NAME: Meith Navlakha
SAP: 60004190068
BDI Practical

EXPERIMENT 4

AIM: Execute HIVE commands to load, insert, retrieve, update, or delete data in the tables.

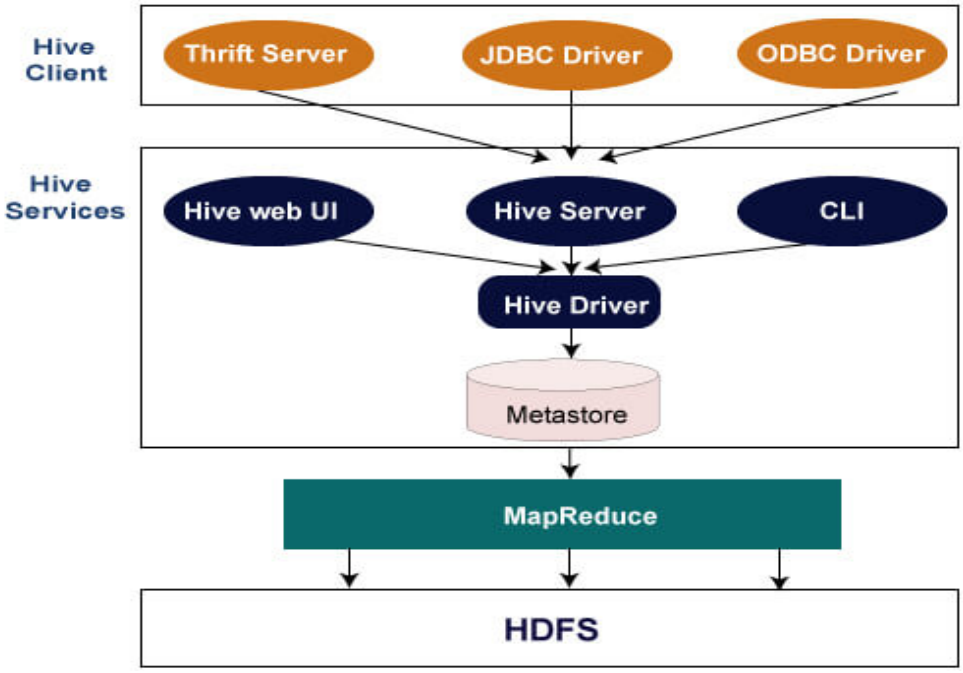
THEORY:

Hive is a data warehouse system which is used to analyze structured data. It is built on the top of Hadoop. Hive provides the functionality of reading, writing, and managing large datasets residing in distributed storage. It runs SQL like queries called HQL (Hive query language) which gets internally converted to MapReduce jobs.
Using Hive, we can skip the requirement of the traditional approach of writing complex MapReduce programs. Hive supports Data Definition Language (DDL), Data Manipulation Language (DML), and User Defined Functions (UDF).

Features of Hive:

- Hive is fast and scalable.
- It provides SQL-like queries (i.e., HQL) that are implicitly transformed to MapReduce or Spark jobs.
- It is capable of analyzing large datasets stored in HDFS.
- It allows different storage types such as plain text, RCFile, and HBase.
- It uses indexing to accelerate queries.
- It can operate on compressed data stored in the Hadoop ecosystem.

Architecture of Hive:



Hive Client

Hive allows writing applications in various languages, including Java, Python, and C++. It supports different types of clients such as:-

- **Thrift Server** - It is a cross-language service provider platform that serves the request from all those programming languages that supports Thrift.
- **JDBC Driver** - It is used to establish a connection between hive and Java applications.
- **ODBC Driver** - It allows the applications that support the ODBC protocol to connect to Hive.

Hive Services

The following are the services provided by Hive:-

- **Hive CLI -** The Hive CLI (Command Line Interface) is a shell where we can execute Hive queries and commands.
- **Hive Web User Interface -** The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
- **Hive MetaStore -** It is a central repository that stores all the structure information of various tables and partitions in the warehouse. It also includes metadata of column and its type information, the serializers and deserializers which is used to read and write data and the corresponding HDFS files where the data is stored.
- **Hive Server -** It is referred to as Apache Thrift Server. It accepts the request from different clients and provides it to Hive Driver.
- **Hive Driver -** It receives queries from different sources like web UI, CLI, Thrift, and JDBC/ODBC driver. It transfers the queries to the compiler.
- **Hive Compiler -** The purpose of the compiler is to parse the query and perform semantic analysis on the different query blocks and expressions. It converts HiveQL statements into MapReduce jobs.
- **Hive Execution Engine -** Optimizer generates the logical plan in the form of DAG of map-reduce tasks and HDFS tasks. In the end, the execution engine executes the incoming tasks in the order of their dependencies.

Working of Hive:

Step 1: executeQuery: The user interface calls the execute interface to the driver.
Step 2: getPlan: The driver accepts the query, creates a session handle for the query, and passes the query to the compiler for generating the execution plan.
Step 3: getMetaData: The compiler sends the metadata request to the metastore.
Step 4: sendMetaData: The metastore sends the metadata to the compiler.

The compiler uses this metadata for performing type-checking and semantic analysis on the expressions in the query tree. The compiler then generates the execution plan (**Directed acyclic Graph**). For Map Reduce jobs, the plan contains **map operator trees** (operator trees which are executed on mapper) and **reduce operator tree** (operator trees which are executed on reducer).

Step 5: sendPlan: The compiler then sends the generated execution plan to the driver.

Step 6: executePlan: After receiving the execution plan from compiler, driver sends the execution plan to the execution engine for executing the plan.

Step 7: submit job to MapReduce: The execution engine then sends these stages of DAG to appropriate components.

For each task, either mapper or reducer, the deserializer associated with a table or intermediate output is used in order to read the rows from HDFS files. These are then passed through the associated operator tree.

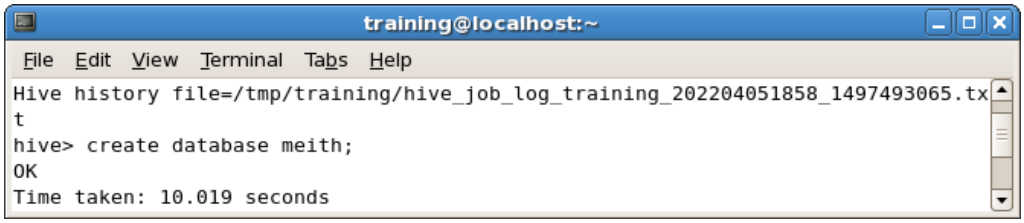
Once the output gets generated, it is then written to the HDFS temporary file through the serializer. These temporary HDFS files are then used to provide data to the subsequent map/reduce stages of the plan.

