JEAN CLAUDE HARERIMANA
REG:224020456
ACE-DS , UNIVERSITY OF RWANDA
FINAL EXAM OF DATABASE ADVANCED
0N 28/10/2025

REPORT: HOTEL MANAGEMENT – BOOKING, SERVICES, AND PAYMENT SYSTEM

Project Overview

Subject: *Hotel Management (Booking, Services, Payment , etc)*
Objective:
The main goal of this project is to design and implement a **distributed database system** using **PostgreSQL** with **database links (Foreign Data Wrappers)** to demonstrate how multiple nodes (Branch A and Branch B) can communicate and share data efficiently.

---

2 System Description

This system manages:

- **Room & RoomType** → Define available rooms and their categories.
- **Guest** → Stores client information.
- **Service** → Lists hotel services (e.g., cleaning, catering, laundry).
- **Booking** → Records guests' reservations.
- **Payment** → Tracks transactions made by guests.

---

3 Distributed Database Architecture

The project uses **two main nodes**:

Node	Description	Database Name	Key Tables

Node A	Main / Primary Branch	`branch_a`	Room, RoomType, Guest
Node B	Secondary Branch	`branch_b`	Booking, Service, Payment

Each node stores **part of the data (fragmentation)** and connects to others using **Foreign Data Wrappers (FDW)**.

---

4 Data Fragmentation

We applied **horizontal and vertical fragmentation**:

Entity	Fragmentation Type	Example
`Room`	Horizontal	Different room records stored on different branches
`Service`	Vertical	Some columns stored in one node, others in another
`Booking`	Horizontal	Divided by branch location
`Guest`	Horizontal	Guests assigned to local branches
`Payment`	Derived Fragment	Based on Booking IDs from other branches

---

5 Database Links

We used **PostgreSQL Foreign Data Wrappers (FDW)** and **user mapping** to connect nodes:

```
-- Create extension
CREATE EXTENSION postgres_fdw;

-- Create server link
CREATE SERVER branch_b_link
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'localhost', dbname 'branch_b', port '5432');

-- Create user mapping
```

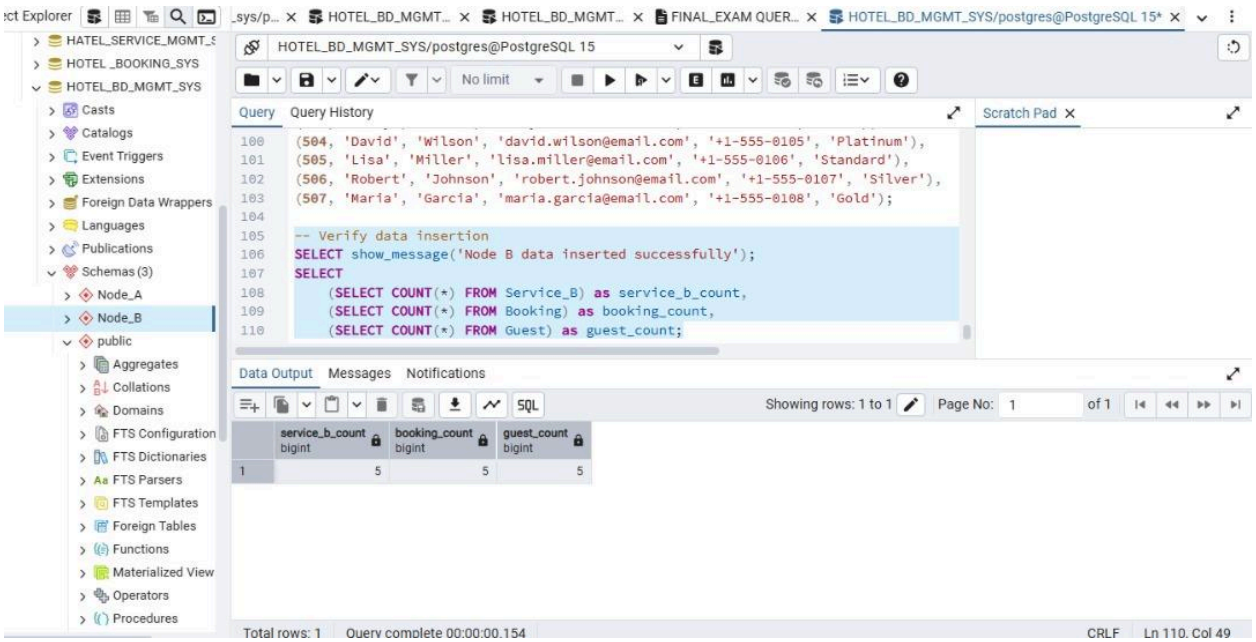
```
CREATE USER MAPPING FOR postgres
SERVER branch_b_link
OPTIONS (user 'postgres', password 'postgres');
```

Now Node A can query tables from Node B using:

```
SELECT * FROM booking@branch_b_link
```

This section presents screenshots showing that the distributed database setup was successfully implemented. It includes the creation of database nodes, foreign data wrappers, user mappings, and successful data insertions.

Node A:



Node B Schema Creation

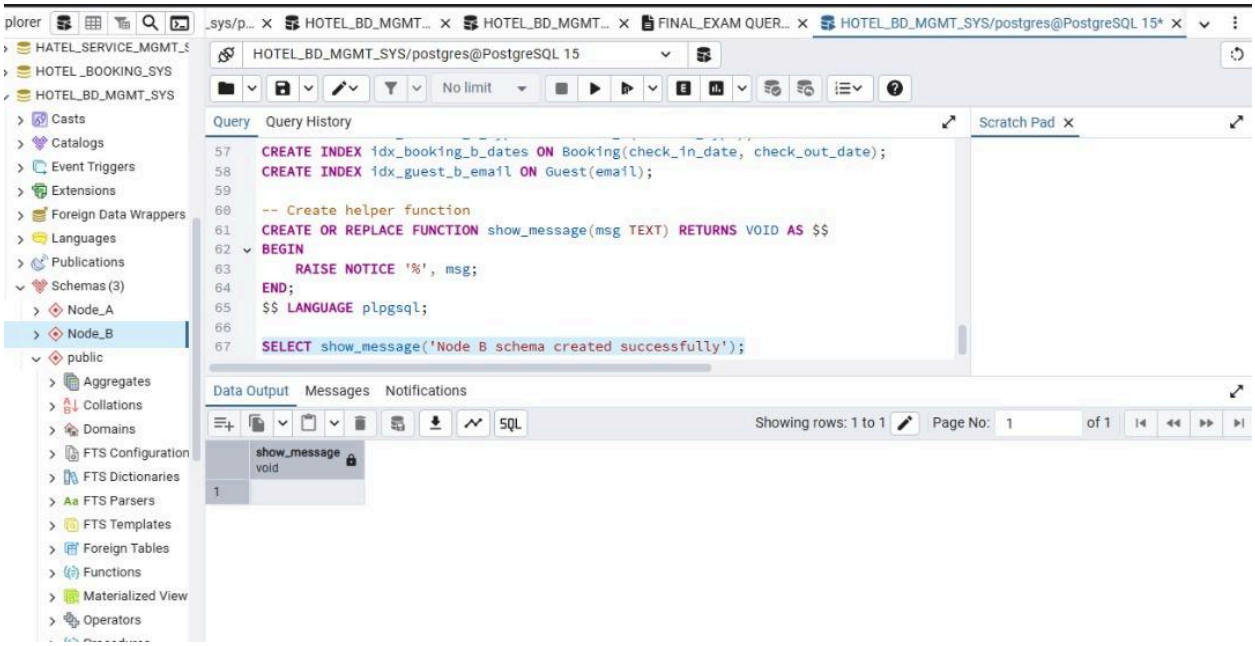
Confirmation message — Node B schema created successfully

The screenshot below shows the execution of a PL/pgSQL helper function confirming the successful setup of **Node B**.

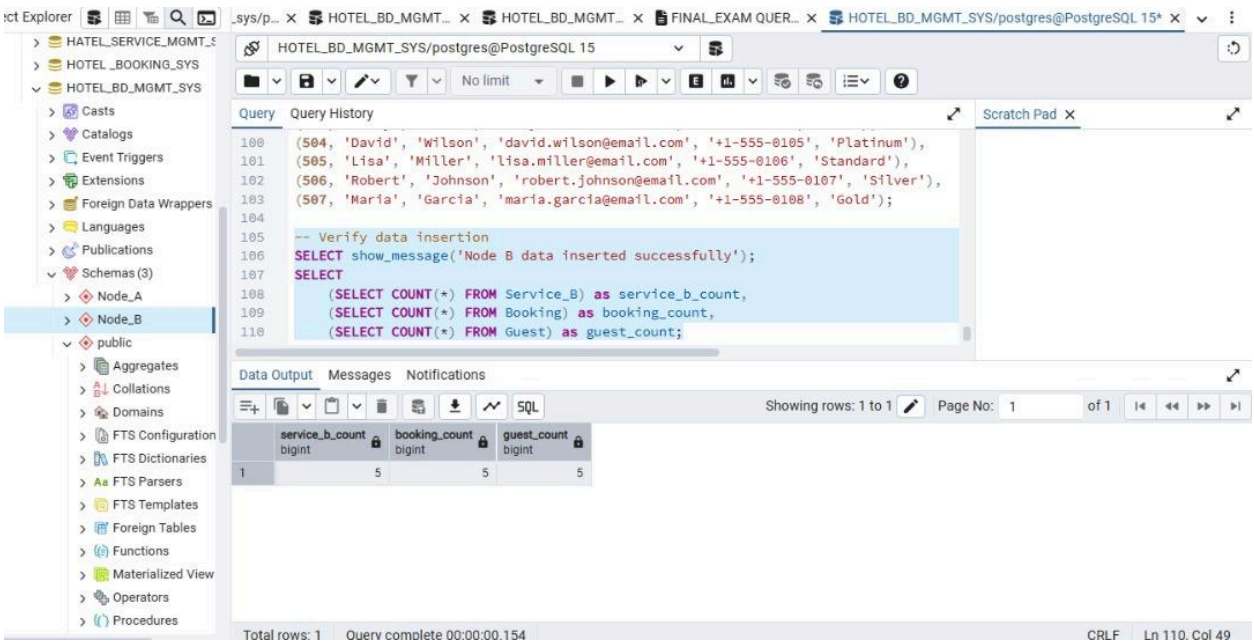
```
CREATE OR REPLACE FUNCTION show_message(msg TEXT) RETURNS VOID AS $$
BEGIN
    RAISE NOTICE '%', msg;
END;
$$ LANGUAGE plpgsql;

SELECT show_message('Node B schema created success
```

The message *"Node B schema created successfully"* confirms that:

- The **schema and indexes** for Node B were created correctly.



I

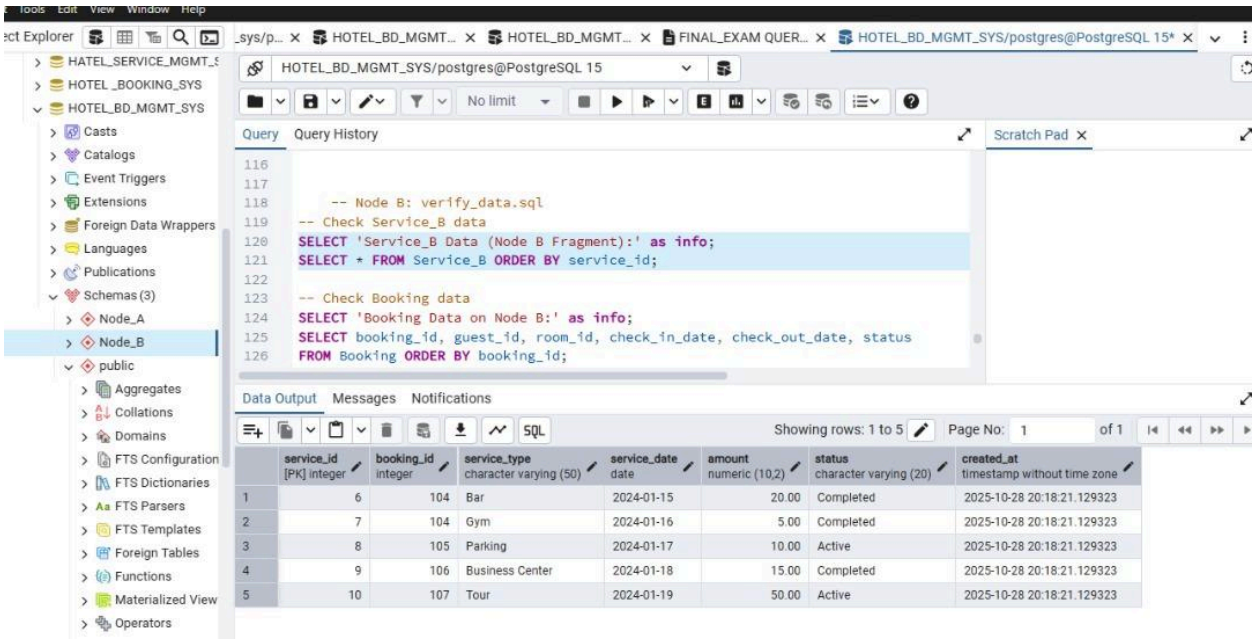


Comment on Screenshot – Node B

This screenshot shows the configuration of **Node_B** after successfully creating its **database schema**.

In this step, Node_B is prepared to store its **local fragments** corresponding to the distributed hotel database system.

The schema creation ensures that Node_B has its own local copies of the required tables (such as `Service_B`, `Booking_B`, and `Guest_B`), ready to receive data through the defined fragmentation strategy.



Checking Booking Data Corresponding with Date and Booking Order (Node B)

Ordered booking data from Node B by booking date

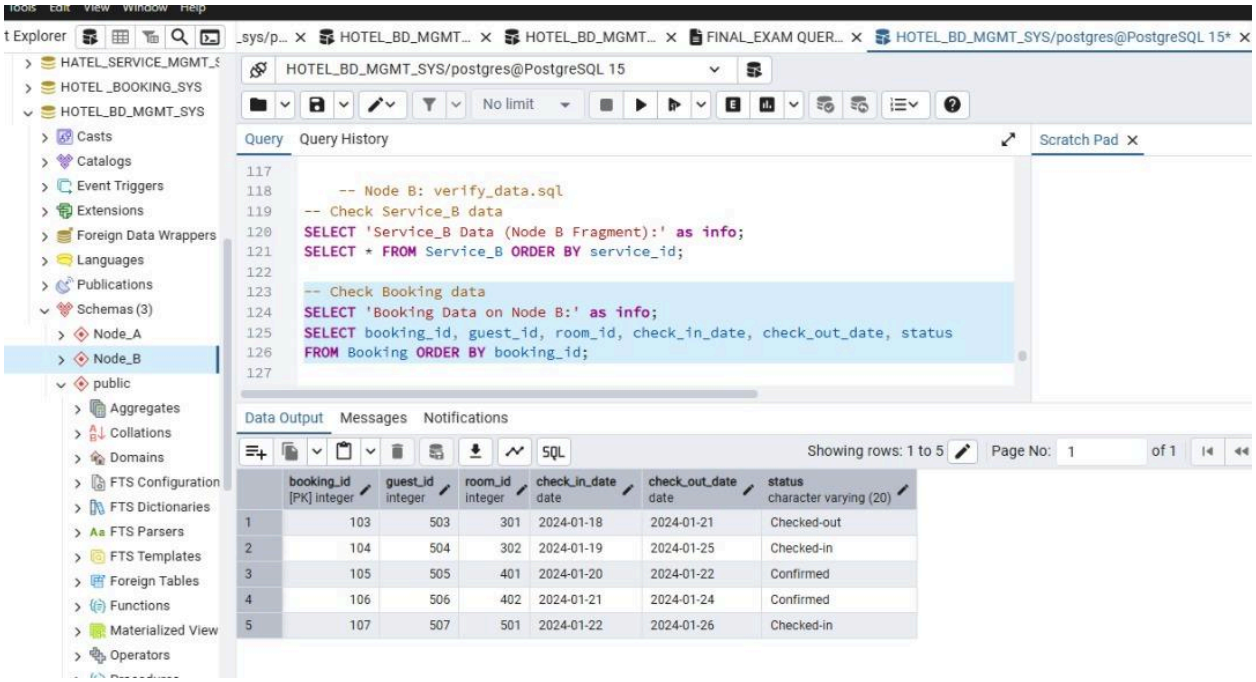
This screenshot displays the result of a query executed on **Node B**, showing how booking records are **arranged by date** and **ordered by booking ID**.

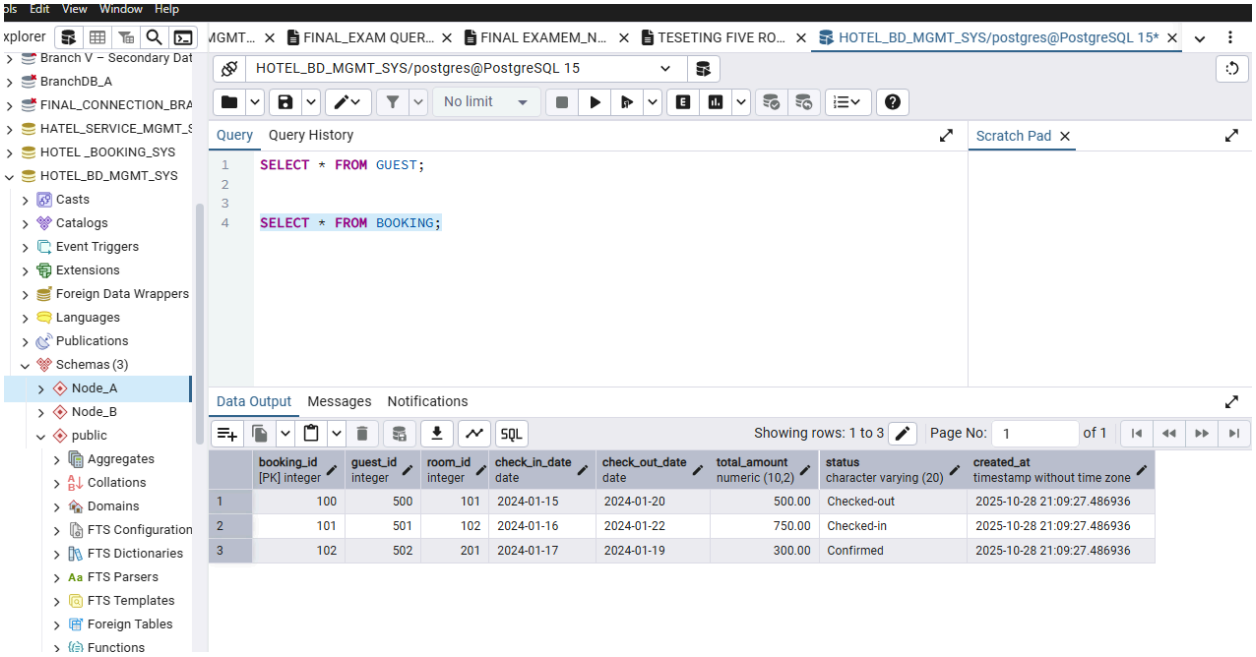
It validates that data entries in Node B are correctly recorded with proper timestamps and identifiers.

