# UNIT 5

# APPLYING STRUCTURE TO HADOOP DATA WITH HIVE

## HIVE Introduction

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on topof Hadoop to summarize Big Data and makes querying and analyzing easy.

The term „Big Data‟ is used for collections of large datasets that include huge volume, high velocity, and a variety of data that is increasing day by day. Using traditional data management systems, it is difficult to process Big Data. Therefore, the Apache Software Foundation introduced a framework called Hadoop to solve Big Data management and processing challenges.

## Hadoop

Hadoop is an open-source framework to store and process Big Data in a distributed environment.It contains two modules, one is MapReduce and another is Hadoop Distributed File System (HDFS).

- **MapReduce**: It is a parallel programming model for processing large amounts of structured, semi-structured, and unstructured data on large clusters of commodity hardware.
- **HDFS**: Hadoop Distributed File System is a part of Had the oop framework, used to store and process the datasets. It provides a fault-tolerant file system to run on commodity hardware.

The Hadoop ecosystem contains different sub-projects (tools) such as Sqoop, Pig, and Hive thatare used to help Hadoop modules.

- **Sqoop**: It is used to import and export data to and from between HDFS and RDBMS.
- **Pig**: It is a procedural language platform used to develop a script for MapReduceoperations.
- **Hive**: It is a platform used to dev SQL-typescriptsipts to do MapReduce operations.

**Note**: There are various ways to execute MapReduce operations:

- The traditional approach using Java MapReduce program for structured, sem semistructured, and unstructured data.
- The scripting approach for MapReduce to process is structured as semiestructureddta usingPig.
- The Hive Query Language (HiveQL or HQL) for MapReduce to process structured datausing Hive.

## What is Hive

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on topof 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 is not**

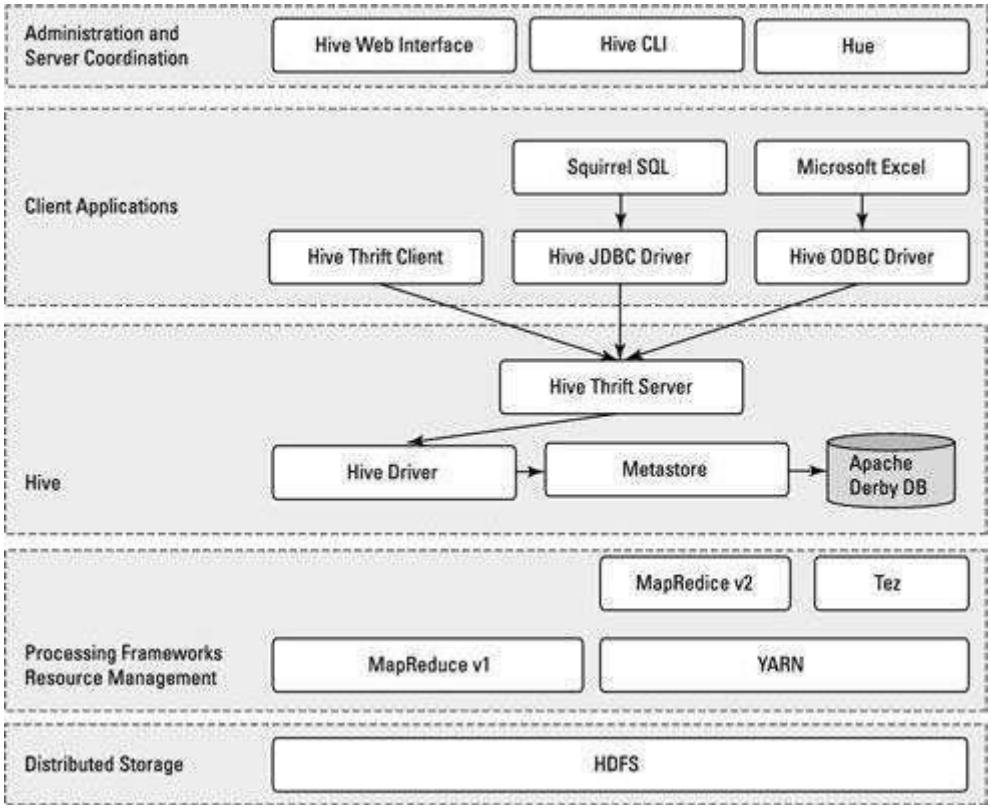
- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

**Features of Hive**

- It stores schema in a database and processes 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.

