Mini Project on Movies

Technologies Covered: Sqoop, HDFS, Hive, Spark (Scala) and MySQL

Develop script for the following tasks using the respective mentioned technologies in the task and execute them.

1. Source Tables setup:
   a. Refer the data model attached in this zip file. Create a new database in MySQL instance called “MVS” and create 2 tables as mentioned in the data model.
   b. The names of the new tables are as follows
      i. Movies
      ii. Movies_Details

Creating databse in MySql

```
create database MVS;
```

Creating Movies table

```
create table Movies (
        movie_id int,
        title text,
        cast text,
        crew text
);
```

Creating Movies Details table

```
create table Movies_Details (
                budget int,
                genres text,
                homepage text,
                id int,
                keywords text,
                original_language text,
```

```
    original_title text,
    overview text,
    popularity decimal(10,6),
    production_companies text,
    production_countries text,
    release_date DateTime,
    revenue int,
    runtime int,
    spoken_languages text,
    status text,
    tagline text,
    title text,
    vote_average decimal(10,1),
    vote_count int
);
```

2. Load data into MySQL –A small sample is provided in zipped file. Download full dataset from https://www.kaggle.com/tmdb/tmdb-movie-metadata/data (zipped size 9 mb)- Import the 2 CSV files to the above created 2 tables. Ensure that all data loaded by checking the count of records and random values check.

Loading data into Movies Details table

```
load data infile '/home/cloudera/Downloads/movies_details.csv'
into table Movies_Details
fields terminated by ','
enclosed by ""
lines terminated by '\n'
ignore 1 lines;
```

Loading data into Movies table

```
load data infile '/home/cloudera/Downloads/movies.csv'
into table Movies
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 lines;
```

3. Create a new database in Hive called 'MVS'.

```
create database mvs;
use mvs;
```

4. Data Ingestion
   a. Import all the tables in HIVE using Sqoop. Don't create the table manually in Hive. Let Sqoop create the equivalent tables in Hive (under the new database movies) and load the data into those tables. Create one script for each table.
   b. Now, do the above thing(4a) using Spark. If the target table exists, your script need to drop the table and recreate it with the respective source table data. If your script got executed multiple times also, it should not give any error, each time it should drop the target table and load the data from source table. Create one script for each table. It is also called Truncate reload method.

DATA INGESTION

importing Movies table from mysql to hive through sqoop

```
sqoop import
--connect jdbc:mysql://localhost/MVS
--username root
--password cloudera
--table Movies
--hive-import
```

--create-hive-table

--hive-table MVS.Movies

--target-dir a1

--m 1

importing Movies Details table from mysql to hive through sqoop

sqoop import

--connect jdbc:mysql://localhost/MVS

--username root

--password cloudera

--table Movies_Details

--hive-import

--create-hive-table

--hive-table MVS.Movies_Details

--target-dir a2

--m 1

importing Movies table from mysql to hive through Spark

```
val result = sqlContext.read.format("jdbc")
        .option("url", "jdbc:mysql://localhost/MVS")
        .option("dbtable", "Movies")
        .option("user", "root")
        .option("password", "cloudera")
        .load()
        .write
        .mode("overwrite")
        .saveAsTable("MVS.Movies")
```

importing Movies Details table from mysql to hive through Spark

```
val result = sqlContext.read.format("jdbc").
        .option("url", "jdbc:mysql://localhost/MVS?zeroDateTimeBehavior=convertToNull")
        .option("dbtable", "Movies_Details")
        .option("user", "root")
        .option("password", "cloudera")
        .load()
        .write
        .mode("overwrite")
        saveAsTable("MVS.Movies_Details")
```

5. Explore/Suggest other Data Validation checks for this dataset import from MySql to Hive and apply at least one them. Compare the time taken of two imports (Sqoop vs Spark).

