UNIT-5

HIVE

UNIT-V

HIVE- A data warehouse in Hadoop

Syllabus

What is Hive? The Hive shell, Hive Services, The Meatastore, Comparison with traditional Databases, HiveQL, Data types, Operators and Functions, Tables, Managed tables and External Tables, Partitions and Buckets, Importing data, Altering Tables, Dropping Tables, Querying Data, Sorting and Aggregating, Joins, Subqueries, View, What is UDF? Types of Hive UDFs.

HIVE

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize big data, and makes querying and analyzing easy.

Initially hive was developed by Jeff Hammerbacher at f acebook, later the apache software foundation took it up and developed it further as an open source under the apache hive.

Hive is not

- A relational database
- A design for online transaction processing(OLTP)
- A language for real time queries and row level updates

Features of hive

- It stores schema in a database and processed data into HDFS.
- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

Hive Shell

The shell is th e primary way that we will interact with Hive, by issuing commands in HiveQL . HiveQL is Hive's query language, a dialect of SQL. It is heavily influenced by MySQL, so if you are familiar with MySQL you should feel at home using Hive.

When starting Hive for the first time, we can check that it is working by listing its tables: there should be none. The command must be terminated with a semicolon to tell Hive to execute it:

Ex: hive> SHOW TABLES;

OK

Time taken: 10.425 seconds

Features of shell

It is po ssible to run the hive shell in non -interactive mode. The –f option runs the commands in the specified file.

ex: script.q

% hive -f script.q

For short scripts, you can use the -e option to specify the commands inline, in which case the final semicolon is not required:

% hive -e 'SELECT * FROM dummy'

Hive history file=/ tmp/ tom/ hive_job_log_tom_201005042112_1906486281.txt

OK

Time taken: 4.734 seconds

In both interactive and non -interactive mode, Hive will print information to standard error —such as the time taken to run a query, to surpress these messages using the –s option it shows only the result of the query.

% hive -S -e 'SELECT * FROM dummy'

X

Other useful Hive shell features include the ability to run commands on the host operating system by using a! Prefix to the command and the ability to access Hadoop filesystems using the dfs command .

Hive Services

The Hive shell is only one of several services that you can run using the hive command.

You can specify the service to run using the -- service option. Type hive –service help to get a list of available service names; the most useful are described below.

1. cli

The command line interface to Hive (the shell). This is the default service.

2. hiveserver

Runs Hive as a server exposing a Thri ft service, enabling access from a range of clients written in different languages. Applications using the Thrift, JDBC, and ODBC connectors need to run a Hive server to communicate with Hive. Set the HIVE_PORT environment variable to specify the port the server will listen on (defaults to 10,000).

3. hwi

The Hive Web Interface

4. Jar

The Hive equivalent to hadoop jar, a convenient way to run Java applications that includes both Hadoop and Hive classes on the classpath.

5. Metastore

Using this service, it is possible to run the metastore as a standalone (remote) process.

- **Hive Clients**

Hive as a server (hive -- service hiveserver), then there are a number of different mechanisms for connecting to it from applications.

The relationship between Hive c lients and Hive services is illustrated in Figure

