

# Working with Hive

Tushar B. Kute,  
<http://tusharkute.com>

# Hadoop Ecosystem

- The Hadoop ecosystem contains different sub-projects (tools) such as Sqoop, Pig, and Hive that are used to help Hadoop modules.
  - **Sqoop:** It is used to import and export data to and fro between HDFS and RDBMS.
  - **Pig:** It is a procedural language platform used to develop a script for MapReduce operations.
  - **Hive:** It is a platform used to develop SQL type scripts to do MapReduce operations.

# 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.

# Integral Types

Type	Postfix	Example
TINYINT	Y	10Y
SMALLINT	S	10S
INT	-	10
BIGINT	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-types escape characters.

Data Type	Length
VARCHAR	1 to 65535
CHAR	255

# Column Types

- **Timestamp**

- It supports traditional UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM:SS.ffffffffff” and format “yyyymmdd hh:mm:ss.ffffffffff”.

- **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 Big Decimal format of Java. It is used for representing immutable arbitrary precision. The syntax and example is as follows:

```
DECIMAL(precision, scale)
```

```
decimal(10,0)
```

# 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 is composed of DOUBLE data type.
- **Decimal Type**
  - Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The range of decimal type is approximately  $-10^{-308}$  to  $10^{308}$ .
- **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 is similar to using complex data with comment.  
Syntax: `STRUCT<col_name : data_type [COMMENT col_comment], ...>`

# Database Operations

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 Hive database. Hive contains a default database named **default**.

# 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. 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.

# Create Database

- The following query is executed to create a database named mydb:

```
hive> CREATE DATABASE [IF NOT EXISTS] mydb;
```

or

```
hive> CREATE SCHEMA mydb;
```

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

```
hive> SHOW DATABASES;
```

```
default
```

```
mydb
```

# Drop Database

- Drop Database is a statement that drops all the tables and deletes the database.
  - Its syntax is as follows:
- The following queries are used to drop a database. Let us assume that the database name is mydb.

```
hive> DROP DATABASE IF EXISTS mydb;
```

# Drop Database

- 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 ;
```

- This clause was added in Hive 0.6.

# Create Table

- 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]
```



# Create Table : Example

Sr. No.	Field Name	Data type
1	Eid	Int
2	Name	String
3	Salary	Float
4	Designation	String

# Create Table : Example

- 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 '\t'
> 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: one is from local file system and second is from Hadoop file system.
- Syntax:
  - `LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]`

# Load data statement

```
1201    Gopal    45000    Technical manager
1202    Manisha 45000    Proof reader
1203    Masthanvali    40000    Technical writer
1204    Krian    40000    Hr Admin
1205    Kranthi 30000    Op Admin
```

```
LOAD DATA LOCAL INPATH '/home/rashmi/sample.txt'
> OVERWRITE INTO TABLE employee;
```

# Alter 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 ...])

# Alter Table – Rename to...

```
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 bold).

Field Name	Convert from Data Type	Change Field Name	Convert to Data Type
eid	int	eid	int
<b>name</b>	String	<b>ename</b>	String
salary	<b>Float</b>	salary	<b>Double</b>
designation	String	designation	String

# Change statement example

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



# Add column statement

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

# Replace statement

```
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;`

# Partitioning

- Hive organizes tables into partitions. It is a way of dividing a table into related parts based on the values of partitioned columns such as date, city, and department. Using partition, it is easy to query a portion 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.

# Partitioning - Example

- ALTER TABLE table\_name ADD [IF NOT EXISTS] PARTITION partition\_spec [LOCATION 'location1'] partition\_spec [LOCATION 'location2'] ...;
- partition\_spec: (p\_column = p\_col\_value, p\_column = p\_col\_value, ...)

```
hive> select * from file;
OK
102      Rajesh   IT      2010
103      Suresh  CS      2012
103      Awez    CS      2012
104      Parmeet CS      2010
Time taken: 0.074 seconds, Fetched: 4 row(s)
```

# Built-in operators

- There are four types of operators in Hive:
  1. Relational Operators
  2. Arithmetic Operators
  3. Logical Operators
  4. Complex Operators

# Relational operators

- $A = B$
- $A \neq B$
- $A < B$
- $A = B$
- $A \geq B$
- $A \leq B$
- A IS NULL
- A IS NOT NULL

