Code examples from lectures Views

Views

1. **Create a View for Products with Quantity Greater than 20:**

```
CREATE VIEW myshop.horizontal_view AS
SELECT *
FROM myshop.products
WHERE quantity > 20;
```

   - `CREATE VIEW`: Creates a virtual table based on the result set of a SELECT statement.
   - `AS`: Defines the SELECT statement for the view.
   - Creates a view named `horizontal_view` that shows all columns from `products` where the quantity is greater than 20.

2. **Create a View for Selected Columns from Products Table:**

```
CREATE VIEW myshop.vertical_view AS
SELECT name, quantity
FROM myshop.products;
```

   - Creates a view named `vertical_view` that shows only the `name` and `quantity` columns from the `products` table.

3. **Create a View for Clients with More than One Order:**

```
CREATE VIEW myshop.mixed_view AS
SELECT name
FROM myshop.clients
WHERE client_id IN (SELECT client_id FROM myshop.orders GROUP BY client_id
HAVING COUNT(order_id) > 1);
```

   - `IN`: Checks if a value matches any value in a list or subquery.
   - `GROUP BY`: Groups rows that have the same values into summary rows.
   - `HAVING`: Filters groups based on aggregate functions.

- Creates a view named `mixed_view` that shows the names of clients who have placed more than one order.

4. **Create a View for Products and Their Categories:**

```
CREATE VIEW myshop.join_view AS
SELECT p.name AS product_name, c.name AS category_name
FROM myshop.products AS p
JOIN myshop.product_category AS pc ON p.product_id = pc.product_id
JOIN myshop.categories AS c ON pc.category_id = c.category_id;
```

- `JOIN`: Combines rows from two or more tables based on a related column.
- Creates a view named `join_view` that shows product names alongside their category names.

5. **Create a View for Products and Their Category Count:**

```
CREATE VIEW myshop.subquery_view AS
SELECT p.name, (SELECT COUNT(*) FROM myshop.product_category AS pc WHERE
pc.product_id = p.product_id) AS category_count
FROM myshop.products AS p;
```

- `COUNT`: An aggregate function that returns the number of rows that matches a specified condition.
- Creates a view named `subquery_view` that shows product names and the number of categories they are associated with.

6. **Create a View Combining Product and Category Names:**

```
CREATE VIEW myshop.union_view AS
SELECT name FROM myshop.products
UNION
SELECT name FROM myshop.categories;
```

- `UNION`: Combines the result sets of two or more SELECT statements, removing duplicates.
- Creates a view named `union_view` that combines all product names and category names into a single list.

7. **Create a View Based on Another View with Additional Computation:**

```sql
CREATE VIEW myshop.based_on_other_view AS
SELECT name, quantity * 2 AS double_quantity
FROM myshop.vertical_view;
```

- Creates a view named `based_on_other_view` that doubles the quantities from the `vertical_view`.

8. **Create a View with Check Option for Data Integrity:**

```sql
CREATE VIEW myshop.check_option_view AS
SELECT *
FROM myshop.products
WHERE quantity < 50
WITH CHECK OPTION;
```

- **`WITH CHECK OPTION`**: Ensures that all inserts and updates through the view meet the view's condition.
- Creates a view named `check_option_view` that allows only products with a quantity less than 50 to be inserted or updated.

9. **Create a Materialized View for Total Product Sales:**

```sql
CREATE MATERIALIZED VIEW myshop.total_product_sales AS
SELECT p.name AS product_name, SUM(op.quantity) AS total_quantity
FROM myshop.products AS p
JOIN myshop.ordered_products AS op ON p.product_id = op.product_id
GROUP BY p.name;
```

- **`CREATE MATERIALIZED VIEW`**: Creates a materialized view that stores the result set of a query.
- **`SUM`**: An aggregate function that returns the sum of a numeric column.
- Creates a materialized view named `total_product_sales` that shows total quantities sold for each product.

10. **Refresh the Materialized View:**

```sql
REFRESH MATERIALIZED VIEW myshop.total_product_sales;
```

  - `REFRESH MATERIALIZED VIEW`: Updates the data in the materialized view to reflect the current state of the underlying tables.
  - Refreshes the `total_product_sales` materialized view to update its data.