Through Sqoop

```
Time taken: 3.908 seconds

Loading data to table mvs.movies
Table mvs.movies stats: [numFiles=1, totalSize=35099001]
OK

Time taken: 2.221 seconds

Logging initialized using configuration in
jar:file:/usr/lib/hive/lib/hive-common-1.1.0-cdh5.12.0.jar!/hive-log4j.properties
OK

Time taken: 2.685 seconds

Loading data to table mvs.movies_details
Table mvs.movies_details stats: [numFiles=1, totalSize=5210385]
OK

Time taken: 0.364 seconds
```

Through Spark

It takes lesser time compared to sqoop import...(some milli seconds probably)

6. Create 2 new external tables in Hive with the following structure. Change the data type for the following columns (existing data type json String) to hive complex datatype (Array /Struct/Map) as appropriate –

   Movies – cast, crew

   Movies_details – geners, keywords, production_companies, production_countries, spoken_languages.

Table Name	Columns	Partition Column
Movies	Same as MySql table	No partitioning.
Movies_Details_part	All columns + Add new column for Year of Release(Populate it by parsing year from release_year)	Year of Release

7. Develop Hive Scripts, to load the above new partitioned tables from the respective Hive tables. Make sure that your script completely overwrites the existing data in the target table. Load method is Truncate reload. Remove any incomplete data.

Source Table	Target Table
Movies	Movies
Movies_Details	Movies_Details_part

Creating Movies external table

```
create external table Movies2
(
 movie_id SMALLINT,
 title String,
 castt array < struct   <
                                    cast_id : SMALLINT,
                                    character : String,
                                    credit_id : String,
                                     gender : SMALLINT,
```

```
                                        id : INT,
                                        name : String,
                                        order : SMALLINT
                                    >>,

 crew array <   struct   <
                                        credit_id : String,
                                        department : String,
                                        gender : SMALLINT,
                                        id : SMALLINT,
                                        job : String,
                                        name : String
                                    >>
      )
     row format delimited
     fields terminated by '|'   \
     collection items terminated by ','
     lines terminated by '\n';
     insert overwrite table Movies2
     select movie_id,title,castt,crew from Movies;
```

Creating Movies Details external table

```
create external table Movies_Details_Part
(
            budget int,
            genres array < struct<
                                        id : SMALLINT,
                                        name : String
```

```
                                        >>,
homepage String,
id int,
keywords array< struct <
                                                id:SMALLINT,
                                                name:String
                                            >>,
original_language String,
original_title String,
overview String,
popularity decimal(10,6),
production_companies array     < struct <
                                                        name:String,
                                                        id:SMALLINT
                                                    >>,
production_countries array < struct   <
                                                         iso_3166_1 : String,
                                                         name : String
                                                    >>,
release_date Date,
revenue int,
runtime int,
spoken_languages array   <   struct <
                                                          iso_639_1:String,
                                                          name:String>>,
status String,
tagline String,
```

```
        title String,
        vote_average decimal(10,1),
        vote_count int
 )
partitioned by (year_of_release int)
row format delimited
fields terminated by '|'
collection items terminated by ','
lines terminated by '\n'
set hive exec.dynamic.partition.mode = nonstrict;
insert overwrite table movies_details_part
partition(year_of_release)
select budget, genres,homepage,id,keywords,original_language,original_title,overview,
popularity,production_companies,production_countries,release_date,revenue,runtime,sp
oken_languages,status, tagline,title, vote_average,vote_count,year(release_date) from
Movies_Details;
```

NOTE:

We cannot insert data into above created external tables from internal tables imported through sqoop and spark because data is not in proper format.....

Therefore, create new internal table as below and load the formatted data and then use this as source table....