**Architecture of Hive**

The following component diagram depicts the architecture of Hive:



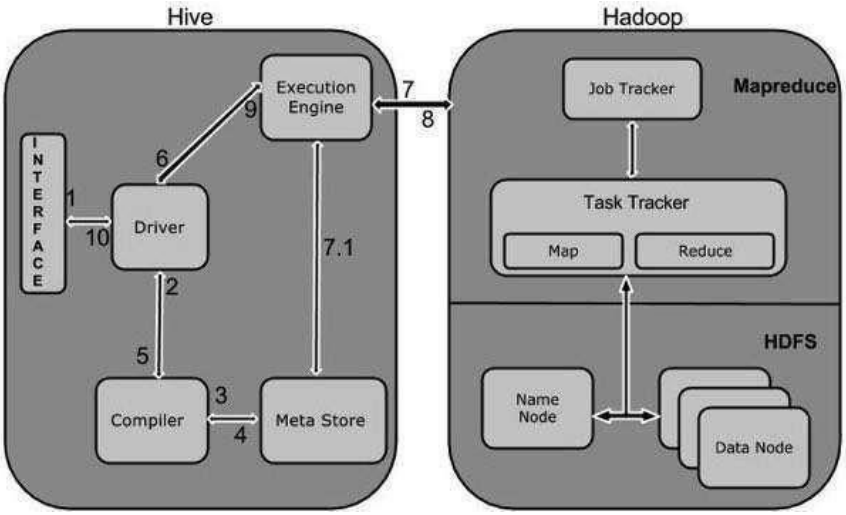

Apache Hive architecture

This component diagram contains different units. The following table describes each unit:

| Unit Name | Operation |
|---|---|
| User Interface | Hive is a data warehouse infrastructure software that can create interaction between users and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows) |
| Meta Store | Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping. |
| HiveQL Process Engine | HiveQL is similar to SQL for querying schema info on the Metastore. It is one of the replacements for the traditional approach or the MapReduce program. Instead of writing MapReduce program in Java, we can write a query for the Math aare MapReduceand process it. |
| Execution Engine | The conjunction part of HiveQL process Engine and MapReduce is Hive Execution EnginThe execution engineprocesses the query and generates results as same as MapReduce results. It uses the flavor of MapReduce |
| HDFS oHBaseSE | Hadoop distributed file systems or HBASE is the data storage techniques to store data in the file system. |

**Working of Hive**

The following diagram depicts the workflow between Hive and Hadoop.




The following table defines how Hive interacts with the Hadoop framework:

| Step No. | Operation |
|---|---|
| 1 | **Execute Query**<br>The Hive interfaces such Command-Lineine or Web UI sena ds query to Driver (anydatabase driver such as JDBC, ODBC, etc.) to execute. |
| 2 | **Get Plan**<br>The driver takes the helpofn the f query compiler that parses the query to check thesyntax and query plan or the requirement of the f query. |

| | |
|---|---|
| 3 | **Get Metadata**<br>The compiler sends metadata requests to Metastore (any database). |
| 4 | **Send Metadata**<br>Metastore sends metadata as a response to the compiler. |
| 5 | **Send Plan**<br>The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query are complete. |
| 6 | **Execute Plan**<br>The driver sends the execute plan to the execution engine. |
| 7 | **Execute Job**<br>Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in the Name node and it assigns this job to TaskTracker, which is in the Data node. Here, the query executes the MapReduce job. |
| 7.1 | **Metadata Ops**<br>Meanwhile, in execution, the execution engine can execute metadata operationswith Metastore. |
| 8 | **Fetch Result**<br>The execution engine receives the results from Data nodes. |
| 9 | **Send Results**<br>The execution engine sends those resultant values to the driver. |
| 10 | **Send Results**<br>The driver sends the results to Hive Interfaces. |

