**ASSIGNMENT 3.3**

**Problem Statement:**

Hive Supports DDL, DML,UDF, UDTF and UDAF. Also supports UDF, UDTF and UDAF

**1.Hive Data Definition Language**

{Create, alter, Drop} {Table, View, Partition}

**2.Hive Data Manipulation Language:**

* Insert Overwrite
* It supports Sub-queries in from clause.
* Equi joins
* Multi-Table insert
* Sampling
* Lateral Views

User can embed custom map-reduce scripts.

**1. Hive Data Definition Language:**

**Create/Alter /Drop Databases:**

Hive will throw an error if financials already exists. You can suppress these warnings

with this variation:

CREATE DATABASE IF NOT EXISTS financials;

To view the existing databases:

SHOW DATABASES;

To view db starts with “h”:

SHOW DATABASES LIKE 'h.\*';

Hive will create a directory for each database. Tables in that database will be stored in

subdirectories of the database directory. The exception is tables in the default database,

which doesn’t have its own directory.

Can override this default location for the new directory as shown in this example:

CREATE DATABASE financials

LOCATION '/my/preferred/directory'

COMMENT 'Holds all financial tables';

To show the directory location for the database:

DESCRIBE DATABASE financials;

To set a database as your working database:

USE financials;

Property to print the current database as part of the prompt:

hive> set hive.cli.print.current.db=true;

hive (financials)> USE default;

To drop a database:

DROP DATABASE IF EXISTS financials;

By default, Hive won’t permit you to drop a database if it contains tables. Can either

drop the tables first or append the CASCADE keyword to the command, which will cause

the Hive to drop the tables in the database first:

DROP DATABASE IF EXISTS financials CASCADE;

ALTER DATABASE:

Can set key-value pairs in the DBPROPERTIES associated with a database using the

ALTER DATABASE command. No other metadata about the database can be changed,

including its name and directory location:

ALTER DATABASE financials SET DBPROPERTIES ('edited-by' = 'Joe Dba');

**Create/Alter/Drop Table:**

CREATE TABLE IF NOT EXISTS mydb.employees (

name STRING COMMENT 'Employee name',

salary FLOAT COMMENT 'Employee salary',

subordinates ARRAY<STRING> COMMENT 'Names of subordinates',

deductions MAP<STRING, FLOAT>

COMMENT 'Keys are deductions names, values are percentages',

address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>

COMMENT 'Home address')

COMMENT 'Description of the table'

TBLPROPERTIES ('creator'='me', 'created\_at'='2012-01-02 10:00:00', ...)

LOCATION '/user/hive/warehouse/mydb.db/employees';

**External Table**

* The EXTERNAL keyword lets you create a table and provide a LOCATION so that Hive does not use a default location for this table. This comes in handy if you already have data generated.
* When dropping an EXTERNAL table, data in the table is *not* deleted from the file system.
* An EXTERNAL table points to any HDFS location for its storage, rather than being stored in a folder specified by the configuration propertyhive.metastore.warehouse.dir
* An EXTERNAL table points to any HDFS location for its storage, rather than being stored in a folder specified by the configuration propertyhive.metastore.warehouse.dir.

CREATE EXTERNAL TABLE IF NOT EXISTS stocks (

exchange STRING,symbol STRING,

ymd STRING,price\_open FLOAT,price\_high FLOAT,price\_low FLOAT,price\_close FLOAT,

volume INT,price\_adj\_close FLOAT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

LOCATION '/data/stocks';

**Partitioned, Managed Tables**

Partitioned tables can be created using the PARTITIONED BY clause.

A table can have one or more partition columns and a separate data directory is created for each distinct value combination in the partition columns.

Further, tables or partitions can be bucketed using CLUSTERED BY columns, and data can be sorted within that bucket via SORT BY columns. This can improve performance on certain kinds of queries.

Partitioning tables changes how Hive structures the data storage. If we create this table

in the mydb database, there will still be an employees directory for the table:

hdfs://master\_server/user/hive/warehouse/mydb.db/employees

However, Hive will now create subdirectories reflecting the partitioning structure. For

example:

...

.../employees/country=CA/state=AB

.../employees/country=CA/state=BC

...

