APACHE HIVE

Transactional System :

- Is the one where we deal with day to day data.
- Present data. Inserts and updates
- Ex. ATM
- Fast real-tome access
- RDBMS best suited for transaction systems
- MySQL, Oracle
- For a TS, a monolithic(one machine) system is the best suited

Analytical System:

- Is the one where we deal with historical data.
- Ex. Lets say running a campaign. If you have to analyze that why are sales less in this month.
- In such a system we mostly talk about reads not write.
- DataWareHouse are best suited for Analytical systems.
- Ex. Teradata
- Long running jobs and multiple data sources
- For AS, Distributed is best suited.
- Hive is meant to solve analytical problems

- **HIVE** is an open source datawarehouse. Code can be changed as it is open source.
- We mostly talk about reads in Hive.
- Is meant for analytical problems.

Language:

Mapreduce is in Java
So if we have structured data, then is there a way that we can visualize this structured data in the form of table. And is there a way that we fire some SQL like queries.
As a developer we will write simple SQL like queries (**HQL**)
What hive will do is, it will convert these queries as a map reduce job and will submit on the hadoop cluster.

Hive is nothing but abstracting the complexities from the user.
In hive, we will see data in form of tables.

A Hive table has 2 parts:

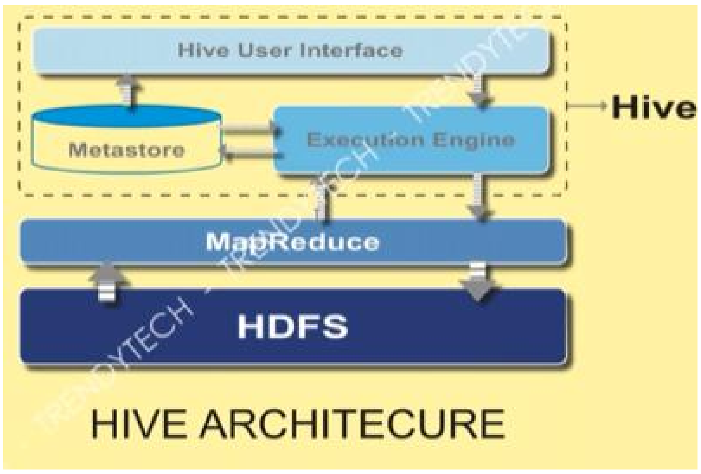
- **Actual Data** (HDFS)
- **Metadata** (Schema) - is stored in Database

The metadata is not stored in HDFS coz

- In HDFS we cannot do edits.
- If something is stored in hdfs then we cannot retrieve it very quickly. (**High Latency**)
- We store metadata in database like mysql, coz db provide low latency and also we can edit the schema easily.
- At runtime ie when we execute select * from employee; then on data in hdfs schema is applied so we get a tabular view at runtime, both of these are merged to show you like a table.

HIVE: is a data warehouse tool to process structured data on top of Hadoop.
Traditional Dbs does not provide parallelism, but Hive provides parallelism.

Hive Architecture



Summary

- ✧ Hive is an abstraction layer on top of Map reduce.
- ✧ Apache Hive is an open source data warehouse and part of the larger ecosystem.
- ✧ Hive stores its data in HDFS.
- ✧ Hive runs all processes in the form of Map Reduce jobs.
- ✧ We write SQL-like query in HiveQL and submit it to Hive.
- ✧ Hive will translate the query to Map Reduce tasks and run them on Hadoop cluster.
- ✧ Hive abstracts away the details of the underlying MapReduce jobs.
- ✧ Work with Hive almost exactly like you work with a traditional database.
- ✧ In hadoop data is stores in the form of files in HDFS.
- ✧ Mysql is the db where hive metadata is stored. Inside this, there is a db with the name **metastore.**
- ✧ When data is stored in the form of files in HDFS then how we will get a tabular view?
- ✧ Hive exposes files in HDFS in the form of tables to the user.

Actual data	**Meta Data**
Sumit Bangalore 1000	Emp Name(string)
Satish Delhi 20000	City (String)
Ankita Ludhiana 30000	Salary(int)
Venkat Bangalore 18000	
Atul Cochi 15000	

- ✧ If you go with open source or raw version of hadoop then it comes by default with derby database.
- ✧ limitations of derby db: light weight, lot of restrictions, not production ready so not recommended.
- ✧ Recommended db for production: MySQL.

