Lab: Hive Management

Managing & Using Hive/HiveQL

Lab: Hive Management

1. Table of Contents

2. Accessing Hive With Beeline

The goal of this lab is to demonstrate the usage of Beeline to connect to Hive

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Launch beeline

```
beeline -u jdbc:hive2://master1.cluster:10000/default -n admin
```

3. Lets run a simple query

```
SELECT count(1) FROM zomato;
```

3. Accessing Hive With Squirrel SQL

The goal of this lab is to install Squirrel SQL and use it to connect to Hive database

1. Dependency:
   - Download JRE : https://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html
   - Download SquirrelSQL : http://squirrel-sql.sourceforge.net/#installation
   - Download Hive JDBC 4.1 driver : https://hortonworks.com/downloads/
   - Ensure the VM have port forwarding configured for port 10000 and 10500

Lets install Squirrel SQL and Hive2 driver:

1. On your main host (outside the VM), install JRE and then run Squirrel SQL jar
2. Extract JDBC driver JAR from the zip file, and save it in SquirrelSQL's lib directory
3. Launch SquirrelSQL, and click "Drivers" tab on the left
4. Click + button on the left bar to register a new driver
   - Name: HiveServer2
   - Example URL: jdbc:hive2://localhost:10000/default
   - Extra Class Path: add the HiveJDBC4.1.jar into the path list, select it, then click List Drivers to update list of Class Name
   - Class Name: com.simba.hive.jdbc41.HS2Driver
5. Click OK to Save

Lab: Hive Management

Lets create a new connection alias to Hive

1. Launch SquirrelSQL and click Aliases tab on the left
2. Click + button to add new alias
    - Name: Sandbox Hive
    - Driver: HiveServer2
    - URL: jdbc:hive2://localhost:10000/default
    - User Name: admin
3. Click Test to test the driver for connectivity and if it function correctly, click OK to save.

Lets connect to Hive

1. Launch SquirrelSQL and click Aliases tab on the left
2. Select Sandbox Hive, right click, connect
3. On the SQL tab, enter:

```
SELECT count(1) FROM zomato;
```

4. Select the statement and Ctrl+Enter to execute

4. Accessing Hive With Python

The goal of this lab is to demonstrate how to access Hive programmatically through Python.

1. Launch Cockpit and SSH to edge node and switch to admin

```
ssh root@edge.cluster
su - admin
```

2. Launch the python shell

```
python
```

3. Run the following code in the shell

```
from sqlalchemy import create_engine
engine = create_engine('hive://admin@master1.cluster:10000/default')
result = engine.execute('select count(1) from zomato')
print(r.fetchall())
```

5. Hive Access Control With Ranger

The goal of this lab is to demonstrate the process of setting Hive table access control with Ranger:

Configure access control

1. Login to Ambari as admin
2. Navigate to Ranger service and through Quick Links, navigate to Ranger Admin UI
3. Login to Ranger Admin UI
4. Under Hive, click SandboxCluster_hive service
5. Lets allow user 'jeff' to access zomato table on database default, click Add New Policy
   - Policy Name: Allow jeff to zomato
   - Database: default
   - Table: zomato
   - Hive Column: *
   - Permissions: select, read
6. Click Save

Lets try to select some data as user jeff

1. Login to Cockpit and ssh to root@edge.cluster

```
ssh root@edge.cluster
su - admin
```

2. Using beeline, connect to Hive as user jeff

```
beeline -u jdbc:hive2://master1.cluster:10000/default -n jeff
```

3. In the shell, lets run following queries

```
show tables;
select count(1) from zomato;
select * from zomato_country_code;
```

6. Setting Up Hive LLAP

The goal of this lab activity is to demonstrate the process of enabling LLAP

1. Login to Ambari as admin
2. Navigate to YARN Queue Manager and lets adjust some parameters to give more resources to llap queue
   - Reduce 'demo' queue to have only 15% cluster allocation
   - Increase queue 'llap' to have 75% minimum allocation
3. Navigate to Ranger Admin UI, and create a new policy for YARN
   - Policy Name: hive llap
   - Queue: root.llap
   - User: hive
   - Permission: submit-app
4. Navigate to Hive settings, and enable Interactive Query
5. Select to run HiveServer2 Interactive component in master1.cluster
6. Select 'llap' in Interactive Query Queue
7. Under Advanced tab, set
   - "Number of Nodes for running Hive LLAP daemon" to 1
   - "Number of executors per LLAP daemon" to 2
8. Click Save, and accept the default recommendations.
9. Save and refresh queue
10. Restart/start any necessary services

