**Sakina Banu**

**Step-by-Step PostgreSQL SQL Execution (Google Cloud SQL)**

**Step 1: Create Partitioned Table (orders)**

We started by creating a parent table called orders that is partitioned by **order\_date**, using **range partitioning**. The PRIMARY KEY includes the partitioning column (order\_date) to avoid errors.

CREATE TABLE orders (

id SERIAL,

product\_name TEXT NOT NULL,

order\_amount NUMERIC(10, 2),

order\_date DATE NOT NULL,

PRIMARY KEY (id, order\_date)

) PARTITION BY RANGE (order\_date);

**Step 1.1: Create Year-Based Partitions**

We then created two child tables (partitions) — one for 2023 and one for 2024 — using FOR VALUES FROM ... TO.

CREATE TABLE orders\_2023 PARTITION OF orders

FOR VALUES FROM ('2023-01-01') TO ('2024-01-01');

CREATE TABLE orders\_2024 PARTITION OF orders

FOR VALUES FROM ('2024-01-01') TO ('2025-01-01');

A screenshot of a computer

AI-generated content may be incorrect.

**Step 2: Create Index on product\_name**

We added an index on product\_name to improve query performance when filtering or searching by this column.

CREATE INDEX idx\_product\_name ON orders(product\_name);

**Step 3: Insert Sample Data**

We inserted two sample records. PostgreSQL automatically directed each row into the correct partition based on the order\_date.

INSERT INTO orders (product\_name, order\_amount, order\_date)

VALUES

('Laptop', 1200.00, '2023-07-10'),

('Tablet', 450.00, '2024-02-15');

**Step 4: Create View for High-Value Orders**

We created a view called high\_value\_orders that filters all orders with an amount over 1000.

CREATE VIEW high\_value\_orders AS

SELECT id, product\_name, order\_amount, order\_date

FROM orders

WHERE order\_amount > 1000;

A screenshot of a computer code

AI-generated content may be incorrect.

**Step 5: Create Function to Calculate Tax**

We created a function called calculate\_tax that returns 18% tax on any numeric amount.

CREATE FUNCTION calculate\_tax(amount NUMERIC)

RETURNS NUMERIC AS $$

BEGIN

RETURN amount \* 0.18;

END;

$$ LANGUAGE plpgsql;

We used the function like this:

SELECT product\_name, order\_amount, calculate\_tax(order\_amount) AS tax

FROM orders;

**Step 6: Create and Call Procedure to Insert Orders**

We created a stored procedure called insert\_order that allows new records to be added by passing values as parameters.

CREATE PROCEDURE insert\_order(p\_name TEXT, p\_amount NUMERIC, p\_date DATE)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO orders(product\_name, order\_amount, order\_date)

VALUES (p\_name, p\_amount, p\_date);

END;

$$;

We called the procedure like this:

CALL insert\_order('Smartphone', 900.00, '2024-05-08');

A screenshot of a computer code

AI-generated content may be incorrect.

**Step 7: Verification Queries**

We verified the results using the following queries:

* View all 2024 orders:

SELECT \* FROM orders\_2024;

* View all high-value orders:

SELECT \* FROM high\_value\_orders;

* View all orders with calculated tax:

SELECT product\_name, order\_amount, calculate\_tax(order\_amount) AS tax

FROM orders;

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion**

This hands-on practice helped us understand how to:

* Use range partitioning in PostgreSQL
* Create views, functions, and procedures
* Apply indexing for performance
* Organize and query structured data in Google Cloud SQL