**Hive - Data Types**

All the data types in Hive are classified into four types, given as follows:

- Column Types
- Literals
- Null Values
- Complex Types

**Column Types**

Column types are used as column data types of Hive. They are as follows:

**Integral Types**

Integer type data can be specified using integral data types, INT. When the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT. TINYINT is smaller than SMALLINT.

The following table depicts various INT data types:

| Type | Postfix | Example |
|---|---|---|
| TINYINT | Y | 10Y |
| SMALLINT | S | 10S |
| INT | - | 10 |

| | | |
|---|---|---|
| BEGIN | L | 10L |

**String Types**

String type data types can be specified using single quotes (' ') or double quotes (" "). It contains two data types: VARCHAR and CHAR. Hive follows C-type escape characters.

The following table depicts various CHAR data types:

| Data Type | Length |
|---|---|
| VARCHAR | 1 to 65355 |
| CHAR | 255 |

**Timestamp**

It supports traditional UNIX timestamps with optional nanosecond precision. It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM: SS.fffffffff” and format “yyyy-mm-dd hh: mm :ss .fffffffff”.

**Dates**

DATE values are described in year/month/day format in the form {{YYYY- MM-DD}}.

**Decimals**

The DECIMAL type in Hive is as same as the Big Decimal format of Java. It is used for representingimmutable arbitrary precision. The syntax and example are as follows:

**Union Types**

Union is a collection of heterogeneous data types. You can create an instance using create union. The syntax and example is as follows:

```
UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>
{0:1}
{1:2.0}
{2:["three","four"]}
{3:{"a":5,"b":"five"}}
{2:["six","seven"]}
{3:{"a":8,"b":"eight"}}
{0:9}
{1:10.0}
```

**Literals**

The following literals are used in Hive:

**Floating Point Types**

Floating-point types are nothing but numbers with decimal points. Generally, this type of data iscomposed of DOUBLE data types.

**Decimal Type**

Decimal type data is nothing but floating-point value with a higher range than a DOUBLE data type.The range of decimal type is approximately -10 -308 to 10 38.

**Null Value**

Missing values are represented by the special value NULL.

**Complex Types**

The Hive complex data types are as follows:

**Arrays**

Arrays in Hive are used the same way they are used in Java.

Syntax: ARRAY<data_type>

**Maps**

Maps in Hive are similar to Java Maps.

Syntax: MAP<primitive_type, data_type>

**Structs**

Structs in Hive are similar to using complex data with comments.

Syntax: STRUCT<col_name: data_type [COMMENT col_comment], ...>

**Hive - Create Database**

- Hive is a database technology that can define databases and tables to analyze structured data. The theme for structured data analysis is to store the data in a tabular manner and pass queries to analyze it. This chapter explains how to create a Hive database. Hive containsa default database named **default**.

**Create Database Statement**

- Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables. The syntax for this statement is as follows:
  CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
- Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named used:

hive> CREATE DATABASE [IF NOT EXISTS] userdb;
or
hive> CREATE SCHEMA userdb;

- The following query is used to verify a databases list:

hive> SHOW DATABASES;
default
used

**Drop Database Statement**

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

DROP DATABASE StatementDROP (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 used.

hive> DROP DATABASE IF EXISTS used;

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 used;

