Hive Assignment 1

Car Insurance Cold Calls Data Analysis

Note: Replace all the path, directory names, table and column names the way you have created it in your system

Problem 1: Data Loading

1. Create an external table with the given schema and the table should store data as text file from HDFS path.

```
CREATE EXTERNAL TABLE car_insurance_data (
Id INT,
Age INT,
Job STRING,
Marital STRING,
Education STRING,
Default INT,
Balance INT,
HHInsurance INT,
CarLoan INT,
Communication STRING,
LastContactDay INT,
LastContactMonth INT,
NoOfContacts INT,
DaysPassed INT,
PrevAttempts INT,
Outcome STRING,
CallStart STRING,
CallEnd STRING,
CarInsurance INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/hive/data/car_insurance_data/';
```

Problem 2: Data Exploration

1. How many records are there in the dataset?

```
SELECT COUNT(*) FROM car_insurance_data;
```

2. How many unique job categories are there?

```
SELECT COUNT(DISTINCT Job) FROM
car_insurance_data;
```

3. What is the age distribution of customers in the dataset? Provide a breakdown by age group: 18-30, 31-45, 46-60, 61+.

```
SELECT CASE
  WHEN Age BETWEEN 18 AND 30 THEN '18-30'
  WHEN Age BETWEEN 31 AND 45 THEN '31-45'
  WHEN Age BETWEEN 46 AND 60 THEN '46-60'
  ELSE '61+'
END AS age_group,
COUNT(*) AS count
FROM car_insurance_data
GROUP BY CASE
  WHEN Age BETWEEN 18 AND 30 THEN '18-30'
  WHEN Age BETWEEN 31 AND 45 THEN '31-45'
  WHEN Age BETWEEN 46 AND 60 THEN '46-60'
  ELSE '61+'
END;
```

4. Count the number of records that have missing values in any field.

```
SELECT COUNT(*)
FROM car_insurance_data
WHERE Id IS NULL
OR Age IS NULL
OR Job IS NULL
OR Marital IS NULL
OR Education IS NULL
OR Default IS NULL
OR Balance IS NULL
OR HHInsurance IS NULL
OR CarLoan IS NULL
OR Communication IS NULL
OR LastContactDay IS NULL
OR LastContactMonth IS NULL
OR NoOfContacts IS NULL
OR DaysPassed IS NULL
```

```
OR PrevAttempts IS NULL
OR Outcome IS NULL
OR CallStart IS NULL
OR CallEnd IS NULL
OR CarInsurance IS NULL;
```

5. Determine the number of unique 'Outcome' values and their respective counts.

```
SELECT Outcome, COUNT(*)
FROM car_insurance_data
GROUP BY Outcome;
```

6. Find the number of customers who have both a car loan and home insurance.

```
SELECT COUNT(*)
FROM car_insurance_data
WHERE CarLoan = 1 AND HHInsurance = 1;
```

Problem 3: Aggregations

1. What is the average, minimum, and maximum balance for each job category?

```
SELECT Job, AVG(Balance) as average_balance,
MIN(Balance) as min_balance, MAX(Balance) as
max_balance
FROM car_insurance_data
GROUP BY Job;
```

2. Find the total number of customers with and without car insurance.

```
SELECT CarInsurance, COUNT(*)
FROM car_insurance_data
```

3. Count the number of customers for each communication type.

```
SELECT Communication, COUNT(*)
FROM car_insurance_data
GROUP BY Communication;
```

4. Calculate the sum of 'Balance' for each 'Communication' type.

```
SELECT Communication, SUM(Balance)
FROM car_insurance_data
```

```
GROUP BY Communication;
```

5. Count the number of 'PrevAttempts' for each 'Outcome' type.

```
SELECT Outcome, SUM(PrevAttempts)
FROM car_insurance_data
GROUP BY Outcome;
```

6. Calculate the average 'NoOfContacts' for people with and without 'CarInsurance'.

```
SELECT CarInsurance, AVG(NoOfContacts)
FROM car_insurance_data
GROUP BY CarInsurance;
```