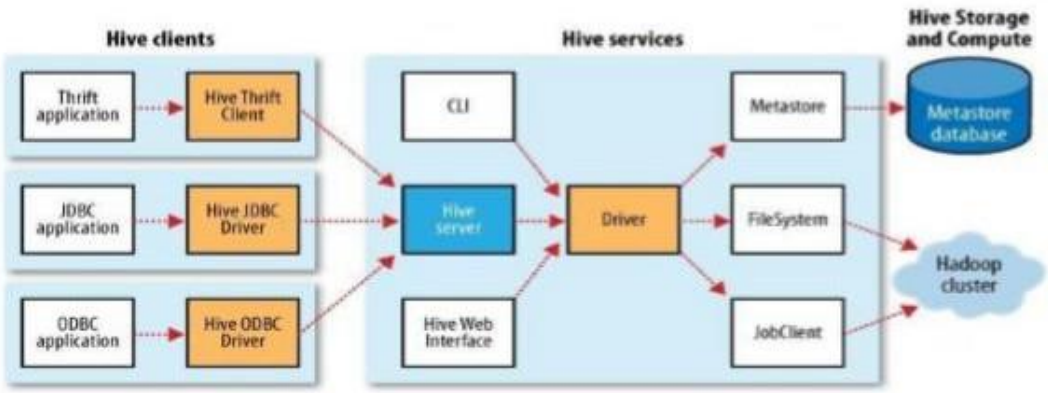

Fig: Hive architecture

- **Thrift Client**

The Hive Thrift Client makes it easy to run Hive commands from a wide range of programming languages. Thrift bindings for Hive are available for C++, Java, PHP, Python, and Ruby.

- **JDBC Driver**

Hive provides a Type 4 (pure Java) JDBC driver, defined in the class org.apache.hadoop.hive.jdbc.HiveDriver. When configured with a JDBC URI of the form

jdbc:hive:/ / host :port / dbname , a Java application will c onnect to a Hive server running in a separate process at the given host and port.

- ODBC Driver

The Hive ODBC Driver allows applications that support the ODBC protocol to connect to Hive.

The metastore

- The metastore is the central repository of Hive me tadata.
- The metastore is divided into two pieces: a service and the backing store for the data. There are 3 different metastore configurations

1. Embedded metastore

- By default, the metastore service runs in the same JVM as the Hive service and conta ins an embedded Derby database instance backed by the local disk. This is called the embedded metastore configuration.
- a simple way to get started with Hive.
- only one embedded Derby database can access the database files on disk at any one time, which mean s you can only have one Hive session.
- if we open another session it attempts to open a connection to the metastore i.e., trying to start a second session gives the error.

Error: Failed to start database 'metastore_db'

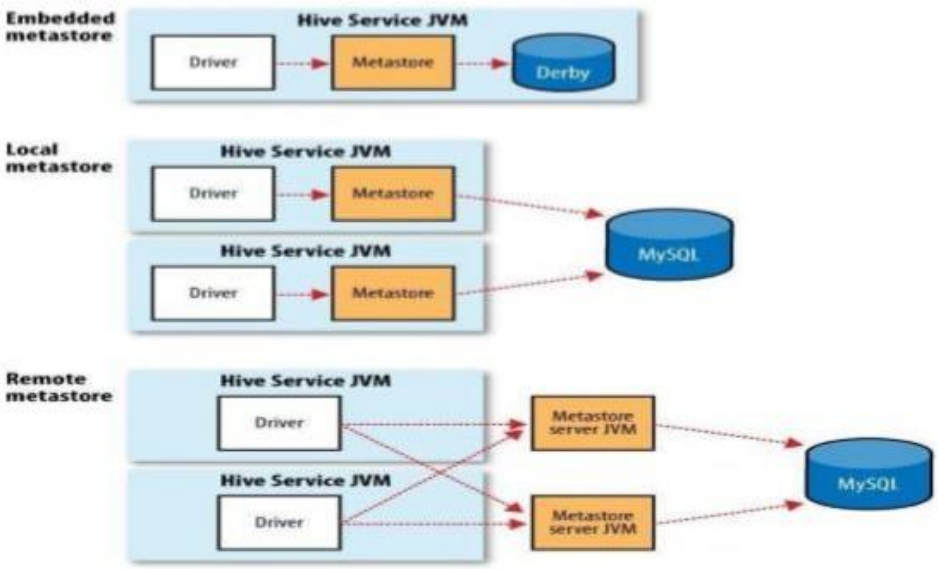

Figure : Metastore configurations

2. local metastore

- To support multiple sessions (and therefore multiple users) is to use a standalone database.
- This configuration is referred to as a local metastore , since the metastore service still runs in the s ame process as the Hive service, but connects to a database running in a separate process, either on the same machine or on a remote machine.