**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_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[ROW FORMAT row_format] [STORED AS file_format]
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 the employee table:

| S. No. | Field Name | Data Type |
|---|---|---|
| 1 | Eid | int |
| 2 | Name | String |
| 3 | Salary | Float |
| 4 | 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\tLINES TERMINATED
BY\nSTORED IN TEXT FILE
```

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

```
hive> CREATE TABLE IF NOT EXISTS employee(eid int, name String, salary String,
destination String)

COMMENT Employee details ROW FORMAT DELIMITED FIELDS TERMINATED
BY\tLINES TERMINATED BY\nStored AS TEXTFILE;
```

If you add the optional IF NOT EXISTS, Hive ignores the statement in case the table already

exists.On successful creation of the table, you get to see the following response:

```
OK
Time took: 5.905 seconds
hive>
```

**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: one is from the local file system and the second is from the Hadoop file system.

**Syntax**

The syntax for load data is as follows:

```
LOAD DATA [LOCAL] INPATH „file path“ [OVERWRITE] INTO TABLE table name
[PARTITION [partcol1=val1, partcol2=val2 ...)]
```

- LOCAL isan 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 **/home/user** directory.

| | | | |
|---|---|---|---|
| 1201 | Gopal | 45000 | Technical manager |
| 1202 | Manisha | 45000 | Proofreader |
| 1203 | Masthanvali | 40000 | Technical writer |
| 1204 | Kiran | 40000 | Hr Admin |
| 1205 | Kranthi | 30000 | Op Admin |

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 took 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 modifya 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 employeetempp.

hive> ALTER TABLE employee RENAME TO emp;

**Change Statement**

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

| Field name | Convert from data type | Change field name | Convert to data type |
|---|---|---|---|
| eid | int | eid | int |
| **name** | name | **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:

hive> ALTER TABLE employee CHANGE name ename String;

hive> ALTER TABLE employee CHANGE salary Double;

**Add Columns Statement**

The following query adds a column named dept to the employee table.

```
hive> ALTER TABLE employee ADD COLUMNS (dept STRING COMMENT
„Department name“);
```

**Replace Statement**

The following query deletes all the columns from the employee table and replaces them with emp andname columns:

```
hive> ALTER TABLE employee REPLACE COLUMNS (eid INT empid Int, name
STRING);
```

**Drop-Table Statement**

Hive Metastore removes the table/column data and their metadata. It can be a normal table (stored in Metastore) or an external table (stored in the local file system); Hive treats both in the samemanner, irrespective of their types.

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;
```

On successful execution of the query, you get to response:

```
OK
Time took: 5.3 seconds
hive>
```

The following query is used to verify the list of tables:

```
hive> SHOW TABLES;
emp ok
Time took: 2.1 seconds
hive>
```

**Operators in HIVE:**

There are four types of operators in Hive:

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Complex Operators

**Relational Operators:** These operators are used to compare two operands. The following table describes the relational operators available in Hive:

| Operator | Operand | Description |
|---|---|---|
| A = B | all primitive types | TRUE if expression A is equivalent to expression B otherwise FALSE. |
| A != B | all primitive types | TRUE if expression A is not equivalent to expression B otherwise FALSE. |
| A < B | all primitive types | TRUE if expression A is less than expression B otherwise FALSE. |
| A <= B | all primitive types | TRUE if expression A is less than or equal to expression B otherwise FALSE. |
| A > B | all primitive types | TRUE if expression A is greater than expression B otherwise FALSE. |
| A >= B | all primitive types | TRUE if expression A is greater than or equal to expression B otherwise FALSE. |
| A IS NULL | all types | TRUE if expression A evaluates to NULL otherwise FALSE. |
| A IS NOT NULL | all types | FALSE if expression A evaluates to NULL otherwise TRUE. |
| ALIKE B | Strings | TRUE if string pattern A matches to B otherwise FALSE. |
| AN LIKE B | Strings | NULL if A or B is NULL, TRUE if any substring of A matches the Java regular expression B, otherwise FALSE. |
| A REGEXP B | Strings | Same as RLIKE. |