For DML operations, the final temporary file is then moved to the table's location.

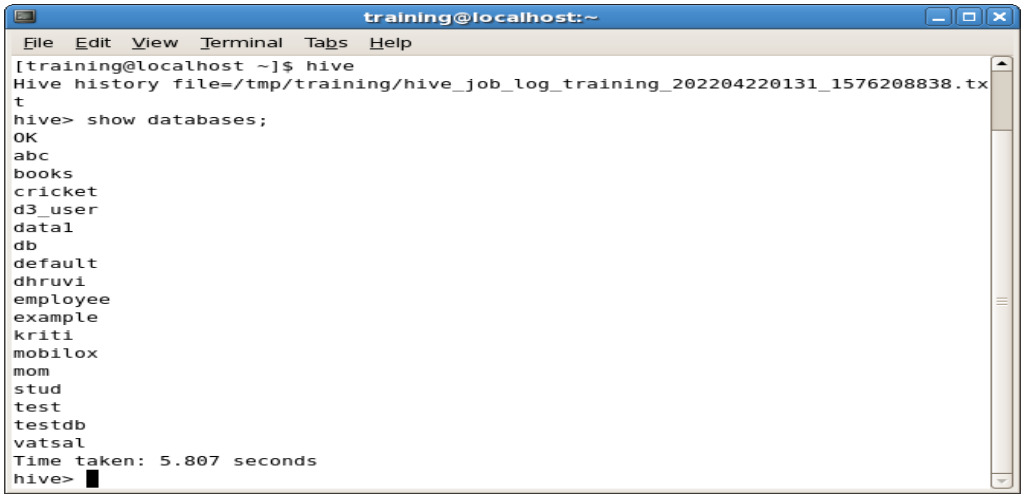
Step 8, 9, 10: sendResult: Now for queries, the execution engine reads the contents of the temporary files directly from HDFS as part of a fetch call from the driver. The driver then sends results to the Hive interface.

QUERIES:

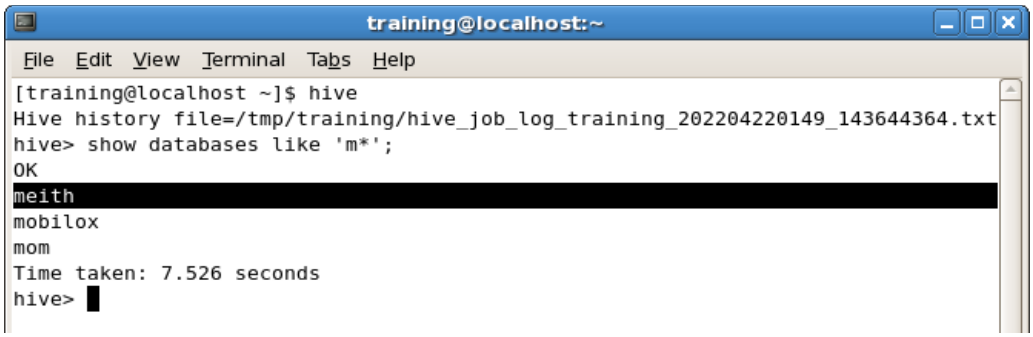
1) create database <database_name>



2) show databases

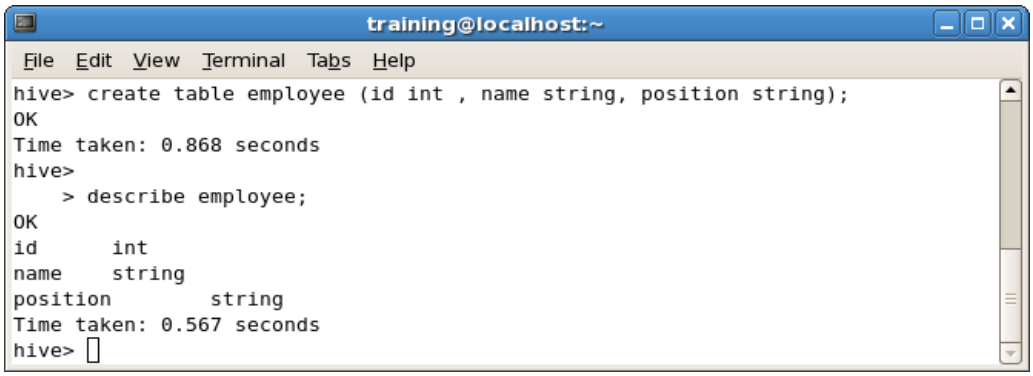


NAME: Meith Navlakha
SAP: 60004190068
BDI Practical

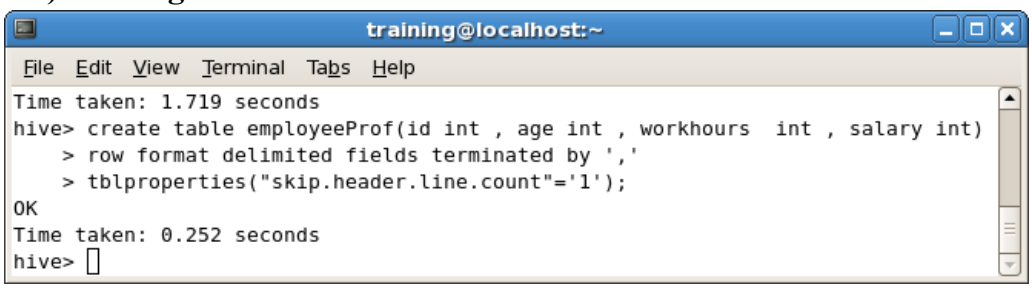


3) Create Table

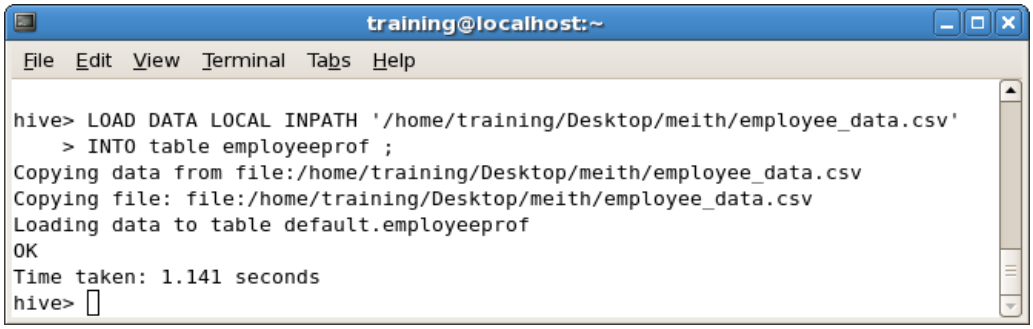
3.1) Basic Table



3.2) Using Delimiter



4) Copy data into table



NAME: Meith Navlakha
SAP: 60004190068
BDI Practical

```
training@localhost:~
File  Edit  View  Terminal  Tabs  Help
Time taken: 0.677 seconds
hive> LOAD DATA LOCAL INPATH '/home/training/Desktop/meith/names.csv' into t
able EmpPersonal ;
Copying data from file:/home/training/Desktop/meith/names.csv
Copying file: file:/home/training/Desktop/meith/names.csv
Loading data to table default.emppersonal
OK
Time taken: 0.555 seconds
hive>
```

5) Display data of table

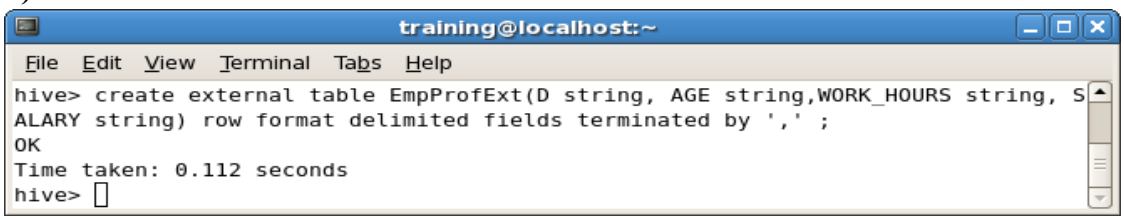
```
training@localhost:~
File  Edit  View  Terminal  Tabs  Help
Time taken: 0.365 seconds
hive> SELECT * FROM EmpProf LIMIT 15;
OK
ID      AGE     WORK_HOURS      SALARY
0       0       36      5
1       1       55      5
2       2       61      1
3       3       29      6
4       4       34      2
5       5       42      3
6       6       53      6
7       7       41      6
8       8       47      6
9       9       31      8
10      10      47      9
11      11      40      1
12      12      41      2
13      13      46      8
Time taken: 0.309 seconds
hive>
```

6) Describe table

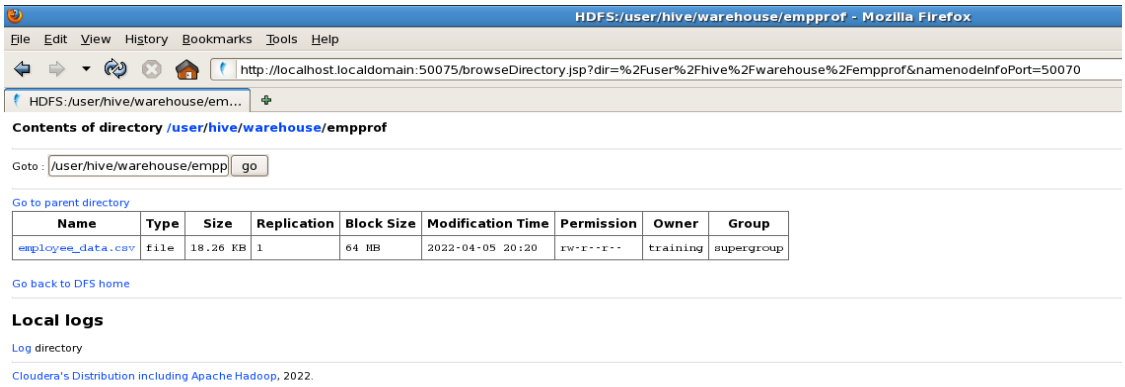
```
training@localhost:~
File  Edit  View  Terminal  Tabs  Help
hive> describe formatted emp;
OK
# col_name              data_type               comment

employee_id             string                  None
first_name              string                  None
last_name               string                  None
email                   string                  None
phone_number            string                  None

# Detailed Table Information
Database:               rahuldb
Owner:                  training
CreateTime:             Tue Apr 05 20:14:19 PDT 2022
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://localhost/user/hive/warehouse/
Table Type:             MANAGED_TABLE
Table Parameters:
        transient_lastDdlTime   1649214859

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             ,
        serialization.format    ,
Time taken: 0.439 seconds
hive>
```

