PostgreSQL Concepts Explained

I Have To Learn Postgresql , For Following Topics
Give me a Example and Explanation of this topic Clear Way For Understand Everyone ?
And Also Provide Some data in table after these every topics provide some output also For eased to understand

PostgreSQL Truncate table
PostgreSQL temporary table
PostgreSQL Column Alias
PostgreSQL Sequence

Identity Column
PostgreSQL Upsert
PostgreSQL Subquery

Here’s a detailed explanation of each topic, along with examples and sample outputs to make the concepts easy to understand:

---

1. PostgreSQL TRUNCATE TABLE

The `TRUNCATE TABLE` command is used to quickly remove all rows from a table. Unlike `DELETE`, it doesn't log individual row deletions, making it faster. However, it cannot be rolled back unless used inside a transaction.

Example:

```sql
-- Create a sample table
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(50)
);

-- Insert some data
INSERT INTO employees (name, department) VALUES
('Alice', 'HR'),
('Bob', 'Engineering'),
('Charlie', 'Sales');

-- View the data
SELECT * FROM employees;

-- Truncate the table
TRUNCATE TABLE employees;
```

```
-- Check the data again
SELECT * FROM employees;
```

Output:

Before `TRUNCATE`:

```bash
 id |  name   |  department
----+---------+-------------
  1 | Alice   | HR
  2 | Bob     | Engineering
  3 | Charlie | Sales
```

After `TRUNCATE`:

```sql
 id | name | department
----+------+------------
(0 rows)
```

---

2. PostgreSQL Temporary Table

A `TEMPORARY` table is a table that exists only for the duration of a session or transaction. It is automatically dropped at the end of the session.

Example:

```sql
-- Create a temporary table
CREATE TEMPORARY TABLE temp_sales (
    id SERIAL PRIMARY KEY,
    product_name VARCHAR(100),
    quantity INT
);

-- Insert some data
INSERT INTO temp_sales (product_name, quantity) VALUES
('Laptop', 10),
('Phone', 25);

-- View the temporary table data
SELECT * FROM temp_sales;
```

Output:

```bash
 id | product_name | quantity
----+--------------+----------
  1 | Laptop       |       10
  2 | Phone        |       25
```

Note: After you close the session, `` `temp_sales` `` will no longer exist.

---

3. PostgreSQL Column Alias

Column aliases provide a temporary name for columns in the result set. They are useful for improving readability.

Example:

```sql
-- Create a table and insert data
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    price NUMERIC
);

INSERT INTO products (name, price) VALUES
('Laptop', 1000),
('Phone', 800);

-- Use column alias
SELECT name AS product_name, price AS product_price FROM products;
```

Output:

```diff
 product_name | product_price
--------------+---------------
 Laptop       |          1000
 Phone        |           800
```

---

4. PostgreSQL Sequence

A `` `SEQUENCE` `` is a database object that generates unique numbers, typically used for primary key values.

Example:

```sql
-- Create a sequence
CREATE SEQUENCE order_seq START 100 INCREMENT 1;
```

```
-- Use the sequence in a table
CREATE TABLE orders (
    id INT DEFAULT nextval('order_seq'),
    product VARCHAR(100),
    quantity INT
);

-- Insert data
INSERT INTO orders (product, quantity) VALUES ('Laptop', 2), ('Phone', 5);

-- View the data
SELECT * FROM orders;

-- Get the current and next values of the sequence
SELECT currval('order_seq') AS current_value, nextval('order_seq') AS next_value;
```

Output:

Table data:

```bash
 id  | product | quantity
-----+---------+----------
 100 | Laptop  |        2
 101 | Phone   |        5
```

Sequence values:

```markdown
 current_value | next_value
---------------+------------
           101 |        102
```

---

5. Identity Column

An `IDENTITY` column auto-generates values, similar to `SERIAL`, but with more control over its behavior. It's ANSI SQL-compliant.

Example:

```sql
-- Create a table with an identity column
CREATE TABLE customers (
    id INT GENERATED ALWAYS AS IDENTITY,
    name VARCHAR(100),
    email VARCHAR(100)
);

-- Insert data
INSERT INTO customers (name, email) VALUES
('Alice', 'alice@example.com'),
('Bob', 'bob@example.com');
```

```
-- View the data
SELECT * FROM customers;
```

Output:

```
sql


 id |  name   |        email
----+---------+---------------------
  1 | Alice   | alice@example.com
  2 | Bob     | bob@example.com
```

---

6. PostgreSQL Upsert

`UPSERT` allows you to insert a new row or update an existing row if a conflict occurs. It uses the `ON CONFLICT` clause.

Example:

```
sql


-- Create a table
CREATE TABLE inventory (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    stock INT
);

-- Insert data
INSERT INTO inventory (product_id, product_name, stock) VALUES
(1, 'Laptop', 50),
(2, 'Phone', 30);

-- Upsert: Update stock if product_id exists, insert otherwise
INSERT INTO inventory (product_id, product_name, stock) VALUES
(1, 'Laptop', 60), -- Update
(3, 'Tablet', 20)  -- Insert
ON CONFLICT (product_id) DO UPDATE
SET stock = EXCLUDED.stock;

-- View the data
SELECT * FROM inventory;
```

Output:

```
markdown


 product_id | product_name | stock
------------+--------------+-------
          1 | Laptop       |    60
          2 | Phone        |    30
          3 | Tablet       |    20
```

7. PostgreSQL Subquery

A subquery is a query nested within another query. It can be used in `SELECT`, `FROM`, or `WHERE` clauses.

Example:

```
sql

  -- Create a table and insert data
  CREATE TABLE sales (
      id SERIAL PRIMARY KEY,
      product_name VARCHAR(100),
      quantity INT,
      price NUMERIC
  );

  INSERT INTO sales (product_name, quantity, price) VALUES
  ('Laptop', 2, 1000),
  ('Phone', 5, 800),
  ('Tablet', 3, 500);

  -- Use a subquery to find the most expensive product
  SELECT product_name, price FROM sales
  WHERE price = (SELECT MAX(price) FROM sales);
```

Output:

```
diff

   product_name | price
  --------------+-------
   Laptop       |  1000
```

Let me know if you need further clarification on any of these topics!