Mukamariechantal609@gmail.com **Advanced Database Technology: MUKANDAYISENGA MARIE CHANTAL_DB_EXAM.**

AFRICAN CENTRE OF EXCELLENCE IN DATA SCIENCE (ACE-DS)

ADVANCED DATABASE PROJECT-BASED EXAM

Module Code: DSM6235

School/Centre: African Centre of Excellence in Data Science

MUKANDAYISENGA MARIE CHANTAL

REGNO: 224019567

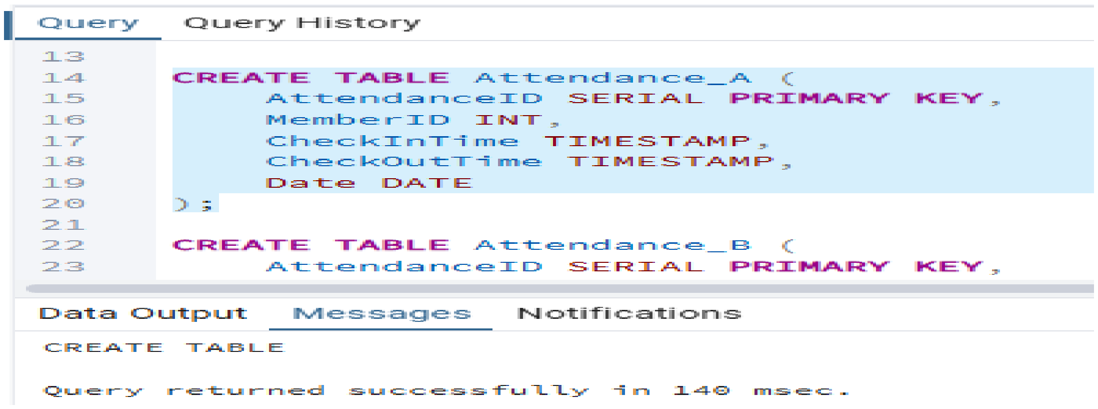
SECTION A

CASE STUDY: Gym Membership and Attendance Tracking System

A1: Fragment & Recombine Main Fact (≤10 rows)

A 1.a: Create horizontally fragmented tables Attendance_A on Node_A and Attendance_B on Node_B using a deterministic rule (HASH or RANGE on a natural key).

```
 CREATE TABLE Attendance_A (
   AttendanceID SERIAL PRIMARY KEY,
   MemberID INT,
   CheckInTime TIMESTAMP,
   CheckOutTime TIMESTAMP,
   Date DATE
);
```


Query Query History

```
CREATE TABLE Attendance_A (
    AttendanceID SERIAL PRIMARY KEY,
    MemberID INT,
    CheckInTime TIMESTAMP,
    CheckOutTime TIMESTAMP,
    Date DATE
);

CREATE TABLE Attendance_B (
    AttendanceID SERIAL PRIMARY KEY,
```

Data Output Messages Notifications

```
CREATE TABLE

Query returned successfully in 140 msec.
```

CREATE TABLE Attendance_B (

AttendanceID SERIAL PRIMARY KEY,

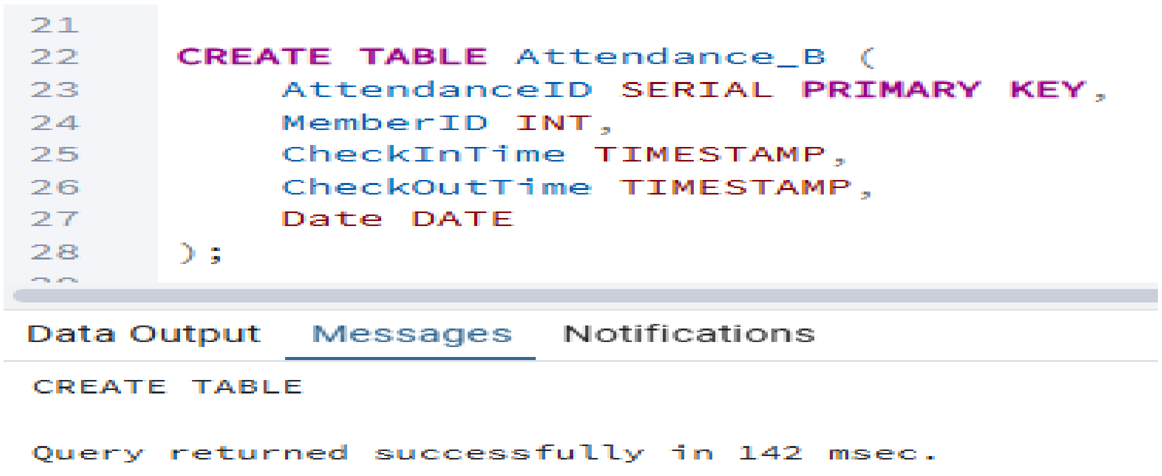
MemberID INT,

CheckInTime TIMESTAMP,

CheckOutTime TIMESTAMP,

Date DATE

);



```
CREATE TABLE Attendance_B (
    AttendanceID SERIAL PRIMARY KEY,
    MemberID INT,
    CheckInTime TIMESTAMP,
    CheckOutTime TIMESTAMP,
    Date DATE
);
```

Data Output Messages Notifications

```
CREATE TABLE

Query returned successfully in 142 msec.
```

A1.b: Insert a TOTAL of ≤10 committed rows split across the two fragments (e.g., 5 on Node_A and 5 on Node_B). Reuse these rows for all remaining tasks.

