

* Apache Pig :-

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data ~~known~~ representing them as data flows. Pig is generally used with Hadoop. We can perform all the data manipulation operations in Hadoop using Apache Pig.

To write data analysis programs, Pig provides a high-level language known as Pig Latin. This language provides various ~~operator~~ operators using which programmers can develop their own functions for reading, writing and processing data.

* Features of Pig :-

1. Rich set of operators :-

It provides many operators to perform operations like join, sort, filter etc.

2. Easy of programming :-

Pig Latin is similar to SQL and it is easy to write a pig script if we are good at SQL.

3. Optimization opportunities :-

The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.

4. Extensibility :-

Using the existing operators, users can develop their own functions to read, process and write data.

5. Handles all kinds of data: →

Apache pig analyzes all kinds of data, both structured as well as ~~unstructured~~ unstructured. It stores the results in HDFS.

* Admiring the pig Architecture: →

Components:

Pig is made up of two (count 'em, too)

1. The language itself: →

As proof that programmers have a sense of humor the programming language for pig is known as pig Latin, a high-level language that allows you to write data processing and analysis programs.

2. The pig Latin Compiler: →

The pig Latin Compiler converts the pig Latin code into executable code. The executable code is either in the form of MapReduce jobs or it can spawn a process where a virtual Hadoop instance is created to run the pig code on a single node.

The sequence of MapReduce program enable pig programs to do data processing and analysis in parallel, leveraging Hadoop MapReduce and HDFS. Running the pig job in virtual Hadoop instance is a useful strategy for testing out our pig scripts.

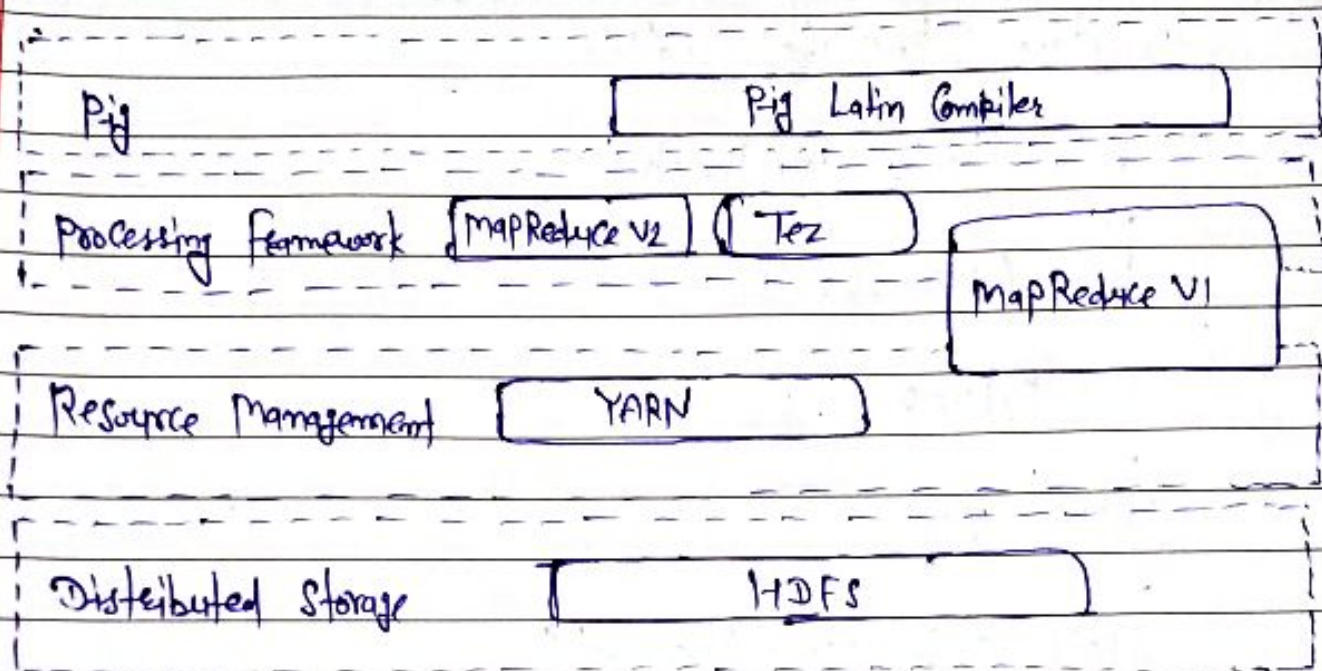


Figure 1: Pig Architecture

Pig Program Can run on MapReduce 1 or MapReduce 2 without any code changes, regardless of what mode your cluster is running. However, Pig Scripts Can also run using the Tez API instead.

Going with the Pig Latin Application flow →

At its Core, Pig Latin is a dataflow language, where ~~you~~^{we} define a data stream and a series of transformations that are applied to the data as it flows through our application. This is in contrast to a Control flow language (like C or Java), where ~~we~~^{we} write a series of instructions. In Control flow language, we use constructs like loop and Conditional logic (like an if statement) we won't find loops and if statements in Pig Latin.

If we need some convincing that working with pig is a significantly easier than having to write Map and Reduce programs, start by taking a look at some real pig syntax.

The following listing specifies sample pig code to illustrate the data processing dataflow.

```
A = LOAD 'data-file.txt';
```

```
...
```

```
B = GROUP ...;
```

```
C = FILTER ...;
```

```
...
```

```
DUMP B;
```

```
..
```

```
STORE C int INTO 'Results';
```

↳

* Working through the ABCs of Pig Latin: →

Pig Latin is the language for pig programs. Pig translates the pig Latin script into MapReduce jobs that can be executed within Hadoop cluster. When coming up with Pig Latin, the development team followed three key design principles:-

(1) Keep it Simple: →

Pig Latin provides a streamlined method for interacting with Java MapReduce. It's an abstraction, in other words, that simplifies the creation of parallel programs on the Hadoop cluster for data flows and analysis.

Complex tasks may require a series of interrelated data transformations - such series are encoded as data flow sequences.

Writing data transformation and flows as Pig Latin scripts instead of Java MapReduce programs make these programs easier to write, understand and maintain.

(2) Make it Smart :→

You may recall that the Pig Latin Compiler does the work of transforming a Pig Latin program into a series of Java MapReduce jobs. The trick is to make sure that the Compiler can optimize the execution of these Java MapReduce jobs automatically, allowing the user to focus on semantics rather than on how to optimize and access the dataset.

(3) Don't limit development :→

make ~~map~~ Pig extensible so that developers can add functions to address their particular business problems.

* Evaluating Local and Distributed modes of Running Pig Scripts :→

Before you can run your first Pig script, you need to have a handle on how Pig programs can be packaged with the Pig Server.

