**Hive Practice**

CONTENTS

[1. Basic Hive Interaction Using Hue (15 points) 3](#_Toc126593965)

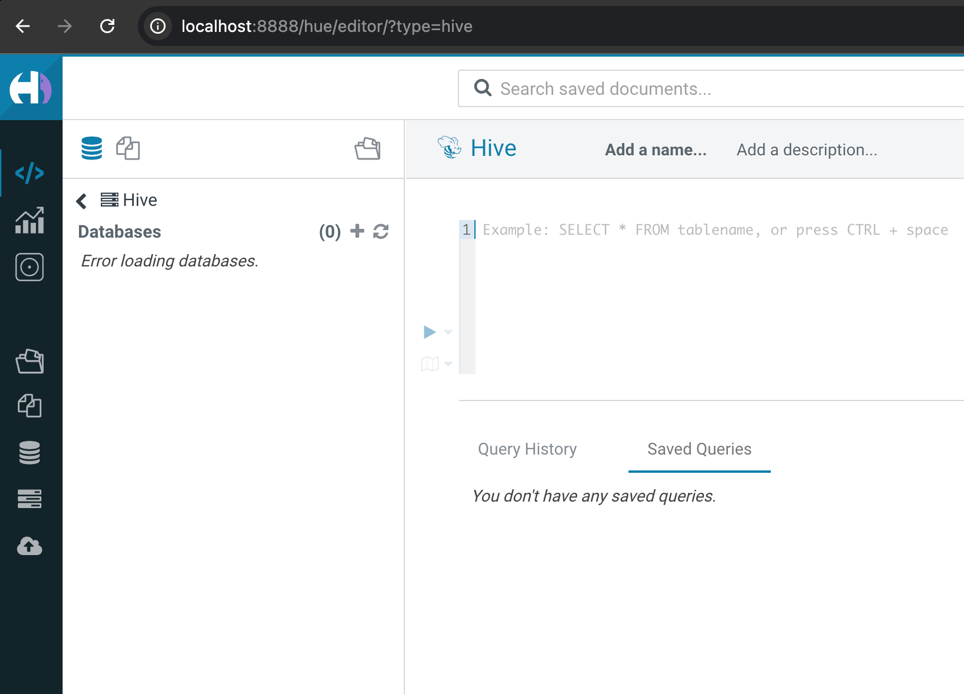
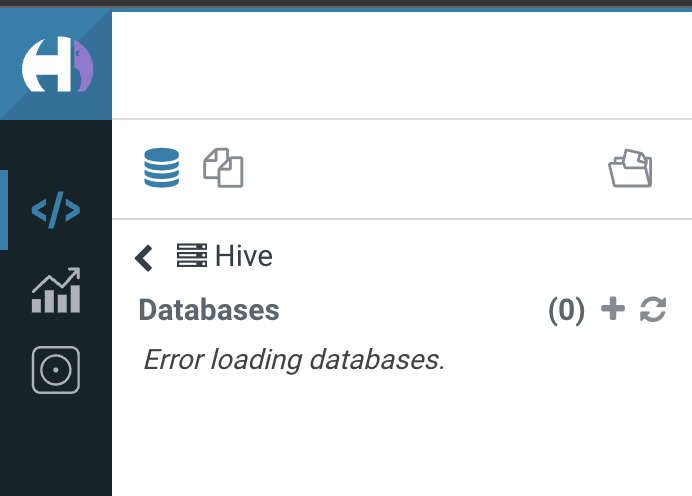
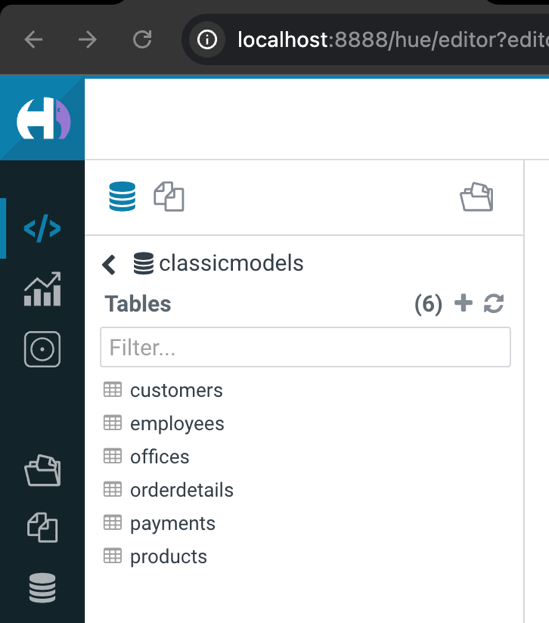
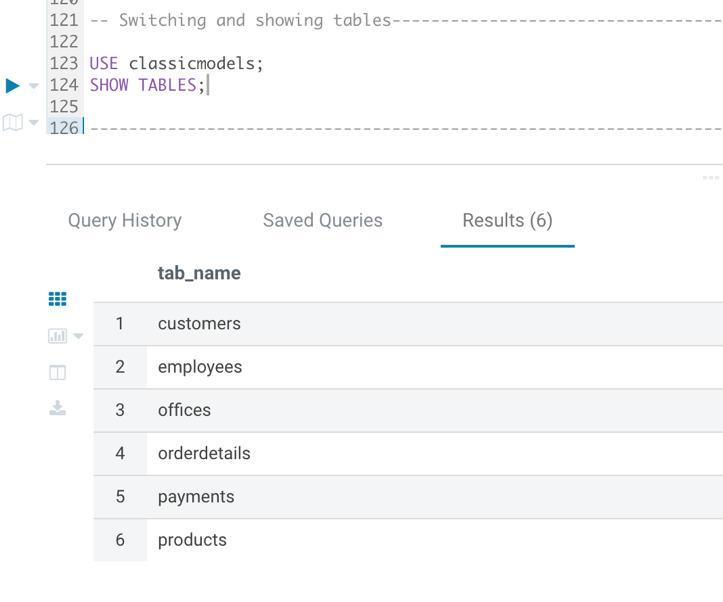
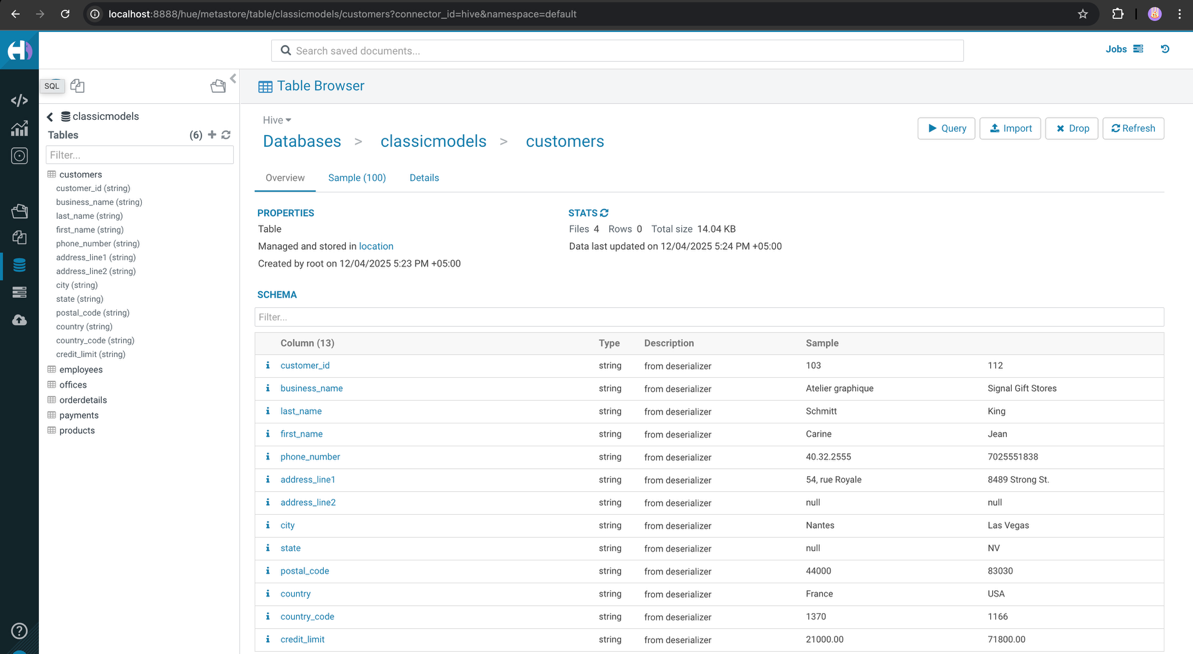
[2. Basic Hive Interaction Using Beeline (10 points) 4](#_Toc126593966)

[3. Managed and External Tables Using Beeline (20 points) 5](#_Toc126593967)