# Relational operators – Example

```
hive> select * from file where yoj<2011;
OK
102      Rajesh  IT      2010
104      Parmeet CS      2010
Time taken: 0.11 seconds, Fetched: 2 row(s)
hive> select * from file where dept='CS';
OK
103      Suresh  CS      2012
103      Awez    CS      2012
104      Parmeet CS      2010
Time taken: 0.084 seconds, Fetched: 3 row(s)
hive> select * from file where dept!='CS';
OK
102      Rajesh  IT      2010
Time taken: 0.124 seconds, Fetched: 1 row(s)
hive> █
```



# Arithmetic operators

- $A + B$
- $A - B$
- $A * B$
- $A / B$
- $A \% B$
- $A \& B$
- $A | B$
- $A \wedge B$
- $\sim A$

# Arithmetic operators

```
hive> SELECT id+1, dept FROM file;
OK
103      IT
104      CS
104      CS
105      CS
Time taken: 0.068 seconds, Fetched: 4 row(s)
hive> SELECT id, yoj%2000 FROM file;
OK
102      10
103      12
103      12
104      10
Time taken: 0.076 seconds, Fetched: 4 row(s)
hive> █
```

# Logical operators

- A AND B
- A && B
- A OR B
- A || B
- NOT A
- !A

# Logical operators

```
hive> SELECT * from file where id!=103 OR name IS NOT NULL;
OK
102      Rajesh  IT      2010
103      Suresh  CS      2012
103      Awez    CS      2012
104      Parmeet CS      2010
Time taken: 0.107 seconds, Fetched: 4 row(s)
hive> SELECT * from file where dept='CS' AND yoj=2012;
OK
103      Suresh  CS      2012
103      Awez    CS      2012
Time taken: 0.069 seconds, Fetched: 2 row(s)
hive> █
```

# Built-in functions

Return Type	Signature	Description
BIGINT	round(double a)	It returns the rounded BIGINT value of the double.
BIGINT	floor(double a)	It returns the maximum BIGINT value that is equal or less than the double.
BIGINT	ceil(double a)	It returns the minimum BIGINT value that is equal or greater than the double.
double	rand(), rand(int seed)	It returns a random number that changes from row to row.
string	concat(string A, string B,...)	It returns the string resulting from concatenating B after A.

# Built-in functions

string	<code>substr(string A, int start)</code>	It returns the substring of A starting from start position till the end of string A.
string	<code>substr(string A, int start, int length)</code>	It returns the substring of A starting from start position with the given length.
string	<code>upper(string A)</code>	It returns the string resulting from converting all characters of A to upper case.
string	<code>ucase(string A)</code>	Same as above.
string	<code>lower(string A)</code>	It returns the string resulting from converting all characters of B to lower case.

# Built-in functions

string	<code>lcase(string A)</code>	Same as above.
string	<code>trim(string A)</code>	It returns the string resulting from trimming spaces from both ends of A.
string	<code>ltrim(string A)</code>	It returns the string resulting from trimming spaces from the beginning (left hand side) of A.
string	<code>rtrim(string A)</code>	It returns the string resulting from trimming spaces from the end (right hand side) of A.

# Built-in functions – Example

```
hive> SELECT concat(name, dept), ucase(name) from file;
OK
RajeshIT          RAJESH
SureshCS         SURESH
AwezCS  AWEZ
ParmeetCS        PARMEET
Time taken: 0.059 seconds, Fetched: 4 row(s)
hive> SELECT round(2.6) from file;
OK
3.0
3.0
3.0
3.0
Time taken: 0.219 seconds, Fetched: 4 row(s)
hive> █
```



# Aggregate functions

Return Type	Signature	Description
BIGINT	count(*), count(expr),	count(*) - Returns the total number of retrieved rows.
DOUBLE	sum(col), sum(DISTINCT col)	It returns the sum of the elements in the group or the sum of the distinct values of the column in the group.
DOUBLE	avg(col), avg(DISTINCT col)	It returns the average of the elements in the group or the average of the distinct values of the column in the group.
DOUBLE	min(col)	It returns the minimum value of the column in the group.
DOUBLE	max(col)	It returns the maximum value of the column in the group.

