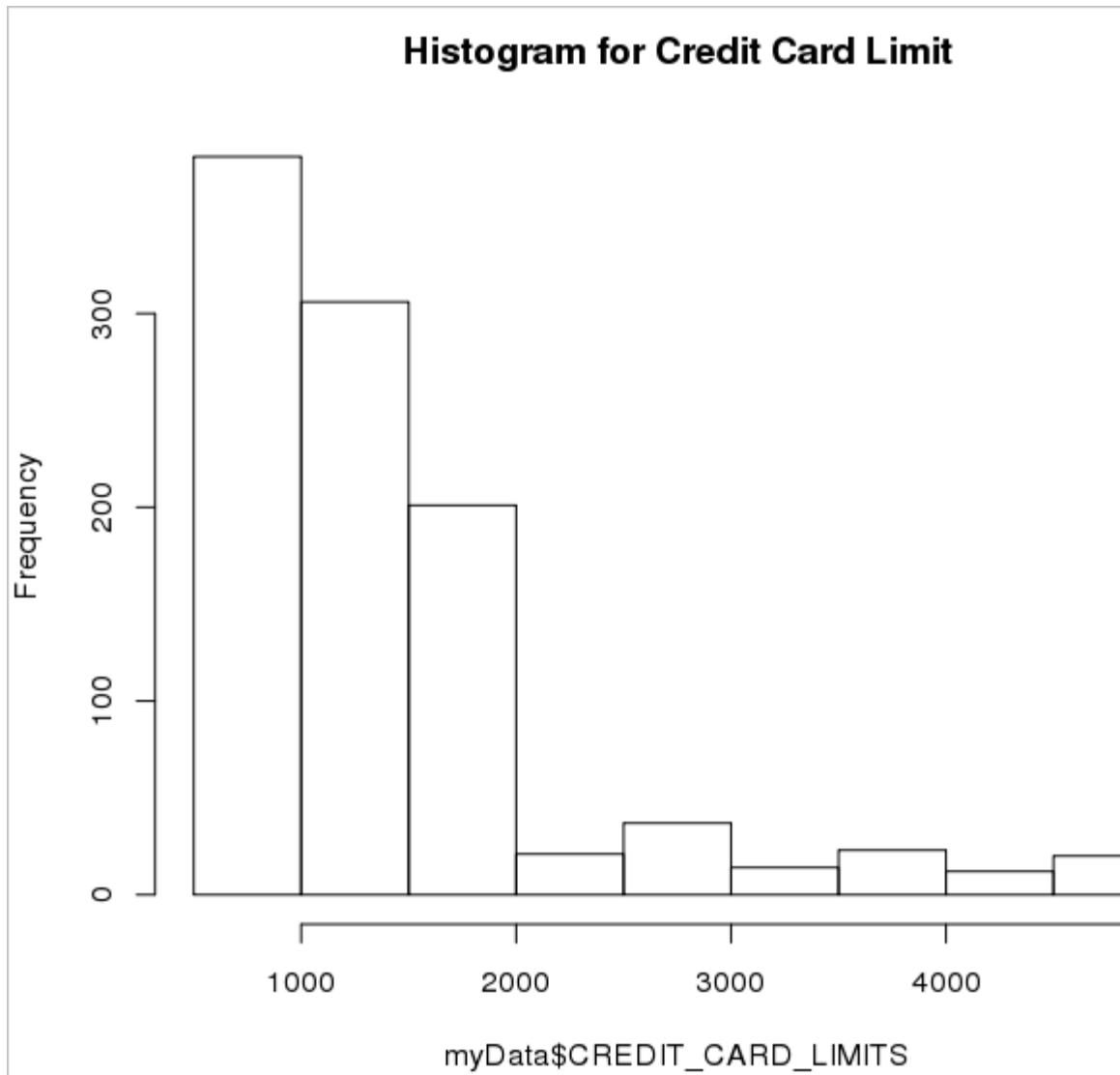


The screenshot shows an R terminal window titled "oracle@localhost:~/exercises/R". The window contains the following R session:

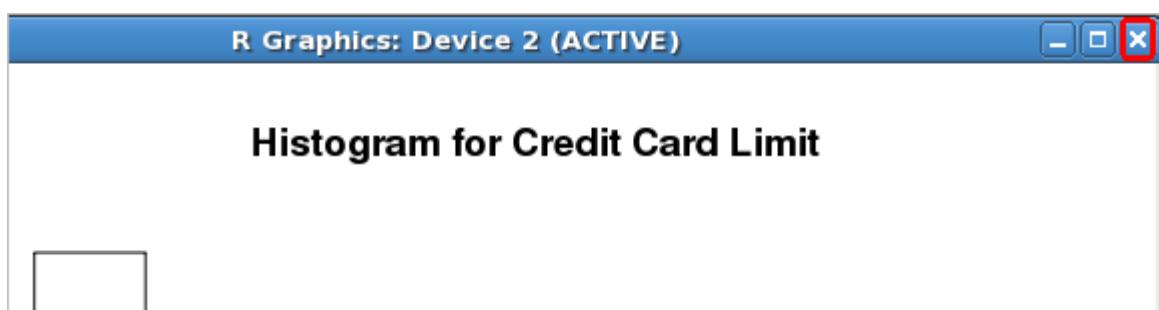
```
File Edit View Terminal Tabs Help  
R> ore.connect("BDA","orcl","localhost","welcome1", all=TRUE)  
Loading required package: ROracle  
Loading required package: DBI  
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)  
Connecting ORCH to RDBMS via [sqoop]  
  Host: localhost  
  Port: 1521  
  SID: orcl  
  User: BDA  
Connected to database "localhost:orcl".  
R> ore.sync()  
R> ore.ls()  
[1] "EXTERNAL_HDFS"  "EXTERNAL_NOSQL" "LOADER_NOSQL"    "ODI_HIVE"  
     "ORACLE_CRM"  
R> ore.attach()  
R> myNoSQLData <- ore.get("LOADER_NOSQL")  
R> myHDFSData <- ore.get("EXTERNAL_HDFS")  
R> my HiveData <- ore.get("ODI_HIVE")  
R> myOracleData <- ore.get("ORACLE_CRM")  
R> myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))  
R> myData <- merge(myData, myHDFSData, by=c("CUSTOMER_ID"))  
R> myData <- merge(myData, myOracleData, by=c("CUSTOMER_ID"))  
R> hist(myData$CREDIT_CARD_LIMITS, main = ("Histogram for Credit Card  
Limit"), plot = TRUE)
```

The command `hist(myData$CREDIT_CARD_LIMITS, main = ("Histogram for Credit Card Limit"), plot = TRUE)` is highlighted with a red rectangle.

A new window appears on the screen, containing a graph of the requested histogram. This graph shows the distribution of values inside the specified column. On the X axis we can see the credit card limit values and on the Y axis we have the frequency for these values.



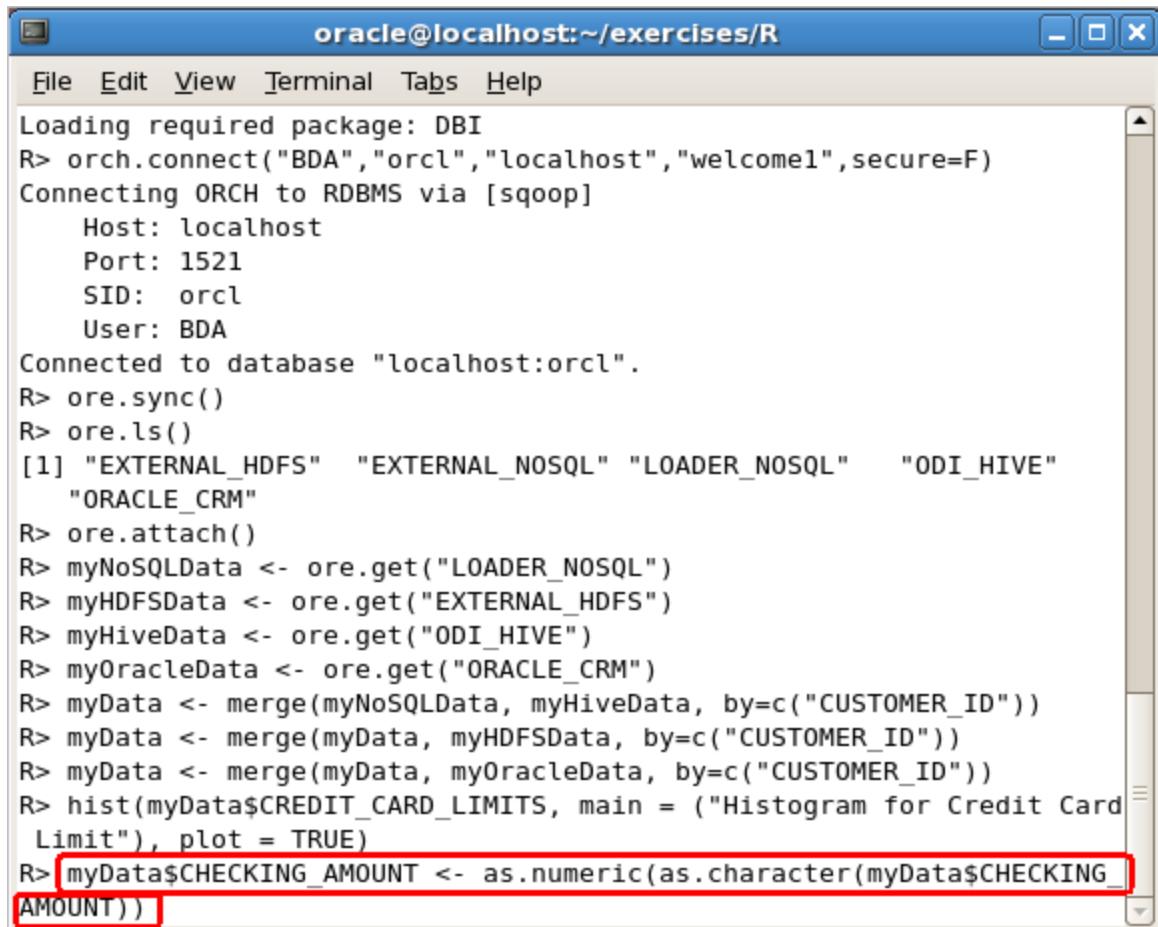
- When you are done evaluating the graph click on the **X** in the right upper corner of the window to close it.



20. All the columns in the EXTERNAL_HDFS table are of type VARCHAR2, and were therefore imported into R as type 'factor' instead of 'numeric'. We will now proceed to change the data type for the 4 columns. First up, CHECKING_AMOUNT. Go to the terminal and type:

```
myData$CHECKING_AMOUNT <-
  as.numeric(as.character(myData$CHECKING_AMOUNT))
```

Then press **Enter**

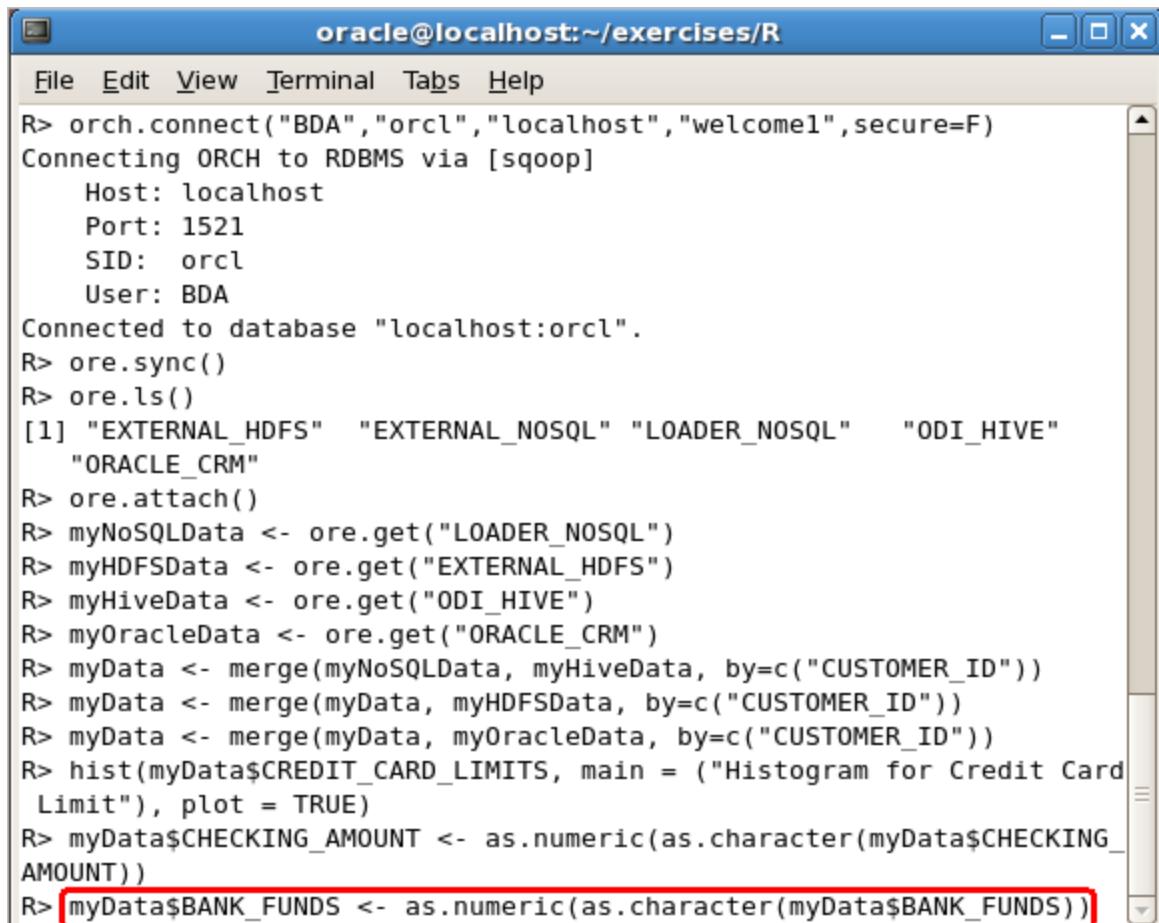


```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
Loading required package: DBI
R> orch.connect("BDA","orcl","localhost","welcome1",secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL"   "LOADER_NOSQL"    "ODI_HIVE"
  "ORACLE_CRM"
R> ore.attach()
R> myNoSQLData <- ore.get("LOADER_NOSQL")
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
R> myHiveData <- ore.get("ODI_HIVE")
R> myOracleData <- ore.get("ORACLE_CRM")
R> myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myHDFSData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myOracleData, by=c("CUSTOMER_ID"))
R> hist(myData$CREDIT_CARD_LIMITS, main = ("Histogram for Credit Card
Limit"), plot = TRUE)
R> myData$CHECKING_AMOUNT <- as.numeric(as.character(myData$CHECKING_
AMOUNT))
```

21. Next column that needs to change its type is BANK_FUNDS. Go to the terminal and type:

```
myData$BANK_FUNDS <- as.numeric(as.character(myData$BANK_FUNDS))
```

Then press **Enter**

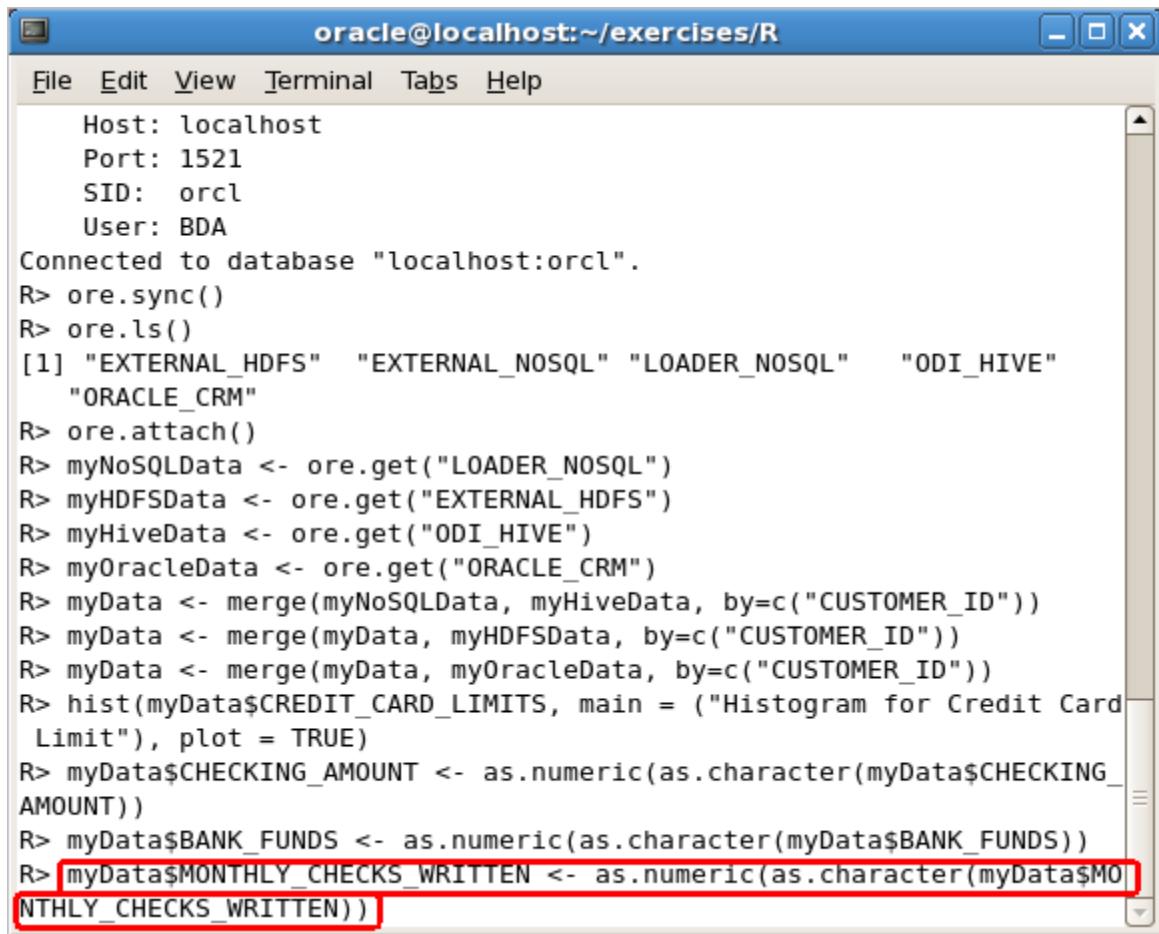


```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
R> orch.connect("BDA", "orcl", "localhost", "welcome1", secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL" "LOADER_NOSQL"      "ODI_HIVE"
"ORACLE_CRM"
R> ore.attach()
R> myNoSQLData <- ore.get("LOADER_NOSQL")
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
R> myHiveData <- ore.get("ODI_HIVE")
R> myOracleData <- ore.get("ORACLE_CRM")
R> myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myHDFSData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myOracleData, by=c("CUSTOMER_ID"))
R> hist(myData$CREDIT_CARD_LIMITS, main = ("Histogram for Credit Card
Limit"), plot = TRUE)
R> myData$CHECKING_AMOUNT <- as.numeric(as.character(myData$CHECKING_
AMOUNT))
R> myData$BANK_FUNDS <- as.numeric(as.character(myData$BANK_FUNDS))
```

22. Next column that needs to change its type is MONTHLY_CHECKS_WRITTEN. Go to the terminal and type:

```
myData$MONTHLY_CHECKS_WRITTEN <-
as.numeric(as.character(myData$MONTHLY_CHECKS_WRITTEN))
```

Then press **Enter**



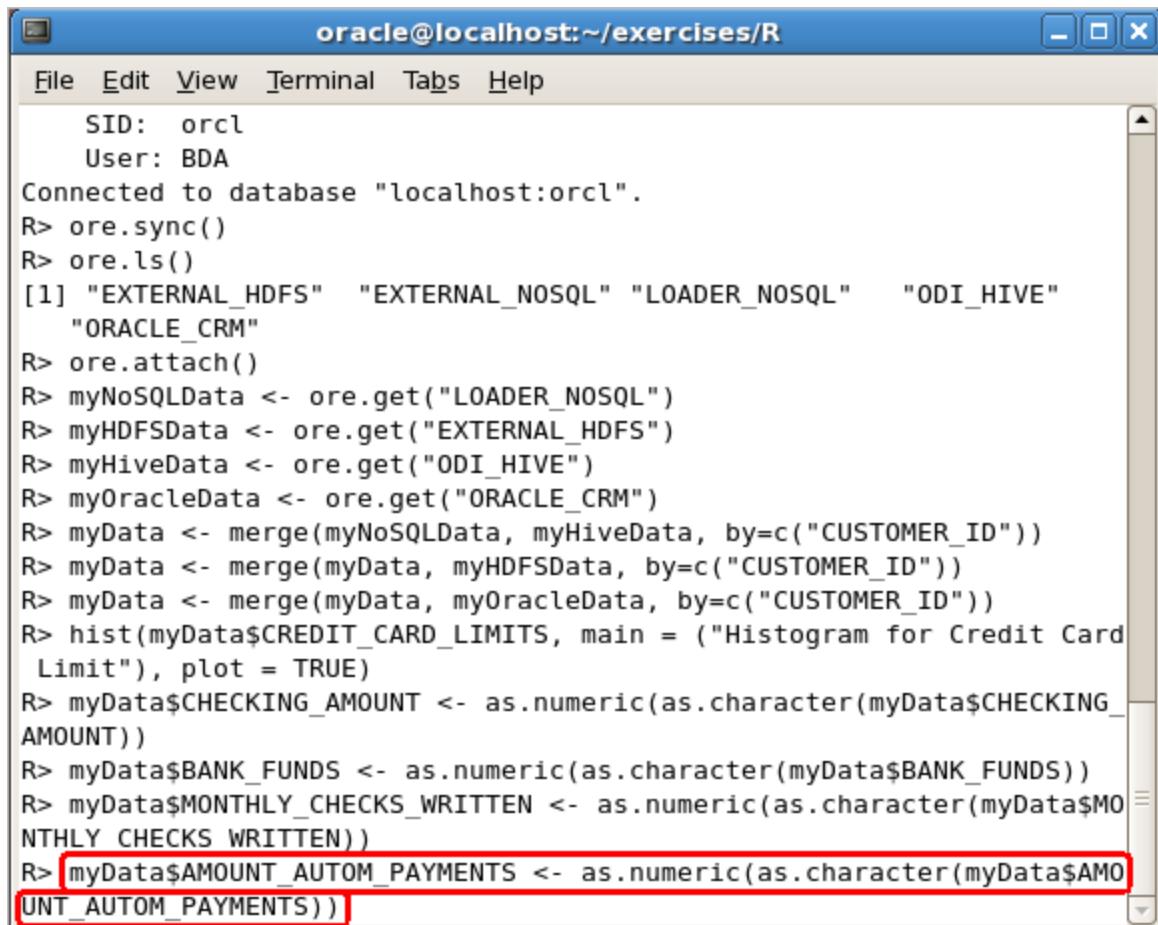
The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains an R session history. The user has connected to a database and loaded several datasets from external sources like HDFS, NOSQL, and ODI_HIVE. They then merge these datasets into a single data frame named "myData". In the final command shown, the user attempts to convert the "MONTHLY_CHECKS_WRITTEN" column from character to numeric type. The line of code is highlighted with a red rectangle.

```
Host: localhost
Port: 1521
SID: orcl
User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL"  "LOADER_NOSQL"    "ODI_HIVE"
"ORACLE_CRM"
R> ore.attach()
R> myNoSQLData <- ore.get("LOADER_NOSQL")
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
R> myHiveData <- ore.get("ODI_HIVE")
R> myOracleData <- ore.get("ORACLE_CRM")
R> myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myHDFSData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myOracleData, by=c("CUSTOMER_ID"))
R> hist(myData$CREDIT_CARD_LIMITS, main = ("Histogram for Credit Card
Limit"), plot = TRUE)
R> myData$CHECKING_AMOUNT <- as.numeric(as.character(myData$CHECKING_
AMOUNT))
R> myData$BANK_FUNDS <- as.numeric(as.character(myData$BANK_FUNDS))
R> myData$MONTHLY_CHECKS_WRITTEN <- as.numeric(as.character(myData$MO
NTHLY_CHECKS_WRITTEN))
```