3. Remote metastore

- Where one or more metastore servers run in separate processes to the Hive service.
- Brings better manageability and security, since the database tier can be completely firewalled off, and the clients no longer need the database credentials.

Table 12-1. Important metastore configuration properties

Property name	Type	Default value	Description
hive.metastore.warehouse.dir	URI	/user/hive/warehouse	The directory relative to fs.default.name where managed tables are stored.
hive.metastore.local	boolean	true	Whether to use an embedded metastore server (true), or connect to a remote instance (false). If false, then hive.metastore.uris must be set.
hive.metastore.uris	comma-separated URIs	Not set	The URIs specifying the remote metastore servers to connect to. Clients connect in a round-robin fashion if there are multiple remote servers.
javax.jdo.option.ConnectionURL	URI	jdbc:derby:;databaseName=metastore_db;create=true	The JDBC URL of the metastore database.
javax.jdo.option.ConnectionDriverName	String	org.apache.derby.jdbc.EmbeddedDriver	The JDBC driver classname.
javax.jdo.option.ConnectionUserName	String	APP	The JDBC user name.
javax.jdo.option.ConnectionPassword	String	mine	The JDBC password.

Comparison with traditional databases

Schema on Read versus Schema on W rite\

- In a traditional database, a table's schema is enforced at data load time.
- Hive does not verify the data when it is loaded but it is verified when the query is issued.
- Traditional db takes longer time to load data.
- Schema on read makes for a very fast

Initial load Query time performance

- Schema on write makes query time performance faster.
- Schema on read makes longer time for query execution.

Transactions

Hive doesnot support transactions .

Indexes

Release 0.7.0 introduced indexes, which can speed up queries.

Locking

Release 0.7.0 introduces table and partitional level locking in hive. Locks are managed transparently by zookeeper.

Hive QL

- HiveQL is Hive's SQL dialect.
- It does not provide th e full features of SQL -92 language constructs. The main difference between HiveQL and
- SQL are Table : A high -level comparison of

SQL and HiveQL

Feature	SQL	HiveQL
Updates	UPDATE, INSERT, DELETE	INSERT OVERWRITE TABLE (populates whole table or partition)
Transactions	Supported	Not supported
Indexes	Supported	Not supported
Latency	Sub-second	Minutes
Data types	Integral, floating point, fixed point, text and binary strings, temporal	Integral, floating point, boolean, string, array, map, struct
Functions	Hundreds of built-in functions	Dozens of built-in functions
Multitable inserts	Not supported	Supported
Create table as select	Not valid SQL-92, but found in some databases	Supported
Select	SQL-92	Single table or view in the FROM clause. SORT BY for partial ordering. LIMIT to limit number of rows re-turned.

Feature	SQL	HiveQL
Joins	SQL-92 or variants (join tables in the FROM clause, join condition in the WHERE clause)	Inner joins, outer joins, semi joins, map joins. SQL-92 syntax, with hinting.
Subqueries	In any clause. Correlated or noncorrelated.	Only in the FROM clause. Correlated subqueries not supported
Views	Updatable. Materialized or nonmaterialized.	Read-only. Materialized views not supported
Extension points	User-defined functions. Stored procedures.	User-defined functions. MapReduce scripts.

Data Types

Hive supports both primitive and complex data types.

