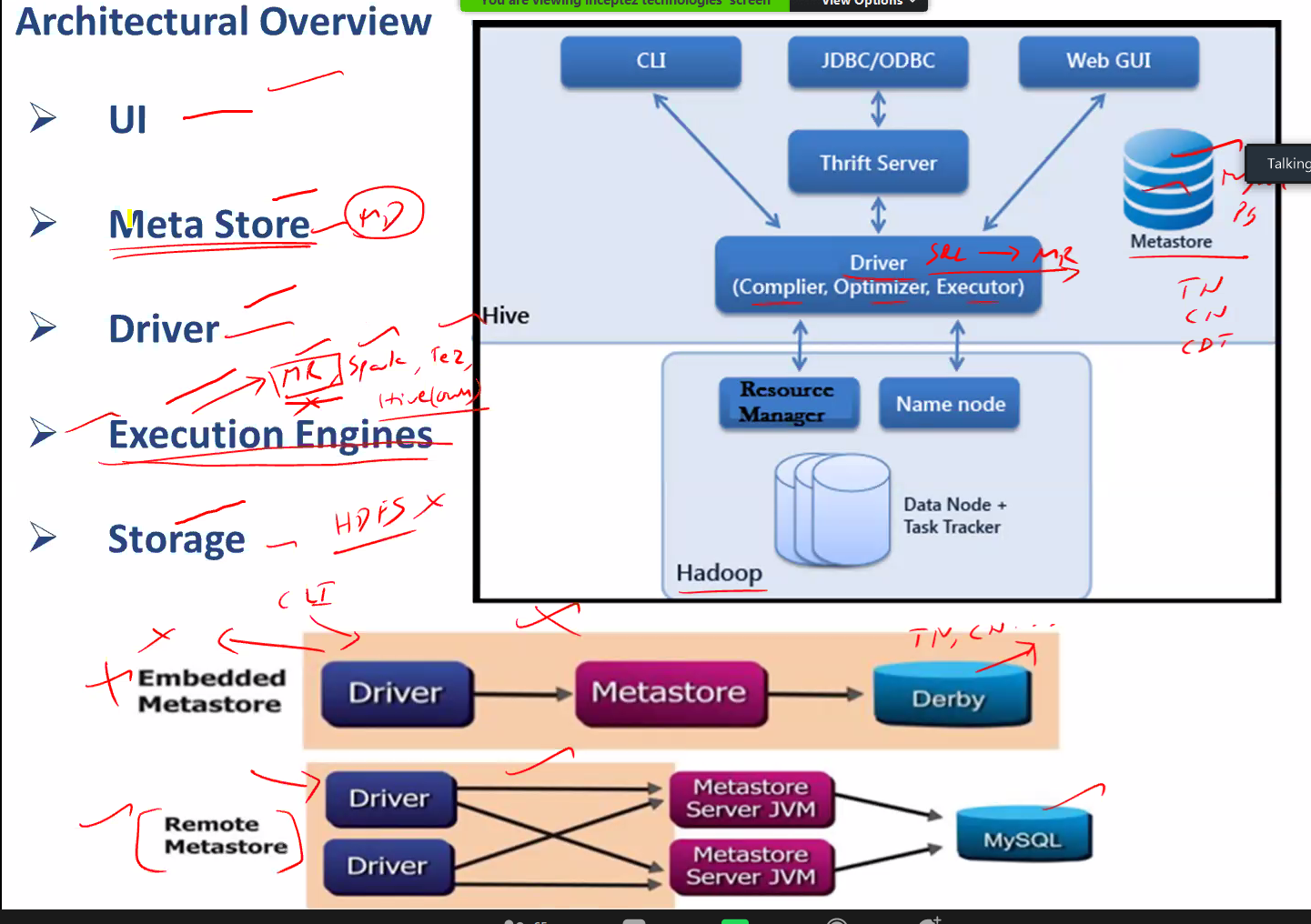
# HIVE

Hive is SQL layer on top of Hadoop. Its called HQL, Hive Query Language.

Hive Architecture:



### Metastore:

Metastore is required to store the Hadoop table structure information. It cannot be saved in Hadoop itself.