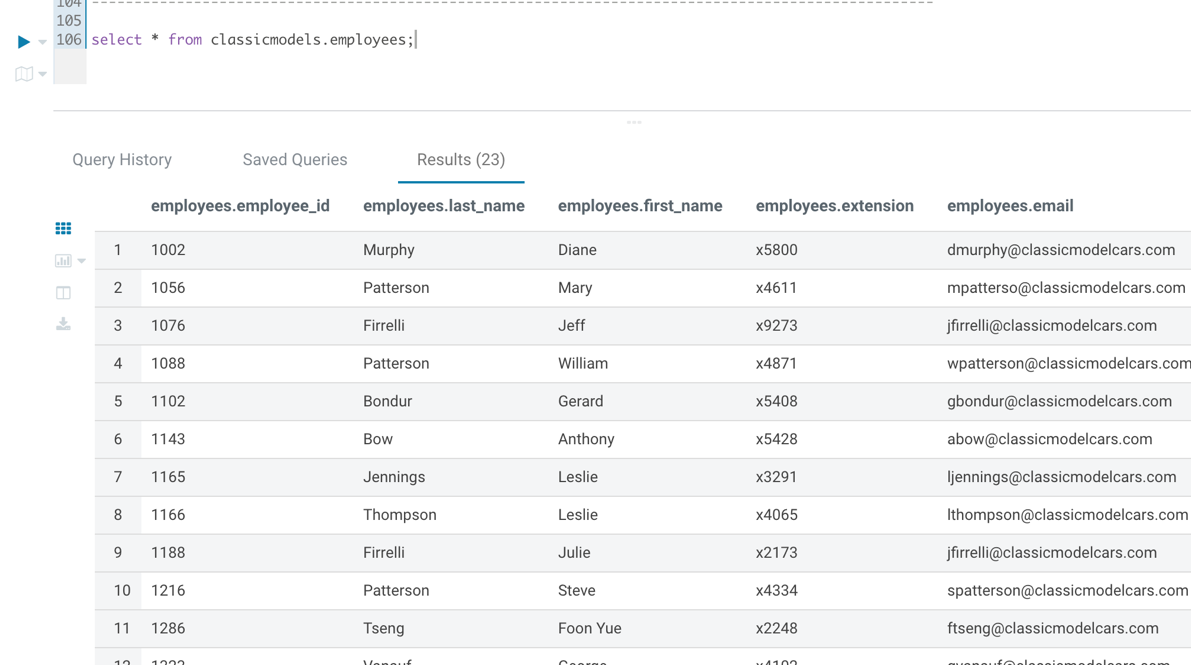
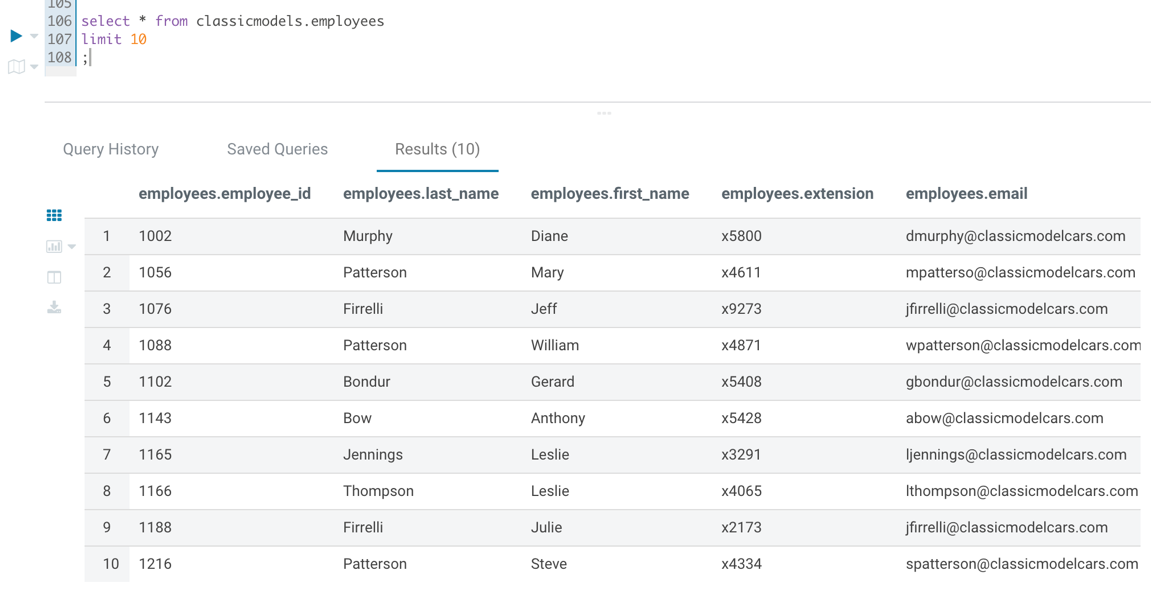
[4. Partitioned Tables (35 points) 6](#_Toc126593968)

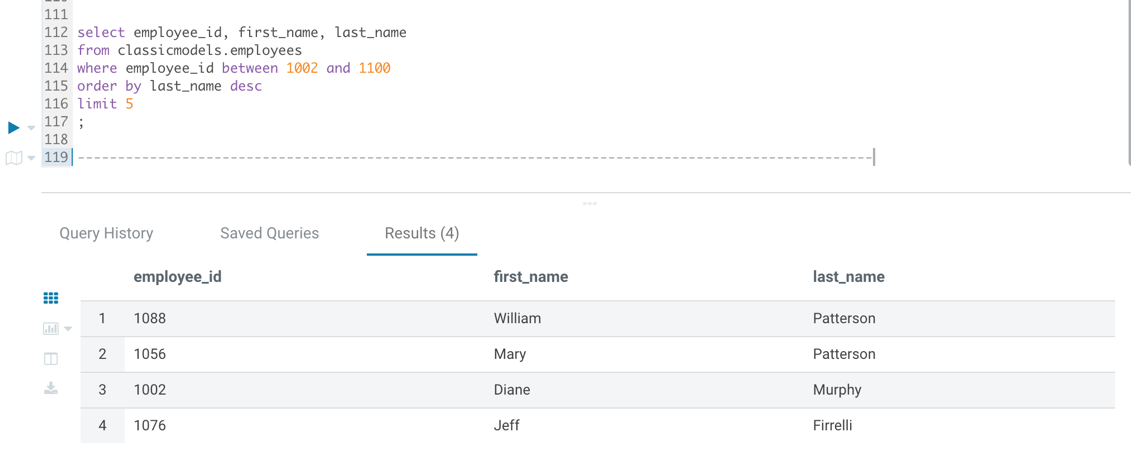
[5. Hive ACID Tables (20 points) 8](#_Toc126593969)

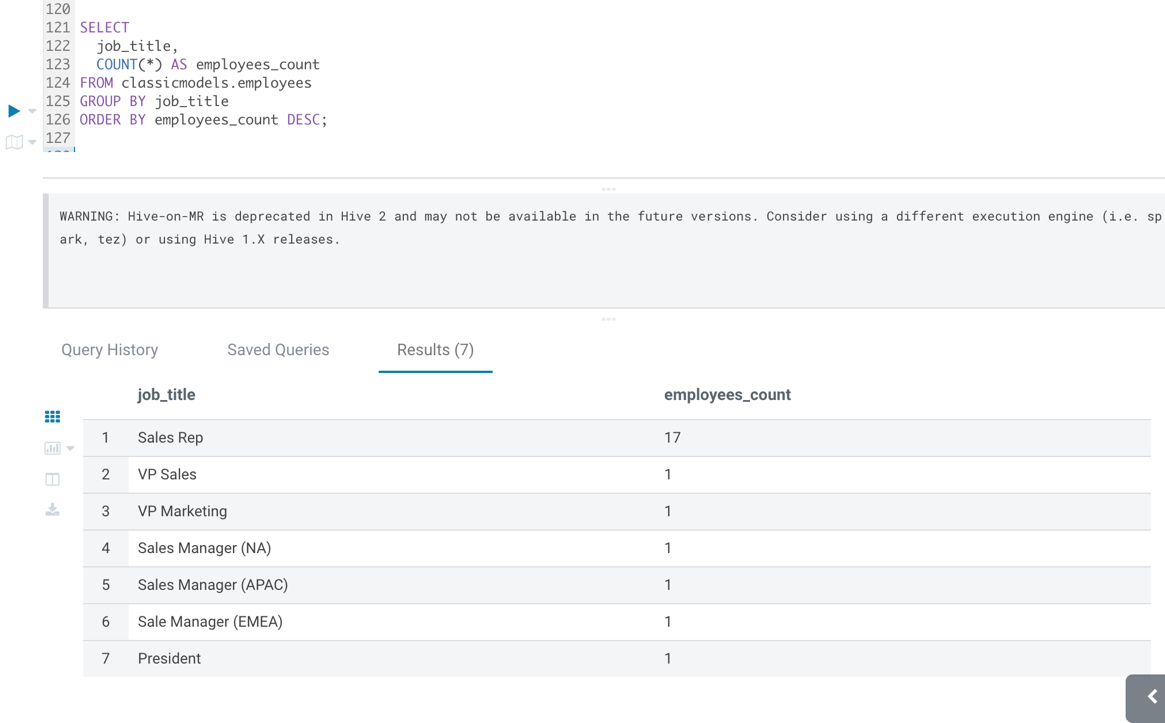
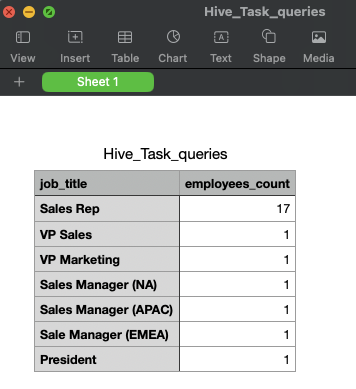
Basic Hive Interaction Using Hue (15 points)