23. Next column that needs to change its type is AMOUNT_AUTOM_PAYMENTS. Go to the terminal and type:

```
myData$AMOUNT_AUTOM_PAYMENTS <-
as.numeric(as.character(myData$AMOUNT_AUTOM_PAYMENTS))
```

Then press **Enter**

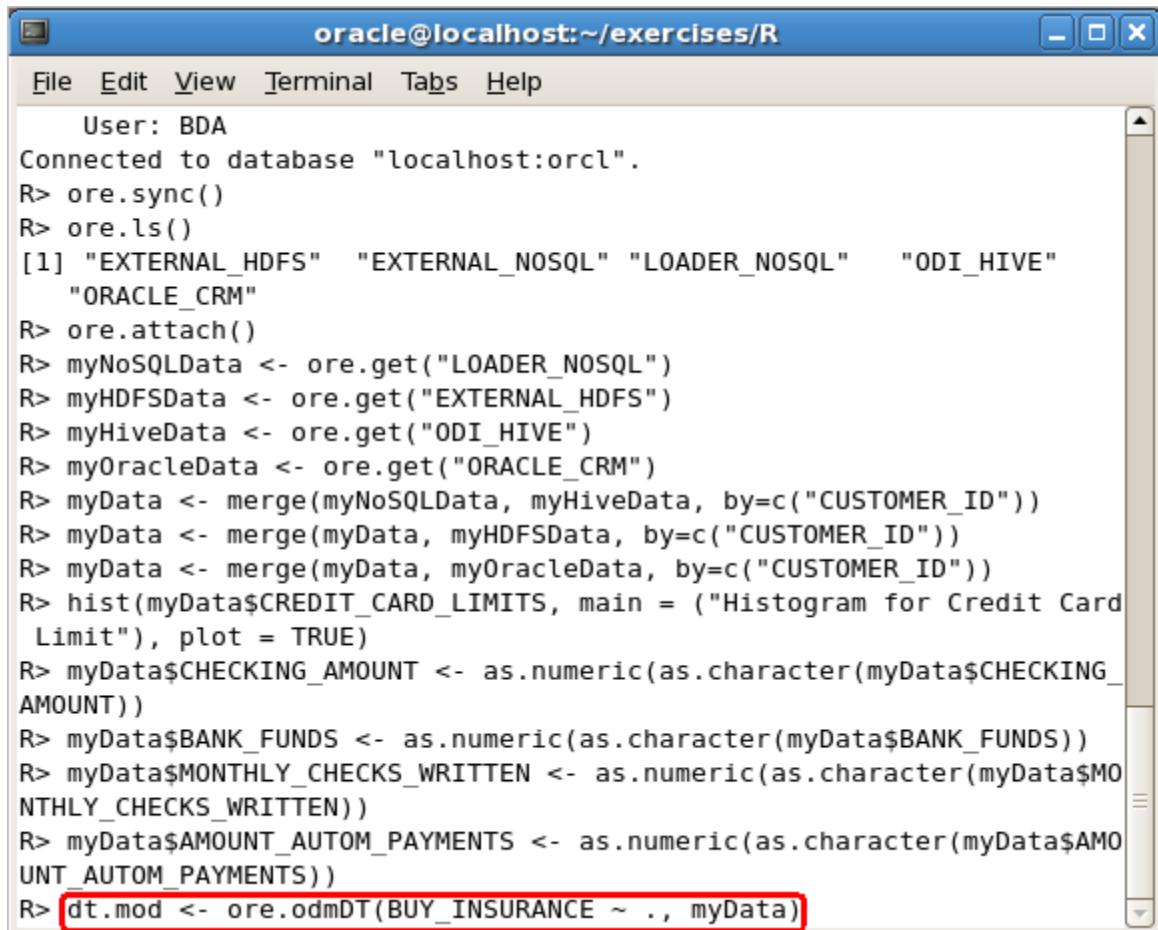


```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
SID: orcl
User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL"  "LOADER_NOSQL"      "ODI_HIVE"
    "ORACLE_CRM"
R> ore.attach()
R> myNoSQLData <- ore.get("LOADER_NOSQL")
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
R> myHiveData <- ore.get("ODI_HIVE")
R> myOracleData <- ore.get("ORACLE_CRM")
R> myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myHDFSData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myOracleData, by=c("CUSTOMER_ID"))
R> hist(myData$CREDIT_CARD_LIMITS, main = ("Histogram for Credit Card
Limit"), plot = TRUE)
R> myData$CHECKING_AMOUNT <- as.numeric(as.character(myData$CHECKING_
AMOUNT))
R> myData$BANK_FUNDS <- as.numeric(as.character(myData$BANK_FUNDS))
R> myData$MONTHLY_CHECKS_WRITTEN <- as.numeric(as.character(myData$MO
NTHLY_CHECKS_WRITTEN))
R> myData$AMOUNT_AUTOM_PAYMENTS <- as.numeric(as.character(myData$AMO
UNT_AUTOM_PAYMENTS))
```

24. Let's run the decision tree algorithm on our data set, to identify the conditions that have to be met for our customers to choose to buy insurance from us. Go to the terminal and type:

```
dt.mod <- ore.odmDT(BUY_INSURANCE ~ ., myData)
```

Then press **Enter**



```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
User: BDA
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"  "EXTERNAL_NOSQL"  "LOADER_NOSQL"    "ODI_HIVE"
   "ORACLE_CRM"
R> ore.attach()
R> myNoSQLData <- ore.get("LOADER_NOSQL")
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
R> myHiveData <- ore.get("ODI_HIVE")
R> myOracleData <- ore.get("ORACLE_CRM")
R> myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myHDFSData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myOracleData, by=c("CUSTOMER_ID"))
R> hist(myData$CREDIT_CARD_LIMITS, main = ("Histogram for Credit Card
Limit"), plot = TRUE)
R> myData$CHECKING_AMOUNT <- as.numeric(as.character(myData$CHECKING_
AMOUNT))
R> myData$BANK_FUNDS <- as.numeric(as.character(myData$BANK_FUNDS))
R> myData$MONTHLY_CHECKS_WRITTEN <- as.numeric(as.character(myData$MO
NTHLY_CHECKS_WRITTEN))
R> myData$AMOUNT_AUTOM_PAYMENTS <- as.numeric(as.character(myData$AMO
UNT_AUTOM_PAYMENTS))
R> dt.mod <- ore.odmDT(BUY_INSURANCE ~ ., myData)
```

25. Let's review the results of the algorithm. Go to the terminal and type:

```
summary(dt.mod)
```

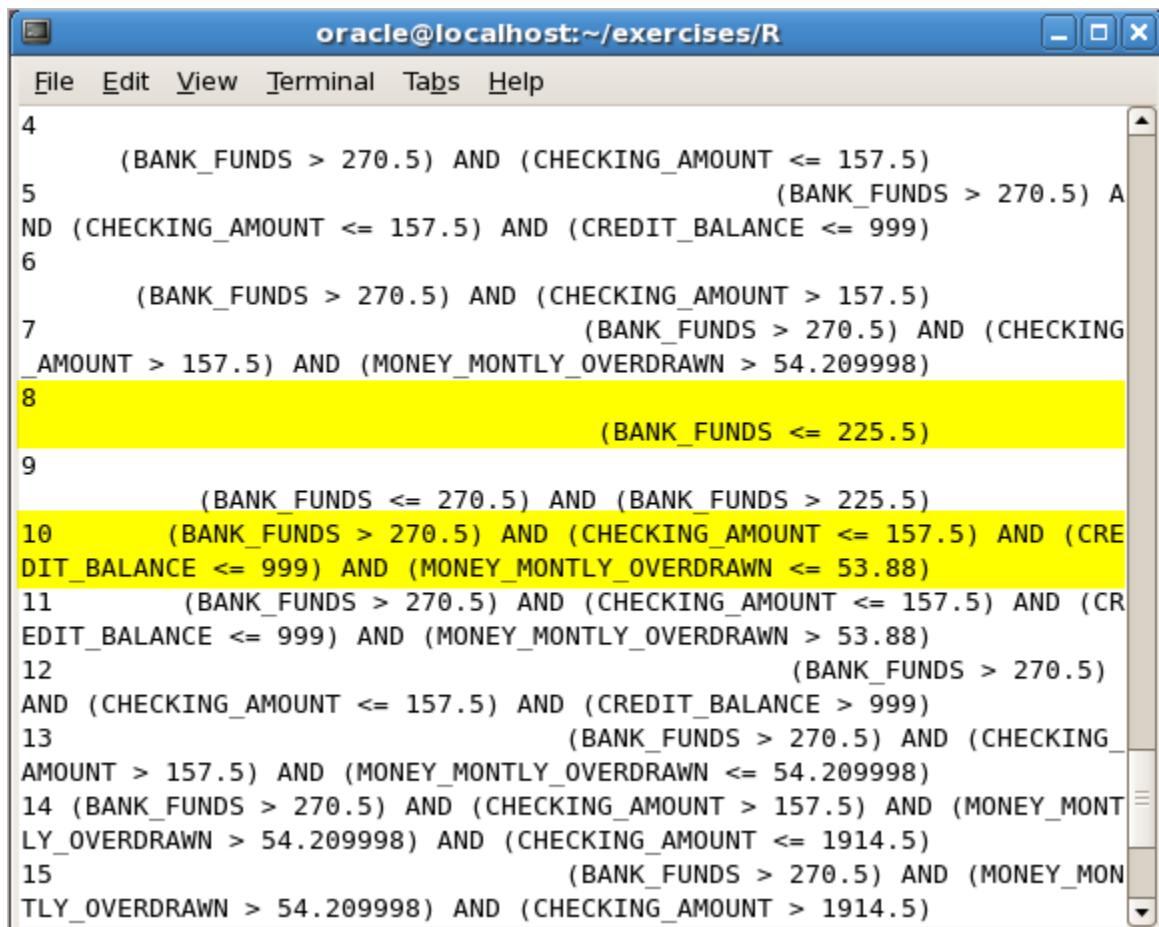
Then press **Enter**

```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
Connected to database "localhost:orcl".
R> ore.sync()
R> ore.ls()
[1] "EXTERNAL_HDFS"   "EXTERNAL_NOSQL" "LOADER_NOSQL"      "ODI_HIVE"
    "ORACLE_CRM"
R> ore.attach()
R> myNoSQLData <- ore.get("LOADER_NOSQL")
R> myHDFSData <- ore.get("EXTERNAL_HDFS")
R> myHiveData <- ore.get("ODI_HIVE")
R> myOracleData <- ore.get("ORACLE_CRM")
R> myData <- merge(myNoSQLData, myHiveData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myHDFSData, by=c("CUSTOMER_ID"))
R> myData <- merge(myData, myOracleData, by=c("CUSTOMER_ID"))
R> hist(myData$CREDIT_CARD_LIMITS, main = ("Histogram for Credit Card
Limit"), plot = TRUE)
R> myData$CHECKING_AMOUNT <- as.numeric(as.character(myData$CHECKING_
AMOUNT))
R> myData$BANK_FUNDS <- as.numeric(as.character(myData$BANK_FUNDS))
R> myData$MONTHLY_CHECKS_WRITTEN <- as.numeric(as.character(myData$MO
NTHLY_CHECKS_WRITTEN))
R> myData$AMOUNT_AUTOM_PAYMENTS <- as.numeric(as.character(myData$AMO
UNT_AUTOM_PAYMENTS))
R> dt.mod <- ore.odmDT(BUY_INSURANCE ~ ., myData)
R> summary(dt.mod)
```

The clusters created by the decision tree algorithm and the predictions for each one are illustrated in the image below. The customers in cluster 8 will most probably refuse to buy insurance from our company, whereas the customers from cluster 10 will most probably buy it.

```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
2      0      1      429      No          (BANK_FUNDS <=
270.5)      (N_TRANS_TELLER <= 1.5)
3      0      2      586      No          (BANK_FUNDS >
270.5)      (N_TRANS_TELLER > 1.5)
4      2      3      351      Yes         (CHECKING_AMOUNT <=
157.5)      (AMOUNT_AUTOM_PAYMENTS <= 3726)
5      3      4      322      Yes         (CREDIT_BALANCE
<= 999)      (AMOUNT_AUTOM_PAYMENTS <= 9145)
6      2      5      235      No          (CHECKING_AMOUNT >
157.5)      (AMOUNT_AUTOM_PAYMENTS > 3726)
7      5      6      52       No          (MONEY_MONTLY_OVERDRAWN > 54.
209998)      (N_TRANS_ATM > 4.5)
8      1      7      402      No          (BANK_FUNDS <=
225.5)      (N_TRANS_TELLER <= 1.5)
9      1      8      27       No          (BANK_FUNDS >
225.5)      (N_TRANS_TELLER > 1.5)
10     4      9      205      Yes         (MONEY_MONTLY_OVERDRAWN <=
53.88)      (N_TRANS_ATM <= 4.5)
11     4      10     117      Yes         (MONEY_MONTLY_OVERDRAWN >
53.88)      (N_TRANS_ATM > 4.5)
12     3      11     29       No          (CREDIT_BALANCE
> 999)      (AMOUNT_AUTOM_PAYMENTS > 9145)
13     5      12     183      No          (MONEY_MONTLY_OVERDRAWN <= 54.
209998)      (N_TRANS_ATM <= 4.5)
```

This screenshot, associated with the previous one, illustrates the conditions that define the clusters. Customers with bank funds less than 225.5 (belonging to cluster 8) will most likely refuse to buy insurance, whereas customers with bank funds greater than 270.5, a checking amount less than 157.5, a credit balance less than 999 and a sum of monthly overdrawn money less than 53.88 (belonging to cluster 10) will most likely buy insurance.

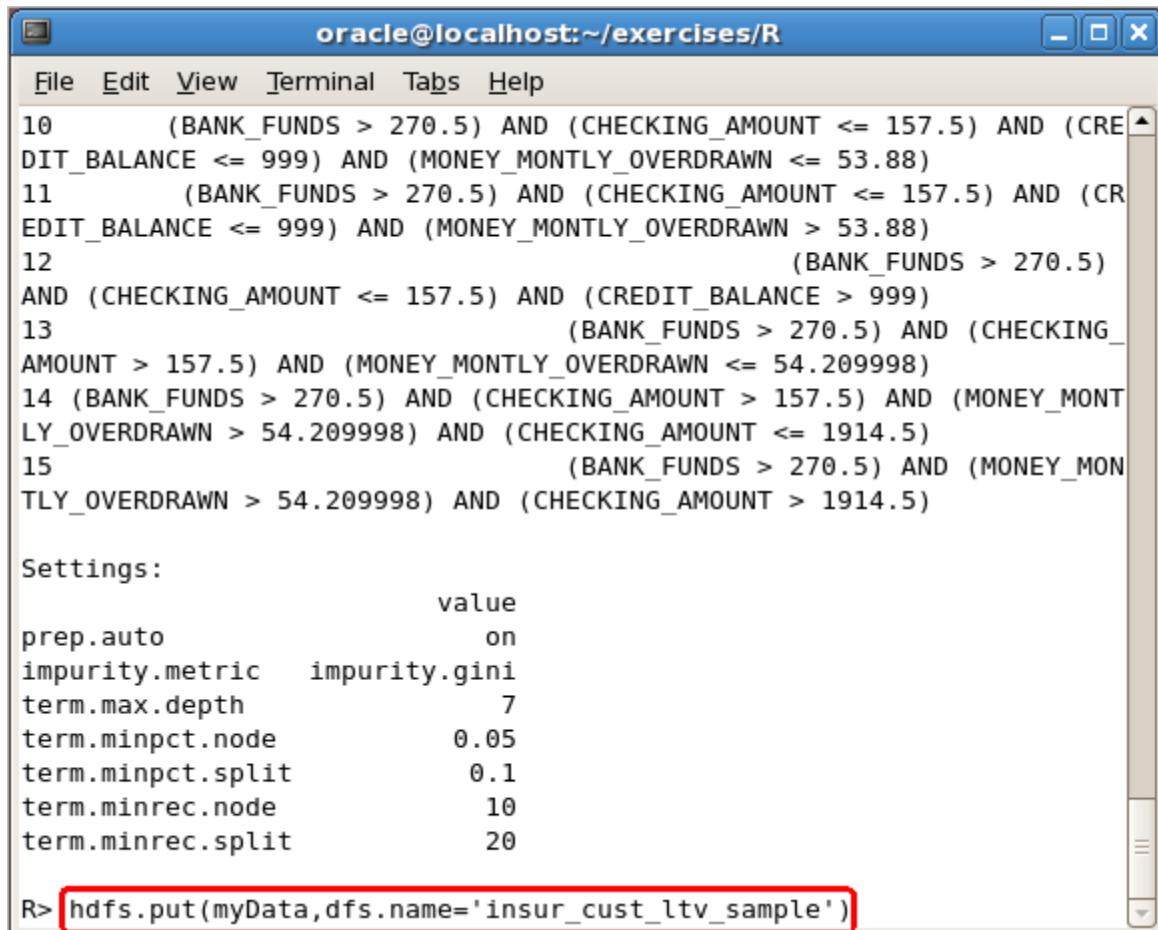


The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains R code for defining clusters based on customer financial characteristics. The code uses logical AND conditions to group customers into different clusters. Clusters 8 and 10 are highlighted in yellow. Cluster 8 is defined by (BANK_FUNDS <= 225.5). Cluster 10 is defined by (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT <= 157.5) AND (CREDIT_BALANCE <= 999) AND (MONEY_MONTLY_OVERDRAWN <= 53.88).

```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
4      (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT <= 157.5)
5                                         (BANK_FUNDS > 270.5) A
ND (CHECKING_AMOUNT <= 157.5) AND (CREDIT_BALANCE <= 999)
6      (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT > 157.5)
7                                         (BANK_FUNDS > 270.5) AND (CHECKING_
_AMOUNT > 157.5) AND (MONEY_MONTLY_OVERDRAWN > 54.209998)
8                                         (BANK_FUNDS <= 225.5)
9
10                                         (BANK_FUNDS <= 270.5) AND (BANK_FUNDS > 225.5)
10                                         (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT <= 157.5) AND (CRE
DIT_BALANCE <= 999) AND (MONEY_MONTLY_OVERDRAWN <= 53.88)
11                                         (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT <= 157.5) AND (CR
EDIT_BALANCE <= 999) AND (MONEY_MONTLY_OVERDRAWN > 53.88)
12                                         (BANK_FUNDS > 270.5)
AND (CHECKING_AMOUNT <= 157.5) AND (CREDIT_BALANCE > 999)
13                                         (BANK_FUNDS > 270.5) AND (CHECKING_
_AMOUNT > 157.5) AND (MONEY_MONTLY_OVERDRAWN <= 54.209998)
14 (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT > 157.5) AND (MONEY_MONT
LY_OVERDRAWN > 54.209998) AND (CHECKING_AMOUNT <= 1914.5)
15                                         (BANK_FUNDS > 270.5) AND (MONEY_MON
TLY_OVERDRAWN > 54.209998) AND (CHECKING_AMOUNT > 1914.5)
```

26. Let's also save the merged data into HDFS in case we need it in the future. Go to the terminal and type:

```
hdfs.put(myData,dfs.name='insur_cust_ltv_sample')  
Then press Enter
```



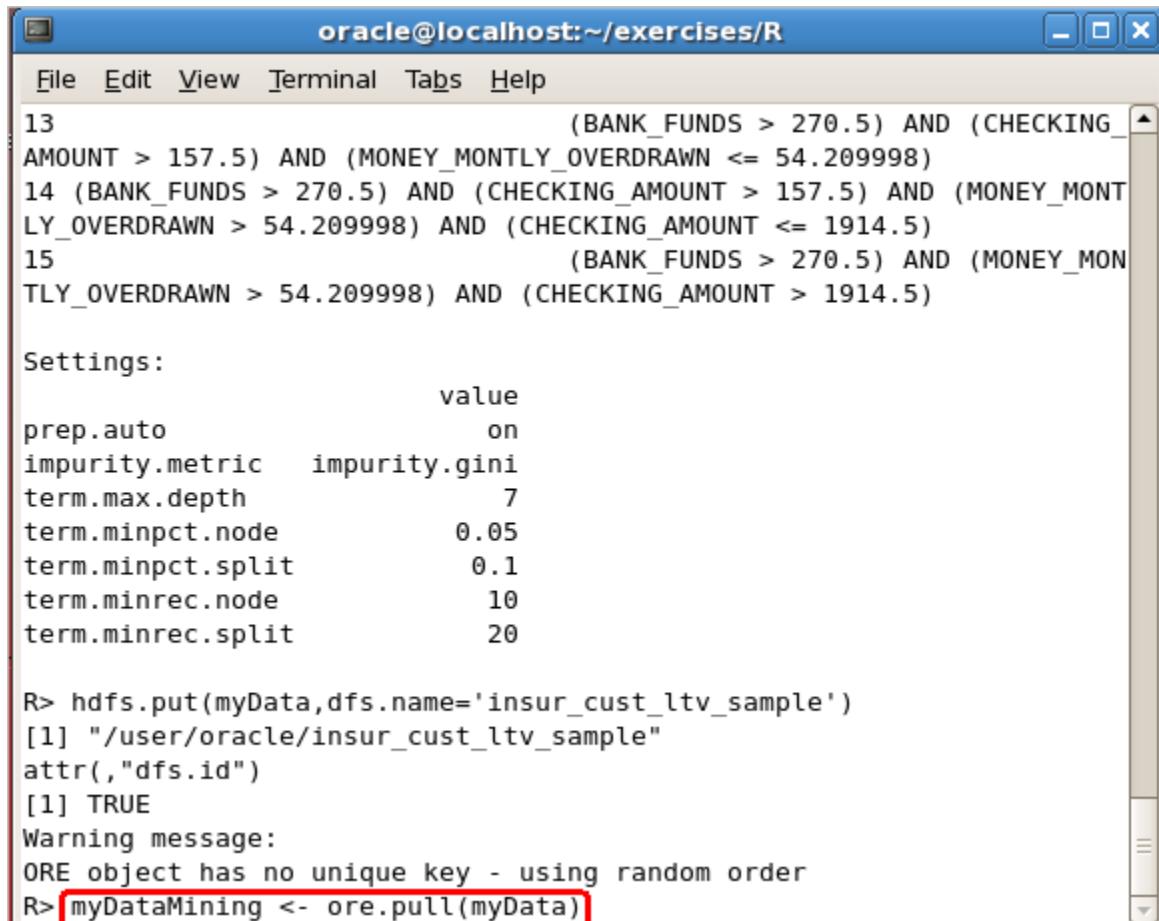
The screenshot shows an R terminal window titled "oracle@localhost:~/exercises/R". The window contains the following text:

```
File Edit View Terminal Tabs Help  
10      (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT <= 157.5) AND (CREDIT_BALANCE <= 999) AND (MONEY_MONTLY_OVERDRAWN <= 53.88)  
11      (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT <= 157.5) AND (CREDIT_BALANCE <= 999) AND (MONEY_MONTLY_OVERDRAWN > 53.88)  
12          (BANK_FUNDS > 270.5)  
AND (CHECKING_AMOUNT <= 157.5) AND (CREDIT_BALANCE > 999)  
13          (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT > 157.5) AND (MONEY_MONTLY_OVERDRAWN <= 54.209998)  
14 (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT > 157.5) AND (MONEY_MONTLY_OVERDRAWN > 54.209998) AND (CHECKING_AMOUNT <= 1914.5)  
15          (BANK_FUNDS > 270.5) AND (MONEY_MONTLY_OVERDRAWN > 54.209998) AND (CHECKING_AMOUNT > 1914.5)  
  
Settings:  
           value  
prep.auto      on  
impurity.metric impurity.gini  
term.max.depth    7  
term.minpct.node   0.05  
term.minpct.split   0.1  
term.minrec.node    10  
term.minrec.split   20  
  
R> hdfs.put(myData,dfs.name='insur_cust_ltv_sample')
```

Do note that you might be seeing a warning about "no unique key". That is an ignorable warning, as the order of the data does not matter.