.../employees/country=US/state=AL

.../employees/country=US/state=AK

Check the partitions that exist:

SHOW PARTITIONS employees;

**Customizing Table Storage Formats**

In this example we have included TEXTFILE as format, and can overload it with default values with various delimiter while creating a table.

Some of the built in file formats are SEQUENCEFILE and RCFILE.

CREATE TABLE employees (

name STRING,

salary FLOAT,

subordinates ARRAY<STRING>,

deductions MAP<STRING, FLOAT>,

address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\001'

COLLECTION ITEMS TERMINATED BY '\002'

MAP KEYS TERMINATED BY '\003'

LINES TERMINATED BY '\n’

**STORED AS TEXTFILE;**

TEXTFILE implies that all fields are encoded using alphanumeric characters, including those from international character sets, although we observed that Hive uses nonprinting characters as “terminators” (delimiters), by default. When TEXTFILE is used, each line is considered a separate record.

The record parsing is handled by a serializer/deserializer or SerDe for short. For TEXTFILE

SerDe Hive uses is another Java class called org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe.

**Drop table**

DROP TABLE removes metadata and data for this table. The data is actually moved to the .Trash/Current directory if Trash is configured (and PURGE is not specified). The metadata is completely lost.

**DROP TABLE IF EXISTS employees;**

When dropping an EXTERNAL table, data in the table will notbe deleted from the file system.

**Alter Table**

Most table properties can be altered with ALTER TABLE statements, which change metadata about the table but not the data itself.

**Renaming a Table**

To rename the table log\_messages to logmsgs:

ALTER TABLE log\_messages RENAME TO logmsgs;

**2.Hive Data Manipulations:**

**Loading Data into Managed Tables:**

Hive does not do any transformation while loading data into tables. Load operations are currently pure copy/move operations that move datafiles into locations corresponding to Hive tables.

Hive has no row-level insert, update, and delete operations, the only way to put data into an table is to use one of the “bulk” load operations.

**LOAD DATA [LOCAL] INPATH'filepath'[OVERWRITE] INTO TABLE tablename[PARTITION (partcol1=val1, partcol2=val2 ...)]**

The keyword ‘Overwrite’ signifies that existing data in the table is deleted.

**Inserting Data into Tables from Queries:**

**INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]] select\_statement1 FROM from\_statement;**

INSERT OVERWRITE will overwrite any existing data in the table or partition unlessIF NOT EXISTSis provided for a partition.

Hive does not verify that the data you are loading matches the schema for the table.However, it will verify that the file format matches the table definition. For example,if the table was created with SEQUENCEFILE storage, the loaded files must be sequencefiles.

**Inserting Data into Tables from Queries:**

The INSERT statement lets you load data into a table from a query.

INSERT OVERWRITE TABLE employees

PARTITION (country = 'US', state = 'OR')

SELECT \* FROM staged\_employees se WHERE se.cnty = 'US' AND se.st = 'OR';

**Creating Tables and Loading Them in One Query**

Can also create a table and insert query results into it in one statement:

CREATE TABLE ca\_employees

AS SELECT name, salary, address

FROM employees

WHERE se.state = 'CA';

Fields for new table are taken from the SELECT clause. It extracts the subset of data from the table.

**Exporting Data**

can use INSERT … DIRECTORY …, as in this example:

INSERT OVERWRITE LOCAL DIRECTORY '/tmp/ca\_employees'

SELECT name, salary, address

FROM employees

WHERE se.state = 'CA';

**3. HiveQL Manipulations**

**SELECT … FROM Clauses**

SELECT scans the table specified by the FROM clause

**hive> SELECT name, salary FROM employees; John Doe 100000.0**

**WHERE Clauses**

WHERE clause is used to filter the result set by using *predicate operators* and *logical operators*. Functions can also be used to compute the condition.

**hive> Select \* from table\_nemWhere col=col\_name;**

**GROUP BY Clauses**

GROUP BY clause is frequently used with *aggregate functions*, to group the result set by columns and apply aggregate functions over each group. Functions can also be used to compute the grouping key.

**hive> SELECT category, count(1) FROM products GROUP BY category;**

**HAVING Clauses**

HAVINGclause lets you filter the groups produced by GROUP BY, by applying predicate operators to each groups.