INSERT INTO Attendance_A (MemberID, CheckInTime, CheckOutTime, Date)

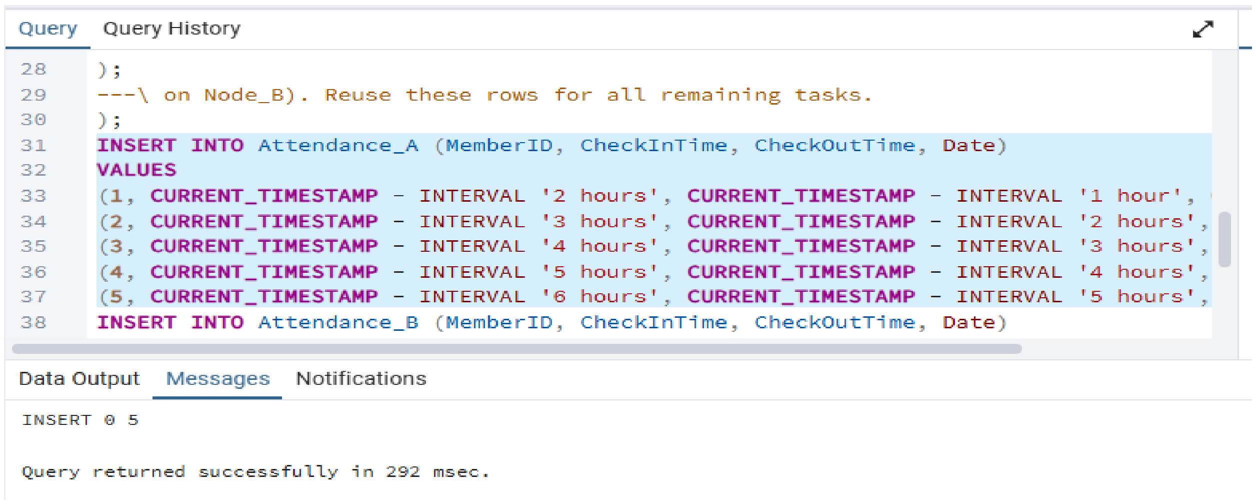
VALUES

(1, CURRENT_TIMESTAMP - INTERVAL '2 hours', CURRENT_TIMESTAMP - INTERVAL '1 hour', CURRENT_DATE), (2, CURRENT_TIMESTAMP - INTERVAL '3 hours', CURRENT_TIMESTAMP - INTERVAL '2 hours', CURRENT_DATE),

(3, CURRENT_TIMESTAMP - INTERVAL '4 hours', CURRENT_TIMESTAMP - INTERVAL '3 hours', CURRENT_DATE),

(4, CURRENT_TIMESTAMP - INTERVAL '5 hours', CURRENT_TIMESTAMP - INTERVAL '4 hours', CURRENT_DATE),

(5, CURRENT_TIMESTAMP - INTERVAL '6 hours', CURRENT_TIMESTAMP - INTERVAL '5 hours', CURRENT_DATE);



```
);
---\ on Node_B). Reuse these rows for all remaining tasks.
);
INSERT INTO Attendance_A (MemberID, CheckInTime, CheckOutTime, Date)
VALUES
(1, CURRENT_TIMESTAMP - INTERVAL '2 hours', CURRENT_TIMESTAMP - INTERVAL '1 hour',
(2, CURRENT_TIMESTAMP - INTERVAL '3 hours', CURRENT_TIMESTAMP - INTERVAL '2 hours',
(3, CURRENT_TIMESTAMP - INTERVAL '4 hours', CURRENT_TIMESTAMP - INTERVAL '3 hours',
(4, CURRENT_TIMESTAMP - INTERVAL '5 hours', CURRENT_TIMESTAMP - INTERVAL '4 hours',
(5, CURRENT_TIMESTAMP - INTERVAL '6 hours', CURRENT_TIMESTAMP - INTERVAL '5 hours',
INSERT INTO Attendance_B (MemberID, CheckInTime, CheckOutTime, Date)
```

```
INSERT 0 5

Query returned successfully in 292 msec.
```

INSERT INTO Attendance_B (MemberID, CheckInTime, CheckOutTime, Date)

VALUES

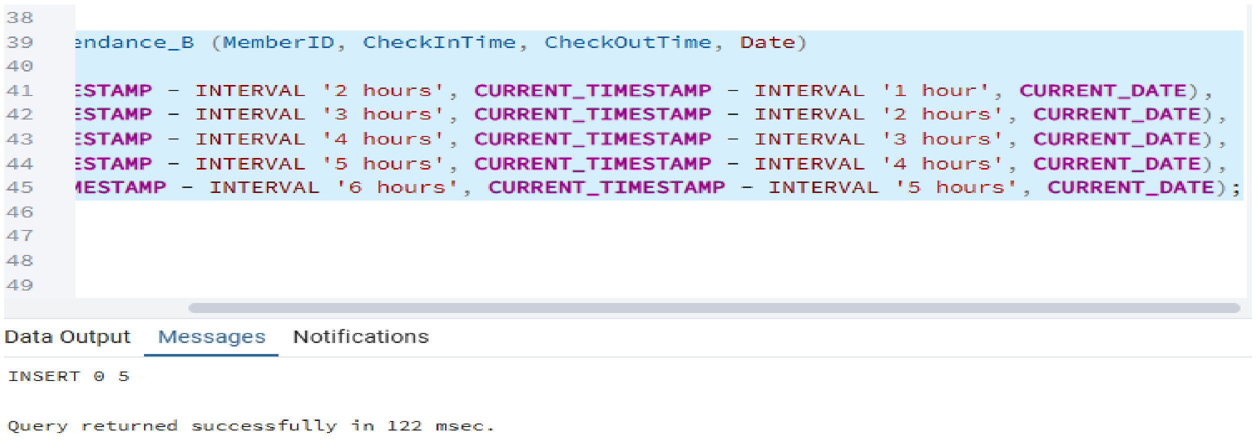
(6, CURRENT_TIMESTAMP - INTERVAL '2 hours', CURRENT_TIMESTAMP - INTERVAL '1 hour', CURRENT_DATE),