Creating managed and external tables for Movies

```
Drop table mov;
Create table if not exists mov(
```

```
id String,
title String,
mov_cast String
)
row format delimited
fields terminated by '|'
lines terminated by '\n'
stored as TextFile;

LOAD DATA LOCAL INPATH
'/home/cloudera/Downloads/final_movie_dataset/movie_1000.csv'
OVERWRITE INTO TABLE mov;
desc mov;
select count(*) from mov;
select id from mov limit 1;
select mov_cast from mov limit 1;

drop table mov_stage;

create table mov_stage
AS select id, title,   movie_cast   from mov
lateral view explode (
                                                split (
                                                regexp_replace (
                                                regexp_replace (   mov_cast,   ' \\}   \\, \\{ ',     ' \\}
\\;   \\{   '   ),
                                               '   \\[   |   \\] ', ''),
```

' \\; ')

) x as movie_cast ;

```
desc mov_stage;
select movie_cast from mov_stage limit 1;
drop table mov_merge;
CREATE TABLE mov_merge AS
SELECT id,
title,
get_json_objectmov_stage .movie_cast , '$.cast_id') AS cast_id,
get_json_object(mov_stage .movie_cast , '$.character') AS character,
get_json_object(mov_stage .movie_cast , '$.credit_id') AS credit_id,
get_json_object(mov_stage .movie_cast , '$.gender') AS gender,
get_json_object(mov_stage .movie_cast , '$.id') AS cstid,
get_json_object(mov_stage .movie_cast , '$.name') AS name,
get_json_object(mov_stage .movie_cast , '$.order') AS order
FROM mov_stage;

desc mov_merge;
select name from mov_merge limit 2;
drop table mov_ext;
create table movies_external (
        id String,
        title String,
        castt struct <
```

```
                cast_id:String,
                character:String,
                credit_id:String,
                gender:String,
                id:String,
                name:String,
                order:String >
        )
        row format delimited

        fields terminated by ','
        lines terminated by '\n'
        stored as textfile;
insert into table movies_external
        select id, title ,
        named_struct (
        'cast_id',mov_merge.cast_id ,
        'character',mov_merge.character,
        'credit_id',mov_merge.credit_id,
        'gender',mov_merge.gender,
        'id',mov_merge.cstid,
        'name',mov_merge.name,
        'order',mov_merge.order
        ) as castt from mov_merge;
```

Creating managed and external tables for Movies Details

```
create table mov_det1 (
budget Int,
genres String,
id Int,
release_date String,
revenue BigInt,
title String,
vote_average decimal(10,1),
vote_count Int
)
row format delimited
fields terminated by '|'
lines terminated by '\n'
stored as TextFile ;

LOAD DATA LOCAL INPATH
'/home/cloudera/Downloads/final_movie_dataset/movie_details.csv'
 OVERWRITE INTO TABLE mov_det1;

select count(*) from mov_det1;
select vote_average from mov_det1 limit 5;
CREATE TABLE   mov_det_stage1 AS
select budget, mov_genres, id, release_date, revenue, title, vote_average, vote_count
from mov_det1
lateral view explode (
                                                split (
```

```
                                                    regexp_replace (

                                                    regexp_replace ( genres, '   \\}   \\,   \\{   ',
' \\}   \\;   \\{   ' ),

                                                  ' \\[   |   \\]   ',   ''   ) ,      '   \\; '     )

                                    ) x as mov_genres ;

CREATE TABLE mov_det_merge1 AS
SELECT budget,
get_json_object(mov_det_stage1.mov_genres, '$.id') AS genre_id,
get_json_object(mov_det_stage1.mov_genres, '$.name') AS genre_name,
id,
release_date,
revenue,
title,
vote_average,
vote_count
FROM mov_det_stage1;
desc mov_det_merge1;
create table movies_details_external (
budget int,
genre struct <
                    genre_id:string,
                  genre_name:string
               >,
id int,
```

```
release_date string,
revenue bigint,
title string,
vote_average decimal(10,1),
vote_count int
)
    PARTITIONED BY (year_of_release String)
    row format delimited
    fields terminated by ','
    lines terminated by '\n'
    stored as textfile;
set hive.exec.dynamic.partition = true;
set hive.exec.dynamic.partition.mode = nonstrict;

insert into table movies_details_external
partition(year_of_release)
select budget,
named_struct( 'genre_id',mov_det_merge1.genre_id, 'genre_name',
mov_det_merge1.genre_name) as genre,
id , release_date, revenue, title, vote_average, vote_count,split(release_date,'/')[2] from
mov_det_merge1;
```