Primitives include

1) numeric

2) boolean

3) string,

4) timestamp

complex data types include

1) arrays

2) maps

3) structures

Category	Type	Description	Literal examples
Primitive	TINYINT	1-byte (8-bit) signed integer, from -128 to 127	1
	SMALLINT	2-byte (16-bit) signed integer, from -32,768 to 32,767	1
	INT	4-byte (32-bit) signed integer, from -2,147,483,648 to 2,147,483,647	1
	BIGINT	8-byte (64-bit) signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	1
	FLOAT	4-byte (32-bit) single-precision floating-point number	1.0
	DOUBLE	8-byte (64-bit) double-precision floating-point number	1.0
	BOOLEAN	true/false value	TRUE
	STRING	Character string	'a', "a"
	BINARY	Byte array	Not supported
	TIMESTAMP	Timestamp with nanosecond precision	1325502245000, '2012-01-02 03:04:05.123456789'
Complex	ARRAY	An ordered collection of fields. The fields must all be of the same type.	array(1, 2)[a]
	MAP	An unordered collection of key-value pairs. Keys must be primitives; values may be any type. For a particular map, the keys must be the same type, and the values must be the same type.	map('a', 1, 'b', 2)
	STRUCT	A collection of named fields. The fields may be of different types.	struct('a', 1, 1.0)[b]

Operators and Functions

Operators

1) arithmetic

2) relational

3) logical

Categories of Functions

1) mathematical

2) statistical

3) string

4) date

5) conditional

6) aggregate

7) Functions working with XML and JSON

Tables

- A Hive table is logically made up of the data being stored and the associated metadata describing the layout of the data in the table.
- The data typically resides in HDFS, although it may reside in any Hadoop filesystem, inclu ding the local filesystem or S3.
- Hive stores the metadata in a relational database —and not in HDFS, Managed Tables and External Tables When you create a table in Hiv e, by default Hive will manage the data, which means that Hive moves the data into its warehouse directory

For example: CREATE TABLE managed_table (dummy STRING);

LOAD DATA INPATH '/ user/ tom/ data.txt' INTO table managed_table; will move the file hdfs:/ / user/ tom/ data.txt into Hive's warehouse directory for the managed_table table, which is hdfs:/ / user/ hive/ warehouse/ managed_table.

If the table is later dropped, using

DROP TABLE managed_table

then the table, including its metadata and its data , is deleted.

An external table behaves differently. You control the creation and deletion of the data.

The location of the external data is specified at table creation time:

CREATE EXTERNAL TABLE external_table (dummy STRING)

LOCATION '/ user/ tom/ externa l_table';

LOAD DATA INPATH '/ user/ tom/ data.txt' INTO TABLE external_table;

Partitions and Buckets

Hive organizes tables into partitions , a way of dividing a table into coarse - grained parts based on the value of a partition column , such as date. Tables or partitions may further be subdivided into buckets , to give extra structure to the data that may be used for more efficient queries.

Partitions

- Hive organizes tables in to partitions.
- A way of dividing a table in to related parts based on the value of a partition column ex: date, city, dept.
- Faster to do queries on slices of the data.
- Tables or partitions are sub -divided into buckets , to provide extra structure to the data that may be used for more efficient querying.
- Bucketing works based on the value of hash function of some column of a table.
- A table named Tab1 contains employee data such as id, name, dept, and yoj , need to retrieve the details of all employees who joined in 2012.
- A query searches the whole table for the required information.
- if you partition the employee data with the year and store it in a separate file, it reduces the query processing time.

The following file contains employee data table. / tab1/ employeedata/ file1 id, name, dept, yoj.

1, gopal, TP,2012
2, kiran,HR,2012
3, kaleel,SC,2013
4,Prasanth,SC,20

The above data is partitioned into two files using year.

/ tab1/ employeedata/ 2012/ file2 / tab1/ employeedata/ 2013/ file3

1, gopal, TP, 2012

2, kiran, HR, 2012

3, kaleel,SC, 2013

4, Prasanth, SC, 2013

Adding a Partition : -

We can add partitions to a table by altering the

table hive> ALTER TABLE employee

> ADD PARTITION (year='2013')

> location '/ 2012/ part2012';

Renaming a Partition

hive> ALTER TABLE employee PARTITION (year='1203')

> RENAME TO PARTITION (Yoj='1203');

Show Partition

hive> show partitions employee;

Dropping a Partition

hive> ALTER TABLE employee DROP [IF EXISTS]

> PARTITION (year='1203');

Buckets

- It is a mechanism to query and examine random samples of data.
- Break data into a set of buckets based on a hash func tion of a ―bucket column.
- Capability to execute queries on a sub -set of random data.
- Doesn't automatically enforce bucketing.