Pig has two modes for running script, as shown in figure :-

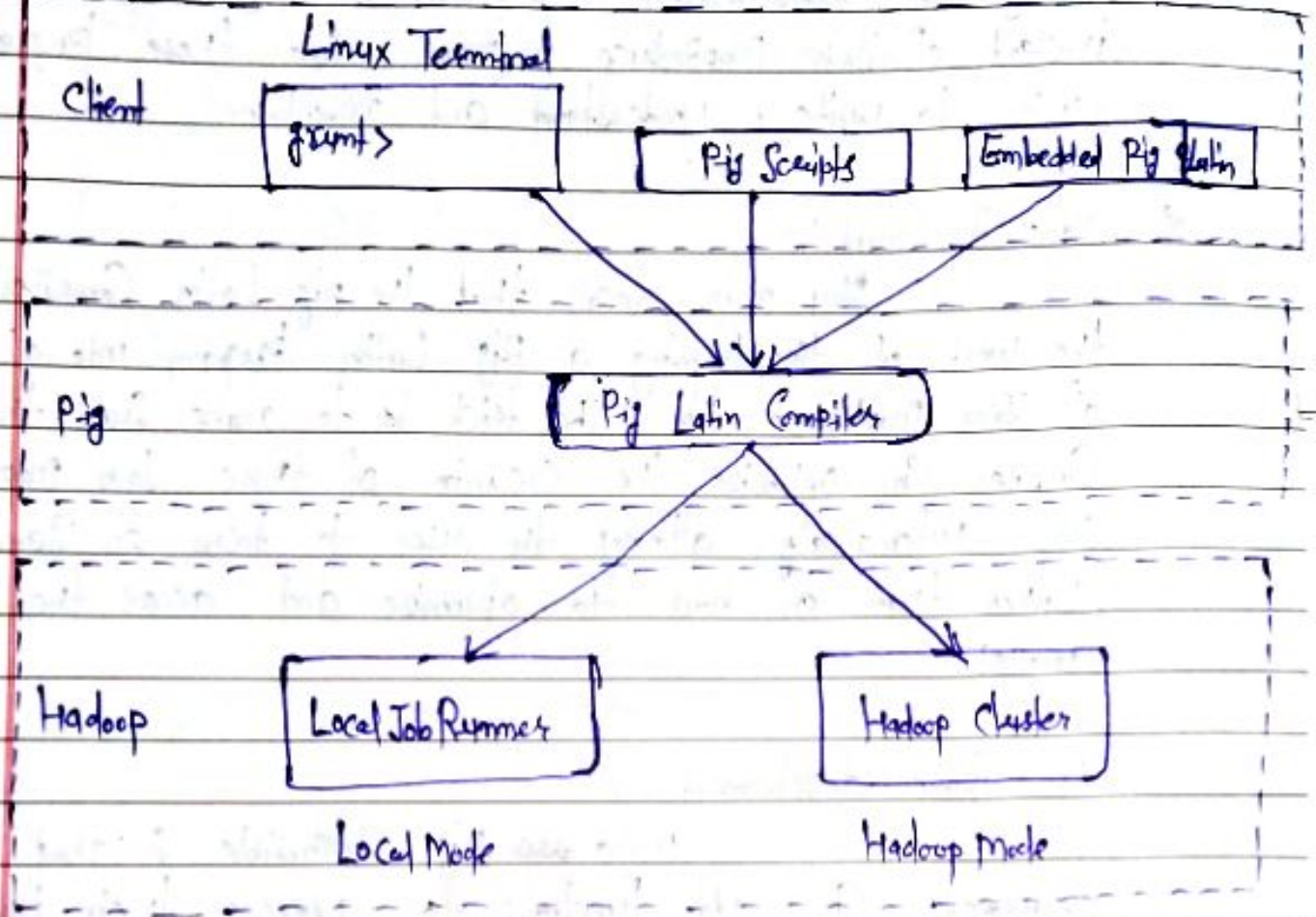


Figure: Pig Modes

497

Local mode:

All Scripts are run on a single machine without requiring Hadoop MapReduce and HDFS. This can be useful for developing and testing pig logic. If you're using a small set of data to develop or test your code, then local mode could be faster than going through the MapReduce infrastructure.

Local mode doesn't require Hadoop. When you run in Local mode, the pig program runs in the context of a local Java Virtual Machine, and data access is via the local file system of a single machine. Local mode is actually

a local simulation of MapReduce in Hadoop's LocalJobRunner class.

MapReduce Mode / Hadoop mode →

MapReduce mode is also known as Hadoop Mode. Pig is executed on the Hadoop cluster. In this case, the Pig script gets converted into a series of MapReduce jobs that are then run on the Hadoop cluster. If you have a terabyte of data that you want to perform operations on and you want to interactively develop a program, you may soon find things slowing down considerably and you may start growing your storage.

* Checking out the Pig Script Interfaces →

The Pig programming language is designed to handle any kind of data tossed its way - structured, semi-structured, unstructured data, etc. Pig programs can be packaged in three different ways:-

(1) Script →

This method is nothing more than a file containing Pig Latin commands, identified by the pig suffix (FlightData.pig). Ending your pig program with the .pig extension is a convention but not required. The commands are interpreted by the Pig Latin compiler and executed in the order determined by the pig optimizer.

(2) Grunt :->

~~Grunt~~ Grunt acts as a Command interpreter where ~~you~~^{we} can interactively enter pig Latin at the Grunt Command line and immediately see the response. This method is helpful for prototyping during initial development and with what-if scenarios.

(3) Embedded :->

Pig Latin statements can be executed within Java, Python or Javascript programs.

Pig Scripts, Grunt shell pig Commands, and Embedded pig programs can run in either Local mode or MapReduce mode. The Grunt shell provides an interactive shell to submit pig Commands or run pig scripts.

To start the Grunt ~~shell~~ ~~is executed~~ ~~is~~ Interactive mode, just submit the Command pig at your shell. To specify whether a script or Grunt shell is executed locally or in Hadoop mode just specify it in the -X flag to the pig Command.

The following is an example of how you'd specify running your pig script in local mode.

- pig -x local milesPerCarrier.pig

Here's how you'd run the pig script in Hadoop mode, which is the default if you don't specify the flag.

- pig -x mapreduce milesPerCarrier.pig

By default, when we specify the pig Command without any parameters, it starts the Grunt shell in Hadoop mode. If we want to start the Grunt shell in local mode just add the -x local flag to the Command.

Here is an example:-

Pig -x local

* Scripting with Pig Latin →

Hadoop is a rich and quickly evolving ecosystem with a growing set of new applications. Rather than try to keep up with all the requirements for new capabilities, pig is designed to be extensible via user-defined functions, also known as ~~UDF~~ UDFs. UDFs can be written in a number of programming languages, including Java, Python and Javascript. Developers are also posting and sharing a growing collection of UDFs online.