NAME: Meith Navlakha
SAP: 60004190068
BDI Practical

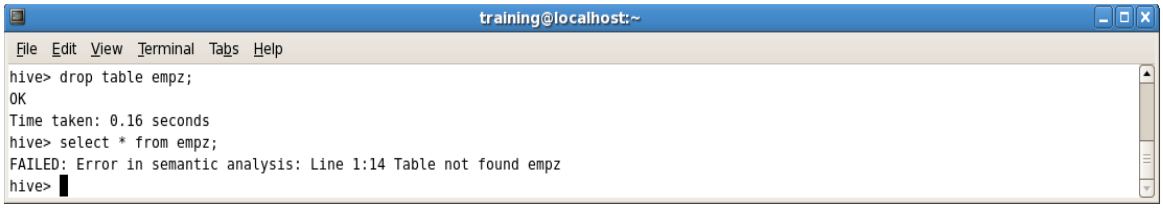
7) Create External Table



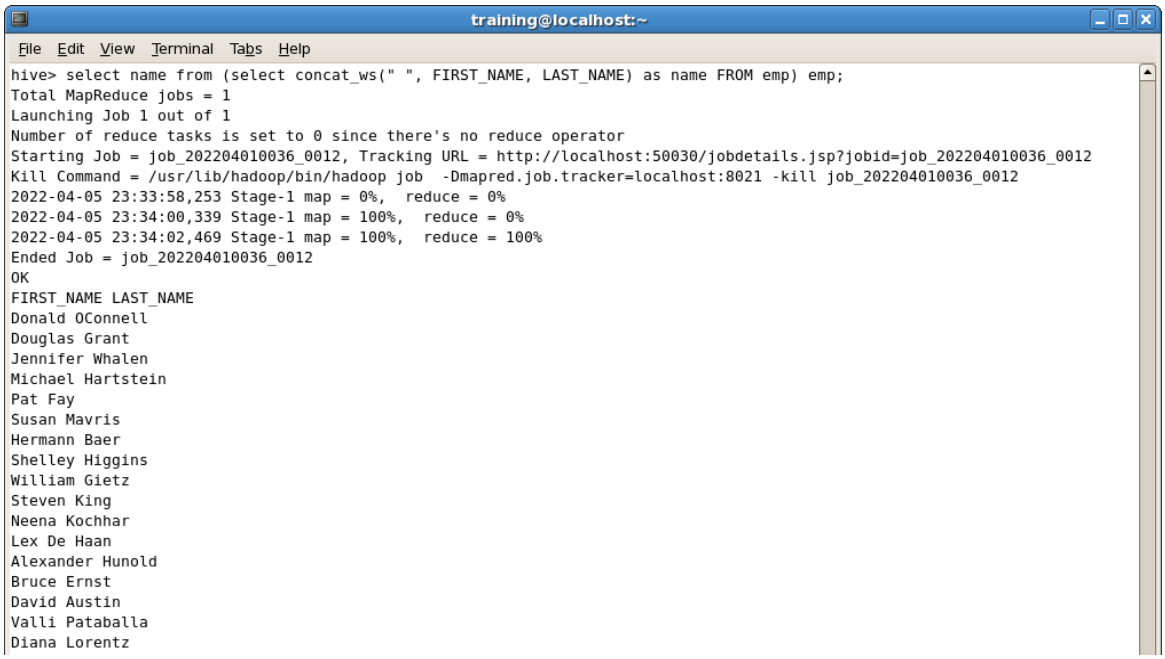
8) External Table in Web Browser UI



9) Drop Table



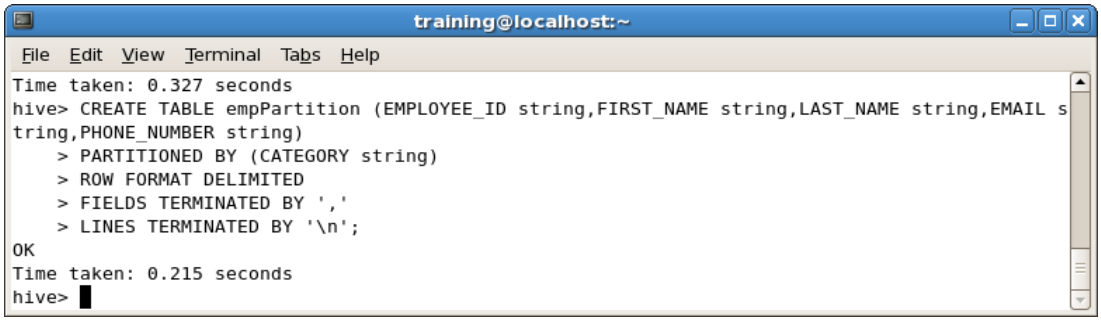
10) Nested Query



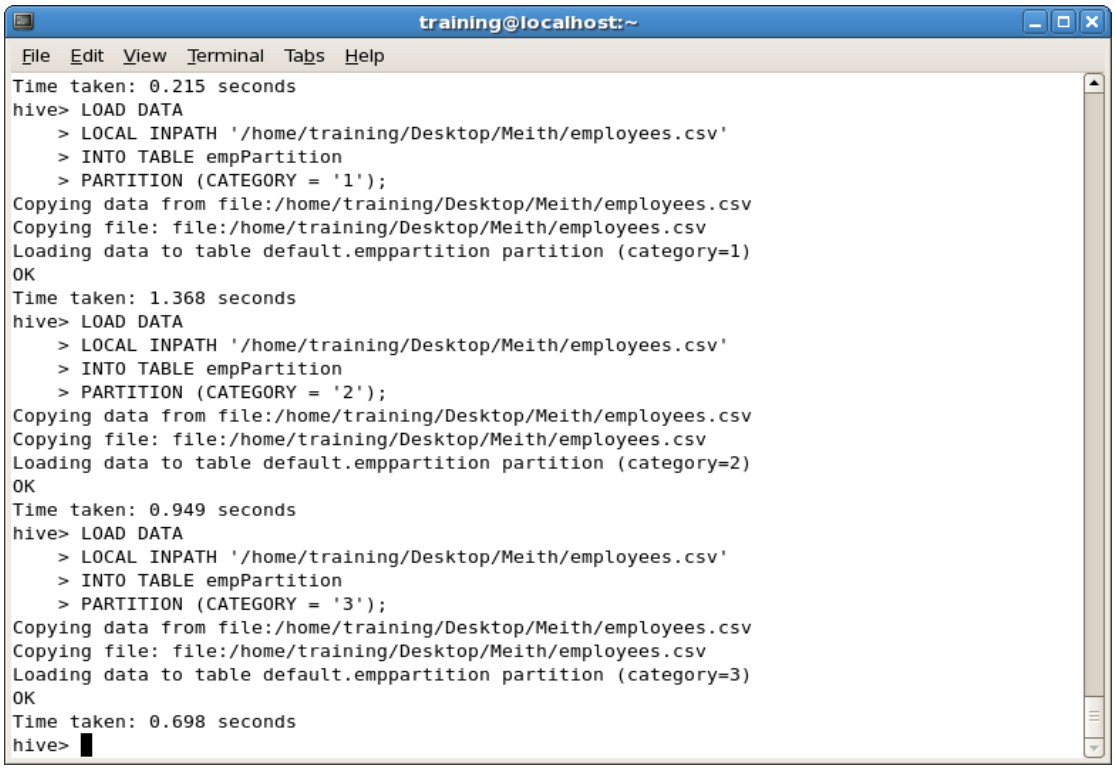
NAME: Meith Navlakha
SAP: 60004190068
BDI Practical