- User is required to specify the number of buckets by setting # of reducer Create and use table with Buckets.

hive>create table po st-count(user String, count Int) >clustered by (user) into 5 buckets;

Use the clustered by clause to specify columns to bucket on and the number of buckets.

Storage Formats

- There are two dimensions that govern table storage in Hive: the row format and th e file format . The row format dictates how rows, and the fields in a particular row, are stored.
- The file format dictates the container format for fields in a row. The simplest format is a plain text file, but there are row -oriented and column -oriented bin ary formats available, too.
- The default storage format: Delimited text.
- When you create a table with no ROW FORMAT or STORED AS clauses, the default format is delimited text, with a row per line.
- The default row delimiter is not a tab character, but the Control -A character from the set of ASCII control codes.
- The choice of Control -A, sometimes written as ^A in documentation, came about since it is less likely to be a part of the field text than a tab character.

  The default collection item delimiter is a Control -B character, used to delimit items in an ARRAY or STRUCT, or key -value pairs in a MAP. The default map key delimiter is a Control -C character, used to delimit the key and value in a MAP. Rows in a table are delimited by a newline character.
- Importing Data

  INSERT OVERWRITE TABLE
- a table with data from another Hive table using an INSERT statement i.e

  hive> INSERT OVERWRITE TABLE

  target >SELECT col1, col2

  FROM

  source;

  Another way

  Hive> FROM source

>INSERT OVERWRITE TABLE tar get

>SELECT col1, col2;

- Multitable insert.
- it's possible to have multiple INSERT clauses in the same query.
- multitable insert is more efficient than multiple INSERT statements.
- the source table need only be scanned once to produce the multiple, disjoint outputs hive> FROM records2.

>INSERT OVERWRITE TABLE stations_by_year

>SELECT year, COUNT(DISTINCT station)

>GROUP BY year

>INSERT OVERWRITE TABLE records_by_year

>SELECT year, COUNT(1)

>GROUP BY year

>INSERT OVERWRITE TABLE good_records_by_year

>SELECT year, COUNT(1)

>WHERE temperature != 9999

>AND (quality = 0 OR quality = 1 OR quality = 4 OR quality = 5 OR quality = 9)

>GROUP BY year;

There is a single source table (records2), but three tables to hold the results from three different queries over the source.

CREATE TABLE...AS SELECT

- To store the output of a Hive query in a new table.
- The new table's column definitions are derived from the columns retrieved by the SELECT clause.

Example

CREATE TABLE target

AS

SELECT col1, co l2

FROM source;

Alter Table

Alter the attributes of a table such as changing its table name, changing column names, adding columns, and deleting or replacing columns.

Syntax

ALTER TABLE name RENAME TO new_name

ALTER TABLE name ADD COLUMNS (col_spec[, c ol_spec ...])

ALTER TABLE name DROP [COLUMN] column_name

ALTER TABLE name CHANGE column_name new_name new_type ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])

Examples

hive> ALTER TABLE employee RENAME TO emp;

hive>ALTER TABLE employee ADD C OLUMNS (dept

string); hive>ALTER TABLE employee DROP dept;

hive> ALTER TABLE employee CHANGE name ename String; hive> ALTER TABLE employee CHANGE salary salary Double;

hive> ALTER TABLE employee REPLACE COLUMNS (eid INT empid Int,

ename STRING name Stri ng);

hive> DROP TABLE IF EXISTS employee;

Select Statements

SELECT statement is used to retrieve the data from a table. WHERE clause works similar to a condition. It filters the data using the condition and gives you a finite result.

Syntax:

SELECT [ALL | DISTINCT] select_expr, select_expr, ...