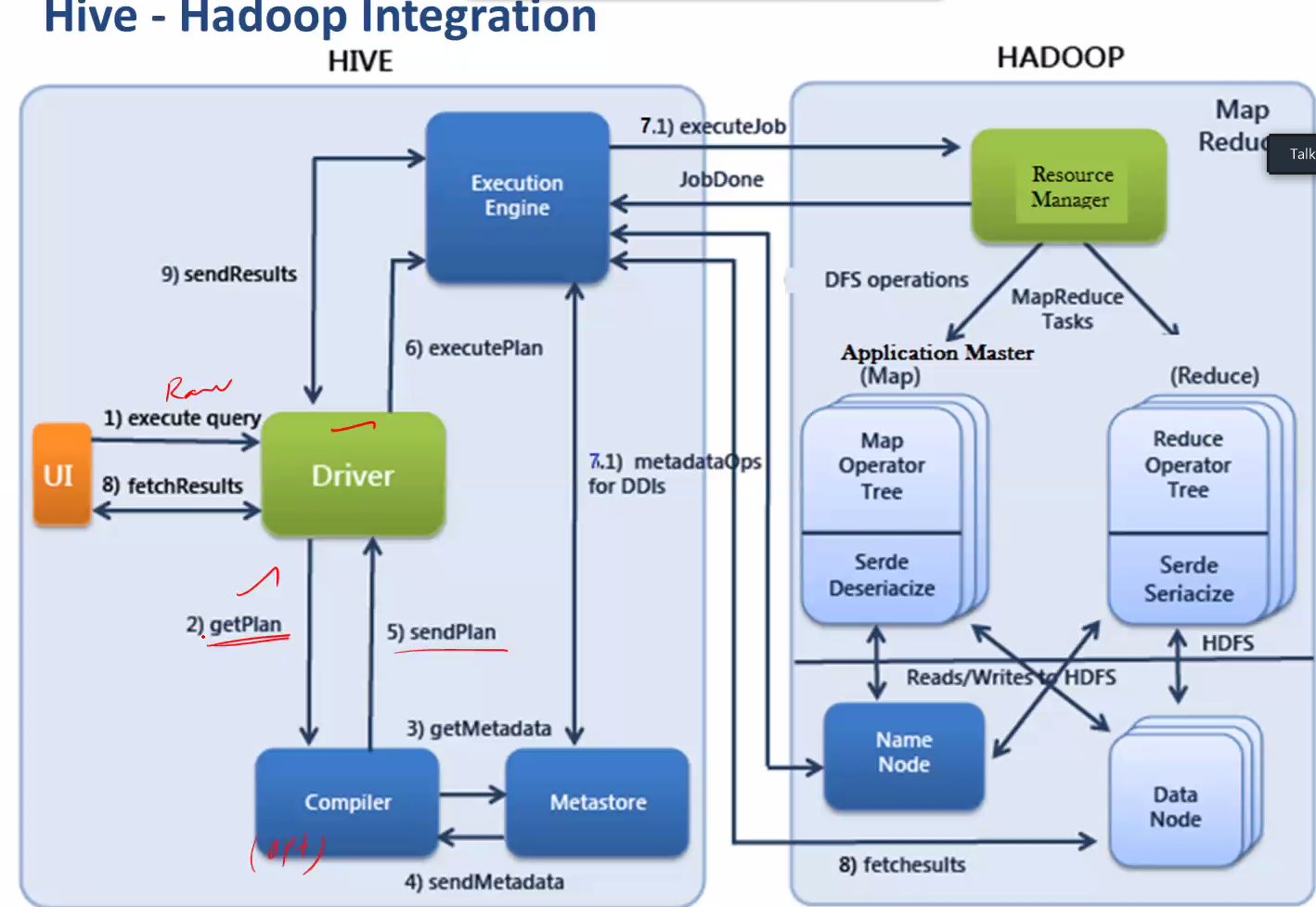
#### Embedded metastore:

Comes default with Hadoop, which contains Derby. Derby has limitation which can be used for Development and testing alone. No two sessions can be started at same time.

#### Metastore:

Hive supports many databases to store the metadata of table information. The metastore server jvm allows to start session with Hive

## Hive – Hadoop integration



5. Execution plan is generated, the best way to reach the database

6. Submit the plan to execution engire

7. Execute the job converting the job in to RM or Spark or Tez depending on the execution engine

NOTE: Some of the Queries can be directly fetched from Hadoop without involving any Hadoop execution engine. Hive to Hadoop directly. No use of RM, Spark or Tez

# SerDes - Serialization and Deserialization

Serialization: Converting the original data to byte code

Deserialization: Converting the byte code to original data

What is a SerDe?

* SerDe is a short name for "Serializer and Deserializer."
* Hive uses SerDe (and FileFormat) to read and write table rows.
* HDFS files --> InputFileFormat --> <key, value> --> Deserializer --> Row object
* Row object --> Serializer --> <key, value> --> OutputFileFormat --> HDFS files

# Hive Workout:

Create database:

CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database\_name

  [COMMENT database\_comment]

  [LOCATION hdfs\_path]

  [MANAGEDLOCATION hdfs\_path]

  [WITH DBPROPERTIES (property\_name=property\_value, ...)];

From above ( ) – is a mandatory parameter

From above [] – is a optional parameter

DATABASE|SCHEMA: These are the same thing. These words can be used interchangeably.

[IF NOT EXISTS]: This is an optional clause. If not used, an error is thrown when there is an attempt to create a database that already exists.

[COMMENT]: This is an optional clause. This is used to place a comment for the database. This comment clause can be used to add a description about the database. The comment must be in single quotes.

[LOCATION]: This is an optional clause. This is used to override the default location with the preferred one.

[WITH DBPROPERTIES]: This is an optional clause. This clause is used to set properties for the database. These properties are key-value pairs that can be associated with the database to attach additional information with the database.