Lab: Hive Management

Lets try to select some data

1. Login to Cockpit and ssh to root@edge.cluster

```
ssh root@edge.cluster
su - admin
```

2. Using beeline, connect to Hive LLAP

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin
```

3. In the shell, lets run following queries

```
show tables;
select count(1) from zomato;
select * from zomato_country_code;
```

4. Run the above queries again to see the speed & caching improvement added by LLAP.

7. Creating Tables With Hive DDL

The goal of this lab is to demonstrate the DDL for different types of tables on Hive

Before we start, lets enable Hive ACID so that we can create ACID tables:

1. Login to Ambari as admin
2. Navigate to Hive configuration and enable ACID Transactions
3. Restart necessary services
4. We will be using Hive LLAP for this lab. So ensure that it is up.

Lab: Hive Management

7.1. Basic Hive Managed Table

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Connect to Hive LLAP through beeline

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin
```

3. Lets create a simple demo table, stored as TextFile

```
CREATE TABLE demo_basic (
    key int,
    value string )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

4. Lets insert some values

```
INSERT INTO TABLE demo1 VALUES (1, 'hello'), (2, 'world');
SELECT * FROM demo_basic;
```

5. Press CTRL+d to exit beeline.

6. Lets check the contents of /apps/hive/warehouse

```
hdfs dfs -ls /apps/hive/warehouse/
hdfs dfs -ls /apps/hive/warehouse/demo_basic/
```

7. Lets see the content of the first file

```
hdfs dfs -cat /apps/hive/warehouse/demo_basic/000000_0
```

8. Lets insert some more data

```
beeline -u jdbc:hive2://master1.cluster:10500/default \
    -n admin -e "INSERT INTO TABLE demo_basic VALUES (3, 'foo'), (4, 'bar')";
```

9. Lets see what happened in the background

```
hdfs dfs -ls /apps/hive/warehouse/demo_basic/
hdfs dfs -cat /apps/hive/warehouse/demo_basic/000000_0
hdfs dfs -cat /apps/hive/warehouse/demo_basic/000000_0_copy_1
```

7.2. Partitioned Table

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Connect to Hive LLAP through beeline

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin
```

3. Lets create a simple partitioned demo table, stored as TextFile

```
CREATE TABLE demo_partition (
    key int,
    value string )
PARTITIONED BY (dt string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

4. Lets insert some values

```
INSERT INTO TABLE demo_partition PARTITION (dt='2018-01-01')
       VALUES (1, 'hello'), (2, 'world');
INSERT INTO TABLE demo_partition PARTITION (dt='2018-01-02')
       VALUES (3, 'foo'), (4, 'bar');
SELECT * FROM demo_partition;
```

5. Press CTRL+d to exit beeline.

6. Lets check the contents of /apps/hive/warehouse. Notice the partition based directory structure

```
hdfs dfs -ls /apps/hive/warehouse/
hdfs dfs -ls /apps/hive/warehouse/demo_partition/
hdfs dfs -ls /apps/hive/warehouse/demo_partition/dt=2018-01-01/
hdfs dfs -cat /apps/hive/warehouse/demo_partition/dt=2018-01-01/000000_0
```

7.3. Bucketed Table

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Connect to Hive LLAP through beeline

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin
```

3. Lets create a simple demo table, stored as TextFile

```
CREATE TABLE demo_bucket (
    key int,
    value string )
CLUSTERED BY (key) INTO 2 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

4. Lets insert some values

```
INSERT INTO TABLE demo_bucket VALUES
        (1, 'hello'), (2, 'world'), (3, 'foo'), (4, 'bar');
SELECT * FROM demo_bucket;
```

5. Press CTRL+d to exit beeline.

6. Lets check the contents of /apps/hive/warehouse

```
hdfs dfs -ls /apps/hive/warehouse/
hdfs dfs -ls /apps/hive/warehouse/demo_bucket/
```

7. Lets see the contents