FROM table_reference

[WHERE where_condition]

[GROUP BY col_list]

[HAVING having_condition]

[CLUSTER BY col_list |

[DISTRIBUTE BY col_list]

[SORT BY col_list]]

[LIMIT number];

Examples:

hive> SELECT * FROM employee WHERE eid=1205;

hiv e> SELECT * FROM employee WHERE salary>=40000;

hive> SELECT Id, Name, Dept FROM employee ORDER BY DEPT;

hive> SELECT Dept,count(*) FROM employee GROUP BY DEPT;

hive> FROM records2 SELECT year, temperature

DISTRIBUTE BY year

SORT BY year ASC, temperature

DESC;

hive> SELECT * FROM employee LIMIT 4;

Querying Data Sorting and Aggregating

Sorting data in Hive can be achieved by use of a standard ORDER BY clause, but there is a catch. ORDER BY produces a result that is totally sorted, as expected, but to do so it sets the number of reducers to one, making it very inefficient for large datasets.

In some cases, you want to control which reducer a particular row goes to, typically so you can perform some subsequent ag gregation. This is what Hive's DISTRIBUTE BY clause does. Here's an example to sort the weather dataset by year and temperature.

SORT BY produces a sorted file per reducer.

hive> FROM re cords2

> SELECT year, temperature

> DISTRIBUTE BY year

> SORT BY year AS C, temperature

DESC; 1949 111 1949 78 1950 22

1950 0 1950 -11

MapReduce Scripts

Using an approach like Hadoop Streaming, the TRANSFORM, MAP, and REDUCE clauses make it possible to invoke an external script or program from Hive.

Example: Python script t o filter out poor quality weather records

```
#!/ usr/ bin/ env python
import re
import sys
for line in sys.stdin;
(year, temp, q) = line.strip().split()
if (temp != "9999" and re.match("[01459]", q)):
 print "%s \ t%s" % (year, temp)
```

We can use the script as follows:

```
hive> ADD FILE / path/ to/ is_good_quality.py;
hive> FROM records2
> SELECT TRANSFORM(year, temperature, quality)
> USING 'is_good_quality.py'
> AS year,
temperature; 1949
111 1949 78 1950 0
 1950 22
 1950 -11
```

Before running the query, we need to register the script with Hive. This is so Hive knows to ship the file to the Hadoop cluster.

The query itself streams the year, temperature, and quality fields as a tab - separated line to the is_good_quality.py script, and parses the tab -separated output i nto year and temperature fields to form the output of the query.

This example has no reducers. If we use a nested form for the query, we can specify a map and a reduce function. This time we use the MAP and REDUCE keywords, but SELECT TRANSFORM in both ca ses would have the same result. The source for the max_temperature_reduce.py script is shown in Example.

```
FROM
(FROMreco s
2
MAP year, temperature,
quality USING
```

'is_good_quality.py'

AS year, temperature)

map_output REDUCE year,

temperature USING

'max_temper ature_reduce.py'

AS year, temperature;

Views

- A view is a sort of —virtual table□ that is defined by a SELECT statement.
- Views can be used to present data to users in a different way to the way it is actually stored on disk.
- Views may also be used to restrict users access to particular subsets of tables that they are authorized to see.
- First create table and then insert data into it.

  hive> create table posts(id int,name string,sal double)

  > row format delimited

  > fields terminated by ','

  □ stored as textfile;

  □ Create View

hive> create view posts_nam e as

> select name from posts;

hive> create view first_id as

select * from posts where id=1;

hive> create view max_sal as

select name ,max(sal) from posts;

Show views

hive> show tables;

□ Alterin g views

hive> alter view first_id rename to 1stid;

□ Drop a view hive> drop view

1stid

Joins

- JOIN is a clause that is used for combining specific fields from two tables by using values common to each one.
- used to combine records from two or more tables in the database.
- similar to SQL JOINS.
- There are different types of joins given as follows:
  - JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN
- Can join multiple tables
- Default join Is Inner join
  - Rows are joined where the keys match.
  - Rows that do not have matches are not included in the result.