To get the database details:

DESCRIBE DATABASE [EXTENDED] db\_name;

### Hive Data Hierarchy

Complex Data Type:

* ARRAY - This is a list of items of the same type - ["apple","orange","mango"]
* MAP - This is a set of key-value pairs - {1: "apple",2: "orange"}
* STRUCT - This is a user-defined structure of any type of field, such as {val1, val2, val3, and so on} - {1, "apple"}

We have employee database:

Michael|Montreal,Toronto|Male,30|DB:80|Product:Developer:Lead

Will|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead

Shelley|New York|Female,27|Python:80|Test:Lead,COE:Architect

Lucy|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead

Create table Employee:

CREATE TABLE employee (

name STRING,

work\_place ARRAY<STRING>,

gender\_age STRUCT<gender:STRING,age:INT>,

skills\_score MAP<STRING,INT>,

depart\_title MAP<STRING,ARRAY<STRING>>

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '|'

COLLECTION ITEMS TERMINATED BY ','

MAP KEYS TERMINATED BY ':'

STORED AS TEXTFILE;

Select \* from table looks like

employee.name employee.work\_place employee.gender\_age employee.skills\_score employee.depart\_title

Michael ["Montreal","Toronto"] {"gender":"Male","age":30} {"DB":80} {"Product":["Developer:Lead"]}

Will ["Montreal"] {"gender":"Male","age":35} {"Perl":85} {"Product":["Lead"],"Test":["Lead"]}

Shelley ["New York"] {"gender":"Female","age":27} {"Python":80} {"Test":["Lead"],"COE":["Architect"]}

Lucy ["Vancouver"] {"gender":"Female","age":57} {"Sales":89,"HR":94} {"Sales":["Lead"]}

#### How to access Array:

work\_place

["Montreal","Toronto"]

["Montreal"]

["New York"]

["Vancouver"]

hive (retail)> SELECT

> work\_place[0] as col\_1, work\_place[1] as col\_2,

> work\_place[2] as col\_3

> FROM employee;

col\_1 col\_2 col\_3

Montreal Toronto NULL

Montreal NULL NULL

New York NULL NULL

Vancouver NULL NULL

#### How to access Struct:

hive (retail)> SELECT gender\_age FROM employee;

gender\_age

{"gender":"Male","age":30}

{"gender":"Male","age":35}

{"gender":"Female","age":27}

{"gender":"Female","age":57}

Access individual items:

hive (retail)> SELECT gender\_age.gender, gender\_age.age FROM employee;

gender age

Male 30

Male 35

Female 27

Female 57

How to access Map items:

hive> SELECT skills\_score FROM employee;

{"DB":80}

{"Perl":85}

{"Python":80}

{"Sales":89,"HR":94}

How to access individual items:

hive> SELECT

> name, skills\_score['DB'] as DB, skills\_score['Perl'] as Perl,

> skills\_score['Python'] as Python,

> skills\_score['Sales'] as Sales,

> skills\_score['HR'] as HR

> FROM employee;

Michael 80 NULL NULL NULL NULL

Will NULL 85 NULL NULL NULL

Shelley NULL NULL 80 NULL NULL

Lucy NULL NULL NULL 89 94

### Database:

The database in Hive describes a collection of tables. If the database is not specified, the **default** database is used and **uses /user/hive/warehouse** in HDFS as its root directory. This path is configurable by the hive.metastore.warehouse.dir property in hive-site.xml. Whenever a new database is created, Hive creates a new directory for each database under /user/hive/warehouse. For example, the myhivebook database is located at /user/hive/datawarehouse/myhivebook.db. In addition, DATABASE has a name alias, SCHEMA, meaning they are the same thing in HQL

#### Create Database:

CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database\_name

  [COMMENT database\_comment]

  [LOCATION hdfs\_path]

  [MANAGEDLOCATION hdfs\_path]

  [WITH DBPROPERTIES (property\_name=property\_value, ...)];

#### Drop database:

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

* A database cannot be dropped until all the tables inside it are empty, unless CASCADE is used

#### Alter Database:

ALTER (DATABASE|SCHEMA) database\_name SET LOCATION hdfs\_path;

### Tables:

#### Managed Table:

1. All the data in the table is stored in this hive user-manageable directory (full permission)
2. When keeping data in the internal tables, the table fully manages the data in it. When an internal table is dropped, its data is deleted together
3. the internal table is often used as an intermediate table during data processing, since it is quite powerful and flexible when supported by HQL.

#### External Table:

1. When data is already stored in HDFS, an external table can be created to describe the data. It is called external because the data in the external table is specified in the LOCATION property
2. when an external table is dropped, the data is not deleted
3. use external tables for source read-only data or sharing the processed data to data consumers giving customized HDFS locations

#### Temporary Table:

Hive also supports creating temporary tables. A temporary table is only visible to the current user session. It's automatically deleted at the end of the session. The data of the temporary table is stored in the user's scratch directory, such as /tmp/hive-<username>

[hduser@Inceptez ~]$ hdfs dfs -cat /tmp/hive/hduser/47d5f037-d988-4f3c-974c-0d2f1b8ce7ef/\_tmp\_space.db/d49720fd-b86a-48b1-a6d1-57801d5fad98/000000\_0

20/09/21 01:07:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

khadhar

#### Create-Table-As-Select:

Tables can also be created and populated by the results of a query in one statement, called Create-Table-As-Select (CTAS). The table created by CTAS is not visible by other users until all the query results are populated.