```
hdfs dfs -cat /apps/hive/warehouse/demo_bucket/000000_0
hdfs dfs -cat /apps/hive/warehouse/demo_bucket/000001_0
```

8. Lets insert some more data

```
beeline -u jdbc:hive2://master1.cluster:10500/default \
    -n admin -e "INSERT INTO TABLE demo_bucket VALUES (5, 'fu'), (6, 'baz')";
```

9. Lets see what happened in the background

```
hdfs dfs -ls /apps/hive/warehouse/demo_bucket/
hdfs dfs -cat /apps/hive/warehouse/demo_bucket/000000_0
hdfs dfs -cat /apps/hive/warehouse/demo_bucket/000000_0_copy_1
hdfs dfs -cat /apps/hive/warehouse/demo_bucket/000001_0
hdfs dfs -cat /apps/hive/warehouse/demo_bucket/000001_0_copy_1
```

7.4. Basic Table with ORC Storage

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Connect to Hive LLAP through beeline

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin
```

3. Lets create a table, using ORC store

```
CREATE TABLE demo_basic_orc (
    key int,
    value string )
STORED AS ORC;
```

4. Lets insert some values

```
INSERT INTO TABLE demo_basic_orc VALUES (1, 'hello'), (2, 'world');
SELECT * FROM demo_basic_orc;
```

5. Press CTRL+d to exit beeline.

6. Lets check the contents of /apps/hive/warehouse

```
hdfs dfs -ls /apps/hive/warehouse/
hdfs dfs -ls /apps/hive/warehouse/demo_basic_orc/
```

7. Lets see the content of the first file

```
hdfs dfs -cat /apps/hive/warehouse/demo_basic_orc/000000_0 | hexdump -C
```

8. Lets insert some more data

```
beeline -u jdbc:hive2://master1.cluster:10500/default \
    -n admin -e "INSERT INTO TABLE demo_basic_orc VALUES (3, 'foo'), (4, 'bar')";
```

9. Lets see what happened in the background

```
hdfs dfs -ls /apps/hive/warehouse/demo_basic_orc/
hdfs dfs -cat /apps/hive/warehouse/demo_basic_orc/000000_0 | hexdump -C
hdfs dfs -cat /apps/hive/warehouse/demo_basic_orc/000000_0_copy_1 | hexdump -C
```

7.5. ACID Table

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Connect to Hive LLAP through beeline

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin
```

3. Lets create a table, using ORC store with ACID

```
CREATE TABLE demo_acid (
    key int,
    value string )
CLUSTERED BY (key) INTO 2 BUCKETS
STORED AS ORC
TBLPROPERTIES ( 'transactional' = 'true');
```

4. Lets insert some values

```
INSERT INTO TABLE demo_acid VALUES (1, 'hello'), (2, 'world');
SELECT * FROM demo_acid;
```

5. Press CTRL+d to exit beeline.

6. Lets check the contents of /apps/hive/warehouse

```
hdfs dfs -ls /apps/hive/warehouse/
hdfs dfs -ls /apps/hive/warehouse/demo_acid/
hdfs dfs -ls /apps/hive/warehouse/demo_acid/delta_0000001_0000001_0000/
```

7. Lets insert some more data

```
beeline -u jdbc:hive2://master1.cluster:10500/default \
    -n admin -e "INSERT INTO TABLE demo_acid VALUES (3, 'foo'), (4, 'bar')";
```

8. Lets update some values

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin

UPDATE demo_acid SET value = 'foobar' where key=3;
SELECT * FROM demo_acid where key=3;
```

7.6. Create Table As Select

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Connect to Hive LLAP through beeline

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin
```

3. Lets create a copy of demo_basic, but convert it to ORC using CTAS

```
CREATE TABLE demo_basic_orcconvert AS SELECT * FROM demo_basic;
```

8. Accessing HCatalog Table Through Pig

The goal of this lab is to demonstrate accessing a table in Hive / Catalog using Pig.

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Launch grunt shell with HCatalog support

```
pig -useHCatalog
```

3. In grunt shell, execute following code:

```
A = LOAD 'zomato' USING org.apache.hive.hcatalog.pig.HCatLoader();
B = LIMIT A 5;
DUMP B;
```

9. Accessing HCatalog Table Using Spark

The goal of this lab is to demonstrate accessing a table in Hive / Catalog using PySpark.

4. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

5. Launch pyspark shell

```
pyspark
```

6. In pyspark, execute following code:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("example app").getOrCreate()
df = spark.sql('SELECT * FROM zomato LIMIT 5')
df.show()
```

10. Managing And Querying Data In Hive

The goal of this lab is to demonstrate common data loading and manipulation activities.