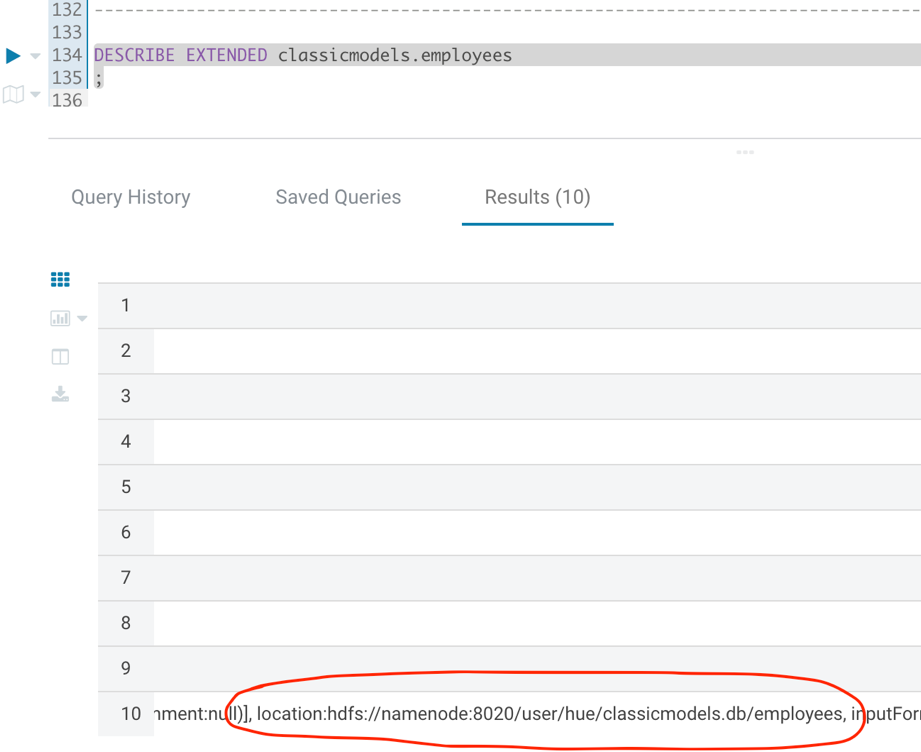
* Connect to Hue using “admin/admin” 
* Browse over to the Hive editor and perform the following:
  + Show available databases under Hive
  + Create the “classicmodels” database and upload the database from the attachment of the Homework module; verify that the separator is considered successfully
  + Switch to it
  + Show all the available tables in the database
  + Expand the “Customers” table and view its columns and data types

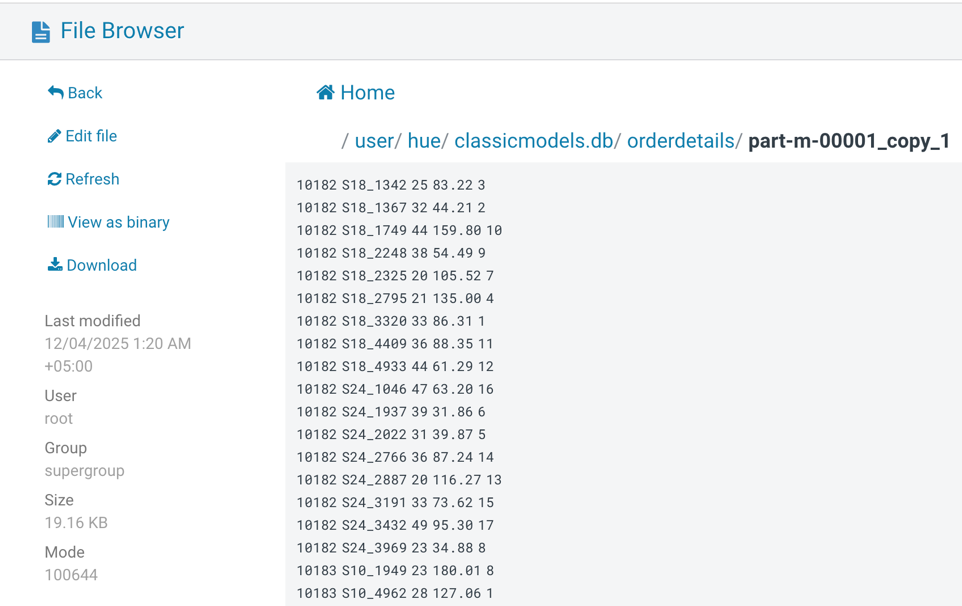
I’ve tried both populating after table creation, as well as populating while creating. Both methods drop all columns data type to string. I found that it is SerDe behaviour, so in general production environment it is better to create staging table with string types and then copy from there to a typed schema table of ORC or Parquet storage. Currently working with untyped tables (all-string tables), but all math operations will work, since Hive manages casting implicitly while comparing.

* Perform the following queries:
  + Query all rows from the “Employees” table
  + Alter the previous query to fetch only the first 10 rows
  + Write a query to fetch the following:
    - The employee ID: first name and last name
    - The employee number should be between 1002 and 1100
    - Order by last name in descending order
    - Fetch only first five rows

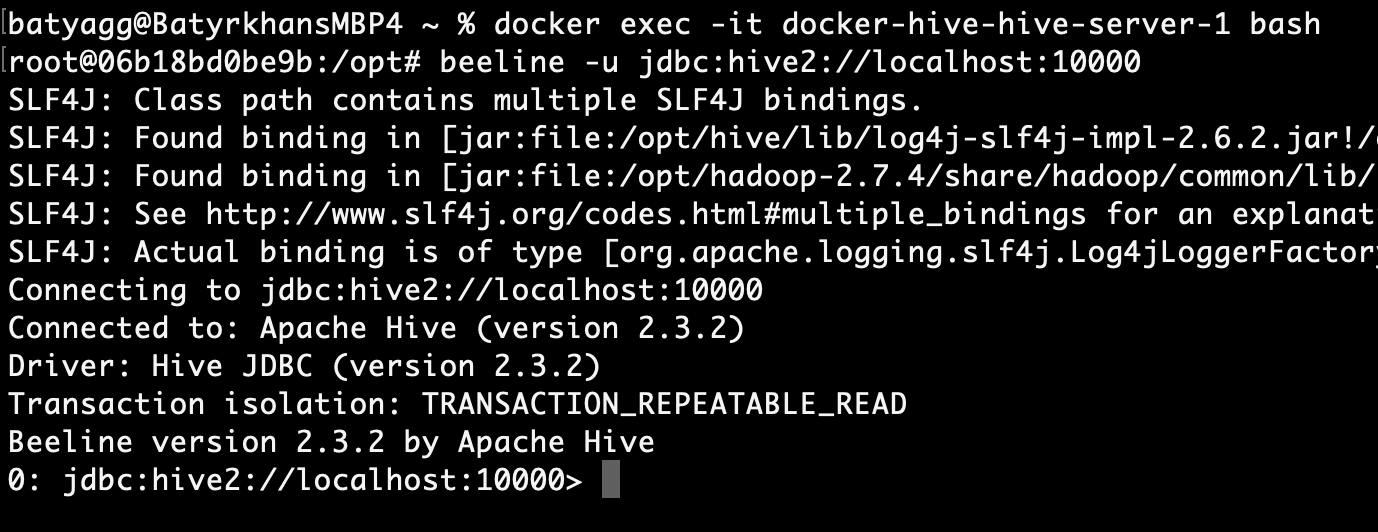
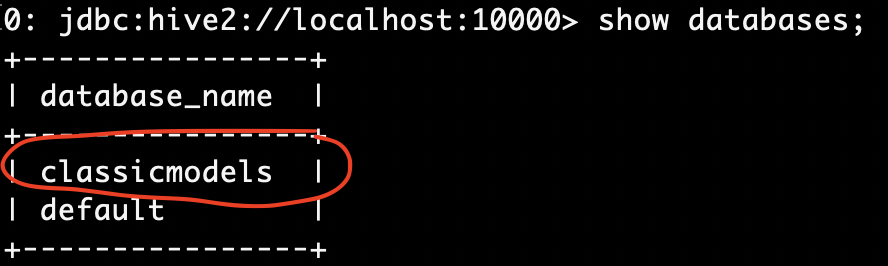
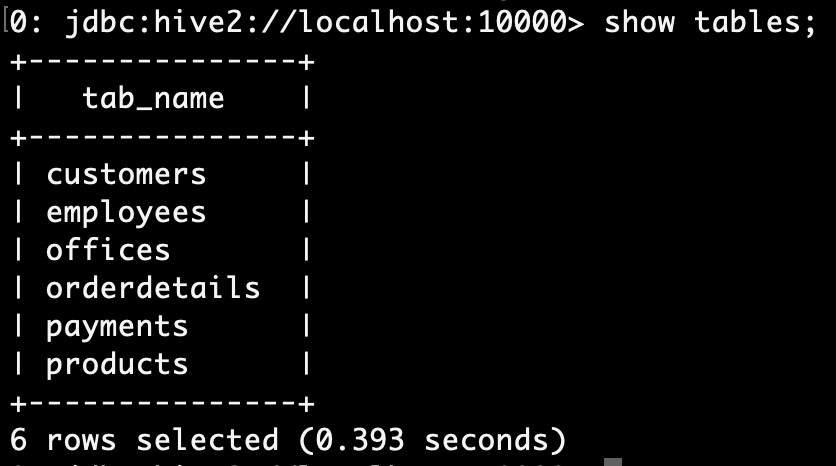
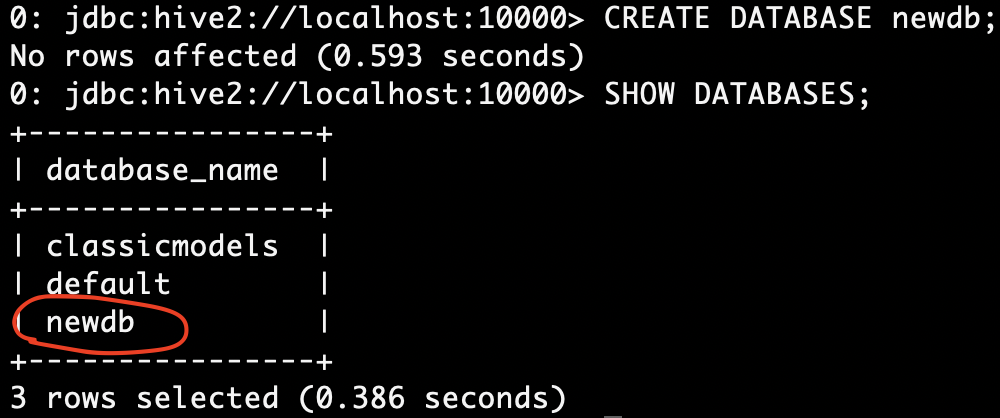
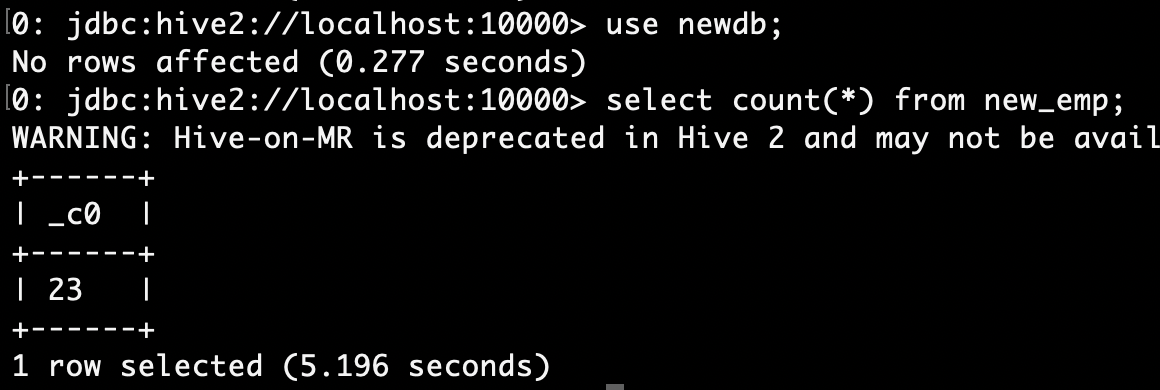


