Snowflake SQL Query Guide

QUERY 1: Update Store Opening Dates

This query updates all stores to have opening dates after January 1, 2014, using random dates within a 10-year period:

```sql
UPDATE DimStoreData
SET StoreOpeningDate = DATEADD(DAY, UNIFORM(0, 3600, RANDOM()), DATE '2014-01-01');
```

Purpose: Ensures all store opening dates fall within the last 10 years, replacing any older dates with random dates between Jan 2014 and Jan 2024.

QUERY 2: Update Recent Store Openings

This query specifically updates stores with IDs between 91 and 100 to have opening dates within the last 12 months:

```sql
UPDATE DimStoreData
SET StoreOpeningDate = DATEADD(DAY, UNIFORM(0, 360, RANDOM()), DATE '2023-01-01')
WHERE STOREID between 91 and 100;
```

Purpose: Designates a specific set of stores as "newly opened" within the last year.

QUERY 3: Update Customer Age Compliance

This query ensures all customers are at least 12 years old by adjusting birth dates for any younger customers:

```sql
UPDATE DIMCUSTOMER
SET DOB = DATEADD(YEAR,-12,DOB)
WHERE DOB >= DATEADD(YEAR,-12,'2024-01-01');
```

Purpose: Maintains data integrity by ensuring no customers are under 12 years old.

QUERY 4: Fix Incorrect Order Dates

This query identifies and corrects order dates that occur before their associated store's opening date:

```sql
UPDATE FACTORDERS F
SET F.DATEID = R.DATEID FROM
(SELECT
    O.OrderID,
    D2.DATEID
FROM (
    SELECT
        F.OrderID,
        DATEADD(
            DAY,
            DATEDIFF(DAY, S.STOREOPENINGDATE, '2024-01-01') * UNIFORM(1, 10, RANDOM()) * 0.1,
            S.STOREOPENINGDATE
        ) AS newdate
    FROM FACTORDERS F
    JOIN DIMDATE D ON F.DATEID = D.DATEID
    JOIN DIMSTOREDATA S ON F.STOREID = S.STOREID
    WHERE D.DATE < S.STOREOPENINGDATE
) O
JOIN DIMDATE D2 ON O.newdate = D2.DATE) R
WHERE F.ORDERID = R.ORDERID;
```

Purpose: Fixes data inconsistencies by ensuring no orders exist with dates prior to their store's opening date. The new date is calculated as a random date between 10% to 90% of the time from store opening to present.

QUERY 5: Identify Inactive Customers

This query finds customers who haven't placed an order in the last 30 days:

```sql
SELECT * FROM DIMCUSTOMER C WHERE C.CUSTOMERID NOT IN
(SELECT DISTINCT C.CUSTOMERID FROM DIMCUSTOMER C
JOIN FACTORDERS F ON C.CUSTOMERID = F.CUSTOMERID
JOIN DIMDATE D ON F.DATEID = D.DATEID
WHERE D.DATE >= DATEADD(MONTH, -1, '2024-01-01'));
```

Purpose: Identifies potentially inactive customers for marketing campaigns or retention strategies.

QUERY 6: Most Recently Opened Store Performance

This query identifies the most recently opened store and calculates its total sales since opening:

```sql
WITH store_rank AS (
    SELECT
        STOREID,
        STOREOPENINGDATE,
        ROW_NUMBER() OVER (ORDER BY STOREOPENINGDATE DESC) AS FRANK
    FROM DIMSTOREDATA
),
recent AS (
    SELECT STOREID
    FROM store_rank
    WHERE FRANK = 1
),
STOREAMOUNT AS
(SELECT
    o.STOREID,
    SUM(o.TOTALAMOUNT) AS TOTAL_SALES
FROM FACTORDERS o
JOIN recent s ON o.STOREID = s.STOREID
GROUP BY o.STOREID
)
SELECT * FROM DIMSTOREDATA S
JOIN STOREAMOUNT A ON S.STOREID = A.STOREID;
```

Purpose: Provides insights into the performance of the newest store location.

QUERY 7: Multi-Category Customers

This query identifies customers who have ordered products from more than 3 different categories in the last 6 months:

```sql
WITH BD AS (
SELECT F.CUSTOMERID, P.CATEGORY FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMPRODUCTDATA P ON F.PRODUCTID = P.PRODUCTID
WHERE D.DATE >= DATEADD(MONTH, -6, '2024-01-01')
GROUP BY F.CUSTOMERID, P.CATEGORY
)

SELECT CUSTOMERID FROM BD
GROUP BY CUSTOMERID
HAVING COUNT(DISTINCT CATEGORY) > 3;
```

Purpose: Identifies customers with diverse purchasing habits, potentially for cross-selling or loyalty rewards.

QUERY 8: Monthly Sales for Current Year

This query calculates the total sales for each month in the current year (2024):

```sql
SELECT D.MONTH, SUM(F.TOTALAMOUNT) FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
WHERE D.YEAR = 2024
GROUP BY D.MONTH
ORDER BY D.MONTH;
```

Purpose: Provides monthly sales trends for the current year for financial analysis.

