Exercise: Creating a Product Catalog with Reviews in BigQuery

Objectives

- Understand and apply the concept of LEFT JOIN in SQL.
- Learn about and use Common Table Expressions (CTEs) in SQL queries.
- Explore the creation of nested tables in BigQuery.
- Complete a skeleton SQL statement to create a nested table in BigQuery.

Instructions

Part 1: Understanding LEFT JOIN

LEFT JOIN combines rows from two or more tables based on a related column. It includes all rows from the left table and the matched rows from the right table. If there is no match, the result is NULL for columns from the right table.

Syntax:

```
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Example: Consider two tables, `employees` and `departments`. If you want to list all employees, including those without a department, you would use a LEFT JOIN:

```
SELECT employees.name, departments.name
FROM employees
LEFT JOIN departments
ON employees.department_id = departments.id;
```

Here, every employee is listed, and for those without a department, the department name is NULL.

Part 2: Common Table Expressions (CTEs)

Common Table Expressions (CTEs) are temporary result sets that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. They help organize complex queries for better readability and maintenance.

Syntax:

```
WITH CTE_Name AS (
    SELECT columns
    FROM table
    WHERE conditions
)
SELECT columns
FROM CTE_Name;
```

Example: If you have a complex query with repeated subqueries, you can use a CTE for simplification:

```
WITH TotalSales AS (
    SELECT employee_id, SUM(sales) AS TotalSales
    FROM sales
    GROUP BY employee_id
)
SELECT name
FROM employees
JOIN TotalSales
ON employees.id = TotalSales.employee_id
WHERE TotalSales > 1000;
```

Here, `TotalSales` is a CTE that simplifies the main query.

Part 3: Nested Tables in BigQuery

Nested tables in BigQuery allow for storing and querying complex hierarchical data within a single table. They are essentially arrays of records within a table.

Syntax: You define nested tables by structuring your SELECT statement to aggregate related rows into an array.

Example: Suppose you have a `orders` table and an `order_items` table. In BigQuery, you can aggregate order items into a nested table within the orders table:

```
SELECT
  order_id,
  ARRAY_AGG(STRUCT(item_id, quantity, price)) AS items
FROM
  order_items
GROUP BY
  order_id;
```

Here, each order has a nested array `items` containing all its items.

Importance of Grouping: When creating nested tables, grouping is essential. The `GROUP BY` clause is used to specify how data should be grouped before it is aggregated into a nested array. This ensures that the nested structure accurately represents the one-to-many relationship inherent in the data.

For example, in the `orders` and `order_items` case, `GROUP BY order_id` ensures that all items related to a specific order are grouped together. Then, `ARRAY_AGG` is used to aggregate these grouped items into a single array associated with each order. Without proper grouping, the aggregation would not correctly represent the relationship between orders and their items.

Part 4: SQL Query Completion

Complete the following SQL statement to create a table in BigQuery that includes product details along with nested product reviews. The source tables are located in the `[YOUR PROJECT].adventureworks` dataset and include `product`, `productsubcategory`, `productcategory`, and `productreview`.

Query Skeleton

```
CREATE OR REPLACE TABLE `[YOUR
 ↪  PROJECT].adventureworks.dwh_product_with_reviews` AS
WITH
  product_details AS (
    SELECT
      -- [Add the necessary columns and JOIN logic here]
    FROM
      `[YOUR PROJECT].adventureworks.product` p
    -- [Specify the LEFT JOINs with the productsubcategory and
     ↪  productcategory tables]
  ),
  product_reviews AS (
    SELECT
      productid,
      -- [Use ARRAY_AGG to create a nested structure of reviews]
    FROM
      `[YOUR PROJECT].adventureworks.productreview`
    -- [Complete the GROUP BY clause]
  )
-- [Complete the final SELECT statement joining product_details and
 ↪  product_reviews]
```

Exercise Part 2: Analyzing Product Ratings in BigQuery

Objectives

- Understand and apply aggregate functions (AVG, MIN, MAX, COUNT) in SQL.
- Grasp the concept and usage of the GROUP BY clause in SQL queries.
- Explore the UNNEST function for handling nested fields in BigQuery.
- Complete a skeleton SQL statement to analyze product ratings.

Instructions

Part 1: Aggregate Functions and Grouping

Understanding GROUP BY The Concept of GROUP BY:

In SQL, the GROUP BY clause is used to group rows that share a common attribute, creating a summary row for each group. It's an essential tool for aggregating data, which means combining multiple rows of data into a single result row. This clause is typically used with aggregate functions, like AVG, SUM, MIN, MAX, and COUNT, to perform calculations on each group of data.

Why Use GROUP BY:

- **Data Summarization:** GROUP BY enables you to create a summarized view of your data. For instance, you can calculate the total sales per region, average ratings per product, or count the number of orders per customer.
- **Statistical Analysis:** It's vital for statistical calculations across different groups in a dataset, like finding the maximum or minimum value in each category.
- **Data Organization:** It helps in organizing data in a way that makes it easier to understand patterns and insights.

Detailed Example:

Consider a `customer_orders` table with columns `customer_id`, `order_date`, and `order_value`. To find the average order value for each customer, you would use the GROUP BY clause like this:

```
SELECT
  customer_id,
  AVG(order_value) as average_order_value
FROM
  customer_orders
```

```
GROUP BY
  customer_id;
```

Here, the GROUP BY customer_id clause groups all orders by each customer. The AVG(order_value) then calculates the average order value for each of these groups, providing a clear insight into each customer's average spending.

Part 2: UNNEST in BigQuery

UNNEST UNNEST is a function used to expand an array or nested field into a set of rows in BigQuery. It's crucial for handling and querying nested data structures.

Detailed Example: Suppose a table product_reviews contains a column review_details as an array of records with fields like rating and comment. To analyze each review entry, use UNNEST:

```
SELECT
  product_id,
  review.rating,
  review.comment
FROM
  product_reviews,
  UNNEST(review_details) as review
```

This query expands each element of the review_details array, allowing for individual analysis of each review.

Part 4: SQL Query Completion

Complete the following SQL statement to analyze the **top ten best-rated products based on their average ratings, along with their minimum, maximum ratings, and total count of ratings**. The data is located in the adventureworks.dwh_product_with_reviews table.

Query Skeleton

```
SELECT
  product_name,
  -- [Calculate the average, minimum, maximum ratings, and count the number
   ↪  of ratings]
FROM
  `adventureworks.dwh_product_with_reviews`,
```

```
  -- [Unnest the reviews array]
GROUP BY
  product_name
-- [Retrieve the top ten products with the highest average rating]
```