Hive and Imapala Implementation

Business Question:

DataCo is a retail store, and our mission is to help the organization get better insight by asking bigger questions.

To analyze the transaction data in the new platform, we need to ingest it into the Hadoop Distributed File System (HDFS). We need to find a tool that easily transfers structured data from a RDBMS to HDFS, while preserving structure. That enables us to query the data, but not interfere with or break any regular workload on it.

Apache Sqoop, which is part of CDH, is that tool. The nice thing about Sqoop is that we can automatically load our relational data from MySQL into HDFS, while preserving the structure.

You should first open a terminal, which you can do by clicking the black "Terminal" icon at the top of your screen. Once it is open, you can launch the Sqoop job:

Sqoop command:

```
sqoop import-all-tables \
   -m 1 \
   --connect jdbc:mysql://quickstart:3306/retail_db \
   --username=retail_dba \
   --password=cloudera \
   --compression-codec=snappy \
   --as-avrodatafile \
   --warehouse-dir=/user/hive/warehouse
```

It is launching MapReduce jobs to export the data from our MySQL database, and put those export files in Avro format in HDFS

Sqoop should also have created schema files for this data in your home directory.

command to check the avro files `ls -l *.avsc`

let's copy them into HDFS in to location `/user/examples`

We're going to use Hue's Impala app to create the metadata for our tables in Hue, and then query them. Hue provides a web-based interface for many of the tools in CDH

Loading HUE Interface

Hue url : http://quickstart.cloudera:8888/about/

username:cloudera
password:cloudera

Once you are inside of Hue, click on Query Editors, and open the Impala Query Editor.

run below commands in impala shell.

Creating tables in Impala using AVRO schemas

```
CREATE EXTERNAL TABLE categories STORED AS AVRO
LOCATION 'hdfs:///user/hive/warehouse/categories'
TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart/user/examples/sqoop_import_categories.avsc');

CREATE EXTERNAL TABLE customers STORED AS AVRO
LOCATION 'hdfs:///user/hive/warehouse/customers'
TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart/user/examples/sqoop_import_customers.avsc');

CREATE EXTERNAL TABLE departments STORED AS AVRO
LOCATION 'hdfs:///user/hive/warehouse/departments'
TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart/user/examples/sqoop_import_departments.avsc');

CREATE EXTERNAL TABLE orders STORED AS AVRO
LOCATION 'hdfs:///user/hive/warehouse/orders'
TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart/user/examples/sqoop_import_orders.avsc');

CREATE EXTERNAL TABLE order_items STORED AS AVRO
LOCATION 'hdfs:///user/hive/warehouse/order_items'
TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart/user/examples/sqoop_import_order_items.avsc');

CREATE EXTERNAL TABLE products STORED AS AVRO
LOCATION 'hdfs:///user/hive/warehouse/products'
TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart/user/examples/sqoop_import_products.avsc');
```

Now, we have created tables using the avro schemas which we have copied into user/examples in earlier step.

check whether tables created or not by clicking on the "Refresh Table List" icon on the left to see your new tables.
or run show tables; command

Now that your transaction data is readily available for structured queries in CDH, it's time to address DataCo's business question.

Query performace

```
-- Most popular product categories
select c.category_name, count(order_item_quantity) as count
from order_items oi
inner join products p on oi.order_item_product_id = p.product_id
inner join categories c on c.category_id = p.product_category_id
group by c.category_name
order by count desc
limit 10;
```

```
-- top 10 revenue generating products
select p.product_id, p.product_name, r.revenue
from products p inner join
(select oi.order_item_product_id, sum(cast(oi.order_item_subtotal as float)) as revenue
from order_items oi inner join orders o
on oi.order_item_order_id = o.order_id
where o.order_status <> 'CANCELED'
and o.order_status <> 'SUSPECTED_FRAUD'
group by order_item_product_id) r
on p.product_id = r.order_item_product_id
order by r.revenue desc
limit 10;
```

run these scripts in both impala editor and hive editor from query editors to check the performance of hive and impala.

Now, we have learned how to create and query tables using Impala and running the scripts using Impala and Hive editors.

Note: We will provide you .csv files and schema files for all the tables. you can directly load the .csv files from local system to HDFS using either copyFromLocal or put commands and can create tables using Avro schemas.

AVRO Demo Implementation

JSON data

We will use below sample data (StudentActivity.json):

{"id":"A91D021BA58444B29D4D42CA5E39F7BF","student_id":100,"university_id":908,"course_details":{"course_id":100,"enroll_date":"2012-02-13 00:00:00.000000000","verb":"completed","result_score":0.9}}

{"id":"502A77CC99B241CB94CA356F5218F1A9","student_id":101,"university_id":112,"course_details":{"course_id":233,"enroll_date":"2011-06-08 00:00:00.000000000","verb":"started","result_score":0.65}}

{"id":"5D04CD5ABF014D6EBA237766F9B470DE","student_id":102,"university_id":340,"course_details":{"course_id":339,"enroll_date":"2012-03-06 00:00:00.000000000","verb":"started","result_score":0.57}}

Note that the JSON records are nested ones. Now,copy the above data and save it as StudentActivity.json

Defining a schema

Avro schemas are defined using JSON. The avro schema for our sample data is defined as below (StudentActivity.avsc):

```
{

   "namespace": "com.rishav.avro",

   "type": "record",

   "name": "StudentActivity",

   "fields": [

      {

         "name": "id",

         "type": "string"

      },
```

```
{
    "name": "student_id",
    "type": "int"
},
{
    "name": "university_id",
    "type": "int"
},
{
    "name": "course_details",
    "type": {
        "name": "Activity",
        "type": "record",
        "fields": [
            {
                "name": "course_id",
                "type": "int"
            },
            {
                "name": "enroll_date",
                "type": "string"
```

```
            },
            {
               "name": "verb",
               "type": "string"
            },
            {
               "name": "result_score",
               "type": "double"
            }
         ]
      }
   }
  ]
}
```

Serialization/Deserialization using Avro command line tools

Avro provides a jar file by name avro-tools-<version>.jar which provides many command line tools as listed below:

```
$ java -jar avro-tools-1.7.5.jar
Version 1.7.5 of Apache Avro
```

For converting json sample data to Avro binary format use "fromjson" option and for getting json data back from Avro files use "tojson" option.

Command for serializing json

Without any compression

```
java -jar avro-tools-1.7.5.jar fromjson --schema-file StudentActivity.avsc StudentActivity.json > StudentActivity.avro
```

With snappy compression

```
java -jar avro-tools-1.7.5.jar fromjson --schema-file StudentActivity.avsc StudentActivity.json > StudentActivity.snappy.avro
```

Command for deserializing json

The same command is used for deserializing both compressed and uncompressed data

```
java -jar avro-tools-1.7.5.jar tojson StudentActivity.avro
java -jar avro-tools-1.7.5.jar tojson StudentActivity.snappy.avro
```

As Avro data file contains the schema also, we can retrieve it using this commmand:

```
java -jar avro-tools-1.7.5.jar getschema StudentActivity.avro
java -jar avro-tools-1.7.5.jar getschema StudentActivity.snappy.avro
```