* + Write a query to fetch the number of employees per job title, ordered by number of employees in descending order
  + Export the query output to a text file
* Check which HDFS folder the “employees” table points at

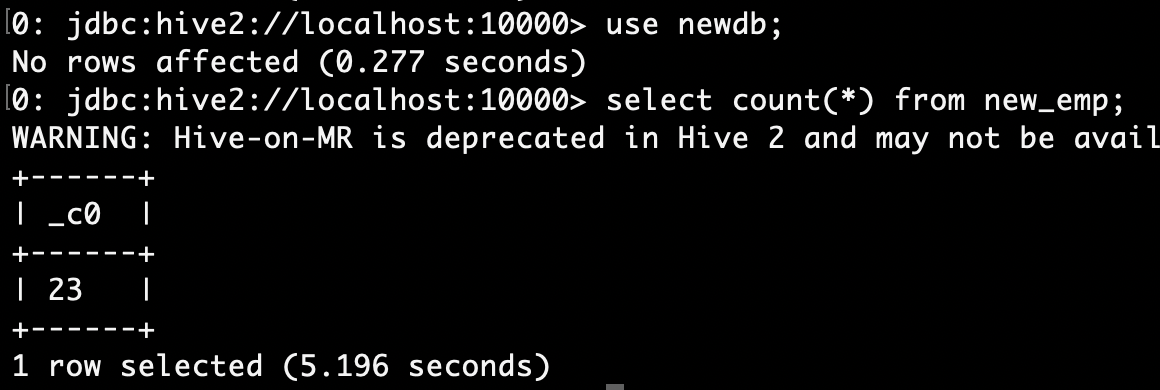
**Hint**: Use the practice guide to see how you can view such details for Hive tables.

* HDFS Browse
  + Use the HDFS browser in Hue to browse over to the HDFS folder and examine its contents
  + Click on one of the files to examine the file contents. Check the following:
    - If the file is human-readable: YES
    - Which Hive table property is responsible for this: ROW FORMAT and SerDe controls how to read files.

Basic Hive Interaction Using Beeline (10 points)

* Open a BASH session to the practice environment and connect to Hive using Beeline
* Show available databases (Verify you see the “ClassisModels” database)
* Switch to use the “ClassisModels” database
* Show all tables in this database
* Create a new database called “newdb” and verify the database was created
* Create a table “new\_emp” in “newdb” identical to the “Employees” table in “ClassicModels” database (Both schema and data), and run a COUNT(\*) to verify the table is populated

Managed and External Tables Using Beeline (20 points)

* Open a BASH session to the practice environment and connect to Hive using Beeline
* Run a COUNT(\*) to verify that “newdb.new\_emp” is in place and populated
* Check using the table properties (without browsing HDFS):

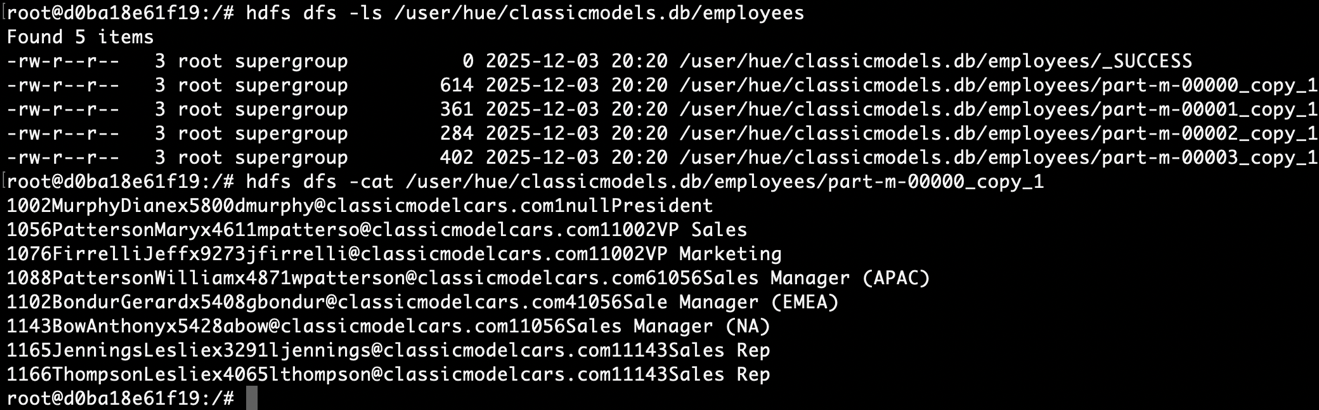


Text in the “Detailed Table Information” is as follows:

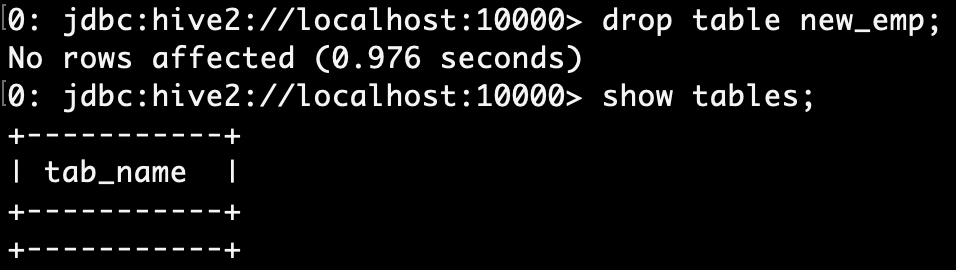
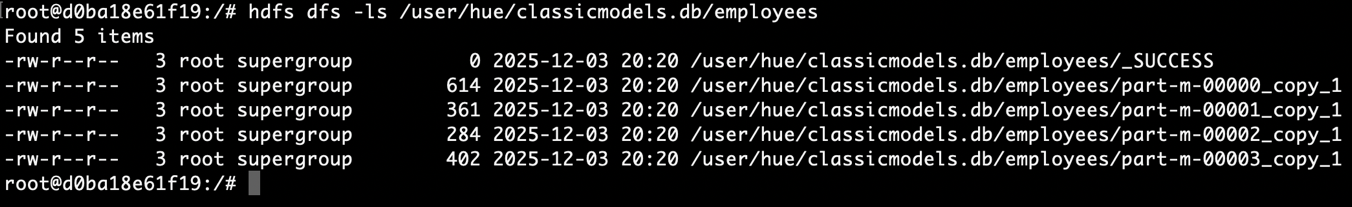
Table(tableName:new\_emp, dbName:newdb, owner:root, createTime:1764860698, lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:[FieldSchema(name:employee\_id, type:int, comment:null), FieldSchema(name:last\_name, type:string, comment:null), FieldSchema(name:first\_name, type:string, comment:null), FieldSchema(name:extension, type:string, comment:null), FieldSchema(name:email, type:string, comment:null), FieldSchema(name:office\_code, type:int, comment:null), FieldSchema(name:reports\_to, type:int, comment:null), FieldSchema(name:job\_title, type:string, comment:null)], location:hdfs://namenode:8020/user/hue/classicmodels.db/employees, inputFormat:org.apache.hadoop.mapred.TextInputFormat, outputFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache.hadoop.hive.serde2.OpenCSVSerde, parameters:{serialization.format=1, separatorChar=}), bucketCols:[], sortCols:[], parameters:{}, skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}), storedAsSubDirectories:false), partitionKeys:[], parameters:{transient\_lastDdlTime=1764860698, totalSize=1661, EXTERNAL=TRUE, numFiles=4}, viewOriginalText:null, viewExpandedText:null, tableType:EXTERNAL\_TABLE, rewriteEnabled:false)

I highlighted parts of the text with answers to the above questions.