Data Warehouse:

- ➢ Long running batch jobs
- ➢ Optimized for read operations
- ➢ Holds data from multiple sources
- ➢ Holds data over a long period of time
- ➢ Data may be lagged, not real-time
- ➢ Ex. Of DW: Vertica, Teradata, Oracle, IBM

Hive Metastore:

- ➢ The bridge b/w data stored in files and the tables exposed to users.
- ➢ Stores metadata for all the tables in Hive.
- ➢ Maps the files and directories in Hive to tables.
- ➢ Holds table definitions and the schema for each table.
- ➢ Has info on converting files to table representations.
- ➢ Any db with a JDBC driver can be used as a metastore.
- ➢ Development environment use the built-in **Derby** db ie **Embedded metastore**

Seek time: time taken to fetch a particular single record is called as the seek time.

Hive	**RDBMS**
Large datasets	Small datasets
Parallel computations	Serial computations (mono)
High latency(bad)	Low latency (seek time is less)
Read operations	Read/write operations
Not ACID compliant by default	ACID complaint
Hive QL	**SQL**
Schema on read, no constraints enforced	Schema on write keys, not null, unique all enforced
Minimal index support	Indexes allowed
Row level updates, deletes as a special case	Row level operations allowed in general
Many more built-in functions	Basic built-in functions
Only equi-joins allowed	No restriction on joins
Restricted subqueries	Whole range of subqueries

Storing Data in Hive

- Data is stored in HDFS, the reliable storage for data in Hadoop.
- Files partitioned across multiple machines in the cluster based on default block size.
- By default, data is stored in **warehouse dir.**
  /user/hive/warehouse - deafult location.
  /user/hive/warehouse/trendytech.db
  /user/hive/warehouse/trendytech.db/employee/files
  /user/hive/warehouse/employee/files <------------ this is in case when we have the table in default db.
- Default loc is set using this property:
- Property set in hive-site.xml ===> hive.metastore.warehouse.dir

Meta Data:

- Metastore, acts as a bridge b/w Hive and files in HDFS.
- A relational db with information on:
  - ✓ Databases, tables
  - ✓ Columns, owners, storage, serialization/de-serialization information
  - ✓ User supplied metadata

Complex data types in Hive:

- Array - Homogenous collection of elements, same data types
- Map - key-value pair, unique keys, unordered
- Struct - is like a class, logical grouping of elements of different data types

```
Ex. Student struct{
            Student_rollno
            Student_name
            Student_age
            Student_marks
}
```

Built-in Functions in Hive: basically divided in 3 categories

- UDF (user defined function)
  - ✧ One to one mapping
  - ✧ Works on single row, outputs a single row
  - ✧ Trim(), concat(), length(), round(), floor()

- UDAF (user defined aggregate function)
  - ✧ Many-one mapping
  - ✧ Works on multiple rows, outputs a single row
  - ✧ Count(*), sum(), avg()

- UDTF (user defined table generating function)
  - ✧ One-many mapping
  - ✧ Single row input, multiple row output
  - ✧ Explode()- flatten the data in arrays and maps, posexplode()-
  - ✧ Usecase: to break a complex data type into simpler datatype
  - ✧ Array to int
  - ✧ Ex. [1,2,3,4] --> 1, 2, 3, 4