(Look for ~~Piggy~~ Piggy Bank and DataFu, to name just two examples of such online collections)

Some of the pig UDFs that are part of these repositories are LOAD/STORE function (XML, for example), date time, functions, text, math, and stats functions.

Pig can also be embedded in host languages such as Java, Python and Javascript, which allows you to integrate pig with your existing applications.

* HDFS Architecture :->

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

HDFS Architecture.

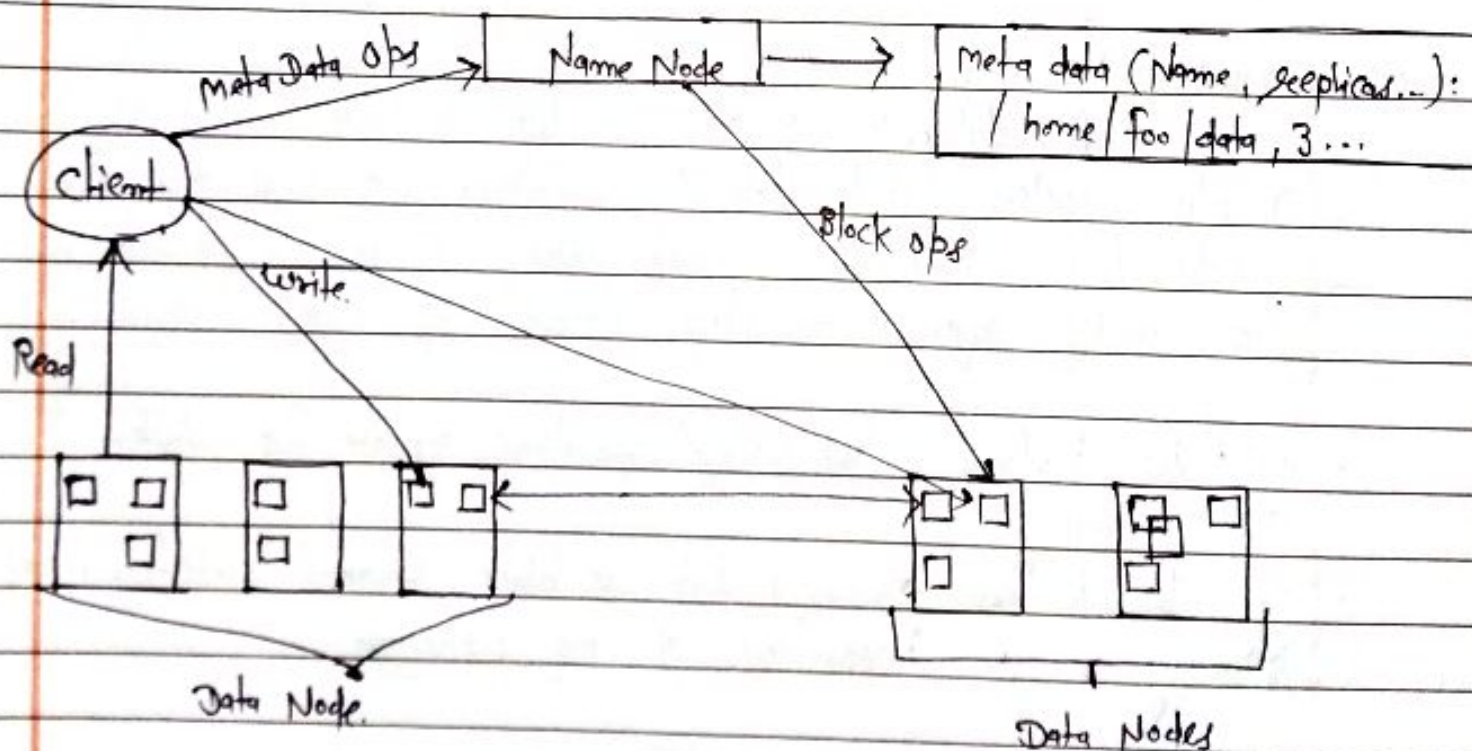


Figure 1: HDFS Architecture.

→ NameNode :-

The NameNode is the commodity hardware that contains the GNU/Linux operating system and the NameNode Software. It is a software that can be run on commodity hardware. The system having the NameNode acts as the master server and it does the following tasks -

- Manages the files
- Store meta data
- It also executes file system operations such as renaming, closing, and opening files and directories.

→ DataNode :-

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (commodity hardware/system) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- DataNodes perform read-write operations on the file system, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the NameNode.

→ NameNode :-

The NameNode is the commodity hardware that contains the GNU/Linux operating system and the NameNode Software. It is a software that can be run on commodity hardware. The system having the NameNode acts as the master server and it does the following tasks -

- Manages the files
- Store meta data
- It also executes file system operations such as renaming, closing, and opening files and directories.

→ DataNode :-

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (commodity hardware/system) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- DataNodes perform read-write operations on the file system, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the NameNode.

* Hive :->

Hive is a data warehouse system which is used to analyze structured data. It is built on the top of Hadoop. It was developed by Facebook.

Hive provides the functionality of reading, writing, and managing large datasets residing in distributed storage. It supports 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 :->

There are the following features of Hive:-

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

7. It supports user-defined functions (UDFs) where user can provide its functionality.

* Limitations of Hive →

1. Hive is not capable of handling real-time data.
2. It is not designed for online transaction processing.
3. Hive queries contain high latency.

* Difference between Hive and Pig →

Hive	Pig
1. Hive is commonly used by Data Analysts.	Pig is commonly used by programmers.
2. It follows SQL-like queries	It follows the data-flow language.
3. It can handle structured data.	It can handle semi-structured data.
4. It works on server-side of HDFS cluster.	It works on client-side of HDFS cluster.
5. Hive is slower than pig.	Pig is comparatively faster than Hive.