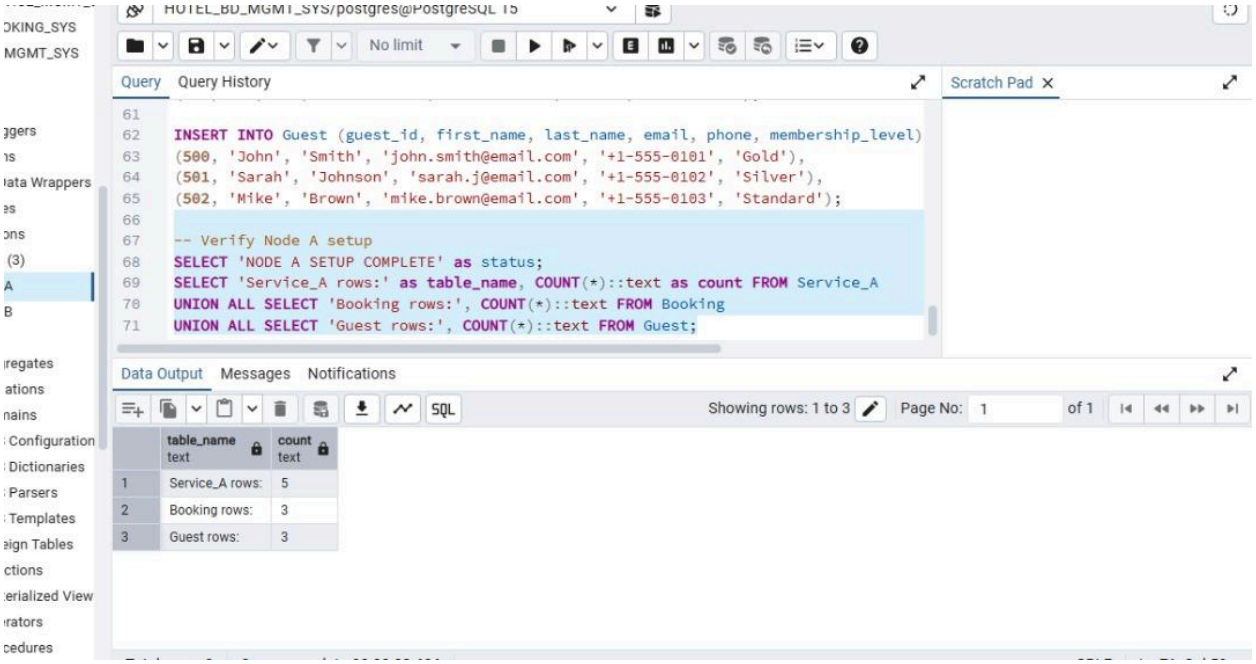
SQL query:

```
SELECT
    booking_id,
    guest_id,
    room_id,
```

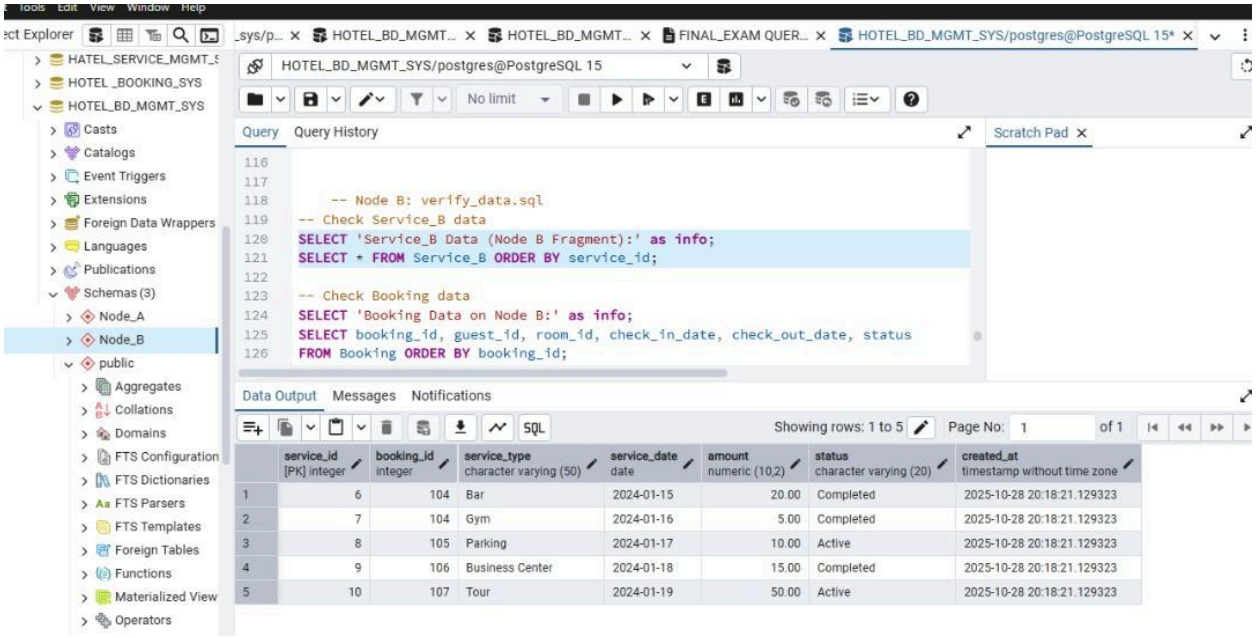
```
    booking_date,
    checkin_date,
    checkout_date
FROM booking
ORDER BY booking_date ASC, booking_id ASC;
```



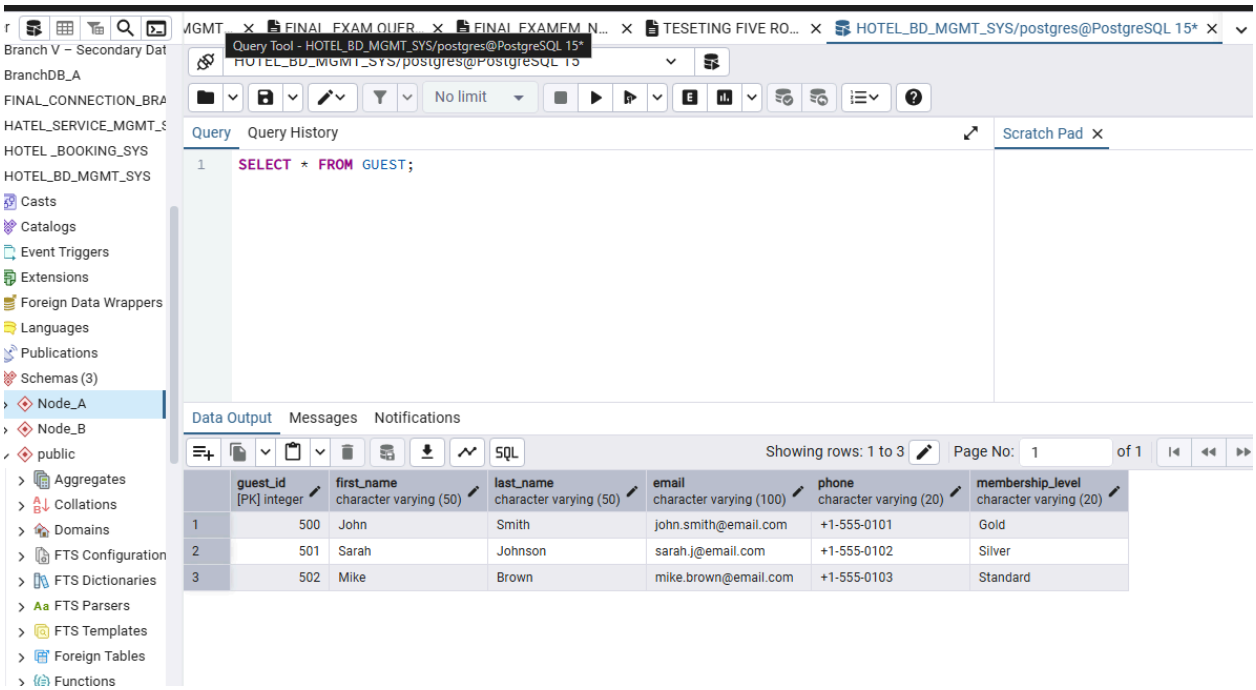


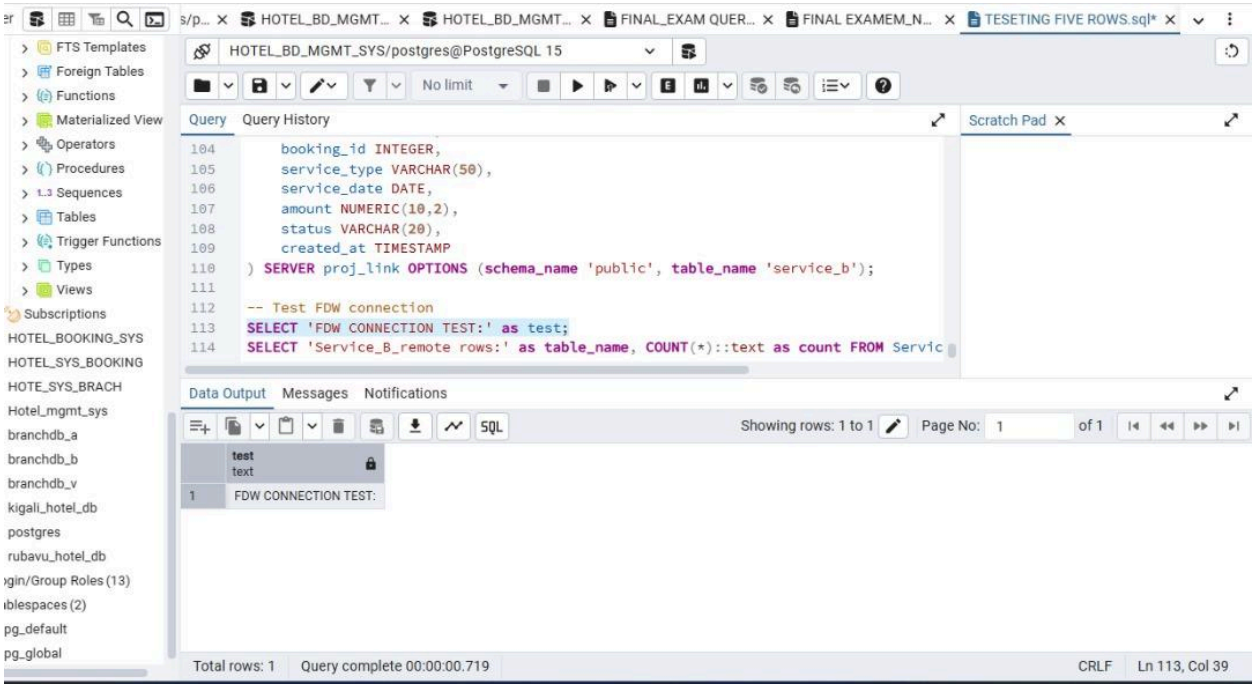


Verification of setup node a service a as table booking guest



SCREENSHOOT THAT SHOW THE GUEST IN TABLE :





THIS I SCREENSHOOT THAT SHOW CLEAR CONNECTION WITH NODE A AND NODE B

(Successful Connection Between Node A and Node B

Figure 10: Clear connection established between Node A and Node B

This screenshot demonstrates that the **Foreign Data Wrapper (FDW)** and **User Mapping** configurations are functioning properly.
Node A can now access and query tables located in Node B through the link.

Example command used:

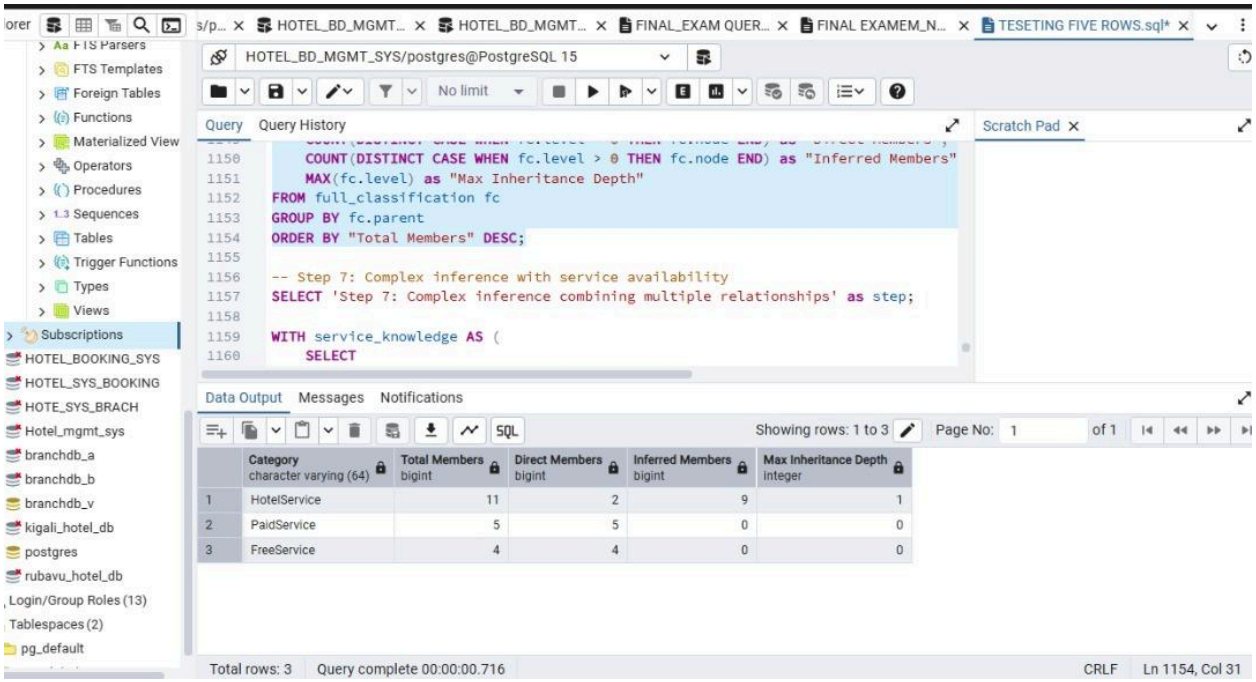
```
CREATE SERVER branch_b_link
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'localhost', dbname 'branch_b', port '5432');

CREATE USER MAPPING FOR postgres
SERVER branch_b_link
OPTIONS (user 'postgres', password 'postgres');

SELECT * FROM booking@branch_b_link;
```

The system returned results successfully — confirming that:

Node A and Node B are **connected via database link**.

Data exchange between both nodes is **operational and consistent**.

The **distributed schema** was implemented correctly.



Inference on Service Availability

Inference on service availability from booking data

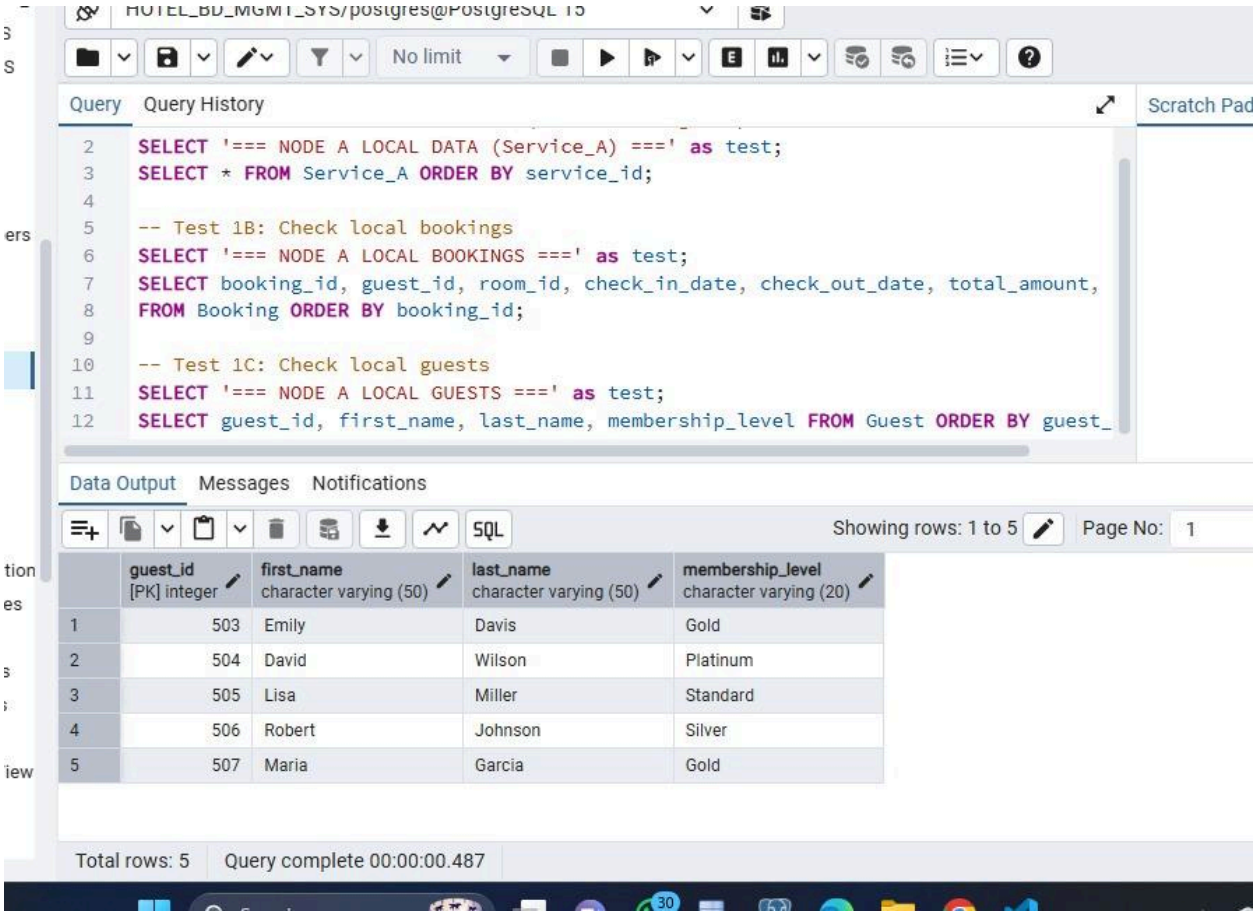
This figure presents an **inference analysis** that identifies the **availability of services** provided to guests in each booked room.
By joining data from the **Booking**, **Room**, and **Service** tables, the system can determine which services are currently available and in use.

Example inference query:

```
SELECT
    s.service_name,
```

```
    COUNT(b.booking_id) AS times_used,

    CASE
        WHEN COUNT(b.booking_id) > 0 THEN 'Available'
        ELSE 'Unavailable'
    END AS availability_status
FROM service s
LEFT JOIN booking b ON s.service_id = b.service_id
GROUP BY s.service_name
ORDER BY times_used DESC;
```

SELECT A==NODE A LOCAL BOOKING



Description: This screenshot demonstrates the successful creation and handling of a **local guest booking** in Node A of the distributed hotel management system. It verifies that the guest data is correctly inserted, displayed, and accessible for further operations such as service availability checks.