Ex. Explode() a map

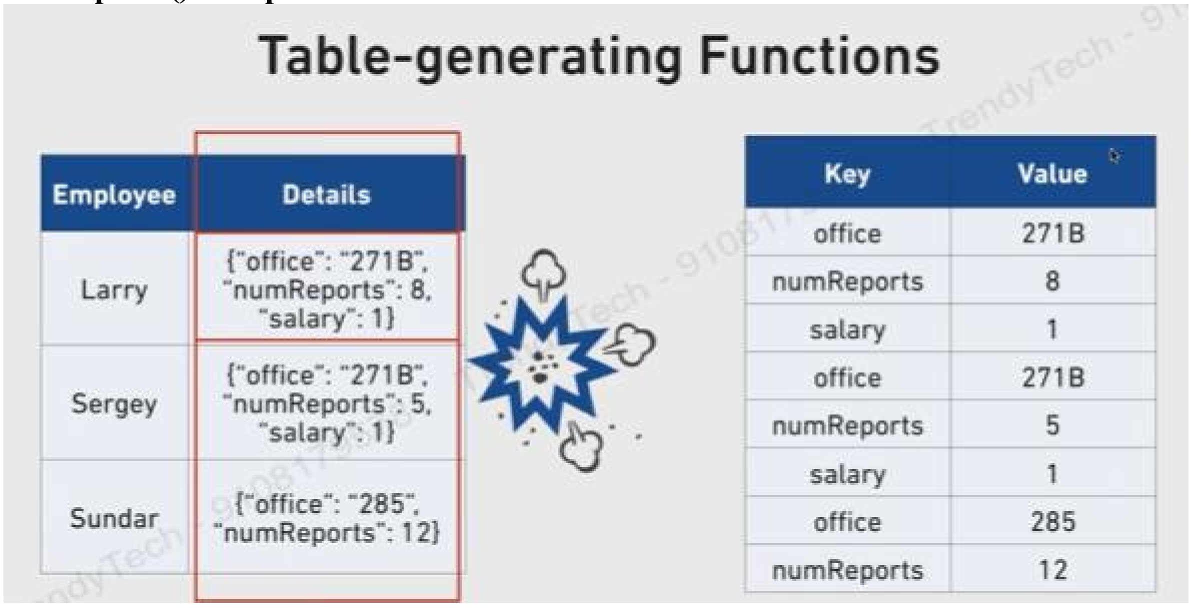


Table-generating Functions

Employee	Details
Larry	{"office": "271B", "numReports": 8, "salary": 1}
Sergey	{"office": "271B", "numReports": 5, "salary": 1}
Sundar	{"office": "285", "numReports": 12}

Key	Value
office	271B
numReports	8
salary	1
office	271B
numReports	5
salary	1
office	285
numReports	12

Lateral View:
When you explode a table you get a virtual table and then you that virtual table to original table.

SET operations in Hive
Union: combine results from 2 queries with the same column types, removes duplicates.
Union All: duplicates are also preserved.

For minus and intersect we can use joins.

Sub-queries in Hive:

- IN/NOT IN:
  - ✧ A query with an **IN** statement
  - ✧ A single list of values to check whether they match with the column values.
  - ✧ The sub-query should be for a **single** column value.
  - ✧ The sub-query should return a **list** of column values.
  - ✧ The sub-query **should not reference** the parent query.
  - ✧ Ex. Select id from customers
    where id **in** (select customer_id from orders);

- EXISTS/NOT EXISTS:
  - The sub-query **should be correlated** I.e. reference the parent query.
  - Ex. Select id from customers

    ```
    Where exists(
    Select customer_id form order
    Where orders.customer_id = customers.id );
    ```
  - It will give a list of customers who have placed at least one order.
  - If **not exists** , it will give me a list of customer ids who have not placed a single order.

VIEWS:

- **A virtual table with a subset of data from a larger, more extensive table.**
- Limited info is shown to a particular group of users.
- Data security
- Avoiding complexity of query:
  - Ex. You are asked to create a complex query with lots of joins and group by to get the desired data.
  - You can create a view on top of your complex query and give the view name to that person
  - And person says select * from view_name so internally that complex query will be run.
- We can create different logical paths from one big table.
- Syntax: **create view view_name as select query**

  Create view product_quantities as select product_id, quantity from orders

Advantages of view:

- ✓ Reduce query complexity
- ✓ Restrict access to data
- ✓ Construct different logical tables from the same physical table

A view is a metadata which is kept in metastore, coz view doesn't hold data by itself

---

Normalized storage in Traditional Databases

Normalization: It is the process of dividing a bigger table into multiple smaller tables with an intention to remove redundancy.

It is preferred in traditional databases.

Stages/types of normalization
1 NF
2 NF
3 NF
BCNF
4 NF

Advantages of normalization:

- Minimizes redundancy, optimizes storage
- Foreign keys to ensure valid joins
- Updates in one location, no duplication of data.

Disadvantages of normalization:

- To get details, you have to join multiple tables
- Joining is a very complex and time consuming activity.

In Hive: Denormalization is preferred.

Denormalization: data is kept in less tables to be read in a single operation. Store the data not in granular form rather bigger tables even if some redundancy is there.