Problem 4: Partitioning and Bucketing

1. Create a partitioned table on 'Education' and 'Marital' status. Load data from the original table to this new partitioned table.

```
CREATE TABLE car_insurance_data_partitioned (
  Id INT,
  Age INT,
  Job STRING,
  Default INT,
  Balance INT,
  HHInsurance INT,
  CarLoan INT,
  Communication STRING,
  LastContactDay INT,
  LastContactMonth INT,
  NoOfContacts INT,
  DaysPassed INT,
  PrevAttempts INT,
  Outcome STRING,
  CallStart STRING,
  CallEnd STRING,
  CarInsurance INT)
PARTITIONED BY (Education STRING, Marital STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

```
INSERT OVERWRITE TABLE
car_insurance_data_partitioned PARTITION(Education,
Marital)
SELECT Id, Age, Job, Default, Balance, HHInsurance,
CarLoan, Communication, LastContactDay,
LastContactMonth, NoOfContacts, DaysPassed,
PrevAttempts, Outcome, CallStart, CallEnd,
CarInsurance, Education, Marital
FROM car_insurance_data;
```

2. Create a bucketed table on 'Age', bucketed into 4 groups (as per the age groups mentioned above). Load data from the original table into this bucketed table.

```
CREATE TABLE car_insurance_data_bucketed (
  Id INT,
  Age INT,
  Job STRING,
  Marital STRING,
  Education STRING,
  Default INT,
  Balance INT,
  HHInsurance INT,
  CarLoan INT,
  Communication STRING,
  LastContactDay INT,
  LastContactMonth INT,
  NoOfContacts INT,
  DaysPassed INT,
  PrevAttempts INT,
  Outcome STRING,
  CallStart STRING,
  CallEnd STRING,
  CarInsurance INT)
CLUSTERED BY (Age) INTO 4 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

```
INSERT OVERWRITE TABLE
car_insurance_data_bucketed
SELECT * FROM car_insurance_data;
```

3. Add an additional partition on 'Job' to the partitioned table created earlier and move the data accordingly.

   If you want to add an additional partition on 'Job' to the previously created partitioned table, you actually have to create a new table as Hive does not allow altering the partitioning of existing tables. However, it's a straightforward task to create a new partitioned table and move the data accordingly.

```
CREATE TABLE car_insurance_data_partitioned_new (
  Id INT,
  Age INT,
  Default INT,
  Balance INT,
  HHInsurance INT,
  CarLoan INT,
  Communication STRING,
  LastContactDay INT,
  LastContactMonth INT,
  NoOfContacts INT,
  DaysPassed INT,
  PrevAttempts INT,
  Outcome STRING,
  CallStart STRING,
  CallEnd STRING,
  CarInsurance INT)
PARTITIONED BY (Education STRING, Marital STRING,
Job STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

```
INSERT OVERWRITE TABLE
car_insurance_data_partitioned_new
PARTITION(Education, Marital, Job)
SELECT Id, Age, Default, Balance, HHInsurance,
CarLoan, Communication, LastContactDay,
LastContactMonth, NoOfContacts, DaysPassed,
PrevAttempts, Outcome, CallStart, CallEnd,
CarInsurance, Education, Marital, Job
FROM car_insurance_data_partitioned;
```

4. Increase the number of buckets in the bucketed table to 10 and redistribute the data.

In Hive, once a table is bucketed, the number of buckets cannot be changed. The process of bucketing happens at the time of table creation and is immutable. Therefore, in order to increase the number of buckets, you will need to create a new table with the desired number of buckets and then insert data into the new table from the existing one.

```
CREATE TABLE car_insurance_data_bucketed_new (
  Id INT,
  Age INT,
  Job STRING,
  Marital STRING,
  Education STRING,
  Default INT,
  Balance INT,
  HHInsurance INT,
  CarLoan INT,
  Communication STRING,
  LastContactDay INT,
  LastContactMonth INT,
  NoOfContacts INT,
  DaysPassed INT,
  PrevAttempts INT,
  Outcome STRING,
  CallStart STRING,
```

```
  CallEnd STRING,
  CarInsurance INT)
CLUSTERED BY (Age) INTO 10 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

INSERT OVERWRITE TABLE
car_insurance_data_bucketed_new
SELECT * FROM car_insurance_data_bucketed;
```

Problem 5: Optimized Joins

1. Join the original table with the partitioned table and find out the average 'Balance' for each 'Job' and 'Education' level.

