O1 BQ Queries, CTE, MV - KirkYagami 👨‍💻🕵️

Dataset link: drive link

CTE Query Example

Explanation:
In this example, we use a CTE to calculate the average weekly sales and average temperature for each store. We then use these results to identify stores with average weekly sales above the overall average.

```
WITH StoreAverages AS (
  SELECT
    Store,
    AVG(Weekly_Sales) AS avg_weekly_sales,
    AVG(Temperature) AS avg_temperature
  FROM
    `bigdata3844.RevData.walmart`
  GROUP BY
    Store
),
OverallAverages AS (
  SELECT
    AVG(avg_weekly_sales) AS overall_avg_sales
  FROM
    StoreAverages
)
SELECT
  sa.Store,
  sa.avg_weekly_sales,
  sa.avg_temperature
FROM
  StoreAverages sa
JOIN
  OverallAverages oa
ON
  sa.avg_weekly_sales > oa.overall_avg_sales
ORDER BY
  sa.avg_weekly_sales DESC;
```

Documentation Link: BigQuery Common Table Expressions (CTEs)

1. Table Sampling

Explanation:
Table sampling allows you to retrieve a random subset of rows from a table. This is useful for analyzing a sample of data when working with large datasets. The `TABLESAMPLE` clause specifies the percentage of rows to include in the sample. The `LIMIT` clause further restricts the number of rows returned to a maximum of 1000.

```
SELECT *
FROM `bigdata3844.RevData.walmart`
TABLESAMPLE SYSTEM (10 PERCENT)
LIMIT 1000;
```

Documentation Link: BigQuery Table Sampling

2. Multi-Statement Transactions

Explanation:
Multi-statement transactions allow you to execute multiple SQL statements as a single transaction. This ensures that either all statements are executed successfully, or none of them are, maintaining data integrity. In BigQuery, you can use the `BEGIN TRANSACTION` and `COMMIT TRANSACTION` commands to handle this.

```
-- Execute all in one script
BEGIN TRANSACTION;

UPDATE `bigdata3844.RevData.walmart`
SET Fuel_Price = Fuel_Price * 1.05
WHERE Date BETWEEN '2010-02-01' AND '2010-02-28';

UPDATE `bigdata3844.RevData.walmart`
SET Temperature = Temperature * 1.8 + 32
WHERE Date BETWEEN '2010-02-01' AND '2010-02-28';

COMMIT TRANSACTION;
```

Documentation Link: BigQuery Transactions

3. Running Parameterized Queries

Explanation:
Parameterized queries allow you to execute SQL queries with dynamic input values. This is useful for running queries with different parameters without modifying the SQL code. Parameters are specified using the `--parameter` flag in the `bq query` command.

```
-- cloud shell or ubuntu or git bash
bq query \
   --use_legacy_sql=false \
   --parameter=start_date:STRING:'2010-02-01' \
   --parameter=end_date:STRING:'2010-02-28' \
   --parameter=min_sales:FLOAT64:1500000 \
   'SELECT
      Date,
      Store,
      Weekly_Sales,
      Temperature,
      Fuel_Price
    FROM
      `bigdata3844.RevData.walmart`
    WHERE
      Date BETWEEN @start_date AND @end_date
    AND
      Weekly_Sales ⩾ @min_sales
    ORDER BY
      Weekly_Sales DESC;'
```

Note: The use of variables in SQL statements is not supported in BigQuery.

```
-- Not supported
DECLARE start_date DATE DEFAULT '2010-02-01';
DECLARE end_date DATE DEFAULT '2010-02-28';

SELECT AVG(Weekly_Sales) as avg_sales, AVG(Temperature) as avg_temp
FROM `bigdata3844.RevData.walmart`
WHERE Date BETWEEN start_date AND end_date;
```

Documentation Link: BigQuery Parameterized Queries

4. Creating and Running Saved Queries

Explanation:
Saved queries allow you to save and reuse complex SQL queries. You can create a saved query function and then execute it with different parameters to retrieve results efficiently.

Creating a Saved Query:

```
-- Query to get sales by date range
SELECT Date, SUM(Weekly_Sales) as total_sales
FROM `bigdata3844.RevData.walmart`
WHERE Date BETWEEN '2010-02-01' AND '2010-02-28'
```

```
GROUP BY Date
ORDER BY Date;
```

Creating a Saved Function:

```
CREATE OR REPLACE TABLE FUNCTION
`bigdata3844.RevData.get_sales_by_date_range`(start_date DATE, end_date DATE)
AS (
  SELECT Date, SUM(Weekly_Sales) as total_sales
  FROM `bigdata3844.RevData.walmart`
  WHERE Date BETWEEN start_date AND end_date
  GROUP BY Date
  ORDER BY Date
);
```

Running the Saved Query:

```
SELECT *
FROM `bigdata3844.RevData.get_sales_by_date_range`('2010-02-01', '2010-02-
28');
```

Documentation Link: BigQuery Saved Queries

5. Logical Views

Explanation:
Logical views provide a way to create a virtual table based on a SQL query. They do not store data physically but present a dynamic view of the data as per the query definition.

```
CREATE OR REPLACE VIEW `bigdata3844.RevData.walmart_sales_view` AS
SELECT
  Date,
  SUM(Weekly_Sales) as total_sales,
  AVG(Temperature) as avg_temperature,
  AVG(Fuel_Price) as avg_fuel_price,
  SUM(CASE WHEN Holiday_Flag = 1 THEN 1 ELSE 0 END) as holiday_count
FROM `bigdata3844.RevData.walmart`
GROUP BY Date
ORDER BY Date;
```

Querying the Logical View:

```
SELECT *
FROM `bigdata3844.RevData.walmart_sales_view`
```

```
WHERE Date BETWEEN '2010-02-01' AND '2010-03-31';
```

Documentation Link: BigQuery Views

6. Materialized Views

Explanation:
Materialized views are similar to logical views but store a snapshot of the data, which can improve query performance. They are refreshed periodically based on the configuration.

```
CREATE MATERIALIZED VIEW `bigdata3844.RevData.walmart_monthly_sales`
AS SELECT
  FORMAT_DATE('%Y-%m', Date) AS month,
  SUM(Weekly_Sales) as total_sales,
  AVG(Temperature) as avg_temperature,
  AVG(Fuel_Price) as avg_fuel_price,
  AVG(CPI) as avg_cpi,
  AVG(Unemployment) as avg_unemployment
FROM `bigdata3844.RevData.walmart`
GROUP BY month
ORDER BY month;
```

Querying the Materialized View:

```
SELECT *
FROM `bigdata3844.RevData.walmart_monthly_sales`
WHERE month BETWEEN '2010-02' AND '2010-03';
```

Documentation Link: BigQuery Materialized Views