# Examples

- `SELECT count(*) from file;`
- `SELECT sum(id) from file;`
- `SELECT avg(yoj) from file;`
- `SELECT max(yoj) from file;`

# Views

- Views are generated based on user requirements. You can save any result set data as a view.
- The usage of view in Hive is same as that of the view in SQL. It is a standard RDBMS concept.
- We can execute all DML operations on a view.
- Creating a view:

```
CREATE VIEW [IF NOT EXISTS] view_name  
[(column_name [COMMENT column_comment], ...)]  
[COMMENT table_comment]  
AS SELECT ...
```

# Views – example

```
hive> CREATE VIEW file_2010 AS
> SELECT * FROM file
> where yoj=2010;
OK
Time taken: 0.103 seconds
hive> select * from file_2010;
OK
102      Rajesh   IT      2010
104      Parmeet  CS      2010
Time taken: 0.08 seconds, Fetched: 2 row(s)
hive> █
```

# Dropping a view

- Use the following syntax to drop a view:

```
DROP VIEW view_name
```

- The following query drops a view named as file\_2010:

```
hive> DROP VIEW file_2010;
```

# Index

- An Index is nothing but a pointer on a particular column of a table.
- Creating an index means creating a pointer on a particular column of a table.
- 
- ```
hive> CREATE INDEX index_yoj ON TABLE file(yoj)  
> AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'  
WITH DEFERRED REBUILD;
```

# Index – Example

```
hive> CREATE INDEX in_salary ON TABLE file(yoj)
> AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;
OK
Time taken: 0.485 seconds
hive> show tables;
OK
class
emp_30000
file
file1
file_2010
tushar__file_in_salary__
tushar__file_index_salary__
Time taken: 0.019 seconds, Fetched: 7 row(s)
hive> drop index tushar__file_in_salary__ on file;
OK
Time taken: 0.027 seconds
hive> █
```

# Drop index

- The following syntax is used to drop an index:

```
DROP INDEX <index_name> ON <table_name>
```

- The following query drops an index named index\_salary:

```
hive> DROP INDEX index_salary ON employee;
```



# 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];
```

# Select ... order by- Example

```
hive> select * from file order by yoj;
Query ID = hduser_20160703164810_7d84d930-f1dd-4ed3-9410-1f09af20a74d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2016-07-03 16:48:13,401 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local590275424_0005
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 6000 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
104      Parmeet  CS      2010
102      Rajesh   IT      2010
103      Awez     CS      2012
103      Suresh  CS      2012
Time taken: 2.462 seconds, Fetched: 4 row(s)
```

# 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];
```

# Select... group by – example

```
hive> select dept, count(*) from file group by dept;
Query ID = hduser_20160703165351_da8962c1-3407-49bd-bd57-c463d2aab7ff
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2016-07-03 16:53:53,780 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1959421652_0007
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 6300 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
CS      3
IT      1
Time taken: 1.86 seconds, Fetched: 2 row(s)
```

# Joins

- JOINS 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 JOINS.

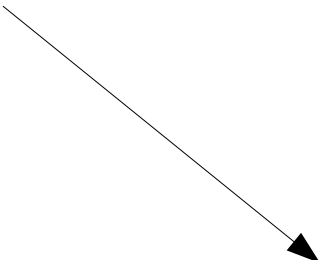
# Joins – Examples

```
hive> select * from customer;  
OK  
1      Kavita  24      Sangvi  34000  
2      Chatur  23      Kothrud 35000  
3      Fatema  31      Lohgad  20000  
4      Rohan   27      Pune Station 22000  
Time taken: 0.061 seconds, Fetched: 4 row(s)
```

```
hive> select * from orders;  
OK  
102    NULL    3      1200  
104    NULL    3      3400  
105    NULL    4      2150  
106    NULL    2      3420  
Time taken: 0.057 seconds, Fetched: 4 row(s)
```

# Joins – Examples

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
> FROM CUSTOMER c JOIN ORDERS o
> ON (c.ID = o.c_id);
Query ID = hduser_20160703175303_ac7c2fcc-c9f2-
Total jobs = 1
```