CTAS has the following restrictions:

* The table created cannot be a partitioned table
* The table created cannot be an external table
* The table created cannot be a list-bucketing table

A CTAS statement always triggers a yarn job to populate the data, although the SELECT \* statement itself does not trigger any yarn job.

#### Create Empty table – Copy only Schema

1. Using CTAS with wrong Where condition

hive> create table employee\_empty as select \* from employee where 1=2;

-rw-r--r-- 1 hduser supergroup 0 2020-09-21 01:37 /user/hive/warehouse/retail.db/employee\_empty/000000\_0

1. With Like statement

hive> CREATE TABLE empty\_like\_employee LIKE employee\_internal;

drwxr-xr-x - hduser supergroup 0 2020-09-21 01:41 /user/hive/warehouse/retail.db/empty\_like\_employee

### DDL Statement:

CREATE TABLE IF NOT EXISTS employee\_internal (

name STRING COMMENT 'this is optinal column comments',

work\_place ARRAY<STRING>, -- table column names are NOT casesensitive

gender\_age STRUCT<gender:STRING,age:INT>,

skills\_score MAP<STRING,INT>, -- columns names are lower case

depart\_title MAP<STRING,ARRAY<STRING>> -- No "," for the last column

)

COMMENT 'This is an internal table' -- This is optional table comments

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '|' -- Symbol to seperate columns

COLLECTION ITEMS TERMINATED BY ',' -- Seperate collection elements

MAP KEYS TERMINATED BY ':' -- Symbol to seperate keys and values

STORED as TEXTFILE; -- Table file format

To view details description of table – command “Describe formatted <<table name>>”

hive> describe formatted employee\_internal;

OK

# col\_name data\_type comment

name string this is optinal column comments

work\_place array<string>

gender\_age struct<gender:string,age:int>

skills\_score map<string,int>

depart\_title map<string,array<string>>

# Detailed Table Information

Database: retail

Owner: hduser

CreateTime: Mon Sep 21 00:19:16 IST 2020

LastAccessTime: UNKNOWN

Protect Mode: None

Retention: 0

Location: hdfs://localhost:54310/user/hive/warehouse/retail.db/employee\_internal

Table Type: MANAGED\_TABLE

Table Parameters:

comment This is an internal table

transient\_lastDdlTime 1600627756

# Storage Information

SerDe Library: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

InputFormat: org.apache.hadoop.mapred.TextInputFormat

OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

Compressed: No

Num Buckets: -1

Bucket Columns: []

Sort Columns: []

Storage Desc Params:

colelction.delim ,

field.delim |

mapkey.delim :

serialization.format |

#### Get Create statement:

hive> show create table employee;

CREATE TABLE `employee`(

`name` string,

`work\_place` array<string>,

`gender\_age` struct<gender:string,age:int>,

`skills\_score` map<string,int>,

`depart\_title` map<string,array<string>>)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '|'

COLLECTION ITEMS TERMINATED BY ','

MAP KEYS TERMINATED BY ':'

STORED AS INPUTFORMAT

'org.apache.hadoop.mapred.TextInputFormat'

OUTPUTFORMAT

'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

LOCATION

'hdfs://localhost:54310/user/hive/warehouse/retail.db/employee'

TBLPROPERTIES (

'COLUMN\_STATS\_ACCURATE'='true',

'numFiles'='1',

'numRows'='0',

'rawDataSize'='0',

'totalSize'='227',

'transient\_lastDdlTime'='1600624052')

### Load Data:

Load from local

hive> LOAD DATA LOCAL INPATH '/home/hduser/employee.txt' overwrite INTO TABLE employee\_internal;

Loading data to table retail.employee\_internal

Load from hdfs

hive> LOAD DATA inpath '/user/hduser/employee\_internal.txt' overwrite INTO TABLE employee\_internal;

Loading data to table retail.employee\_internal

Table retail.employee\_internal stats: [numFiles=1, numRows=0, totalSize=227, rawDataSize=0]

#### Delete data from table:

##### Drop Table:

The drop-table statement on an internal table removes the table completely

##### Truncate Table:

The truncate table statement only removes data from the table. The table still exists, but is empty. Note, truncate table can only apply to an internal table

hive> select \* from employee1;

khadhar basha

Time taken: 0.364 seconds, Fetched: 1 row(s)

hive> select \* from employee2;

khadhar basha

Time taken: 0.302 seconds, Fetched: 1 row(s)

hive> truncate employee1;

FAILED: ParseException line 1:9 missing TABLE at 'employee1' near '<EOF>'

hive> truncate table employee1;

Time taken: 4.566 seconds

hive> drop table employee2;

Time taken: 6.487 seconds

hive> select \* from employee1;

Time taken: 0.549 seconds

hive> select \* from employee2

FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'employee2'

### Indexes:

Unlike any relational database, indexes can be created in Hive also. For better and fast accessing of data.