Example

Let us assume the employee table is composed of fields named Id, Name, Salary, Designation, and Dept as shown below. Generate a query to retrieve the employee details whose Id is 1205.

```
+-----+--------------+--------+---------------------------+------+
| Id | Name          | Salary | Designation               | Dept |
+-----+--------------+--------+---------------------------+------+
|1201 | Gopal        | 45000  | Technical manager         | TP   |
|1202 | Manisha      | 45000  | Proofreader               | PR   |
|1203 | Masthanvali  | 40000  | Technical writer          | TP   |
|1204 | Kiran        | 40000  | Hr Admin                  | HR   |
|1205 | Kranthi      | 30000  | Op Admin                  | Admin|
+-----+--------------+--------+---------------------------+------+
```

The following query is executed to retrieve the employee details using the above table:

```
hive> SELECT * FROM employee WHERE Id=1205;
```

On successful execution of the query, you get to see the following response:

```
+-----+-----------+-----------+---------------------------------+
| ID | Name      | Salary | Designation                    | Dept |
+-----+--------------+-------+---------------------------------+
|1205 | Kranthi | 30000  | Op Admin                        | Admin|
+-----+-----------+-----------+---------------------------------+
```

The following query is executed to retrieve the employee details whose salary is more than or equal to Rs 40000.

```
hive> SELECT * FROM employee WHERE Salary>=40000;
```

On successful execution of query, you get to see the following response:

```
+-----+------------+--------+----------------------------+------+
| ID  | Name       | Salary | Designation                | Dept |
+-----+------------+--------+----------------------------+------+
|120  | Gopal      | 45000  | Technical manager          | TP   |
|120  | Manisha    | 45000  | Proofreader                | PR   |
|1203 | Masthanvali| 40000  | Technical writer           | TP   |
|1204 | Kiran      | 40000  | Hr Admin                   | HR   |
+-----+------------+--------+----------------------------+------+
```

**Arithmetic Operators**

These operators support various common arithmetic operations on the operands. All of them return number types. The following table describes the arithmetic operators available in Hive:

| Operators | Operand | Description |
|---|---|---|
| A + B | all number types | Gives the result of adding A and B. |
| A - B | all number types | Gives the result of subtracting B from A. |
| A * B | all number types | Gives the result of multiplying A and B. |
| A / B | all number types | Gives the result of dividing B from A. |
| A % B | all number types | Gives the remainder resulting from dividing A by B. |
| A & B | all number types | Gives the result of bitwise AND of A and B. |
| A \| B | all number types | Gives the result of bitwise OR of A and B. |
| A ^ B | all number types | Gives the result of bitwise XOR of A and B. |
| ~A | all number types | Gives the result of bitwise NOT of A. |

Example

The following query adds two numbers, 20 and 30.

```
hive> SELECT 20+30 ADD FROM temp;
```

On successful execution of the query, you get to see the following response:

```
+_____+
| ADD |
+_____+
|  50 |
+_____+
```

**Logical operators**

The operators are logical expressions. All of them return either TRUE or FALSE.

| Operators | Operands | Description |
|---|---|---|
| A AND B | boolean | TRUE if both A and B are TRUE, otherwise FALSE. |
| A && B | boolean | Same as A AND B. |

| | | |
|---|---|---|
| AN ORB | boolean | TRUE if either A or B or both are TRUE, otherwise FALSE. |
| A \|\| B | boolean | Same as A ORB. |
| NOT A | boolean | TRUE if A is FALSE, otherwise FALSE. |
| !A | boolean | Same as NNOTA |

Example

The following query is used to retrieve employee details whose Department is TP and Salary is more than Rs 40000.

```
hive> SELECT * FROM employee WHERE Salary>40000 && Dept=TP;
```

On successful execution of the query, you get to see the following response:

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
+------+--------------+-------------+-------------------+--------+
```

**Complex Operators**

These operators provide an expression to access the elements of Complex Types.

| Operator | Operand | Description |
|---|---|---|
| A[n] | A is an Array and n is an int | It returns the nth element in the array A. The first element has an index of 0. |
| M[key] | M is a Map<K, V> and key has type K | It returns the value corresponding to the key in the map. |
| S.x | S is a struct | It returns the x field of S. |

**HiveQL - Select-Where**

- The Hive Query Language (HiveQL) is a query language for Hive to process and analyze structured data in a Metastore. This chapter explains how to use the SELECT statement with the WHERE clause.
- 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 fulfills the condition.

Syntax

Given below is the syntax of the SELECT query:

```
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];
```

Example

Let us take an example for SELECT…WHERE clause. Assume we have the employee table as given below, with fields named Id, Name, Salary, Designation, and Dept. Generate a query to retrieve the employee details who earn a salary of more than Rs 30000.

| ID | Name | Salary | Designation | Dept |
|---|---|---|---|---|
| 1201 | Gopal | 45000 | Technical manager | TP |
| 1202 | Manisha | 45000 | Proofreader | PR |
| 1203 | Masthanvali | 40000 | Technical writer | TP |
| 1204 | Krian | 40000 | Hr Admin | HR |
| 1205 | Kranthi | 30000 | Op | Admin |

The following query retrieves the employee details using the above scenario:

```
hive> SELECT * FROM employee WHERE salary>30000;
```

On successful execution of the query, you get to see the following response:

| ID | Name | Salary | Designation | Dept |
|---|---|---|---|---|
| 1201 | Gopal | 45000 | Technical manager | TP |
| 1202 | Manisha | 45000 | Proofreader | PR |
| 1203 | Masthanvali | 40000 | Technical writer | TP |
| 1204 | Krian | 40000 | Hr Admin | HR |

**HiveQL - Select-Order By**

This chapter explains how to use the 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.

Syntax

Given below is the syntax of the ORDER BY clause:

```
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];
```

Example

- Let us take an example for SELECT...ORDER BY clause. Assume the employee table as givenbelow, with the fields named Id, Name, Salary, Designation, and Dept.
  Generate a query to retrieve the employee details in order by using the Department name.

```
+------+--------------+-------------+--------------------+--------+
| ID   | Name         | Salary      | Designation        | Dept   |
+------+--------------+-------------+--------------------+--------+
|1201 | Gopal         | 45000       | Technical manager  | TP     |
|1202 | Manisha       | 45000       | Proofreader        | PR     |
|1203 | Masthanvali   | 40000       | Technical writer   | TP     |
|1204 | Krian         | 40000       | Hr Admin           | HR     |
|1205 | Kranthi       | 30000       | Op Admin           | Admin  |
+------+--------------+-------------+--------------------+--------+
```

The following query retrieves the employee details using the above scenario:

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

**HiveQL - Select-Group By**

This chapter explains the details of the GROUP BY clause in a SELECT statement. The GROUP BYclause is used to group all the records in a result set using a particular collection column. It is usedto query a group of records.

The syntax of the GROUP BY clause is as follows:

```
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

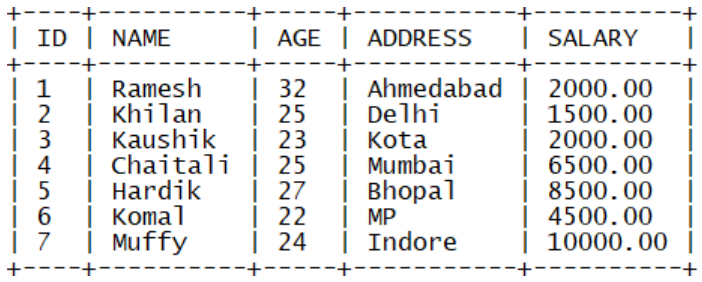
join_condition

| table_reference LEFT SEMI JOIN table_reference join_condition

| table_reference CROSS JOIN table_reference [join_condition]