27. Before continuing this exercise, we have to pull the data set into a local R variable. Go to the terminal and type:

```
myDataMining <- ore.pull(myData)  
Then press Enter
```



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following R session:

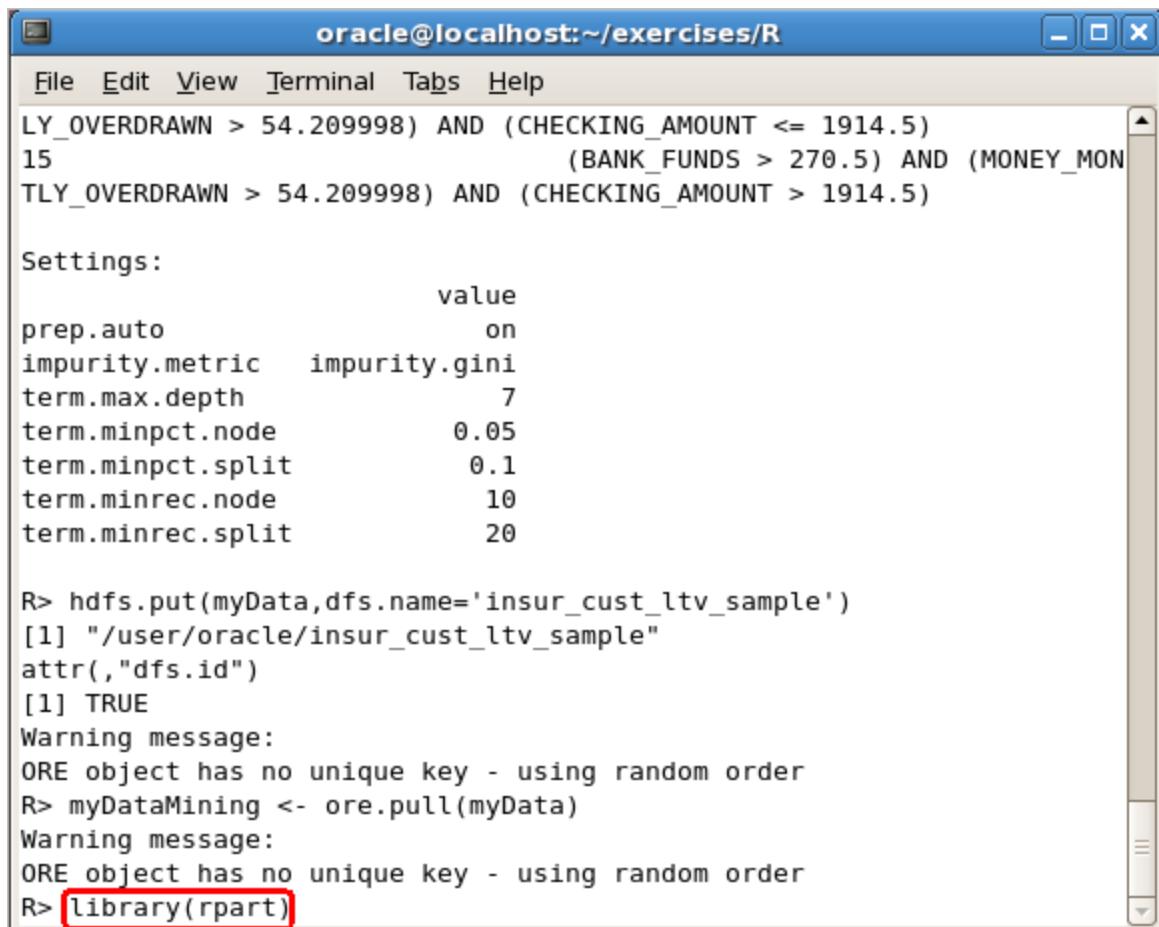
```
File Edit View Terminal Tabs Help  
13          (BANK_FUNDS > 270.5) AND (CHECKING_  
AMOUNT > 157.5) AND (MONEY_MONTLY_OVERDRAWN <= 54.209998)  
14 (BANK_FUNDS > 270.5) AND (CHECKING_AMOUNT > 157.5) AND (MONEY_MONT  
LY_OVERDRAWN > 54.209998) AND (CHECKING_AMOUNT <= 1914.5)  
15          (BANK_FUNDS > 270.5) AND (MONEY_MON  
TLY_OVERDRAWN > 54.209998) AND (CHECKING_AMOUNT > 1914.5)  
  
Settings:  
          value  
prep.auto      on  
impurity.metric impurity.gini  
term.max.depth    7  
term.minpct.node   0.05  
term.minpct.split   0.1  
term.minrec.node    10  
term.minrec.split   20  
  
R> hdfs.put(myData,dfs.name='insur_cust_ltv_sample')  
[1] "/user/oracle/insur_cust_ltv_sample"  
attr(),"dfs.id")  
[1] TRUE  
Warning message:  
ORE object has no unique key - using random order  
R> myDataMining <- ore.pull(myData)
```

Do note that you might be seeing a warning about “no unique key”. That is an ignorable warning, because for the algorithm that we intend to run the order of the data does not matter.

28. Let's now run a simplified version of the decision tree algorithm on the same data set, but this time by using a standard R package called *rpart*. The *rpart* package is mostly used to create regression trees, which are normally used when the value we're trying to predict is numeric, such as a price, for example. For the purpose of this workshop, however, we will use *rpart* to predict whether the customers will buy insurance or not, in other words, we will build a classification tree. Both classification trees and regression trees are types of decision trees. We will choose the variables that will be taken into consideration by the algorithm and the maximum depth of the tree, in order to simplify it. We will also create a graph to illustrate the results of the algorithm. First we need to load the necessary library. Go to the terminal and type:

```
library(rpart)
```

Then press **Enter**



```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
LY_OVERDRAWN > 54.209998) AND (CHECKING_AMOUNT <= 1914.5)
15          (BANK_FUNDS > 270.5) AND (MONEY_MON
TLY_OVERDRAWN > 54.209998) AND (CHECKING_AMOUNT > 1914.5)

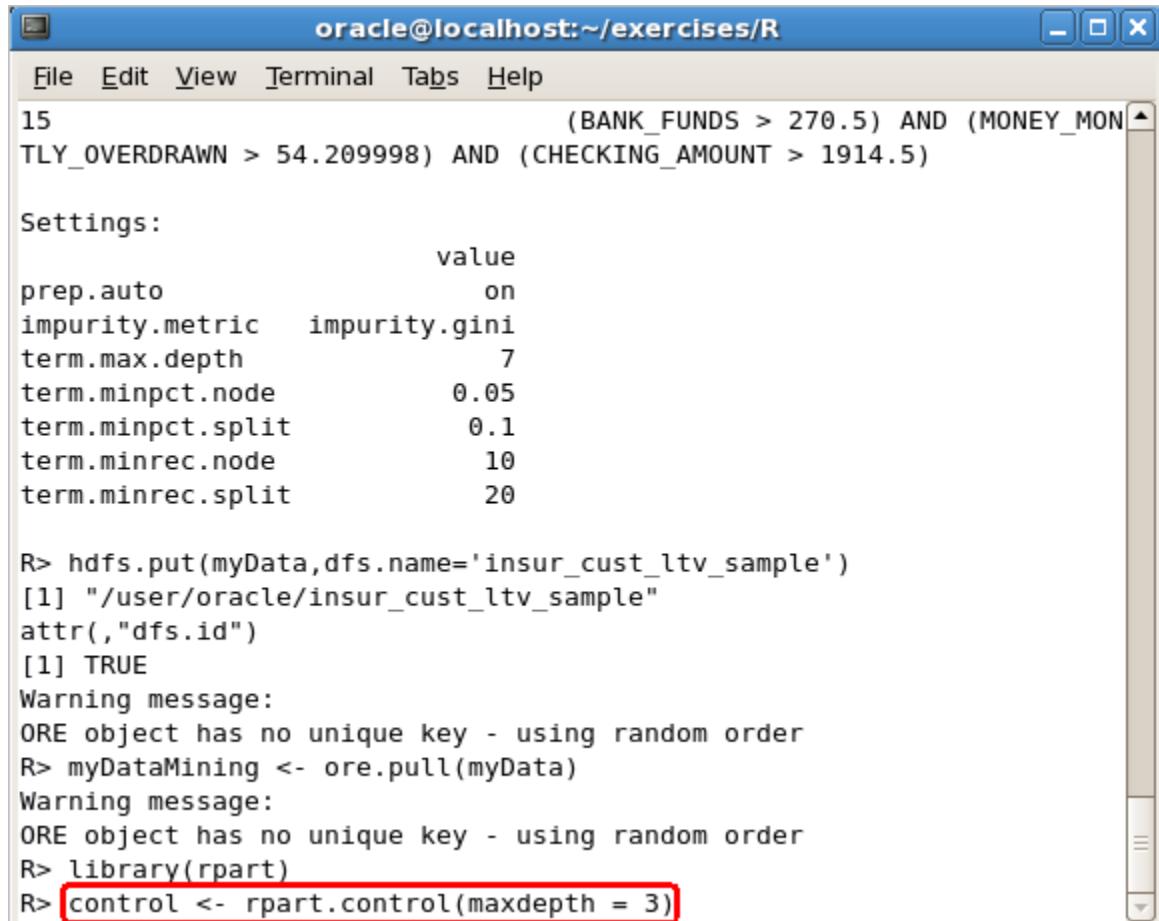
Settings:
      value
prep.auto      on
impurity.metric impurity.gini
term.max.depth    7
term.minpct.node   0.05
term.minpct.split   0.1
term.minrec.node    10
term.minrec.split   20

R> hdfs.put(myData,dfs.name='insur_cust_ltv_sample')
[1] "/user/oracle/insur_cust_ltv_sample"
attr(),"dfs.id")
[1] TRUE
Warning message:
ORE object has no unique key - using random order
R> myDataMining <- ore.pull(myData)
Warning message:
ORE object has no unique key - using random order
R> library(rpart)
```

29. Before running the algorithm, we should make sure we won't have to deal with a very large number of nodes, as the results will be very hard to interpret. We will set the maximum depth of the decision tree to 3. Go to the terminal and type:

```
control <- rpart.control(maxdepth = 3)
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following R session:

```
File Edit View Terminal Tabs Help
15          (BANK_FUNDS > 270.5) AND (MONEY_MON-
TLY_OVERDRAWN > 54.209998) AND (CHECKING_AMOUNT > 1914.5)

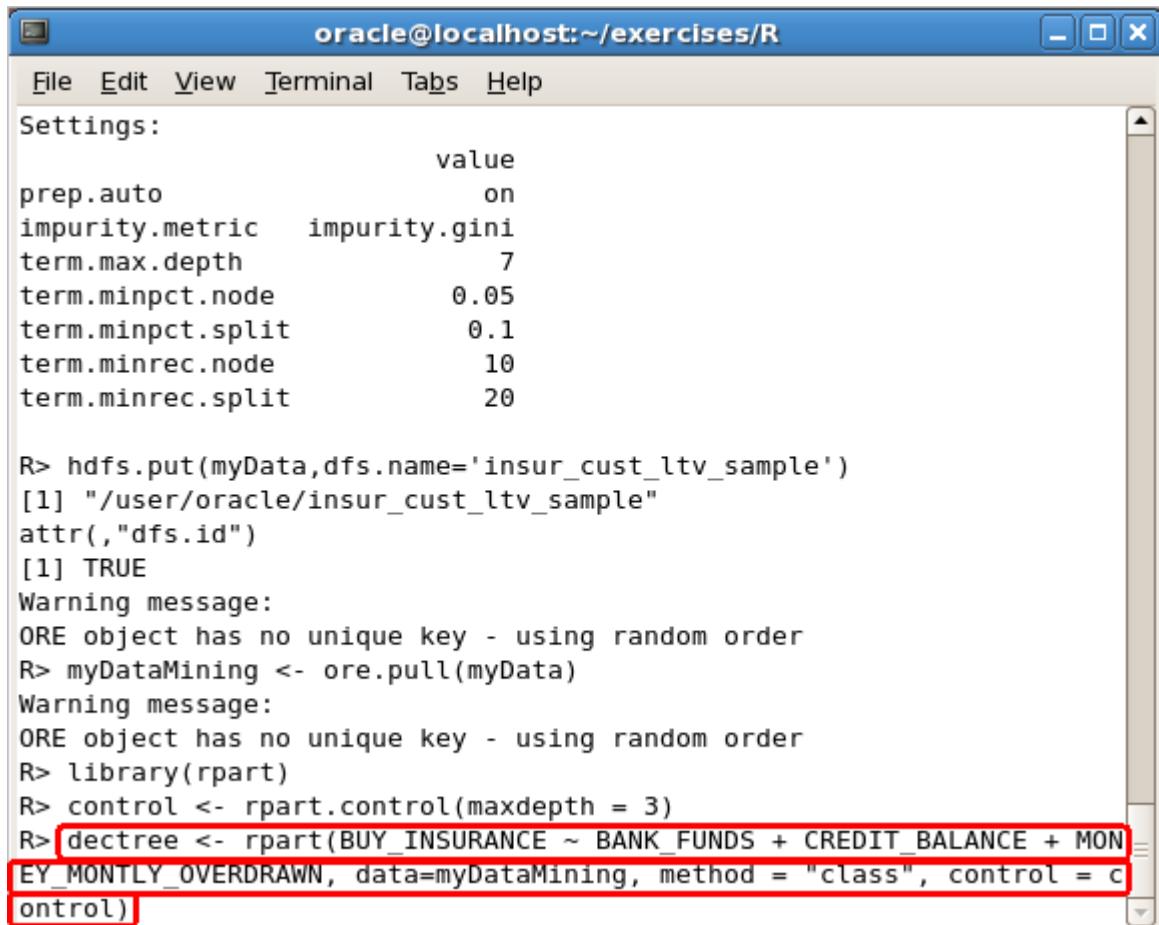
Settings:
      value
prep.auto      on
impurity.metric impurity.gini
term.max.depth    7
term.minpct.node   0.05
term.minpct.split   0.1
term.minrec.node    10
term.minrec.split   20

R> hdfs.put(myData,dfs.name='insur_cust_ltv_sample')
[1] "/user/oracle/insur_cust_ltv_sample"
attr(),"dfs.id")
[1] TRUE
Warning message:
ORE object has no unique key - using random order
R> myDataMining <- ore.pull(myData)
Warning message:
ORE object has no unique key - using random order
R> library(rpart)
R> control <- rpart.control(maxdepth = 3)
```

30. Time to run the algorithm. In the command that runs the algorithm we will specify the variables we want to consider (BANK_FUNDS, CREDIT_BALANCE and MONEY_MONTLY_OVERDRAWN) and the variable we want to predict (BUY_INSURANCE). By specifying the exact variables we want to consider in the algorithm we will simplify it a little, so that it's easier to interpret. The "class" method is being used, because we are trying to create a classification tree. Notice that the control variable that was defined earlier, specifying the maximum depth of the decision tree, is also included in the command. Go to the terminal and type:

```
dectree <- rpart(BUY_INSURANCE ~ BANK_FUNDS + CREDIT_BALANCE +
MONEY_MONTLY_OVERDRAWN, data=myDataMining, method = "class", control =
control)
```

Then press **Enter**



The screenshot shows an R terminal window titled "oracle@localhost:~/exercises/R". The window contains the following R session:

```
File Edit View Terminal Tabs Help
Settings:
      value
prep.auto          on
impurity.metric   impurity.gini
term.max.depth     7
term.minpct.node   0.05
term.minpct.split  0.1
term.minrec.node   10
term.minrec.split  20

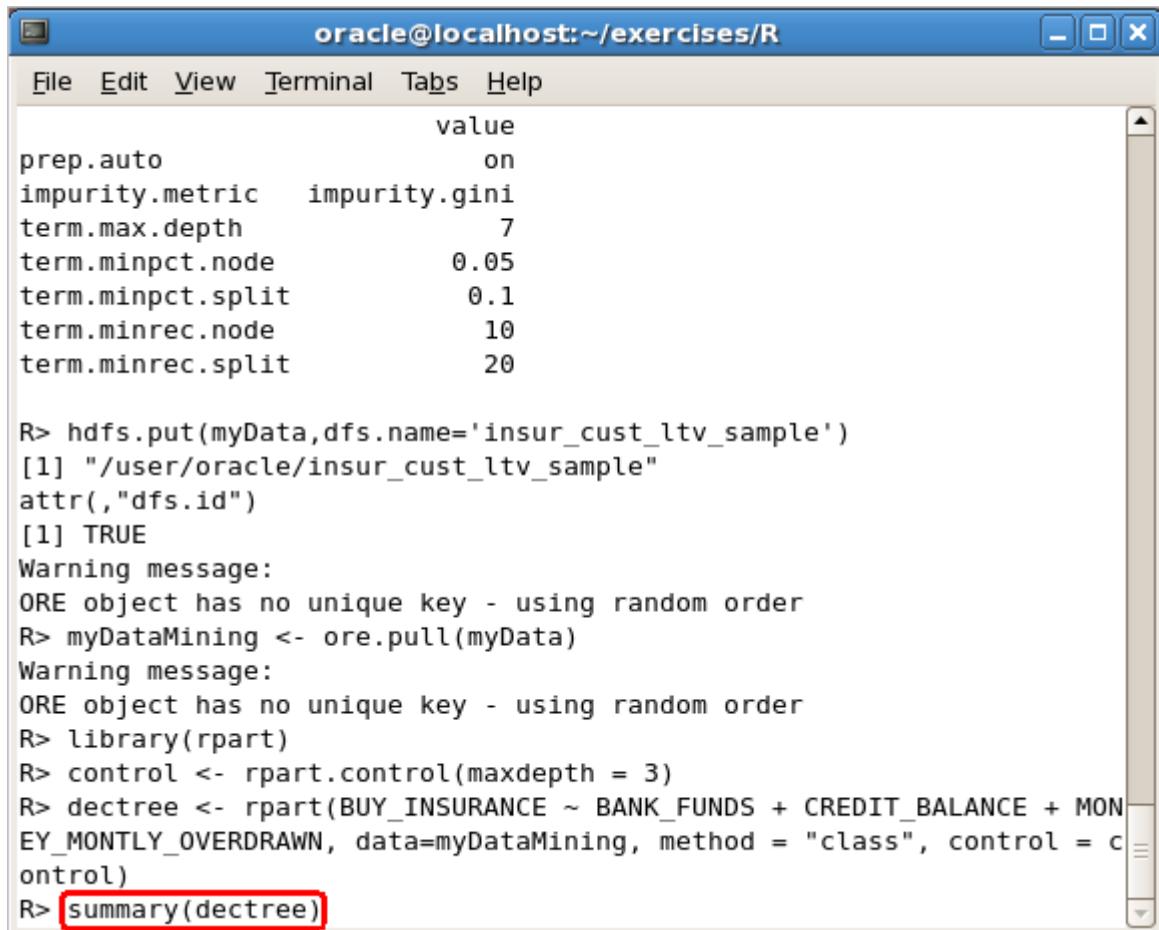
R> hdfs.put(myData,dfs.name='insur_cust_ltv_sample')
[1] "/user/oracle/insur_cust_ltv_sample"
attr(),"dfs.id")
[1] TRUE
Warning message:
ORE object has no unique key - using random order
R> myDataMining <- ore.pull(myData)
Warning message:
ORE object has no unique key - using random order
R> library(rpart)
R> control <- rpart.control(maxdepth = 3)
R> dectree <- rpart(BUY_INSURANCE ~ BANK_FUNDS + CREDIT_BALANCE + MON
EY MONTLY OVERDRAWN, data=myDataMining, method = "class", control = c
ontrol)
```

The last command, `dectree <- rpart(BUY_INSURANCE ~ BANK_FUNDS + CREDIT_BALANCE + MONEY_MONTLY_OVERDRAWN, data=myDataMining, method = "class", control = control)`, is highlighted with a red rectangle.

31. The decision tree is now created, so let's review the results. Go to the terminal and type:

```
summary(dectree)
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following R session output:

```
summary(dectree)
Then press Enter

      value
prep.auto          on
impurity.metric   impurity.gini
term.max.depth     7
term.minpct.node   0.05
term.minpct.split  0.1
term.minrec.node   10
term.minrec.split  20

R> hdfs.put(myData,dfs.name='insur_cust_ltv_sample')
[1] "/user/oracle/insur_cust_ltv_sample"
attr(),"dfs.id")
[1] TRUE
Warning message:
ORE object has no unique key - using random order
R> myDataMining <- ore.pull(myData)
Warning message:
ORE object has no unique key - using random order
R> library(rpart)
R> control <- rpart.control(maxdepth = 3)
R> dectree <- rpart(BUY_INSURANCE ~ BANK_FUNDS + CREDIT_BALANCE + MONEY_MONTLY_OVERDRAWN, data=myDataMining, method = "class", control = control)
R> summary(dectree)
```