**hive> SELECT category, count(1) AS cnt FROM products GROUP BY category HAVING cnt> 10;**

**LIMIT Clause**

The results of a typical query can return a large number of rows. The LIMIT clause puts an upper limit on the number of rows returned:

hive> SELECT upper(name), salary, deductions["Federal Taxes"],

> round(salary \* (1 - deductions["Federal Taxes"])) FROM employees

> LIMIT 2;

**Column Aliases**

It’s sometimes useful to give those anonymous columns a name, called a

column alias. Here is the previous query with column aliases for the third and fourth columns returned by the query, fed\_taxes and salary\_minus\_fed\_taxes, respectively:

hive> SELECT upper(name), salary, deductions["Federal Taxes"] as fed\_taxes,

> round(salary \* (1 - deductions["Federal Taxes"])) as salary\_minus\_fed\_taxes

> FROM employees LIMIT 2;

**Nested SELECT Statements**

The column alias feature is especially useful in nested select statements. Let’s use the

previous example as a nested query:

hive> FROM (

> SELECT upper(name), salary, deductions["Federal Taxes"] as fed\_taxes,

> round(salary \* (1 - deductions["Federal Taxes"])) as salary\_minus\_fed\_taxes

> FROM employees

> ) e

> SELECT e.name, e.salary\_minus\_fed\_taxes

> WHERE e.salary\_minus\_fed\_taxes > 70000;

The previous result set is aliased as e, from which we perform a second query to select

the name and the salary\_minus\_fed\_taxes, where the latter is greater than 70,000. (We’ll

cover WHERE clauses in “WHERE Clauses” on page 92 below.)

**CASE … WHEN … THEN Statements**

The CASE … WHEN … THEN clauses are like if statements for individual columns in query

results. For example:

hive> SELECT name, salary,

> CASE

> WHEN salary < 50000.0 THEN 'low'

SELECT … FROM Clauses | 91

www.it-ebooks.info

> WHEN salary >= 50000.0 AND salary < 70000.0 THEN 'middle'

> WHEN salary >= 70000.0 AND salary < 100000.0 THEN 'high'

> ELSE 'very high'

> END AS bracket FROM employees;

**LIKE and RLIKE:**

The string after the RLIKE keyword has the following interpretation. A period (.) matches

any character and a star (\*) means repeat the “thing to the left” (period, in the two cases

shown) zero to many times. The expression (x|y) means match *either* x *or* y.

For example, the following three queries select the employee names and addresses

where the street ends with Ave., the city begins with O, and the street contains Chicago:

**hive> SELECT name, address.street FROM employees WHERE address.street LIKE '%Ave.';**

John Doe 1 Michigan Ave.

Todd Jones 200 Chicago Ave.

**which finds all the employees whose street contains the word Chicago or Ontario:**

hive> SELECT name, address.street

> FROM employees WHERE address.street RLIKE '.\*(Chicago|Ontario).\*';

**Mary Smith 100 Ontario St.**

**Todd Jones 200 Chicago Ave.**

**JOIN Statements**

JOIN is a clause that is used for combining specific fields from two tables by using values common to each one. It is used to combine records from two or more tables in the database. It is more or less similar to SQL JOIN.

There are different types of joins given as follows:

* Inner
* Left outer
* Full outer

**Inner Join**

The inner join returns the rows when there is at least one match in both tables.

hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNTFROM CUSTOMERS c JOIN ORDERS o ON (c.ID = o.CUSTOMER\_ID);

**Left Outer Join**

The HiveQLLEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right table, the JOIN still returns a row in the result, but with NULL in each column from the right table.

**hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATEFROM CUSTOMERS c LEFT OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER\_ID);**

**Right Outer Join**

The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.

**hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATEFROM CUSTOMERS c RIGHT OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER\_ID);**

**Full Outer Join**

The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

**hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATEFROM CUSTOMERS c FULL OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER\_ID);**

**LEFT SEMI-JOIN**

A left semi-join returns records from the lefthand table if records are found in the righthand

table that satisfy the ON predicates. It’s a special, optimized case of the more general

inner join.