(7, CURRENT_TIMESTAMP - INTERVAL '3 hours', CURRENT_TIMESTAMP - INTERVAL '2 hours', CURRENT_DATE),

(8, CURRENT_TIMESTAMP - INTERVAL '4 hours', CURRENT_TIMESTAMP - INTERVAL '3 hours', CURRENT_DATE),

(9, CURRENT_TIMESTAMP - INTERVAL '5 hours', CURRENT_TIMESTAMP - INTERVAL '4 hours', CURRENT_DATE),

(10, CURRENT_TIMESTAMP - INTERVAL '6 hours', CURRENT_TIMESTAMP - INTERVAL '5 hours', CURRENT_DATE);



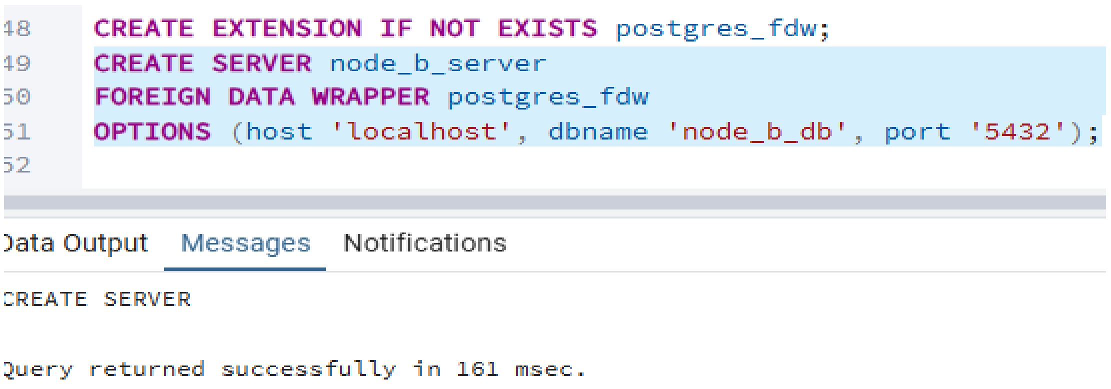
```
endance_B (MemberID, CheckInTime, CheckOutTime, Date)

ESTAMP - INTERVAL '2 hours', CURRENT_TIMESTAMP - INTERVAL '1 hour', CURRENT_DATE),
ESTAMP - INTERVAL '3 hours', CURRENT_TIMESTAMP - INTERVAL '2 hours', CURRENT_DATE),
ESTAMP - INTERVAL '4 hours', CURRENT_TIMESTAMP - INTERVAL '3 hours', CURRENT_DATE),
ESTAMP - INTERVAL '5 hours', CURRENT_TIMESTAMP - INTERVAL '4 hours', CURRENT_DATE),
MESTAMP - INTERVAL '6 hours', CURRENT_TIMESTAMP - INTERVAL '5 hours', CURRENT_DATE);
```

Data Output Messages Notifications

```
INSERT 0 5

Query returned successfully in 122 msec.
```

A1.c: On Node_A, create view Attendance_ALL as UNION ALL of Attendance_A and Attendance_B@proj_link

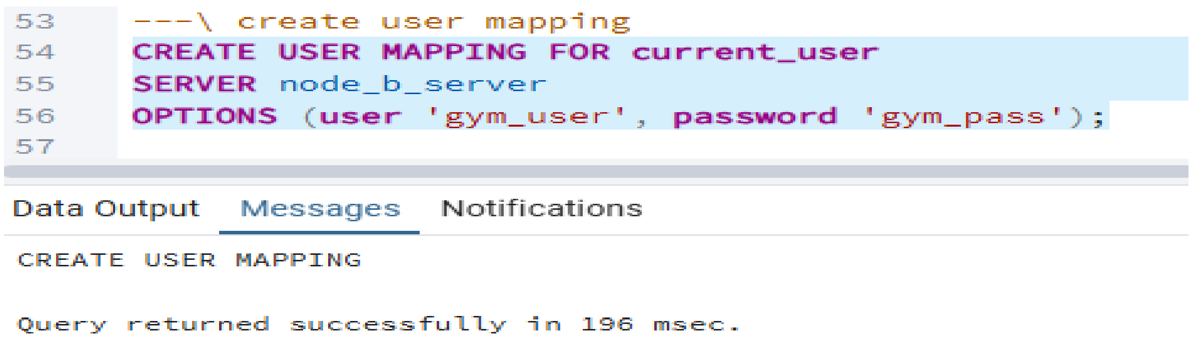
Create server node_b_server and extension postegres_fdw.



```
CREATE EXTENSION IF NOT EXISTS postgres_fdw;
CREATE SERVER node_b_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'localhost', dbname 'node_b_db', port '5432');
```

Data Output Messages Notifications

```
CREATE SERVER

Query returned successfully in 161 msec.
```

