

7. Hadoop ETL

Lecture

- ❖ Hadoop ETL Development,
- ❖ ETL Process in Hadoop,
- ❖ Discussion of ETL functions,
- ❖ Data Extractions,
- ❖ Need of ETL tools
- ❖ Advantages of ETL tools.

Lab-Assignment:

- ❖ Understand the file formats and read the provided links

11. Apache Airflow

Lecture

- ❖ Introduction to Data Warehousing and Data Lakes
- ❖ Designing Data warehousing for an ETL Data Pipeline
- ❖ Designing Data Lakes for an ETL Data Pipeline
- ❖ ETL vs ELT
- ❖ Fundamentals of Airflow
- ❖ Work management with Airflow
- ❖ Automating an entire Data Pipeline with Airflow

Lab-Assignment:

- ❖ Create a airflow DAG for Extract -> Transform -> Load

12, Apache Spark 13 Apache kafka

Hive contd.,

hive > SET mapreduce.framework.name=local;

Creating an hive sql script and executing it;

\$ vim sample.sql

CREATE DATABASE IF NOT EXISTS sampled;

USE sampled;

create table if not exists product (productid int, productname string, price float, category string);

describe product;

insert into table sampled.product values (1,'Books',25.0,'stationery');

insert into table sampled.product values (2,'Pens',10.0,'stationery');

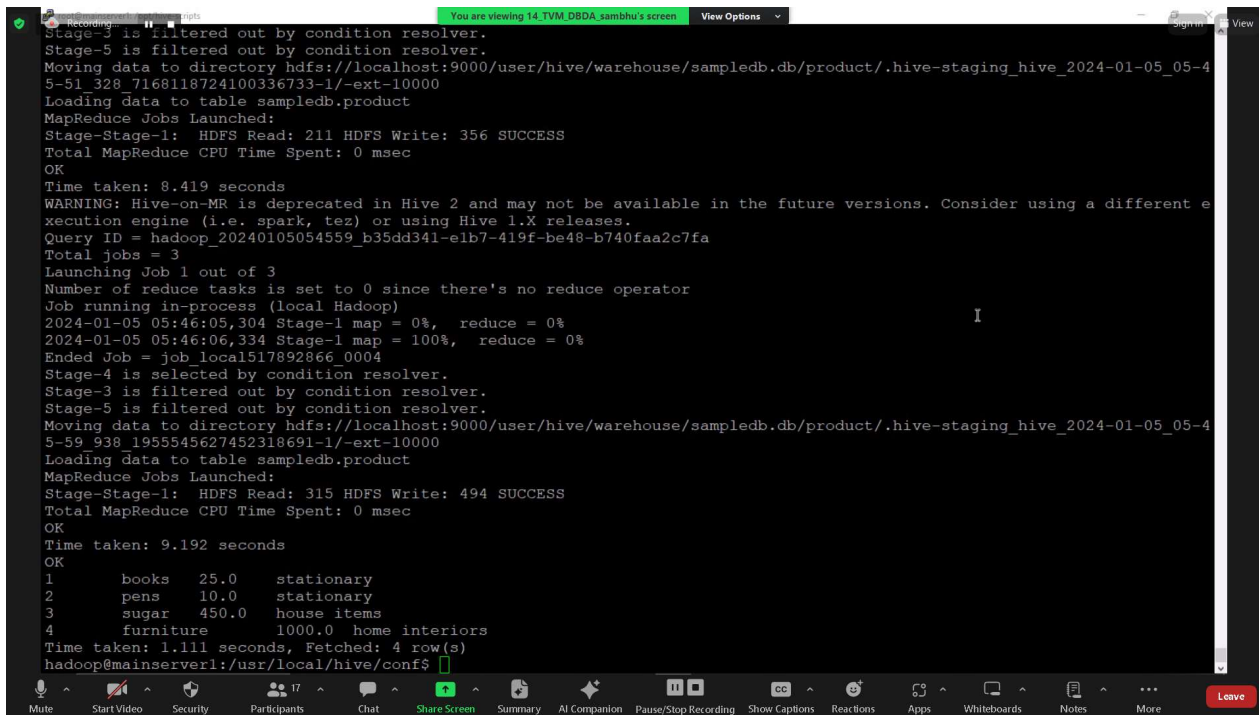
insert into table sampled.product values (3,'sugar',45.0,'House Item');

insert into table sampled.product values (4,'Furniture',10000.0,'Home interiors');

select * from product;