Why denormalization:

- Disk space is very cheap
- No foreign key constraints
- Read operations, no data updates

Advantages:

- Optimize the number of disk seeks.
- Store data for an entity in one location
- Ignore redundancy, minimize joins
- Get all details about employee in one read operation

Theory :

- Do not worry about redundancy rather worry about storing the data in less number of tables.
- If we make updates, we can run into data inconsistency issues?
- Hive is not a system meant to do updates.
- Since we do not update the data we wont run into inconsistency issues.
- Storage space is very cheap and if we spend little more storage then its not a big deal.

Advantage to keep data in denormalized way in hive is that we do not have to operate the joins.
Our intention is to minimize the joins, coz joins take time.

PRACTICAL HIVE

Commands:

To enter hive	Hive
List of databases	Show databases;
Create db	Create database db_name;
Connect to a db	Use db_name;
Create a table	Create table customers(Id bigint, Name string, Address string);
List tables	Show tables;
Schema of tables	Describe tb_name;
Schema in detail	Describe formatted customers;
Insert a record	Insert into customers values (1111, “John”, “WA”),(2222, “Emily”, “WA”);
To check hive warehouse dir	Hadoop fs -ls /hive/user/warehouse
Display table data with where clause	Select * from customers where address = “WA”;
Map reduce job invoked in:	Distinct, order by, count(*), group by,
To exit form shell	Exit;

Note: inserts are not performed in hive. Insert will trigger a map reduce job in background.
Hive is not a transactional system.

Where exactly is the data stored by default?
A hive tables has 2 things:

- Data- is stored in hdfs (default location is /user/hive/warehouse)
- Metadata- is stores in a database why?
  Coz the schema has to be updated frequently and moreover quick access to this schema is required. Databases supports updations and also supports quick access that’s why its and ideal choice to keep the metadata. Ie low latency and edit options

---

Create a table with <if not exists> statement:
If the table with same name already exists, the above st. Wont do anything.
If the table does not exists, then it will create a new table.

```
Create table if  not exists orders(
Id bigint,
Product_id string,
Customer_id bigint,
Quantity int,
Amount double):
```

What are different ways to connect with hive?
1. Hive in terminal
2. Hue - Hadoop User Experience(UI)
3. Beeline (most preferred choice) - beeline gives you a better view also, it is secured.

Beeline:
a) To connect to hive: **beeline -u jdbc:hive2://**
b) To exit beeline: **!q**
c) Execute beeline query from terminal: **beeline -u jdbc:hive2:// -e "select * from trendytech.customers"**
The above command performs 3 things:
It will connect to beeline
It will execute the query "select * from trendytech.customers"
It will come out of beeline.

Run a beeline script from terminal: we will create a file with multiple hql queries and will try to run that file. The file is nothing but a script.
Steps:
- Create a gedit file using hql in /home/cloudera : **gedit myscript.hql**
- Enter following beeline queries inside myscript.hql and save:
  Show databases;
  Use trendytech;
  Select * from orders;
- Execute the beeline script file from terminal: below cmd has to be executed from terminal
  beeline -u jdbc:hive2:// -f /home/cloudera/myscript.hql
  -f is indicating we are executing a file.

Run beeline script file from beeline itself:
- Enter into beeline: **beeline -u jdbc:hive2://**
- Execute the beeline script: **source /home/cloudera/myscript.hql**

HIVE METASTORE:

The metadata for Hive tables are stored in the Hive Metastore.
By default, the Hive Metastore.db stores all Hive metadat in Derby database.
Derby is not recommended in production env as it only allows one connection at a time.
MySQL is the best choice for the metastore because it is the most popular among the Hive user community.

Hive warehouse directory:
In Hive, actual data is stores in directories under Hive's warehouse dir (usually in /user/hive/warehouse)
Display hive warehouse file structure: **hadoop fs -ls /user/hive/warehouse**

To access hdfs inside hive: dfs -ls /user/hive/warehouse/db_name;

Types of tables in Hive:
- Managed table -
  - When we create a table in Hive, by default it is Managed table.
  - Managed table is the one where hive manages the data also.
  - That means hive has the complete control of your data also.
  - Deleting(Drop) a managed tables deletes both the data and metadata.
  - To check type of table: **describe formatted table_name;**
- External table
- Temporary table (not imp)

How many are there to insert data in a table?