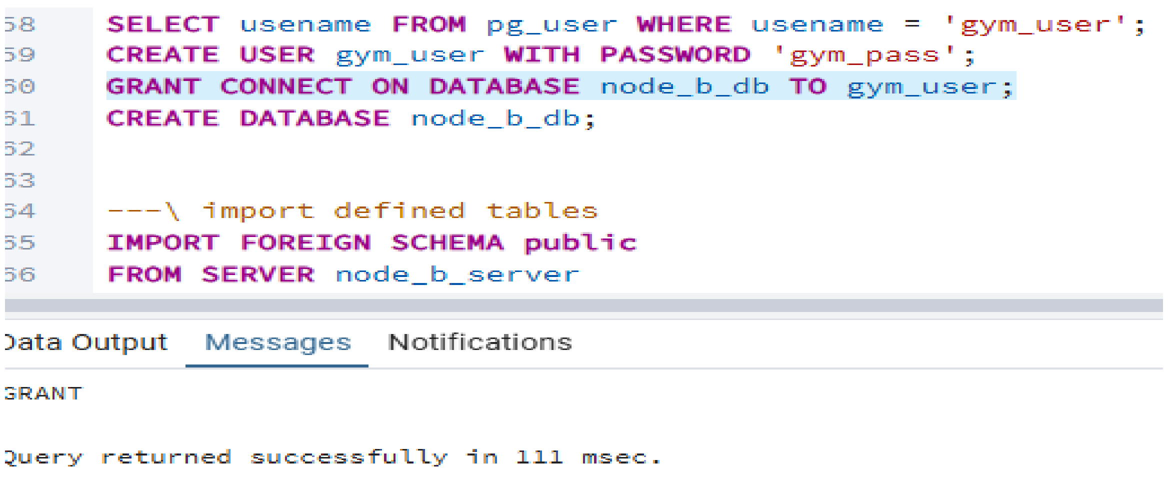


```
---\ create user mapping
CREATE USER MAPPING FOR current_user
SERVER node_b_server
OPTIONS (user 'gym_user', password 'gym_pass');
```

Data Output Messages Notifications

```
CREATE USER MAPPING

Query returned successfully in 196 msec.
```

For this step I create database node_b_db[; and grant it to the gym_user.

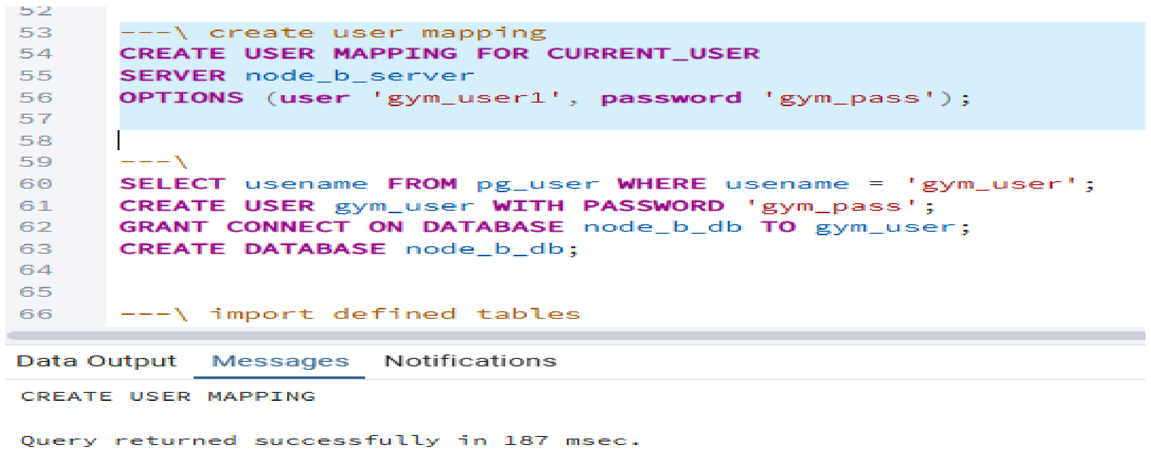


```
SELECT usename FROM pg_user WHERE usename = 'gym_user';
CREATE USER gym_user WITH PASSWORD 'gym_pass';
GRANT CONNECT ON DATABASE node_b_db TO gym_user;
CREATE DATABASE node_b_db;


---\ import defined tables
IMPORT FOREIGN SCHEMA public
FROM SERVER node_b_server
```

Data Output Messages Notifications

```
GRANT

Query returned successfully in 111 msec.
```

In this step I drop user mapping because is appear more than once, then after dropping it I create a new once user for this mapping

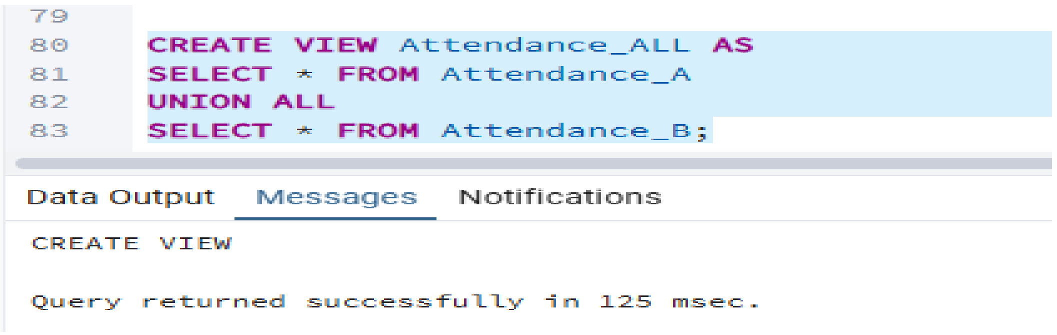


```
---\ create user mapping
CREATE USER MAPPING FOR CURRENT_USER
SERVER node_b_server
OPTIONS (user 'gym_user1', password 'gym_pass');


---\
SELECT usename FROM pg_user WHERE usename = 'gym_user';
CREATE USER gym_user WITH PASSWORD 'gym_pass';
GRANT CONNECT ON DATABASE node_b_db TO gym_user;
CREATE DATABASE node_b_db;


---\ import defined tables
```