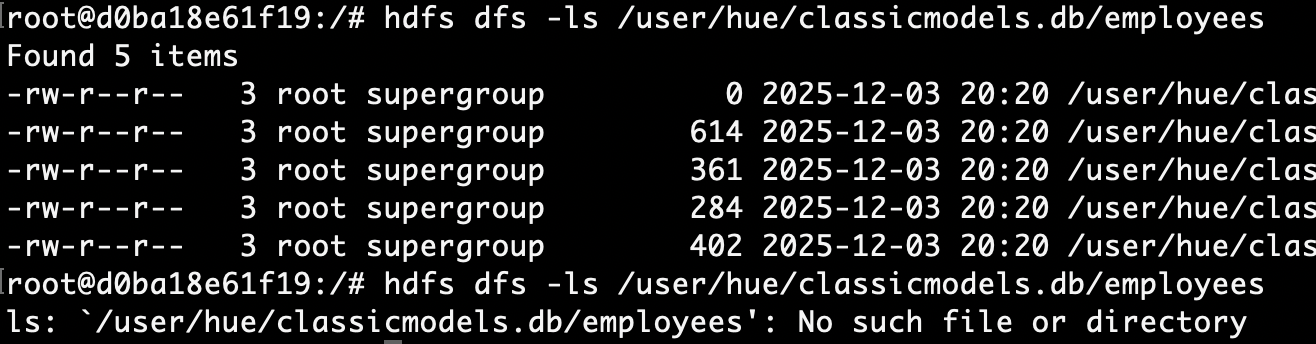
* + Where in HDFS the data for this table is located: location:hdfs://namenode:8020/user/hue/classicmodels.db/employees
  + The file type for this table: Text Type table both input and output (CSV is also considered as Text). We can also check SerDe, which is OpenCSVSerde meaning that the physical files are text csv files.
  + If the table Managed or External? EXTERNAL\_TABLE
  + How many physical data files belong to this table. numFiles=4
* Exit Beeline and use the HDFS CLI to examine the HDFS directory for this table. Do the following:
  + Check how many files there are
  + View the file contents and see if they are readable; explain why

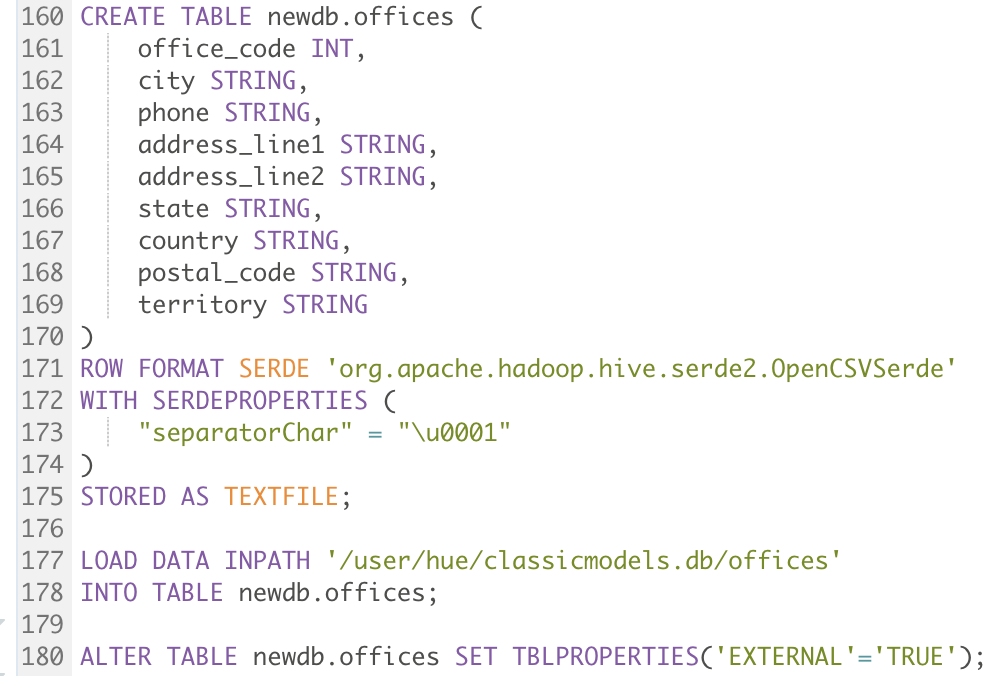


There are 5 files, 1 of which is empty \_SUCCESS file. So there are 4 data files of which the table is constructed, getting schema from them to be stored in metadata. Contents are readable, we could clarify that with -cat command.

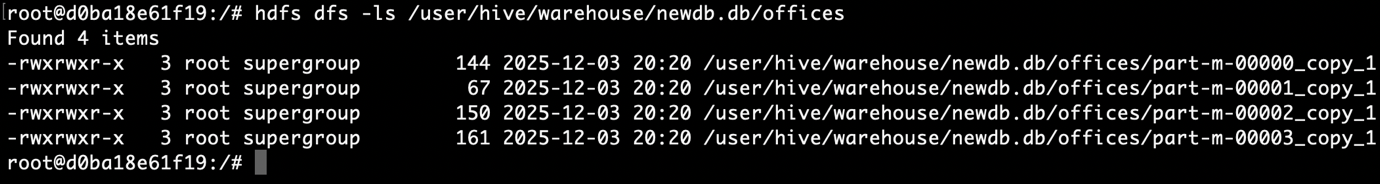
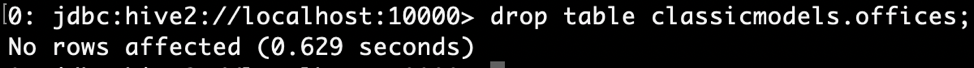
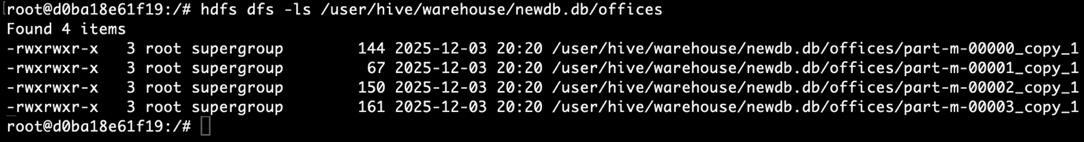
* Go back to Beeline and drop the “new\_emp” tableNo rows affected since it is EXTERNAL one.
* Check in the HDFS CLI again if the HDFS directory and files still exist; explain why Still there, since it was EXTERNAL TABLE. The contents of source directory are not managed by Hive, so it is not altered in any way.
* Go back to Beeline and create the “new\_emp” table again; this time, create it as an EXTERNAL table. Check what error you received and explain why.

I’ve already done the first part of the task using EXTERNAL table. Nothing was written about the type of the table, it was just mentioned to create in “identical to the…” manner. My “identical” way was using EXTERNAL table. That’s why I decided to create INTERNAL table this time to capture the difference.

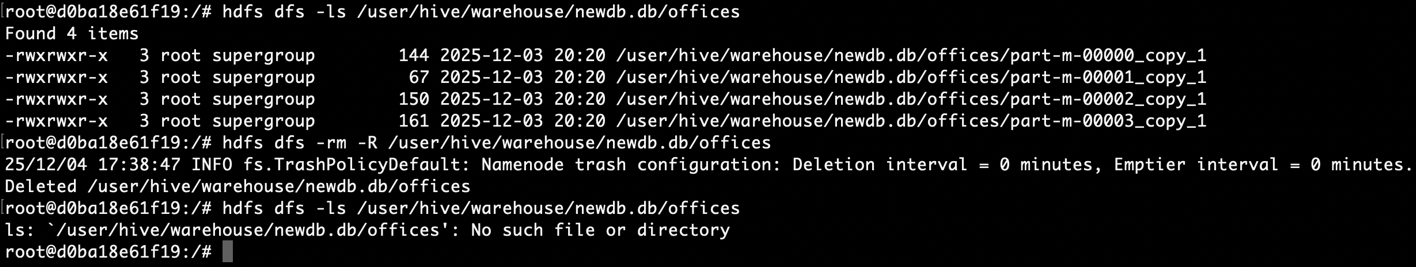
Dropped the table and realized that all files and the folder itself were lost, and that’s why I would have error.

* Create the table as MANAGED (this is the default), and change it manually to EXTERNAL after its creation.

| Detailed Table Information | Table(tableName:offices, dbName:newdb, owner:root, createTime:1764868765, lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:[FieldSchema(name:office\_code, type:int, comment:null), FieldSchema(name:city, type:string, comment:null), FieldSchema(name:phone, type:string, comment:null), FieldSchema(name:address\_line1, type:string, comment:null), FieldSchema(name:address\_line2, type:string, comment:null), FieldSchema(name:state, type:string, comment:null), FieldSchema(name:country, type:string, comment:null), FieldSchema(name:postal\_code, type:string, comment:null), FieldSchema(name:territory, type:string, comment:null)], location:hdfs://namenode:8020/user/hive/warehouse/newdb.db/offices, inputFormat:org.apache.hadoop.mapred.TextInputFormat, outputFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache.hadoop.hive.serde2.OpenCSVSerde, parameters:{serialization.format=1, separatorChar=}), bucketCols:[], sortCols:[], parameters:{}, skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}), storedAsSubDirectories:false), partitionKeys:[], parameters:{totalSize=522, last\_modified\_time=1764869031, numRows=0, rawDataSize=0, EXTERNAL=TRUE, numFiles=4, transient\_lastDdlTime=1764869031, last\_modified\_by=root}, viewOriginalText:null, viewExpandedText:null, tableType:EXTERNAL\_TABLE, rewriteEnabled:false)