> Note: Optionally, you can also set up automatic refresh using external scheduling tools or PostgreSQL's event triggers if the database supports that.

Bonus: Common Table Expressions

Common Table Expressions (CTEs) in PostgreSQL are a way to create temporary result sets that can be referenced within a `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement. They are particularly useful for organizing complex queries and improving readability by breaking them into simpler, more manageable parts. A CTE is defined using the `WITH` clause and can be recursive or non-recursive. Recursive CTEs allow queries to refer to themselves, enabling the processing of hierarchical or tree-structured data.

Here's a basic example of a non-recursive CTE:

```
WITH cte_name AS (
    SELECT column1, column2
    FROM table_name
    WHERE condition
)
SELECT *
FROM cte_name;
```

And an example of a recursive CTE:

```
WITH RECURSIVE cte_name AS (
    SELECT column1, column2
    FROM table_name
    WHERE initial_condition
    UNION ALL
    SELECT t.column1, t.column2
    FROM table_name t
    JOIN cte_name c ON t.some_column = c.some_column
)
SELECT *
FROM cte_name;
```

Examples

1. **List of Orders with Client Names and Total Quantities Ordered:**

```
WITH OrderSummary AS (
    SELECT op.order_id, sum(op.quantity) as total_quantity
    FROM ordered_products op
    GROUP BY op.order_id
)
SELECT o.order_id, c.name as client_name, os.total_quantity
FROM orders o
JOIN clients c ON o.client_id = c.client_id
JOIN OrderSummary os ON o.order_id = os.order_id;
```

   - `WITH`: Defines a common table expression (CTE) for temporary result sets.
   - `SUM`: An aggregate function that returns the sum of a numeric column.
   - `GROUP BY`: Groups rows that have the same values into summary rows.
   - `JOIN`: Combines rows from two or more tables based on a related column.
   - `ON`: Specifies the condition for the join.
   - Creates a CTE `OrderSummary` that calculates the total quantity for each order. The main query selects `order_id`, `client_name`, and `total_quantity` by joining `orders`, `clients`, and `OrderSummary`.

2. **Find Top Selling Products:**

```
WITH TotalSales AS (
    SELECT p.product_id, p.name, SUM(op.quantity) as total_sold
    FROM products p
    JOIN ordered_products op ON p.product_id = op.product_id
    GROUP BY p.product_id, p.name
)
SELECT product_id, name, total_sold
FROM TotalSales
WHERE total_sold = (SELECT MAX(total_sold) FROM TotalSales);
```

   - `MAX`: An aggregate function that returns the maximum value.
   - Creates a CTE `TotalSales` that calculates the total quantity sold for each product. The main query selects `product_id`, `name`, and `total_sold` for the product with the maximum `total_sold`.

3. **Clients and Their Last Order Date:**

```sql
WITH LatestOrder AS (
    SELECT client_id, MAX(order_date) as last_order_date
    FROM orders
    GROUP BY client_id
)
SELECT c.name as client_name, lo.last_order_date
FROM clients c
JOIN LatestOrder lo ON c.client_id = lo.client_id;
```

   - Creates a CTE `LatestOrder` that calculates the latest order date for each client. The main query selects `client_name` and `last_order_date` by joining `clients` and `LatestOrder`.

4. **Categorize Clients Based on Order Volume:**

```sql
WITH ClientOrderCount AS (
    SELECT client_id, COUNT(order_id) as num_orders
    FROM orders
    GROUP BY client_id
)
SELECT c.name,
       CASE
           WHEN coc.num_orders > 10 THEN 'High'
           WHEN coc.num_orders BETWEEN 5 AND 10 THEN 'Medium'
           ELSE 'Low'
       END as volume_category
FROM clients c
JOIN ClientOrderCount coc ON c.client_id = coc.client_id;
```

   - **`CASE`**: Provides conditional logic in sql queries.
   - Creates a CTE `ClientOrderCount` that calculates the number of orders for each client. The main query selects `name` and `volume_category` by categorizing clients based on their order volume using `CASE` statements.