Hive indexes stores the pair of the indexed column's value and its block ID

#### Record:

> select \* from txnrecords where txnno=95903;

txnrecords.txnno txnrecords.txndate txnrecords.custno txnrecords.amount txnrecords.category txnrecords.product txnrecords.city txnrecords.state txnrecords.spendby

95903 09-05-2011 4005514 52.82 Jumping Pogo Sticks Scottsdale Arizona credit

#### Index:

95903hdfs://localhost:54310/user/hive/warehouse/retail.db/txnrecords/txns8471992

Index creation:

CREATE INDEX idx\_id\_employee\_id

ON TABLE employee\_id (employee\_id)

AS 'COMPACT'

WITH DEFERRED REBUILD;

The WITH DEFERRED REBUILD option in this example prevents the index from immediately being built. To build the index, we can issue the ALTER...REBUILD

ALTER INDEX idx\_id\_employee\_id ON employee\_id REBUILD;

#### Drop index:

DROP INDEX index\_name ON table\_name

## Partitions

Eeach partition corresponds to a predefined partition column(s), which maps to subdirectories in the table's directory in HDFS. When the table gets queried, only the required partitions (directory) of data in the table are being read, so the I/O and time of the query is greatly reduced. Using partition is a very easy and effective way to improve performance in Hive.

### Static Partition:

Static partition is often used for an external table containing data newly landed in HDFS. In this case, it often uses the date, such as yyyyMMdd, as the partition column. Whenever the data of the new day arrives, we add the day-specific static partition (by script) to the table, and then the newly arrived data is queryable from the table immediately

Steps to create Partitions:

* Insert input data files individually into a partition table is Static Partition.
* Usually when loading files (big files) into Hive tables static partitions are preferred.
* Static Partition saves your time in loading data compared to dynamic partition.
* You “statically” add a partition in the table and move the file into the partition of the table.
* You can perform Static partition on Hive Manage table or external table.

How to create:

hive> CREATE External TABLE employee\_partitioned (

> name STRING,

> work\_place ARRAY<STRING>,

> gender\_age STRUCT<gender:STRING,age:INT>,

> skills\_score MAP<STRING,INT>,

> depart\_title MAP<STRING,ARRAY<STRING>>

> )

> PARTITIONED BY (year INT, month INT)

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY '|'

> COLLECTION ITEMS TERMINATED BY ','

> MAP KEYS TERMINATED BY ':';

!! Partitions are not added at the time of table creation 🡪 But only columns names are specified. No folders are created

hive> show partitions employee\_partitioned;

partition <<empty>>

##### Partition creation Type1:

1. Partitions can be added using ALTER command, which creates a folder inside hdfs folder
2. Place the data inside the folder
3. Query the table with where condition

ALTER TABLE

hive> ALTER TABLE employee\_partitioned ADD

> PARTITION (year=2018, month=11) PARTITION (year=2018,

> month=12);

Partition folders are created in HDFS

20/09/21 05:52:18 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

drwxr-xr-x - hduser supergroup 0 2020-09-21 05:50 /user/hive/warehouse/retail.db/employee\_partitioned/year=2018/month=11

drwxr-xr-x - hduser supergroup 0 2020-09-21 05:50 /user/hive/warehouse/retail.db/employee\_partitioned/year=2018/month=12

hive> show partitions employee\_partitioned;

OK

year=2018/month=11

year=2018/month=12

copy the data into the HDFS location

[hduser@Inceptez ~]$ hdfs dfs -put employee.txt /user/hive/warehouse/retail.db/employee\_partitioned/year=2018/month=11

Now, select \* from table where year=2018

hive> select \* from employee\_partitioned where year=2018;

Michael ["Montreal","Toronto"] {"gender":"Male","age":30} {"DB":80} {"Product":["Developer:Lead"]} 2018 11

Will ["Montreal"] {"gender":"Male","age":35} {"Perl":85} {"Product":["Lead"],"Test":["Lead"]} 2018 11

Shelley ["New York"] {"gender":"Female","age":27} {"Python":80} {"Test":["Lead"],"COE":["Architect"]} 2018 11

Lucy ["Vancouver"] {"gender":"Female","age":57} {"Sales":89,"HR":94} {"Sales":["Lead"]} 2018 11

Data is automatically picked from partitions

##### Type2:

Create partitions while loading the data using LOAD + partition command

hive> load data local inpath '/home/hduser/employee.txt' overwrite into table employee\_partitioned partition(year=2020, month=2);

Loading data to table retail.employee\_partitioned partition (year=2020, month=2)

Partition retail.employee\_partitioned{year=2020, month=2} stats: [numFiles=1, numRows=0, totalSize=227, rawDataSize=0]

Partition created

hive> show partitions employee\_partitioned;

year=2018/month=11

year=2018/month=12

year=2020/month=2

Files in HDFS created with load command

drwxr-xr-x - hduser supergroup 0 2020-09-21 06:14 /user/hive/warehouse/retail.db/employee\_partitioned/year=2020

drwxr-xr-x - hduser supergroup 0 2020-09-21 06:14 /user/hive/warehouse/retail.db/employee\_partitioned/year=2020/month=2