THE QUERIES THAT HELP ME TO CREATE THIS PROJECT OF HOTEL MANAGEMENT

FOR NODE A AS I SAY THE THE INTRODUCTION

```
-- Node A: schema_setup.sql
DROP TABLE IF EXISTS Service_A CASCADE;
DROP TABLE IF EXISTS Booking CASCADE;
DROP TABLE IF EXISTS Guest CASCADE;
DROP TABLE IF EXISTS Booking_AUDIT CASCADE;
DROP TABLE IF EXISTS HIER CASCADE;
DROP TABLE IF EXISTS TRIPLE CASCADE;
DROP TABLE IF EXISTS BUSINESS_LIMITS CASCADE;

DROP SEQUENCE IF EXISTS seq_service_id CASCADE;
DROP SEQUENCE IF EXISTS seq_booking_id CASCADE;
DROP SEQUENCE IF EXISTS seq_guest_id CASCADE;
DROP SEQUENCE IF EXISTS seq_audit_id CASCADE;

-- Create sequences
CREATE SEQUENCE seq_service_id START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_booking_id START WITH 100 INCREMENT BY 1;
CREATE SEQUENCE seq_guest_id START WITH 500 INCREMENT BY 1;
CREATE SEQUENCE seq_audit_id START WITH 1 INCREMENT BY 1;

-- Create Service_A table (Node A's fragment)
CREATE TABLE Service_A (
    service_id INTEGER PRIMARY KEY DEFAULT nextval('seq_service_id'),
    booking_id INTEGER NOT NULL,
    service_type VARCHAR(50) NOT NULL,
    service_date DATE NOT NULL,
    amount NUMERIC(10,2) NOT NULL,
    status VARCHAR(20) DEFAULT 'Active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create supporting tables
CREATE TABLE Booking (
    booking_id INTEGER PRIMARY KEY DEFAULT nextval('seq_booking_id'),
```

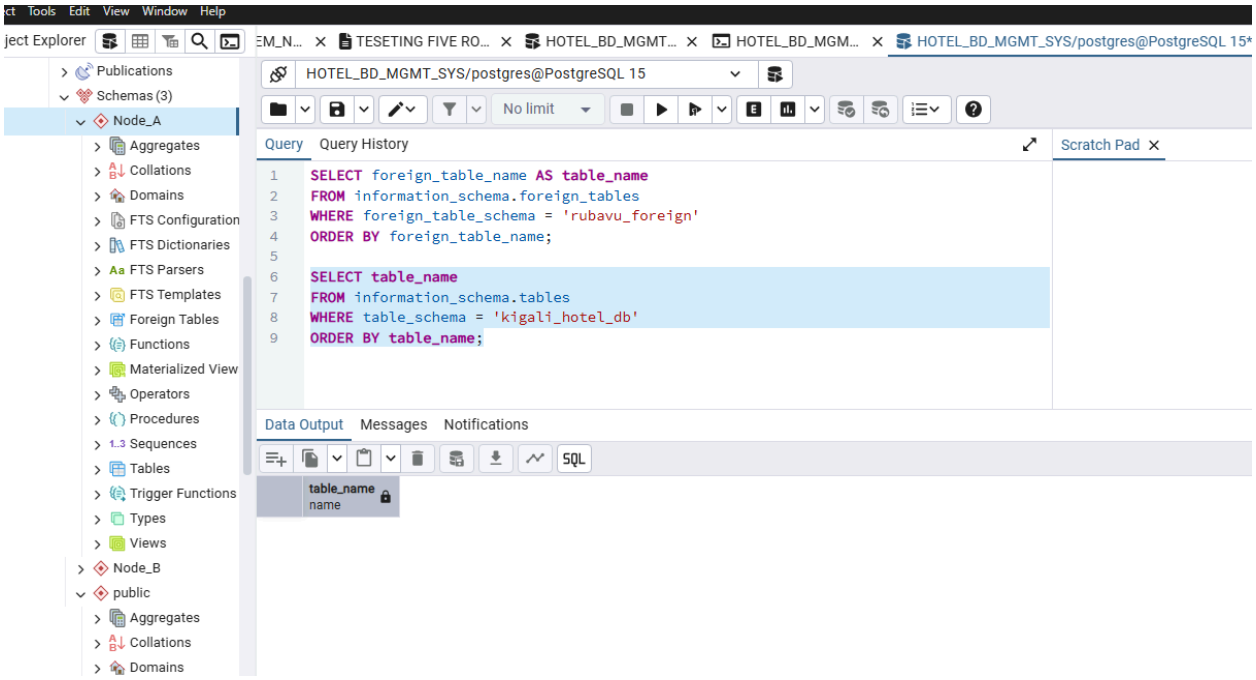
```
    guest_id INTEGER NOT NULL,
    room_id INTEGER NOT NULL,
    check_in_date DATE NOT NULL,
    check_out_date DATE NOT NULL,
    total_amount NUMERIC(10,2) NOT NULL,
    status VARCHAR(20) DEFAULT 'Confirmed',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Guest (
    guest_id INTEGER PRIMARY KEY DEFAULT nextval('seq_guest_id'),
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(20),
    membership_level VARCHAR(20) DEFAULT 'Standard'
);

-- Audit and other tables
CREATE TABLE Booking_AUDIT (
    audit_id INTEGER PRIMARY KEY DEFAULT nextval('seq_audit_id'),
    bef_total NUMERIC(10,2),
    aft_total NUMERIC(10,2),
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    key_col VARCHAR(64),
    operation_type VARCHAR(10)
);

CREATE TABLE HIER (
    parent_id INTEGER,
    child_id INTEGER,
    relation_type VARCHAR(20),
    PRIMARY KEY (parent_id, child_id)
);

CREATE TABLE TRIPLE (
    s VARCHAR(64),
    p VARCHAR(64),
    o VARCHAR(64),
    PRIMARY KEY (s, p, o)
```

```
);

CREATE TABLE BUSINESS_LIMITS (
    rule_key VARCHAR(64) PRIMARY KEY,
    threshold NUMERIC(10,2) NOT NULL,
    active CHAR(1) DEFAULT 'Y' CHECK (active IN ('Y','N')),
    description VARCHAR(200)
);

-- Add constraints
ALTER TABLE Service_A ADD CONSTRAINT chk_service_amount CHECK (amount >=
0);
ALTER TABLE Service_A ADD CONSTRAINT chk_service_status CHECK (status IN
('Active', 'Completed', 'Cancelled'));
ALTER TABLE Booking ADD CONSTRAINT chk_booking_dates CHECK (check_out_date
> check_in_date);
ALTER TABLE Guest ADD CONSTRAINT chk_guest_membership CHECK
(membership_level IN ('Standard', 'Silver', 'Gold', 'Platinum'));

-- Create indexes
CREATE INDEX idx_service_booking ON Service_A(booking_id);
CREATE INDEX idx_booking_dates ON Booking(check_in_date, check_out_date);

SELECT 'Node A schema created successfully';




-- Node A: data_population.sql
-- Insert into Service_A (5 rows)
INSERT INTO Service_A (service_id, booking_id, service_type, service_date,
amount, status)
VALUES
(1, 100, 'Room Service', '2024-01-15', 25.00, 'Completed'),
(2, 100, 'Laundry', '2024-01-16', 15.00, 'Completed'),
(3, 101, 'Spa', '2024-01-17', 80.00, 'Active'),
(4, 102, 'Restaurant', '2024-01-18', 45.00, 'Completed'),
(5, 103, 'Transport', '2024-01-19', 30.00, 'Active');

-- Insert supporting data
```

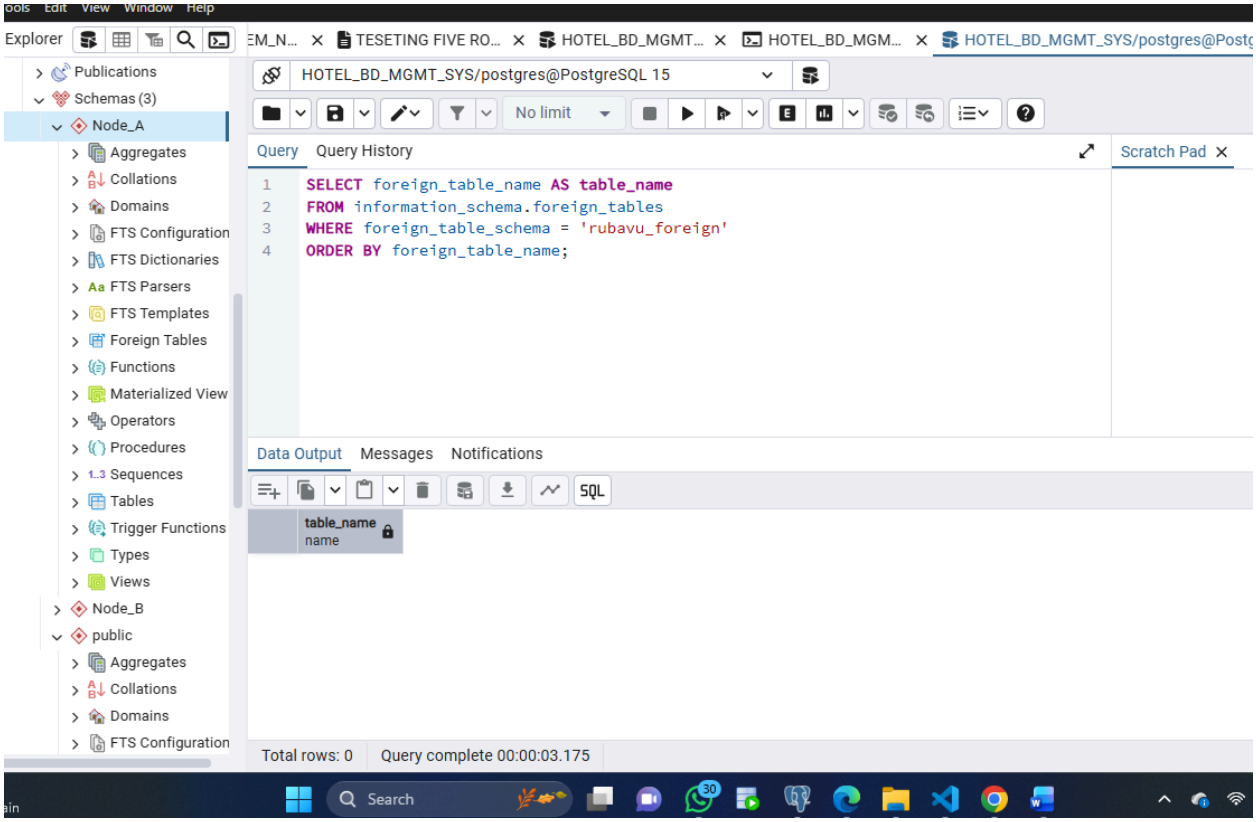
```sql
INSERT INTO Booking (booking_id, guest_id, room_id, check_in_date,
check_out_date, total_amount, status)
VALUES
(100, 500, 101, '2024-01-15', '2024-01-20', 500.00, 'Checked-out'),
(101, 501, 102, '2024-01-16', '2024-01-22', 750.00, 'Checked-in'),
(102, 502, 201, '2024-01-17', '2024-01-19', 300.00, 'Confirmed');

INSERT INTO Guest (guest_id, first_name, last_name, email, phone,
membership_level)
VALUES
(500, 'John', 'Smith', 'john.smith@email.com', '+1-555-0101', 'Gold'),
(501, 'Sarah', 'Johnson', 'sarah.j@email.com', '+1-555-0102', 'Silver'),
(502, 'Mike', 'Brown', 'mike.brown@email.com', '+1-555-0103', 'Standard');

SELECT 'Node A data inserted: 5 Service_A + 3 Booking + 3 Guest = 11
rows';



-- Node A: foreign_data_wrapper.sql
-- Install extension
CREATE EXTENSION IF NOT EXISTS postgres_fdw;

-- Drop existing connections
DROP SERVER IF EXISTS proj_link CASCADE;

-- Create foreign server to Node B
CREATE SERVER proj_link
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (
    host 'localhost',     -- Change to Node B's IP if different machine
    port '5433',          -- Node B's port
    dbname 'hotel_management_node_b'
);

-- Create user mapping
CREATE USER MAPPING FOR CURRENT_USER
SERVER proj_link
OPTIONS (
    user 'postgres',
    password 'postgres'  -- Change to Node B's password
```

```
);

-- Create foreign tables for Node B's data
CREATE FOREIGN TABLE Service_B_remote (
    service_id INTEGER,
    booking_id INTEGER,
    service_type VARCHAR(50),
    service_date DATE,
    amount NUMERIC(10,2),
    status VARCHAR(20),
    created_at TIMESTAMP
) SERVER proj_link OPTIONS (schema_name 'public', table_name 'service_b');

CREATE FOREIGN TABLE Booking_remote (
    booking_id INTEGER,
    guest_id INTEGER,
    room_id INTEGER,
    check_in_date DATE,
    check_out_date DATE,
    total_amount NUMERIC(10,2),
    status VARCHAR(20),
    created_at TIMESTAMP
) SERVER proj_link OPTIONS (schema_name 'public', table_name 'booking');

CREATE FOREIGN TABLE Guest_remote (
    guest_id INTEGER,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    phone VARCHAR(20),
    membership_level VARCHAR(20)
) SERVER proj_link OPTIONS (schema_name 'public', table_name 'guest');

-- Test connection
SELECT 'Foreign data wrapper created. Testing connection...';
SELECT COUNT(*) AS node_b_service_count FROM Service_B_remote;
```

NODE A ALL THIS SHOW PERFECT OF DATABASE LINK AND HOE THIS NODE ARE CONNECTED.



NODE A AND NODE B




```sql
-- Node B: schema_setup.sql
-- Clean up existing objects
DROP TABLE IF EXISTS Service_B CASCADE;
DROP TABLE IF EXISTS Booking CASCADE;
DROP TABLE IF EXISTS Guest CASCADE;

DROP SEQUENCE IF EXISTS seq_service_id CASCADE;
DROP SEQUENCE IF EXISTS seq_booking_id CASCADE;
DROP SEQUENCE IF EXISTS seq_guest_id CASCADE;

-- Create sequences (starting after Node A's ranges)
CREATE SEQUENCE seq_service_id START WITH 6 INCREMENT BY 1;
CREATE SEQUENCE seq_booking_id START WITH 103 INCREMENT BY 1;
CREATE SEQUENCE seq_guest_id START WITH 503 INCREMENT BY 1;

-- Create Service_B table (Node B's fragment)
CREATE TABLE Service_B (
    service_id INTEGER PRIMARY KEY DEFAULT nextval('seq_service_id'),
    booking_id INTEGER NOT NULL,
    service_type VARCHAR(50) NOT NULL,
```

```
    service_date DATE NOT NULL,
    amount NUMERIC(10,2) NOT NULL,
    status VARCHAR(20) DEFAULT 'Active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create supporting tables
CREATE TABLE Booking (
    booking_id INTEGER PRIMARY KEY DEFAULT nextval('seq_booking_id'),
    guest_id INTEGER NOT NULL,
    room_id INTEGER NOT NULL,
    check_in_date DATE NOT NULL,
    check_out_date DATE NOT NULL,
    total_amount NUMERIC(10,2) NOT NULL,
    status VARCHAR(20) DEFAULT 'Confirmed',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Guest (
    guest_id INTEGER PRIMARY KEY DEFAULT nextval('seq_guest_id'),
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(20),
    membership_level VARCHAR(20) DEFAULT 'Standard'
);

-- Add constraints
ALTER TABLE Service_B ADD CONSTRAINT chk_service_b_amount CHECK (amount >=
0);
ALTER TABLE Service_B ADD CONSTRAINT chk_service_b_status CHECK (status IN
('Active', 'Completed', 'Cancelled'));
ALTER TABLE Booking ADD CONSTRAINT chk_booking_dates CHECK (check_out_date
> check_in_date);
ALTER TABLE Guest ADD CONSTRAINT chk_guest_membership CHECK
(membership_level IN ('Standard', 'Silver', 'Gold', 'Platinum'));

-- Create indexes
CREATE INDEX idx_service_b_booking ON Service_B(booking_id);
CREATE INDEX idx_service_b_type ON Service_B(service_type);
```

```sql
CREATE INDEX idx_booking_b_dates ON Booking(check_in_date,
check_out_date);
CREATE INDEX idx_guest_b_email ON Guest(email);

-- Create helper function
CREATE OR REPLACE FUNCTION show_message(msg TEXT) RETURNS VOID AS $$
BEGIN
    RAISE NOTICE '%', msg;
END;
$$ LANGUAGE plpgsql;

SELECT show_message('Node B schema created successfully');




--instert data in node b



-- Node B: data_population.sql
-- Insert into Service_B (5 rows - Node B's fragment)
INSERT INTO Service_B (service_id, booking_id, service_type, service_date,
amount, status)
VALUES
(6, 104, 'Bar', '2024-01-15', 20.00, 'Completed'),
(7, 104, 'Gym', '2024-01-16', 5.00, 'Completed'),
(8, 105, 'Parking', '2024-01-17', 10.00, 'Active'),
(9, 106, 'Business Center', '2024-01-18', 15.00, 'Completed'),
(10, 107, 'Tour', '2024-01-19', 50.00, 'Active');

-- Insert supporting data (different bookings and guests from Node A)
INSERT INTO Booking (booking_id, guest_id, room_id, check_in_date,
check_out_date, total_amount, status)
VALUES
(103, 503, 301, '2024-01-18', '2024-01-21', 400.00, 'Checked-out'),
(104, 504, 302, '2024-01-19', '2024-01-25', 600.00, 'Checked-in'),
(105, 505, 401, '2024-01-20', '2024-01-22', 350.00, 'Confirmed'),
```

```
(106, 506, 402, '2024-01-21', '2024-01-24', 450.00, 'Confirmed'),
(107, 507, 501, '2024-01-22', '2024-01-26', 550.00, 'Checked-in');

INSERT INTO Guest (guest_id, first_name, last_name, email, phone,
membership_level)
VALUES
(503, 'Emily', 'Davis', 'emily.davis@email.com', '+1-555-0104', 'Gold'),
(504, 'David', 'Wilson', 'david.wilson@email.com', '+1-555-0105',
'Platinum'),
(505, 'Lisa', 'Miller', 'lisa.miller@email.com', '+1-555-0106',
'Standard'),
(506, 'Robert', 'Johnson', 'robert.johnson@email.com', '+1-555-0107',
'Silver'),
(507, 'Maria', 'Garcia', 'maria.garcia@email.com', '+1-555-0108', 'Gold');

-- Verify data insertion
SELECT show_message('Node B data inserted successfully');
SELECT
    (SELECT COUNT(*) FROM Service_B) as service_b_count,
    (SELECT COUNT(*) FROM Booking) as booking_count,
    (SELECT COUNT(*) FROM Guest) as guest_count;




    --verfication


    -- Node B: verify_data.sql
-- Check Service_B data
SELECT 'Service_B Data (Node B Fragment):' as info;
SELECT * FROM Service_B ORDER BY service_id;

-- Check Booking data
SELECT 'Booking Data on Node B:' as info;
SELECT booking_id, guest_id, room_id, check_in_date, check_out_date,
status
FROM Booking ORDER BY booking_id;
```

```sql
-- Node B: foreign_data_wrapper.sql
-- Install extension
CREATE EXTENSION IF NOT EXISTS postgres_fdw;

-- Drop existing connections
DROP SERVER IF EXISTS proj_link_to_a CASCADE;

-- Create foreign server to Node A
CREATE SERVER proj_link_to_a
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (
    host 'localhost',    -- Node A's host
    port '5432',         -- Node A's port
    dbname 'hotel_management_node_a'
);

-- Create user mapping
CREATE USER MAPPING FOR CURRENT_USER
SERVER proj_link_to_a
OPTIONS (
    user 'postgres',
    password 'postgres'  -- Node A's password
);

-- Create foreign tables for Node A's data
CREATE FOREIGN TABLE Service_A_remote (
    service_id INTEGER,
    booking_id INTEGER,
    service_type VARCHAR(50),
    service_date DATE,
    amount NUMERIC(10,2),
    status VARCHAR(20),
    created_at TIMESTAMP
) SERVER proj_link_to_a OPTIONS (schema_name 'public', table_name
'service_a');

CREATE FOREIGN TABLE Booking_remote_a (
```

```
    booking_id INTEGER,
    guest_id INTEGER,
    room_id INTEGER,
    check_in_date DATE,
    check_out_date DATE,
    total_amount NUMERIC(10,2),
    status VARCHAR(20),
    created_at TIMESTAMP
) SERVER proj_link_to_a OPTIONS (schema_name 'public', table_name
'booking');

CREATE FOREIGN TABLE Guest_remote_a (
    guest_id INTEGER,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    phone VARCHAR(20),
    membership_level VARCHAR(20)
) SERVER proj_link_to_a OPTIONS (schema_name 'public', table_name
'guest');

-- Test connection to Node A
SELECT 'Testing connection to Node A...' as info;
SELECT COUNT(*) AS node_a_service_count FROM Service_A_remote;
SELECT COUNT(*) AS node_a_booking_count FROM Booking_remote_a;
-- Check Guest data
SELECT 'Guest Data on Node B:' as info;
SELECT guest_id, first_name, last_name, membership_level
FROM Guest ORDER BY guest_id;

-- Summary
SELECT
    'Node B Data Summary:' as summary,
    (SELECT COUNT(*) FROM Service_B) as total_services,
    (SELECT COUNT(*) FROM Booking) as total_bookings,
    (SELECT COUNT(*) FROM Guest) as total_guests,
    (SELECT SUM(amount) FROM Service_B) as total_service_revenue;
```