11)Static Partition

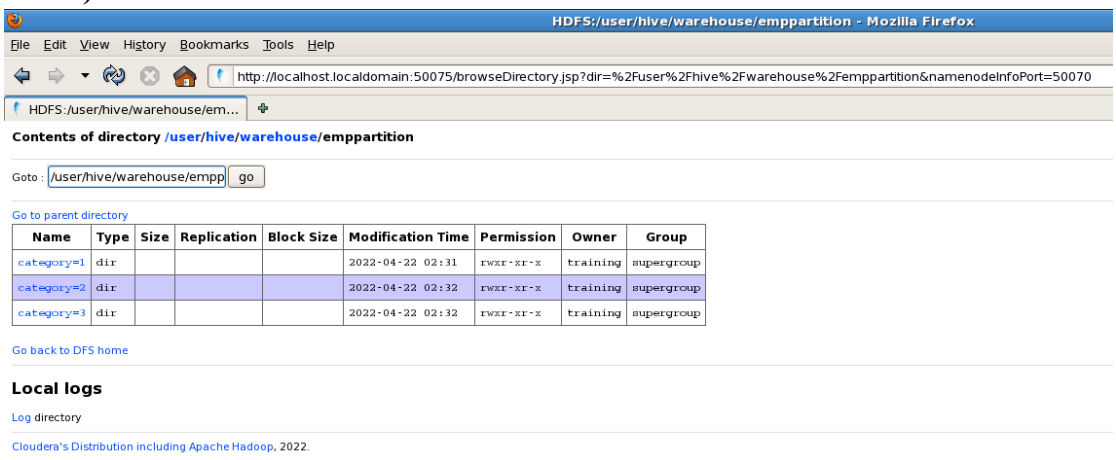
11.1) Static Partitioning



11.2) Load Data into Partitions



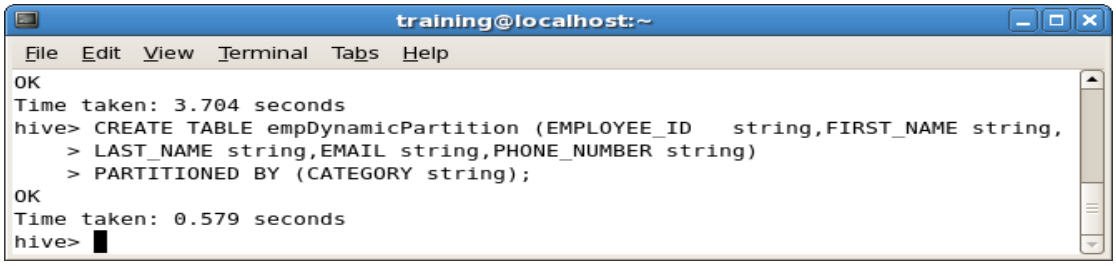
11.3) Static Partitions on Web Browser



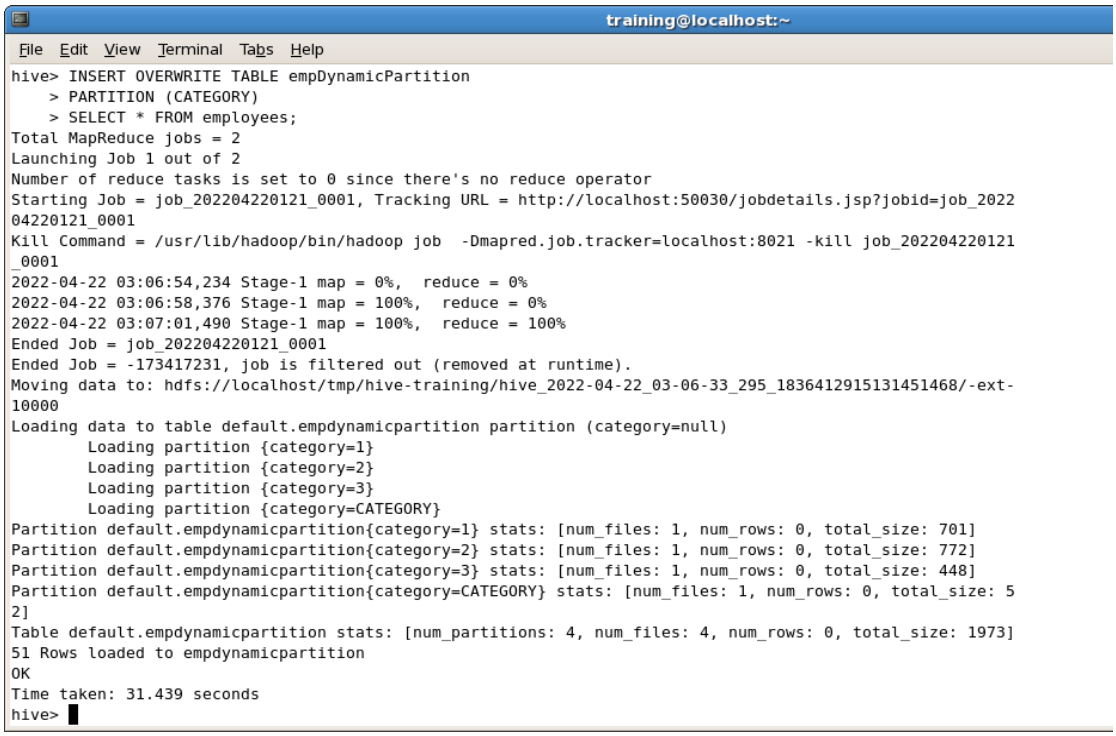
NAME: Meith Navlakha
SAP: 60004190068
BDI Practical