* Modes of Hive :-

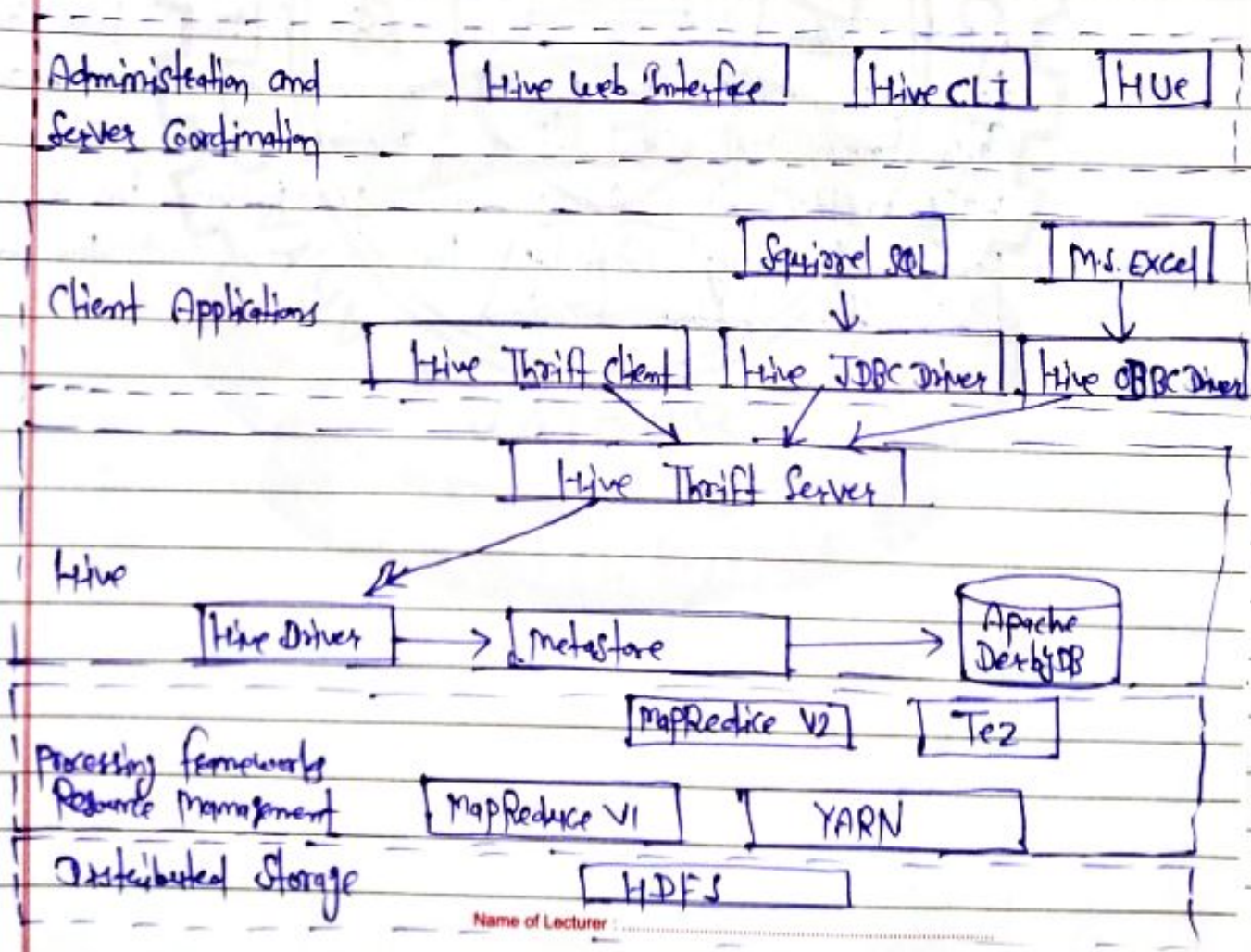
Hive can operate in two modes depending on the size of data Nodes in Hadoop.

These modes are:-

1. Local mode
2. Map reduce mode

* ~~Hive~~ Hive is put Together :-

In this we illustrate the architecture of Apache Hive and explain its various components, as shown in the illustration in figure:- following figure:-



In the above figure we can see at the bottom the Hive sits on top of the Hadoop Distributed File System (HDFS) and MapReduce systems. In the case of MapReduce, figure 1 shows both the Hadoop1 and Hadoop2 components. With Hadoop 1, Hive queries are converted to MapReduce code and executed using the MapReduce V1 (MRV1) infrastructure, like the JobTracker and TaskTracker. With Hadoop 2, YARN has decoupled resource management and scheduling from the MapReduce framework. Hive queries can still be converted to MapReduce code and executed, now with MapReduce framework V2 (MRV2) and the YARN infrastructure.

There is a new framework under development called Apache Tez, which is designed to improve Hive performance for batch-style queries and support smaller interactive (also known as real-time) queries.

HDFS provides the storage, and MapReduce provides the parallel processing capability for higher-level functions within the Hadoop ecosystem.

* Getting started with Apache Hive :-

Installation :-

The setup steps run something like this :-

1. Download the latest Hive ->
we downloaded Hive Version 11.0. You also need the Hadoop and MapReduce Subsystems, so be sure to complete Step-2
2. Download Hadoop Version 1.2.1
3. Using the Commands in the following listing, place the releases in separate directories, and then Uncompress and Untar them.

(Untar is one of those pesky Unix terms which simply means to expand an archived software package.)

```
$ mkdir hadoop; cp hadoop-1.2.1.tar.gz hadoop; cd hadoop
$ gunzip hadoop-1.2.1.tar.gz
$ tar xvf *.tar
$ mkdir hive; cp hive-0.11.0.tar.gz hive; cd hive
$ gunzip hive-0.11.0.tar.gz
$ tar xvf *.tar
```

4. Using the commands in the following listing, set up your Apache Hive Environment Variable, including ~~hadoop~~ HADOOP-HOME, JAVA-HOME, HIVE-HOME and PATH, in your shell profile script:

```
export HADOOP-HOME=/home/user/hive/hadoop/hadoop-1.2.1
export JAVA-HOME=/opt/jdk
export HIVE-HOME=/home/user/hive/hive-0.11.0
export PATH=$HADOOP-HOME/bin:$HIVE-HOME/bin:$JAVA-HOME/bin:$PATH
```

5. Create the Hive Configuration file that you'll use to define specific Hive Configuration Settings

```
$ cd $HIVE-HOME/conf
$ cp hive-default.xml.template hive-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="Configuration.xsl"?>
<configuration>
<!-- Hive Execution Parameters -->
<property>
  <name>hive.metastore.warehouse.dir</name>
  <value>/home/biadmin/hive/warehouse</value>
  <description>location of default database for the warehouse</description>
</property>
</configuration>
```

Because you're running Hive in stand-alone mode on a Virtual machine rather than in a real-life Apache Hadoop Cluster, Configure the System to use local storage rather than the HDFS: Simply set the `hive.metastore.warehouse.dir` parameter.

When you start a Hive client, the `$HIVE_HOME` environment variable tells the client that it should look for your Configuration file (`hive-site.xml`) in the `Conf` directory.