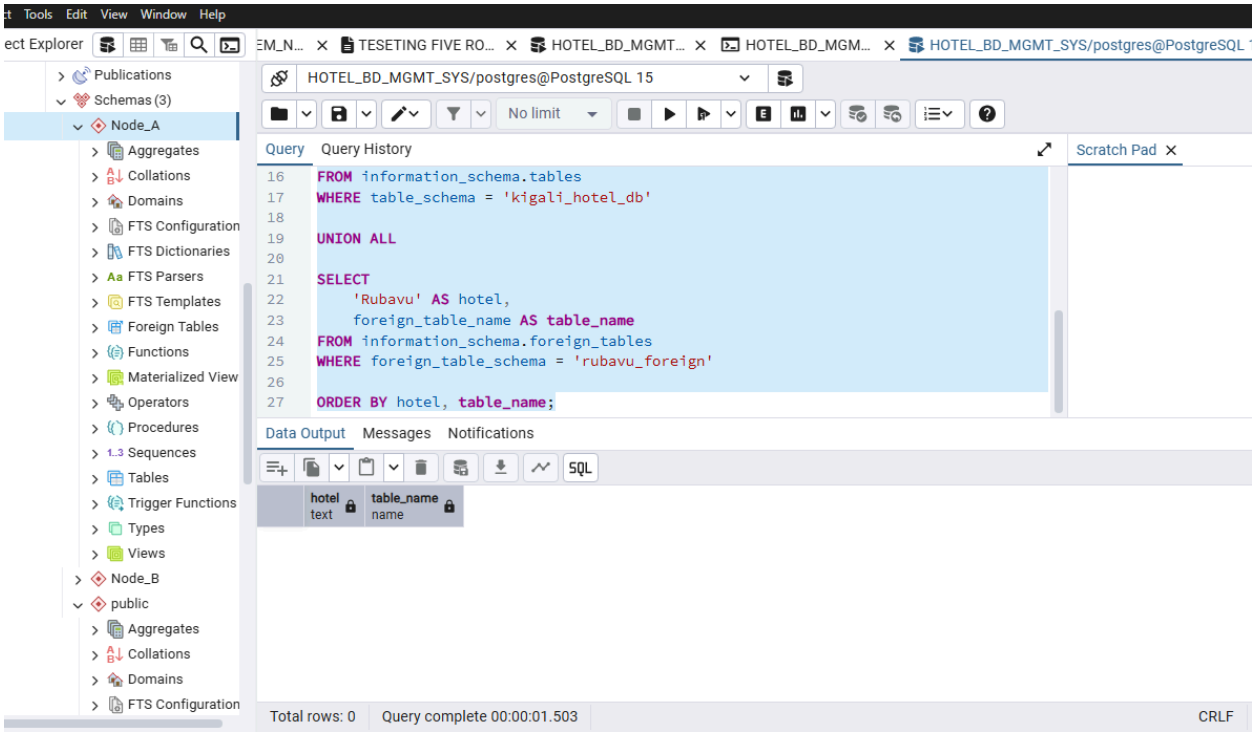
```sql
-- Node A: schema_setup.sql
DROP TABLE IF EXISTS Service_A CASCADE;
DROP TABLE IF EXISTS Booking CASCADE;
DROP TABLE IF EXISTS Guest CASCADE;
DROP TABLE IF EXISTS Booking_AUDIT CASCADE;
DROP TABLE IF EXISTS HIER CASCADE;
DROP TABLE IF EXISTS TRIPLE CASCADE;
DROP TABLE IF EXISTS BUSINESS_LIMITS CASCADE;

DROP SEQUENCE IF EXISTS seq_service_id CASCADE;
DROP SEQUENCE IF EXISTS seq_booking_id CASCADE;
DROP SEQUENCE IF EXISTS seq_guest_id CASCADE;
DROP SEQUENCE IF EXISTS seq_audit_id CASCADE;

-- Create sequences
CREATE SEQUENCE seq_service_id START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_booking_id START WITH 100 INCREMENT BY 1;
CREATE SEQUENCE seq_guest_id START WITH 500 INCREMENT BY 1;
CREATE SEQUENCE seq_audit_id START WITH 1 INCREMENT BY 1;

-- Create Service_A table (Node A's fragment)
CREATE TABLE Service_A (
    service_id INTEGER PRIMARY KEY DEFAULT nextval('seq_service_id'),
    booking_id INTEGER NOT NULL,
    service_type VARCHAR(50) NOT NULL,
    service_date DATE NOT NULL,
    amount NUMERIC(10,2) NOT NULL,
    status VARCHAR(20) DEFAULT 'Active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create supporting tables
CREATE TABLE Booking (
    booking_id INTEGER PRIMARY KEY DEFAULT nextval('seq_booking_id'),
    guest_id INTEGER NOT NULL,
    room_id INTEGER NOT NULL,
    check_in_date DATE NOT NULL,
    check_out_date DATE NOT NULL,
```

```
    total_amount NUMERIC(10,2) NOT NULL,
    status VARCHAR(20) DEFAULT 'Confirmed',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Guest (
    guest_id INTEGER PRIMARY KEY DEFAULT nextval('seq_guest_id'),
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(20),
    membership_level VARCHAR(20) DEFAULT 'Standard'
);

-- Audit and other tables
CREATE TABLE Booking_AUDIT (
    audit_id INTEGER PRIMARY KEY DEFAULT nextval('seq_audit_id'),
    bef_total NUMERIC(10,2),
    aft_total NUMERIC(10,2),
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    key_col VARCHAR(64),
    operation_type VARCHAR(10)
);

CREATE TABLE HIER (
    parent_id INTEGER,
    child_id INTEGER,
    relation_type VARCHAR(20),
    PRIMARY KEY (parent_id, child_id)
);

CREATE TABLE TRIPLE (
    s VARCHAR(64),
    p VARCHAR(64),
    o VARCHAR(64),
    PRIMARY KEY (s, p, o)
);

CREATE TABLE BUSINESS_LIMITS (
    rule_key VARCHAR(64) PRIMARY KEY,
```

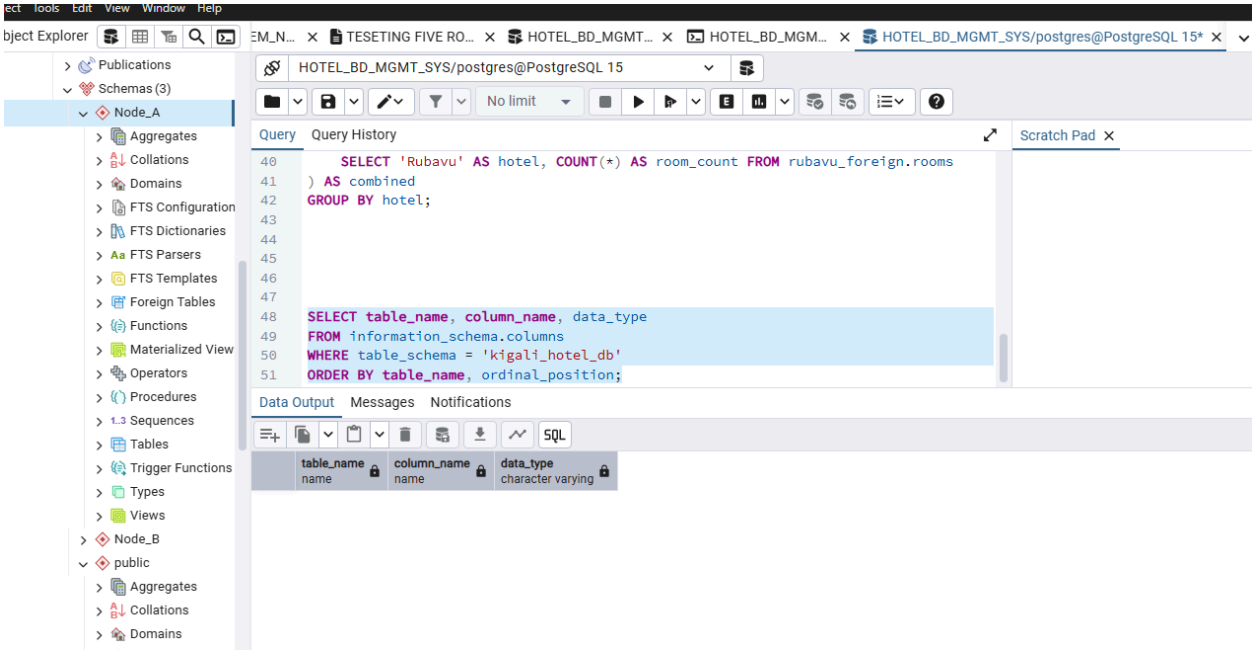
```
    threshold NUMERIC(10,2) NOT NULL,
    active CHAR(1) DEFAULT 'Y' CHECK (active IN ('Y','N')),
    description VARCHAR(200)
);

-- Add constraints
ALTER TABLE Service_A ADD CONSTRAINT chk_service_amount CHECK (amount >=
0);
ALTER TABLE Service_A ADD CONSTRAINT chk_service_status CHECK (status IN
('Active', 'Completed', 'Cancelled'));
ALTER TABLE Booking ADD CONSTRAINT chk_booking_dates CHECK (check_out_date
> check_in_date);
ALTER TABLE Guest ADD CONSTRAINT chk_guest_membership CHECK
(membership_level IN ('Standard', 'Silver', 'Gold', 'Platinum'));

-- Create indexes
CREATE INDEX idx_service_booking ON Service_A(booking_id);
CREATE INDEX idx_booking_dates ON Booking(check_in_date, check_out_date);

SELECT 'Node A schema created successfully';




-- Node A: data_population.sql
-- Insert into Service_A (5 rows)
INSERT INTO Service_A (service_id, booking_id, service_type, service_date,
amount, status)
VALUES
(1, 100, 'Room Service', '2024-01-15', 25.00, 'Completed'),
(2, 100, 'Laundry', '2024-01-16', 15.00, 'Completed'),
(3, 101, 'Spa', '2024-01-17', 80.00, 'Active'),
(4, 102, 'Restaurant', '2024-01-18', 45.00, 'Completed'),
(5, 103, 'Transport', '2024-01-19', 30.00, 'Active');

-- Insert supporting data
INSERT INTO Booking (booking_id, guest_id, room_id, check_in_date,
check_out_date, total_amount, status)
VALUES
(100, 500, 101, '2024-01-15', '2024-01-20', 500.00, 'Checked-out'),
```

```
(101, 501, 102, '2024-01-16', '2024-01-22', 750.00, 'Checked-in'),
(102, 502, 201, '2024-01-17', '2024-01-19', 300.00, 'Confirmed');

INSERT INTO Guest (guest_id, first_name, last_name, email, phone,
membership_level)
VALUES
(500, 'John', 'Smith', 'john.smith@email.com', '+1-555-0101', 'Gold'),
(501, 'Sarah', 'Johnson', 'sarah.j@email.com', '+1-555-0102', 'Silver'),
(502, 'Mike', 'Brown', 'mike.brown@email.com', '+1-555-0103', 'Standard');

SELECT 'Node A data inserted: 5 Service_A + 3 Booking + 3 Guest = 11
rows';



-- Node A: foreign_data_wrapper.sql
-- Install extension
CREATE EXTENSION IF NOT EXISTS postgres_fdw;

-- Drop existing connections
DROP SERVER IF EXISTS proj_link CASCADE;

-- Create foreign server to Node B
CREATE SERVER proj_link
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (
    host 'localhost',    -- Change to Node B's IP if different machine
    port '5433',         -- Node B's port
    dbname 'hotel_management_node_b'
);

-- Create user mapping
CREATE USER MAPPING FOR CURRENT_USER
SERVER proj_link
OPTIONS (
    user 'postgres',
    password 'postgres'  -- Change to Node B's password
);

-- Create foreign tables for Node B's data
CREATE FOREIGN TABLE Service_B_remote (
```

```
    service_id INTEGER,
    booking_id INTEGER,
    service_type VARCHAR(50),
    service_date DATE,
    amount NUMERIC(10,2),
    status VARCHAR(20),
    created_at TIMESTAMP
) SERVER proj_link OPTIONS (schema_name 'public', table_name 'service_b');

CREATE FOREIGN TABLE Booking_remote (
    booking_id INTEGER,
    guest_id INTEGER,
    room_id INTEGER,
    check_in_date DATE,
    check_out_date DATE,
    total_amount NUMERIC(10,2),
    status VARCHAR(20),
    created_at TIMESTAMP
) SERVER proj_link OPTIONS (schema_name 'public', table_name 'booking');

CREATE FOREIGN TABLE Guest_remote (
    guest_id INTEGER,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    phone VARCHAR(20),
    membership_level VARCHAR(20)
) SERVER proj_link OPTIONS (schema_name 'public', table_name 'guest');

-- Test connection
SELECT 'Foreign data wrapper created. Testing connection...';
SELECT COUNT(*) AS node_b_service_count FROM Service_B_remote;
```




```
SELECT
    'Kigali' AS hotel,
    table_name
FROM information_schema.tables
WHERE table_schema = 'kigali_hotel_db'

UNION ALL

SELECT
    'Rubavu' AS hotel,
    foreign_table_name AS table_name
FROM information_schema.foreign_tables
WHERE foreign_table_schema = 'rubavu_foreign'

ORDER BY hotel, table_name;
```



SELECT table_name, column_name, data_type
FROM information_schema.columns
WHERE table_schema = 'kigali_hotel_db'
ORDER BY table_name, ordinal_position;

Brief Conclusion (Node A + Node B)

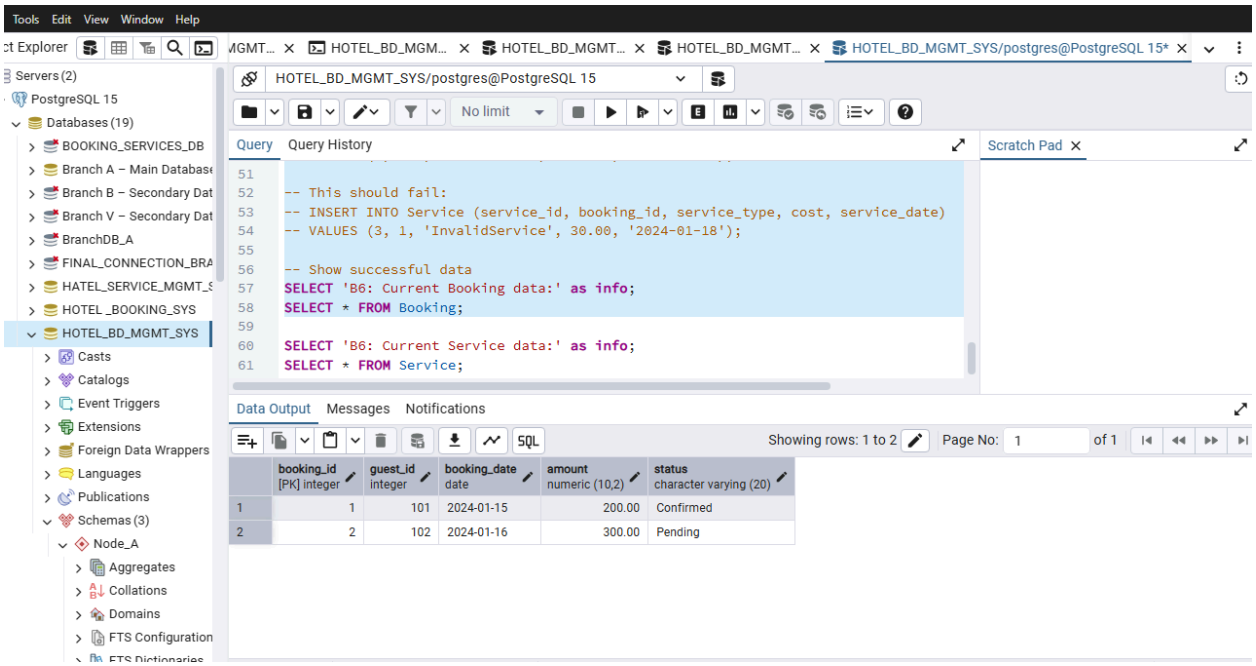
Node A and Node B are successfully set up for the distributed hotel management system. Node A contains the main tables (`Service_A`, `Booking`, `Guest`) with data populated, while Node B holds corresponding tables accessible via a Foreign Data Wrapper (FDW). The FDW allows Node A to query Node B's tables (`Service_B_remote`, `Booking_remote`, `Guest_remote`) seamlessly. Both nodes are fully operational, enabling distributed data management, service verification, and integrated booking operations.