Data Output Messages Notifications

```
CREATE USER MAPPING

Query returned successfully in 187 msec.
```

Create the view

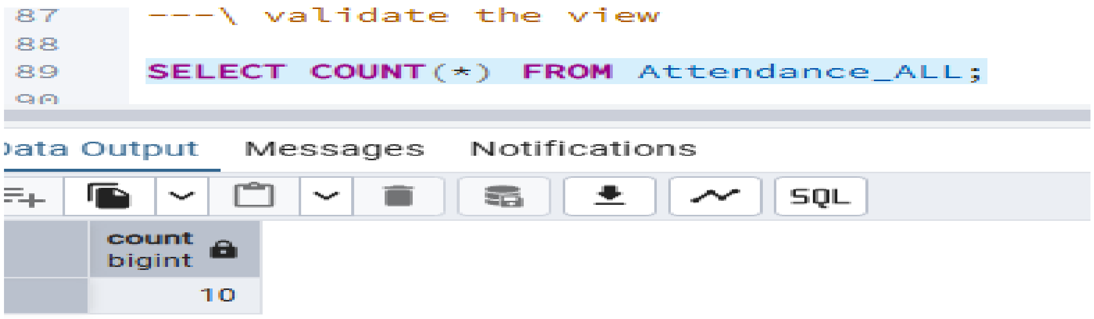


```
CREATE VIEW Attendance_ALL AS
SELECT * FROM Attendance_A
UNION ALL
SELECT * FROM Attendance_B;
```

Data Output Messages Notifications

```
CREATE VIEW

Query returned successfully in 125 msec.
```

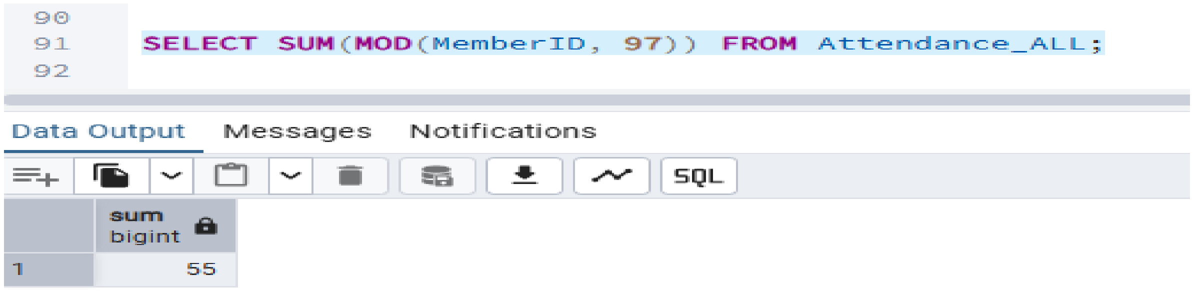
Validate the view.



```
---\ validate the view

SELECT COUNT(*) FROM Attendance_ALL;
```

```
count
bigint
   10
```

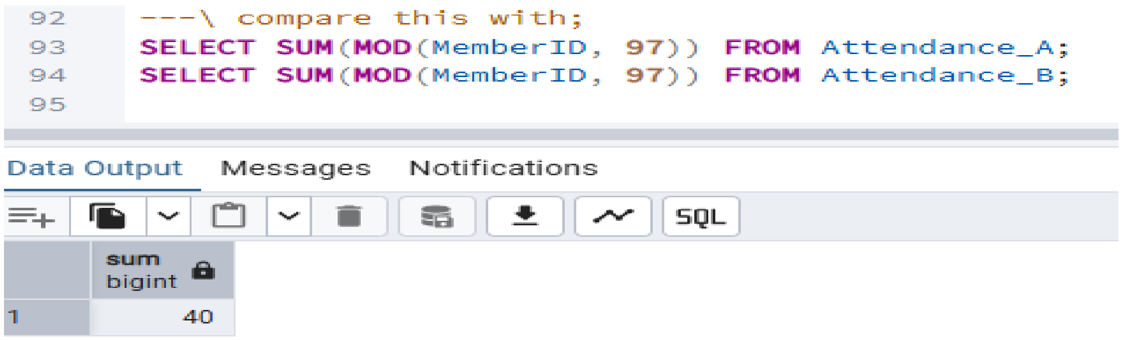
Checksum



```
SELECT SUM(MOD(MemberID, 97)) FROM Attendance_ALL;
```

```
  sum
  bigint
1    55
```

Compare this with;