10.1. Loading Data Into Hive & Querying Complex Table

Lets create a table in Hive, using JSONL data and load some data in

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Download a sample mock dataset and upload to HDFS

```
wget http://repo.kagesenshi.org/hdptraining/people.jsonl
hdfs dfs -put people.jsonl /tmp/
hdfs dfs -chmod a+rwx /tmp/people.jsonl
```

3. Launch beeline

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin
```

4. Lets create a table to store the dataset

```
CREATE TABLE people_json (
    customer STRUCT<first_name:string, last_name:string, address:string>,
     address STRUCT<street:string, postcode:string, state:string>
)
ROW FORMAT SERDE   'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE;
LOAD DATA INPATH '/tmp/people.jsonl' INTO TABLE people_json;
```

5. Lets select some data

```
SELECT * FROM people_json LIMIT 5;
SELECT customer.first_name FROM people_json LIMIT 5;
SELECT address.postcode FROM people_json LIMIT 5;
SELECT customer.last_name,customer.first_name FROM people_json LIMIT 5;
```

10.2. Querying JSON string using GET_JSON_OBJECT

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Upload another copy of people.jsonl into HDFS

```
hdfs dfs -put people.jsonl /tmp/
hdfs dfs -chmod a+rwx /tmp/people.jsonl
```

3. Launch beeline

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin
```

4. Lets create a table to store the dataset

```
CREATE TABLE people_jsonstring (
    json_data string
)
STORED AS TEXTFILE;
LOAD DATA INPATH '/tmp/people.jsonl' INTO TABLE people_jsonstring;
```

5. Lets select some data

```
SELECT * FROM people_jsonstring LIMIT 5;
SELECT GET_JSON_OBJECT(json_data, '$.customer.first_name')
    FROM people_jsonstring LIMIT 5;
SELECT GET_JSON_OBJECT(json_data, '$.customer.address')
    FROM people_jsonstring LIMIT 5;
SELECT
    GET_JSON_OBJECT(json_data, '$.customer.last_name'),
    GET_JSON_OBJECT(json_data, '$.customer.first_name')
FROM people_jsonstring LIMIT 5;
```

11. Using Streaming Join and Map Side Join

The goal of this lab is to demonstrate the process of using Streaming Join and Map Side Join

Before we try doing the joins, lets prepare some data:

1. Login to Cockpit and ssh to root@edge.cluster and switch to user admin

```
ssh root@edge.cluster
su - admin
```

2. Download sample dataset and upload them to HDFS

```
wget http://repo.kagesenshi.org/hdptraining/address.jsonl
wget http://repo.kagesenshi.org/hdptraining/bank_account.jsonl
wget http://repo.kagesenshi.org/hdptraining/employees.jsonl
wget http://repo.kagesenshi.org/hdptraining/sales.jsonl
hdfs dfs -mkdir /tmp/address/ /tmp/bank_account/ /tmp/employees/ /tmp/sales
hdfs dfs -put address.jsonl /tmp/address
hdfs dfs -put bank_account.jsonl /tmp/bank_account
hdfs dfs -put employees.jsonl /tmp/employees
hdfs dfs -put sales.jsonl /tmp/sales
```