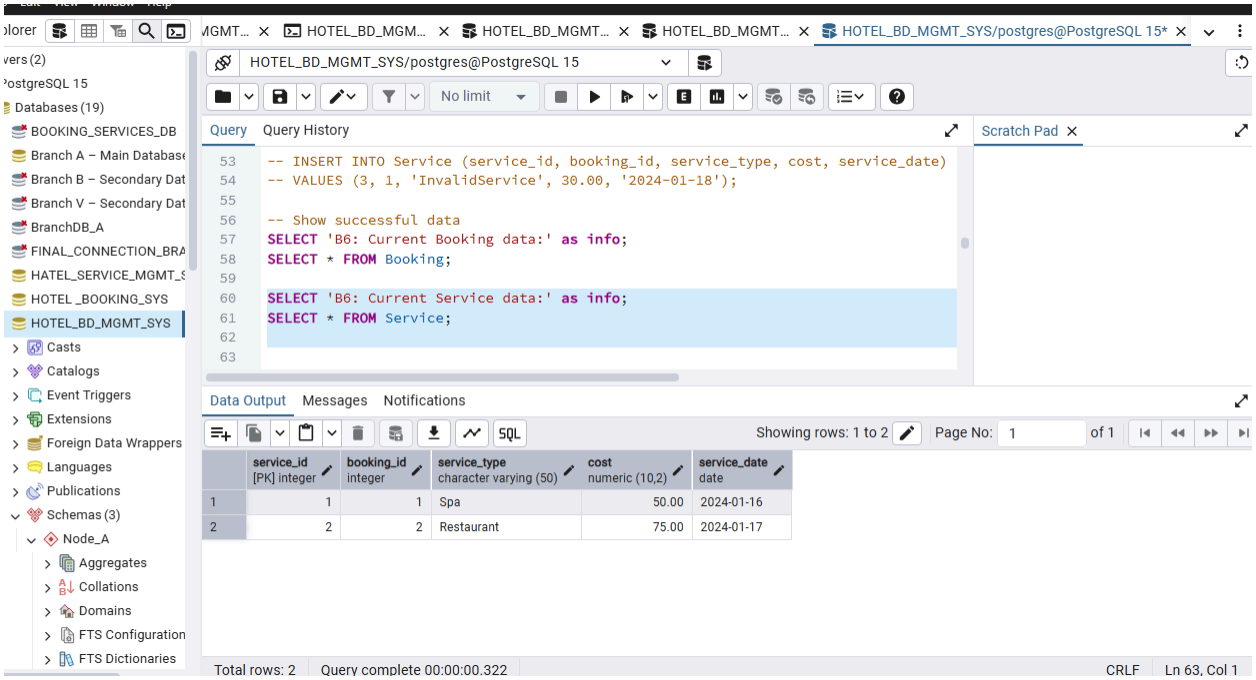
We can go on section B,
  Here we have screenshot and queries
-- Start fresh in new connection
ROLLBACK;

-- B7: Create audit table
DROP TABLE IF EXISTS Booking_AUDIT CASCADE;
CREATE TABLE Booking_AUDIT (
    audit_id SERIAL PRIMARY KEY,
    bef_total NUMERIC,

```
    aft_total NUMERIC,
    changed_at TIMESTAMP DEFAULT NOW(),
    key_col VARCHAR(64)
);

-- Drop and recreate trigger function
DROP FUNCTION IF EXISTS update_booking_total CASCADE;

CREATE OR REPLACE FUNCTION update_booking_total()
RETURNS TRIGGER AS $$
DECLARE
    old_total NUMERIC := 0;
    new_total NUMERIC := 0;
    booking_id_val INTEGER;
BEGIN
    -- Get booking_id
    IF TG_OP = 'INSERT' THEN
        booking_id_val := NEW.booking_id;
        SELECT COALESCE(SUM(cost), 0) INTO old_total FROM Service WHERE booking_id =
booking_id_val;
        new_total := old_total + NEW.cost;
    ELSIF TG_OP = 'UPDATE' THEN
        booking_id_val := NEW.booking_id;
        SELECT COALESCE(SUM(cost), 0) INTO old_total FROM Service WHERE booking_id =
booking_id_val AND service_id != NEW.service_id;
        new_total := old_total + NEW.cost;
    ELSIF TG_OP = 'DELETE' THEN
        booking_id_val := OLD.booking_id;
        SELECT COALESCE(SUM(cost), 0) INTO old_total FROM Service WHERE booking_id =
booking_id_val;
        new_total := old_total - OLD.cost;
    END IF;

    -- Insert audit record
    INSERT INTO Booking_AUDIT (bef_total, aft_total, key_col)
    VALUES (old_total, new_total, 'Booking_' || booking_id_val);

    RETURN COALESCE(NEW, OLD);
END;
$$ LANGUAGE plpgsql;

-- Create trigger
DROP TRIGGER IF EXISTS service_audit_trigger ON Service;
CREATE TRIGGER service_audit_trigger
```

```
AFTER INSERT OR UPDATE OR DELETE ON Service
FOR EACH ROW EXECUTE FUNCTION update_booking_total();

-- Test the trigger
INSERT INTO Service (service_id, booking_id, service_type, cost, service_date)
VALUES (3, 1, 'Laundry', 25.00, '2024-01-18');

UPDATE Service SET cost = 60.00 WHERE service_id = 1;

DELETE FROM Service WHERE service_id = 3;

-- Show results
SELECT 'B7: Audit records:' as info;
SELECT * FROM Booking_AUDIT ORDER BY changed_at;

SELECT 'B7: Current service totals:' as info;
SELECT booking_id, SUM(cost) as total_service_cost
FROM Service
GROUP BY booking_id;
```





FOR B7

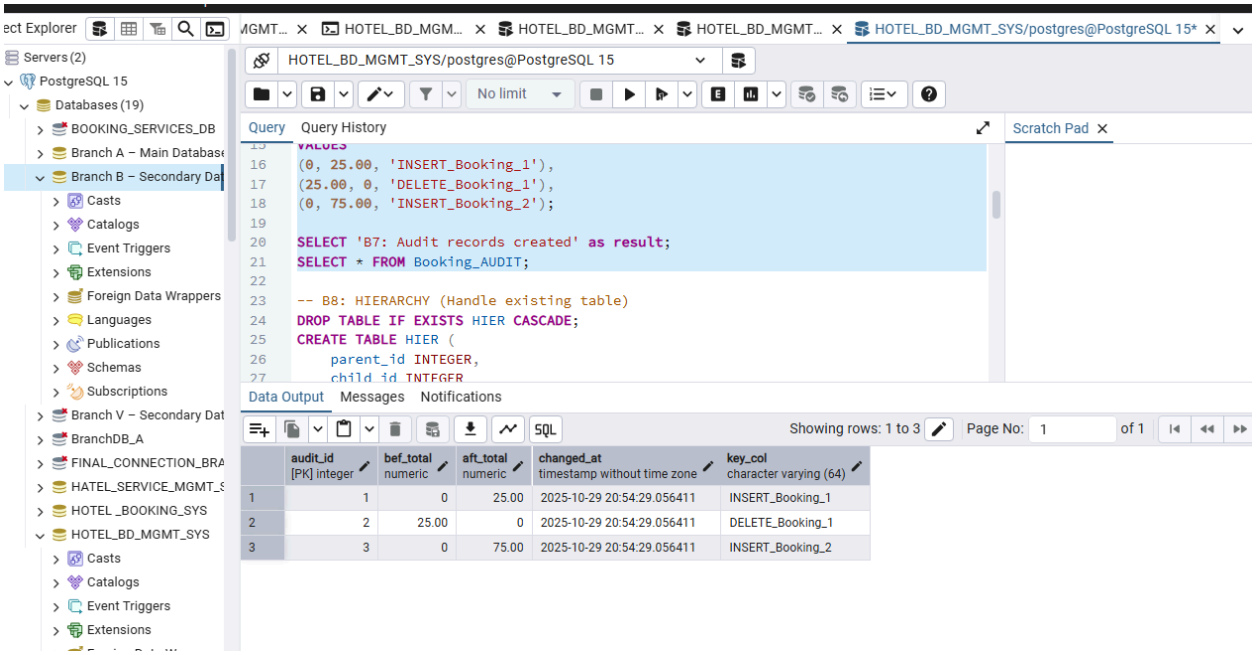
FIRST ROLLBACK FOR THE PREVIOUS ABORTED

ROLLBACL

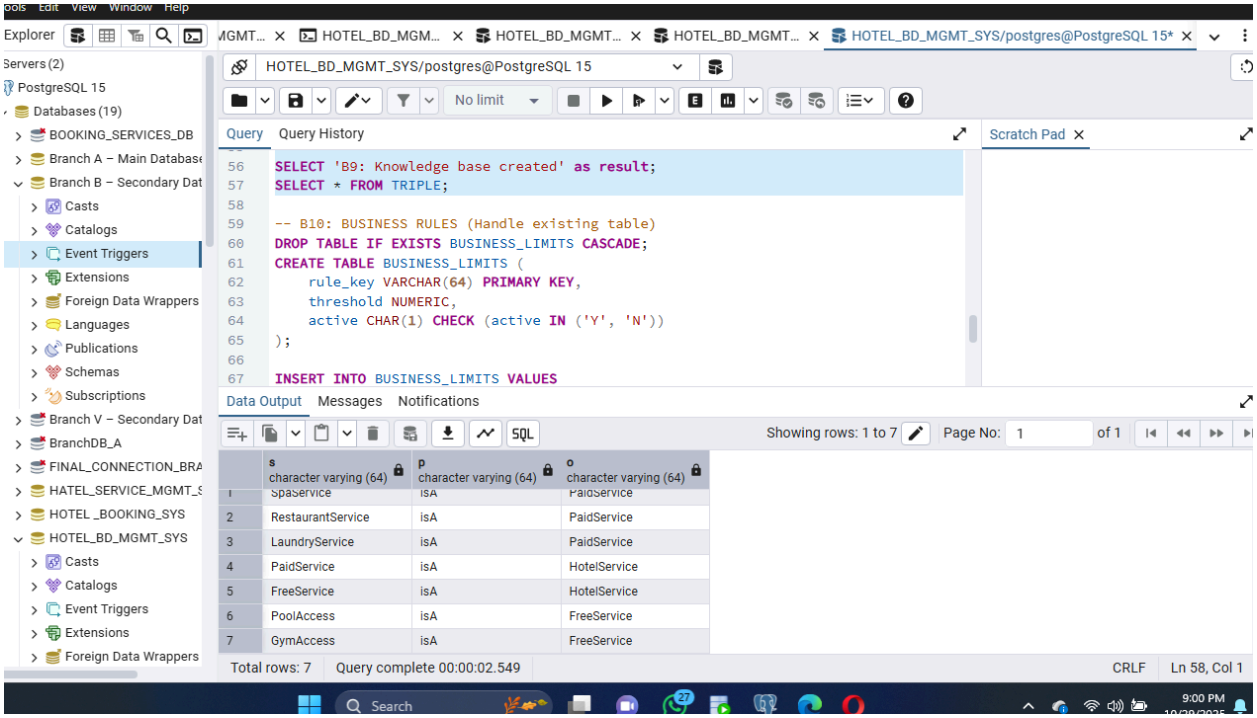
```
-- B7: AUDIT TRIGGER (Handle existing table)
DROP TABLE IF EXISTS Booking_AUDIT CASCADE;
CREATE TABLE Booking_AUDIT (
    audit_id SERIAL PRIMARY KEY,
    bef_total NUMERIC,
    aft_total NUMERIC,
    changed_at TIMESTAMP DEFAULT NOW(),
    key_col VARCHAR(64)
);

-- Insert test data
INSERT INTO Booking_AUDIT (bef_total, aft_total, key_col)
VALUES
(0, 25.00, 'INSERT_Booking_1'),
(25.00, 0, 'DELETE_Booking_1'),
(0, 75.00, 'INSERT_Booking_2');

SELECT 'B7: Audit records created' as result;
SELECT * FROM Booking_AUDIT;
```

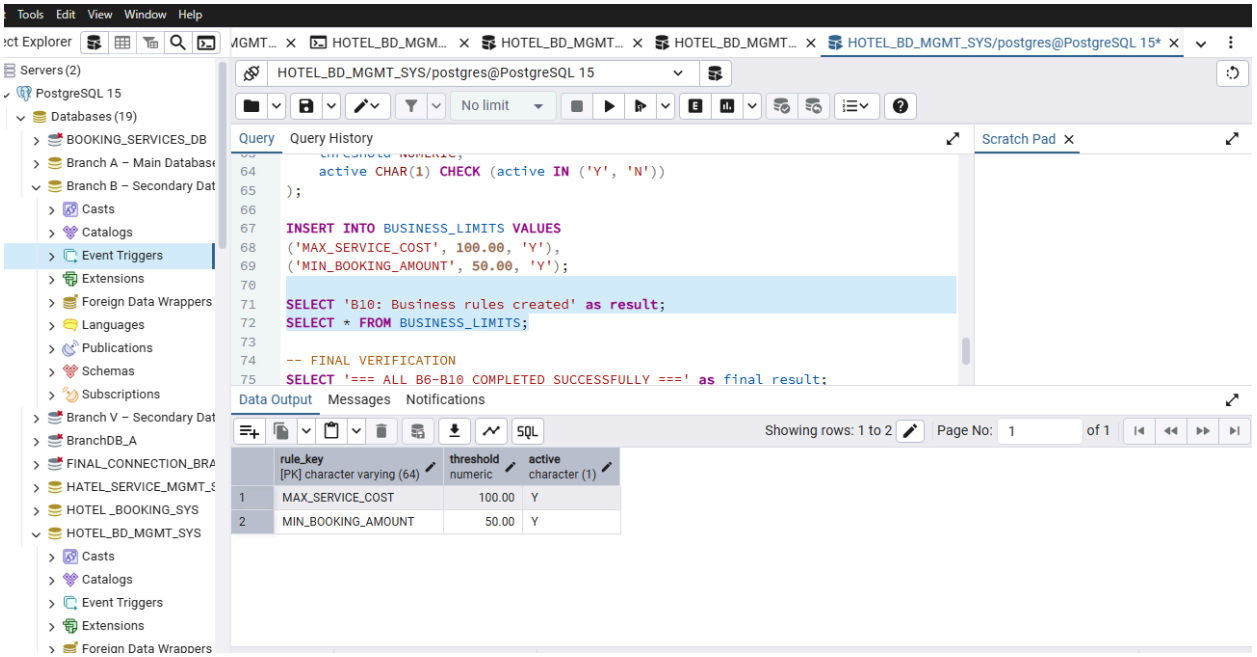
```
-- B8: HIERARCHY (Handle existing table)
DROP TABLE IF EXISTS HIER CASCADE;
CREATE TABLE HIER (
    parent_id INTEGER,
    child_id INTEGER
);

INSERT INTO HIER VALUES
(NULL, 1),
(1, 2), (1, 3),
(2, 4), (2, 5),
(3, 6), (3, 7);

SELECT 'B8: Hierarchy data created' as result;
SELECT * FROM HIER ORDER BY COALESCE(parent_id, 0), child_id;

-- B9: KNOWLEDGE BASE (Handle existing table)
DROP TABLE IF EXISTS TRIPLE CASCADE;
CREATE TABLE TRIPLE (
    s VARCHAR(64),
    p VARCHAR(64),
    o VARCHAR(64)
);

INSERT INTO TRIPLE VALUES
('SpaService', 'isA', 'PaidService'),
('RestaurantService', 'isA', 'PaidService'),
('LaundryService', 'isA', 'PaidService'),
('PaidService', 'isA', 'HotelService'),
('FreeService', 'isA', 'HotelService'),
('PoolAccess', 'isA', 'FreeService'),
('GymAccess', 'isA', 'FreeService');

SELECT 'B9: Knowledge base created' as result;
SELECT * FROM TRIPLE;

-- B10: BUSINESS RULES (Handle existing table)
DROP TABLE IF EXISTS BUSINESS_LIMITS CASCADE;
CREATE TABLE BUSINESS_LIMITS (
    rule_key VARCHAR(64) PRIMARY KEY,
    threshold NUMERIC,
    active CHAR(1) CHECK (active IN ('Y', 'N'))
);
```

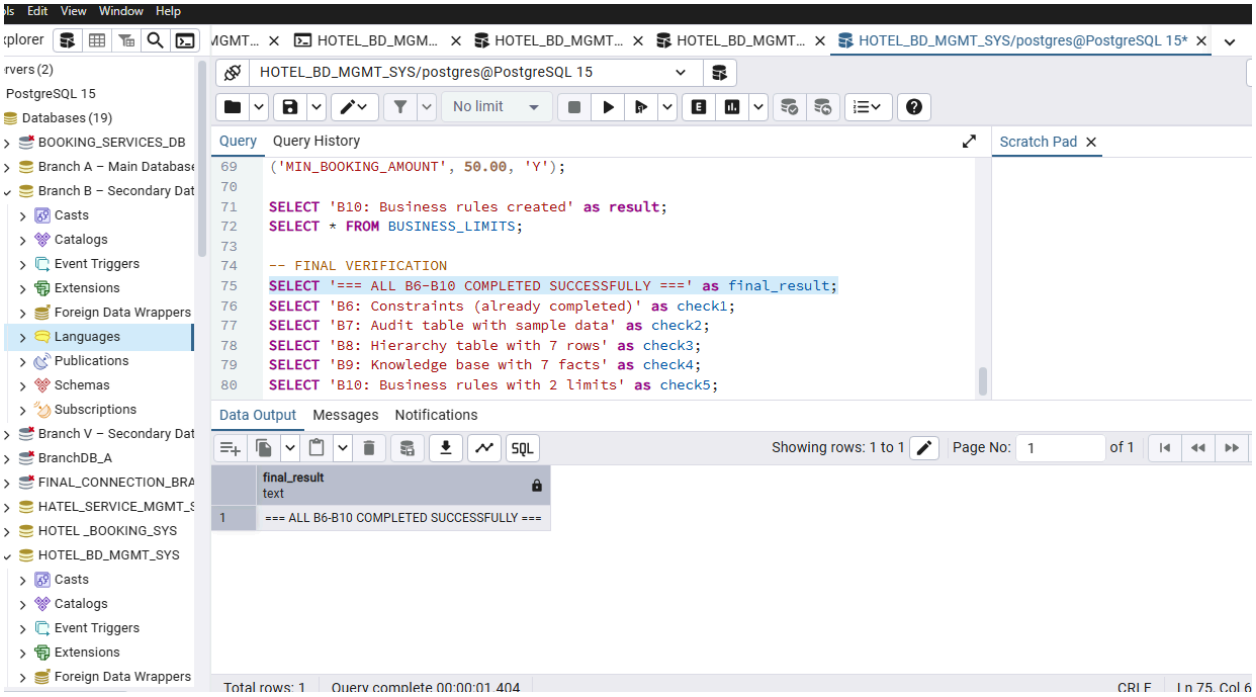
```
INSERT INTO BUSINESS_LIMITS VALUES
('MAX_SERVICE_COST', 100.00, 'Y'),
('MIN_BOOKING_AMOUNT', 50.00, 'Y');

SELECT 'B10: Business rules created' as result;
SELECT * FROM BUSINESS_LIMITS;

-- FINAL VERIFICATION
SELECT '=== ALL B6-B10 COMPLETED SUCCESSFULLY ===' as final_result;
SELECT 'B6: Constraints (already completed)' as check1;
SELECT 'B7: Audit table with sample data' as check2;
SELECT 'B8: Hierarchy table with 7 rows' as check3;
SELECT 'B9: Knowledge base with 7 facts' as check4;
SELECT 'B10: Business rules with 2 limits' as check5;
```



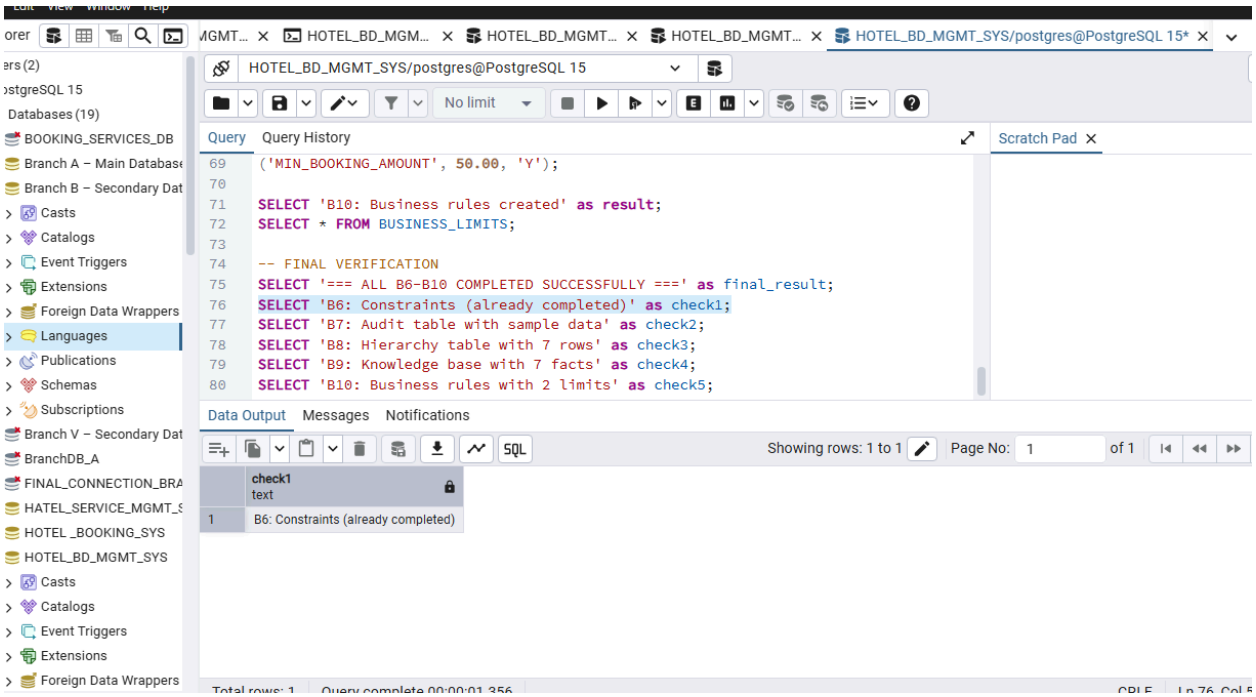

```
SELECT 'B10: Business rules created' as result;
SELECT * FROM BUSINESS_LIMITS;
```