In the following screenshot we've highlighted two nodes, the first of which is a leaf node (for node 6, notice that there are no splits specified). We can see the number of observations for each node, the complexity parameter, probabilities and split conditions (the latter only for node 7). We can also see the sons of each "non-leaf" node. For example, we can see that node 15 is node 7th's right son, being defined by the following conditions: CREDIT_BALANCE < 2180 and BANK_FUNDS < 9150. These are not the only conditions defining the node, as all the conditions that apply to the parent (node 7), also apply to the child. By scrolling down to see the description of node 15, we can see that the predictive confidence for this node is 80.7%, and the expected loss 19.3%. In other words, customers that fit the conditions that define node 15 are 80.7% likely to buy insurance from our company.

```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
(0 split)
    BANK_FUNDS      < 6750      to the left,  agree=0.739, adj=0.105,
(0 split)

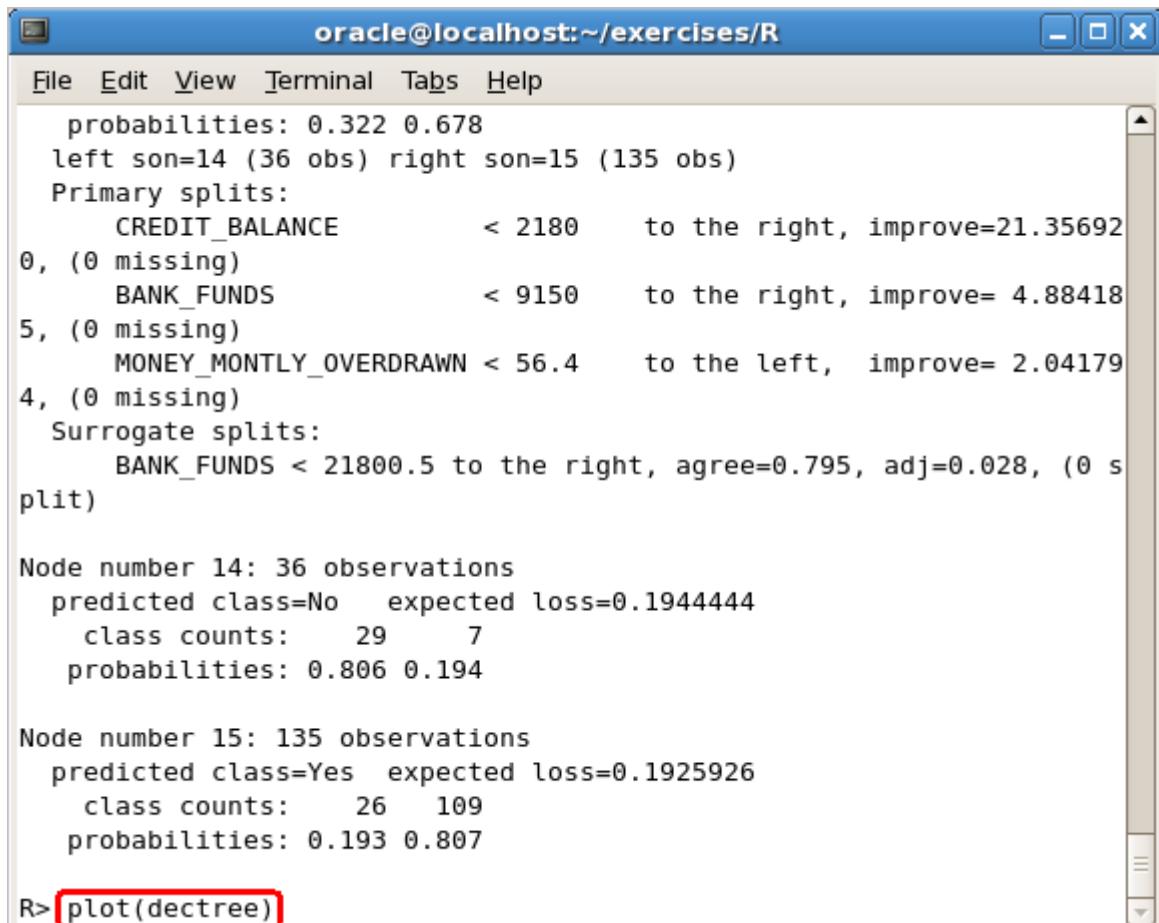
Node number 6: 415 observations
predicted class=No  expected loss=0.3614458
  class counts:  265  150
  probabilities: 0.639 0.361

Node number 7: 171 observations,    complexity param=0.08058608
predicted class=Yes  expected loss=0.3216374
  class counts:  55  116
  probabilities: 0.322 0.678
  left son=14 (36 obs) right son=15 (135 obs)
  Primary splits:
    CREDIT_BALANCE      < 2180      to the right, improve=21.35692
0, (0 missing)
    BANK_FUNDS          < 9150      to the right, improve= 4.88418
5, (0 missing)
    MONEY_MONTLY_OVERDRAWN < 56.4    to the left,  improve= 2.04179
4, (0 missing)
  Surrogate splits:
    BANK_FUNDS < 21800.5 to the right, agree=0.795, adj=0.028, (0 s
plit)
```

32. Let's also create a plot for the decision tree, so that we can provide an easy, graphical interpretation of the results. Go to the terminal and type:

```
plot(dectree)
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window displays the output of an R script. The output details the structure of a decision tree, including primary splits and surrogate splits, and provides statistics for two nodes: Node number 14 (36 observations) and Node number 15 (135 observations). The R command "plot(dectree)" is visible at the bottom of the terminal window.

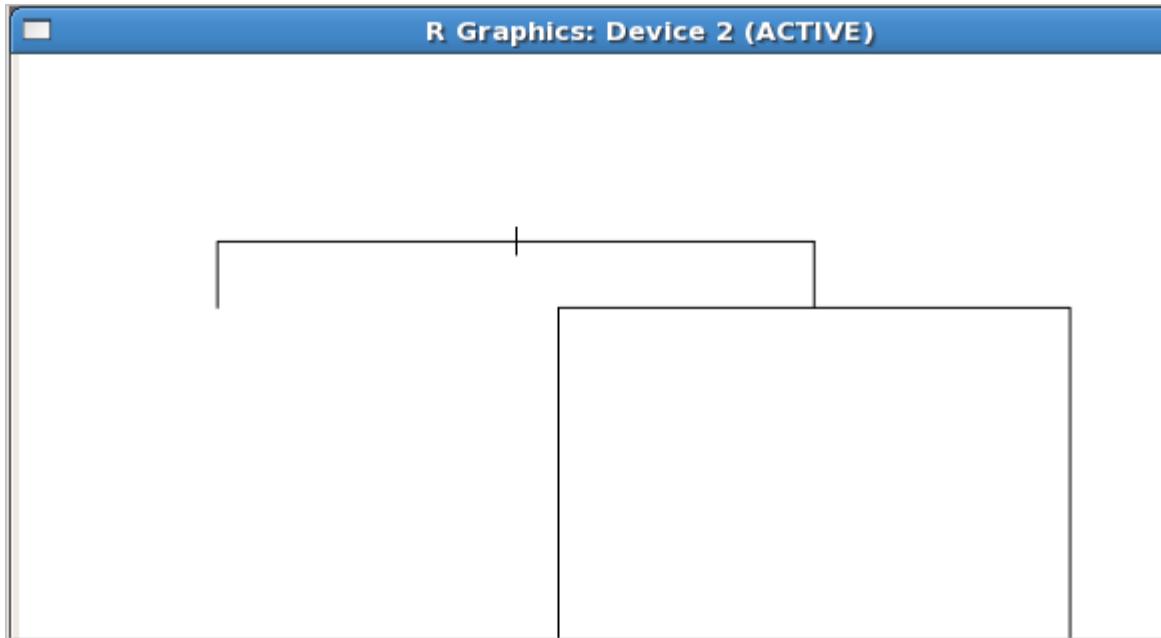
```
probabilities: 0.322 0.678
left son=14 (36 obs) right son=15 (135 obs)
Primary splits:
  CREDIT_BALANCE      < 2180      to the right, improve=21.35692
0, (0 missing)
  BANK_FUNDS          < 9150      to the right, improve= 4.88418
5, (0 missing)
  MONEY_MONTLY_OVERDRAWN < 56.4    to the left,  improve= 2.04179
4, (0 missing)
Surrogate splits:
  BANK_FUNDS < 21800.5 to the right, agree=0.795, adj=0.028, (0 split)

Node number 14: 36 observations
predicted class=No  expected loss=0.1944444
  class counts:   29    7
  probabilities: 0.806 0.194

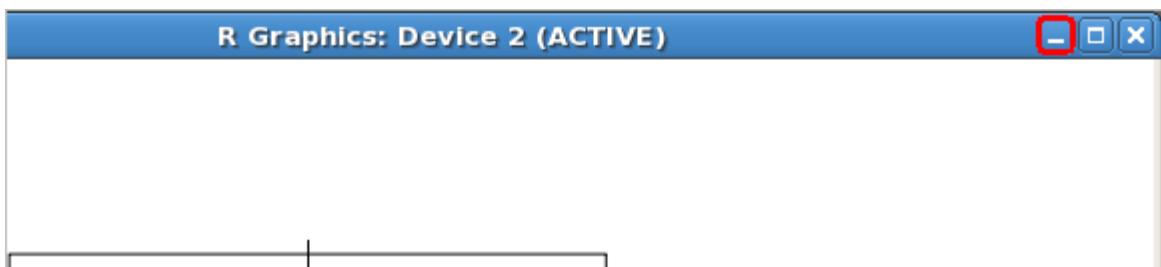
Node number 15: 135 observations
predicted class=Yes  expected loss=0.1925926
  class counts:   26   109
  probabilities: 0.193 0.807

R> plot(dectree)
```

An empty (non-labeled) plot appears on the screen.



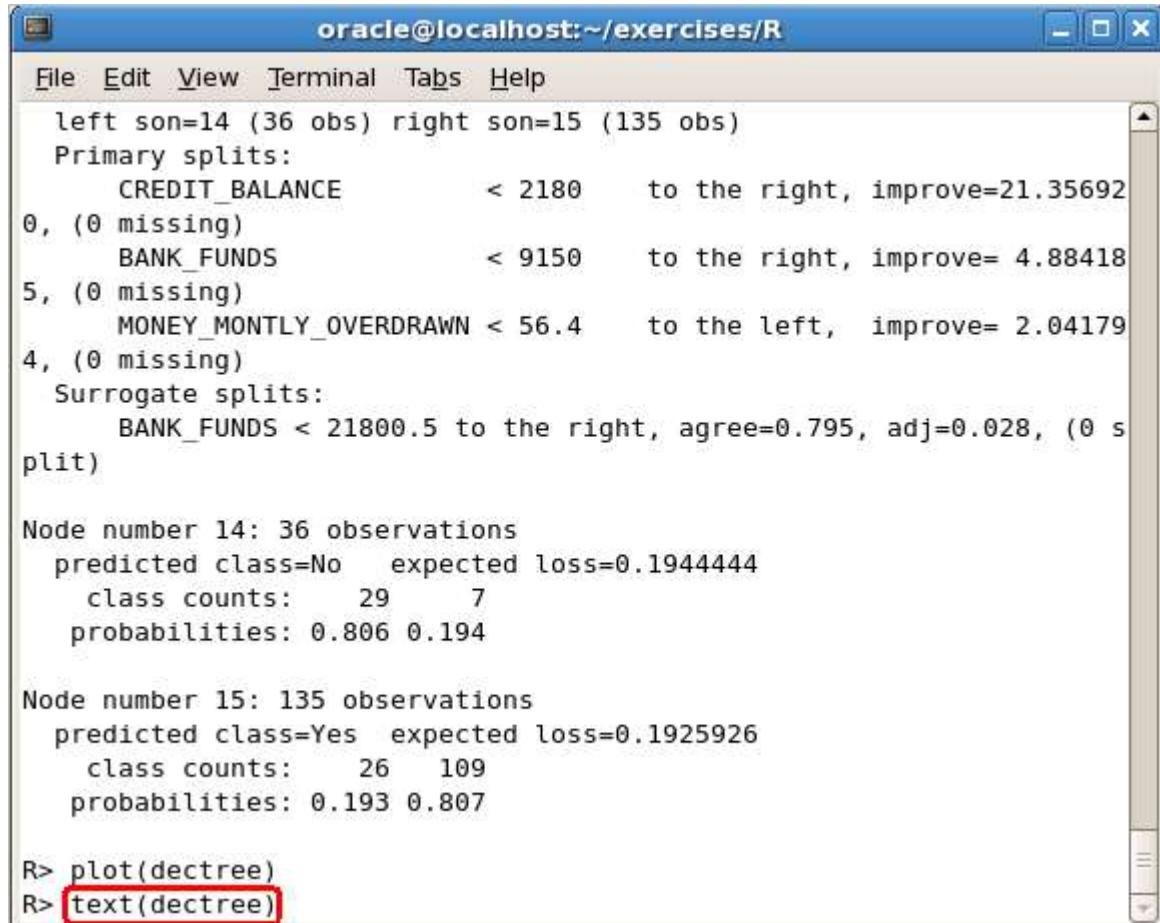
33. Let's minimize this window for now, so **click** on _ in the top right corner of the window.



34. We need to add labels to the plot. Go to the terminal and type:

```
text(dectree)
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window displays the output of an R script. The script starts by printing the structure of a decision tree object named "dectree". It shows primary splits based on "CREDIT_BALANCE" and "BANK_FUNDS", and a surrogate split for "BANK_FUNDS". It then details node 14 (36 observations) with predicted class "No", expected loss 0.1944444, class counts 29/7, and probabilities 0.806/0.194. Node 15 (135 observations) is also detailed with predicted class "Yes", expected loss 0.1925926, class counts 26/109, and probabilities 0.193/0.807. Finally, the command "text(dectree)" is shown at the bottom, with the word "text" highlighted by a red rectangle.

```
left son=14 (36 obs) right son=15 (135 obs)
Primary splits:
  CREDIT_BALANCE      < 2180      to the right, improve=21.35692
0, (0 missing)
  BANK_FUNDS          < 9150      to the right, improve= 4.88418
5, (0 missing)
    MONEY_MONTLY_OVERDRAWN < 56.4   to the left,  improve= 2.04179
4, (0 missing)
Surrogate splits:
  BANK_FUNDS < 21800.5 to the right, agree=0.795, adj=0.028, (0 s
plit)

Node number 14: 36 observations
predicted class=No  expected loss=0.1944444
  class counts: 29    7
  probabilities: 0.806 0.194

Node number 15: 135 observations
predicted class=Yes  expected loss=0.1925926
  class counts: 26   109
  probabilities: 0.193 0.807

R> plot(dectree)
R> text(dectree)
```

35. This command makes the plot easier to interpret. Go to the terminal and type:

```
post(dectree, file="")
```

Then press **Enter**

```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
Primary splits:
  CREDIT_BALANCE      < 2180      to the right, improve=21.35692
0, (0 missing)
  BANK_FUNDS          < 9150      to the right, improve= 4.88418
5, (0 missing)
  MONEY_MONTLY_OVERDRAWN < 56.4    to the left,  improve= 2.04179
4, (0 missing)
Surrogate splits:
  BANK_FUNDS < 21800.5 to the right, agree=0.795, adj=0.028, (0 splits)

Node number 14: 36 observations
predicted class=No  expected loss=0.1944444
  class counts:   29     7
  probabilities: 0.806 0.194

Node number 15: 135 observations
predicted class=Yes  expected loss=0.1925926
  class counts:   26    109
  probabilities: 0.193 0.807

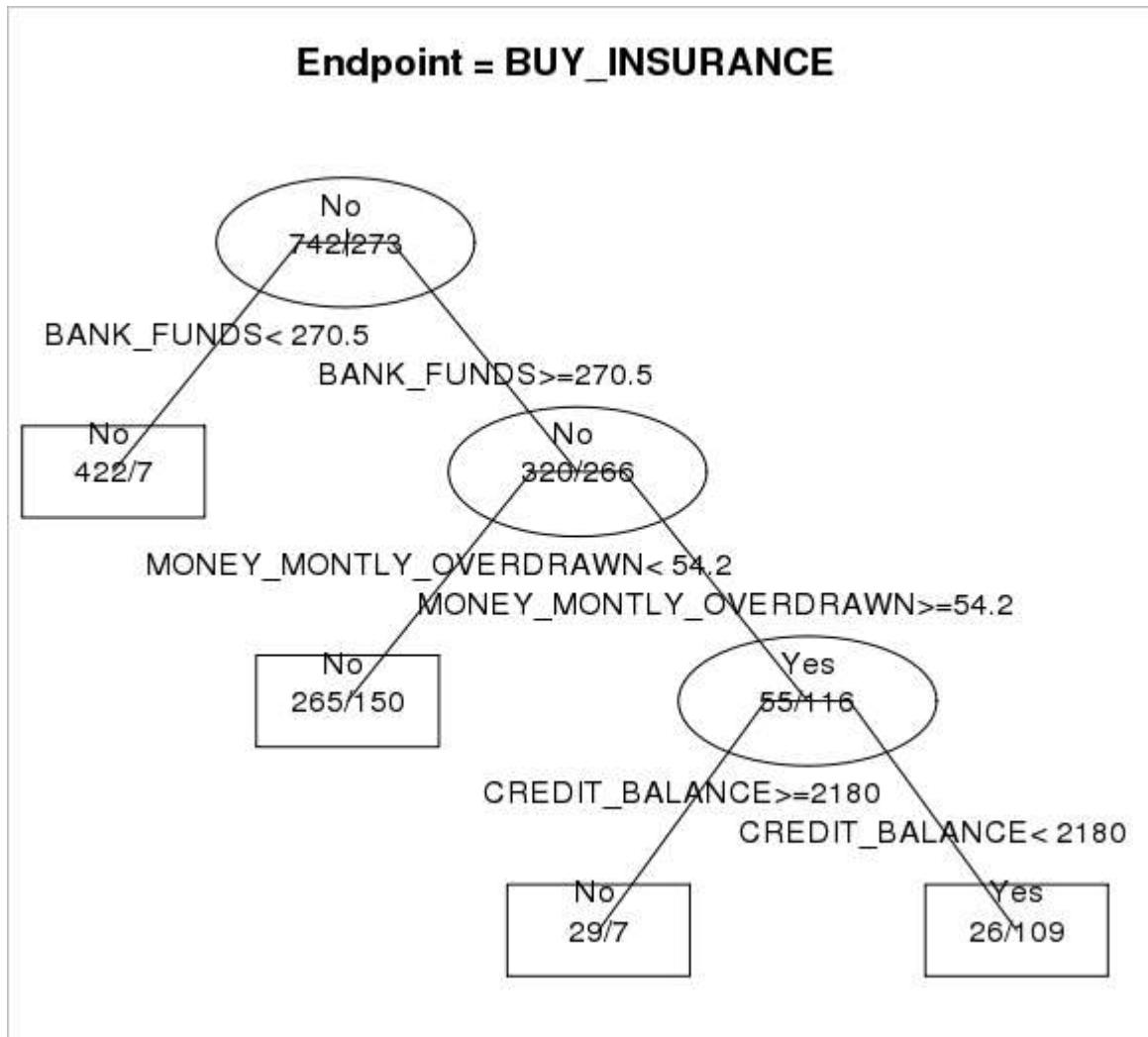
R> plot(dectree)
R> text(dectree)
R> post(dectree, file="")
```

36. Time to review the plot. Maximize the window by **clicking** on it at the bottom of the screen.

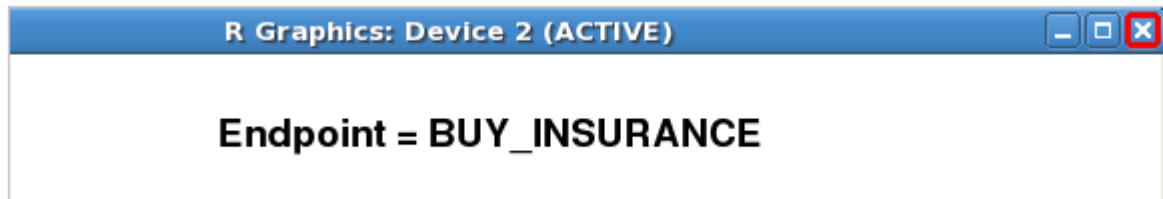


The plot shows the nodes of the decision tree, the predicted outcome for each node and the conditions that define the nodes. It also shows the number of customers who didn't buy/bought insurance for each node.

A very simple interpretation of the graph, not taking into consideration all the splits performed by the algorithm (which can be seen in the summary from above), would sound like this: let's consider the node in the bottom right corner of the graph (node 15, that was discussed a little earlier); if a customer's bank funds are higher than 270.5, the sum of monthly overdrawn money is greater than 54.2 and the credit balance is lower than 2180, then it is likely that the customer will buy insurance from our company.



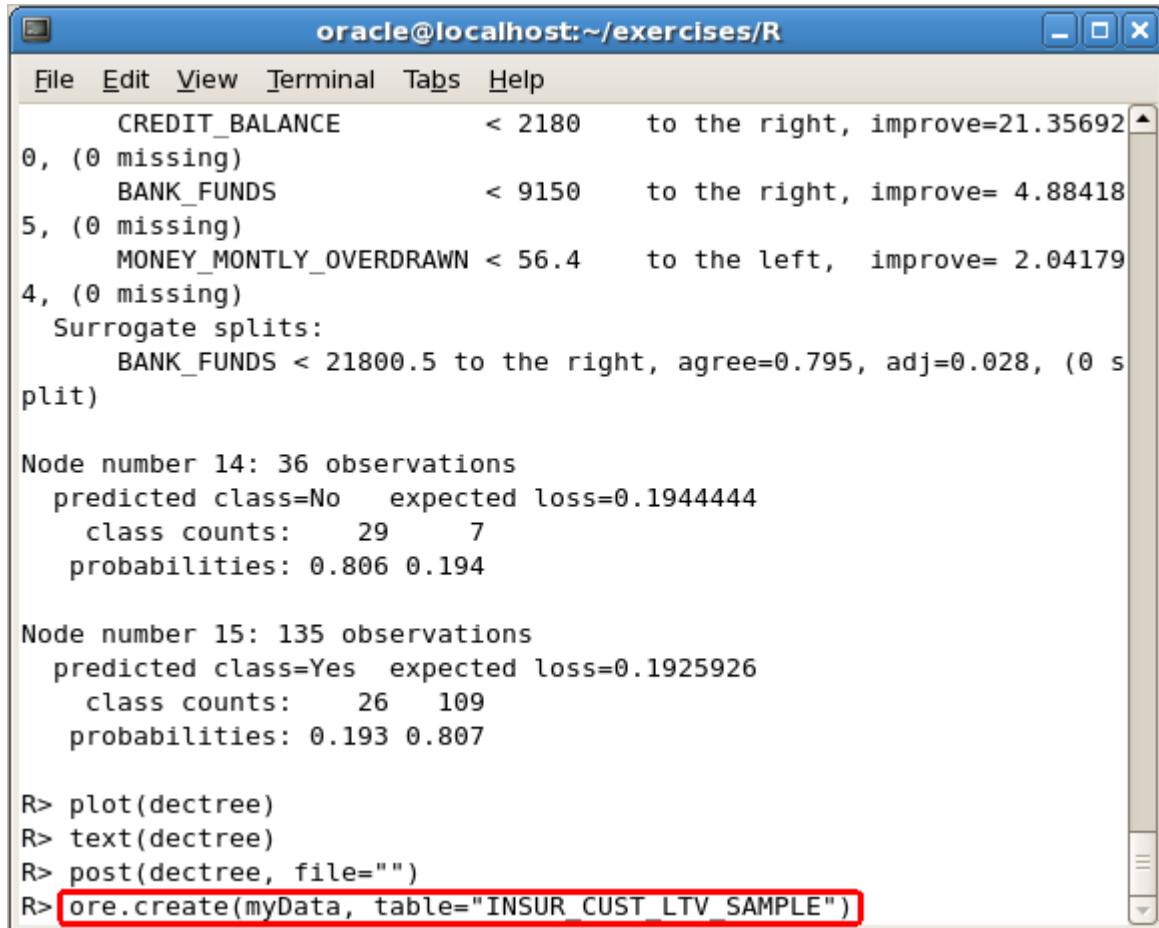
37. When you are done evaluating the graph you can click on the X in the right upper corner of the screen to close the window.