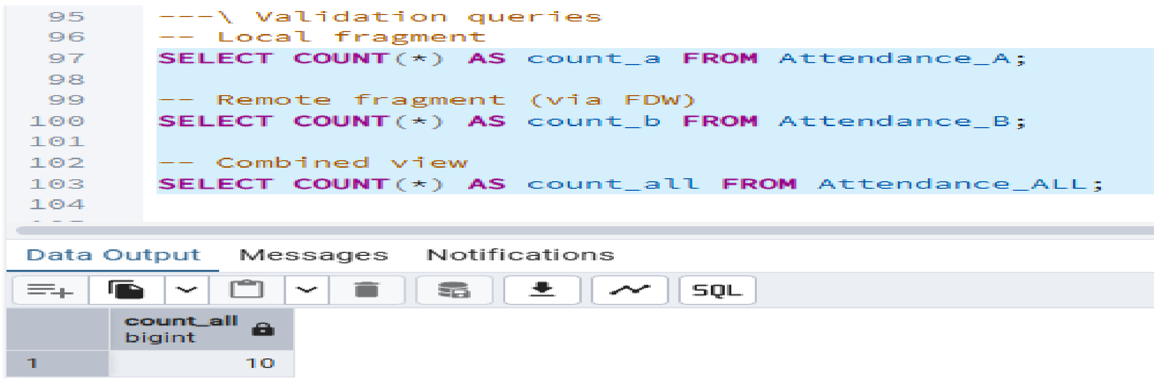


```
---\ compare this with;
SELECT SUM(MOD(MemberID, 97)) FROM Attendance_A;
SELECT SUM(MOD(MemberID, 97)) FROM Attendance_B;
```

```
  sum
  bigint
1    40
```

A1.d: Validate with COUNT(*) and a checksum on a key column (e.g., SUM(MOD(primary_key,97))) :results must match fragments vs Attendance_ALL.

Validation query



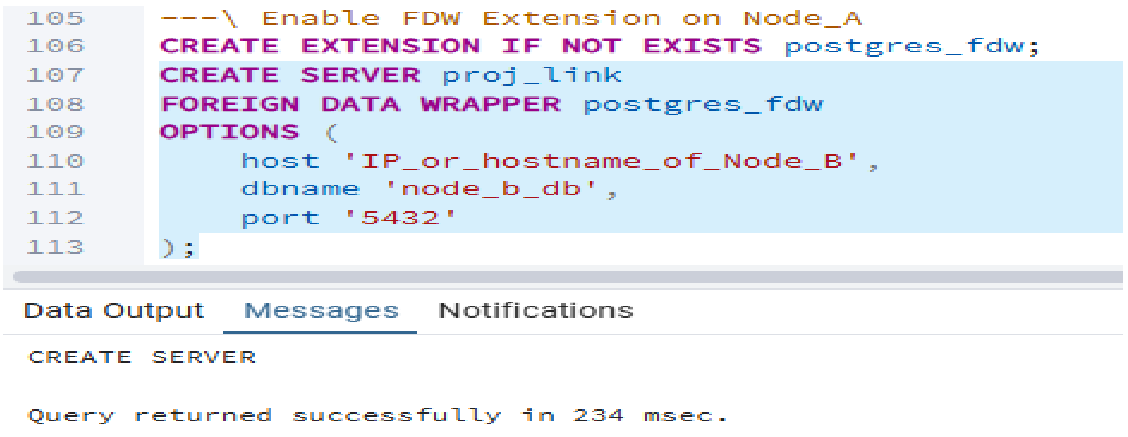
```
---\ Validation queries
-- Local fragment
SELECT COUNT(*) AS count_a FROM Attendance_A;

-- Remote fragment (via FDW)
SELECT COUNT(*) AS count_b FROM Attendance_B;

-- Combined view
SELECT COUNT(*) AS count_all FROM Attendance_ALL;
```

```
  count_all
  bigint
1        10
```

A2 :Database Link & Cross-Node Join (3–10 rows result)

A2.a: From Node_A, create database link 'proj_link' to Node_B.

Enable FDW Extension on Node_A

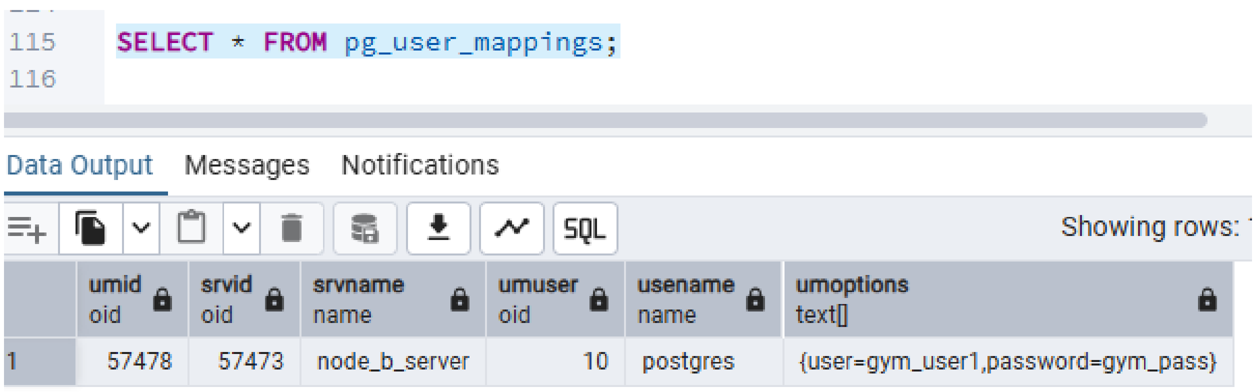


```
---\ Enable FDW Extension on Node_A
CREATE EXTENSION IF NOT EXISTS postgres_fdw;
CREATE SERVER proj_link
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (
    host 'IP_or_hostname_of_Node_B',
    dbname 'node_b_db',
    port '5432'
);
```

Data Output Messages Notifications

```
CREATE SERVER

Query returned successfully in 234 msec.
```