<save and exit>

\$ hive -f ./sample.sql



```
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/sampledb.db/product/.hive-staging_hive_2024-01-05_05-45-51_328_7168118724100336733-1/-ext-10000
Loading data to table sampledb.product
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 211 HDFS Write: 356 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 8.419 seconds
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = hadoop_20240105054559_b35dd341-e1b7-419f-be48-b740faa2c7fa
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Job running in-process (local Hadoop)
2024-01-05 05:46:05,304 Stage-1 map = 0%,  reduce = 0%
2024-01-05 05:46:06,334 Stage-1 map = 100%,  reduce = 0%
Ended Job = job_local517892866_0004
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/sampledb.db/product/.hive-staging_hive_2024-01-05_05-45-59_938_1955545627452318691-1/-ext-10000
Loading data to table sampledb.product
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 315 HDFS Write: 494 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 9.192 seconds
OK
1      books      25.0      stationary
2      pens       10.0      stationary
3      sugar      450.0     house items
4      furniture  1000.0    home interiors
Time taken: 1.111 seconds, Fetched: 4 row(s)
hadoop@mainserver1:/usr/local/hive/conf$
```

This is the final output of the script.

```
hive> use cdacdb;
hive> select * from student;
hive> select * from student where age < 20;
hive> select * from student where age <= 20;
hive> select * from student where age > 20;
hive> select * from student where age >= 20;
hive> select avg(age),max(age),min(age) from student;
hive> select name,avg(age) from student group by name;
hive> select id,age from student group by id;
hive> select id,avg(age) from student group by id,age;
hive> select * from student  sort by id desc;
hive> select name,age from student  sort by id desc;
hive> select name,age from student  sort by age desc;

hive> use sampledb;
hive> select * from product;

hive> create view prodview as select productname,category from product;
hive> select * from prodview;
hive> describe prodview;
```

Now perform the following activity in hive

```
hive> create table store (
storeid int,
storename string,
storeprice float,
storetype string
);

hive> insert into table store values (1,'rajstore',25000.0,'book stall');
hive> insert into table store values (2,'reynolds',10000.0,'pen store');
hive> insert into table store values (3,'cadburys',450.0,'chocolate factory');
hive> insert into table store values (4,'dumro',10000.0,'Home furnitures');

hive> select * from store;
```

JOIN two tables

```
hive> select product.*,store.* from product join store on product.productid=store.storeid ;
```

Inorder to update and delete a particular row from the table in hive we need to create the table with the following options:

Sample data:-

```
hive> SET hive.support.concurrency=true;
hive> SET hive.enforce.bucketing=true;
hive> SET hive.exec.dynamic.partition.mode=nonstrict;
hive> SET hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;hive>
hive> SET hive.compactor.initiator.on=true;
hive> SET hive.compactor.worker.threads=1;
```

```
hive> create table newstore (
storeid int,
storename string,
storeprice float,
storetype string
) CLUSTERED By (storeid) into 2 Buckets stored AS ORC TBLPROPERTIES ('transactional'='true');
```

```
hive> insert into table newstore values (1,'rajustore',25000.0,'book stall');
hive> insert into table newstore values (2,'reynolds',10000.0,'pen sotre');
hive> insert into table newstore values (3,'cadburys',450.0,'chocolate factory');
hive> insert into table newstore values (4,'dumro',10000.0,'Home furnitures');
```

```
hive> describe newstore;
hive> select * from newstore;
```

Update the records from the table

```
hive> update newstore set storeprice = 45000.0 where storename='cadburys';
```

Delete a particular record form the table

```
hive> delete from newstore where storeprice = 25000.0;
```

Hive Storage Format

There are various formats supported by hive for storing data and depends up the requirements we can select what is best for us (based on impact on performance, compression the way how query executes etc ..,)

```
Hive > create database newdb;
```

1. File extension '.txt' , '.csv' { these are human-readable and used where interoperability is required }
 - a. .txt is the plain text
 - b. .csv the comma separated format.

Eg:- hive> create table mytable1 (slno int, username string) row format delimited fields terminated by ',';
{now insert few records to verify how it is storing on hdfs file sytsem }

2. Sequence file : { these formation are suitable for Hadoop Map Reduce Jobs.)
 - a. '.sequence' as the file extension Binary file format used to store key-value pairs
 - b. Helps in processing large-scale data efficiently

Eg: - hive> create table mytable2 (slno int, username string) stored as sequencefile;
{now insert few records to verify how it is storing on hdfs file sytsem }

3. ORC - Optimized Row Columnar (used in analytical queries , best use when your datasets are larger)
 - a. .orc as the file extension
 - b. Columnar storage format
 - c. Designed to high performance and compression

Eg: - hive> create table mytable3 (slno int, username string) stored as ORC;
{now insert few records to verify how it is storing on hdfs file sytsem }

4. Avro.
 - a. .avro as the file extension
 - b. Binary data serialization format which supports schema evolution

c. Suitable for data serialization and interoperability with different platforms

```
Eg:- hive> CREATE TABLE avro_table ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe' STORED AS
INPUTFORMAT 'org.apache.hadoop.hive ql.io.avro.AvroContainerInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive ql.io.avro.AvroContainerOutputFormat' TBLPROPERTIES ('avro.schema.literal'='{ "type": "record",
"name": "User", "fields": [{"name": "sln", "type": "int"}, {"name": "usrname", "type": "string"}]}');
```