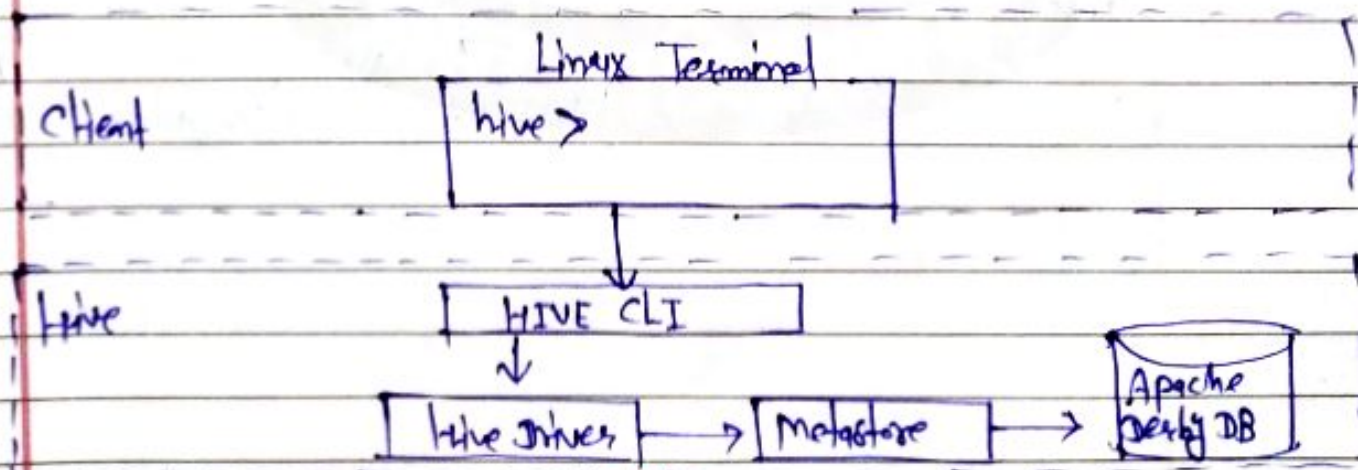
* Examining the Hive Clients: →

In general, we have the following hive clients:

1. Hive Command-Line interface (CLI)
2. Hive Web Interface (HWI) Server
3. Squirrel

1. The Hive CLI Client: →

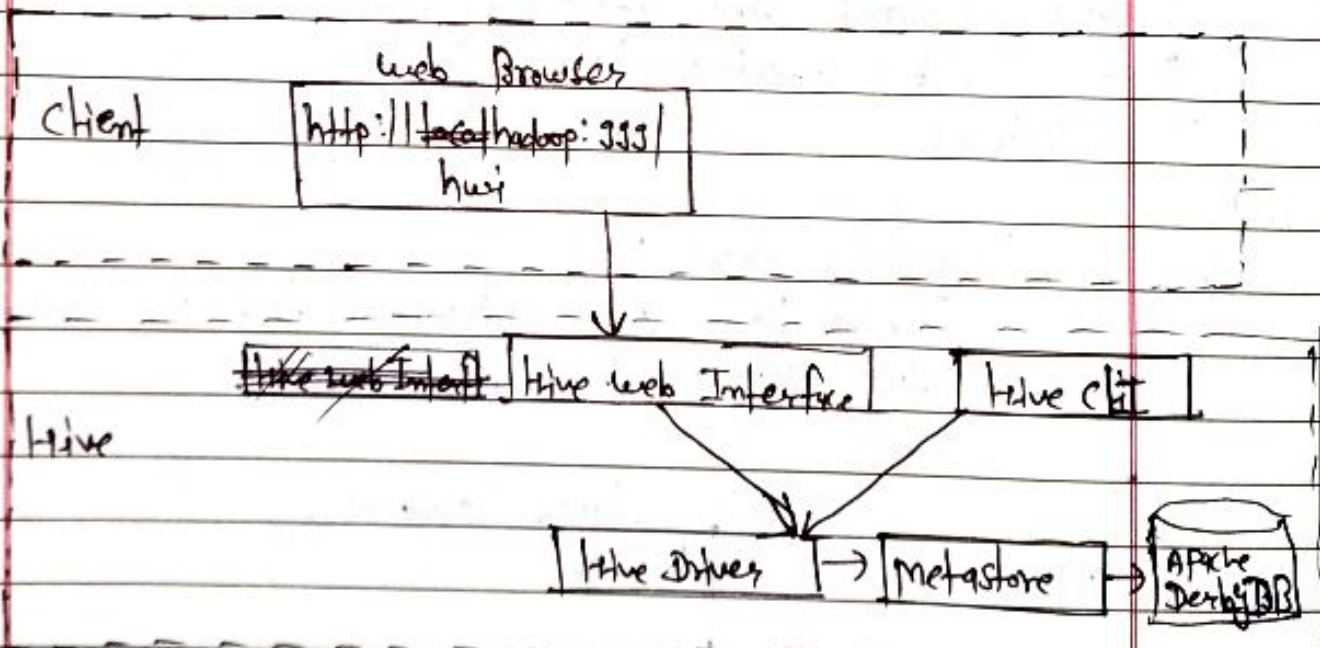
The following figure shows Components that are required when running the CLI on a Hadoop Cluster.



The examples in this chapter, we run Hive in local mode which uses local storage, rather than the HDFS, for your data. To run the HIVE @ CLI, you execute the hive Command and specify the CLI as the Service you want to run.

Q. Hive web Interface (HWI) Server! →

When we want to access Hive using a web browser, you first need to start Hive web Interface (HWI) Server and then point your browser to the port on which the Server is listening. following figure shows the HWI Client Configuration.



The following steps show you what you need to do before you can start the Hwi Server:

1. Configure the `$HIVE-HOME/conf/hive-site.xml` file as below to ensure that Hive can find and load the Hwi's Java Server Pages.

`<property>`

`<name> hive.hwi.war.file </name>`

`<value> $HIVE-HOME/lib/hive-hwi.war </value>`

`<description>`

This is the WAR file with the JSP content for Hive web Interface

`</description>`

`</property>`

2. The Hwi Server requires Apache Ant libraries to run, so download Ant.

3. Install Ant using the following Commands:

`mkdir ant`

`cp apache-ant-1.9.2-bin.tar.gz ant; cd ant`

`gunzip apache-ant-1.9.2-bin.tar.gz`

`tar xvf apache-ant-1.9.2-bin.tar`

4. Set the `$ANT-LIB` environment Variable and start the Hwi Server by using following Commands:

```
$ export ANT_LIB=/home/user/ant/apache-ant-1.9.2/lib
$ bin/hive --service hwi
```

3. Squirrel as Hive Client with the JDBC Driver! →
- The last HIVE client is the Open Source tool Squirrel SQL. It provides a user interface to Hive and simplifies the tasks of querying large tables and analyzing data with Apache Hive.