- ➢ Insert commands (these run mapreduce and takes lot of time. This is not an efficient way)
- ➢ Load data form one table to another.
- ➢ **Loading the data from files** (this is the most preferred or industry ready way).
  - ✧ Load the data from local file system.
  - ✧ Load the data from HDFS

In hive, create a table

```
create table if not exists products_managed(
   > id string,
   > title string,
   > cost float
   > ) row format delimited fields terminated by ',' stored as textfile;
```

Load data into managed table from a local path:
Command: *load data local inpath '/home/cloudera/Desktop/products.csv' into table products_managed;*

= the table we created is a managed table.
= we have not specified the path of data. That means data will be kept in default path (/user/hive/warehouse)
= *dfs -ls /user/hive/warehouse/trendytech.db/products_managed;*
= on loading from local, products.csv is copied from local to hdfs (/user/hive/warehouse).

Load data from HDFS folder to Hive table.
= We should have the folder created in HDFS.
= hadoop fs -ls /data
= hadoop fs -rm -r /data
= create new empty data folder
= *hadoop fs -put Desktop/products.csv /data*

Command: *load data inpath '/data/products.csv' into table products_managed;*
= in this case when we load from hdfs file.
= it will be a cut paste of data, move of data
= that means you wont be able to see the products.csv in /data folder in hdfs.
= coz of replication factor of 3 if there would be copy paste then total 6 copies would be there in total in both hdfs and hive which is not good. So move is there so that 3 copies default could be tthere only.
= it will be seen in /user/hive/warehouse/products_managed.
= the data is appended, and both files are present in hive warehouse dir.

Overwrite a Hive table: it will only show new content
Command: *load data local inpath*
> '/home/cloudera/Desktop/products.csv'
> overwrite into table products_managed;

Data loading using table to table method:
Create one more managed table:

```
create table if not exists prodcuts_managed2(
   > id string,
   > title string,
   > cost float)
   > row format delimited fields terminated by ',' stored as textfile;
```

Load data from existing table to the new table;

Insert into table prodcuts_managed2
*Select * from products_managed;*

When to go for managed table: when you know other technologies like spark doesn't depend on your data.
When to go for external table: when you know other tech are also dependent on your data in hive.

EXTERNAL TABLE:

- Data not fully managed by Hive.
- Share the underlying data across other applications(Hadoop, Pig, HBase, spark, etc)
- Dropping an external table deleted only metadata and not the data.

Creating an external table:

```
create external table products(
   > id string,
   > title string,
   > cost float
   > )
   > location '/data/';
```

Create an external table with delimiters:

```
create external table if not exists products(
   > id string,
   > title string,
   > cost float
   > )
   > row format delimited fields terminated by ','
   > stored as textfile         <--------- hive by default takes textfile format, we can skip this line also
   > location '/data/';
```

Hive Subqueries Views and Index:

Subqueries in Hive:

- Subqueries are queries which return a result set which are nested within other queries.
- Subqueries in Hive can be used for:
  - ✧ FROM clause
  - ✧ WHERE clause

Create a table names *products* inside trendytech db:

```
create table products2(
> id string,
> title string,
> cost float
> )
> row format delimited fields terminated by ',' stored as textfile;
```

Load data into freshproducts table;

```
load data local inpath
 > '/home/cloudera/Desktop/freshproducts.csv'
 > into table freshproducts;
```

Display records using Subqueries using FROM clause:

```
select *
> FROM(
> select id as product_id from products2
> UNION ALL
> select id as product_id from freshproducts
> ) t;
```

2nd example:

```
select distinct(t.product_id)
>from (select product_id from customers join orders where customers.id=orders.customer_id ) t;
```

The above query gives a list of products which are atleast purchased once.
Hive supports 2 types of subqueries in WHERE clause:

- IN/NOT IN
- EXISTS/NOT EXISTS

IN:

```
select name from customers
> where customers.id IN
> (
> select customer_id from orders);
```

The above query gives the name of customers who have done atleast one purchase.
NOT IN:

```
select name from customers
> where customers.id NOT IN
> (
> select customer_id from orders);
```

The above query gives the name of customers who have never made a purchase.