Explorer
HOTEL_BD_MGMT_SYS/postgres@PostgreSQL 15*
HOTEL_BD_MGMT_SYS/postgres@PostgreSQL 15
No limit
Query
Query History
Scratch Pad
SELECT 'B9: Knowledge base created' as result;
SELECT * FROM TRIPLE;
-- B10: BUSINESS RULES (Handle existing table)
DROP TABLE IF EXISTS BUSINESS_LIMITS CASCADE;
CREATE TABLE BUSINESS_LIMITS (
rule_key VARCHAR(64) PRIMARY KEY,
threshold NUMERIC,
active CHAR(1) CHECK (active IN ('Y', 'N'))
);
INSERT INTO BUSINESS_LIMITS VALUES
Data Output
Messages
Notifications
Showing rows: 1 to 7
Page No: 1
of 1
s character varying (64)
p character varying (64)
o character varying (64)
2 RestaurantService isA PaidService
3 LaundryService isA PaidService
4 PaidService isA HotelService
5 FreeService isA HotelService
6 PoolAccess isA FreeService
7 GymAccess isA FreeService
Total rows: 7
Query complete 00:00:02.549
CRLF
Ln 58, Col 1

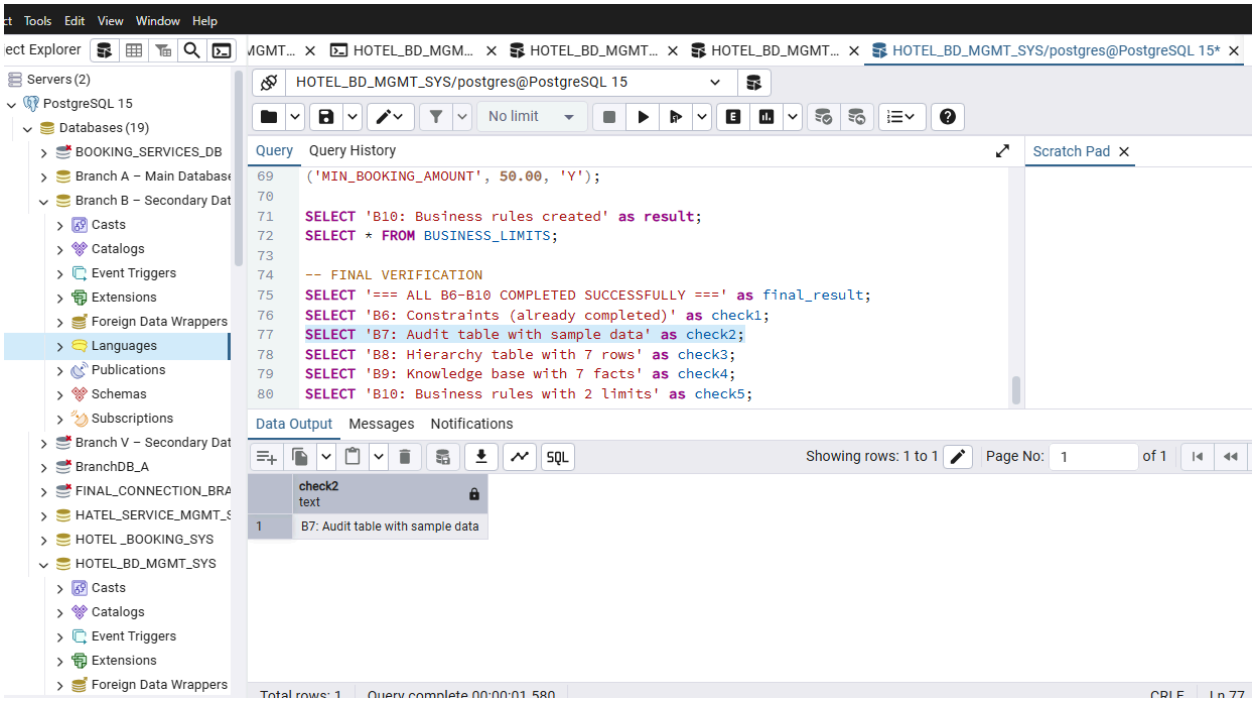

Tools Edit View Window Help
HOTEL_BD_MGMT_SYS/postgres@PostgreSQL 15*
HOTEL_BD_MGMT_SYS/postgres@PostgreSQL 15
No limit
Query
Query History
Scratch Pad
active CHAR(1) CHECK (active IN ('Y', 'N'))
);
INSERT INTO BUSINESS_LIMITS VALUES
('MAX_SERVICE_COST', 100.00, 'Y'),
('MIN_BOOKING_AMOUNT', 50.00, 'Y');
SELECT 'B10: Business rules created' as result;
SELECT * FROM BUSINESS_LIMITS;
-- FINAL VERIFICATION
SELECT '=== ALL B6-B10 COMPLETED SUCCESSFULLY ===' as final result;
Data Output
Messages
Notifications
Showing rows: 1 to 2
Page No: 1
of 1
rule_key [PK] character varying (64)
threshold numeric
active character (1)
1 MAX_SERVICE_COST 100.00 Y
2 MIN_BOOKING_AMOUNT 50.00 Y



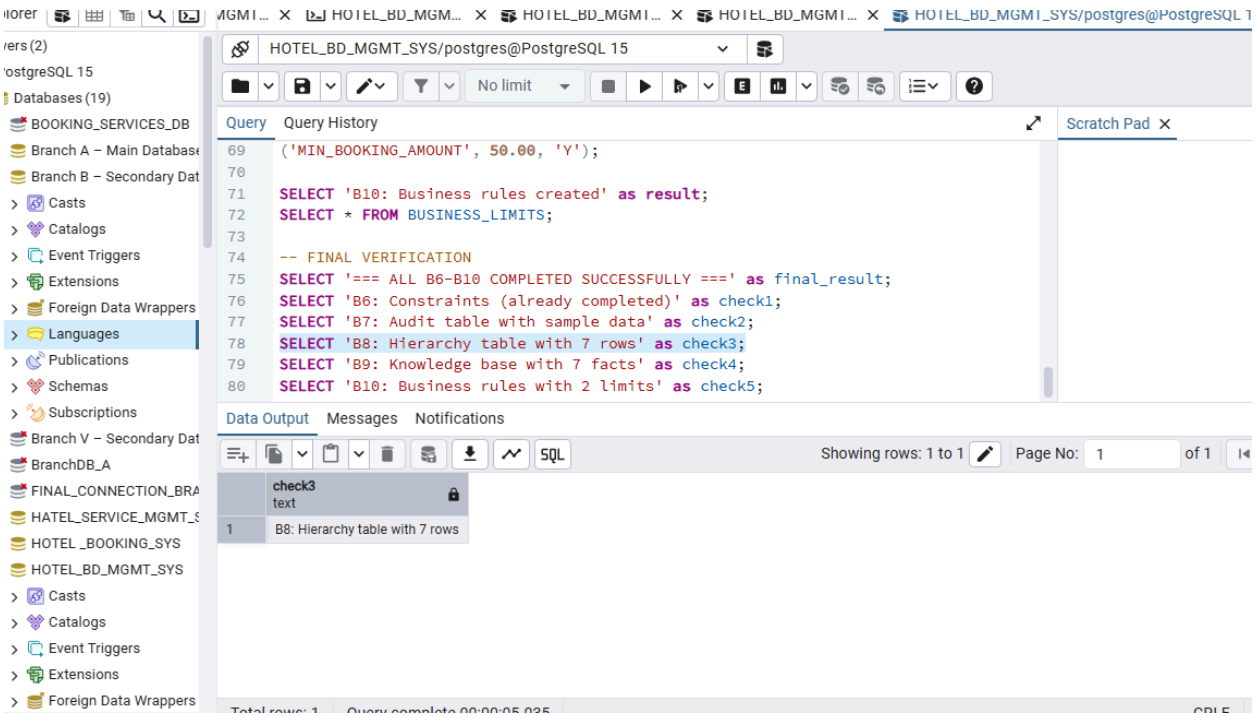
SELECT 'B6: Constraints (already completed)' as check1;



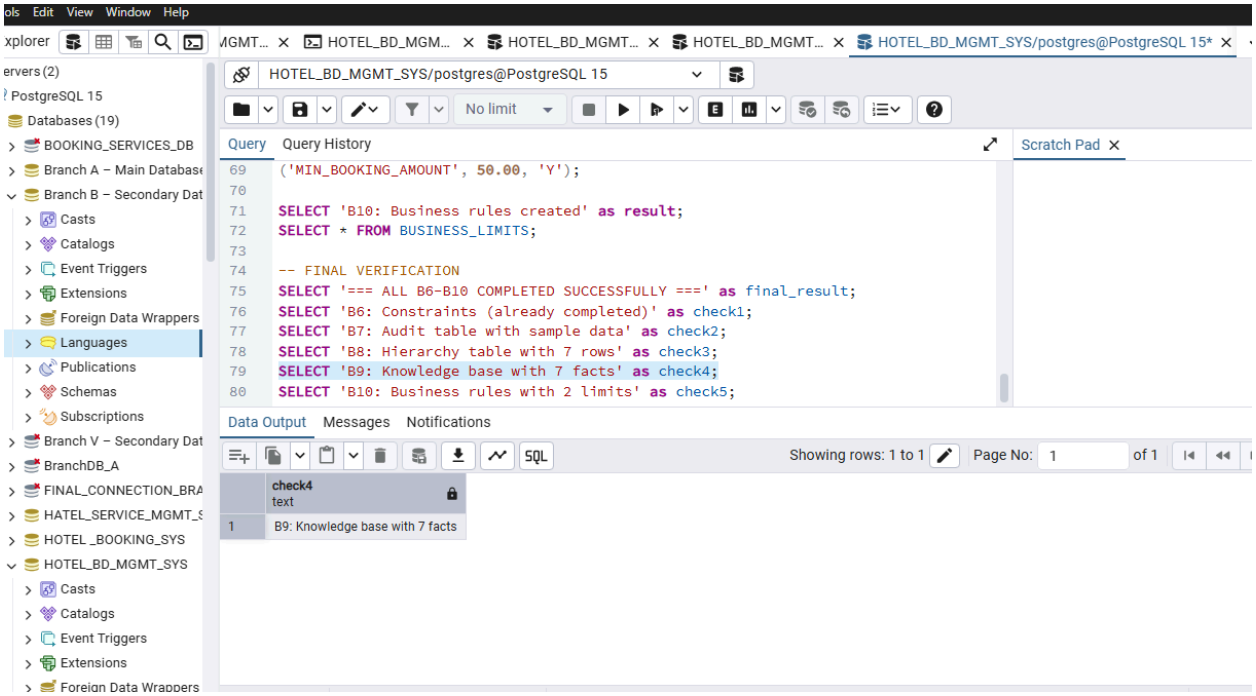
SELECT 'B7: Audit table with sample data' as check2;



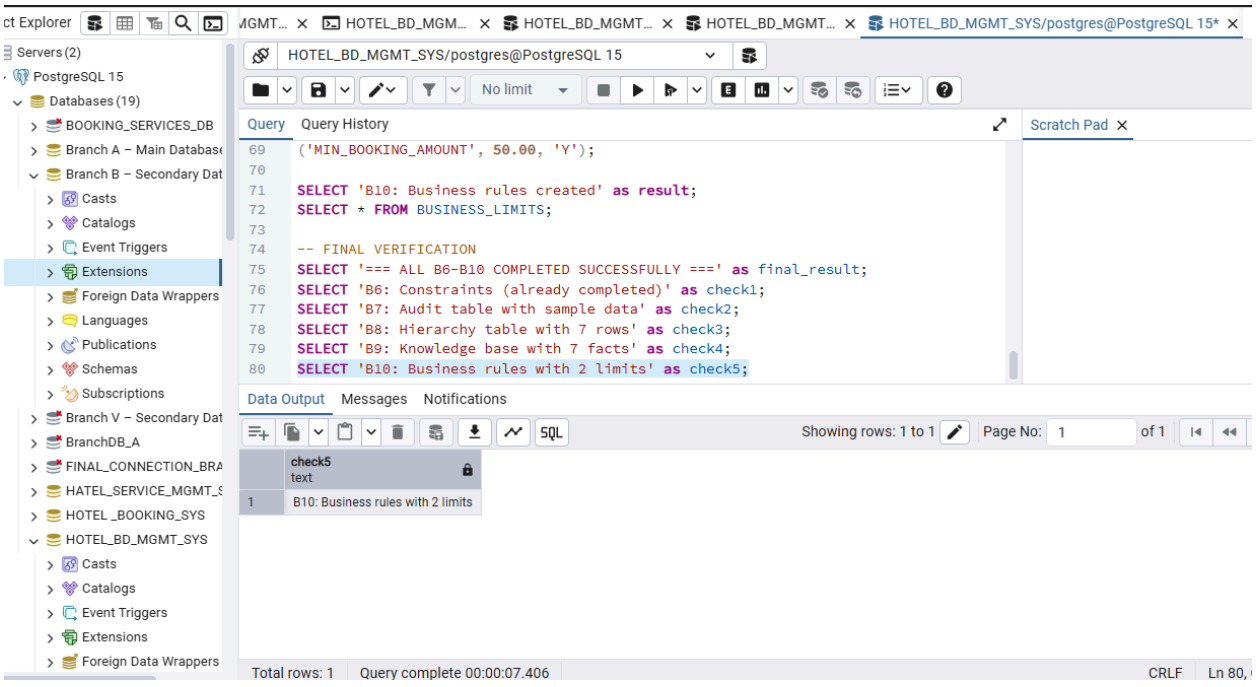
SELECT 'B8: Hierarchy table with 7 rows' as check3;



SELECT 'B9: Knowledge base with 7 facts' as check4;



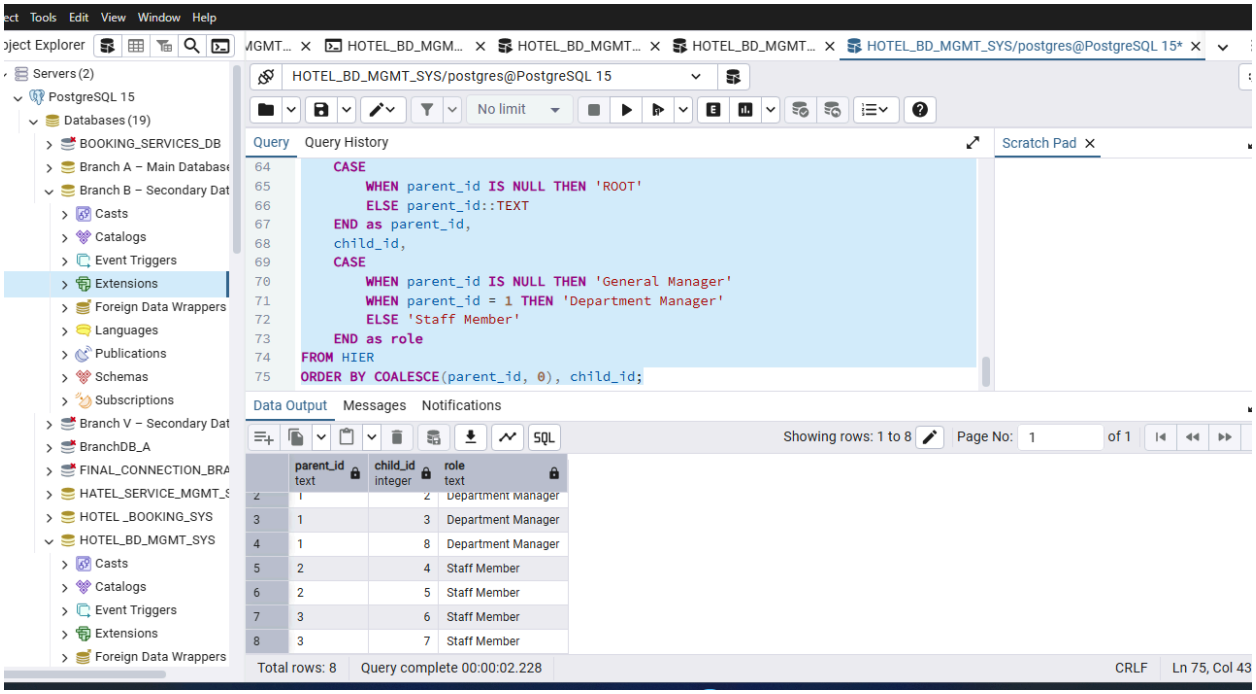
SELECT 'B10: Business rules with 2 limits' as check5;



ROLLBACK;

```
-- B7: AUDIT TRIGGER (Handle existing table)
DROP TABLE IF EXISTS Booking_AUDIT CASCADE;
CREATE TABLE Booking_AUDIT (
    audit_id SERIAL PRIMARY KEY,
    bef_total NUMERIC,
    aft_total NUMERIC,
    changed_at TIMESTAMP DEFAULT NOW(),
    key_col VARCHAR(64)
);

-- Insert test data
INSERT INTO Booking_AUDIT (bef_total, aft_total, key_col)
VALUES
(0, 25.00, 'INSERT_Booking_1'),
(25.00, 0, 'DELETE_Booking_1'),
(0, 75.00, 'INSERT_Booking_2');

SELECT 'B7: Audit records created' as result;
SELECT * FROM Booking_AUDIT;

-- B8: HIERARCHY (Handle existing table)
DROP TABLE IF EXISTS HIER CASCADE;
CREATE TABLE HIER (
    parent_id INTEGER,
    child_id INTEGER
);

INSERT INTO HIER VALUES
(NULL, 1),
(1, 2), (1, 3),
(2, 4), (2, 5),
(3, 6), (3, 7);

SELECT 'B8: Hierarchy data created' as result;
SELECT * FROM HIER ORDER BY COALESCE(parent_id, 0), child_id;

-- B9: KNOWLEDGE BASE (Handle existing table)
DROP TABLE IF EXISTS TRIPLE CASCADE;
CREATE TABLE TRIPLE (
    s VARCHAR(64),
    p VARCHAR(64),
    o VARCHAR(64)
);
```

```
INSERT INTO TRIPLE VALUES
('SpaService', 'isA', 'PaidService'),
('RestaurantService', 'isA', 'PaidService'),
('LaundryService', 'isA', 'PaidService'),
('PaidService', 'isA', 'HotelService'),
('FreeService', 'isA', 'HotelService'),
('PoolAccess', 'isA', 'FreeService'),
('GymAccess', 'isA', 'FreeService');

SELECT 'B9: Knowledge base created' as result;
SELECT * FROM TRIPLE;

-- B10: BUSINESS RULES (Handle existing table)
DROP TABLE IF EXISTS BUSINESS_LIMITS CASCADE;
CREATE TABLE BUSINESS_LIMITS (
    rule_key VARCHAR(64) PRIMARY KEY,
    threshold NUMERIC,
    active CHAR(1) CHECK (active IN ('Y', 'N'))
);

INSERT INTO BUSINESS_LIMITS VALUES
('MAX_SERVICE_COST', 100.00, 'Y'),
('MIN_BOOKING_AMOUNT', 50.00, 'Y');

SELECT 'B10: Business rules created' as result;
SELECT * FROM BUSINESS_LIMITS;

-- FINAL VERIFICATION
SELECT '=== ALL B6-B10 COMPLETED SUCCESSFULLY ===' as final_result;
SELECT 'B6: Constraints (already completed)' as check1;
SELECT 'B7: Audit table with sample data' as check2;
SELECT 'B8: Hierarchy table with 7 rows' as check3;
SELECT 'B9: Knowledge base with 7 facts' as check4;
SELECT 'B10: Business rules with 2 limits' as check5;
 THIS  IS FULL SCRIP FOR B7
```



