Problem Statement 1:

1. Using sqoop, import orders table into hdfs. File should be loaded as Avro File and use snappy compression
2. Using sqoop, import order\_items  table into hdfs. Files should be loaded as avro file and use snappy compression
3. Using Spark Scala load data at above 2 locations as *dataframes*.
4. Please find total orders and total amount per status per day. The result should be sorted by order date in descending, order status in ascending and total amount in descending and total orders in ascending.
5. Store the result as parquet file into hdfs using gzip compression under a folder
6. Store the result as parquet file into hdfs using snappy compression under a folder
7. Store the result as CSV file into hdfs without compression under a folder
8. create a mysql table named result and load data from from the baove output location to mysql table named result

Problem Statement 2:

1. Using sqoop copy data available in mysql products table to a folder on hdfs as text file. Columns should be delimited by pipe '|'
2. Move all the files from the above location to a different location
3. Change permissions of all the files under the above such that owner has read,write and execute permissions, group has read and write permissions whereas others have just read and execute permissions
4. Read data in the above location and do the following operations. Sort the resultant dataset by category id
   * filter such that your RDD\DF has products whose price is lesser than 100 USD
   * on the filtered data set find out the higest value in the product\_price column under each category
   * on the filtered data set also find out total products under each category
   * on the filtered data set also find out the average price of the product under each category
   * on the filtered data set also find out the minimum price of the product under each category
5. Store the result in avro file format using snappy compression under hdfs folder

Problem Statement 3:

1. Import all tables from mysql database into hdfs as avro data files. Use compression and the compression codec should be snappy. The output data warehouse directory should be retail\_stage.db
2. Create a metastore table that should point to the orders data imported by sqoop job above. Name the table orders\_sqoop.
3. Write query in hive that shows all orders belonging to a certain day. This day is when the most orders were placed. Select data from orders\_sqoop.
4. Now create a table named retail.orders\_avro in hive stored as avro, the table should have same table definition as order\_sqoop. Additionally, this new table should be partitioned by the order month i.e -> year-order\_month.(example: 2014-01)
5. Load data into orders\_avro table from orders\_sqoop table.
6. Evolve the avro schema related to orders\_sqoop table by adding more fields named (order\_style String, order\_zone Integer)
7. Insert two more records into orders\_sqoop table.
8. Write query in hive that shows all orders belonging to a certain day. This day is when the most orders were placed. select data from orders\_sqoop

Problem Statement 4:

1. Import orders table from mysql as text file to a destination folder in hdfs. Fields should be terminated by a tab character ("\t") character and lines should be terminated by new line character ("\n").
2. Import orders table from mysql  into hdfs to a destination folder in hdfs. File should be stored as avro file.
3. Import orders table from mysql  into hdfs  to a destination folder in hdfs. File should be stored as parquet file.
4. Transform/Convert data-files at part 2 output location and store the converted file at the following locations and file formats
   * save the data to hdfs using snappy compression as parquet file at /user/cloudera/problem/parquet-snappy-compress
   * save the data to hdfs using gzip compression as text file at /user/cloudera/problem/text-gzip-compress
   * save the data to hdfs using no compression as sequence file at /user/cloudera/problem/sequence
   * save the data to hdfs using snappy compression as text file at /user/cloudera/problem/text-snappy-compress
5. Transform/Convert data-files at /user/cloudera/problem/parquet-snappy-compress and store the converted file at the following locations and file formats
   * save the data to hdfs using no compression as parquet file at /user/cloudera/problem/parquet-no-compress
   * save the data to hdfs using snappy compression as avro file at /user/cloudera/problem/avro-snappy
6. Transform/Convert data-files at /user/cloudera/problem/avro-snappy and store the converted file at the following locations and file formats
   * save the data to hdfs using no compression as json file at /user/cloudera/problem/json-no-compress
   * save the data to hdfs using gzip compression as json file at /user/cloudera/problem/json-gzip
7. Transform/Convert data-files at  /user/cloudera/problem/json-gzip and store the converted file at the following locations and file formats
   * save the data to as comma separated text using gzip compression at   /user/cloudera/problem/csv-gzip
8. Using spark access data at /user/cloudera/problem/sequence and stored it back to hdfs using no compression as ORC file to HDFS to destination /user/cloudera/problem/orc

Problem Statement 5:

**Pre-Work:**Please perform these steps before solving the problem  
1. Login to MySQL using below commands on a fresh terminal window  
    mysql -u retail\_dba -p  
    Password = cloudera  
2. Create a replica product table and name it products\_replica  
    create table products\_replica as select \* from products  
3. Add primary key to the newly created table  
    alter table products\_replica add primary key (product\_id);  
4. Add two more columns  
    alter table products\_replica add column (product\_grade int, product\_sentiment varchar(100))  
5. Run below two update statements to modify the data  
    update products\_replica set product\_grade = 1  where product\_price > 500;  
    update products\_replica set product\_sentiment  = 'WEAK'  where product\_price between 300 and  500;

1. Using sqoop, import products\_replica table from MYSQL into hdfs such that fields are separated by a '|' and lines are separated by '\n'. Null values are represented as -1 for numbers and "NOT-AVAILABLE" for strings. Only records with product id greater than or equal to 1 and less than or equal to 1000 should be imported and use 3 mappers for importing. The destination file should be stored as a text file to directory  **/user/cloudera/problem/products-text**.
2. Using sqoop, import products\_replica table from MYSQL into hdfs such that fields are separated by a '\*' and lines are separated by '\n'. Null values are represented as -1000 for numbers and "NA" for strings. Only records with product id less than or equal to 1111 should be imported and use 2 mappers for importing. The destination file should be stored as a text file to directory  **/user/cloudera/problem/products-text-part1**.
3. Using sqoop, import products\_replica table from MYSQL into hdfs such that fields are separated by a '\*' and lines are separated by '\n'. Null values are represented as -1000 for numbers and "NA" for strings. Only records with product id greater than 1111 should be imported and use 5 mappers for importing. The destination file should be stored as a text file to directory  **/user/cloudera/problem/products-text-part2.**
4. Using sqoop merge data available in **/user/cloudera/problem/products-text-part1**and **/user/cloudera/problem/products-text-part2**to produce a new set of files in **/user/cloudera/problem/products-text-both-parts**
5. Using sqoop do the following. Read the entire steps before you create the sqoop job.
   * create a sqoop job Import Products\_replica table as text file to directory **/user/cloudera/problem/products-incremental**. Import all the records.
   * insert three more records to Products\_replica from mysql
   * run the sqoop job again so that only newly added records can be pulled from mysql
   * insert 2 more records to Products\_replica from mysql
   * run the sqoop job again so that only newly added records can be pulled from mysql
   * Validate to make sure the records have not be duplicated in **HDFS**
6. Using sqoop do the following. Read the entire steps before you create the sqoop job.
   * create a hive table in database named **problem**using below command
   * create table **products\_hive**(product\_id int, product\_category\_id int, product\_name string, product\_description string, product\_price float, product\_imaage string,product\_grade int,  product\_sentiment string);
   * create a sqoop job Import Products\_replica table as hive table to database named **problem**. name the table as **products\_hive**.
   * insert three more records to Products\_replica from mysql
   * run the sqoop job again so that only newly added records can be pulled from mysql
   * insert 2 more records to Products\_replica from mysql
   * run the sqoop job again so that only newly added records can be pulled from mysql
   * Validate to make sure the records have not been duplicated in **Hive**table
7. Using sqoop do the following. .
   * insert 2 more records into **products\_hive**table using hive.
   * create table in mysql using below command
   * create table products\_external  (product\_id int(11) primary Key, product\_grade int(11), product\_category\_id int(11), product\_name varchar(100), product\_description varchar(100), product\_price float, product\_impage varchar(500), product\_sentiment varchar(100));
   * export data from products\_hive (hive) table to (mysql) products\_external table.
   * insert 2 more records to Products\_hive table from hive
   * export data from products\_hive table to products\_external table.
   * Validate to make sure the records have not be duplicated in **mysql**table

Problem Statement 6:

1. Create a hive meta store database named **problem6**and import all tables from mysql retail\_db database into hive meta store.
2. On spark shell use data available on meta store as source and perform step 3,4,5 and 6.
3. Rank products within department by price and order by department ascending and rank descending
4. Find top 10 customers with most unique product purchases. if more than one customer has the same number of product purchases then the customer with the lowest customer\_id will take precedence
5. On dataset from step 3, apply filter such that only products less than 100 are extractedand on dataset from step 4, extract details of products purchased by top 10 customers which are priced at less than 100 USD per unit
6. Store the result of 5 and 6 in new meta store tables within hive