-rw-r--r-- 1 hduser supergroup 227 2020-09-21 06:14 /user/hive/warehouse/retail.db/employee\_partitioned/year=2020/month=2/employee.txt

#### Drop partition:

DROP partition on a MANAGED table will purge the data from HDFS

DROP partition on a EXTERNAL table will not removed the data or sub folders

<< We are NOT able to change a non-partition table to a partition table directly >>

### Dynamic Partitions:

To avoid manually adding static partitions, dynamic partition insert (or multipartition insert) is designed for dynamically determining which partitions should be added and populated while scanning the input table

When inserting data into the partitions, we need to specify the partition columns. Instead of specifying static partition values, Hive also supports dynamically giving partition values. Dynamic partitions are useful when it is necessary to populate partitions dynamically from data values. Dynamic partitions are disabled by default because a careless dynamic partition insert could create many partitions unexpectedly. We have to set the following properties to enable dynamic partitions:

SET hive.exec.dynamic.partition=true;

* Dynamic partitions can be only loaded using insert select
* Both managed and external table can be loaded with partitions

hive> Insert into table managedtxnrecsbygames partition (category)

> select txnno,txndate,custno,amount, product,city,state,spendby,category

> from txnrecords;

Loading data to table retail.managedtxnrecsbygames partition (category=null)

Time taken for load dynamic partitions : 10277

Loading partition {category=Gymnastics}

Loading partition {category=Dancing}

Loading partition {category=Water Sports}

Loading partition {category=Games}

Loading partition {category=Jumping}

Loading partition {category=Outdoor Recreation}

Loading partition {category=Exercise & Fitness}

Loading partition {category=Team Sports}

Loading partition {category=Outdoor Play Equipment}

Loading partition {category=Racquet Sports}

Loading partition {category=Combat Sports}

Loading partition {category=Puzzles}

Loading partition {category=Indoor Games}

Loading partition {category=Winter Sports}

Loading partition {category=Air Sports}

Partitions are created in meta data

hive> show partitions managedtxnrecsbygames;

category=Air Sports

category=Combat Sports

category=Dancing

category=Exercise & Fitness

category=Games

category=Gymnastics

category=Indoor Games

category=Jumping

category=Outdoor Play Equipment

category=Outdoor Recreation

category=Puzzles

category=Racquet Sports

category=Team Sports

category=Water Sports

category=Winter Sports

HDFS Folders and Sub folders are created respectively

[hduser@Inceptez ~]$ hdfs dfs -ls -R /user/hive/warehouse/retail.db/managedtxnrecsbygames/

20/09/21 06:58:37 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

drwxr-xr-x - hduser supergroup 0 2020-09-21 06:50 /user/hive/warehouse/retail.db/managedtxnrecsbygames/category=Air Sports

-rw-r--r-- 1 hduser supergroup 132406 2020-09-21 06:50 /user/hive/warehouse/retail.db/managedtxnrecsbygames/category=Air Sports/000000\_0

drwxr-xr-x - hduser supergroup 0 2020-09-21 06:50 /user/hive/warehouse/retail.db/managedtxnrecsbygames/category=Combat Sports

-rw-r--r-- 1 hduser supergroup 208453 2020-09-21 06:50 /user/hive/warehouse/retail.db/managedtxnrecsbygames/category=Combat Sports/000000\_0

drwxr-xr-x - hduser supergroup 0 2020-09-21 06:50 /user/hive/warehouse/retail.db/managedtxnrecsbygames/category=Dancing

-rw-r--r-- 1 hduser supergroup 54574 2020-09-21 06:50 /user/hive/warehouse/retail.db/managedtxnrecsbygames/category=Dancing/000000\_0

drwxr-xr-x - hduser supergroup 0 2020-09-21 06:50 /user/hive/warehouse/retail.db/managedtxnrecsbygames/category=Exercise & Fitness

-rw-r--r-- 1 hduser supergroup 1046637 2020-09-21 06:50 /user/hive/warehouse/retail.db/managedtxnrecsbygames/category=Exercise & Fitness/000000\_0

drwxr-xr-x - hduser supergroup 0 2020-09-21 06:50

#### Remove partitions:

-- For internal table, we use truncate

> TRUNCATE TABLE employee\_partitioned PARTITION (year=2018,month=12);

-- For external table, we have to use hdfs command

> dfs -rm -r -f /user/dayongd/employee\_partitioned;

## Buckets:

Bucketing is a technique that allows you to decompose your data into more manageable parts, that is, fix the number of buckets. Usually, partitioning provides a way of segregating the data of a Hive table into multiple files or directories. Partitioning is used to increase the performance of queries, but the partitioning technique is efficient only if there is a limited number of partitions

In bucketing, we specify the fixed number of buckets in which entire data is to be decomposed. Bucketing concept is based on the hashing principle, where same type of keys are always sent to the same bucket