```
SELECT o.Job, p.Education, AVG(o.Balance) as
average_balance
FROM car_insurance_data o
JOIN car_insurance_data_partitioned_new p ON
o.Id = p.Id
GROUP BY o.Job, p.Education;
```

2. Join the original table with the bucketed table and calculate the total 'NoOfContacts' for each 'Age' group.

```
SELECT o.Age, SUM(o.NoOfContacts) as
total_contacts
FROM car_insurance_data o
JOIN car_insurance_data_bucketed_new b ON
o.Id = b.Id
GROUP BY o.Age;
```

3. Join the partitioned table and the bucketed table based on the 'Id' field and find the total balance for each education level and marital status for each age group.

```
SELECT p.Age, p.Education, p.Marital,
SUM(b.Balance) AS total_balance
FROM car_insurance_data_partitioned_new p
```

```
JOIN car_insurance_data_bucketed_new b ON
p.Id = b.Id
GROUP BY p.Age, p.Education, p.Marital;
```

Problem 6: Window Function

1. Calculate the cumulative sum of 'NoOfContacts' for each 'Job' category, ordered by 'Age'.

```
SELECT Age, Job, NoOfContacts,
SUM(NoOfContacts) OVER (PARTITION BY
Job ORDER BY Age) AS cumulative_sum
FROM car_insurance_data
ORDER BY Age, Job;
```

2. Calculate the running average of 'Balance' for each 'Job' category, ordered by 'Age'.

```
SELECT Age, Job, Balance, AVG(Balance)
OVER (PARTITION BY Job ORDER BY Age)
AS running_average
FROM car_insurance_data
ORDER BY Age, Job;
```

3. For each 'Job' category, find the maximum 'Balance' for each 'Age' group using window functions.

```
SELECT Age, Job, Balance
FROM (
  SELECT Age, Job, Balance,
ROW_NUMBER() OVER (PARTITION BY
Job, Age ORDER BY Balance DESC) AS rn
  FROM car_insurance_data
) t
WHERE rn = 1
ORDER BY Job, Age;
```

4. Calculate the rank of 'Balance' within each 'Job' category, ordered by 'Balance' descending.

```
SELECT Age, Job, Balance, RANK() OVER
(PARTITION BY Job ORDER BY Balance
DESC) AS balance_rank
FROM car_insurance_data
ORDER BY Job, Balance DESC;
```

Problem 7: Advanced Aggregations

1. Find the job category with the highest number of car insurances.

```
SELECT Job
FROM (
   SELECT Job, COUNT(*) AS
car_insurance_count
   FROM car_insurance_data
   WHERE CarInsurance = 1
   GROUP BY Job
) t
ORDER BY car_insurance_count DESC
LIMIT 1;
```

2. Which month has seen the highest number of last contacts?

```
SELECT LastContactMonth, COUNT(*) AS
contact_count
FROM car_insurance_data
GROUP BY LastContactMonth
ORDER BY contact_count DESC
LIMIT 1;
```

3. Calculate the ratio of the number of customers with car insurance to the number of customers without car insurance for each job category.

```
SELECT t1.Job, t1.car_insurance_count /
t2.no_car_insurance_count AS
car_insurance_ratio
FROM (
   SELECT Job, COUNT(*) AS
car_insurance_count
   FROM car_insurance_data
```

```
    WHERE CarInsurance = 1
    GROUP BY Job
) t1
JOIN (
    SELECT Job, COUNT(*) AS
no_car_insurance_count
    FROM car_insurance_data
    WHERE CarInsurance = 0
    GROUP BY Job
) t2
ON t1.Job = t2.Job;
```

4. Find out the 'Job' and 'Education' level combination which has the highest number of car insurances.

```
SELECT Job, Education
FROM (
    SELECT Job, Education, COUNT(*)
AS car_insurance_count
    FROM car_insurance_data
    WHERE CarInsurance = 1
    GROUP BY Job, Education
) t
ORDER BY car_insurance_count
DESC
LIMIT 1;
```

5. Calculate the average 'NoOfContacts' for each 'Outcome' and 'Job' combination.

```
SELECT Outcome, Job,
AVG(NoOfContacts) AS
average_contacts
FROM car_insurance_data
GROUP BY Outcome, Job;
```

6. Determine the month with the highest total 'Balance' of customers.