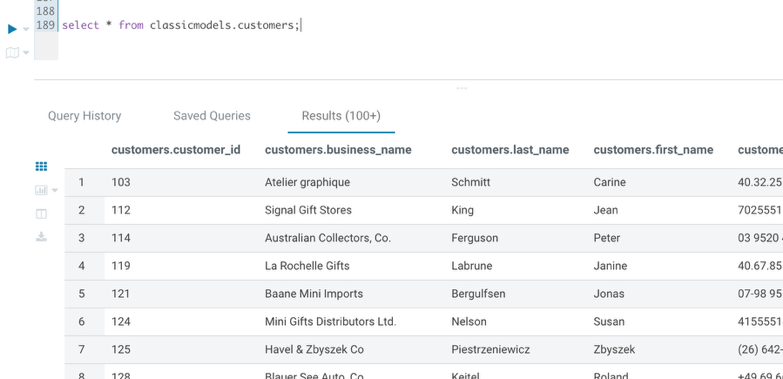
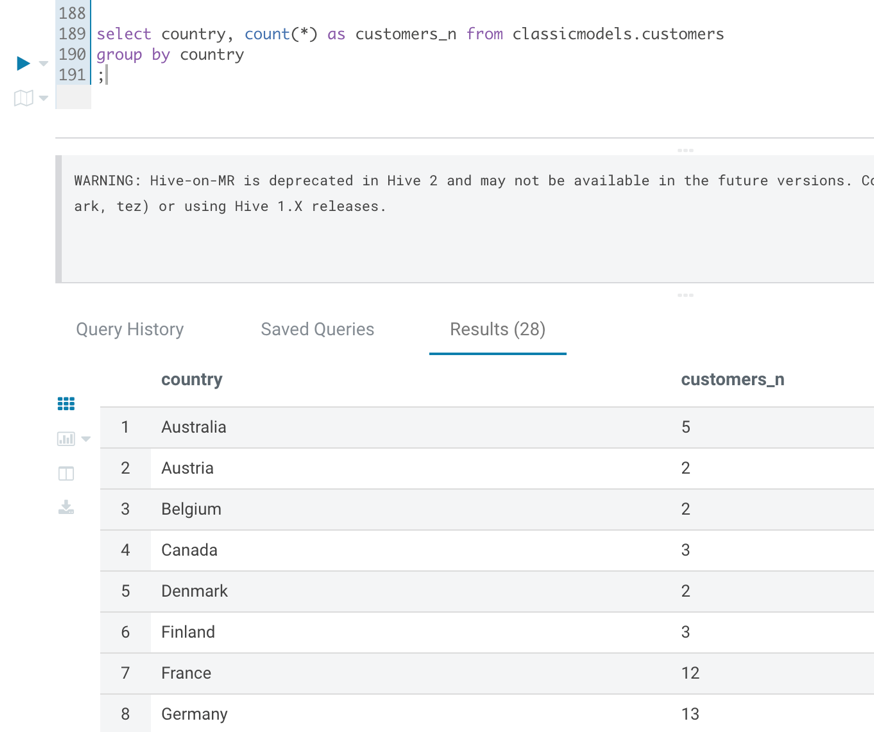
* Check again:
  + Where in HDFS the data for this table is located? In Hive user folder, inside warehouse folder. /user/hive/warehouse/newdb.db/offices
  + The file type for this table. Input: TextInputFormat, Output: HiveIgnoreKeyTextOutputFormat, SerDe: serializationLib:org.apache.hadoop.hive.serde2.OpenCSVSerde
  + If the table Managed or External EXTERNAL\_TABLE
* Exit Beeline and use the HDFS CLI to examine the HDFS directory for this table; check how many files there are. 4 files are there.
* Go back to Beeline, and drop the “new\_emp” table (I’ve created offices table, instead of new\_emp)
* Check again in the HDFS CLI if the HDFS directory and file still exist; explain why

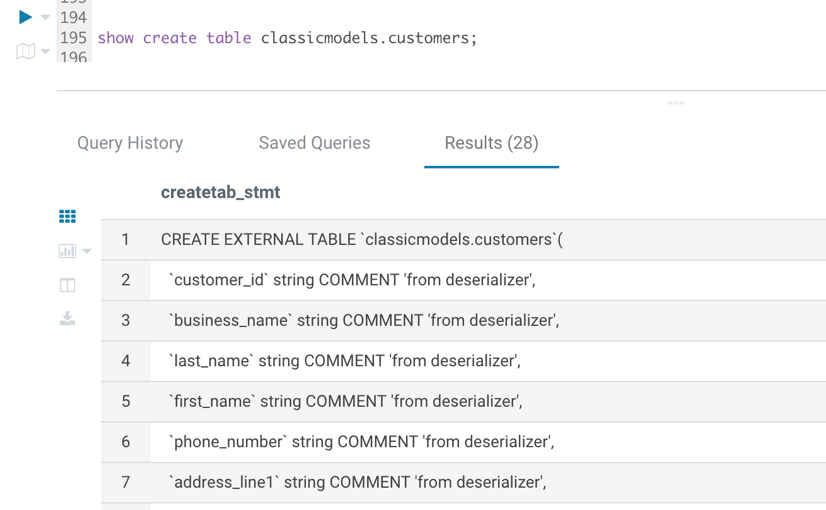
The files are still there, since it is external table (that is the correct behaviour). Even though the files are still in warehouse (which is Hive home folder for data), appropriate records are removed from metadata, and this table and files are not tracked by Hive anymore.

* Remove the HDFS folder manually using the HDFS command CLI (Careful…). and verify that the directory does not exist

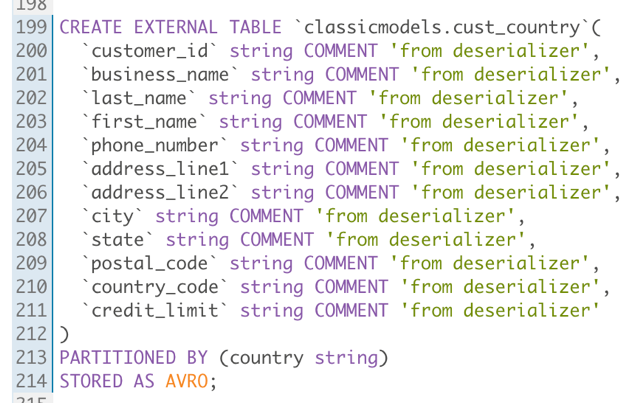
Partitioned Tables (35 points)

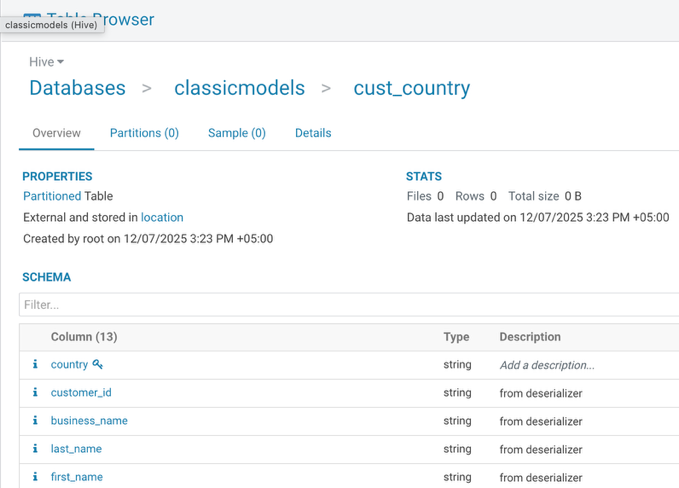
You can complete this task through Hue or Beeline. Use the command prompt to perform HDFS tasks if the HDFS browser is not available in your Hue environment.

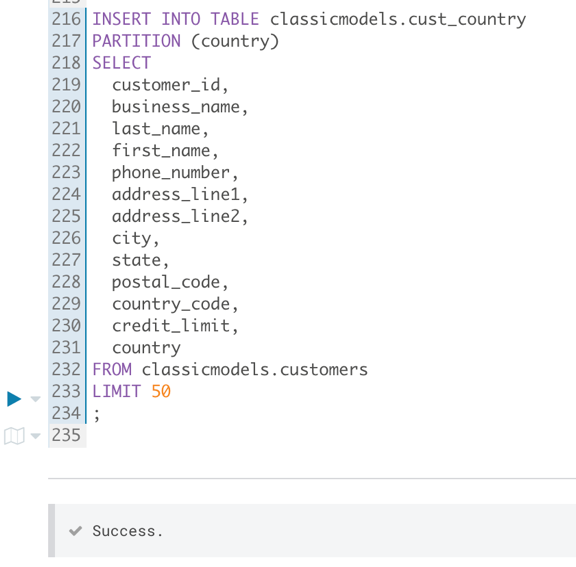
* Query the customers using a simple “SELECT \*” to view the sample data
* Run another query, this time to see the number of customers in each country (GROUP BY..)
* Get the DDL of the customers' table