{now insert few records to verify how it is storing on hdfs file system }

Sample Avro file in json format:

```
{
  "type": "record",
  "name": "User",
  "fields": [
    { "name": "sln", "type": "int"},
    { "name": "usrname", "type": "string"}
  ]
}
```

5. Parquet.

- a. .parquet as the file extension
- b. Binary data serialization format which supports schema evolution, compression
- c. Suitable for interoperability with different platforms with complex nested data structures
- d. Supports various processing engines

```
Eg: - hive> create table mytable4 ( slno int, username string ) stored as parquet;
      {now insert few records to verify how it is storing on hdfs file system }
```

Dummy data inserted in all the tables for testing

```
hive> insert into mytable1 values (1 , 'rama');
hive> insert into mytable1 values (2 , 'sita');
```

```
hive> insert into mytable2 values (1 , 'rama');
hive> insert into mytable2 values (2 , 'sita');
```

```
hive> insert into mytable3 values (1 , 'rama');
hive> insert into mytable3 values (2 , 'sita');
```

```
hive> insert into mytable4 values (1 , 'rama');
hive> insert into mytable4 values (2 , 'sita');
```

```
hive> insert into avro_table values (1 , 'rama');
hive> insert into avro_table values (2 , 'sita');
```

```
hive> select * from mytable1;
hive> select * from mytable2;
hive> select * from mytable3;
hive> select * from mytable4;
hive> select * from avro_table ;
```

NOTE: you will not find any difference in the output but while creating and inserting the amount of time taken there will be difference.

Adding a new column to the existing table:

```
hive> alter table mytable1 add columns ( password string );
```

Note we can use this alter table for csv,txt, sequence and parquet format, We cannot alter avro format directly;

```
CREATE TABLE avro_table1 ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe' STORED AS
INPUTFORMAT 'org.apache.hadoop.hive ql.io.avro.AvroContainerInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive ql.io.avro.AvroContainerOutputFormat' TBLPROPERTIES ('avro.schema.literal'='{
```

```
{
  "type": "record",
  "name": "User",
  "fields": [
    { "name": "sln", "type": "int"},
    { "name": "usrname", "type": "string"}
  ]
}
```

```

        {"name": "password", "type": "string"}
    ]
}
);

```

To remove all the row or data form the table without deleting the structure of the table

```
hive> truncate table mytable1;
```

Importing data preset in one table on to another provided the columns and data type matches;

```
hive > insert into mytable1 from select * from mydata_csv;
```

To create an external table

```
hive > create external table mytable_csv ( slno int, username string, password string ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
Location '/user/hive/warehouse/export';
```

Partitioning and Bucket in HIVE

❖ Two techniques that are used for organizing and structuring the data in tables for better performance and query optimization.

❖ Buketing:

- Data is divided into a fixed number of buckets based on hash function applied to a specified column.
- Each bucket represents a subset of data, and the number of buckets is determined by user.
- Useful for evenly distributing data, which leads to better query performance.

Eg:

```
create table newstore (
storeid int,
storename string,
storeprice float,
storetype string
) CLUSTERED By (storeid) into 2 Buckets ;
```

In the above example we are using storeid column for bucketing and the value 2 represents the number of buckets to create

❖ Partitoning:

- Data is divided into sub-directories based on the values of one or more columns.
- Each sub-dirctory represents a partition
- Helps hive to skip ununecesary data when querying which leads to improvement in performance
- Partitioning is particularly useful when you often filter queries based on specific values of a column.
- It helps reduce the amount of data that needs to be scanned during query execution

Eg: -

```
hive > create table part_table1 ( slnno int, name string, city string, location string, zipcodes int) PARTITIONED BY(state string) ROW FORMAT
DELIMITED FIELDS TERMINATED BY ',';
```

```
Hive> insert into table part_table1 partition (state='karnatka') values (1,'giridhar','bangalore', '4thcross', 560001);
Hive> insert into table part_table1 partition (state='mumbai') values (2,'suri','xyz', '5thcross', 660001);
Hive> insert into table part_table1 partition (state='chennai') values (3,'christy','abc', '6thcross', 760002);
Hive> insert into table part_table1 partition (state='apandtg') values (4,'mr.bean','new1', '7thcross', 160021);
Hive> insert into table part_table1 partition (state='kerala') values (5,'tommy','new2', '8thcross', 450001);
```

```
Hive> select * from part_table1;
```

Verification of the partition can be done using the hdfs command as follows:

```
$ hdfs dfs -ls /user/hive/warehouse/newdb.db/part_table1/
```