Cheching user on Node_A



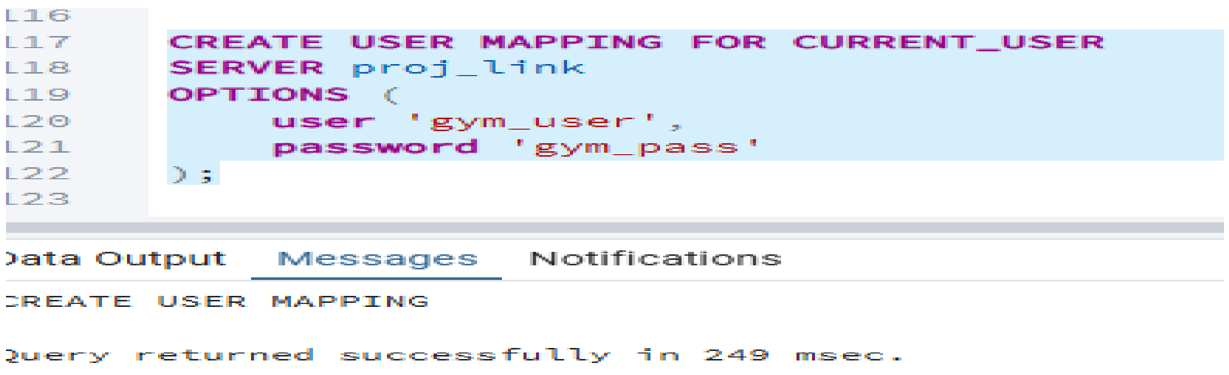
```
SELECT * FROM pg_user_mappings;
```

Data Output Messages Notifications

Showing rows:

	umid oid	srvid oid	srvname name	umuser oid	usename name	umoptions text[]
1	57478	57473	node_b_server	10	postgres	{user=gym_user1,password=gym_pass}

Create user mapping

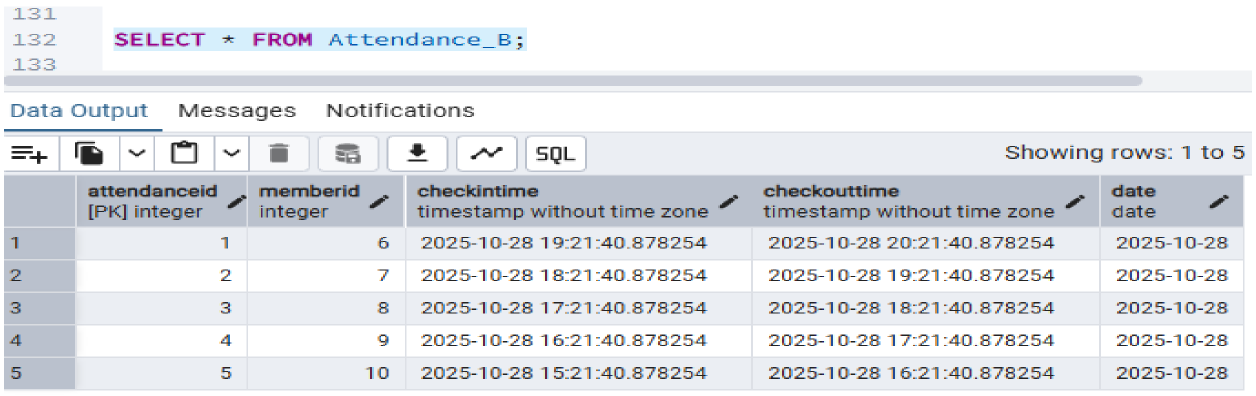


```
CREATE USER MAPPING FOR CURRENT_USER
SERVER proj_link
OPTIONS (
        user 'gym_user',
        password 'gym_pass'
);
```

Data Output Messages Notifications

```
CREATE USER MAPPING

Query returned successfully in 249 msec.
```

Test the link



```
SELECT * FROM Attendance_B;
```

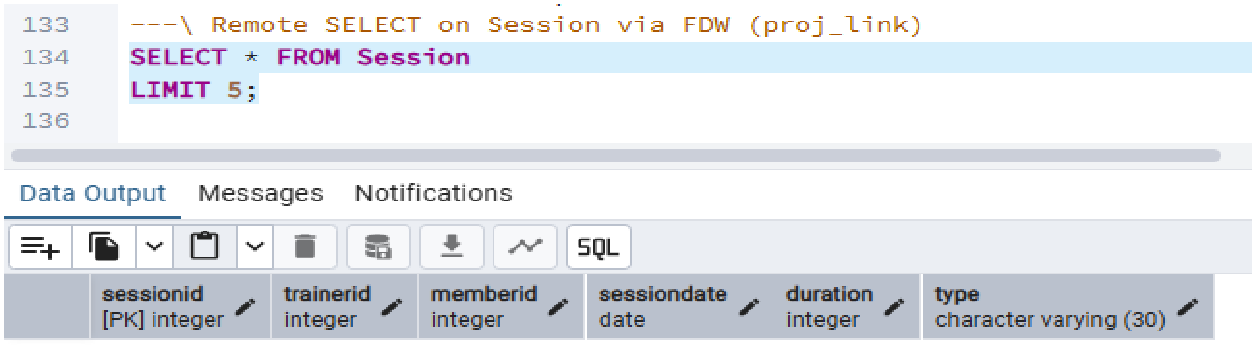
Data Output | Messages | Notifications

Showing rows: 1 to 5

	attendanceid [PK] integer	memberid integer	checkintime timestamp without time zone	checkouttime timestamp without time zone	date date
1	1	6	2025-10-28 19:21:40.878254	2025-10-28 20:21:40.878254	2025-10-28
2	2	7	2025-10-28 18:21:40.878254	2025-10-28 19:21:40.878254	2025-10-28
3	3	8	2025-10-28 17:21:40.878254	2025-10-28 18:21:40.878254	2025-10-28
4	4	9	2025-10-28 16:21:40.878254	2025-10-28 17:21:40.878254	2025-10-28
5	5	10	2025-10-28 15:21:40.878254	2025-10-28 16:21:40.878254	2025-10-28

A2.b: Run remote SELECT on Session@proj_link showing up to 5 sample rows.