EXISTS:

```
Select id from customers
>where EXISTS
>(select customer_id from orders where orders.customer_id = customer.id );
```

NON EXISTS:

```
Select id from customers
>where not exists
>( select customer_id from orders
  Where orders.customer_id = customers.id );
```

VIEWS IN HIVE

- A view is a virtual table, which provides access to a subset of data from one or more table.
- Stored as a query in Hive's metastore.
- Executed when used.
- Updated when data in the underlying table changes.
- Contains data from single or multiple tables.
- Frozen in time, not affected by table changes.

Steps:

- Before creating a view describe the *customers* and *orders* table to display schema information:
  Describe orders;
  describe customers;
- create a view on customers and orders table;
  create view customer_purchase as
  > select customer_id, product_id, address from customers join orders
  > where customers.id = orders.customer_id;
- To display the view (virtual) table;
  Show tables;
- Run select query to display records of a view:
  *select * from customer_purchase;*

HIVE INDEX:

- Hive index is used to speed up the performance of queries on certain columns of a table.
- Without an index, queries with predicates like 'WHERE tab1.col1 = 10' load the entire table or partition and process all the rows. But if an index exists for col1, then only a portion of the file needs to be loaded and processed.
- Indexing can be used:
  - When the data set is very large.
  - When the query execution is taking more amount of time than expected.
  - When a fast query execution is required.
- Note: Indexing is removed since Hive version 3.0

- Create an index on customer_id column of orders table;

```
Create index orders_index
On table orders(customer_id)
As 'COMPACT' WITH DEFERRED REBUILD;
```

- Here, COMPACT means we are creating a compact index for the table.
- the WITH DEFERRED REBUILD st. Is because we need to alter the index in later stages using this st.
- Show the index which we have created on orders table:

```
Show indexes on orders;
```

- Show table display index table:

```
Show tables;
```

- Drop an index;

```
Drop index orders_index on ORDERS;
```

EXPLODE & LATERAL VIEW:

The system defined function in hive fall in 3 categories.

- UDF - one line as input and one line as output - length
- UDAF - multiple input lines and one output row - max, min, sum
- UDTF - one input and multiple output - **explode** function

UDTF - user defined table generating function

- create a table with an array column

```
Use trendytech;
create table author_details
(author_name string,
book_names array<string>)
row format delimited fields terminated by ','
collection items terminated by ":";
```

- Load data:

```
load data local inpath '/home/cloudera/Desktop/authors.csv'
into table author_details;
```

- select * from author_details;
- **Select explode(book_names) from author_details;**

Limitation :

Try selecting other columns with exploded ones:

```
Select author_name, explode(book_names) from author_details;
```

This will error out.

Solution:
We need to use **Lateral View**. The exploded column should be made as virtual table and should be joined with actual table.

```
select author_name, b_name from author_details lateral view
explode(book_names) book_table as b_name;
```

This means:

Table1	author_details
Join	lateral_view
Table2	explode(book_names)

Alias name of table2 is book_table
B_name is the column name in vir tual table (book_table)

COMPLEX DATA TYPES IN HIVE:

Hive has support for 4 types of complex data:

- ✓ Array
- ✓ Map
- ✓ Struct
- ✓ Union (rarely used coz of incomplete support in Hive)

ARRAY:
Collection of items of same data type. An array can contain one or more values of the same data type.

- ➢ Create a table with array data types:

```
create table mobilephones
   > ( id string,
   > title string,
   > cost float,
   > colors array<string>,
   > screen_size array<float>);
```

- ➢ Load data into the array data type table:

```
insert into table mobilephones
select "redminote7", "Redmi Note 7", 300, array("white", "silver", "black"),
array(float(4.5))
union all
select "motoGplus", "Moto G Plus", 200, array("black", "gold"),
array(float(4.5), float(5.5));
```

- ➢ Select id, colors from mobilephones;
- ➢ First element of array:

```
select id, colors[0] from mobilephones;
```

- ➢ Create a new table with array data type with same schema and load data using a file.

```
create table mobilephones_new
( id string,
title string,
cost float,
colors array<string>,
screen_size array<float>)
row format delimited fields terminated by ','
collection items terminated by '#';
```

Note:
My file has 7 columns
However my table has just 5 columns
What will happen if I load this file into this table.
It will consider only the first 5 columns of the file.
And it will ignore the last 2 columns.