8. Load the data from above two external tables in data frames and Join them in One Data frame.

```
import org.apache.spark.sql.hive.HiveContext
val hctx = new HiveContext(sc)
val movies_details = hctx.table("mvs.mov_det_ext1")
```

```
val mov = hctx.table("mvs.mov_ext ")

val movies = mov.drop("title")

val df1 = movies.join(movies_details,"id")
```

9. Cache this data frame.

```
val cached_df = df1.cache
```

10. Develop Spark Scala script for the following. Query the data frame for top 3 movies of each year by Gross Revenue Collection

```
val df2=df1.select("year_of_release","id","revenue").distinct

import org.apache.spark.sql.expressions.Window

import org.apache.spark.sql.functions.{row_number, max, broadcast}

val w =  Window.partitionBy($"year_of_release").orderBy($"revenue".desc)

val res = df2.withColumn("rn", row_number.over(w)).where($"rn" <= 3)

res.show
```

11. Query the Data frame to find

    a. Find all the movies with Genre “Action” and whose vote_average was 7 and higher.

    b. Also write the output to a new HDFS file called /User/YourName/Results.csv. Load method is truncate reload. Fields comma separated.

```
val df2 = df1.select("vote_average","id","genre.genre_name").distinct

val df3 = df2.filter(df2("genre_name")==="Action")

val df4 = df3.filter(df3("vote_average") >= 7)

df4.show()

df4.write.format("json").mode("overwrite").save("user/KTR/query11.json")
```

12. Query the Data frame to find

a. Find all the distinct Genre across all the movies and count the No of movies each Genre. If required, make a new hive table for required transformation.

b. Also write the output to a new HDFS file called /User/YourName/Results.csv. Load method is truncate reload. Fields comma separated.

```
val df2 = df1.select("title", "genre.genre_name").distinct

val df3 = df2.groupBy($"genre_name").count

df3.show()

df3.write.format("json").mode("overwrite").save("user/KTR/query12.json")
```

13. Query the Data frame to find

a. For each year find the movies which had single Genre.

b. Also write the output to a new HDFS file called /User/YourName/Results.csv. Load method is truncate reload. Fields comma separated.

```
val df2 = df1.select("year_of_release","title","genre.genre_name").distinct

val df3 = df2.groupBy($"year_of_release", $"title")

val df4 = df3.count

val df5 = df4.filter($"count" === 1)

df5.show

df5.write.format("json").mode("overwrite").save("user/KTR/query13.json")
```

14. Query the Dataframe to find

a. Find out the Movies that had “Walt Disney Pictures” as one of the production companies and was also produced outside US.

b. Also write the output to a new HDFS file called /User/YourName/Results.csv. Load method is truncate reload. Fields comma separated.

```
create table mov_prod(

id String,

title String,
```

```
prod_comp String,
prod_count String
)
row format delimited
fields terminated by '|'
lines terminated by '\n'
stored as TextFile ;

LOAD DATA LOCAL INPATH
'/home/cloudera/Downloads/movie_production.txt'
OVERWRITE INTO TABLE mov_prod;
select count(*) from mov_prod;
select title from mov_prod limit 5;
CREATE TABLE   mov_prod_stage AS
select id,title,prod_company,prod_country from mov_prod
lateral view explode   (split   (
                                        regexp_replace (
                                        regexp_replace   (   prod_comp, '   \\}   \\,   \\{ '   ,   '
\\}   \\;   \\{   '   ),
                                        '   \\[   |   \\] ',   ' ' ),
                                         '   \\;   '    )   ) x as prod_company
 lateral view explode      (split     (
                                       regexp_replace   (
                                       regexp_replace   (   prod_count,   '   \\}   \\, \\{   ',    '
\\}   \\;    \\{   '   ),
                                       '   \\[   |   \\]   ',   '' ),
```