12)Dynamic Partition

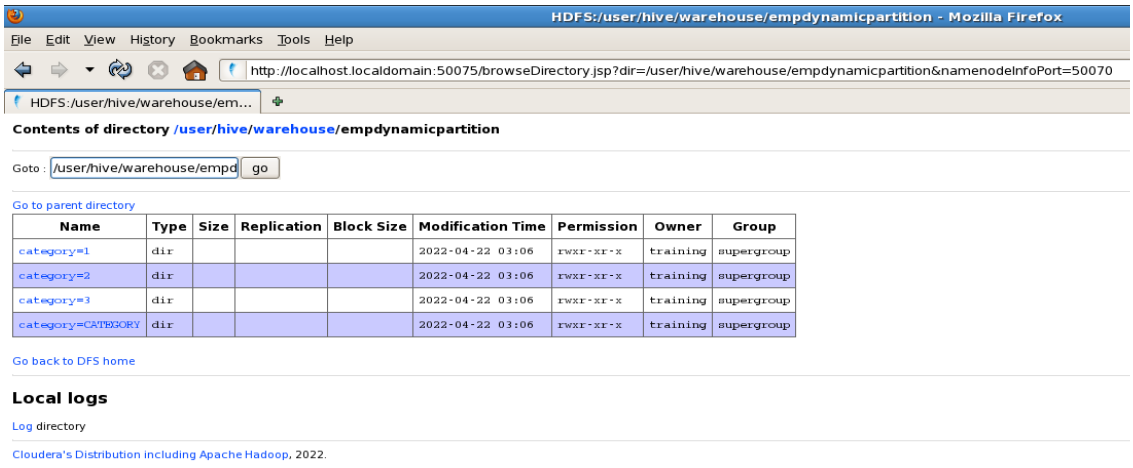
12.1) Dynamic Partitioning



12.2) Load Data into Partitions



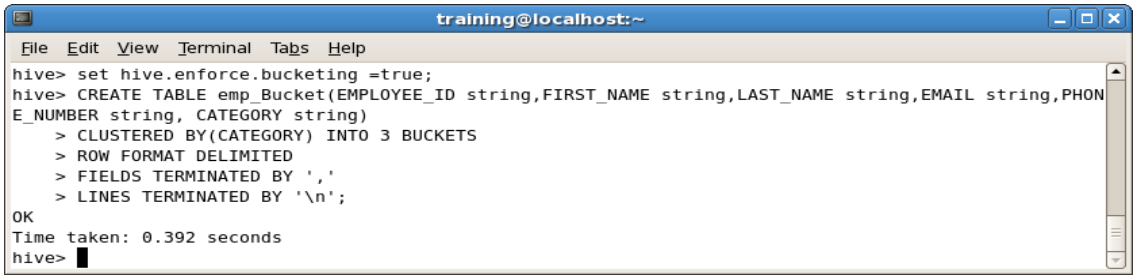
12.3) Dynamic Partitions on Web Browser



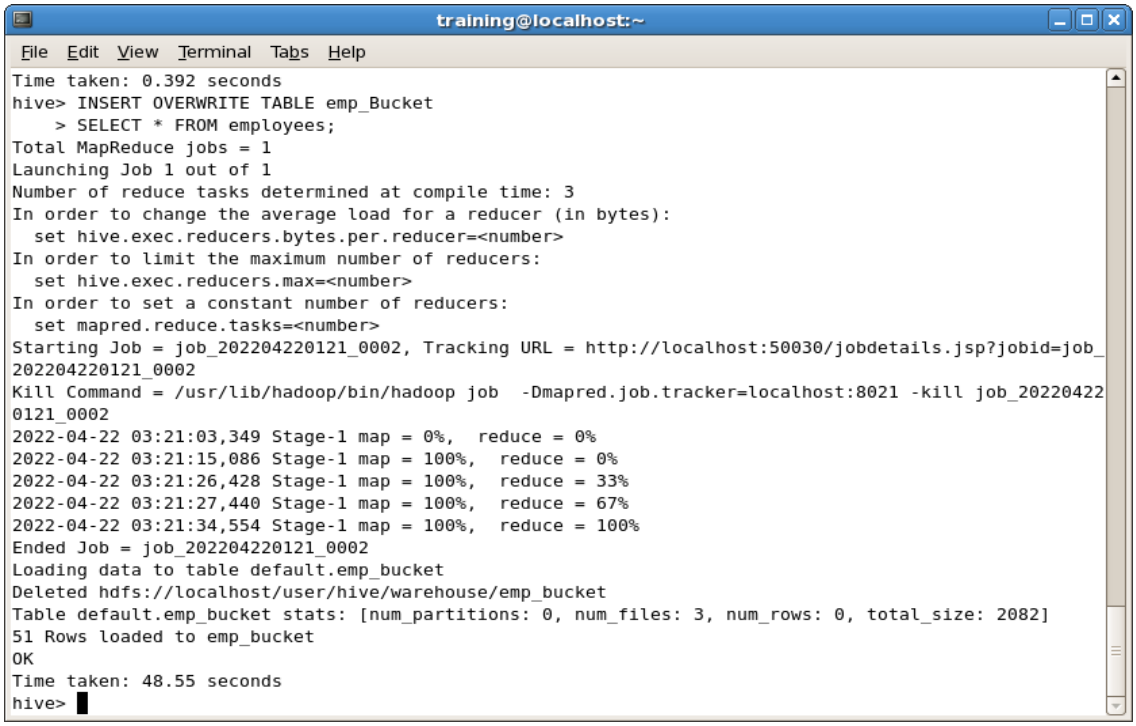
NAME: Meith Navlakha
SAP: 60004190068
BDI Practical