The simplest kind of join is the inner join, where each match in the input tables results in a row in the output table it is being joined to (things): hive>

SELECT * FROM sales;

joe 2

Hank 4

Ali 0

Eve 3

Hank 2

hive> SELECT * FROM things;

2 Tie

4 Coat

3 Hat

1 Scarf

hive> SELECT sales.*, things.*

> FROM sales JOIN things ON (sales.id = things.id);

Joe 2 2 Tie

Hank 2 2 Tie

Eve 3 3 Hat

Hank 4 4 Coat

- The table in the FROM clause (sales) is joined with the table in the JOIN clause (things), using the predicate in the ON clause.
- Hive only supports equijoins, which means that only equality can be used in the join predicate, which here matches on the id column in both tables.
- the row for Ali did not appear in the output, since the ID of the item she purchased was not present in the things table

Left Outer Join

- Outer joins allow you to find non matches in the tables being joined.
- If we change the join type to LEFT OUTER JOIN, th en the query will return a row for every row in the left table (sales), even if there is no corresponding row in the table it is being joined to (things):

hive> SELECT * FROM sales;

Joe 2

Hank 4

Ali 0

Eve 3

Hank 2

hive> SELECT * FROM things;

2 Tie

4 Coat

3 Hat

1 Scarf

The row for Ali is now returned, and the columns from the things table are NULL, since there is no match.

i.e Row from the first table are included whether they have a match or not. Columns from the unmatched(second) table are set to null.

Right Outer Join

- Opposite of Left Outer Join, Rows from the second table are included no matter what. Columns from the unmatched (first) table are set to null.
- all items from the things table are included, even those that weren't purchased by anyone (a scarf):

hive> SELECT * FROM sales;

Joe 2

Hank 4

Ali 0

Eve 3

Hank 2

hive> SELECT * FROM things;

2 Tie

4 Coat

3 Hat

1 Scarf

We can perform left outer join on the two tables as follows:

hive> SELECT sales.*, things.*

> FROM sales RIGHT OUTER JOIN things ON (sales.id =

things.id); NULL NULL 1 Scarf

Joe 2 2 Tie

Hank 2 2 Tie

Eve 3 3 Hat

Hak 4 4 Coat

Full Outer Join

- Rows from both sides are includes. For unmatched rows the columns from the other table are set to null.
- In full ou ter join, the output has a row for each row from both tables in the join:

hive> SELECT * FROM sales;

Joe 2

Hank 4

Ali 0

Eve 3

We can perform left outer join on the two tables as follows:

hive> SELECT sales.*, things.*

Ali

Joe

Han Eve

Hank

> FROM sales LEFT OUTER JOIN

0 NULL NULL

2 2 Tie

2 2 Tie

3 3 Hat

4 4 Coat

Subqueries

- A subquery is a SELECT statement that is embedded in another SQL statement.
- Hive has limited support for subqueries

The query finds the mean maximum temperature for every year and weather

station:

SELECT station, year, AVG(max_temperature)

FROM (SELECT station, year, MAX(temperature) AS

max_temperature FROM records2 WHERE temperature != 9999

AND (quality = 0 OR quality = 1 OR qua lity = 4 OR quality = 5 OR quality = 9)

```
GROUP BY station, year
) mt
GROUP BY station, year;
```

- The subquery is used to find the maximum temperature for each station/ date combination.
- the outer query uses the AVG aggregate function to find the average of the maximum temperature readings for each station/ date combination.
- The outer query accesses the results of the subquery like it does a table, which is why the subquery must be given an alias (mt).
- The columns of the subquery have to be given unique name s so that the outer query can refer to them.

User -Defined functions

- Write the query that can't be expressed easily using built -in functions.
- Write a User -Defined Function(UDF) .
- Easy to plug in own processing code and invoke it from a Hive Query.

There are 3 types of UDF in Hive.

1) Regular UDF s

Operates on a single row and produces a single row as its output.