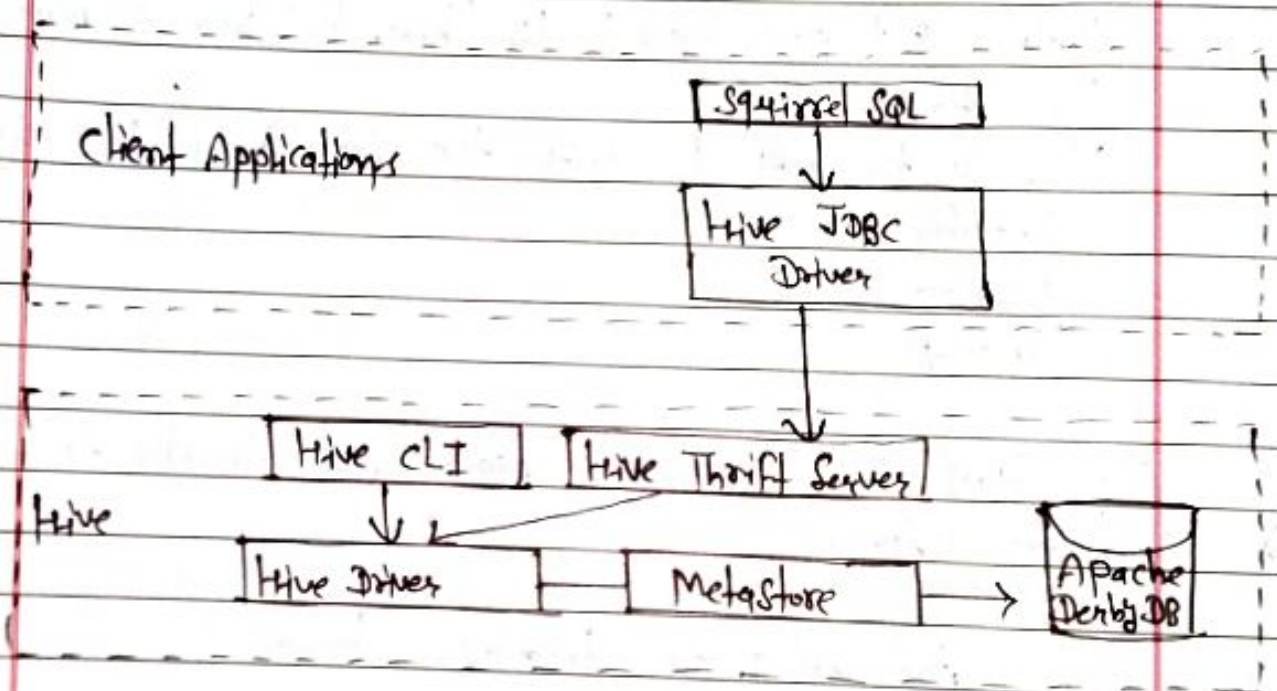


Figure:- Using the Squirrel client with Apache Hive.

In the above figure, we can see that the Squirrel client uses the JDBC APIs to pass commands to the Hive Driver by way of the Hive Thrift Server.

* Hive Data types :->

Data types are very important elements in Hive Query language and data modeling. For defining the table Column types, we must have to know about the data types and its usage.

The following gives brief overview of some data types in Hive these are:-

1. Numeric Types
2. String Types
3. Date/Time Types
4. Complex Types

1. Numeric Types :->

Type	Memory Allocation
(1) TINY INT	It is 1-byte signed integer (-128 to 127)
(2) SMALL INT	2-byte signed integer (-32768 to 32767)
(3) INT	4-byte signed integer
(4) BIG INT	8-byte signed integer
(5) FLOAT	4-byte signed precision floating point number
(6) DOUBLE	8-byte double precision floating point number
(7) DECIMAL	We can define precision and scale in this type.

2. String Types :-

Type	Length
(1) CHAR	955
(2) VARCHAR	1 to 65355
(3) STRING	we can define length here (no limit)

3. Date / Time types :-

Type	Usage
(1) Timestamp	Supports traditional Unix timestamp with optional nanosecond
(2) Date	<ul style="list-style-type: none"> It's in YYYY-MM-DD format The range of values supported for the Date type is from 0000-01-01 to 9999-12-31 dependent on support by the primitive <code>DATE</code> type.

4. Complex Types :-

Type	Usage
(1) Arrays	<code>Array<data-type></code> Negative values and non-constant expressions not allowed
(2) Maps	<code>MAP<primitive-type, data-type></code> Negative values and non-constant expressions not allowed.
(3) Structs	<code>STRUCT<col_name : data-type, ...></code>
(4) Union	<code>UNIONTYPE<data-type, data-type, ...></code>

* Creating and Managing Databases and Tables :->

=> Create Database :->

Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables.

Syntax of the Create Database :->

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

=> Drop Database Statement :->

Drop Database is a statement that drops all the tables and deletes the database. Its syntax is as follows:

```
[ DROP DATABASE Statement DROP (DATABASE | SCHEMA) [IF EXISTS]  
  database_name [RESTRICT | CASCADE]; ]
```

The following queries are used to drop a database.
Let us assume that the database name is `Myedb`.

```
[ hive> DROP DATABASE IF EXISTS Myedb; ]
```

The following query drops the database using CASCADE.
It means dropping respective tables before dropping the database.

```
[hive> DROP DATABASE IF EXISTS userdb CASCADE;]
```

The following query drops the database using SCHEMA.

```
[hive> DROP SCHEMA userdb;]
```

⇒ Create Table Statement :-

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:-

Syntax:-

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS]
[db-name] table-name
[col(col-name data-type [COMMENT col-comment],...)]
[COMMENT table-comment]
[ROW FORMAT row-format]
[STORED AS file-format]
```

~~If let us assume~~

Example:-

~~Let us assume here give the data~~

Example

Let us assume you need to create a table named Employee using CREATE TABLE statement. The following table lists the fields and their data types in Employee table:

Sr. No.	Field Name	Data Type
1	Eid	int
2	Name	String
3	Salary	float
4	Dest Designation	String

The following data is a Comment, Row formatted fields such as field terminator, lines terminator and stored file type.

```
COMMENT 'Employee details'
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n'
STORED IN TEXT FILE
```

The following query creates a table named Employee using the above data.

```
CREATE TABLE IF NOT EXISTS employee (eid int, name String,
Salary float, designation String)
COMMENT 'Employee details'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
```

LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

⇒ Load Data Statement →

(Generally, after creating a table in SQL, we can insert data using the Insert Statement. But in HIVE, we can insert data using the Load DATA Statement. While inserting data into HIVE, it is better to use LOAD DATA to store bulk records.

~~There are two ways to load data:~~

~~1. from local file system~~

~~2. from Hadoop file system.~~

Syntax:-

The syntax for load data is as follows:

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

- LOCAL is identifier to specify the local path. It is optional.
- OVERWRITE is optional to overwrite the data in the table.
- PARTITION is optional.

Example :-

We will insert the following data into the table. It is a text file named sample.txt in the ~~directory~~ /home/user directory.

1901	Gopal	50000	OP Admin
1902	Ram	45000	HR Admin
1903	Shyam	55000	Prof Reader
1904	Mohan	35000	Technical Manager
1905	Manisha	60000	Branch Manager

The following query loads the given text into the table.

```
[ hive> LOAD DATA LOCAL INPATH '/home/user/sample.txt'
  OVERWRITE INTO TABLE Employee;
```

On successful download, you get to see the following response

OK

Time taken: 15.905 seconds

hive>

* Alter Table Statement :->

It is used to alter a table in Hive.

Syntax :-

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

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 ...])

Rename To ... Statement :->

The following query renames the table from Employee to Emp.