QUERY 9: Highest Discount in Last Year

This query finds the order with the highest discount amount in the past year:

```sql
WITH BS AS (
    SELECT
        F.*,
        ROW_NUMBER() OVER (ORDER BY F.DISCOUNTAMOUNT DESC) AS DRANK
    FROM FACTORDERS F
    JOIN DIMDATE D ON F.DATEID = D.DATEID
    WHERE D.DATE >= DATEADD(YEAR, -1, '2024-01-01')
)

SELECT *
FROM BS
WHERE DRANK = 1;
```

Purpose: Identifies the most heavily discounted order for discount policy analysis.

QUERY 10: Calculate Total Sales by Unit Price

This query calculates the total theoretical sales by multiplying unit price by quantity ordered:

```sql
SELECT SUM(QUANTITYORDERED * UNITPRICE) AS TOTALSALES FROM FACTORDERS F
JOIN DIMPRODUCTDATA P ON F.PRODUCTID = P.PRODUCTID;
```

Purpose: Calculates gross sales before any discounts or adjustments.

QUERY 11: Customer with Maximum Lifetime Discount

This query identifies the customer who has received the highest total discount amount across all their orders:

```sql
SELECT CUSTOMERID, SUM(DISCOUNTAMOUNT) AS DIA FROM FACTORDERS F
GROUP BY CUSTOMERID
ORDER BY DIA DESC LIMIT 1;
```

Purpose: Identifies customers who may be particularly discount-sensitive or have received exceptional promotional benefits.

QUERY 12: Customer with Maximum Order Count

This query finds the customer who has placed the highest number of orders:

```sql
WITH BASEDA AS (
    SELECT CUSTOMERID, COUNT(ORDERID) AS OC
    FROM FACTORDERS F
    GROUP BY CUSTOMERID
),
ORDATA AS (
    SELECT B.*, ROW_NUMBER() OVER (ORDER BY OC DESC) AS ORD
    FROM BASEDA B
)
SELECT *
FROM ORDATA
WHERE ORD = 1;
```

Purpose: Identifies the most frequent ordering customer, potentially for loyalty recognition.

QUERY 13: Top 3 Brands by Sales

This query identifies the top 3 performing brands based on sales in the last year:

```sql
WITH BRANDSALES AS (
    SELECT P.BRANDNAME, SUM(F.TOTALAMOUNT) AS TOTALSALES
    FROM FACTORDERS F
    JOIN DIMDATE D ON F.DATEID = D.DATEID
    JOIN DIMPRODUCTDATA P ON F.PRODUCTID = P.PRODUCTID
    WHERE D.DATE >= DATEADD(YEAR, -1, '2024-01-01')
    GROUP BY P.BRANDNAME
),
BRANDSALESRANK AS (
    SELECT S.*, ROW_NUMBER() OVER (ORDER BY TOTALSALES DESC) AS SR
    FROM BRANDSALES S
)
SELECT *
FROM BRANDSALESRANK
WHERE SR <= 3;
```

Purpose: Reveals the highest-performing brands for inventory management and partnership decisions.

QUERY 14: Discount and Shipping Cost Impact Analysis

This query checks whether a standardized discount (5%) and shipping cost (8%) would result in a higher or lower total amount compared to current pricing:

```sql
SELECT CASE WHEN SUM(ORDERAMOUNT - ORDERAMOUNT * 0.05 - ORDERAMOUNT * 0.08) > SUM(TOTALAMOUNT)
THEN 'YES' ELSE 'NO'
END FROM FACTORDERS F LIMIT 10;
```

Purpose: Evaluates potential impact of standardizing discount and shipping rates across all orders.

QUERY 15: Customer Distribution by Loyalty Tier

This query counts the number of customers in each loyalty program tier:

```sql
SELECT L.PROGRAMTIER, COUNT(CUSTOMERID) FROM DIMCUSTOMER D
JOIN DIMLOYALTYINFO L ON D.LOYALTYINFOID = L.LOYALTYINFOID
GROUP BY L.PROGRAMTIER;
```

Purpose: Provides a breakdown of customer distribution across loyalty program tiers.

QUERY 16: Regional Category Sales for Last 6 Months

This query calculates total sales by region and product category for the last 6 months:

```sql
SELECT REGION, CATEGORY, SUM(TOTALAMOUNT) AS TOTSALES FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMPRODUCTDATA P ON F.PRODUCTID = P.PRODUCTID
JOIN DIMSTOREDATA S ON F.STOREID = S.STOREID
WHERE D.DATE >= DATEADD(MONTH,-6,'2024-01-01')
GROUP BY REGION, CATEGORY;
```

Purpose: Provides regional sales performance by product category for targeted marketing strategies.