38. Before we leave R, let's save the data in the Oracle Database, in case we need to perform additional analyses at a later time. Go to the terminal and type:

```
ore.create(myData, table="INSUR_CUST_LTV_SAMPLE")
```

Then press **Enter**



```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
      CREDIT_BALANCE < 2180 to the right, improve=21.35692
0, (0 missing)
      BANK_FUNDS < 9150 to the right, improve= 4.88418
5, (0 missing)
      MONEY_MONTLY_OVERDRAWN < 56.4 to the left, improve= 2.04179
4, (0 missing)
Surrogate splits:
      BANK_FUNDS < 21800.5 to the right, agree=0.795, adj=0.028, (0 splits)

Node number 14: 36 observations
predicted class=No expected loss=0.1944444
  class counts: 29 7
  probabilities: 0.806 0.194

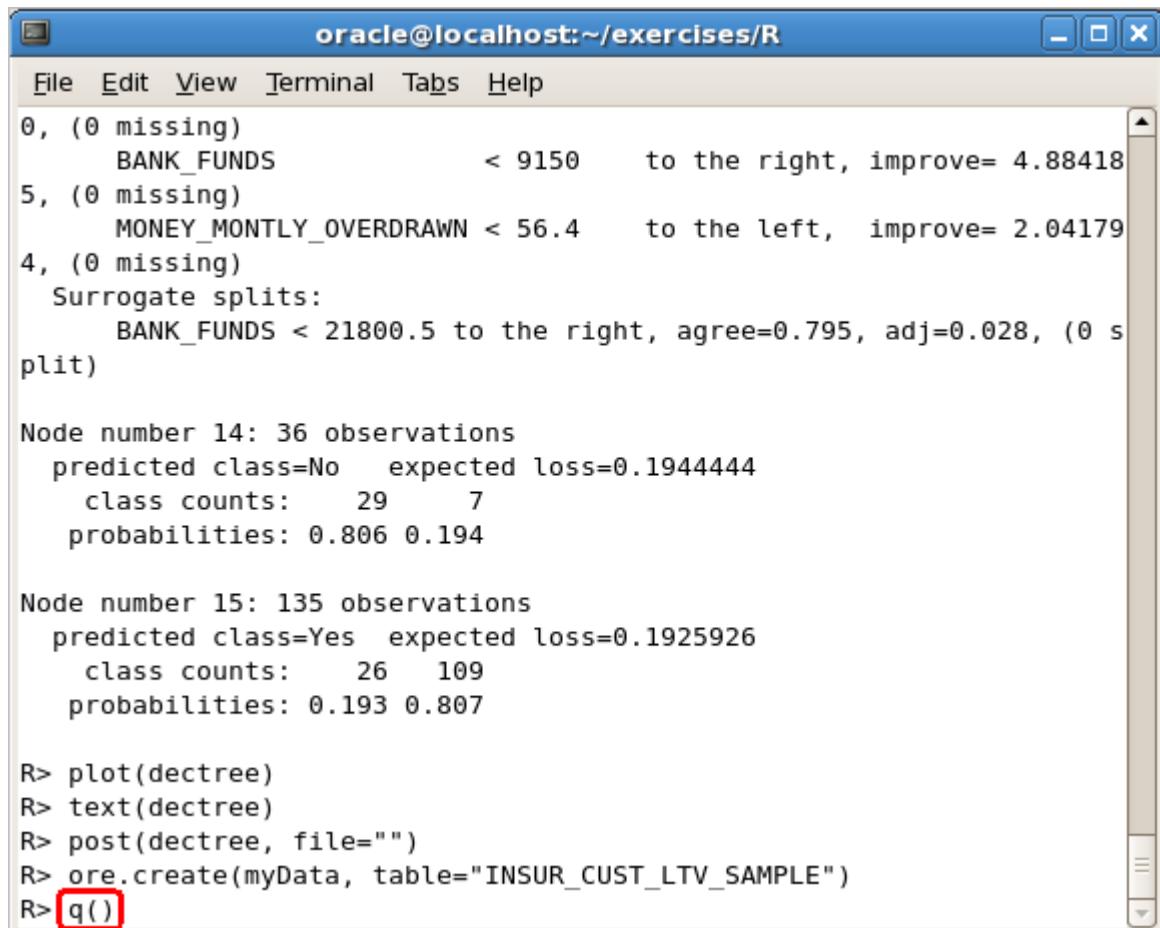
Node number 15: 135 observations
predicted class=Yes expected loss=0.1925926
  class counts: 26 109
  probabilities: 0.193 0.807

R> plot(dectree)
R> text(dectree)
R> post(dectree, file="")
R> ore.create(myData, table="INSUR_CUST_LTV_SAMPLE")
```

39. We are now ready to leave the R console. Go to the terminal and type:

```
q()
```

Then press **Enter**



```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
0, (0 missing)
    BANK_FUNDS < 9150      to the right, improve= 4.88418
5, (0 missing)
    MONEY_MONTLY_OVERDRAWN < 56.4      to the left,  improve= 2.04179
4, (0 missing)
    Surrogate splits:
        BANK_FUNDS < 21800.5 to the right, agree=0.795, adj=0.028, (0 splits)

Node number 14: 36 observations
predicted class=No  expected loss=0.1944444
  class counts:   29     7
  probabilities: 0.806 0.194

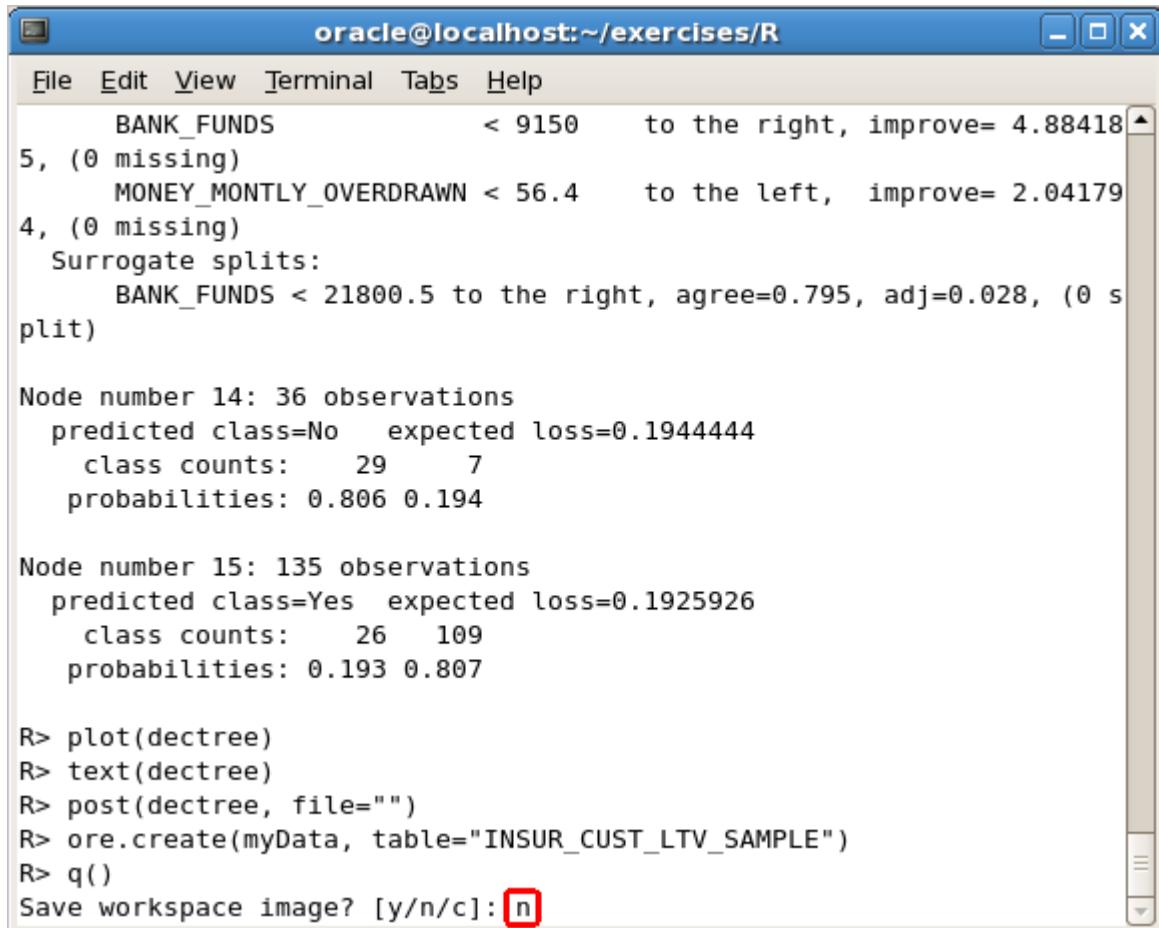
Node number 15: 135 observations
predicted class=Yes  expected loss=0.1925926
  class counts:   26   109
  probabilities: 0.193 0.807

R> plot(dectree)
R> text(dectree)
R> post(dectree, file="")
R> ore.create(myData, table="INSUR_CUST_LTV_SAMPLE")
R> q()
```

40. When asked if you want to save the workspace image, go to the terminal and type:

n

Then press **Enter**



The screenshot shows an R terminal window titled "oracle@localhost:~/exercises/R". The window displays the output of an R script that plots a decision tree and prints node statistics. At the bottom, it asks "Save workspace image? [y/n/c]:", with the input field containing "n" and a red box highlighting it.

```
BANK_FUNDS < 9150      to the right, improve= 4.88418
5, (0 missing)
  MONEY_MONTLY_OVERDRAWN < 56.4      to the left,  improve= 2.04179
4, (0 missing)
  Surrogate splits:
    BANK_FUNDS < 21800.5 to the right, agree=0.795, adj=0.028, (0 splits)

Node number 14: 36 observations
  predicted class=No  expected loss=0.1944444
    class counts:   29     7
    probabilities: 0.806 0.194

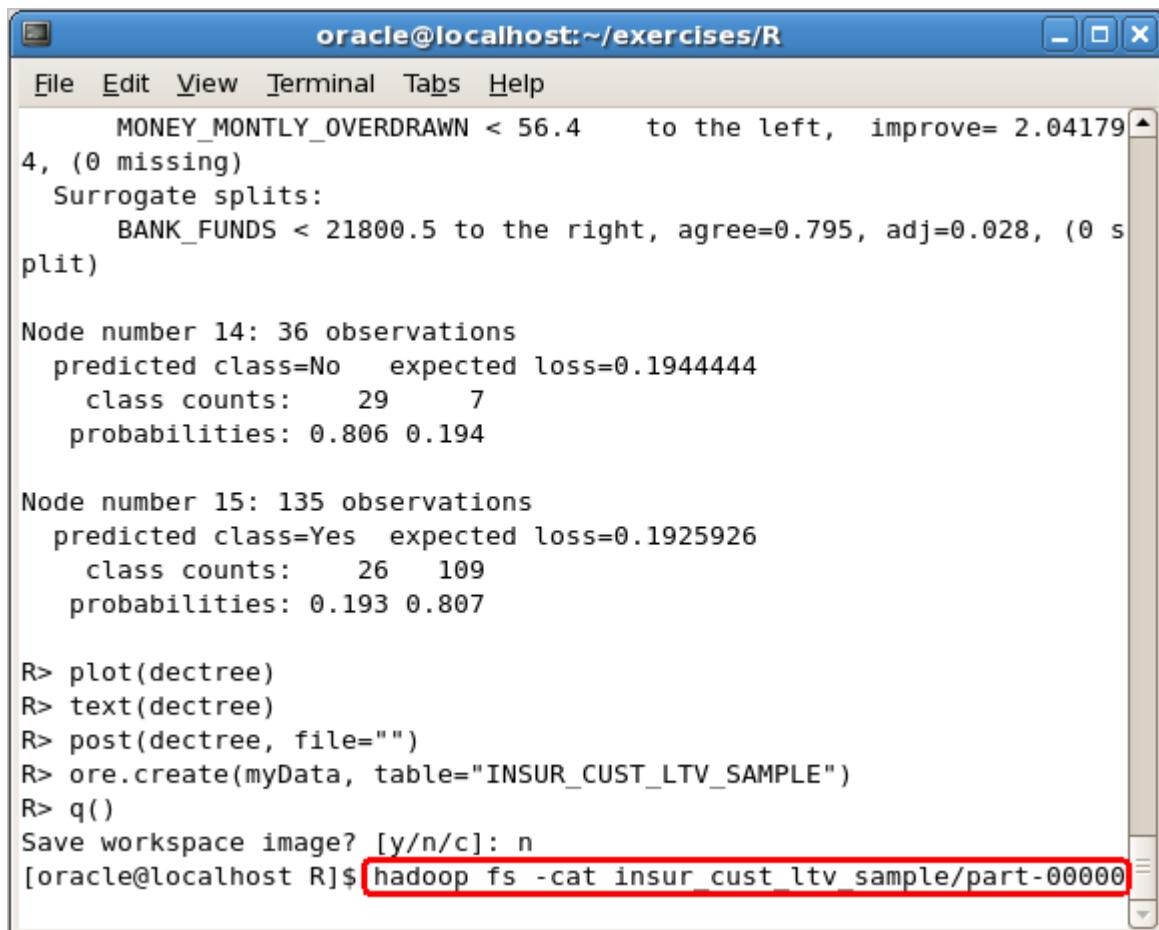
Node number 15: 135 observations
  predicted class=Yes  expected loss=0.1925926
    class counts:   26    109
    probabilities: 0.193 0.807

R> plot(dectree)
R> text(dectree)
R> post(dectree, file="")
R> ore.create(myData, table="INSUR_CUST_LTV_SAMPLE")
R> q()
Save workspace image? [y/n/c]: n
```

41. Let's review the contents of the HDFS file in which we saved the merged R dataset using ORCH.
Go to the terminal and type:

```
hadoop fs -cat insur_cust_ltv_sample/part-00000
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following text:

```
File Edit View Terminal Tabs Help
MONEY_MONTLY_OVERDRAWN < 56.4      to the left,  improve= 2.04179
4, (0 missing)
Surrogate splits:
  BANK_FUNDS < 21800.5 to the right, agree=0.795, adj=0.028, (0 splits)

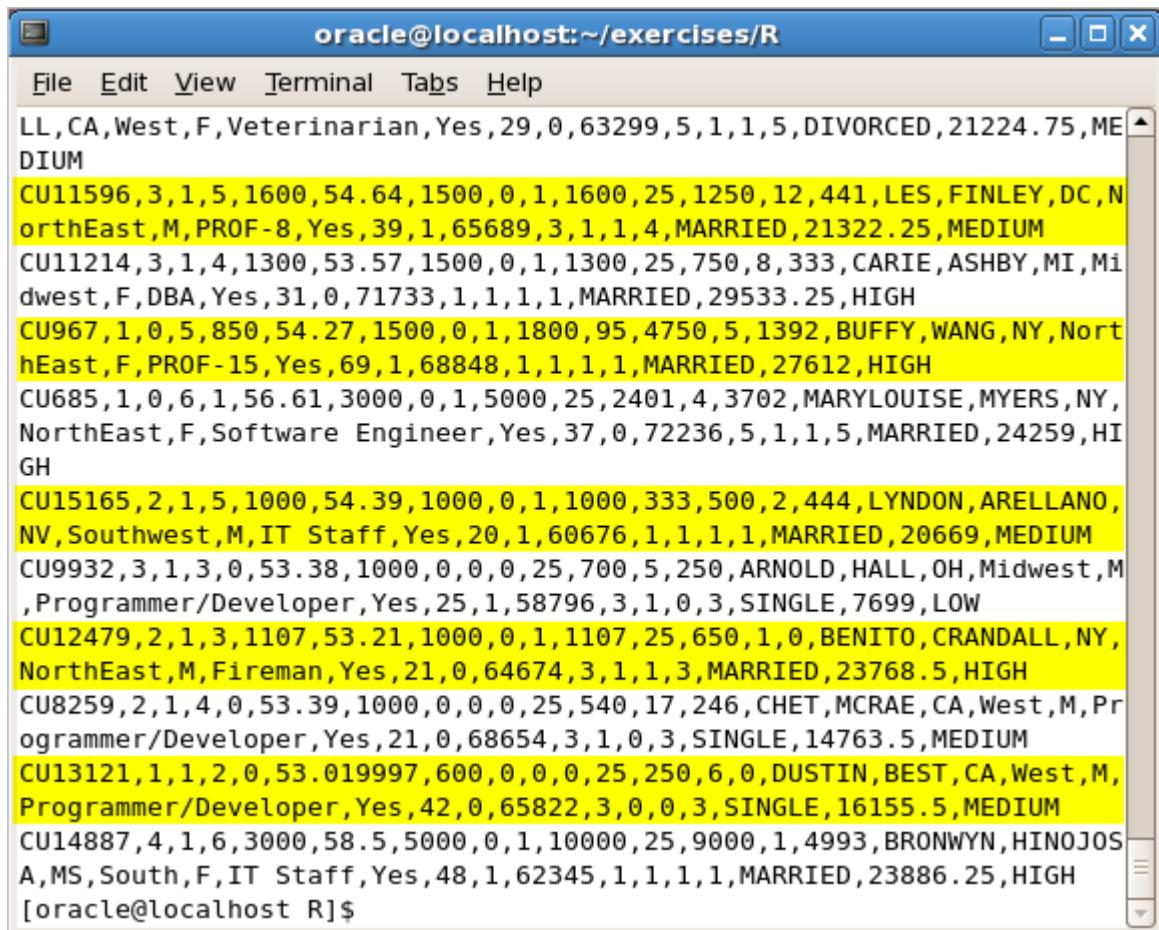
Node number 14: 36 observations
predicted class=No    expected loss=0.1944444
  class counts:   29     7
  probabilities: 0.806 0.194

Node number 15: 135 observations
predicted class=Yes   expected loss=0.1925926
  class counts:   26    109
  probabilities: 0.193 0.807

R> plot(dectree)
R> text(dectree)
R> post(dectree, file="")
R> ore.create(myData, table="INSUR_CUST_LTV_SAMPLE")
R> q()
Save workspace image? [y/n/c]: n
[oracle@localhost R]$
```

The command "hadoop fs -cat insur_cust_ltv_sample/part-00000" is highlighted with a red rectangle at the bottom of the terminal window.

The merged dataset, that represents the 360 degree view of our customers, is now available in HDFS.

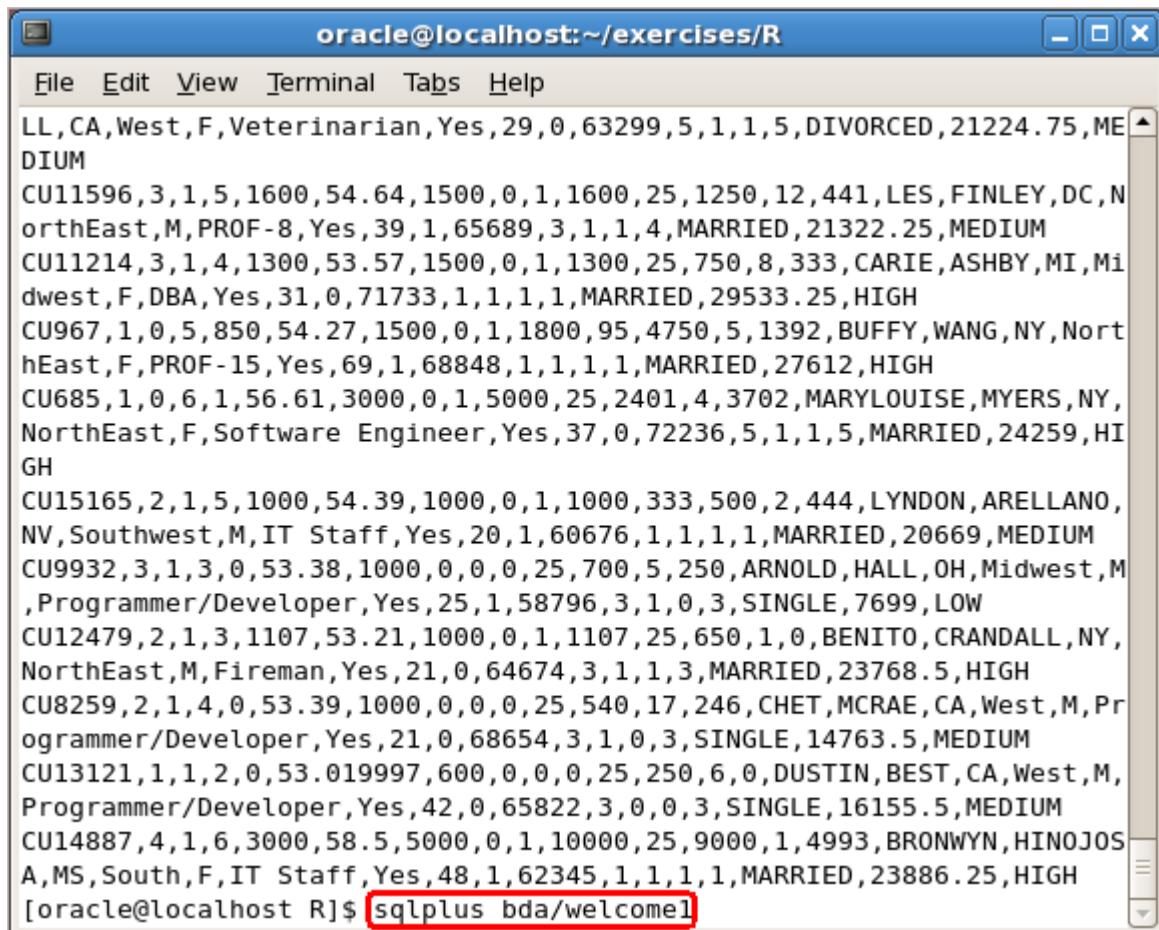


```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
LL,CA,West,F,Veterinarian,Yes,29,0,63299,5,1,1,5,DIVORCED,21224.75,ME
DIUM
CU11596,3,1,5,1600,54.64,1500,0,1,1600,25,1250,12,441,LES,FINLEY,DC,N
orthEast,M,PROF-8,Yes,39,1,65689,3,1,1,4,MARRIED,21322.25,MEDIUM
CU11214,3,1,4,1300,53.57,1500,0,1,1300,25,750,8,333,CARIE,ASHBY,MI,Mi
dwest,F,DBA,Yes,31,0,71733,1,1,1,1,MARRIED,29533.25,HIGH
CU967,1,0,5,850,54.27,1500,0,1,1800,95,4750,5,1392,BUFFY,WANG,NY,Nort
hEast,F,PROF-15,Yes,69,1,68848,1,1,1,1,MARRIED,27612,HIGH
CU685,1,0,6,1,56.61,3000,0,1,5000,25,2401,4,3702,MARYLOUISE,MYERS,NY,
NorthEast,F,Software Engineer,Yes,37,0,72236,5,1,1,5,MARRIED,24259,HI
GH
CU15165,2,1,5,1000,54.39,1000,0,1,1000,333,500,2,444,LYNDON,ARELLANO,
NV,Southwest,M,IT Staff,Yes,20,1,60676,1,1,1,1,MARRIED,20669,MEDIUM
CU9932,3,1,3,0,53.38,1000,0,0,0,25,700,5,250,ARNOLD,HALL,OH,Midwest,M
,Programmer/Developer,Yes,25,1,58796,3,1,0,3,SINGLE,7699,LOW
CU12479,2,1,3,1107,53.21,1000,0,1,1107,25,650,1,0,BENITO,CRANDALL,NY,
NorthEast,M,Fireman,Yes,21,0,64674,3,1,1,3,MARRIED,23768.5,HIGH
CU8259,2,1,4,0,53.39,1000,0,0,0,25,540,17,246,CHET,MCRAE,CA,West,M,Pr
ogrammer/Developer,Yes,21,0,68654,3,1,0,3,SINGLE,14763.5,MEDIUM
CU13121,1,1,2,0,53.019997,600,0,0,0,25,250,6,0,DUSTIN,BEST,CA,West,M,
Programmer/Developer,Yes,42,0,65822,3,0,0,3,SINGLE,16155.5,MEDIUM
CU14887,4,1,6,3000,58.5,5000,0,1,10000,25,9000,1,4993,BRONWYN,HINOJOS
A,MS,South,F,IT Staff,Yes,48,1,62345,1,1,1,1,MARRIED,23886.25,HIGH
[oracle@localhost R]$
```

42. Let's also query the Oracle Database table that we've created from the dataset that we've merged within R. First, we need to connect to SQL*Plus. Go to the terminal and type:

```
sqlplus bda/welcome1
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following text:

```
File Edit View Terminal Tabs Help
LL,CA,West,F,Veterinarian,Yes,29,0,63299,5,1,1,5,DIVORCED,21224.75,ME
DIUM
CU11596,3,1,5,1600,54.64,1500,0,1,1600,25,1250,12,441,LES,FINLEY,DC,N
orthEast,M,PROF-8,Yes,39,1,65689,3,1,1,4,MARRIED,21322.25,MEDIUM
CU11214,3,1,4,1300,53.57,1500,0,1,1300,25,750,8,333,CARIE,ASHBY,MI,Mi
dwest,F,DBA,Yes,31,0,71733,1,1,1,1,MARRIED,29533.25,HIGH
CU967,1,0,5,850,54.27,1500,0,1,1800,95,4750,5,1392,BUFFY,WANG,NY,Nort
hEast,F,PROF-15,Yes,69,1,68848,1,1,1,1,MARRIED,27612,HIGH
CU685,1,0,6,1,56.61,3000,0,1,5000,25,2401,4,3702,MARYLOUISE,MYERS,NY,
NorthEast,F,Software Engineer,Yes,37,0,72236,5,1,1,5,MARRIED,24259,HI
GH
CU15165,2,1,5,1000,54.39,1000,0,1,1000,333,500,2,444,LYNDON,ARELLANO,
NV,Southwest,M,IT Staff,Yes,20,1,60676,1,1,1,1,MARRIED,20669,MEDIUM
CU9932,3,1,3,0,53.38,1000,0,0,0,25,700,5,250,ARNOLD,HALL,OH,Midwest,M
,Programmer/Developer,Yes,25,1,58796,3,1,0,3,SINGLE,7699,LOW
CU12479,2,1,3,1107,53.21,1000,0,1,1107,25,650,1,0,BENITO,CRANDALL,NY,
NorthEast,M,Fireman,Yes,21,0,64674,3,1,1,3,MARRIED,23768.5,HIGH
CU8259,2,1,4,0,53.39,1000,0,0,0,25,540,17,246,CHET,MCRAE,CA,West,M,Pr
ogrammer/Developer,Yes,21,0,68654,3,1,0,3,SINGLE,14763.5,MEDIUM
CU13121,1,1,2,0,53.019997,600,0,0,0,25,250,6,0,DUSTIN,BEST,CA,West,M,
Programmer/Developer,Yes,42,0,65822,3,0,0,3,SINGLE,16155.5,MEDIUM
CU14887,4,1,6,3000,58.5,5000,0,1,10000,25,9000,1,4993,BRONWYN,HI
NOJOS,A,MS,South,F,IT Staff,Yes,48,1,62345,1,1,1,1,MARRIED,23886.25,HIGH
[oracle@localhost R]$ sqlplus bda/welcome1
```

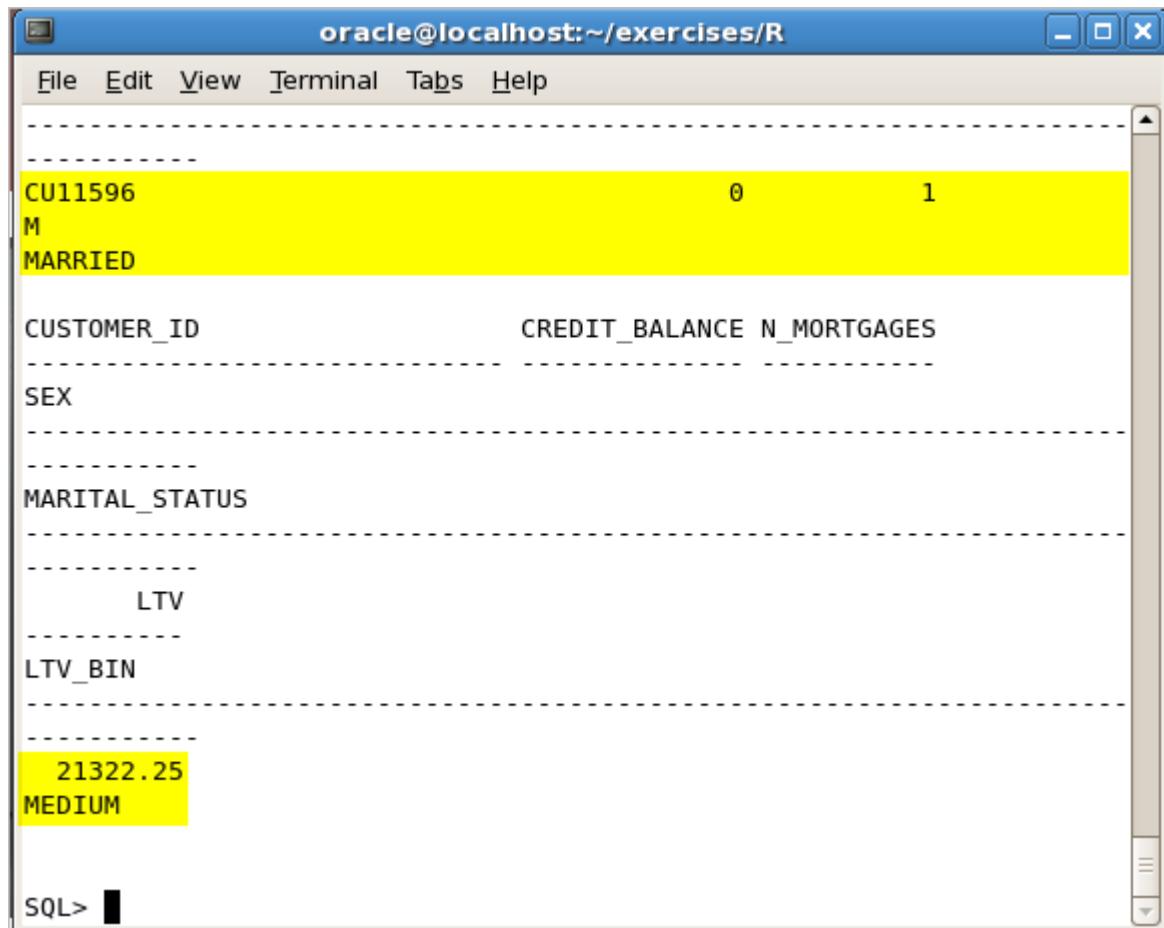
43. Now let's query some information for customer with the id "CU11596". Go to the terminal and type:

```
select CUSTOMER_ID, CREDIT_BALANCE, N_MORTGAGES, SEX, MARITAL_STATUS,  
LTV, LTV_BIN from insur_cust_ltv_sample where customer_id = 'CU11596';  
Then press Enter
```

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following text:

```
File Edit View Terminal Tabs Help  
NorthEast,M,Fireman,Yes,21,0,64674,3,1,1,3,MARRIED,23768.5,HIGH  
CU8259,2,1,4,0,53.39,1000,0,0,0,25,540,17,246,CHET,MCRAE,CA,West,M,Pr  
ogrammer/Developer,Yes,21,0,68654,3,1,0,3,SINGLE,14763.5,MEDIUM  
CU13121,1,1,2,0,53.019997,600,0,0,0,25,250,6,0,DUSTIN,BEST,CA,West,M,  
Programmer/Developer,Yes,42,0,65822,3,0,0,3,SINGLE,16155.5,MEDIUM  
CU14887,4,1,6,3000,58.5,5000,0,1,10000,25,9000,1,4993,BRONWYN,HINOJOS  
A,MS,South,F,IT Staff,Yes,48,1,62345,1,1,1,1,MARRIED,23886.25,HIGH  
[oracle@localhost R]$ sqlplus bda/welcome1  
  
SQL*Plus: Release 12.1.0.1.0 Production on Mon Sep 2 05:57:13 2013  
  
Copyright (c) 1982, 2013, Oracle. All rights reserved.  
  
Last Successful login time: Mon Sep 02 2013 05:53:14 -04:00  
  
Connected to:  
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Pro  
duction  
With the Partitioning, OLAP, Advanced Analytics and Real Application  
Testing options  
  
SQL> select CUSTOMER_ID, CREDIT_BALANCE, N_MORTGAGES, SEX, MARITAL_ST  
ATUS, LTV, LTV_BIN from insur_cust_ltv_sample where customer_id = 'CU  
11596';
```

The data associated with the specified customer is displayed in the terminal.



A screenshot of an Oracle terminal window titled "oracle@localhost:~/exercises/R". The window contains the following output:

```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
-----
CU11596          0          1
M
MARRIED

CUSTOMER_ID      CREDIT_BALANCE N_MORTGAGES
-----
SEX

-----
MARITAL_STATUS

-----
LTV
-----
LTV_BIN
-----
21322.25
MEDIUM

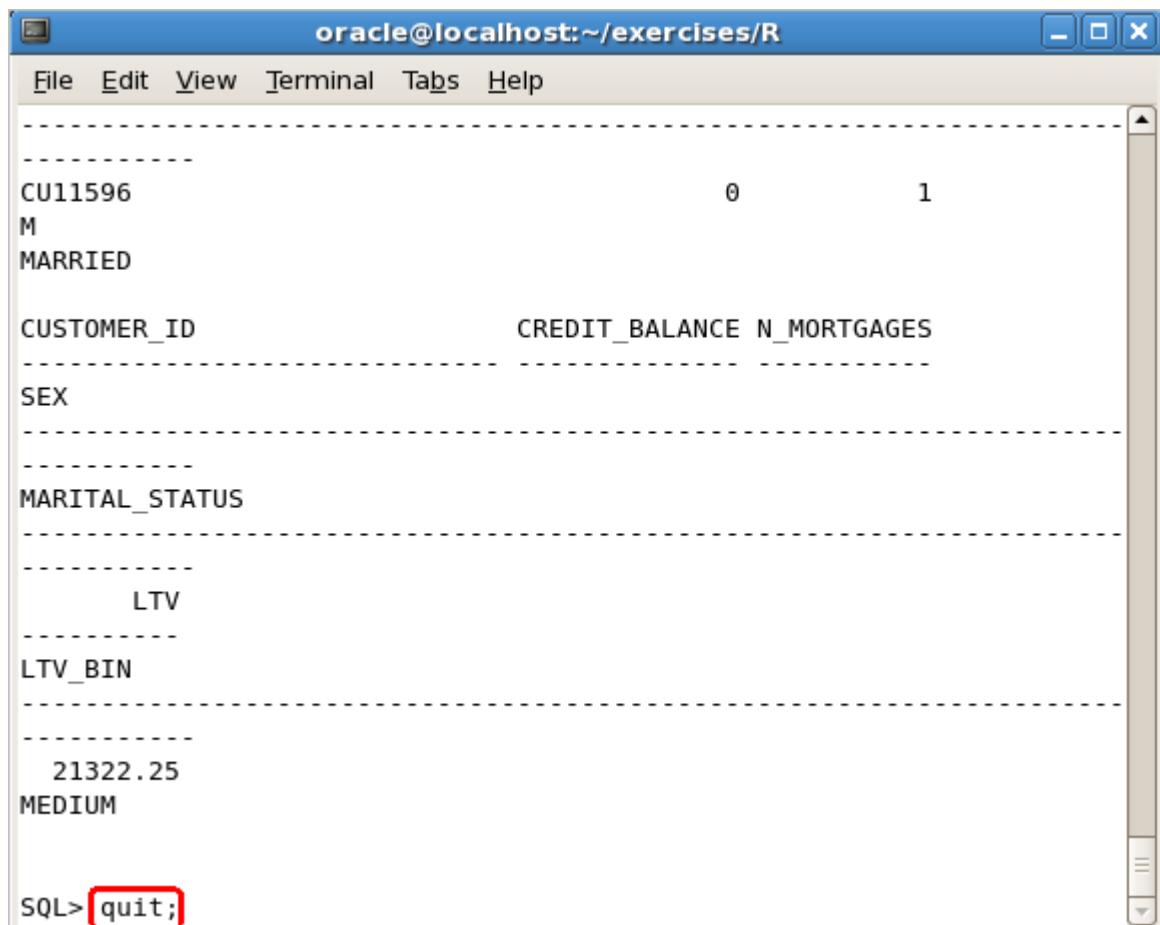
SQL>
```

The data is highlighted in yellow boxes. The first row shows CUSTOMER_ID CU11596, CREDIT_BALANCE 0, and N_MORTGAGES 1. The second row shows SEX M. The third row shows MARITAL_STATUS MARRIED. The fourth row shows LTV. The fifth row shows LTV_BIN. The sixth row shows CREDIT_BALANCE 21322.25 and N_MORTGAGES MEDIUM.

44. This exercise is now finished. First we have to exit SQL*Plus, so go to the terminal and type:

```
quit;
```

Then press **Enter**



```
oracle@localhost:~/exercises/R
File Edit View Terminal Tabs Help
-----
CU11596          0          1
M
MARRIED

CUSTOMER_ID      CREDIT_BALANCE N_MORTGAGES
-----
SEX

-----
MARITAL_STATUS

-----
LTV

-----
LTV_BIN

-----
21322.25
MEDIUM

SQL> quit;
```

45. We can now close the terminal window. Go to the terminal and type:

```
exit
```

Then press **Enter**

The screenshot shows a terminal window titled "oracle@localhost:~/exercises/R". The window contains the following text:

```
CUSTOMER_ID          CREDIT_BALANCE N_MORTGAGES
-----
SEX
-----
MARITAL_STATUS
-----
LTV
-----
LTV_BIN
-----
21322.25
MEDIUM

SQL> quit;
Disconnected from Oracle Database 12c Enterprise Edition Release 12.1
.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost R]$ exit
```

The word "exit" at the bottom of the terminal window is highlighted with a red rectangle.

12.4 Summary

In this exercise you were introduced to the R programming language and how to perform a decision tree algorithm using the programming language. You also saw one of the advantages of Oracle R Enterprise where you can save your results into the Oracle database as well as extract data from the database for further calculations. Oracle R Enterprise also has a small set of functions which can be run on data in the database directly in the database. This enables the user to use very large data sets which would not fit into the normal memory of R.

Oracle R Enterprise provides the following packages:

- DBI - R Database Interface
- ORE - Oracle R Enterprise
- OREbase - Oracle R Enterprise - base
- OREdm - Oracle R Enterprise - dm
- OREeda - Oracle R Enterprise - exploratory data analysis
- OREgraphics - Oracle R Enterprise - graphics
- OREPredict - Oracle R Enterprise - model predictions
- OREstats - Oracle R Enterprise - stats
- ORExml - Oracle R Enterprise - R objects to XML
- ROOracle - OCI based Oracle database interface for R
- XML - Tools for parsing and generating XML within R and S-Plus.
- bitops - Functions for Bitwise operations
- png - Read and write PNG images

Oracle R Connector for Hadoop provides access from a local R client to Apache Hadoop using functions with these prefixes:

- hadoop: Identifies functions that provide an interface to Hadoop Map/Reduce
- hdfs: Identifies functions that provide an interface to HDFS
- orch: Identifies a variety of functions; orch is a general prefix for ORCH functions
- ore: Identifies functions that provide an interface to a Hive data store

Oracle R Connector for Hadoop uses data frames as the primary object type, but it can also operate on vectors and matrices to exchange data with HDFS. The APIs support the numeric, integer, and character data types in R.

13. WORKING WITH CLOUDERA SEARCH (SOLR)

13.1 Introducing Cloudera Search

Cloudera Search is one of Cloudera's near-real-time access products. Cloudera Search enables non-technical users to search and explore data stored in or ingested into Hadoop and HBase. Users do not need SQL or programming skills to use Cloudera Search because it provides a simple, full-text interface for searching.

Another benefit of Cloudera Search, compared to stand-alone search solutions, is the fully integrated data processing platform. Search uses the flexible, scalable, and robust storage system included with CDH. This eliminates the need to move larger data sets across infrastructures to address business tasks.

Cloudera Search incorporates Apache Solr, which includes Apache Lucene, SolrCloud, Apache Tika, and Solr Cell. Cloudera Search 1.x is tightly integrated with Cloudera's Distribution, including Apache Hadoop (CDH) and is included with CDH 5. Cloudera Search provides these key capabilities:

- Near-real-time indexing
- Batch indexing
- Simple, full-text data exploration and navigated drill down

Using Search with the CDH infrastructure provides:

- Simplified infrastructure
- Better production visibility
- Quicker insights across various data types
- Quicker problem resolution
- Simplified interaction with the ability to open the platform to more users and use cases
- Scalability, flexibility, and reliability of search services on the same platform as where you can execute other types of workloads on the same data

How Cloudera Search Works

In a near-real-time indexing use case, Cloudera Search indexes events that are streamed through Apache Flume on their way into storage in CDH. Fields and events are mapped to standard Solr indexable schemas. Lucene indexes events, and the integration through Cloudera Search allows the index to be directly written and stored in standard Lucene index files in HDFS. Flume's capabilities to route events and have data stored in partitions in HDFS can also be applied. Events can be routed and streamed through multiple Flume agents and written to separate Lucene indexers that can write into separate index shards, for better scale when indexing and quicker responses when searching. The indexes are loaded from HDFS to Solr cores, exactly like Solr would have read from local disk. The difference in the design of Cloudera Search is the robust, distributed, and scalable storage layer of HDFS, which helps eliminate costly downtime and allows for flexibility across workloads without having to move data. Search queries can then be submitted to Solr through either the standard Solr API, or through a simple search GUI application, included in Cloudera Search, which can easily be deployed in Hue.

Cloudera Search batch-oriented indexing capabilities can address needs for searching across batch uploaded files or large data sets that are less frequently updated and less in need of near-real-time indexing. For such cases, Cloudera Search includes a highly scalable indexing workflow based on MapReduce. A MapReduce workflow is launched onto specified files or folders in HDFS, and the field extraction and Solr schema mapping is executed during the mapping phase. Reducers use Solr to write the

data as a single index or as index shards, depending on your configuration and preferences. Once the indexes are stored in HDFS, they can be queried using standard Solr mechanisms, as previously described above for the near-real-time indexing use case.

The Lily HBase Indexer Service is a flexible, scalable, fault tolerant, transactional, Near Real Time (NRT) oriented system for processing a continuous stream of HBase cell updates into live search indexes. Typically the time between data ingestion using the Flume sink to that content potentially appearing in search results is on the order of seconds, though this duration is tunable. The Lily HBase Indexer uses Solr to index data stored in HBase. As HBase applies inserts, updates, and deletes to HBase table cells, the indexer keeps Solr consistent with the HBase table contents, using standard HBase replication features. The indexer supports flexible custom application-specific rules to extract, transform, and load HBase data into Solr. Solr search results can contain columnFamily:qualifier links back to the data stored in HBase. This way, applications can use the Search result set to directly access matching raw HBase cells. Indexing and searching do not affect operational stability or write throughput of HBase because the indexing and searching processes are separate and asynchronous to HBase.

13.2 Cloudera Search Features

Unified Management and Monitoring with Cloudera Manager

Cloudera Manager provides a unified and centralized management and monitoring experience for both CDH and Cloudera Search. Cloudera Manager simplifies deployment, configuration, and monitoring of your search services. This differs from many existing search solutions that lack management and monitoring capabilities and that fail to provide deep insight into utilization, system health, trending, and various other supportability aspects.

Index Storage in HDFS

Cloudera Search is integrated with HDFS for index storage. Indexes created by Solr/Lucene can be directly written in HDFS with the data, instead of to local disk, thereby providing fault tolerance and redundancy.

Cloudera has optimized Cloudera Search for fast read and write of indexes in HDFS while indexes are served and queried through standard Solr mechanisms. Also, because data and indexes are co-located, once data is found, processing does not require transport or separately managed storage.

Batch Index Creation through MapReduce

To facilitate index creation for large sets of data, Cloudera Search has built-in MapReduce jobs for indexing data stored in HDFS. As a result, the linear scalability of MapReduce is applied to the indexing pipeline.

Real-time and Scalable Indexing at Data Ingest

Cloudera Search provides integration with Flume to support near-real-time indexing. As new events pass through a Flume hierarchy and are written to HDFS, those same events can be written directly to Cloudera Search indexers.

In addition, Flume supports routing events, filtering, and adding annotations on data on its passage to CDH. These features work with Cloudera Search for improved index sharding, index separation, and document-level access control.

Easy Interaction and Data Exploration through Hue

A Cloudera Search GUI is provided as Hue plug-in, enabling users to interactively query data, view result files, and do faceted exploration. Hue can also schedule standing queries and explore index files. This GUI uses the Cloudera Search API, which is based on the standard Solr API.

Simplified Data Processing for Search Workloads

Cloudera Search relies on Apache Tika for parsing and preparation of many of the standard file formats for indexing. Additionally, Cloudera Search supports Avro, Hadoop Sequence, and Snappy file format mappings, as well as support for Log file formats, JSON, XML, and HTML. Cloudera Search also provides

data preprocessing using Morphlines. This built-in support simplifies index configuration for these formats, which you can use for other applications such as MapReduce jobs.

HBase Search

Cloudera Search integrates with HBase, enabling full-text search of data stored in HBase. This functionality, which does not affect HBase performance, is based on a listener that monitors the replication event stream. The listener captures each write or update-replicated event, enabling extraction and mapping. The event is then sent directly to Solr indexers, deployed on HDFS, and written to indexes in HDFS, using the same process as for other indexing workloads of Cloudera Search. The indexes can then immediately be served, enabling near real time search of HBase data.

13.3 Cloudera Search Tasks and Processes

For content to be searchable, it must exist in CDH and be indexed. Content can either already exist in CDH and be indexed on demand or it can be updated and indexed continuously. The first step towards making content searchable is to ensure it is ingested or stored in CDH.

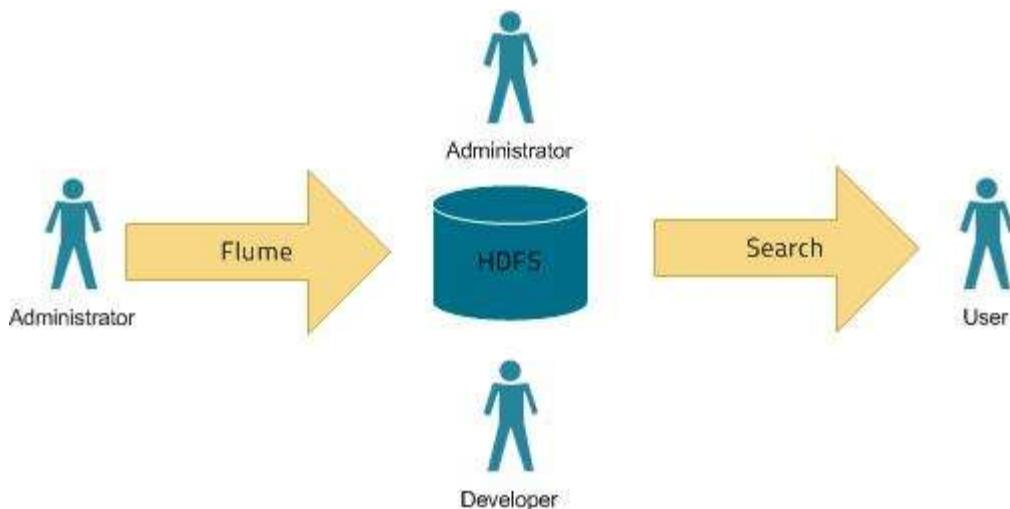
Ingestion

Content can be moved to CDH through techniques such as using:

- Flume, a flexible, agent-based data ingestion framework.
- A copy utility such as distcp for HDFS.
- Sqoop, a structured data ingestion connector.
- fuse-dfs.

In a typical environment, administrators establish systems for search. For example, HDFS is established to provide storage; Flume or distcp are established for content ingestion. Once administrators establish these services, users can use ingestion technologies such as file copy utilities or Flume sinks.