```
SELECT LastContactMonth,
SUM(Balance) AS total_balance
FROM car_insurance_data
GROUP BY LastContactMonth
```

```
ORDER BY total_balance DESC
LIMIT 1;
```

Problem 8: Complex joins and aggregations

1. For customers who have both a car loan and home insurance, find out the average 'Balance' for each 'Education' level.

```
SELECT Education, AVG(Balance) AS
average_balance
FROM (
   SELECT Education, Balance
   FROM car_insurance_data
   WHERE CarLoan = 1 AND
HHInsurance = 1
) t
GROUP BY Education;
```

2. Identify the top 3 'Communication' types for customers with 'CarInsurance', and display their average 'NoOfContacts'.

```
SELECT Communication,
AVG(NoOfContacts) AS
average_contacts
FROM (
   SELECT Communication,
NoOfContacts
   FROM car_insurance_data
   WHERE CarInsurance = 1
) t
GROUP BY Communication
ORDER BY average_contacts DESC
LIMIT 3;
```

3. For customers who have a car loan, calculate the average balance for each job category.

```
SELECT Job, AVG(Balance) AS
average_balance
FROM car_insurance_data
WHERE CarLoan = 1
```

```
GROUP BY Job;
```

4. Identify the top 5 job categories that have the most customers with a 'default', and show their average 'balance'.

```
SELECT Job, AVG(Balance) AS
average_balance
FROM (
   SELECT Job, Balance
   FROM car_insurance_data
   WHERE Default = 1
) t
GROUP BY Job
ORDER BY COUNT(*) DESC
LIMIT 5;
```

Problem 9: Advanced Window Functions

1. Calculate the difference in 'NoOfContacts' between each customer and the customer with the next highest number of contacts in the same 'Job' category.

```
SELECT c1.Id, c1.Job, c1.NoOfContacts,
c1.NoOfContacts - c2.NextHighestContacts
AS ContactDifference
FROM car_insurance_data c1
JOIN (
   SELECT c1.Job, c1.NoOfContacts,
MIN(c2.NoOfContacts) AS
NextHighestContacts
   FROM car_insurance_data c1
   LEFT JOIN car_insurance_data c2 ON
c1.Job = c2.Job AND c1.NoOfContacts <
c2.NoOfContacts
   GROUP BY c1.Job, c1.NoOfContacts
) c2 ON c1.Job = c2.Job AND
c1.NoOfContacts = c2.NoOfContacts;
```

2. For each customer, calculate the difference between their 'balance' and the average 'balance' of their 'job' category.

```
SELECT c.Id, c.Job, c.Balance,
c.Balance - j.AvgBalance AS
BalanceDifference
FROM car_insurance_data c
JOIN (
   SELECT Job, AVG(Balance) AS
AvgBalance
   FROM car_insurance_data
   GROUP BY Job
) j ON c.Job = j.Job;
```

3. For each 'Job' category, find the customer who had the longest call duration.

```
SELECT Job, Id, CallDuration
FROM (
   SELECT Job, Id, CallDuration,
ROW_NUMBER() OVER (PARTITION BY Job
ORDER BY CallDuration DESC) AS rn
   FROM car_insurance_data
) t
WHERE rn = 1;
```

4. Calculate the moving average of 'NoOfContacts' within each 'Job' category, using a window frame of the current row and the two preceding rows.

```
SELECT Id, Job, NoOfContacts,
 AVG(NoOfContacts) OVER (PARTITION BY
Job ORDER BY Id ROWS BETWEEN 2
PRECEDING AND CURRENT ROW) AS
moving_average
FROM car_insurance_data;
```

Problem 10: Performance Tuning

1. Experiment with different file formats (like ORC, Parquet) and measure their impact on the performance of your Hive queries.
2. Use different levels of compression and observe their effects on storage and query performance.

3. Compare the execution time of join queries with and without bucketing.
4. Optimize your Hive queries using different Hive optimization techniques (for example, predicate pushdown, map-side joins, etc.). Discuss the difference in performance.

Please make sure to submit the HiveQL queries, along with their results and your observations. This assignment not only tests your understanding of Apache Hive but also requires you to derive meaningful insights from the data.