3. Launch beeline

```
beeline -u jdbc:hive2://master1.cluster:10500/default -n admin
```

4. Create table and load data

```
-- address table

CREATE TEMPORARY EXTERNAL TABLE address_tmp (
    employee_id int,
    street string,
    postcode string,
    state string
)
ROW FORMAT SERDE   'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION '/tmp/address';
```

```
CREATE TABLE address (
    employee_id int,
    street string,
    postcode string,
    state string
)
CLUSTERED BY (employee_id) SORTED BY (employee_id) INTO 3 BUCKETS
STORED AS ORC;

INSERT OVERWRITE TABLE address SELECT * FROM address_tmp;

-- bank account

CREATE TEMPORARY EXTERNAL TABLE bank_account_tmp (
    employee_id int,
    iban string,
    bank_country string
)
ROW FORMAT SERDE   'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION '/tmp/bank_account';

CREATE TABLE bank_account (
    employee_id int,
    iban string,
    bank_country string
)
CLUSTERED BY (employee_id) SORTED BY (employee_id) INTO 3 BUCKETS
STORED AS ORC;

INSERT OVERWRITE TABLE bank_account SELECT * FROM bank_account_tmp;

-- employees

CREATE TEMPORARY EXTERNAL TABLE employees_tmp (
    id int,
    lastname string,
    firstname string,
    branch string,
    age int
)
ROW FORMAT SERDE   'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION '/tmp/employees';
```

```
    CREATE TABLE employees (
        id int,
        lastname string,
        firstname string,
        branch string,
        age int
    )
    CLUSTERED BY (id) SORTED BY (id) INTO 3 BUCKETS
    STORED AS ORC;

    INSERT OVERWRITE TABLE employees SELECT * FROM employees_tmp;
```

```
-- sales

CREATE TEMPORARY EXTERNAL TABLE sales_tmp (
    sales_date date,
    employee_id int,
    product_group string,
    product string,
    sales_price float
)
ROW FORMAT SERDE   'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION '/tmp/sales';

CREATE TABLE sales (
    sales_date date,
    employee_id int,
    product_group string,
    product string,
    sales_price float
)
CLUSTERED BY (employee_id) SORTED BY (employee_id) INTO 3 BUCKETS
STORED AS ORC;

INSERT OVERWRITE TABLE sales SELECT * FROM sales_tmp;
```

5. Lets enable Hive View & Hive View 2.0 to use LLAP. Login to Ambari as admin and navigate to Manage Ambari > Views
6. Select Hive , and click Hive View
7. Click Edit link on the Settings box
8. Set Use Interactive Mode to 'true' and save
9. Repeat steps above for Hive View 2.0

Lab: Hive Management

11.1. Streaming Join

1. Login to Ambari as admin, and navigate to Hive View 2.0
2. Enter following query into the SQL editor

```
SELECT sales.*, employees.*, address.* FROM sales
       JOIN employees
       JOIN address
    ON sales.employee_id=employees.id
    AND address.employee_id=employees.id;
```

3. Click Visual Explain
4. Enter following query into the SQL editor

```
SELECT /*+STREAMTABLE(sales) */ sales.*, employees.*, address.* FROM sales
       JOIN employees
       JOIN address
    ON sales.employee_id=employees.id
    AND address.employee_id=employees.id;
```

5. Click Visual Explain
6. Compare the differences in execution graph of both queries

11.2. Map Side Join

1. Login to Ambari as admin, and navigate to Hive View 2.0
2. Enter following query into the SQL editor

```
SELECT sales.*, employees.*, address.* FROM sales
       JOIN employees
       JOIN address
    ON sales.employee_id=employees.id
    AND address.employee_id=employees.id;
```

3. Click Visual Explain
4. Enter following query into the SQL editor

```
SELECT /*+MAPJOIN(employees,address) */ sales.*, employees.*, address.* FROM sales
       JOIN employees
       JOIN address
    ON sales.employee_id=employees.id
    AND address.employee_id=employees.id;
```

5. Click Visual Explain
6. Compare the differences in execution graph of both queries

11.3. Bucketized Map Side Join

1. Login to Ambari as admin, and navigate to Hive View 2.0
2. Enter following query into the SQL editor

```
SELECT sales.*, employees.*, address.* FROM sales
       JOIN employees
       JOIN address
    ON sales.employee_id=employees.id
    AND address.employee_id=employees.id;
```

3. Click Visual Explain
4. Enter following query into the SQL editor

```
set hive.auto.convert.sortmerge.join=true;
set hive.optimize.bucketmapjoin = true;
set hive.optimize.bucketmapjoin.sortedmerge = true;
set hive.convert.join.bucket.mapjoin.tez=true;

SELECT sales.*, employees.*, address.* FROM sales
       JOIN employees
       JOIN address
    ON sales.employee_id=employees.id
    AND address.employee_id=employees.id;
```

5. Click Visual Explain
6. Compare the differences in execution graph of both queries

12. Frequency Statistics On Hive

13. Window Based Analysis Using Hive