Indexing



Content must be indexed before it can be searched. Indexing is comprised of a set of steps:

ETL Steps — Extraction, Transformation, and Loading (ETL) is handled using existing engines or frameworks such as Apache Tika or Cloudera Morphlines.

- Content and metadata extraction.
- Schema mapping

Index creation — Indexes are created by Lucene.

- Index creation
- Index serialization

Indexes are typically stored on a local file system. Lucene supports additional index writers and readers. One such index interface is HDFS-based and has been implemented as part of Apache Blur. This index interface has been integrated with Cloudera Search and modified to perform well with CDH-stored indexes. All index data in Cloudera Search is stored in HDFS and served from HDFS.

There are three ways to index content:

Batch indexing using MapReduce

To use MapReduce to index documents, documents are first written to HDFS. A MapReduce job can then be run on the content in HDFS, producing a Lucene index. The Lucene index is written to HDFS, and this index is subsequently used by search services to provide query results.

Batch indexing is most often used when bootstrapping a search cluster. The Map component of the MapReduce task parses input into indexable documents and the Reduce component contains an embedded Solr server that indexes the documents produced by the Map. A MapReduce-based indexing job can also be configured to utilize all assigned resources on the cluster, utilizing multiple reducing steps for intermediate indexing and merging operations, with the last step of reduction being to write to the configured set of shard sets for the service. This makes the batch indexing process as scalable as MapReduce workloads.

Near Real Time (NRT) indexing using Flume

Flume events are typically collected and written to HDFS. While any Flume event could be written, logs are a common case.

Cloudera Search includes a Flume sink that includes the option to directly write events to the indexer. This sink provides a flexible, scalable, fault tolerant, near real time (NRT) system for processing continuous streams of records, creating live-searchable, free-text search indexes. Typically it is a matter of seconds from data ingestion using the Flume sink to that content potentially appearing in search results, though this duration is tunable.

The Flume sink has been designed to meet the needs of identified use cases that rely on NRT availability. Data can flow from multiple sources through multiple flume nodes. These nodes, which can be spread across a network route this information to one or more Flume indexing sinks. Optionally, you can split the data flow, storing the data in HDFS while also writing it to be indexed by Lucene indexers on the cluster. In that scenario data exists both as data and as indexed data in the same storage infrastructure. The indexing sink extracts relevant data, transforms the material, and loads the results to live Solr search servers. These Solr servers are then immediately ready to serve queries to end users or search applications.

This system is flexible and customizable, and provides a high level of scaling as parsing is moved from the Solr server to the multiple Flume nodes for ingesting new content.

Search includes parsers for a set of standard data formats including Avro, CSV, Text, HTML, XML, PDF, Word, and Excel. While many formats are supported, you can extend the system by adding additional custom parsers for other file or data formats in the form of Tika plug-ins. Any type of data can be indexed: a record is a byte array of any format and parsers for any data format and any custom ETL logic can be established.

In addition, Cloudera Search comes with a simplifying Extract-Transform-Load framework called Cloudera Morphlines that can help adapt and pre-process data for indexing. This eliminates the need for specific parser deployments, replacing them with simple commands.

Cloudera Search has been designed to efficiently handle a variety of use cases.

Search supports routing to multiple Solr collections as a way of making a single set of servers support multiple user groups (multi-tenancy).

Search supports routing to multiple shards to improve scalability and reliability.

Index servers can be either co-located with live Solr servers serving end user queries or they can be deployed on separate commodity hardware, for improved scalability and reliability.

Indexing load can be spread across a large number of index servers for improved scalability, and indexing load can be replicated across multiple index servers for high availability.

This is a flexible, scalable, highly available system that provides low latency data acquisition and low latency querying. Rather than replacing existing solutions, Search complements use-cases based on batch analysis of HDFS data using MapReduce. In many use cases, data flows from the producer through Flume to both Solr and HDFS. In this system, NRT ingestion, as well as batch analysis tools can be used.

NRT indexing using some other client that uses the NRT API

Documents written by a third-party directly to HDFS can trigger indexing using the Solr REST API. This API can be used to complete a number of steps:

- Extract content from the document contained in HDFS where the document is referenced by a URL.
- Map the content to fields in the search schema.
- Create or update a Lucene index.

This could be useful if you do indexing as part of a larger workflow. For example, you might choose to trigger indexing from an Oozie workflow.

Querying

Once data has been made available as an index, the query API provided by the search service allows for direct queries to be executed, or facilitated through some third party, such as a command line tool or graphical interface. Cloudera Search provides a simple UI application that can be deployed with Hue, but it is just as easy to develop a custom application, fitting your needs, based on the standard Solr API. Any application that works with Solr is compatible and runs as a search-serving application for Cloudera Search, as Solr is the core.

13.4 Exercises

In order to start using Solr for indexing the data, you must configure a collection holding the index. A configuration for a collection requires a solrconfig.xml file, a schema.xml and any helper files may be referenced from the xml files. The solrconfig.xml file contains all of the Solr settings for a given collection, and the schema.xml file specifies the schema that Solr uses when indexing documents. For more details on how to configure it for your data set see <http://wiki.apache.org/solr/SchemaXml>.

This exercise will use the results data we got from the R / ORAAH exercise and index the "customer_ltv_sample" data and make it searchable. We will use Oracle OXH (XQuery for Hadoop) to generate the necessary data and load it into SOLR for indexing.

Configuration files for a collection are managed as part of the instance directory.

1. We will begin by using solrctl command line tool. Generate a skeleton of the instance directory, change directory to /home/oracle/exercises/solr and run:

```
[oracle@bigdatalite solr]$ solrctl instancedir --generate solr_configs
[oracle@bigdatalite solr]$ ls -l solr_configs/conf/
total 348
-rw-r--r-- 1 oracle oinstall 1092 Jan 14 00:17 admin-extra.html
-rw-r--r-- 1 oracle oinstall 953 Jan 14 00:17 admin-extra.menu-bottom.html
-rw-r--r-- 1 oracle oinstall 951 Jan 14 00:17 admin-extra.menu-top.html
-rw-r--r-- 1 oracle oinstall 4041 Jan 14 00:17 currency.xml
-rw-r--r-- 1 oracle oinstall 1386 Jan 14 00:17 elevate.xml
drwxr-xr-x 2 oracle oinstall 4096 Jan 14 00:17 lang
-rw-r--r-- 1 oracle oinstall 82327 Jan 14 00:17 mapping-FoldToASCII.txt
-rw-r--r-- 1 oracle oinstall 3114 Jan 14 00:17 mapping-ISOLatin1Accent.txt
-rw-r--r-- 1 oracle oinstall 894 Jan 14 00:17 protwords.txt
-rw-r--r-- 1 oracle oinstall 59635 Jan 14 00:17 schema.xml
-rw-r--r-- 1 oracle oinstall 921 Jan 14 00:17 scripts.conf
-rw-r--r-- 1 oracle oinstall 72218 Jan 14 00:17 solrconfig.xml
-rw-r--r-- 1 oracle oinstall 73601 Jan 14 00:17 solrconfig.xml.secure
-rw-r--r-- 1 oracle oinstall 16 Jan 14 00:17 spellings.txt
-rw-r--r-- 1 oracle oinstall 795 Jan 14 00:17 stopwords.txt
-rw-r--r-- 1 oracle oinstall 1148 Jan 14 00:17 synonyms.txt
-rw-r--r-- 1 oracle oinstall 1469 Jan 14 00:17 update-script.js
drwxr-xr-x 2 oracle oinstall 4096 Jan 14 00:17 velocity
drwxr-xr-x 2 oracle oinstall 4096 Jan 14 00:17 xslit
[oracle@bigdatalite solr]$
```

You can customize it by directly editing the solrconfig.xml and schema.xml files that have been created in /home/oracle/exercises/solr/solr_configs/conf.

Open solr_configs/conf/schema.xml and solr_configs/conf/solrconfig.xml for viewing.

2. Once you have viewed the configuration, you can make it available for Solr to use by uploading the content of the entire instance directory to ZooKeeper.
3. Creating Your First Solr Collection : A complete logical index in a SolrCloud cluster. It is associated with a config set (instancedir) and is made up of one or more shards. If the number of shards is more than one, it is a distributed index, but Solr lets you refer to it by the collection name.
4. For this exercise a script has already been created to generate the collection and index the HDFS data.

Open the file oxh_solr.xq : You will remember from the OXH exercise that OXH is an XQuery tool for generating map reduce. In this case we are parsing the HDFS file and writing the output to a SOLR index.

```

lite conf]$ cd
lite [1] oracle@bigdatalite:~/exercises/solr
lite File Edit View Search Terminal Help
lite
:h import module "oxh:text";
:h import module "oxh:solr";
:h
:li for $i in text:collection(
    oxh:property("oxh_tck.xq.in") || "/part*"
)
:inv
:io let $f := tokenize($i, ",")
:io let $cust_id := xs:string($f[1])
:io let $cust_n := concat(xs:string($f[15]), ' ', xs:string($f[16]))
:ed let $age := xs:integer($f[22])
:ed let $prof := xs:string($f[20])
:fi
:r return solr:put(
<doc>
    <field name="id">{ $cust_id }</field>
    <field name="name">{ $cust_n }</field>
    <field name="title">{ $prof }</field>
</doc>
:vi
)

```

20,0-1 Bot

Open the file oxh_solr.sh to view the script to generate the collection and run the OXH XQuery tool against our data from ORAAH.

```

oracle@bigdatalite:~/exercises/solr
File Edit View Search Terminal Help
#
# export ZKHOSTS=bigdatalite.localdomain:2181/solr
# export ZKQUORUM=bigdatalite.localdomain:2181
# export COLLECTION=collection1
# export HDFS_DATA=/user/oracle/insur_cust_ltv_sample
# export HDFS_TMP=/user/oracle/oxh_temp

./cleanup.sh

solrctl instancedir --generate solr_configs
solrctl instancedir --create collection1 solr_configs
solrctl collection --create collection1 -s 1

hadoop jar $OXH_HOME/lib/oxh.jar \
-Dzkhosts=$ZKHOSTS -Dzkquorum=$ZKQUORUM -Dcollection=$COLLECTION \
-Dhdfs_data=$HDFS_DATA -Dhdfs_tmp=$HDFS_TMP \
-conf ./oxh_solr.xml oxh_solr.xq -clean -output $HDFS_TMP/oxh_solr

"oxh_solr.sh" 24L, 685C

```

24,0-1 Bot

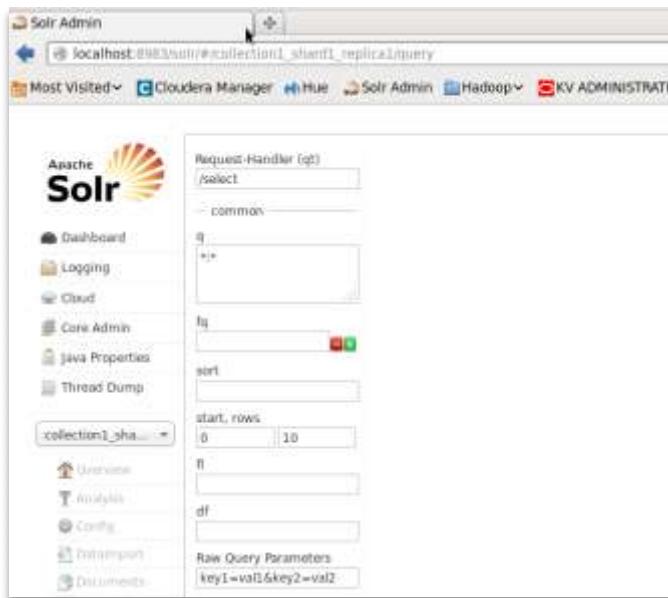
5. Run the script `./oxh_solr.sh` to index the HDFS file for SOLR.
6. We can open the SOLR webpage at `http://bigdatalite.localdomain:8983/solr` and browse the configs and collection we have created,

Running Queries

We have now indexed the HDFS file and can run a few queries.

To run a query:

1. Open the following link in a browser: http://localhost:8983/solr/#/collection1_shard1_replica1/query.



2. In the box labeled "q" we have by default `*.*`, click on "Execute Query" at the bottom of the page and Solr will return its entire index.

Note: Choose wt as json and select the indent option in the web GUI to see more human readable output.

The screenshot shows the Solr Admin interface at `localhost:8983/solr/#/collection1_shard1_replica1/query`. The left sidebar lists various Solr components like Dashboard, Logging, Cloud, Core Admin, Java Properties, Thread Dump, and a selected 'collection1_shard1_replica1' node. The main panel has several input fields: 'fq' (empty), 'sort' (empty), 'start', 'rows' (0 to 10), 'df' (empty), and 'Raw Query Parameters' containing `key1=val1&key2=val2`. Below these are checkboxes for 'wt' (set to 'json'), 'indent' (checked), and other Solr parameters like 'dismax', 'edismax', 'hl', 'facet', 'spatial', and 'spellcheck'. A large blue button labeled 'Execute Query' is at the bottom. To the right, the search results are displayed as JSON. The results show three documents with IDs CU1000, CU10000, and CU10011, each with a name, title, and version number.

```

{
  "q": "*:*",
  "fq": "name:RANDELL POLLOCK",
  "sort": "id asc",
  "start": 0,
  "rows": 10,
  "df": "name",
  "wt": "json"
},
{
  "response": {
    "numFound": 1015,
    "start": 0,
    "docs": [
      {
        "id": "CU1000",
        "name": "RANDELL POLLOCK",
        "title": [
          "Not specified"
        ],
        "_version_": 1479215450285932580
      },
      {
        "id": "CU10000",
        "name": "GREGORIO LAWRENCE",
        "title": [
          "Lab Technician"
        ],
        "_version_": 1479215450489356380
      },
      {
        "id": "CU10011",
        "name": "MOZELLA CAREY",
        "title": [
          "PROF-16"
        ],
        "_version_": 1479215450491453480
      }
    ]
  }
}

```

- Lets try some additional queries. The “q” is the query or text to search for; “fq” is for filters. Try these queries :

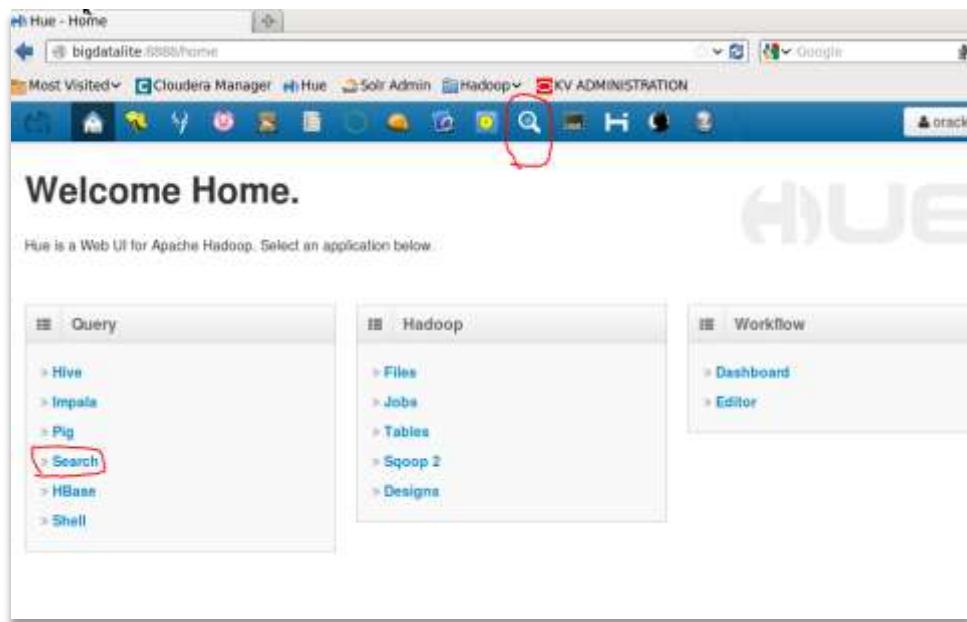
`q = Clerical`

`q= DBA`

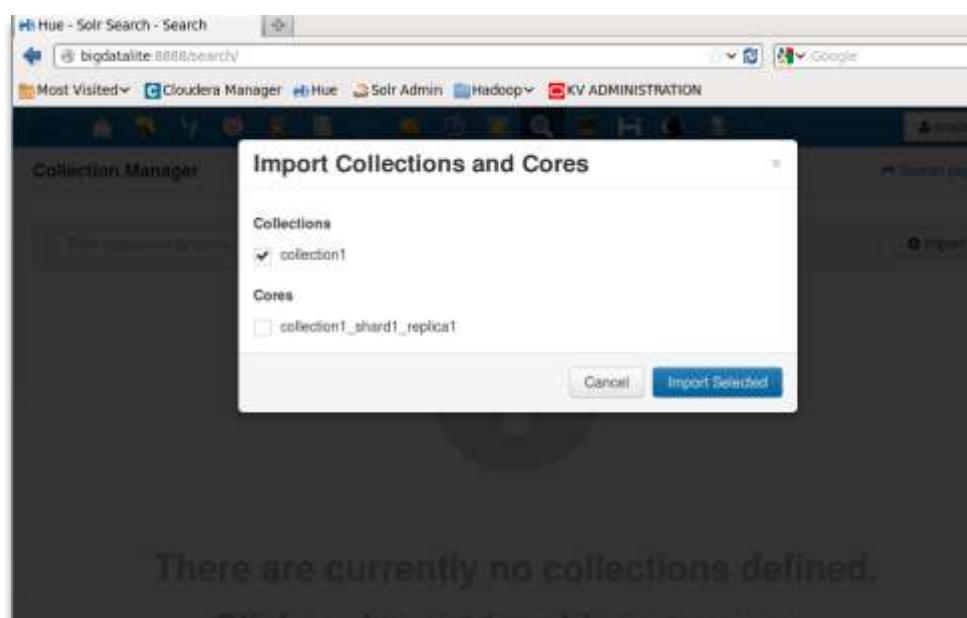
- Try some queries on your own ..

HUE also can be used to work with Solr.

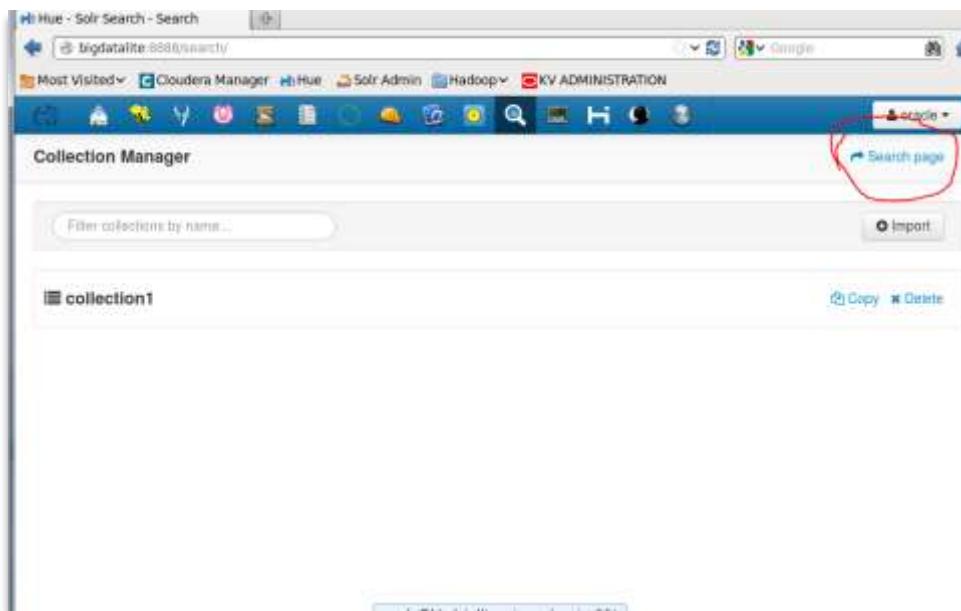
1. Open HUE and go to Solr Search:



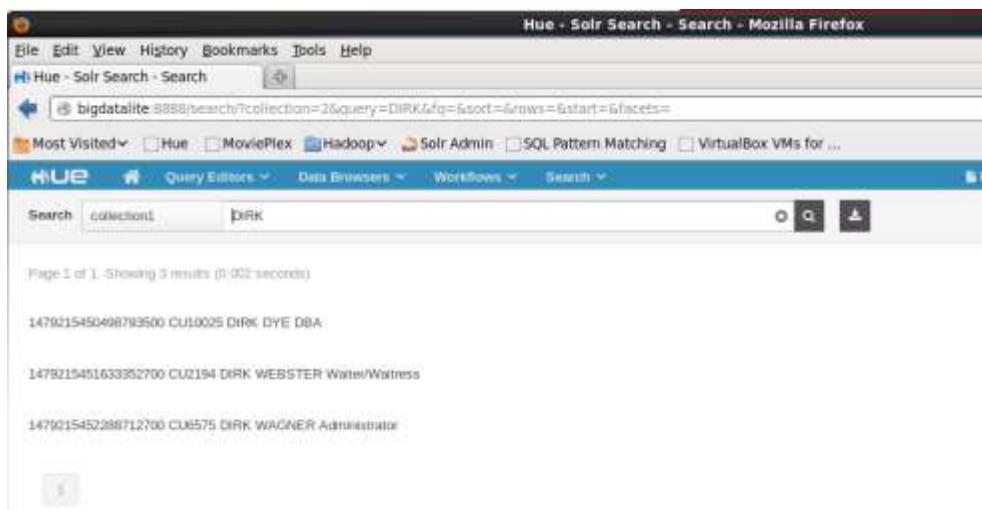
2. Since this is the first time you have used search, it will offer to import known collections into HUE. Check collection1 and click on import selected.



3. You are now presented with a collections page showing you that you have collection1 enabled in HUE. Click on “Search page” to go to the Search screen.



4. Do a simple search such as “DIRK” to file all the customers named DIRK. The default HUE search page will return the entire indexed record.



Now try some searches on your own !

13.5 Summary

Cloudera Search (Solr) can be a very powerful tool to index and search content in your Hadoop environment. The exercise that you just completed was a very simplistic example of what Search can do but you have gone through the necessary steps to create a collection, index it and search, along with customizing the search in HUE in a very short amount of time. Hopefully this exercise has given you a feel for Cloudera Search (Solr) and shown you how you can implement a search index.

14. WORKING WITH SPARK (WORD COUNT REVISITED)

14.1 Introducing Spark

Apache Spark

Fast Analytics and Stream Processing

Apache Spark (incubating) is an open source, parallel data processing framework that complements Apache Hadoop to make it easy to develop fast, unified Big Data applications combining batch, streaming, and interactive analytics on all your data. It was originally developed in 2009 in UC Berkeley's AMPLab, and open sourced in 2010.

Spark Maturity

In subsequent years it has seen rapid adoption, used by enterprises small and large across a wide range of industries. It has quickly become one of the largest open source communities in big data, with over 200 contributors from 50+ organizations.

Speed

Spark is not tied to the two-stage MapReduce paradigm..Spark enables applications in Hadoop clusters to run up to 100x faster in memory, and 10x faster even when running on disk.

Ease of Use

Spark lets you quickly write applications in Java, Scala, or Python. It comes with a built-in set of over 80 high-level operators. And you can use it interactively to query data within the shell.

Sophisticated Analytics

In addition to simple “map” and “reduce” operations, Spark supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms out-of-the-box. Better yet, users can combine all these capabilities seamlessly in a single workflow.