13) Bucketing

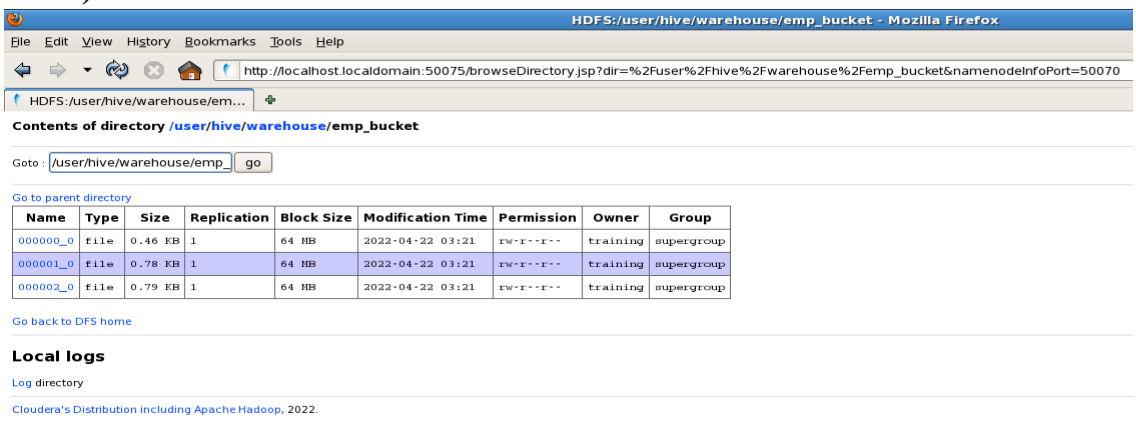
13.1) Create Buckets



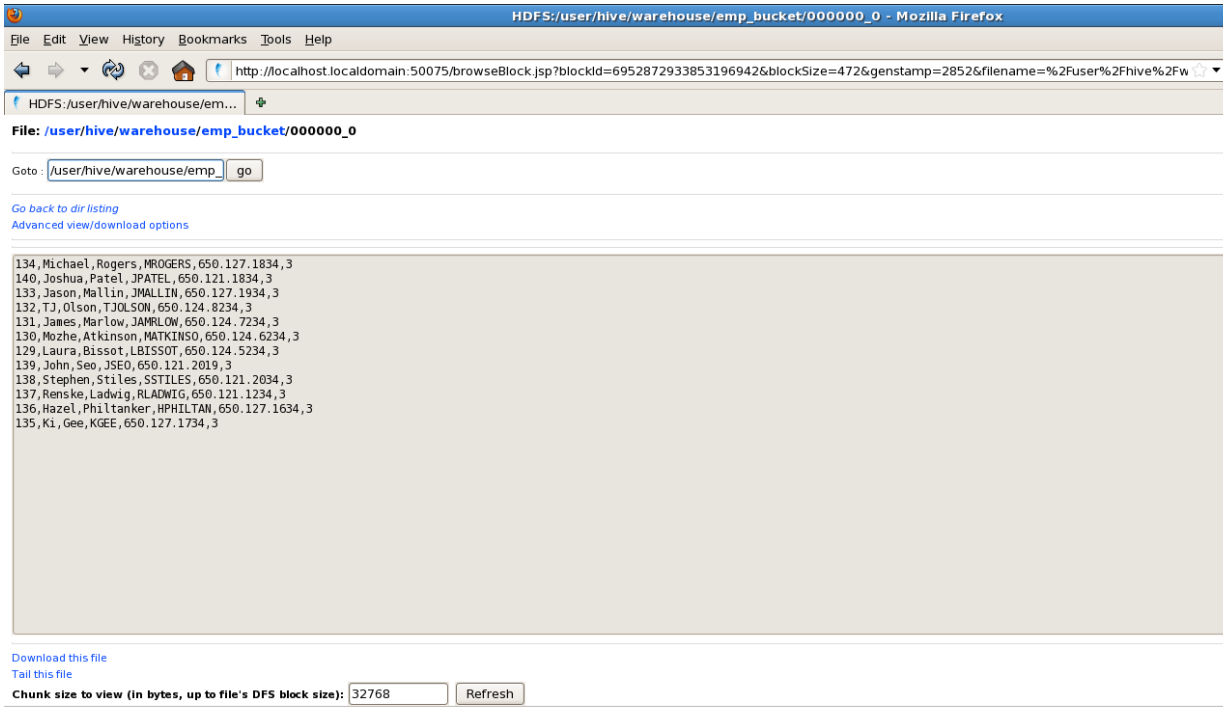
13.2) Load Data into Buckets



13.3) Buckets on Web Browser

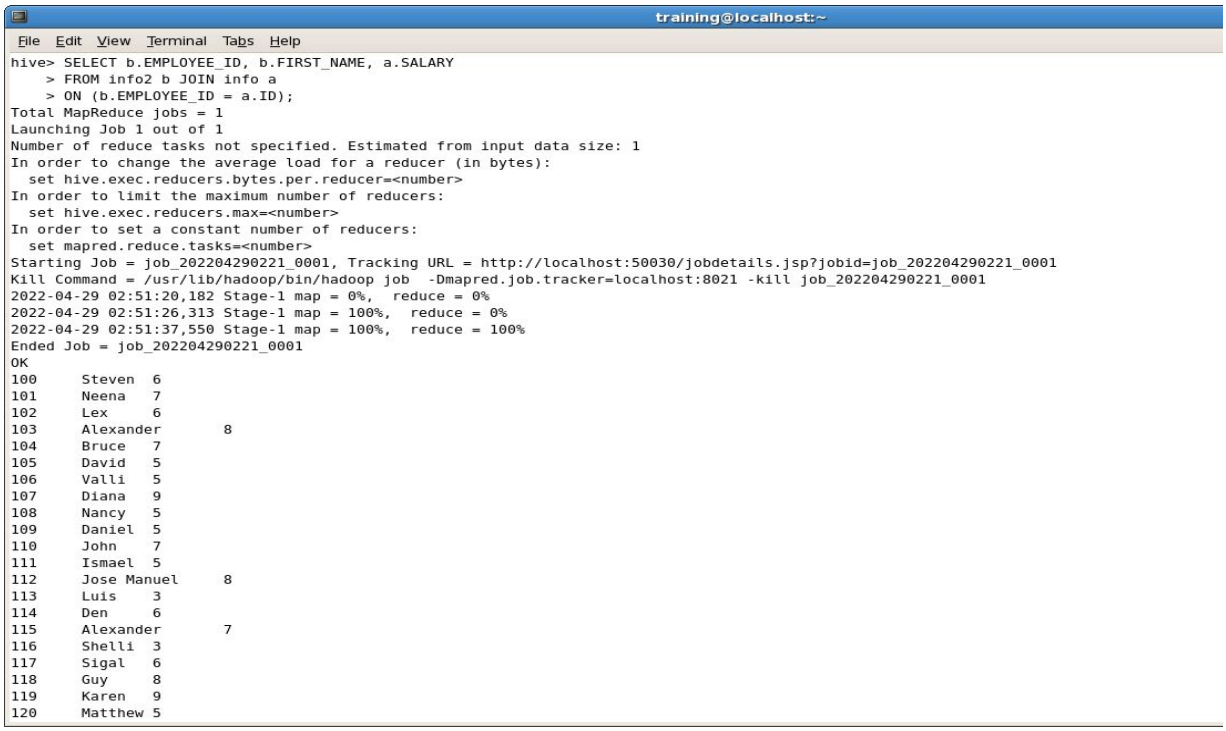


NAME: Meith Navlakha
SAP: 60004190068
BDI Practical



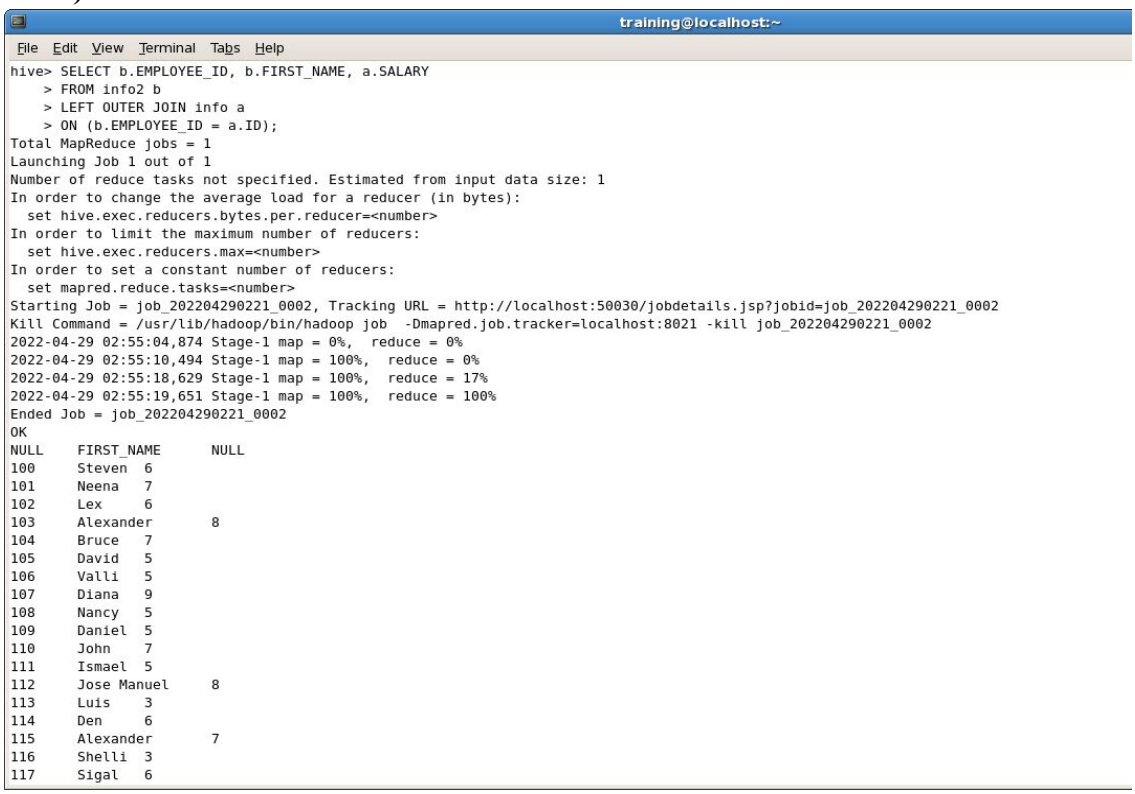
14) JOINS

14.1) Inner join

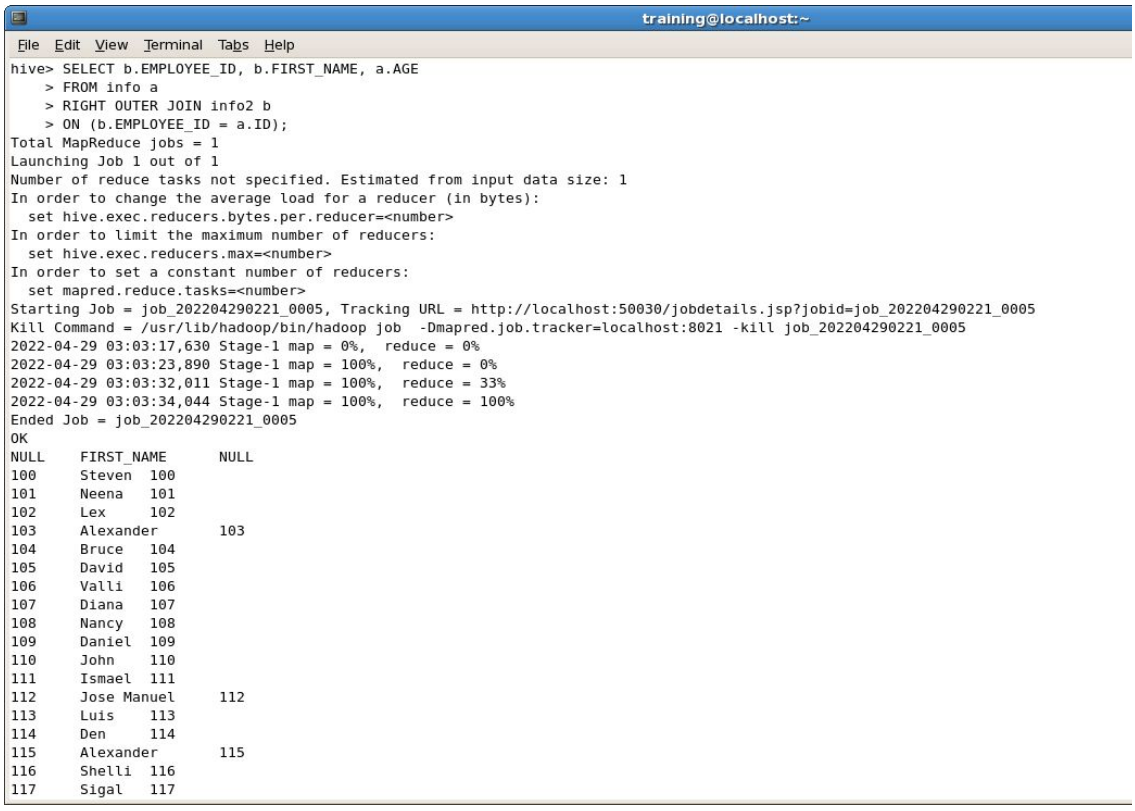


NAME: Meith Navlakha
SAP: 60004190068
BDI Practical

14.2) Left Outer Join



11.3) Right Outer Join



CONCLUSION:

In this experiment, I have implemented HIVE commands (HIVEQL) like creating database, tables, inserting data into the tables from csv files, partitioning and bucketing the tables, different joins on the tables and finally deleting the tables on Cloudera using VMware player. The queries were much like SQL queries, and it masked the mapReduce of the Hadoop framework which was evident while executing the queries in the terminal.