**hive> SELECT s.ymd, s.symbol, s.price\_close**

**> FROM stocks s LEFT SEMI JOIN dividends d ON s.ymd = d.ymd AND s.symbol = d.symbol;**

**Cartesian Product JOINs**

A Cartesian product is a join where all the tuples in the left side of the join are paired

with all the tuples of the right table. If the left table has 5 rows and the right table has

6 rows, 30 rows of output will be produced:

SELECTS \* FROM stocks JOIN dividends;

**ORDER BY**

ORDER BY clause in a SELECT statement. The ORDER BY clause is used to retrieve the details based on one column and sort the result set by ascending or descending order.

**hive> SELECT Id, Name, DeptFROM employee ORDER BY DEPT;**

**CLUSTER BY,DISTRIBUTE BY,SORT BYspecify the sort order and algorithm**

* ClusterBy and DistributeBy are used mainly with theTransform/Map-Reduce Scripts. But, it is sometimes useful in SELECT statements if there is a need to partition and sort the output of a query for subsequent queries.
* Cluster By is a shortcut for both DistributeBy and Sort By.
* Hive uses the columns in DistributeBy to distribute the rows among reducers. All rows with the same Distribute Bycolumns will go to the same reducer.

**hive> SELECT col1, col2 FROM t1 DISTRIBUTE BY col1 SORT BY col1 ASC, col2 DESC;**

The ORDER BY clause is familiar from other SQL dialects. It performs a total ordering of

the query result set. This means that all the data is passed through a single reducer,

which may take an unacceptably long time to execute for larger data sets.

Hive adds an alternative, SORT BY, that orders the data only within each reducer, thereby

performing a local ordering, where each reducer’s output will be sorted. Better performance

is traded for total ordering.

Here is an example using ORDER BY:

SELECT s.ymd, s.symbol, s.price\_close

FROM stocks s

ORDER BY s.ymd ASC, s.symbol DESC;

Here is the same example using SORT BY instead:

SELECT s.ymd, s.symbol, s.price\_close

FROM stocks s

SORT BY s.ymd ASC, s.symbol DESC;

**DISTRIBUTE BY with SORT BY**

DISTRIBUTE BY controls how map output is divided among reducers. All data that flows

through a MapReduce job is organized into key-value pairs. Hive must use this feature

internally when it converts your queries to MapReduce jobs.

**Casting**

**This is Type Conversion Function**used to convert from one data type to another. The only type conversion function is CAST.

The CAST function converts the expr into the specified type.

**hive> SELECT \* From table\_name**

**Where CAST( COlas <type> );**

**Queries that Sample Data**

For very large data sets, sometimes you want to work with a representative sample of

a query result, not the whole thing. Hive supports this goal with queries that sample

tables organized into buckets

We can sample using the rand() function, which returns a random number. In the first

two queries, two distinct numbers are returned for each query. In the third query, no

results are returned:

**hive> SELECT \* from numbers TABLESAMPLE(BUCKET 3 OUT OF 10 ON rand()) s;**

**2**

**4**

**hive> SELECT \* from numbers TABLESAMPLE(BUCKET 3 OUT OF 10 ON rand()) s;**

**7**

**10**

**UNION ALL**

UNION is used to combine the result from multiple SELECT statements into a single result set.

UNION ALL combines two or more tables. Each subquery of the union query must produce

the same number of columns, and for each column, its type must match all the

column types in the same position.

**SELECT name, id, category FROM source\_table\_1**

**UNION ALL**

**SELECT name, id, "Category159" FROM source\_table\_2**

**Views:**

View is a purely logical object with no associated storage. No support for materialized views is currently available in Hive. When a query references a view, the view's definition is evaluated in order to produce a set of rows for further processing by the query.

Views are read-only and may not be used as the target of LOAD/INSERT/ALTER.

It is common for Hive queries to have many levels of nesting. In the following example,

the nested portion of the query is turned into a view:

CREATE VIEW shorter\_join AS

SELECT \* FROM people JOIN cart

ON (cart.people\_id=people.id) WHERE firstname='john';

DROP VIEW IF EXISTS shipments;

Finally, views are read-only. You can only alter the metadata TBLPROPERTIES for a view:

ALTER VIEW shipments SET TBLPROPERTIES ('created\_at' = 'some\_timestam’);

You cannot use a view as a target of an INSERT or LOAD command.

View is used whenever the query becomes longer and complex, that hides the complexity by dividing the query into sub query

Data may be restricted based on conditions. Where clause is used to filter the data on condition and query selected data.