' \\; ')) y as prod_country;

```
CREATE TABLE mov_prod_merge AS
SELECT id,
title,
get_json_object(mov_prod_stage.prod_company, '$.name') AS comp_name,
get_json_object(mov_prod_stage.prod_company, '$.id') AS comp_id,
get_json_object(mov_prod_stage.prod_country, '$.iso_3166_1') AS country_id,
get_json_object(mov_prod_stage.prod_country, '$.name') AS country_name
FROM mov_prod_stage;
desc mov_prod_merge;
val ddf = hctx.table("mvs.mov_prod_merge")
val df2 = ddf.select("id", "title","comp_name","country_id").distinct
val df3 = df2.filter($"comp_name" === "Walt Disney Pictures")
val df4 = df3.filter(not($"country_id" === "US"))
val df5 = df4.groupBy("id").agg(collect_list("country_id"))
df5.show
```

15. Query the Dataframe to find
    a. Find average of “Vote_avergae” for all the Movies where “James Bond” was one of the cast.
    b. Also write the output to a new HDFS file called /User/YourName/Results.csv. Load method is truncate reload. Fields comma separated

```
val df2 = df1.select("title","castt.character","vote_average").distinct
val df3 = df2.filter($"character" === "James Bond")
val df4 = df3.groupBy($"character")
```

```
val df5 = df4.avg("vote_average")

df5.show

df5.write.format("json").mode("overwrite").save("user/KTR/query15.json")
```

16. Query the Dataframe to find

    a. Find top 3 movies in each year that earned most Profit. Are there any movies which made loss? If there is any then list them all with their Years

    b. Also write the output to a new HDFS file called /User/YourName/Results.csv. Load method is truncate reload. Fields comma separated

```
val df2 = df1.select("year_of_release", "title","revenue","budget").distinct

val df3 = df2.withColumn("result",$"revenue"-$"budget")

val calc1 = udf (

(result: Double)=> {

if(result > 0 ) result

else 0

})

val calc2 = udf (

(result: Double)=> {

if(result > 0 ) 0

else result

})

val df4 = df3.withColumn("Profit", calc1($"result"))

val df5 =    df4.withColumn("Loss", calc2($"result"))

val df6 = df5.filter($"profit" > 0)

val w =    Window.partitionBy($"year_of_release").orderBy($"profit".desc)

val profit_res = df6.withColumn("rn", row_number.over(w)).where($"rn" <= 3)
```

```
val df7 = df5.filter($"loss" < 0)

val w1 =    Window.partitionBy($"year_of_release").orderBy($"loss".asc)

val loss_res = df7.withColumn("rn", row_number.over(w1)).where($"rn" <= 3)
```

17. Create a new Hive Managed table where file format is Parquet.

Movies_Parq
Id
Title
Budget
Revenue
Vote_Avergae
Year

```
create table Movies_Parq (

        id String,

        title String,

        Budget Int,

        Revenue BigInt,

        vote_average decimal(10,1),

        year String
)
STORED AS PARQUET;
```

18. Transform the original cached Dataframe to a data frame as required by above Parquet table and then Save the data frame in table “Movies_Parq”.

```
val dff = cached_df.select("id", "title", "budget", "revenue", "vote_average",
"year_of_release").distinct

val dfr = dff.withColumnRenamed("year_of_release", "year")

dfr.write.mode("overwrite").format("parquet").saveAsTable("mvs.movies_parq")
```

19. Load and Query the “Movies_Parq” table for count of movies released in each year.

```
val df_mov_parq = hctx.table("mvs.movies_parq")

val df = df_mov_parq.groupBy($"year").count

df.show()
```