```
Total MapReduce CPU Time Spent: 0 msec
OK
2      Chatur   23      3420
3      Fatema   31      1200
3      Fatema   31      3400
4      Rohan    27      2150
Time taken: 9.21 seconds, Fetched: 4 row(s)
```

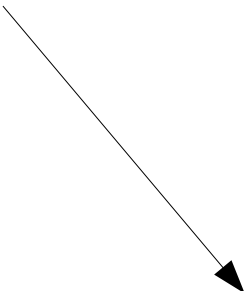
# Left outer join

- 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.



# Left outer join

```
hive> select c.ID, c.NAME, o.AMOUNT  
> FROM CUSTOMER c  
> LEFT OUTER JOIN ORDERS o  
> ON (c.ID = o.C_ID);
```



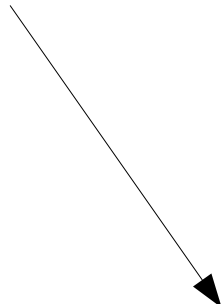
```
MapReduce Jobs Launched:  
Stage-Stage-3: HDFS Read: 106 HDFS Write: 0 SUCCESS  
Total MapReduce CPU Time Spent: 0 msec  
OK  
1      Kavita  NULL  
2      Chatur  3420  
3      Fatema  1200  
3      Fatema  3400  
4      Rohan   2150  
Time taken: 11.194 seconds, Fetched: 5 row(s)
```

# 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.

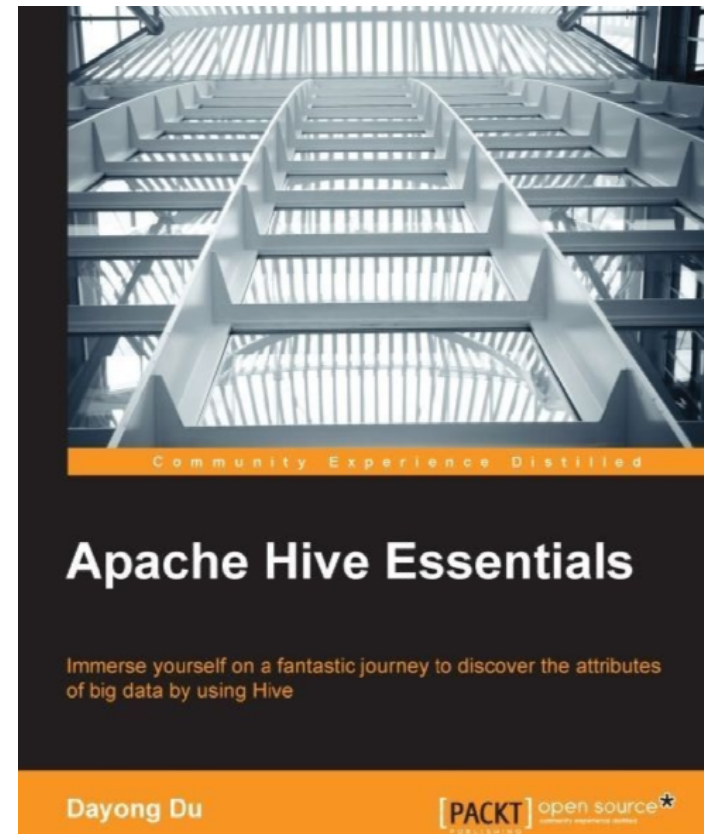
# Right outer join – Example

```
hive> select c.ID, c.NAME, o.AMOUNT  
> FROM CUSTOMER c  
> RIGHT OUTER JOIN ORDERS o  
> ON (c.ID = o.C_ID);
```



```
Stage-Stage-3: HDFS Read: 162 HDFS Write: 0 SUCCESS  
Total MapReduce CPU Time Spent: 0 msec  
OK  
3      Fatema  1200  
3      Fatema  3400  
4      Rohan   2150  
2      Chatur  3420  
Time taken: 18.488 seconds, Fetched: 4 row(s)
```

# References



# Thank you

*This presentation is created using LibreOffice Impress 4.2.8.2, can be used freely as per GNU General Public License*

## **Web Resources**

<http://mitu.co.in>  
<http://tusharkute.com>

## **Blogs**

<http://digitallocha.blogspot.in>  
<http://kyamputar.blogspot.in>

**[tushar@tusharkute.com](mailto:tushar@tusharkute.com)**