➢ Load data to the above table:
```
 load data local inpath '/home/cloudera/Desktop/mobilephones.csv'
   > into table mobilephones_new;
```

MAP:

- ✓ The Map is a collection of key-value pairs or unordered collection of pairs.
- ✓ Map has no fixed size.
- ✓ The value is accessed using a unique key.
- ✓ Keys and values have their own data types.

➢ Drop the mobilephones table which we have created earlier:
```
drop table mobilephones;
```
➢ Create new table:
```
      create table mobilephones (
         > id string,
         > title string,
         > cost float,
         > colors array<string>,
         > screen_size array<float>,
         > features map<string, boolean>
         > )row format delimited fields terminated by ','
         > collection items terminated by '#'
         > map keys terminated by ':';
```
➢ Load data:
```
     load data local inpath '/home/cloudera/Desktop/mobilephones.csv'
     into table mobilephones;
```
➢ Select id, features[‘camera’] from mobileophones;

STRUCT:

- ✓ Struct logically groups different types of data together into one entity.
- ✓ Struct is like a class
- ✓ Ex. Student struct
  - Student_id
  - Student_name
  - Student_marks
- ✓ Each individual bit of data within a struct can have different data types.
- ✓ Struct can hold any number of values.
- ✓ Each value referenced by a name.

- Drop the mobilephones table which we have created earlier:
  drop table mobilephones;
- Create a new tables:

```
create table mobilephones(
id string,
title string,
cost float,
colors array<string>,
screen_size array<float>,
features map<string, boolean>,
information struct<battery:string,camera:string>
)
row format delimited fields terminated by ','
collection items terminated by '#'
map keys terminated by ':';
```

- Display individual records:

```
Select id, features[‘camera’], information.battery from mobilephones;
```

User Defined Functions (UDF) in Hive:

Hive already has lot of in-build functions.
What will you do if none of the system defined functions fits your need.
Write your own user defined function (UDF).

Scenario:
Consider you have a table with 2 columns name and count.
Name(string)
Count(int)
We want to convert all the names to upper case.
We will be writing a UDF to convert all the names to uppercase

Solution:
Create a new java project (udf_example) in eclipse.
Project name: udf_example
Package name: udf_example
Class name: DataStandardization

Copy the given code in this DataStandardization.java file

```
package udf_example;
import org.apache.hadoop.hive.ql.exec.UDF;
public class DataStandardization extends UDF {
        public String evaluate(String input){
                if(input==null)
                {
                        return null;
                }
                return (input.toString().toUpperCase());
                }
        }
```

Download the necessary jar
https://mvnrepository.com/artifact/org.apache.hive/hive-exec/1.2.2
Add these jar files and now errors will be gone.

Now, in hive create 2 tables and load data

```
create table if not exists sample_table
   > (name string,
   > count int)
   > row format delimited fields terminated by ','
   > lines terminated by '\n'
   > stored as textfile;
```

Now load data:

```
load data local inpath '/home/cloudera/Desktop/sample_data.txt'
   > overwrite into table sample_table;
```

Export the jar file for the java project

Right click project -> export -> jar file
With this give the desktop path and jar will be created on desktop.

In hive, add the jar & create temporary function:

```
ADD JAR /home/cloudera/Desktop/udf_example.jar;
Create temporary function standardize as ‘udf_example.DataStandardization’;
```

Here, standardize ==> function name.
‘udf_example.DataStandardization’ ==> ‘package_name.class_name’

Now we are all set, try calling this UDF on a column in a table:

```
Select standardization(name) from sample_table;
```

What’s the issue?

These are temporary things. This jar & function is valid only for current session. That means if you come out of hive and go back in, then you won’t be able to call this function.

Solution? ---> Create a permanent function.

For this we need to have our jars in HDFS.
Let us transfer the jars to HDFS.

In normal terminal:

```
hadoop fs -put udf_example.jar /user/cloudera
```

We need to create a permanent function:

```
CREATE FUNCTION standardize_permanent AS
   > 'udf_example.DataStandardization' using JAR
   > 'hdfs://localhost:8020/user/cloudera/udf_example.jar';
```

Try calling the function now:

```
select standardize_permanent(name) from sample_table;
```