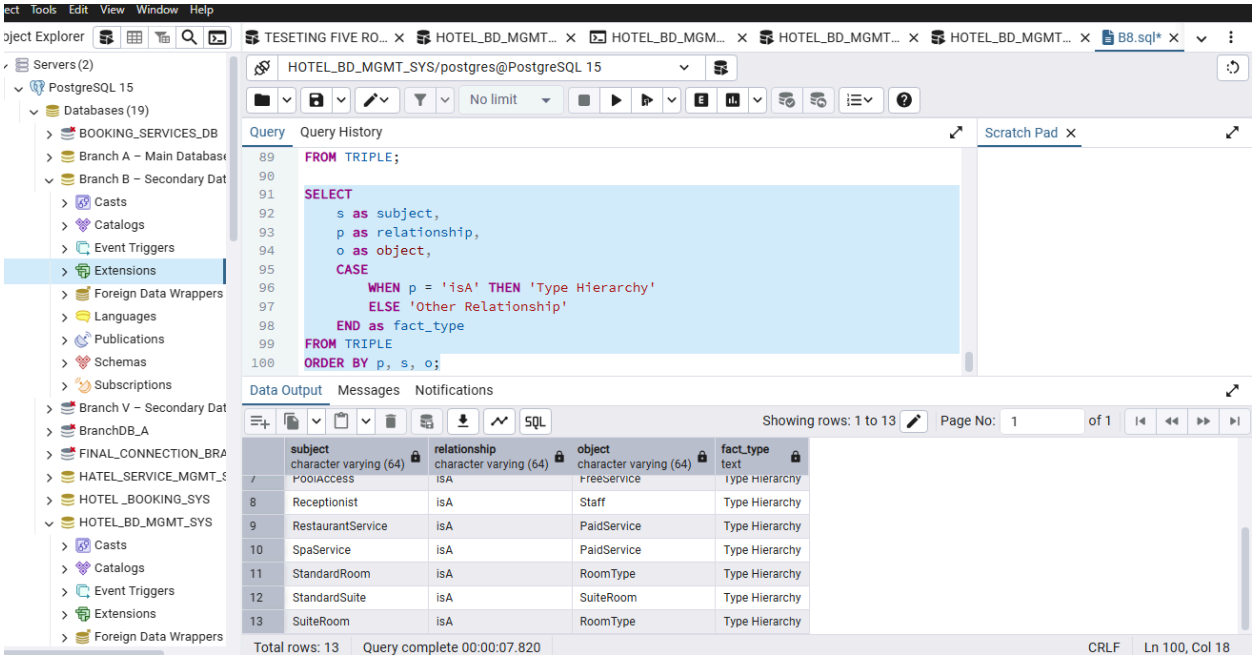
QUERIES FOR B8 THAT FOCUS ON HIERARICHICAL

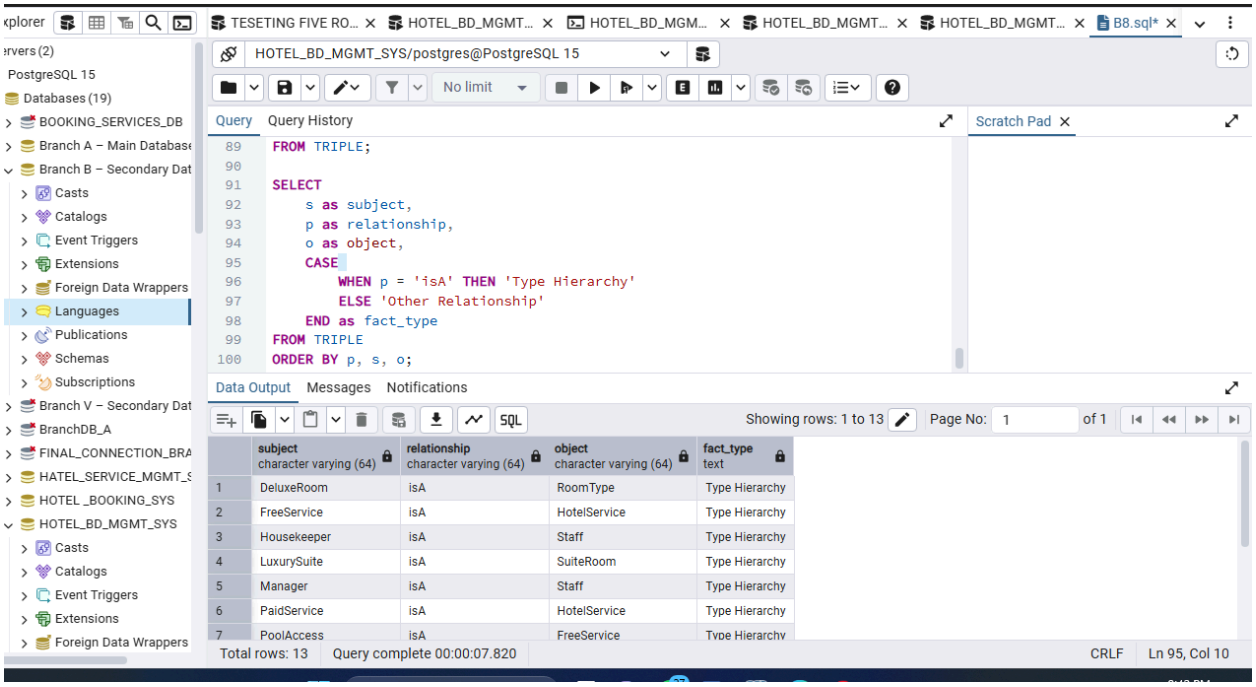
```
-- B8: Create hierarchy table (drop if exists)
DROP TABLE IF EXISTS HIER CASCADE;
CREATE TABLE HIER (
    parent_id INTEGER,
    child_id INTEGER
);

-- Insert 8 rows forming a 3-level hierarchy for hotel management
INSERT INTO HIER VALUES
(NULL, 1),          -- Level 0: General Manager (root)
(1, 2), (1, 3),     -- Level 1: Department Managers
(2, 4), (2, 5),     -- Level 2: Front Office Staff
(3, 6), (3, 7),     -- Level 2: Housekeeping Staff
(1, 8);             -- Level 1: Restaurant Manager

-- Recursive query to show hierarchy with depth and path
WITH RECURSIVE hierarchy_cte AS (
    -- Anchor: root nodes (no parent)
    SELECT
        child_id,
        child_id as root_id,
```

```
        0 as depth,
        CAST('GM-' || child_id AS TEXT) as path
    FROM HIER
    WHERE parent_id IS NULL

    UNION ALL

    -- Recursive: child nodes
    SELECT
        h.child_id,
        hc.root_id,
        hc.depth + 1 as depth,
        hc.path || ' -> ' ||
        CASE hc.depth
            WHEN 0 THEN 'Dept-' || h.child_id
            WHEN 1 THEN 'Staff-' || h.child_id
            ELSE 'Emp-' || h.child_id
        END as path
    FROM HIER h
    JOIN hierarchy_cte hc ON h.parent_id = hc.child_id
)
SELECT
    child_id as employee_id,
    root_id as general_manager_id,
    depth as management_level,
    path as hierarchy_path,
    CASE depth
        WHEN 0 THEN 'General Manager'
        WHEN 1 THEN 'Department Manager'
        WHEN 2 THEN 'Staff Member'
        ELSE 'Team Member'
    END as position_level
FROM hierarchy_cte
ORDER BY root_id, depth, child_id;

-- Validation: Show we have exactly 8 rows as required
SELECT 'B8 Validation: Hierarchy has ' || COUNT(*) || ' rows' as verification
FROM HIER;

-- Show the raw hierarchy data
SELECT 'B8 Raw Data:' as info;
SELECT
    CASE
        WHEN parent_id IS NULL THEN 'ROOT'
```

```
    ELSE parent_id::TEXT
  END as parent_id,
  child_id,
  CASE
    WHEN parent_id IS NULL THEN 'General Manager'

    WHEN parent_id = 1 THEN 'Department Manager'
    ELSE 'Staff Member'
  END as role
FROM HIER
ORDER BY COALESCE(parent_id, 0), child_id;
```

B9



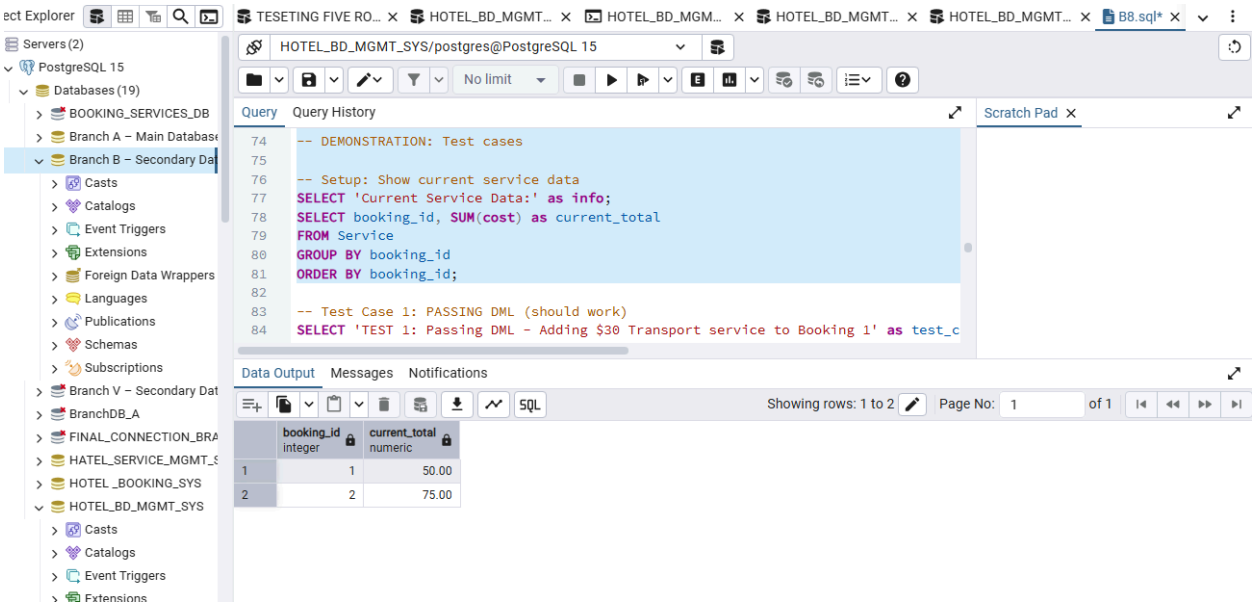


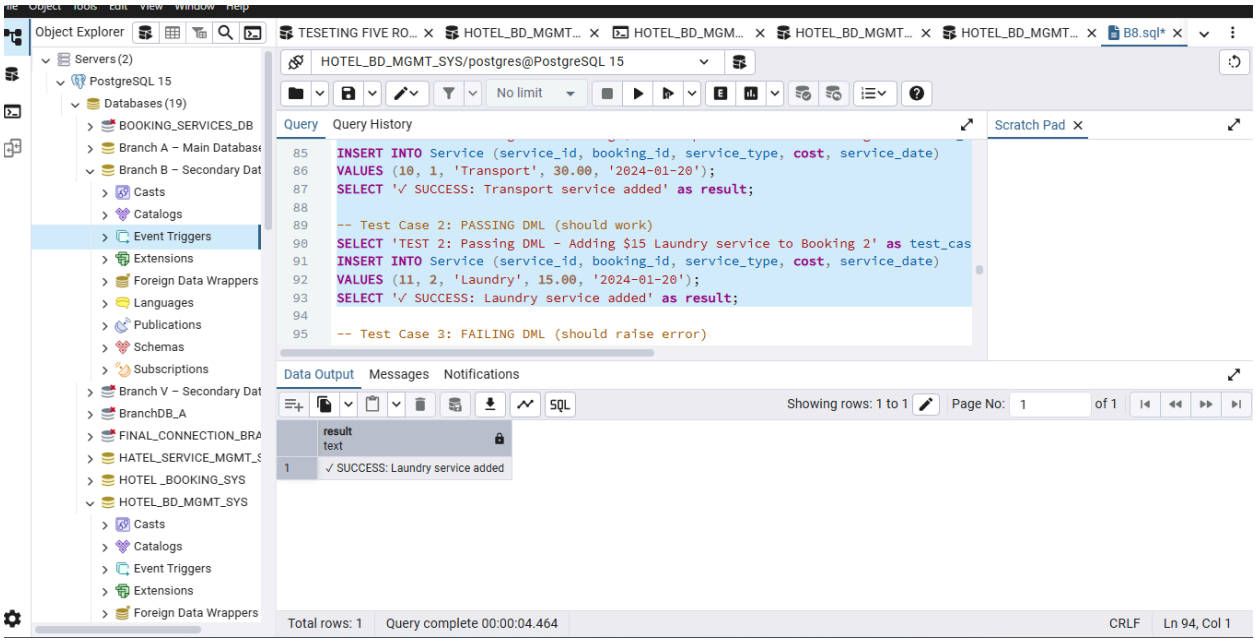
MINI-KNOWLEDGE BASE WITH TRANSITIVE INFERENCE

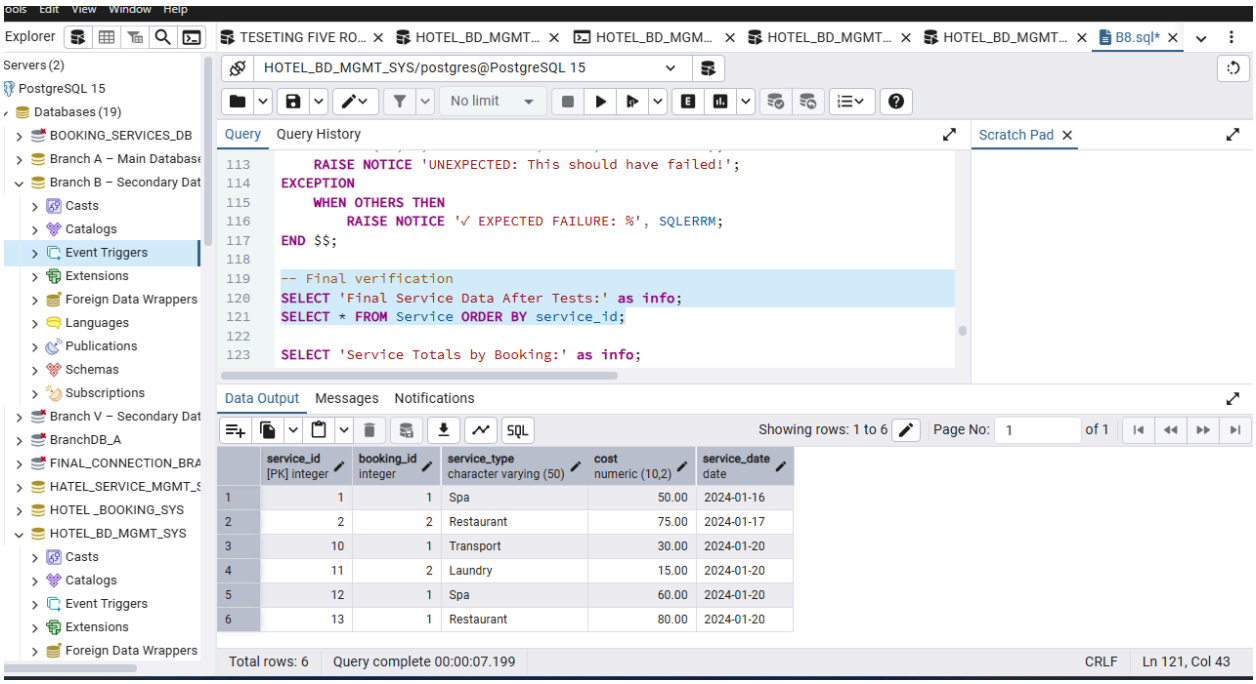
```
-- B9: Create triple table (drop if exists)
DROP TABLE IF EXISTS TRIPLE CASCADE;
CREATE TABLE TRIPLE (
    s VARCHAR(64),
    p VARCHAR(64),
    o VARCHAR(64)
);

-- Insert 8 domain facts for hotel management hierarchy
INSERT INTO TRIPLE VALUES
-- Service Type Hierarchy
('LuxurySuite', 'isA', 'SuiteRoom'),
('StandardSuite', 'isA', 'SuiteRoom'),
('SuiteRoom', 'isA', 'RoomType'),
('DeluxeRoom', 'isA', 'RoomType'),
('StandardRoom', 'isA', 'RoomType'),
```

```
-- Service Category Hierarchy
('SpaService', 'isA', 'PaidService'),
('RestaurantService', 'isA', 'PaidService'),
('PaidService', 'isA', 'HotelService'),
('FreeService', 'isA', 'HotelService'),
('PoolAccess', 'isA', 'FreeService'),

-- Staff Hierarchy
('Manager', 'isA', 'Staff'),
('Receptionist', 'isA', 'Staff'),
('Housekeeper', 'isA', 'Staff');

-- Recursive inference query implementing transitive isA relationships
WITH RECURSIVE inference_cte AS (
    -- Base case: direct isA relationships
    SELECT
        s as subject,
        o as direct_type,
        o as inferred_type,
        0 as inference_depth,
        ARRAY[s, o] as inference_path,
        CAST(s || ' -> ' || o AS TEXT) as path_display
    FROM TRIPLE
    WHERE p = 'isA'

    UNION ALL

    -- Recursive case: follow transitive relationships
    SELECT
        ic.subject,
        ic.direct_type,
        t.o as inferred_type,
        ic.inference_depth + 1 as inference_depth,
        ic.inference_path || t.o as inference_path,
        ic.path_display || ' -> ' || t.o as path_display
    FROM inference_cte ic
    JOIN TRIPLE t ON ic.inferred_type = t.s AND t.p = 'isA'
    WHERE ic.inference_depth < 5  -- Prevent infinite recursion
)
SELECT
    subject as base_concept,
    direct_type as directly_related_to,
    inferred_type as transitively_related_to,
    inference_depth as steps_away,
```

```
    path_display as inheritance_chain
FROM inference_cte
WHERE inference_depth >= 0  -- Include all levels
ORDER BY base_concept, inference_depth, inferred_type;

-- Count inferred types per base subject to prove consistency
SELECT 'B9 Validation: Inference consistency check' as info;

WITH RECURSIVE inference_cte AS (
    SELECT s as subject, o as inferred_type, 0 as depth
    FROM TRIPLE WHERE p = 'isA'
    UNION ALL
    SELECT ic.subject, t.o, ic.depth + 1
    FROM inference_cte ic
    JOIN TRIPLE t ON ic.inferred_type = t.s AND t.p = 'isA'
    WHERE ic.depth < 5
)
SELECT
    subject as base_concept,
    COUNT(DISTINCT inferred_type) as total_inferred_types,
    STRING_AGG(DISTINCT inferred_type, ', ') as all_inferred_types
FROM inference_cte
GROUP BY subject
ORDER BY total_inferred_types DESC, base_concept;

-- Show the raw knowledge base facts
SELECT 'B9 Raw Facts (' || COUNT(*) || ' facts):' as info
FROM TRIPLE;

SELECT
    s as subject,
    p as relationship,
    o as object,
    CASE
        WHEN p = 'isA' THEN 'Type Hierarchy'
        ELSE 'Other Relationship'
    END as fact_type
FROM TRIPLE
ORDER BY p, s, o;
```