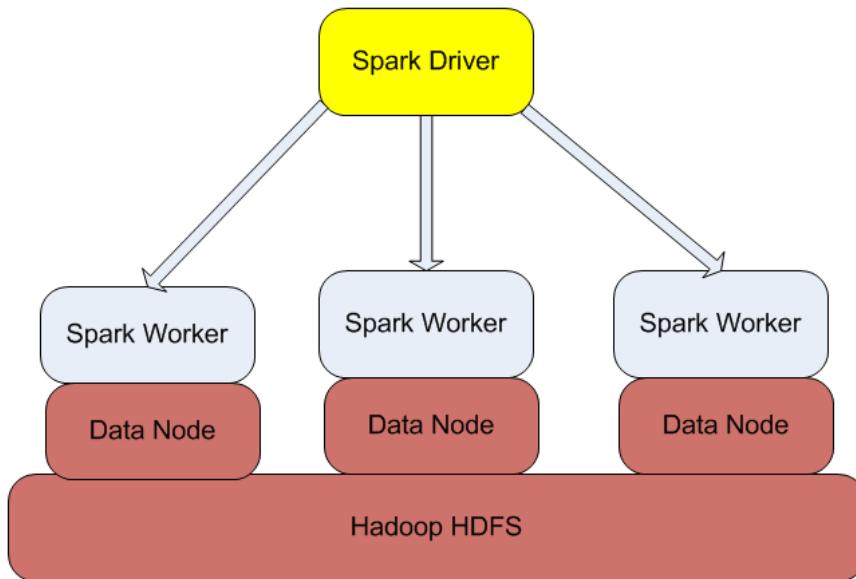
Spark Architecture

From architecture perspective Apache Spark is based on two key concepts; Resilient Distributed Datasets (RDD) and directed acyclic graph (DAG) execution engine. Spark supports two types of RDDs: parallelized collections that are based on existing Scala collections and Hadoop datasets that are created from the files stored on HDFS.

A Resilient Distributed Dataset (RDD), the basic abstraction in Spark. An RDD represents an immutable, partitioned collection of elements that can be operated on in parallel. This class contains the basic operations available on all RDDs, such as map, filter, and persist. In addition, PairRDDFunctions contains operations available only on RDDs of key-value pairs, such as groupByKey and join.

Internally, each RDD is characterized by five main properties:

- A list of splits (partitions)
 - A function for computing each split
 - A list of dependencies on other RDDs
 - Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
 - Optionally, a list of preferred locations to compute each split on (e.g. block locations for an HDFS file)
- The DAG engine helps to eliminate the MapReduce multi-stage execution model and offers significant performance improvements.



Scala Language

Scala's approach is to develop a small set of core constructs that can be combined in flexible ways. Scala smoothly integrates object-oriented and functional programming. It is designed to express common programming patterns in a concise, elegant, and type-safe way.

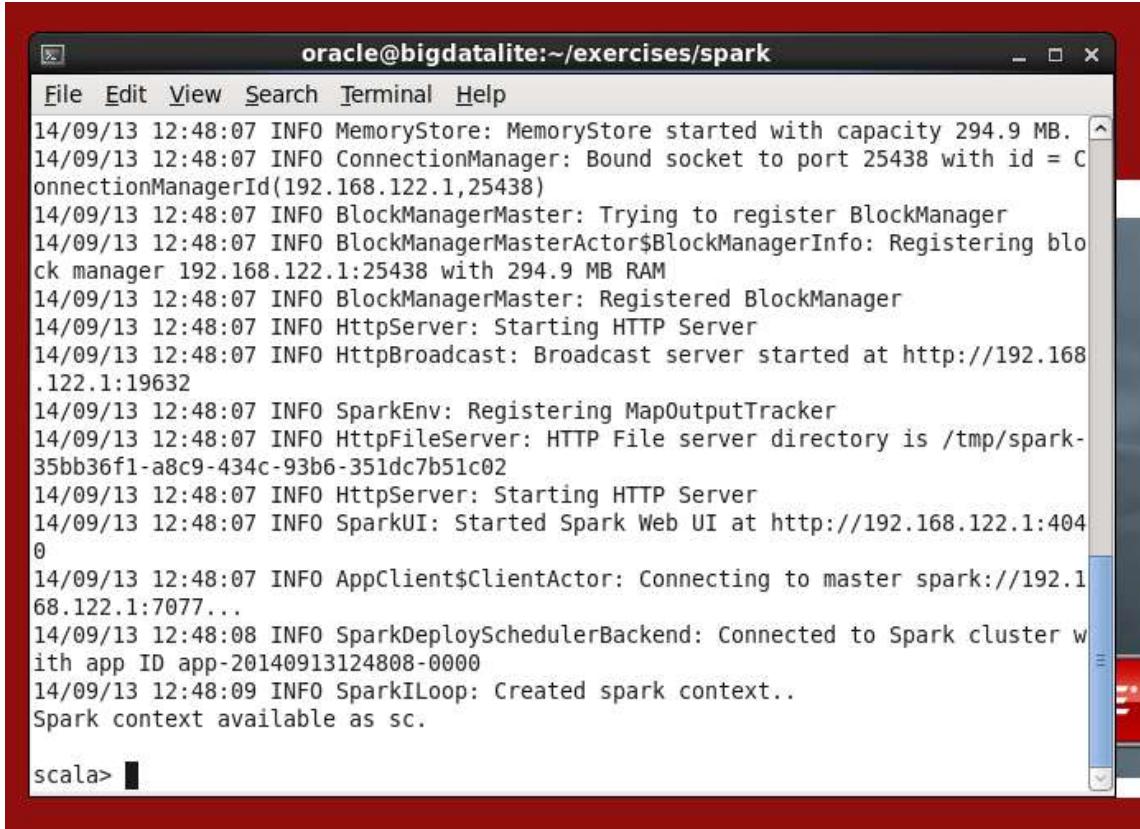
14.2 Exercises

We will revisit the Word Count exercise but this time we will do the word count in Scala and also Python using the interactive Spark shells “spark-shell” for Scala and “pyspark” for Python.

Scala word count

In this exercise we are working with spark-shell interactively.

1. Change directory to the exercises/spark folder, Run spark-shell at the command line to open the interactive scala shell.



```
oracle@bigdatalite:~/exercises/spark
File Edit View Search Terminal Help
14/09/13 12:48:07 INFO MemoryStore: MemoryStore started with capacity 294.9 MB.
14/09/13 12:48:07 INFO ConnectionManager: Bound socket to port 25438 with id = ConnectionManagerId(192.168.122.1,25438)
14/09/13 12:48:07 INFO BlockManagerMaster: Trying to register BlockManager
14/09/13 12:48:07 INFO BlockManagerMasterActor$BlockManagerInfo: Registering block manager 192.168.122.1:25438 with 294.9 MB RAM
14/09/13 12:48:07 INFO BlockManagerMaster: Registered BlockManager
14/09/13 12:48:07 INFO HttpServer: Starting HTTP Server
14/09/13 12:48:07 INFO HttpBroadcast: Broadcast server started at http://192.168.122.1:19632
14/09/13 12:48:07 INFO SparkEnv: Registering MapOutputTracker
14/09/13 12:48:07 INFO HttpFileServer: HTTP File server directory is /tmp/spark-35bb36f1-a8c9-434c-93b6-351dc7b51c02
14/09/13 12:48:07 INFO HttpServer: Starting HTTP Server
14/09/13 12:48:07 INFO SparkUI: Started Spark Web UI at http://192.168.122.1:4040
14/09/13 12:48:07 INFO AppClient$ClientActor: Connecting to master spark://192.168.122.1:7077...
14/09/13 12:48:08 INFO SparkDeploySchedulerBackend: Connected to Spark cluster with app ID app-20140913124808-0000
14/09/13 12:48:09 INFO SparkILoop: Created spark context..
Spark context available as sc.

scala>
```

2. At the scala> prompt enter the following :

```
val files = sc.textFile("hdfs://bigdatalite:8020/user/oracle/wordcount/input")
```

to open our 2 files from the word count exercise and get a handle to the file data.

3. Enter at the scala> prompt :

```
val counts = files.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
```

To set up the execution context for the file data.

4. Next enter :

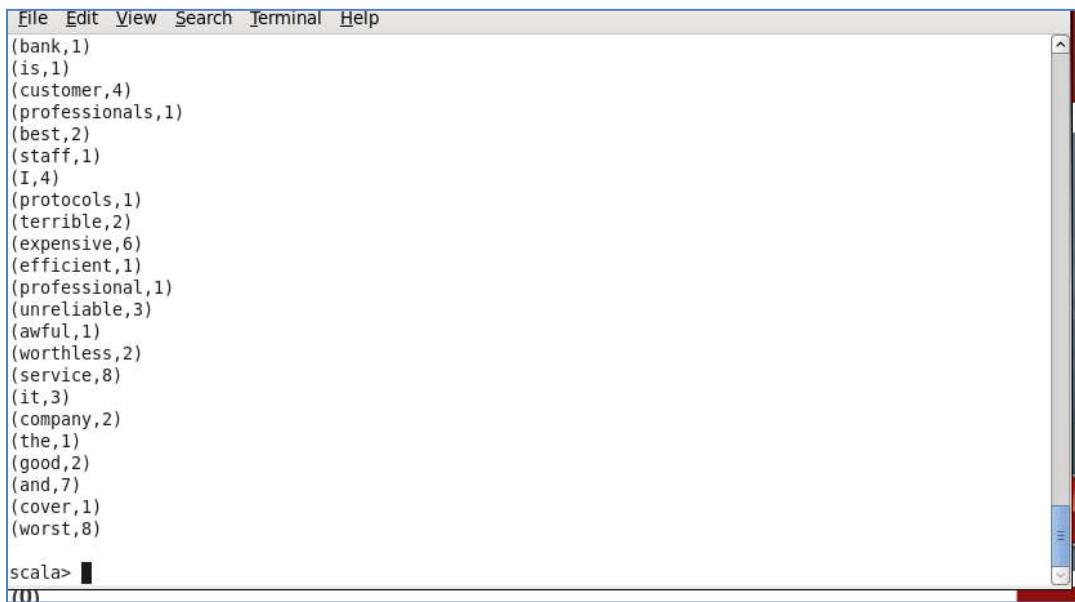
```
counts.saveAsTextFile("hdfs://bigdatalite:8020/user/oracle/wordcount/sp-out")
```

To execute the counts and save the output to a file.

```
counts.collect().foreach(println)
```

This will print out the counts to the console.

5. Lets check our counts :



```
File Edit View Search Terminal Help
(bank,1)
(is,1)
(customer,4)
(professionals,1)
(best,2)
(staff,1)
(I,4)
(protocols,1)
(terrible,2)
(expensive,6)
(efficient,1)
(professional,1)
(unreliable,3)
(awful,1)
(worthless,2)
(service,8)
(it,3)
(company,2)
(the,1)
(good,2)
(and,7)
(cover,1)
(worst,8)

scala> [REDACTED]
701
```

6. Lets check on the files that were created :



```
oracle@bigdatalite:~/exercises/spark
File Edit View Search Terminal Help
[oracle@bigdatalite spark]$ hadoop fs -ls wordcount/sp-out
Found 3 items
-rw-r--r-- 1 oracle oracle          0 2014-09-14 02:35 wordcount/sp-out/_SUCCESS
-rw-r--r-- 1 oracle oracle      149 2014-09-14 02:35 wordcount/sp-out/part-00000
-rw-r--r-- 1 oracle oracle      193 2014-09-14 02:35 wordcount/sp-out/part-00001
[oracle@bigdatalite spark]$ [REDACTED]
```

7. We can also look at the execution by checking the local executor's webpage at bigdatalite.localdomain:4040. This webpage is generated from spark-shell.

The screenshot shows the "Spark Stages" section of the Spark UI. It displays the following statistics:

- Total Duration: 1.1 m
- Scheduling Mode: FIFO
- Active Stages: 0
- Completed Stages: 2
- Failed Stages: 0

Active Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write
0	collect at <console>:17	2014/09/14 02:48:52	175 ms	1/1		
1	reduceByKey at <console>:14	2014/09/14 02:48:51	932 ms	1/1		1095.0 B

Failed Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write

- Type in :quit at the scala> prompt – and we are done with the scala example.

Python and pyspark

In this exercise we will work with python in the pyspark shell interactively

- Run pyspark at the command line to open the pyspark shell

```
oracle@bigdatalite:~/exercises/spark
File Edit View Search Terminal Help
da2
14/09/14 02:55:31 INFO MemoryStore: MemoryStore started with capacity 294.9 MB.
14/09/14 02:55:31 INFO ConnectionManager: Bound socket to port 22455 with id = ConnectionManagerId(bi
gdatalite.localdomain,22455)
14/09/14 02:55:31 INFO BlockManagerMaster: Trying to register BlockManager
14/09/14 02:55:31 INFO BlockManagerMasterActor$BlockManagerInfo: Registering block manager bigdatalit
e.localdomain:22455 with 294.9 MB RAM
14/09/14 02:55:31 INFO BlockManagerMaster: Registered BlockManager
14/09/14 02:55:31 INFO HttpServer: Starting HTTP Server
14/09/14 02:55:32 INFO HttpBroadcast: Broadcast server started at http://127.0.0.1:36827
14/09/14 02:55:32 INFO SparkEnv: Registering MapOutputTracker
14/09/14 02:55:32 INFO HttpFileServer: HTTP File server directory is /tmp/spark-32ab5c4b-8d3b-4ab9-a2
53-ae8d10e3f41a
14/09/14 02:55:32 INFO HttpServer: Starting HTTP Server
14/09/14 02:55:32 INFO SparkUI: Started Spark Web UI at http://bigdatalite.localdomain:4040
Welcome to
t
n
version 0.9.0
Using Python version 2.6.6 (r266:84292, Jul 10 2013 06:42:56)
Spark context available as sc.
>>>
```

- Just like we did for scala in the spark-shell, enter in the following code line by at the >>> prompt:

```

files = sc.textFile("hdfs://bigdatalite.localdomain:8020/user/oracle/wordcount/input")
words = files.flatMap(lambda x: x.split())
wordcounts = words.map(lambda x: (x, 1)).reduceByKey(lambda x,y:x+y).map(lambda x:(x[1],x[0])).sortByKey(False)
wordcounts.saveAsTextFile("hdfs://bigdatalite.localdomain:8020/user/oracle/wordcount/pysp-out")
output = wordcounts.collect()
for (count, word) in output:
    print "%s: %i" % (word, count)

```

3. You will need hit return twice after the last line of code – and be care to use a TAB for the indent for the for loop.
4. Let check our counts :

```

customer: 4
it: 3
recommend: 3
very: 3
unreliable: 3
disappointed: 3
worthless: 2
with: 2
best: 2
good: 2
company: 2
terrible: 2
is: 1
staff: 1
efficient: 1
protocols: 1
found: 1
awful: 1
bank: 1
professionals: 1
cover: 1
will: 1
professional: 1
the: 1
>>>

```

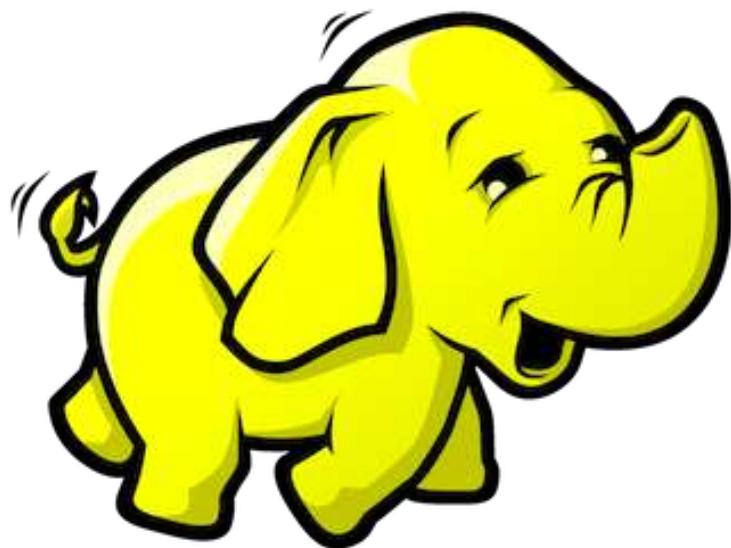
5. Type exit() and the >>> prompt and we are complete.

14.3 Summary

You should have noticed 2 important things in this lesson:

1. The scala and python code was much faster, avg less than a second to execute the same wordcount we did in lesson which took over 45 sec to execute.
2. Simplicity: the word count in exercise 1 was over 50 lines of java code. With spark our word count scripts were 3 or 4 lines.

This concludes our workshop, thank you for attending!



A. SKIP OF ODI EXERCISE

A.1 Running the cheat script for the ODI exercise

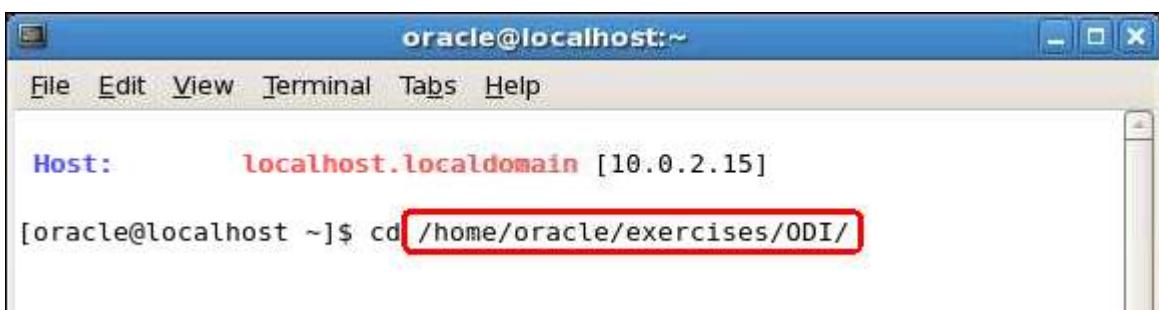
1. This entire process can be done from the terminal, hence open a terminal by double clicking on the **Terminal icon** on the desktop.



2. To get into the folder where the scripts for the ODI exercise are, type in the terminal:

```
cd /home/oracle/exercises/ODI/
```

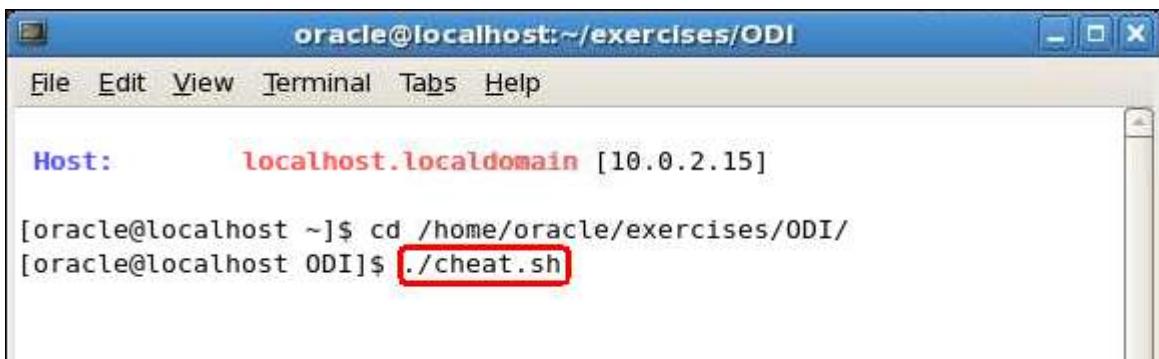
Then press **Enter**



3. Next we need to run the cheat script. This script loads the Hive data into the Oracle Database using the Oracle Loader for Hadoop. Go to the terminal and type:

```
./cheat.sh
```

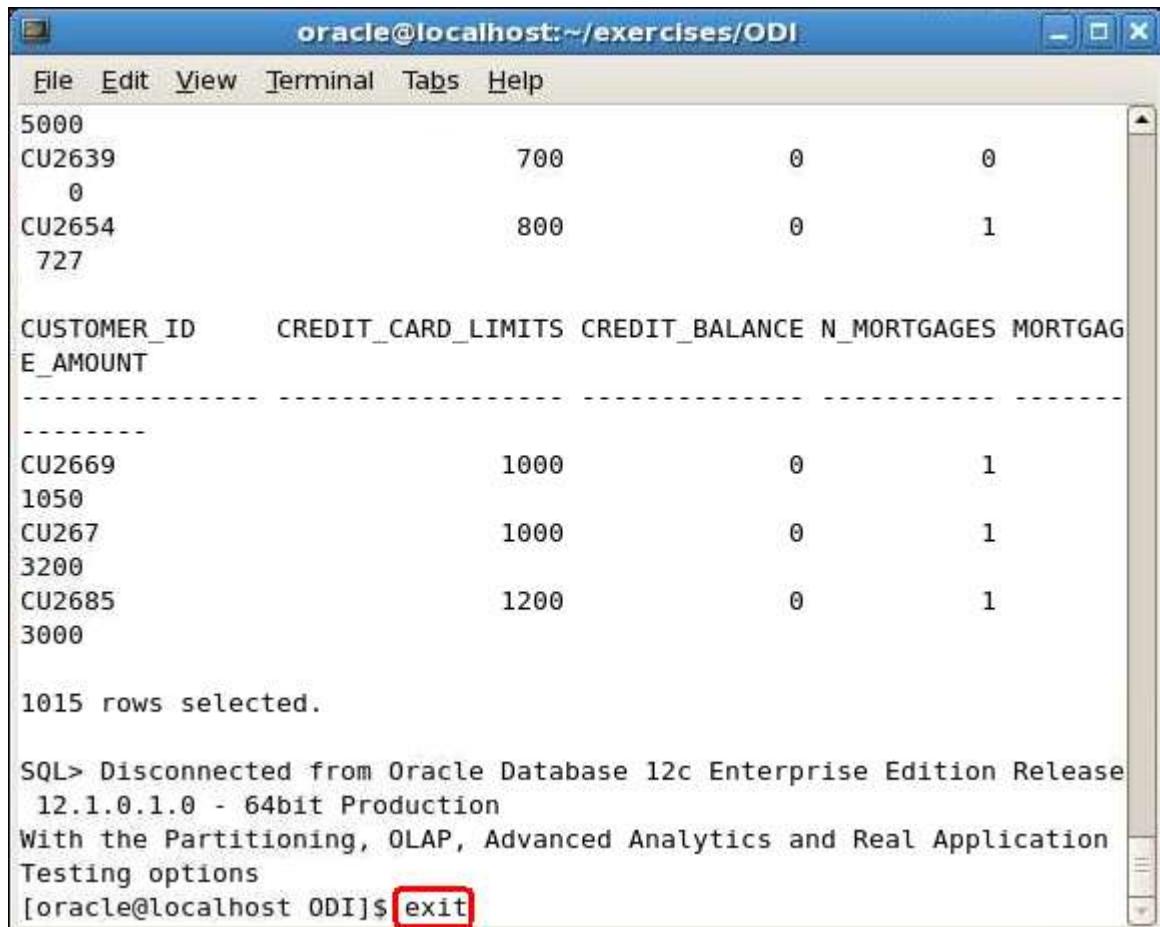
Then press **Enter**



4. Once the script is done running the exercises is complete. When you get the prompt back go the terminal and type:

```
exit
```

Then press **Enter**



The screenshot shows a terminal window titled "oracle@localhost:~/exercises/ODI". The window displays the results of a SQL query and the final command to exit the session.

```
oracle@localhost:~/exercises/ODI
File Edit View Terminal Tabs Help
5000
CU2639          700      0      0
  0
CU2654          800      0      1
  727

CUSTOMER_ID      CREDIT_CARD_LIMITS CREDIT_BALANCE N_MORTGAGES MORTGAG
E_AMOUNT
-----
CU2669          1000      0      1
1050
CU267           1000      0      1
3200
CU2685          1200      0      1
3000

1015 rows selected.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@localhost ODI]$ exit
```

This completes the exercises bypass.