**DESCRIPTION OF HIVE**

## What is 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 Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

**HIVE INSTALLATION**

**Installing HIVE:**

Browse to the link: http://apache.claz.org/hive/stable/

Click the apache-hive-0.13.0-bin.tar.gz

Save and Extract it

Commands-

user@ubuntu:~$ cd /usr/lib/

user@ubuntu:~$ sudo mkdir hive

user@ubuntu:~$ cd Downloads

user@ubuntu:~$ sudo mv apache-hive-0.13.0-bin /usr/lib/hive

**Setting Hive environment variable:**

Commands-

user@ubuntu:~$ cd

user@ubuntu:~$ sudo gedit ~/.bashrc

Copy and paste the following lines at end of the file

# Set HIVE\_HOME

export HIVE\_HOME="/usr/lib/hive/apache-hive-0.13.0-bin"

export PATH=$PATH:$HIVE\_HOME/bin

**Setting HADOOP\_PATH in HIVE config.sh**

Commands-

user@ubuntu:~$ cd /usr/lib/hive/apache-hive-0.13.0-bin/bin

user@ubuntu:~$ sudo gedit hive-config.sh

Go to the line where the following statements are written

# Allow alternate conf dir location.

HIVE\_CONF\_DIR="${HIVE\_CONF\_DIR:-$HIVE\_HOME/conf"

export HIVE\_CONF\_DIR=$HIVE\_CONF\_DIR

export HIVE\_AUX\_JARS\_PATH=$HIVE\_AUX\_JARS\_PATH

Below this write the following

export HADOOP\_HOME=/usr/local/hadoop (write the path where hadoop file is there)

**HIVE launch**

user@ubuntu:~$ hive

**HIVE STATEMENTS**

**Create Database Statement**

CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>

**Drop Database Statement**

DROP DATABASE |SCHEMA [IF EXISTS] database\_name [RESTRICT|CASCADE];

**Create Table Statement**

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db\_name.] table\_name

[(col\_name data\_type [COMMENT col\_comment], ...)]

[COMMENT table\_comment]

[ROW FORMAT row\_format]

[STORED AS file\_format]

**Alter Table Statement**

ALTER TABLE name RENAME TO new\_name

ALTER TABLE name ADD COLUMNS (col\_spec[, col\_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 ...])

**Drop Table Statement**

DROP TABLE [IF EXISTS] table\_name;

**Creating a View**

CREATE VIEW [IF NOT EXISTS] view\_name [(column\_name [COMMENT column\_comment], ...) ] [COMMENT table\_comment] AS SELECT ...

**Dropping a View**

DROP VIEW view\_name

**Creating an Index**

CREATE INDEX index\_name

ON TABLE base\_table\_name (col\_name, ...)

AS 'index.handler.class.name'

[WITH DEFERRED REBUILD]

[IDXPROPERTIES (property\_name=property\_value, ...)]

[IN TABLE index\_table\_name]

[PARTITIONED BY (col\_name, ...)]

[

[ ROW FORMAT ...] STORED AS ...

| STORED BY ...

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (...)]

**Dropping an Index**

DROP INDEX <index\_name> ON <table\_name>

**HiveQL - Select-Where**

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. The built-in operators and functions generate an expression, which fulfils the condition.

**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];

**HiveQL - Select-Order By**

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

**Syntax:**

SELECT [ALL | DISTINCT] select\_expr, select\_expr, ...

FROM table\_reference

[WHERE where\_condition]

[GROUP BY col\_list]

[HAVING having\_condition]

[ORDER BY col\_list]]

[LIMIT number];

**HiveQL - Select-Group By**

The GROUP BY clause is used to group all the records in a result set using a particular collection column. It is used to query a group of records.

**Syntax:**

SELECT [ALL | DISTINCT] select\_expr, select\_expr, ...

FROM table\_reference

[WHERE where\_condition]

[GROUP BY col\_list]

[HAVING having\_condition]

[ORDER BY col\_list]]

[LIMIT number];

**HiveQL - Select-Joins**

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.

**Syntax:**

join\_table:

table\_reference JOIN table\_factor [join\_condition]

| table\_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table\_reference

join\_condition

| table\_reference LEFT SEMI JOIN table\_reference join\_condition

| table\_reference CROSS JOIN table\_reference [join\_condition]

There are different types of joins given as follows:

JOIN

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

**JOIN**

JOIN clause is used to combine and retrieve the records from multiple tables. JOIN is same as OUTER JOIN in SQL. A JOIN condition is to be raised using the primary keys and foreign keys of the tables.

**LEFT OUTER JOIN**

The HiveQL LEFT 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.

A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.

**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.

A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate.

**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.

**Pig vs SQL**

The DBMS systems that SQL operates on, are considered to be faster than MapReduce (operated on by Pig through the PigLatin platform). However, it is the loading of data that is more challenging in case of RDBMS, making the set up difficult. PigLatin offers a number of advantages in terms of declaring execution plans, ETL routines and pipeline modification.

SQL is declarative and PigLatin is procedural to a large extent. What we mean by this is in SQL, we largely specify “what” is to be accomplished and in Pig, we mention “how” a task is to be performed. A script written in Pig is essentially converted to a MapReduce job in the background before it is executed. A Pig script is shorter than the corresponding MapReduce job, which significantly cuts down development time.

**Hive vs SQL**

SQL is a general purpose database language that has extensively been used for both transactional and analytical queries. Hive, on the other hand, is built with an analytical focus. What this means is Hive lacks update and delete functions but is superfast in reading and processing huge volumes of data faster than SQL. Hence, even though Hive SQL is SQL-like, lack of support for modifying or deleting data is a major difference.