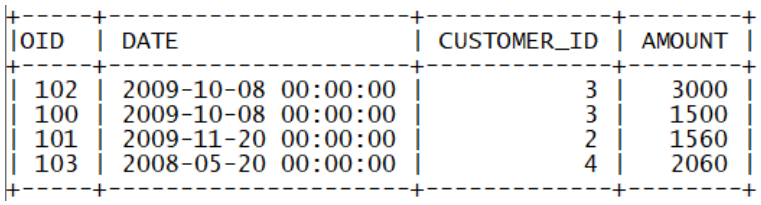
Example

We will use the following two tables in this chapter. Consider the following table named CUSTOMERS.



| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | Kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | MP | 4500.00 |
| 7 | Muffy | 24 | Indore | 10000.00 |

Consider another table ORDERS as follows:



| OID | DATE | CUSTOMER_ID | AMOUNT |
|---|---|---|---|
| 102 | 2009-10-08 00:00:00 | 3 | 3000 |
| 100 | 2009-10-08 00:00:00 | 3 | 1500 |
| 101 | 2009-11-20 00:00:00 | 2 | 1560 |
| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

**JOIN**

There are different types of joins given as follows:

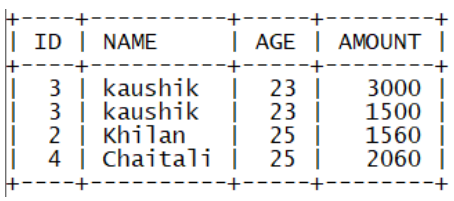
- JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

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

The following query executes JOIN on the CUSTOMER and ORDER tables, and retrieves the records:

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT FROM CUSTOMERS c JOIN
ORDERS o ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:



| ID | NAME | AGE | AMOUNT |
|---|---|---|---|
| 3 | kaushik | 23 | 3000 |
| 3 | kaushik | 23 | 1500 |
| 2 | Khilan | 25 | 1560 |
| 4 | Chaitali | 25 | 2060 |

**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, that 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.

The following query demonstrates LEFT OUTER JOIN between CUSTOMER and ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c LEFT
OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  1 | Ramesh   |   NULL | NULL                |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|  5 | Hardik   |   NULL | NULL                |
|  6 | Komal    |   NULL | NULL                |
|  7 | Muffy    |   NULL | NULL                |
+----+----------+--------+---------------------+
```

**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.

The following query demonstrates RIGHT OUTER JOIN between the CUSTOMER and ORDER tables.

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c RIGHT
OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    3 | kaushik  | 3000   | 2009-10-08          |
|    3 | kaushik  | 1500   | 2009-10-08          |
|    2 | Khilan   | 1560   | 2009-11-20          |
|    4 | Chaitali | 2060   | 2008-05-20          |
+------+----------+--------+---------------------+
```

**FULL OUTER JOIN**

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

The following query demonstrates FULL OUTER JOIN between CUSTOMER and ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c FULL
OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   | NULL   | NULL                |
|    2 | Khilan   | 1560   | 2009-11-20 00:00:00 |
|    3 | kaushik  | 3000   | 2009-10-08 00:00:00 |
|    3 | kaushik  | 1500   | 2009-10-08 00:00:00 |
|    4 | Chaitali | 2060   | 2008-05-20 00:00:00 |
|    5 | Hardik   | NULL   | NULL                |
|    6 | Komal    | NULL   | NULL                |
|    7 | Muffy    | NULL   | NULL                |
|    3 | kaushik  | 3000   | 2009-10-08 00:00:00 |
|    3 | kaushik  | 1500   | 2009-10-08 00:00:00 |
|    2 | Khilan   | 1560   | 2009-11-20 00:00:00 |
|    4 | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```