For Eg: If CLUSTERED STATE INTO 2 BUCKETS

State = TN,AP,KA,TE,MP

If hash value of TN and AP = 1 and others are 2

Then TN and AP always goes to same BUCKET, other states will go to same BUCKET

##### Syntax:

hive> CREATE TABLE employee\_id\_buckets (

> name STRING,

> employee\_id INT, -- Use this table column as bucket column later

> work\_place ARRAY<STRING>,

> gender\_age STRUCT<gender:string,age:int>,

> skills\_score MAP<string,int>,

> depart\_title MAP<string,ARRAY<string >>

> )

> CLUSTERED BY (employee\_id) INTO 2 BUCKETS -- Support more columns

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY '|'

> COLLECTION ITEMS TERMINATED BY ','

> MAP KEYS TERMINATED BY ':';

Once the table is created, show tables properties and data location;

drwxr-xr-x - hduser supergroup 0 2020-09-29 22:26 /user/hive/warehouse/retail.db/employee\_id\_buckets

# Storage Information

SerDe Library: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

InputFormat: org.apache.hadoop.mapred.TextInputFormat

OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

Compressed: No

Num Buckets: 2

Bucket Columns: [employee\_id]

##### Loading data into Buckets:

For loading data into BUCKETS we cannot use LOAD command – as it will just dump the data, but will not segregate data into their hash values

For this we have to use INSERT

insert into table employee\_id\_buckets SELECT \* FROM employee\_id;

this command invokes reducers, since the data has to be split into buckets based on HASH values

## VIEWS:

Views are logical data structures. It is called logical because views are only defined in metastore without the footprint in HDFS. Unlike what's in the relational database, views in HQL do not store data or get materialized. Once the view is created, its schema is frozen immediately. Subsequent changes to the underlying tables (for example, adding a column) will not be reflected in the view's schema. If an underlying table is dropped or changed, subsequent attempts to query the invalid view will fail. In addition, views are read-only and may not be used as the target of the LOAD/INSERT/ALTER statements

hive> CREATE VIEW IF NOT EXISTS employee\_skills

> AS

> SELECT

> name, skills\_score['DB'] as DB,

> skills\_score['Perl'] as Perl,

> skills\_score['Python'] as Python,

> skills\_score['Sales'] as Sales,

> skills\_score['HR'] as HR

> FROM employee\_id;

# LOAD

### INPATH

LOAD DATA LOCAL INPATH – when local keyword is given it perfor -copyFromLocal command in the back to place the data from unix shell to HDFS. Data copied from local is retrieved

When LOCAL is not specified data is loaded from INPATH and source data is deleted. It perform -mv command internally in HDFS folders

#### Example:

Existing table contains 4 rows:

hive> select name from employee\_id;

name

Michael

Will

Shelley

Lucy

Time taken: 0.364 seconds, Fetched: 4 row(s)

When we do load again, the data is placed again in the folder

hive> load data local inpath '/home/hduser/employee.txt' into table employee\_id;

Loading data to table retail.employee\_id

hive> select name from employee\_id;

name

Michael

Will

Shelley

Lucy

Michael

Will

Shelley

Lucy

Time taken: 0.184 seconds, Fetched: 8 row(s)

Data is loaded again. Duplicated.

### OVERWRITE:

Overwrite command erases existing data from the table location and loads the new data.

hive> load data local inpath '/home/hduser/employee.txt' OVERWRITE into table employee\_id;

Loading data to table retail.employee\_id

hive> select name from employee\_id;

name

Michael

Will

Shelley

Lucy

Time taken: 0.218 seconds, Fetched: 4 row(s)

Existing files are removed and loaded freshly

##### Loading data into partitions:

hive> show partitions employee\_partitioned;

partition

year=2018/month=11

year=2018/month=12

year=2020/month=2

hive> load data local inpath '/home/hduser/employee.txt' overwrite into table employee\_partitioned partition(year=2020, month=2);

# INSERT

### INSERT INTO TABLE

Insert into table supports inserting data into table from other tables as a result of select statement

hive> create external table check\_ctas (empname string);

hive> insert into table check\_ctas select name from employee\_partitioned;

The INSERT OVERWRITE statement will replace the data in the target table/partition, while INSERT INTO will append data.

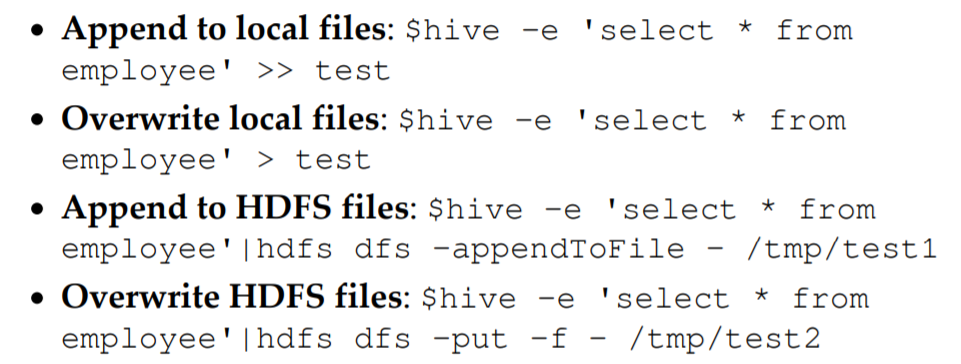
### INSERT to data files

INSERT also supports writing data to files, which is the opposite operation compared to LOAD. It is usually used to extract data from SELECT statements to files in the local/HDFS directory. However, it only supports the OVERWRITE keyword, which means we can only overwrite rather than append data to the data files