Ex: Mathematical and String functions.

2) UDAF (User -defined aggregate functions)

□ works on multiple input rows and creates a single output row.

□ Aggregate functions include such functions as COUNT and MAX.

3) UDTF s (user -defined table -generating functions)

□ operates on a single row and produces multiple rows —a table —as output

Assignment -Cum -Tutorial Questions

SECTION –A

Objective Questions

1. Which of the following command sets the value of a particular configuration variable.

A) Set-v B) set <key>=<value> C) set D) reset

2. Which of the following operator executes a shell command from the Hive shell?

A) | B) ! C^ D) +

3. Which of the following will remove the resource(s) from the distributed cache?

A) Delete FILE[S] <fi lepath>*

B) Delete JAR[S]<filepath>*

C) Delete ARCHIVE[S]<filepath>*

D) All

4. Is a shell utility which can be used to run Hive queries in either interactive or batch mode.

A) $HIVE/ bin/ hive

B) $HIVE_HOME/ hive

C) $HIVE_HOME/ bin/ hive

D) All

5. Which of the followin g is a command line option?

A) –d,-define <key=value>

B) –e,-define<key=value>

C) –f,-define<key=value>

D) None

6. Hive uses_____for logging

A) logj4 B) log41 C) log4i D) log4j

7. Hive Server2 introduced in HIVE 0.11 has new CLI called.

A) BeeLine B) SQLLine C)HIVELine D) CLILin

8. Hcatalog is installed with HIVE, starting with HIVE relase

A) 0.10.0 B) 0.9.0 C)0.11.0 D)0.1.20

9. supports a new command shell Beeline that works with HIVE Server2.

A) HiveServer2 B) Hiv eServer3 C) HiveServer4 D) None

10. In mode HiveServer2 only accepts valid Thrift calls.

A) Remote B) HTTP C) Embedded D) Interactive

11. Hive specific commands can be run from Beeline, When the Hive driver is used

A) ODBC B) JDBC C) ODBC-JDBC D) ALL

12. The allow users to read or write Avro data s Hive Table

A) AvroSerde B) HiveSerde C) SQLSerde D) None

13. Starting in Hive_____the Avro schema can be inferred from the hive table schema.

A) 0.14 B) 0.1 2 C) 0.13 D) 0.11

14. Which of the following data type is supported by HIVE.

A) map B) record C) string D) enum

15. which of the following data type is converted to Array prior to Hive 0.12.0

A) map B) long C) float D) bytes

16. Avr o-backed tables can simply be created by using in a DDL statement.

A) "STORED AS AVERO" C. –STORED AS AVROHIVE

B) –STORED AS HIVE D. –STORED AS SERED

17. Types that may be null must be defined as a of that type and NULL within AVRO.

A) Union B) intersection C) Set D) All

18. use _____ and embed the schema in the create statement.

A) schema.literal B) schema.lit C) row.literal D) All

19. Serialization of string columns uses a to form unique column value.

A) Footer B) STRIPES C) Dictionary D) Index

20. Hive uses _____ -Style escaping within the strings.

A) C B) JAVA C) python D) Scala

SECTION –B

SUBJECTIVE QUESTIONS

1. What is hive? List the features of hive?

2. List out hive Services

3. What is metasto re? What are different types of metastores?

4. What are megastore configuration properties?

5. Comapre the SQL and HIVEQL

6. List out Hive Data Types?
7. Explain about partitions and buckets?
8. Outline about Querying Data?
9. What are user -defined functions?
10. Explain joins?
11. Explain about HIVEQL in Hadoop System
12. Illustrate the HIVE Shell?
13. Describe about the tables in HIVE.
14. Explain about HIVE architecture?
15. Compare HIVE with traditional database?
16. Elaborate on HIVE QL data manipulation and queries in details
17. Discuss about the relationship between HIVE clients and HIVE Services with a neat diagram?
18. Explain in detail about Map side and Reduce Side joins.