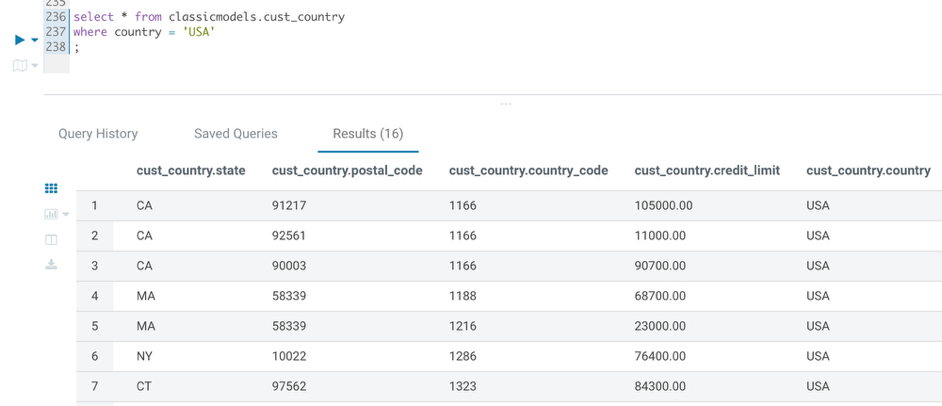


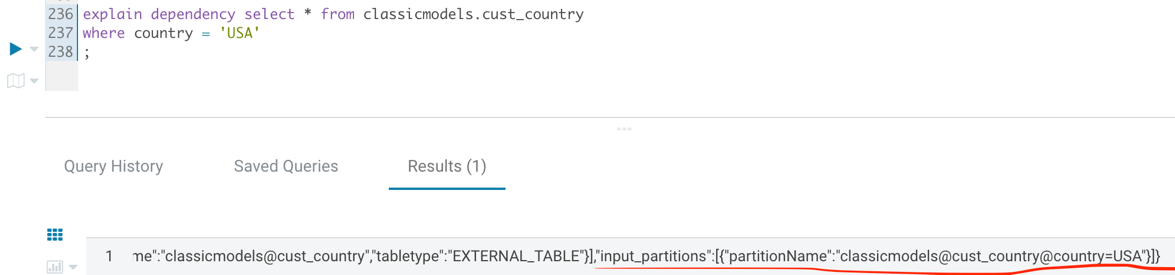
* Copy only the main CREATE TABLE section to a text editor, without all the properties:
  + *CREATE TABLE customers  
    (customernumber int,  
    ..  
    creditlimit double);*
* Modify the CREATE TABLE command, to create a new table with the following characteristics:
  + DB Name: “ClassicModels”
  + Table name: “cust\_country”
  + Partitioned By: “country” column
  + File type: AVRO

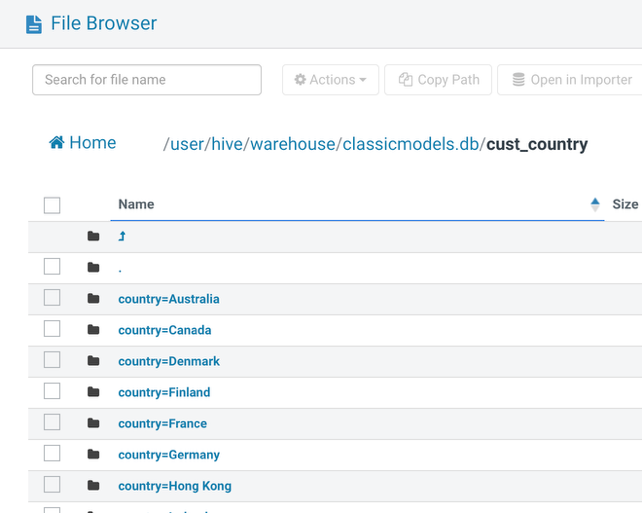


* Verify the table was created properly and view its properties
* Insert data into the “Cust\_Country” from the “Customers” table (Limit to 50 rows) so that partitions will be generated and populated dynamically.



* Run a query from “Cust\_Country” to view all customers from “USA” 
* Examine the execution plan of the query to verify partition elimination has occurred. Answer the following questions:



* + Which EXPLAIN command was required to view partition related details? EXPLAIN DEPENDENCY
  + Why is partitioning so important for query performance? To prune partition paths avoiding unnecessary operations. It dramatically improves performance of queries decreasing the number of unnecessary data reads and transformations by avoiding visiting unnecessary partition folders.
* Go over to HDFS and see the directory structure created for the partitioned table. Answer the following questions:
  + What are the contents of the main directory for this table? Folders for all unique countries in the table.
  + What are the names of the subdirectories? country=*CountryName*

Hive ACID Tables (20 points)

This task can be completed through Hue or Beeline.

**Note**:

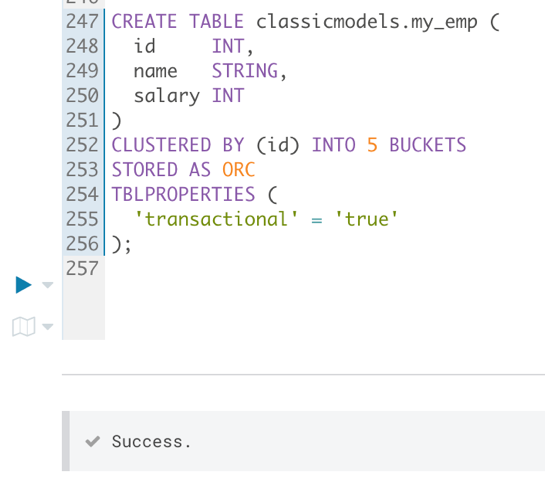
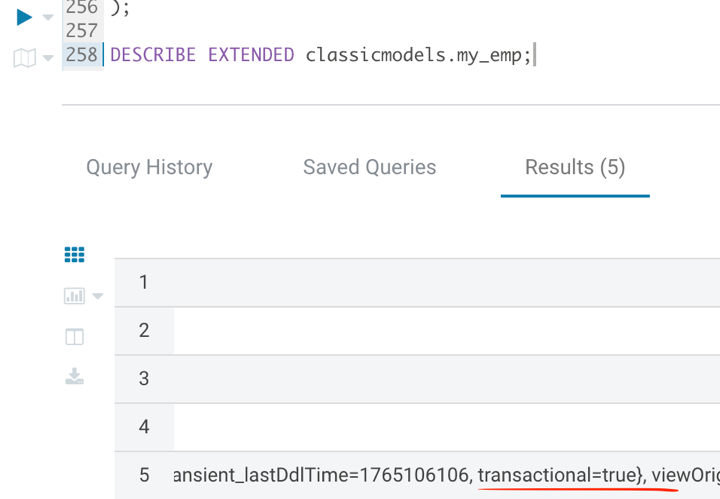
* The practice environment requires setting the following to support transactions:

*SET hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;*

*SET hive.support.concurrency=true;*

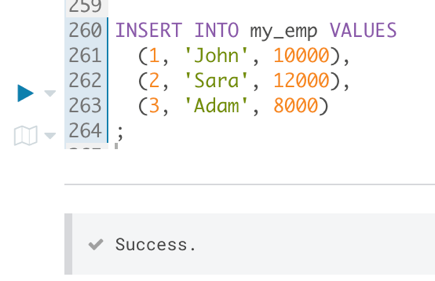
*SET hive.enforce.bucketing=true;*

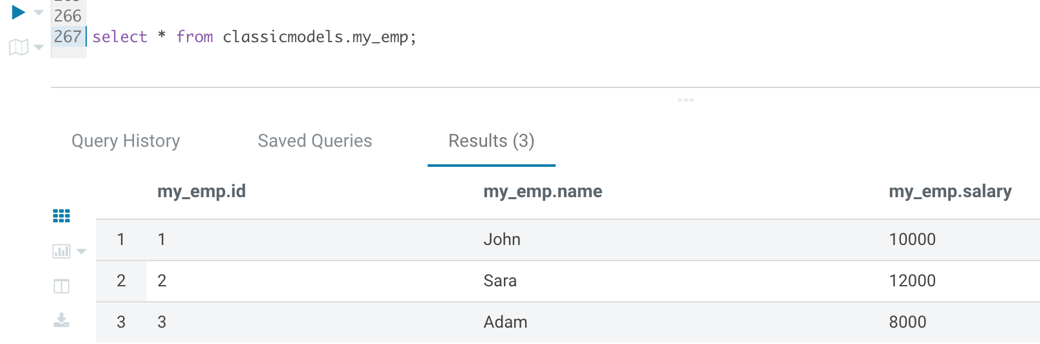
*SET hive.exec.dynamic.partition.mode=nonstrict;*

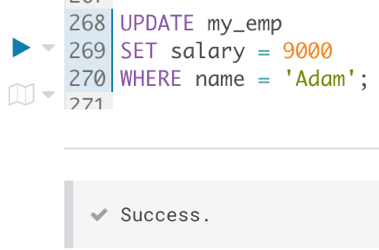
* Please run these SET commands in the Hue/Beeline window prior to performing this exercise.
* Create a new transactional table called “my\_emp” with the following properties:
  + Columns:
    - ID – INT
    - Name – STRING
    - Salary – INT
  + File type: ORC
  + Transactional… 
* Check if this table supports DML operations and which DESCRIBE operation is required Also, it is ORC file storage and bucketed, which are mandatory properties of transactional (ACID) tables in Hive. Below is the full output text of the field.