>> Data cannot be appended, its always overwritten <<

hive> insert overwrite local directory '/home/hduser/hiveexternaldata' select \* from buckettxnrecsbycat;

Combine HIVE + HDFS commands



# DML Statements in HIVE table:

In Hive Update and Delete work based on some limitations

* It can only be performed on tables that support ACID.
* If a table is to be used in ACID writes (insert, update, delete) then the table property “transactional” must be set on that table.
* Only ORC file format is supported in this.
* Tables must be bucketed to make use of these features.

NOTE: HIVE creates Delta files for each Update, Insert and Delete operations. Old data is still available in HDFS until COMPACTION is performed in HIVE

#### Create Transactional Table:

create table empdml (

EmployeeID Int,

FirstName String,

Designation String,

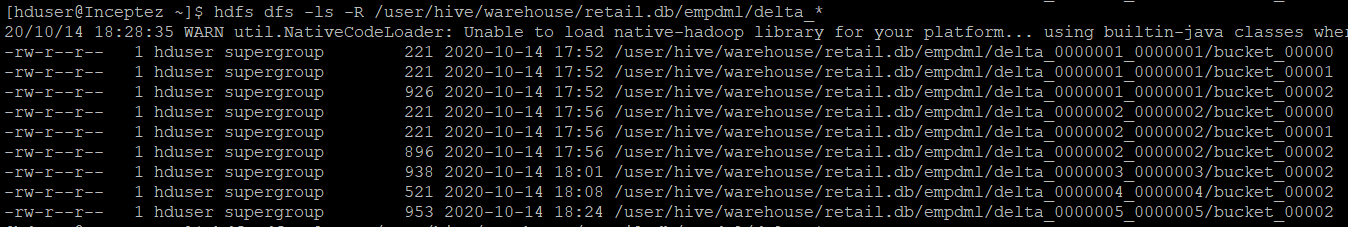
Salary Int,

Department String)

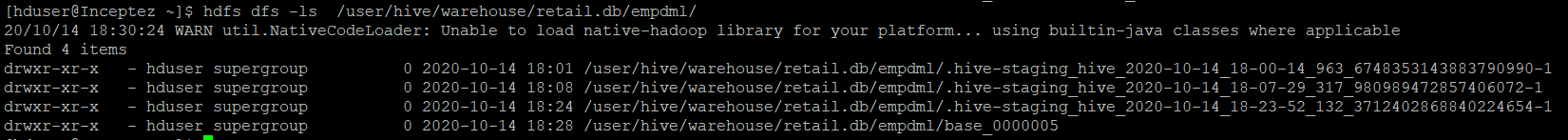
clustered by (department) into 3 buckets

stored as orc TBLPROPERTIES ('transactional'='true');

#### Delta files in HIVE before compaction:



#### Delta files after Compaction:



#### How to do COMPACTION:

hive> alter table empdml COMPACT 'major';

Compaction enqueued.

# Dynamic Schema – Hive Table Creation:

Download arvo jar in [**https://mvnrepository.com/artifact/org.apache.avro/avro-tools/1.8.1**](https://mvnrepository.com/artifact/org.apache.avro/avro-tools/1.8.1)

**Sqoop import data from DB as arvo fomat:**

sqoop import -Dmapreduce.job.user.classpath.first=true \

--connect jdbc:mysql://localhost/custpayments --username root --password root \

-table customers -m 3 --split-by customernumber \

--target-dir /user/hduser/custavro \

--delete-target-dir \

--as-avrodatafile;;

**Get Schema from the Avro format data:**

hadoop jar /home/hduser/Downloads/avro-tools-1.8.1.jar getschema /user/hduser/custavro/part-m-00000.avro > /home/hduser/customer.avsc

[hduser@Inceptez ~]$ cat /home/hduser/customer.avsc

{

"type" : "record",

"name" : "customers",

"doc" : "Sqoop import of customers",

"fields" : [ {

"name" : "customerNumber",

"type" : [ "null", "int" ],

"default" : null,

"columnName" : "customerNumber",

"sqlType" : "4"

}, {

"name" : "customerName",

"type" : [ "null", "string" ],

"default" : null,

"columnName" : "customerName",

"sqlType" : "12"

}, {

Put the schema in HDFS:

hadoop fs -put -f customer.avsc /tmp/customer.avsc

Create table in Hive using schema(without column specification):

create external table customeravro

stored as AVRO

location '/user/hduser/custavro'

TBLPROPERTIES('avro.schema.url'='hdfs:///tmp/customer.avsc');