```
ALTER TABLE Employee RENAME TO Emp;
```

* Change Statement :->

The following table contains the fields of employee table and it shows the fields to be changed (in Black)

Field Name	Convert from Data type	change Field Name	Convert to Data type
eid	int	eid	int
name	String	ename	String
Salary	float	Salary	Double
designation	String	designation	String

The following queries rename the Column name and Column data type using the above data.

```
[ ALTER TABLE Employee CHANGE name ename String;  
  ALTER TABLE Employee CHANGE Salary Salary Double; ]
```

* Add Column Statement :->

The following query adds a Column named dept to the Employee table.

```
hite> ALTER TABLE Employee ADD Col COLUMN (dept  
      STRING COMMENT 'Department name');
```

* Replace Statement :->

The following query deletes all the columns from the Employee table and replaces it with Emp and name columns:

```
hive> ALTER TABLE Employee REPLACE COLUMNS (  
    eid INT Empid INT,  
    Ename STRING name String);
```

* Drop TABLE Statement :->

The syntax is as follows:-

```
[ DROP TABLE [IF EXISTS] table-name; ]
```

The following query drops a table named Employee.

```
hive> DROP TABLE IF EXISTS Employee;
```

* Hive Data Manipulation Language (DML) Commands :->

Hive DML (Data Manipulation Language) Commands are used to Insert, update, retrieve, and delete data from the Hive table once the table and database Schema has been defined using Hive DDL Commands.

The Various Hive DML Commands are :

1. ~~Load~~ LOAD
2. SELECT
3. INSERT
4. DELETE
5. UPDATE
6. EXPORT
7. IMPORT

1. LOAD Command :->

The LOAD Statement in Hive is used to move data files into the locations corresponding to Hive tables.

- > If a LOCAL keyword is specified, then the LOAD Command will look for the file path in the local file system.
- > If the LOCAL keyword is not specified, then the Hive will need the absolute URI of the file.
- > In Case the keyword OVERWRITE is specified, then the contents of the target table / partition will be deleted and replaced by the files referred by file path.
- > If the OVERWRITE keyword is not specified, then the files referred by file path will be appended to the table.

Syntax →

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE  
tablename [PARTITION (partCol1 = Val1, partCol2 = Val2 ...)];
```

Example →

Here we are trying to load data from the 'dab' file in the local filesystem to the 'emp-data' table.

```
> LOAD DATA LOCAL INPATH '/home/dataflair/dab' INTO TABLE  
emp-data;
```

9. SELECT COMMAND →

The SELECT statement in Hive is similar to the SELECT statement in SQL used for retrieving data from the database.

Syntax →

```
SELECT col1, col2 FROM tablename;
```

Example →

```
SELECT * FROM emp-data;
```

3. INSERT Command :->

The INSERT Command in Hive loads the data into a Hive table. We can do insert to both the Hive table or partition.

(a) INSERT INTO :->

The INSERT INTO Statement appends the data into existing data in the table or partition. INSERT INTO Statement works from Hive Version 0.8.

Syntax ->

```
[ INSERT INTO TABLE tablename1 [PARTITION (partcol1 = val1,  
partcol2 = val2 ...)] select-statement1 FROM from-statement;
```

Example :->

Here in this example, we are trying to Insert the data of 'emp-data' table created above into the table 'example'.

firstly we create the table :->

```
> CREATE TABLE IF NOT EXISTS example (id STRING, name  
STRING, dep STRING, state STRING, salary STRING, year  
STRING);
```

Now Insert Statement to load data into table "example".

```
[ > INSERT INTO TABLE example SELECT Emp.Emp-Id, Emp.Emp-Name  
, Emp.Emp-dep, Emp.State, Emp.Salary, Emp.Year-of-Joining FROM  
Emp-data Emp; ]
```

by Dubey

Name of Lecturer :

(b) INSERT OVERWRITE :-

The INSERT OVERWRITE table overwrites the existing data in the table or partition.

Syntax :-

```
[ INSERT OVERWRITE TABLE tablename [ PARTITION (partcol1 =  
Val1, ...) [ IF NOT EXISTS ] select - statement FROM  
from - statement ; ]
```

Example :-

Here we are overwriting the existing data of the table 'example' with the data of ~~table~~ ^{Database} 'dummy' using INSERT OVERWRITE statement.

```
> INSERT OVERWRITE TABLE Example SELECT dummy. enroll, dummy.  
name, dummy. department, dummy. state, dummy. salary, dummy. year  
FROM dummy dummy;
```

By using the SELECT statement we can verify whether the existing data of the table 'example' is overwritten by the data of table 'dummy' or not.

```
• set SELECT * from example;
```

(C). INSERT.. VALUES ! →

INSERT.. VALUES statement in Hive inserts data into the table directly from SQL. It is available from Hive 0.14.

Syntax →

```
[ INSERT INTO TABLE tablename [PARTITION (partCol1 [= Val1],  
[ partCol2 [= Val2] ... ) ] VALUES values_row [, values_row ... ]
```

Example! →

Inserting data into the 'student' table using INSERT.. VALUES statement.

```
> INSERT INTO TABLE student VALUES (101, 'Gillen', 'IT', '7.8'),  
(103, 'Joseph', 'CS', '8.2'), (105, 'Alex', 'IT', '7.5');
```

```
> SELECT * FROM student;
```

4. DELETE Command :->

The DELETE statement in hive deletes the table data. If the WHERE clause is specified, then it deletes the rows that satisfy the condition in where clause.

The DELETE statement can only be used on the hive tables that support ACID (ACID property -> Atomicity, Consistency, Isolation, Durability).

Syntax ->

DELETE FROM tablename [WHERE expression];

Example :->

In the below example, we are deleting the data of the student from table 'student' whose roll-no is 105.

```
> DELETE FROM student WHERE roll-no = 105;
```

By using the SELECT statement we can verify whether the data of the student from table 'student' whose roll-no is 105 is deleted or not.

```
> SELECT * FROM student;
```

5. UPDATE Command :->

The update can be performed on the hive tables that support ACID.

The update statement in hive deletes the table data. If the WHERE clause is specified, then it updates the column of the rows that satisfy the condition in WHERE clause.

Syntax:-

```
UPDATE tablename SET Column = Value [, Column = Value...]  
[WHERE expression];
```

Example:->

In this example, we are updating the branch of the student whose roll-no is 103 in the 'student' table using an ~~update~~ UPDATE statement.

```
> UPDATE student SET branch = 'IT' WHERE roll-no = 103;
```

6. Export EXPORT Command :->

The hive EXPORT statement exports the table's ^{partitions} data along with the metadata to the specified output location in the HDFS.

Metadata is exported in a -metadata file, and data is exported in a subdirectory 'data'.

