✅ Task 4: Create a Stored Procedure for Easier Updates

🎯 Goal

Bundle all previous steps (table creation, data ingestion, transformations) into a **stored procedure** so it can be reused and updated easily.

🔷 Step 1: Create the Stored Procedure

✅ SQL Query:

SQL

```
CREATE OR REPLACE PROCEDURE
  `thelook_ecommerce.sp_create_load_tables`()
BEGIN

-- Create product_orders_fulfillment table
CREATE OR REPLACE TABLE
  `thelook_ecommerce.product_orders_fulfillment` (
    order_id INT64,
    user_id INT64,
    status STRING,
    product_id INT64,
    created_at TIMESTAMP,
    returned_at TIMESTAMP,
    shipped_at TIMESTAMP,
    delivered_at TIMESTAMP,
    cost NUMERIC,
    sale_price NUMERIC,
    retail_price NUMERIC,
    category STRING,
    name STRING,
    brand STRING,
    department STRING,
    sku STRING,
    distribution_center_id INT64
  );

-- Load product_orders_fulfillment from public dataset
CREATE OR REPLACE TABLE
  `thelook_ecommerce.product_orders_fulfillment` AS
SELECT
  items.*,
  products.id AS product_id_products,
  products.name AS product_name,
```

```
  products.category AS product_category
FROM
  `bigquery-public-data.thelook_ecommerce.order_items` AS items
JOIN
  `bigquery-public-data.thelook_ecommerce.products` AS products
ON
  items.product_id = products.id;

-- Create centers table
CREATE OR REPLACE TABLE
  `thelook_ecommerce.centers` (
    id INT64,
    name STRING,
    latitude FLOAT64,
    longitude FLOAT64,
    point_location GEOGRAPHY
  );

-- Load centers with geography transformation
CREATE OR REPLACE TABLE
  `thelook_ecommerce.centers` AS
SELECT
  id,
  name,
  latitude,
  longitude,
  ST_GEOGPOINT(dcenters.longitude, dcenters.latitude) AS
point_location
FROM
  `bigquery-public-data.thelook_ecommerce.distribution_centers` AS
dcenters;

-- Create customers table
CREATE OR REPLACE TABLE
  `thelook_ecommerce.customers` (
    id INT64,
    first_name STRING,
    last_name STRING,
    email STRING,
    age INT64,
    gender STRING,
    state STRING,
    street_address STRING,
    postal_code STRING,
    city STRING,
    country STRING,
    traffic_source STRING,
    created_at TIMESTAMP,
    latitude FLOAT64,
```

```
    longitude FLOAT64,
    point_location GEOGRAPHY
  );

-- Load customers with geography transformation
CREATE OR REPLACE TABLE
  `thelook_ecommerce.customers` AS
SELECT
  id,
  first_name,
  last_name,
  email,
  age,
  gender,
  state,
  street_address,
  postal_code,
  city,
  country,
  traffic_source,
  created_at,
  latitude,
  longitude,
  ST_GEOGPOINT(users.longitude, users.latitude) AS point_location
FROM
  `bigquery-public-data.thelook_ecommerce.users` AS users;

END;
```

Afficher plus de lignes

Click **Run** to create the stored procedure.

🔷 Step 2: Save the Procedure

- Click **Save** → **Save query**
- Name: `sp_create_load_tables` ✅
- Region: Select your region
- Click **Save**

You'll now see the procedure listed under the **Routines** section of the `thelook_ecommerce` dataset.

🔷 Step 3: Run the Distance Calculation Query Again

✅ SQL Query:

SQL
```
SELECT
  customers.id AS customer_id,
  (
    SELECT
      MIN(ST_DISTANCE(centers.point_location,
customers.point_location)) / 1000
    FROM
      `thelook_ecommerce.centers` AS centers
  ) AS distance_to_closest_center
FROM
  `thelook_ecommerce.customers` AS customers;
```

Click **Run** to view the results.

🏁 Conclusion

🎉 **Well done!** You've successfully built a flexible SQL pipeline using BigQuery.

✅ What You Accomplished:

1. **Created a dataset** and defined schemas for ingestion.
2. **Performed transformations** using SQL and geography functions.
3. **Loaded transformed data** into structured tables.
4. **Wrapped everything into a stored procedure** for easy reuse and updates.
5. Learned how to **calculate distances** using `ST_DISTANCE`.
6. Understood how to **schedule queries** (though not executed in this lab).

💡 Key Takeaways:

- Stored procedures simplify pipeline maintenance.
- Geography functions like `ST_GEOGPOINT` and `ST_DISTANCE` are powerful for location-based analysis.
- BigQuery enables scalable, efficient data transformation directly in SQL.