Table(tableName:my\_emp, dbName:classicmodels, owner:root, createTime:1765106106, lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:[FieldSchema(name:id, type:int, comment:null), FieldSchema(name:name, type:string, comment:null), FieldSchema(name:salary, type:int, comment:null)], location:hdfs://namenode:8020/user/hive/warehouse/classicmodels.db/my\_emp, inputFormat:org.apache.hadoop.hive.ql.io.orc.OrcInputFormat, outputFormat:org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat, compressed:false, numBuckets:5, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache.hadoop.hive.ql.io.orc.OrcSerde, parameters:{serialization.format=1}), bucketCols:[id], sortCols:[], parameters:{}, skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}), storedAsSubDirectories:false), partitionKeys:[], parameters:{totalSize=0, numRows=0, rawDataSize=0, COLUMN\_STATS\_ACCURATE={"BASIC\_STATS":"true"}, numFiles=0, transient\_lastDdlTime=1765106106, transactional=true}, viewOriginalText:null, viewExpandedText:null, tableType:MANAGED\_TABLE, rewriteEnabled:false)

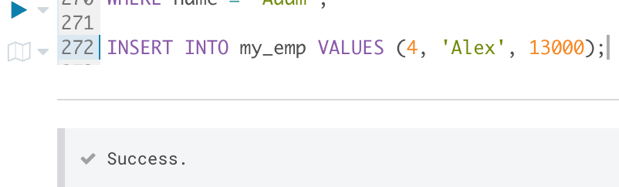
* Insert 3 rows to this table in a single INSERT command:
  + 1, John, 10000
  + 2, Sara, 12000
  + 3, Adam, 8000



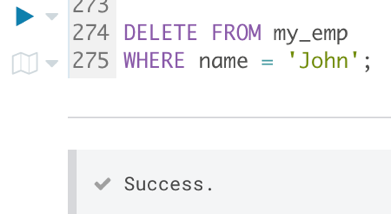
* Query the table to verify all rows were inserted
* Update Adam’s salary in “my\_emp” to 9000

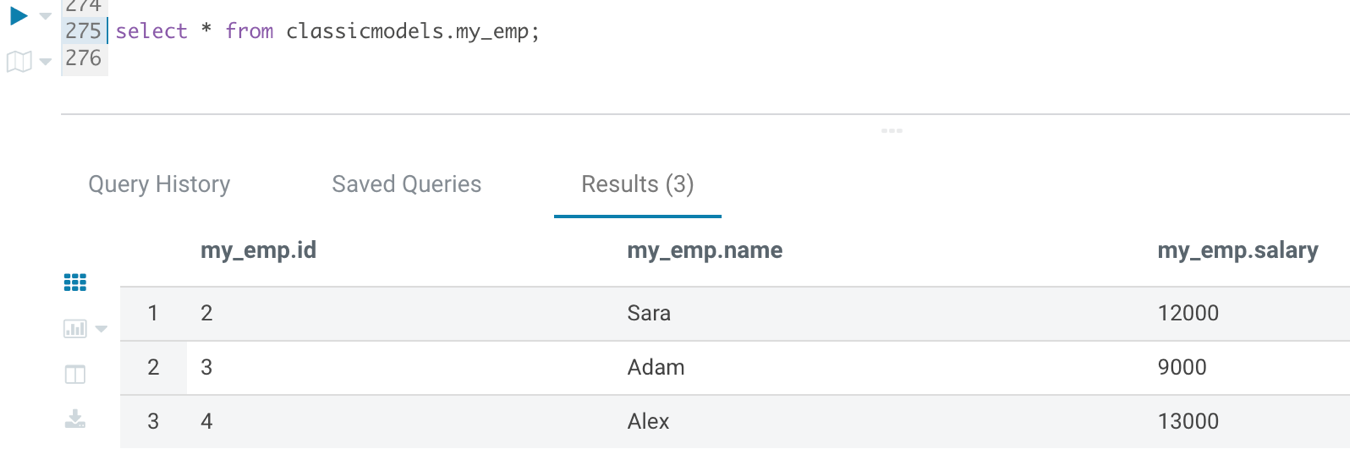


* Insert a new row to “my\_emp”—4, Alex, 13000



* Delete John from the table.



* Query “my\_emp” to verify you see all changes performed.

SQL Queries file used for all tasks

*-- Creating and loading tables*

CREATE EXTERNAL TABLE classicmodels.customers (

customer\_id INT,

business\_name STRING,

last\_name STRING,

first\_name STRING,

phone\_number STRING,

address\_line1 STRING,

address\_line2 STRING,

city STRING,

state STRING,

postal\_code STRING,

country STRING,

country\_code INT,

credit\_limit DECIMAL(15,2)

)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

"separatorChar" = "\u0001"

)

STORED AS TEXTFILE

LOCATION '/user/hue/classicmodels.db/customers';

CREATE EXTERNAL TABLE classicmodels.employees (

employee\_id INT,

last\_name STRING,

first\_name STRING,

extension STRING,

email STRING,

office\_code INT,

reports\_to INT,

job\_title STRING

)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

"separatorChar" = "\u0001"

)

STORED AS TEXTFILE

LOCATION '/user/hue/classicmodels.db/employees';

CREATE EXTERNAL TABLE classicmodels.offices (

office\_code INT,

city STRING,

phone STRING,

address\_line1 STRING,

address\_line2 STRING,

state STRING,

country STRING,

postal\_code STRING,

territory STRING

)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

"separatorChar" = "\u0001"

)

STORED AS TEXTFILE

LOCATION '/user/hue/classicmodels.db/offices';

CREATE EXTERNAL TABLE classicmodels.orderdetails (

order\_number INT,

product\_code STRING,

quantity\_ordered INT,

price\_each DECIMAL(10,2),

order\_line\_number INT

)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

"separatorChar" = "\u0001"

)

STORED AS TEXTFILE

LOCATION '/user/hue/classicmodels.db/orderdetails';

CREATE EXTERNAL TABLE classicmodels.payments (

customer\_number INT,

check\_number STRING,

payment\_date DATE,

amount DECIMAL(10,2)

)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

"separatorChar" = "\u0001"

)

STORED AS TEXTFILE

LOCATION '/user/hue/classicmodels.db/payments';

CREATE EXTERNAL TABLE classicmodels.products (

product\_code STRING,

product\_name STRING,

product\_line STRING,

product\_scale STRING,

product\_vendor STRING,

product\_description STRING,

quantity\_in\_stock INT,

buy\_price DECIMAL(10,2),

msrp DECIMAL(10,2)

)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

"separatorChar" = "\u0001"

)

STORED AS TEXTFILE

LOCATION '/user/hue/classicmodels.db/products';

*---------------------------------------------------------------------------------------------------*

select \* from classicmodels.employees

limit 10

;

*---------------------------------------------------------------------------------------------------*

select employee\_id, first\_name, last\_name

from classicmodels.employees

where employee\_id between 1002 and 1100

order by last\_name desc

limit 5

;

*---------------------------------------------------------------------------------------------------*

SELECT

job\_title,

COUNT(\*) AS employees\_count

FROM classicmodels.employees

GROUP BY job\_title

ORDER BY employees\_count DESC

;

*---------------------------------------------------------------------------------------------------*

SHOW CREATE TABLE classicmodels.employees;

*---------------------------------------------------------------------------------------------------*

DESCRIBE EXTENDED classicmodels.employees;

*---------------------------------------------------------------------------------------------------*

select \* from newdb.new\_emp;

*---------------------------------------------------------------------------------------------------*

CREATE TABLE newdb.new\_emp (

employee\_id INT,

last\_name STRING,

first\_name STRING,

extension STRING,

email STRING,

office\_code INT,

reports\_to INT,

job\_title STRING

)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

"separatorChar" = "\u0001"

)

STORED AS TEXTFILE

LOCATION '/user/hue/classicmodels.db/employees';

CREATE TABLE newdb.offices (

office\_code INT,

city STRING,

phone STRING,

address\_line1 STRING,

address\_line2 STRING,

state STRING,

country STRING,

postal\_code STRING,

territory STRING

)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

"separatorChar" = "\u0001"

)

STORED AS TEXTFILE;

LOAD DATA INPATH '/user/hue/classicmodels.db/offices'

INTO TABLE newdb.offices;

ALTER TABLE newdb.offices SET TBLPROPERTIES('EXTERNAL'='TRUE');

DROP TABLE newdb.offices;

DESCRIBE EXTENDED newdb.offices;

*---------------------------------------------------------------------------------------------------*

select country, count(\*) as customers\_n from classicmodels.customers

group by country

;

*---------------------------------------------------------------------------------------------------*

set hive.exec.dynamic.partition.mode=nonstrict;

show create table classicmodels.customers;

CREATE EXTERNAL TABLE `classicmodels.cust\_country`(

`customer\_id` string COMMENT 'from deserializer',

`business\_name` string COMMENT 'from deserializer',

`last\_name` string COMMENT 'from deserializer',

`first\_name` string COMMENT 'from deserializer',

`phone\_number` string COMMENT 'from deserializer',

`address\_line1` string COMMENT 'from deserializer',

`address\_line2` string COMMENT 'from deserializer',

`city` string COMMENT 'from deserializer',

`state` string COMMENT 'from deserializer',

`postal\_code` string COMMENT 'from deserializer',

`country\_code` string COMMENT 'from deserializer',

`credit\_limit` string COMMENT 'from deserializer'

)

PARTITIONED BY (country string)

STORED AS AVRO;

INSERT INTO TABLE classicmodels.cust\_country

PARTITION (country)

SELECT

customer\_id,

business\_name,

last\_name,

first\_name,

phone\_number,

address\_line1,

address\_line2,

city,

state,

postal\_code,

country\_code,

credit\_limit,

country

FROM classicmodels.customers

LIMIT 50

;

explain dependency select \* from classicmodels.cust\_country

where country = 'USA'

;

*---------------------------------------------------------------------------------------------------*

SET hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;

SET hive.support.concurrency=true;

SET hive.enforce.bucketing=true;

SET hive.exec.dynamic.partition.mode=nonstrict;

CREATE TABLE classicmodels.my\_emp (

id INT,

name STRING,

salary INT

)

CLUSTERED BY (id) INTO 5 BUCKETS

STORED AS ORC

TBLPROPERTIES (

'transactional' = 'true'

);

DESCRIBE EXTENDED classicmodels.my\_emp;

INSERT INTO my\_emp VALUES

(1, 'John', 10000),

(2, 'Sara', 12000),

(3, 'Adam', 8000)

;

UPDATE my\_emp

SET salary = 9000

WHERE name = 'Adam';

INSERT INTO my\_emp VALUES (4, 'Alex', 13000);

DELETE FROM my\_emp

WHERE name = 'John';

select \* from classicmodels.my\_emp;

DESCRIBE EXTENDED classicmodels.offices;

*---------------------------------------------------------------------------------------------------*