Hive Partitions! →

Apache Hive organizes table into partitions. Partitioning is a way of dividing a table into related parts based on the values of particular columns like date, City and department. Each table in the hive can have one or more partition keys to identify a particular partition. Using partition it is easy to do queries on slices of the data.

Syntax! →

Export

```
EXPORT TABLE tablename [PARTITION (part-column = "value" [, ...])] TO 'export-target-path' [FOR replication ('eventid')];
```

Example! →

Here in this example, we are exporting the Student table to the HDFS directory "export-from-hive".

```
> EXPORT TABLE Student TO 'export-from-hive';
```

The table successfully exported. You can check for the — metadata file and data sub-directory using ls command

```
$ hadoop fs -ls -R /user/data/hive/export-from-hive/;
```

7. ~~Impo~~

7. IMPORT Command !→

The Hive IMPORT Command imports the data from a specified location to a new table or already existing table.

Syntax !→

```
IMPORT [EXTERNAL] TABLE new-or-original-tablename  
[PARTITION (part-column = "Value" [, ...])] ]  
FROM 'Source-path' [LOCATION 'import-target-path'];
```

Example !→

Here in this example, we are importing the data exported in the above example into a new Hive table 'imported-table'.

```
> import IMPORT TABLE imported-table from '/user/datafilex/export-  
from-hive';
```

* Querying and Analyzing Data: →

- Five data types, Hive's DDL and Hive's DML, helps creating and managing tables but now we help you explore some HiveQL features for querying and analysing data. We begin by exploring table joins in Hive.

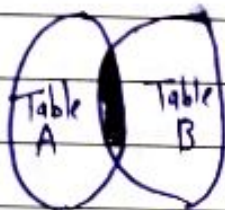
Joining tables with Hive: →

- In relational database modelling, we split the tables for normalization purpose and use join operation get the data when it is required.
- Database normalization is a technique that guards against data loss, redundancy, and other anomalies as data is update and retrieved.
- MapReduce is the engine for joining tables, and the Hadoop file system (HDFS) is the underlying storage.
- Hive table reads and writes via HDFS usually involve very large blocks of data, more data you can manage altogether in one table, the better the overall performance.
- With this background information in mind, we can tackle making joins with Hive. Fortunately, the Hive development community was realistic and understood that users would want and need to join tables with HiveQL.
- Hive supported Equijoin, a specific type of join that only uses equality comparisons in the join predicate.
- This ~~restriction~~ restriction is only because of limitations on the underlying MapReduce engine.

Hive Join Example :-

1. Inner Join :-

~~Full outer join :-~~



Basically, to combine and retrieve the records from multiple tables we use Hive Join clause. Moreover, in SQL JOIN is as same as OUTER JOIN. Moreover, by using the primary keys and foreign keys of the tables JOIN condition is to be raised.

Furthermore, the below query executes JOIN the CUSTOMER and ORDER tables. Then further retrieves the records.

```
> SELECT C.ID, C.Name, C.AGE, O.AMOUNT FROM CUSTOMER  
C JOIN ORDERS O ON (C.ID = O.CUSTOMER-ID);
```

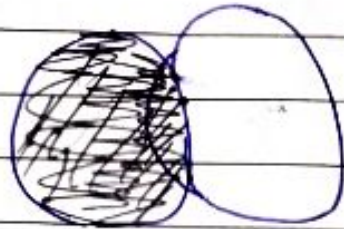
2. Left Outer Join :-

On defining HiveQL Left Outer Join, even if there are no matches in the right table it returns all the rows from the left table.

In addition, it returns all the values from the left table. Also, the matched values from the right table, or NULL in case of no matching JOIN predicate.

However, the below query shows LEFT OUTER JOIN between CUSTOMER as well as ORDER tables.

> SELECT C.ID, C.Name, O.Amount, O.DATE FROM CUSTOMERS C LEFT OUTER JOIN ORDERS O ON (C.ID = O.CUSTOMER ID);



3. Full Outer Join →

The major purpose of this HiveQL Full Outer Join is it combines the records of both the left and the right outer tables which fulfill the Hive JOIN Condition. Moreover, this joined table contains either all the records from both the tables or fills in NULL values for missing matches on either side.

However, the below query shows FULL OUTER JOIN between CUSTOMER as well as ORDER tables:

> SELECT C.ID, C.NAME, O.AMOUNT, O.DATE FROM CUSTOMERS C FULL OUTER JOIN ORDERS O ON (C.ID = O.CUSTOMER-ID);



⇒ Improving Your Hive queries with Indexes :→

Creating an Index is common practice with relational databases when we want to speed access to a Column or Set of Columns in your database. Without an index, the database system has to read all rows in the table to find the data we have selected. Indexes become even more essential when the tables grow extremely large. Hive supports index creation on tables.

⇒ Hive Indexes :→

Hive indexes are implemented as tables. ~~This is why we need to first~~ Create the index table and then build it to populate the table.

Therefore, we can use indexes in at least two ways.

1. Count on the system to automatically use indexes that you create.
2. Rewrite some queries to leverage the new index table.

⇒ Windowing in HiveQL :→

The Concept of Windowing, introduced in the SQL:2003 standard, allows the SQL programmer to create a frame from the data against which aggregate and other window functions can operate.

HiveQL now supports windowing per the SQL standard. Examples are quite helpful when explaining windowing and aggregate functions.

- We first discovered this data set was, 'what exactly is the average flight delay per day?' So we created a query in Listing 13-13 that produces the average departure delay per day in 2008.

* Other Key HIVEQL Features :->

1. Security :->

Apache Hive provides a security subsystem that can be quite helpful in preventing accidental data corruption or compromise among trusted members of workgroup.

2. Multi-User Locking :->

Hive supports multi-user warehouse access when configured with Apache Zookeeper. Without this support, one user may read a table at the same time another user is deleting that table - which is, obviously, unacceptable.

3. Functions :->

HIVEQL provides a rich set of built-in operators, built-in functions, built-in aggregate functions, and built-in table-generating functions. Several examples in this chapter use built-in operators as well as built-in aggregate function (AVG, MIN, and COUNT, for example).

To list all built-in functions for any particular Hive ~~release~~ release, use the SHOW FUNCTIONS HIVEQL Command. You can also retrieve information about a built-in function by using the HIVEQL Commands DESCRIBE FUNCTION function-name and DESCRIBE FUNCTION EXTENDED function-name.

4. Com Compression! →

Data Compression can not only save space on the HDFS but also improve performance by reducing the overall size of input/output operations.

Additionally, Compression between the Hadoop mappers and reducers can improve performance, because less data is passed between nodes in the cluster.

Hive supports intermediate Compression between the mappers and reducers as well as table output.

Compression. Hive also understands how to ingest Compressed data into the warehouse. Files Compressed with Gzip or Bzip2 can be read by Hive's LOAD DATA Command.