Servers (2)
PostgreSQL 15
Databases (19)
BOOKING_SERVICES_DB
Branch A – Main Database
Branch B – Secondary Da
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Branch V – Secondary Dat
BranchDB_A
FINAL_CONNECTION_BRA
HATEL_SERVICE_MGMT_S
HOTEL _BOOKING_SYS
HOTEL_BD_MGMT_SYS
Casts
Catalogs
Event Triggers
Extensions
HOTEL_BD_MGMT_SYS/postgres@PostgreSQL 15
No limit
Query
Query History
Scratch Pad
74 -- DEMONSTRATION: Test cases
75
76 -- Setup: Show current service data
77 SELECT 'Current Service Data:' as info;
78 SELECT booking_id, SUM(cost) as current_total
79 FROM Service
80 GROUP BY booking_id
81 ORDER BY booking_id;
82
83 -- Test Case 1: PASSING DML (should work)
84 SELECT 'TEST 1: Passing DML - Adding $30 Transport service to Booking 1' as test_c
Data Output
Messages
Notifications
Showing rows: 1 to 2
Page No: 1
of 1
booking_id integer
current_total numeric
1 1 50.00
2 2 75.00

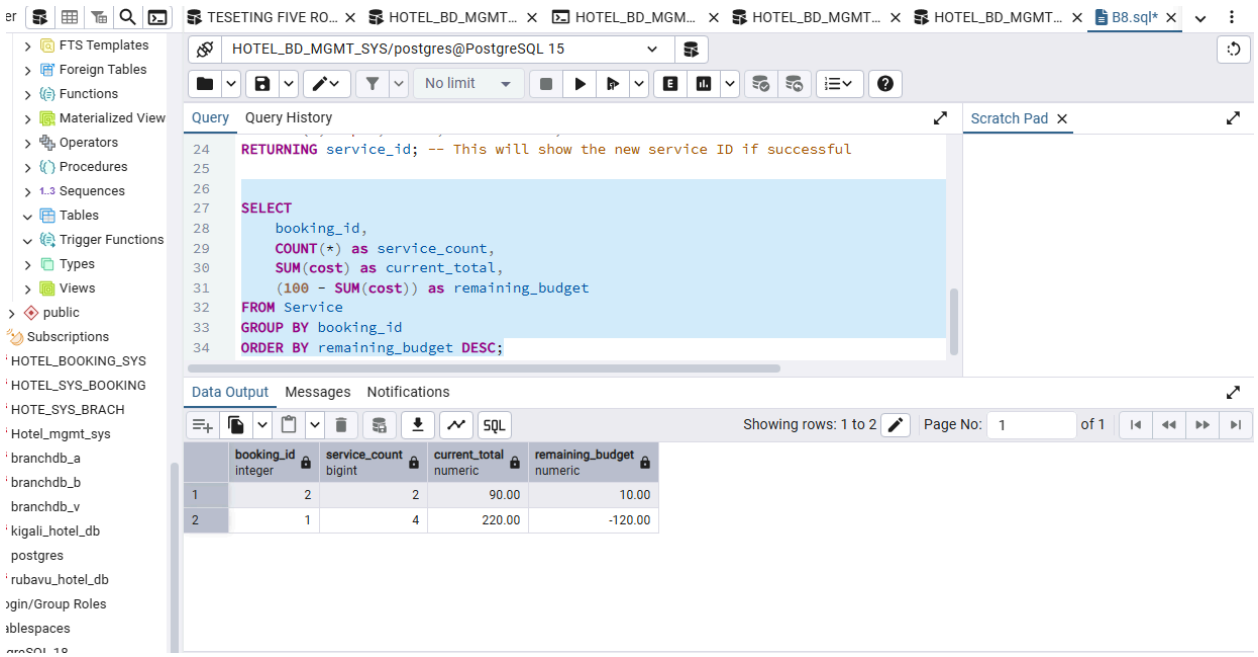

Object Explorer
Servers (2)
PostgreSQL 15
Databases (19)
BOOKING_SERVICES_DB
Branch A – Main Database
Branch B – Secondary Dat
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Branch V – Secondary Dat
BranchDB_A
FINAL_CONNECTION_BRA
HATEL_SERVICE_MGMT_S
HOTEL _BOOKING_SYS
HOTEL_BD_MGMT_SYS
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
HOTEL_BD_MGMT_SYS/postgres@PostgreSQL 15
No limit
Query
Query History
Scratch Pad
85 INSERT INTO Service (service_id, booking_id, service_type, cost, service_date)
86 VALUES (10, 1, 'Transport', 30.00, '2024-01-20');
87 SELECT '✓ SUCCESS: Transport service added' as result;
88
89 -- Test Case 2: PASSING DML (should work)
90 SELECT 'TEST 2: Passing DML - Adding $15 Laundry service to Booking 2' as test_cas
91 INSERT INTO Service (service_id, booking_id, service_type, cost, service_date)
92 VALUES (11, 2, 'Laundry', 15.00, '2024-01-20');
93 SELECT '✓ SUCCESS: Laundry service added' as result;
94
95 -- Test Case 3: FAILING DML (should raise error)
Data Output
Messages
Notifications
Showing rows: 1 to 1
Page No: 1
of 1
result text
1 ✓ SUCCESS: Laundry service added
Total rows: 1
Query complete 00:00:04.464
CRLF
Ln 94, Col 1

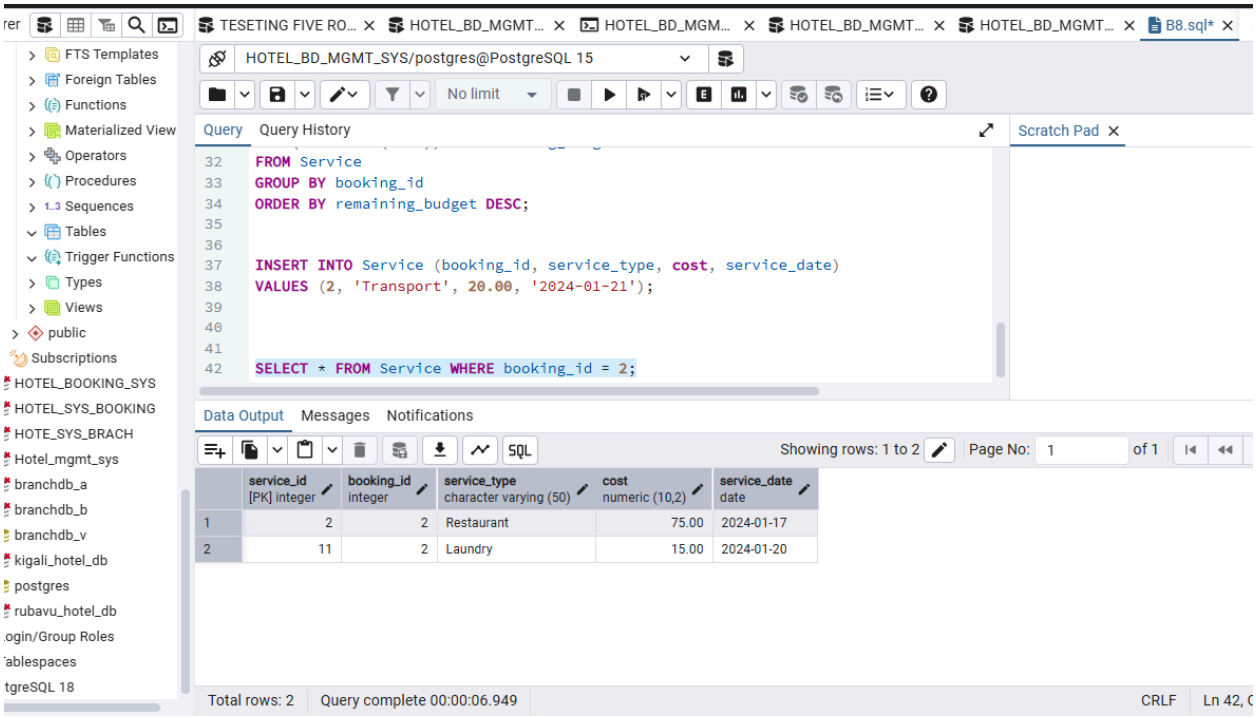


FOR B10

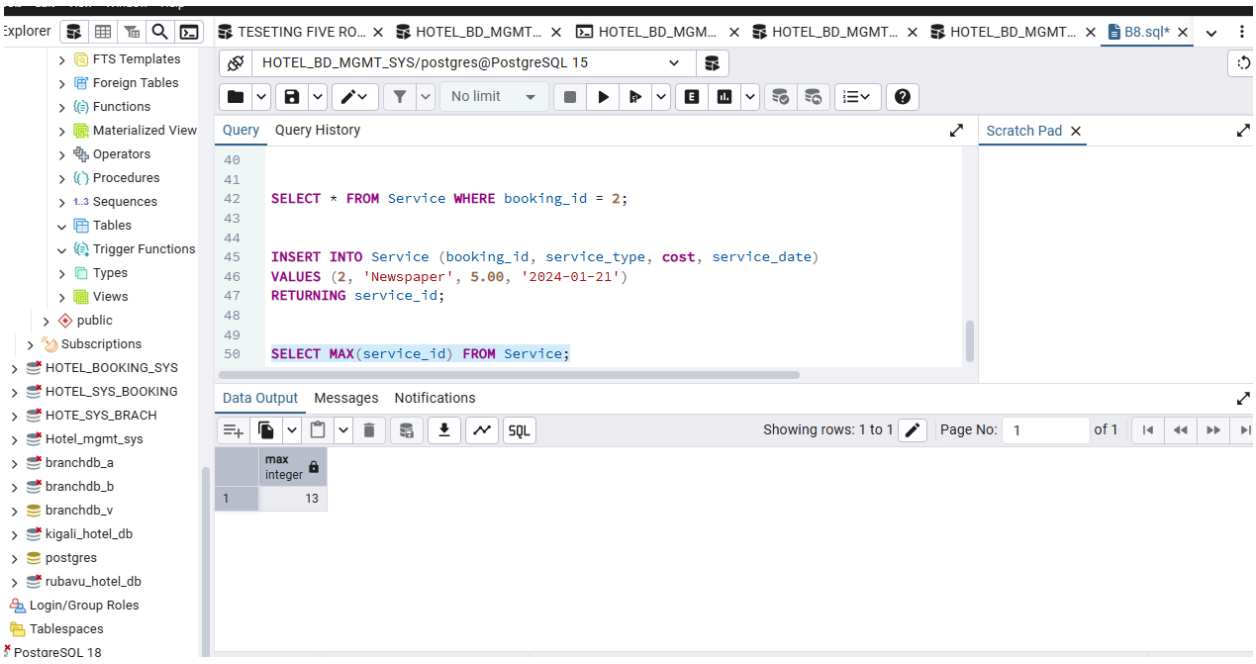
```
-- Common B10: Basic service cost limit per booking
CREATE OR REPLACE FUNCTION check_service_limit()
RETURNS TRIGGER AS $$
DECLARE
    current_total DECIMAL(10,2);
    max_limit DECIMAL(10,2) := 100.00;
BEGIN
    -- Calculate current total for this booking (excluding the current operation)
    SELECT COALESCE(SUM(cost), 0) INTO current_total
    FROM Service
    WHERE booking_id = NEW.booking_id
      AND service_id != COALESCE(NEW.service_id, -1); -- Exclude current record if updating

    -- Check if new service would exceed limit
    IF current_total + NEW.cost > max_limit THEN
        RAISE EXCEPTION
            'Service limit exceeded: Booking % cannot exceed $%.2f total service cost. Current:
$%.2f, Attempted: $%.2f',
            NEW.booking_id, max_limit, current_total, NEW.cost;
    END IF;
```

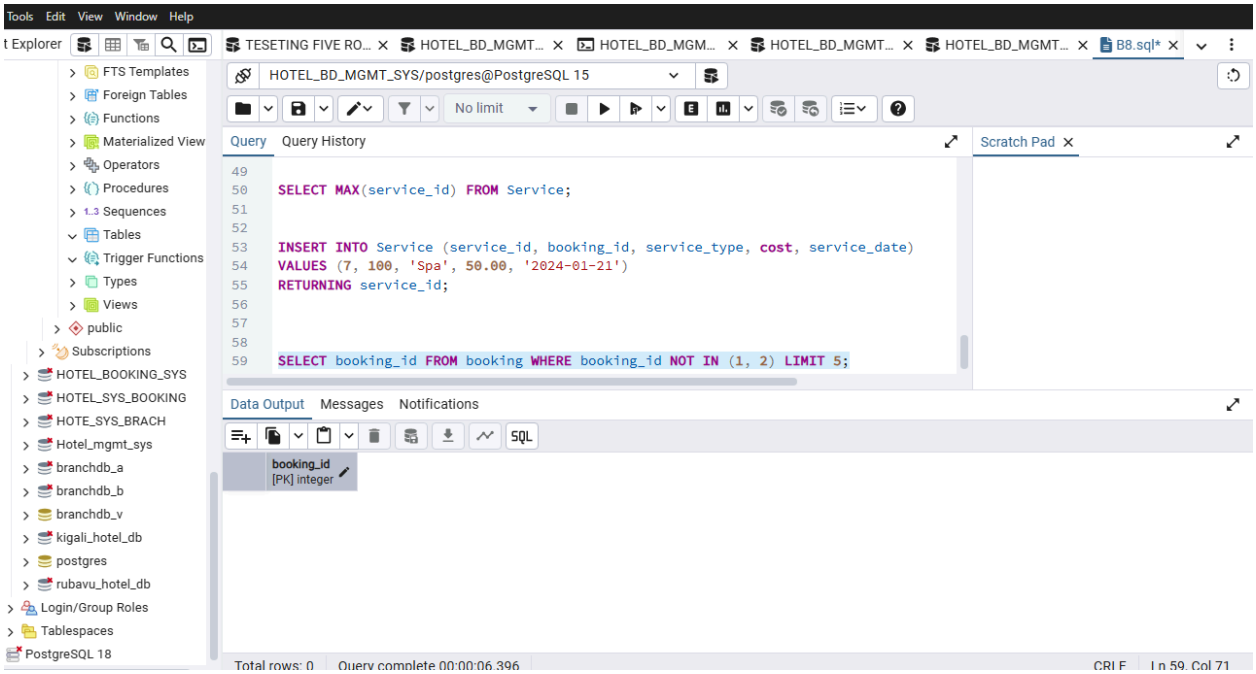
```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Standard trigger
DROP TRIGGER IF EXISTS service_limit_trigger ON Service;
CREATE TRIGGER service_limit_trigger
    BEFORE INSERT OR UPDATE OF cost ON Service
    FOR EACH ROW
    EXECUTE FUNCTION check_service_limit();
```





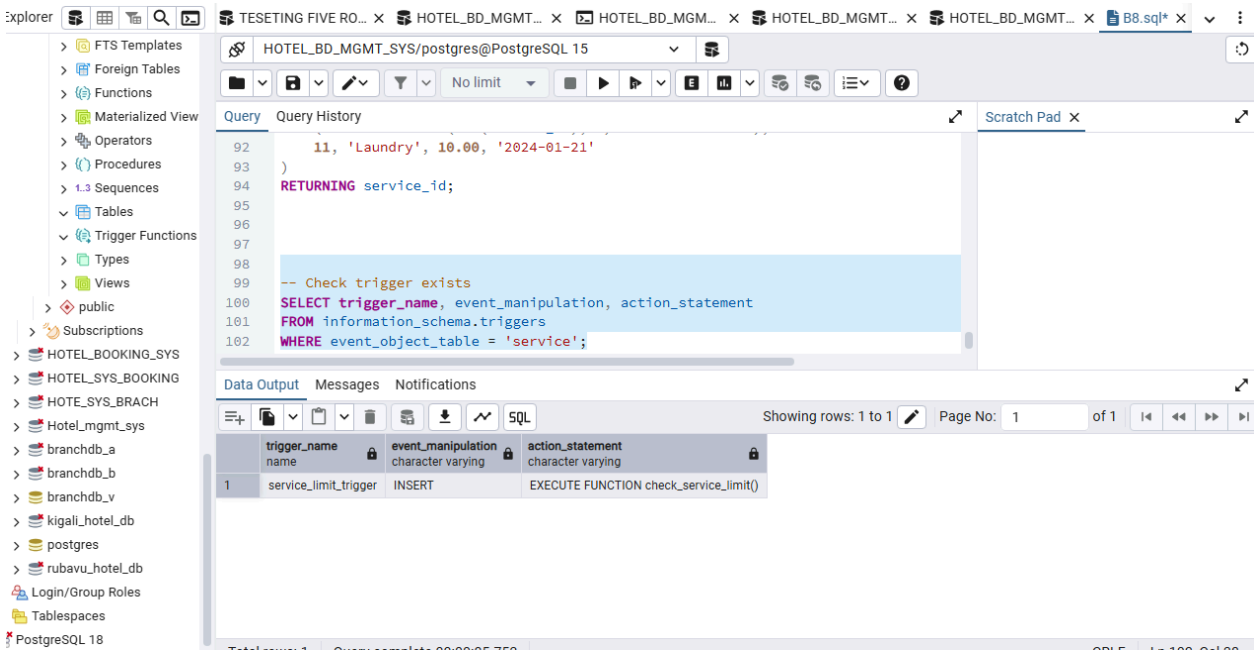
SELECT MAX(service_id) FROM Service;






```
CREATE OR REPLACE FUNCTION check_service_limit()
RETURNS TRIGGER AS $$
DECLARE
    current_total DECIMAL(10,2);
    new_total DECIMAL(10,2);
BEGIN
    -- Calculate current total for this booking (excluding current row if updating)
    SELECT COALESCE(SUM(cost), 0) INTO current_total
    FROM service
    WHERE booking_id = NEW.booking_id
      AND service_id != COALESCE(NEW.service_id, 0);

    -- Calculate what the new total would be
    new_total := current_total + NEW.cost;

    -- Check if new total would exceed $100 limit
    IF new_total > 100 THEN
        RAISE EXCEPTION
            'BUSINESS_RULE_VIOLATION: Total service cost for booking % would exceed $100
limit. Current total: $%, New service: $%, Would be: $%',
            NEW.booking_id, current_total, NEW.cost, new_total;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
-- Check trigger exists
SELECT trigger_name, event_manipulation, action_statement
FROM information_schema.triggers
WHERE event_object_table = 'service';
```



==========END==========