QUERY 17: Top 5 Products by Quantity Ordered

This query identifies the top 5 products based on quantity ordered in the last 3 years:

```sql
WITH QUANT AS (
    SELECT F.PRODUCTID, SUM(QUANTITYORDERED) AS TOT
    FROM FACTORDERS F
    JOIN DIMDATE D ON F.DATEID = D.DATEID
    WHERE D.DATE >= DATEADD(YEAR, -3, '2024-01-01')
    GROUP BY F.PRODUCTID
),

QUANTRANK AS (
    SELECT Q.PRODUCTID, Q.TOT,
           ROW_NUMBER() OVER (ORDER BY Q.TOT DESC) AS QUANTWISE
    FROM QUANT Q
)

SELECT PRODUCTID, TOT
FROM QUANTRANK
WHERE QUANTWISE <= 5;
```

Purpose: Identifies the most popular products by volume for inventory optimization.

QUERY 18: Total Sales by Loyalty Tier

This query calculates the total sales amount for each loyalty program tier since 2023:

```sql
SELECT L.PROGRAMNAME, SUM(TOTALAMOUNT) AS TS
FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMCUSTOMER C ON F.CUSTOMERID = C.CUSTOMERID
JOIN DIMLOYALTYINFO L ON C.LOYALTYINFOID = L.LOYALTYINFOID
WHERE D.YEAR >= 2023
GROUP BY L.PROGRAMNAME;
```

Purpose: Evaluates loyalty program effectiveness by measuring sales contribution from each tier.

QUERY 19: Revenue by Store Manager

This query calculates the revenue generated by each store manager in June 2024:

```sql
SELECT C.MANAGERNAME, SUM(TOTALAMOUNT) AS TS
FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMSTOREDATA C ON F.STOREID = C.STOREID
WHERE D.YEAR = 2023 and D.MONTH = 6
GROUP BY C.MANAGERNAME;
```

Purpose: Evaluates store manager performance for a specific month.

QUERY 20: Average Order Amount by Store Type

This query calculates the average order amount per store along with store name and type for 2023:

```sql
SELECT C.STORENAME, C.STORETYPE, AVG(TOTALAMOUNT) AS TS
FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMSTOREDATA C ON F.STOREID = C.STOREID
WHERE D.YEAR = 2023
GROUP BY C.STORENAME, C.STORETYPE;
```

Purpose: Compares performance between different store types and locations.

QUERY 21: Reading Data from CSV Files

This query demonstrates how to read data from a CSV file stored in a Snowflake stage:

```sql
SELECT $1, $2, $3
FROM
@"testdb"."testschema".teststage/DimCustomerData/DimCustomerData.csv
(FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT');
```

Purpose: Shows how to access external data stored in CSV format using positional references ($1, $2, etc).

QUERY 22: Counting Records in Staged Files

This query counts the number of records in a customer CSV file from a stage:

```sql
SELECT count($1)
FROM
@"testdb"."testschema".teststage/DimCustomerData/DimCustomerData.csv
(FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT');
```

Purpose: Demonstrates how to perform aggregations on staged data before loading it.

QUERY 23: Filtering Data from Files

This query filters records from a customer CSV file where a specific date condition is met:

```sql
SELECT $1, $2, $3, $4, $5, $6
FROM @"testdb"."testschema".teststage/DimCustomerData/DimCustomerData.csv
(FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT')
WHERE $4 = '2000-01-01';
```

Purpose: Shows how to apply WHERE conditions to staged data.

QUERY 24: Joining Data from Multiple Files

This query joins customer and loyalty data from different CSV files to show customer names with their loyalty program tier:

```sql
WITH customerdata AS (
  SELECT $1 as firstt, $12 as loyalid FROM
  @"testdb"."testschema".teststage/DimCustomerData/DimCustomerData.csv
  (FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT')
),

ldata AS (
  SELECT $1 as lid, $3 as pt FROM
  @"testdb"."testschema".teststage/DimLoyaltyInfo/DimLoyaltyInfo.csv
  (FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT')
)

SELECT firstt, pt FROM customerdata c
JOIN ldata l ON c.loyalid = l.lid;
```

Purpose: Demonstrates how to join data from multiple staged files before loading into tables.

QUERY 25: Grouping Data from Multiple Files

This query joins customer and loyalty data, then groups by program tier to count customers in each tier:

```sql
WITH customerdata AS (
  SELECT $1 AS firstt, $12 AS loyalid
  FROM @"testdb"."testschema".teststage/DimCustomerData/DimCustomerData.csv
  (FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT')
),

ldata AS (
  SELECT $1 AS lid, $3 AS pt
  FROM @"testdb"."testschema".teststage/DimLoyaltyInfo/DimLoyaltyInfo.csv
  (FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT')
)

SELECT l.pt, COUNT(*) AS TotalCount
FROM customerdata c
JOIN ldata l ON c.loyalid = l.lid
GROUP BY l.pt;
```

Purpose: Shows how to perform aggregations across joined staged files to get business insights before loading data into tables.