You already created a **foreign table** called Session on Node_A

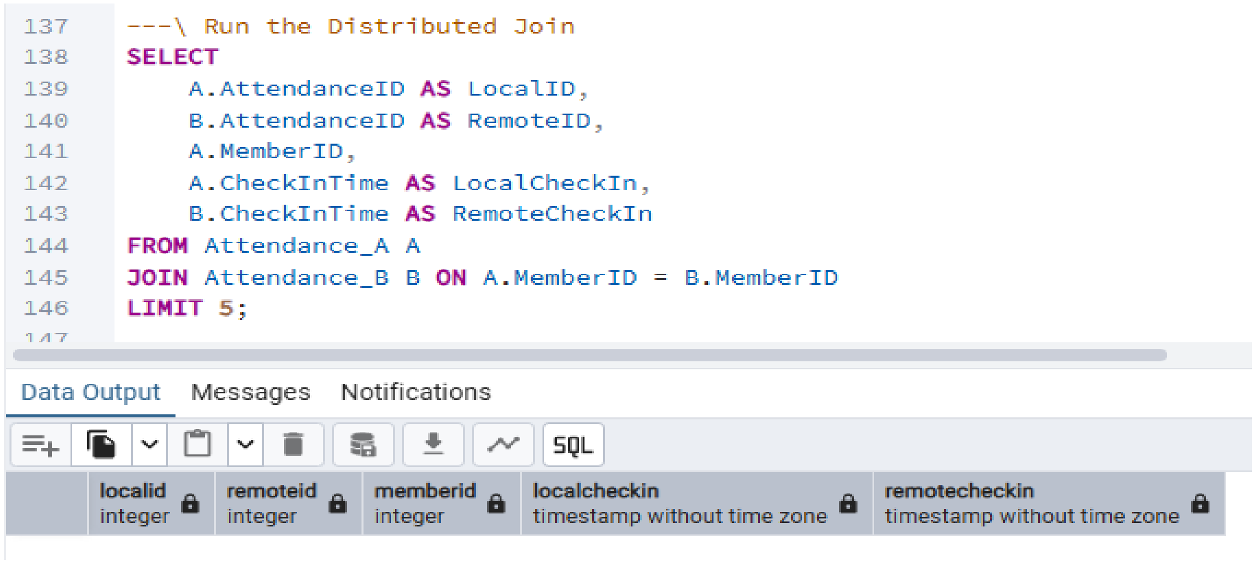


```
---\ Remote SELECT on Session via FDW (proj_link)
SELECT * FROM Session
LIMIT 5;
```

Data Output | Messages | Notifications

sessionid [PK] integer	trainerid integer	memberid integer	sessiondate date	duration integer	type character varying (30)

A2.c: Run a distributed join: local Attendance_A (or base Attendance) joined with remote

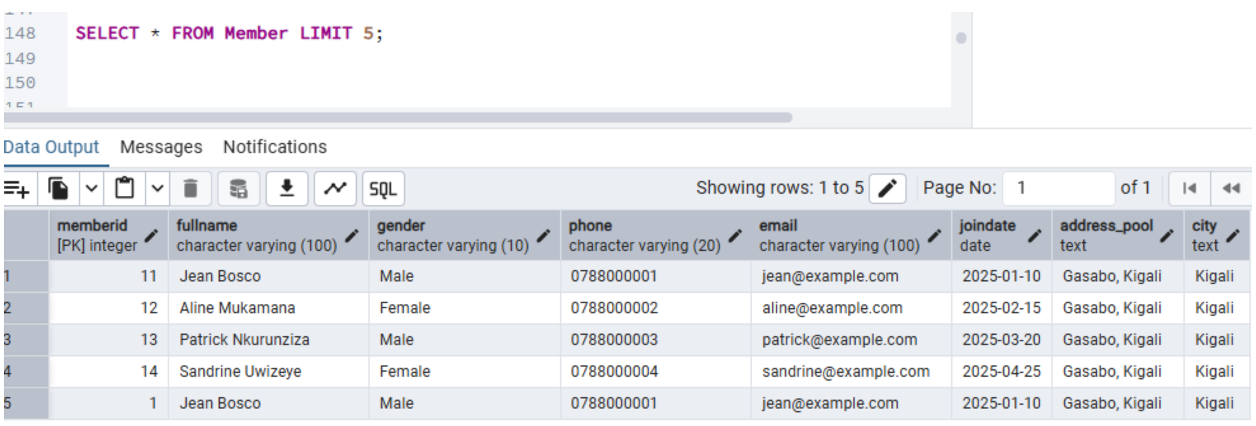


```
---\ Run the Distributed Join
SELECT
    A.AttendanceID AS LocalID,
    B.AttendanceID AS RemoteID,
    A.MemberID,
    A.CheckInTime AS LocalCheckIn,
    B.CheckInTime AS RemoteCheckIn
FROM Attendance_A A
JOIN Attendance_B B ON A.MemberID = B.MemberID
LIMIT 5;
```

Data Output | Messages | Notifications

localid integer	remoteid integer	memberid integer	localcheckin timestamp without time zone	remotecheckin timestamp without time zone

Test foreign table member



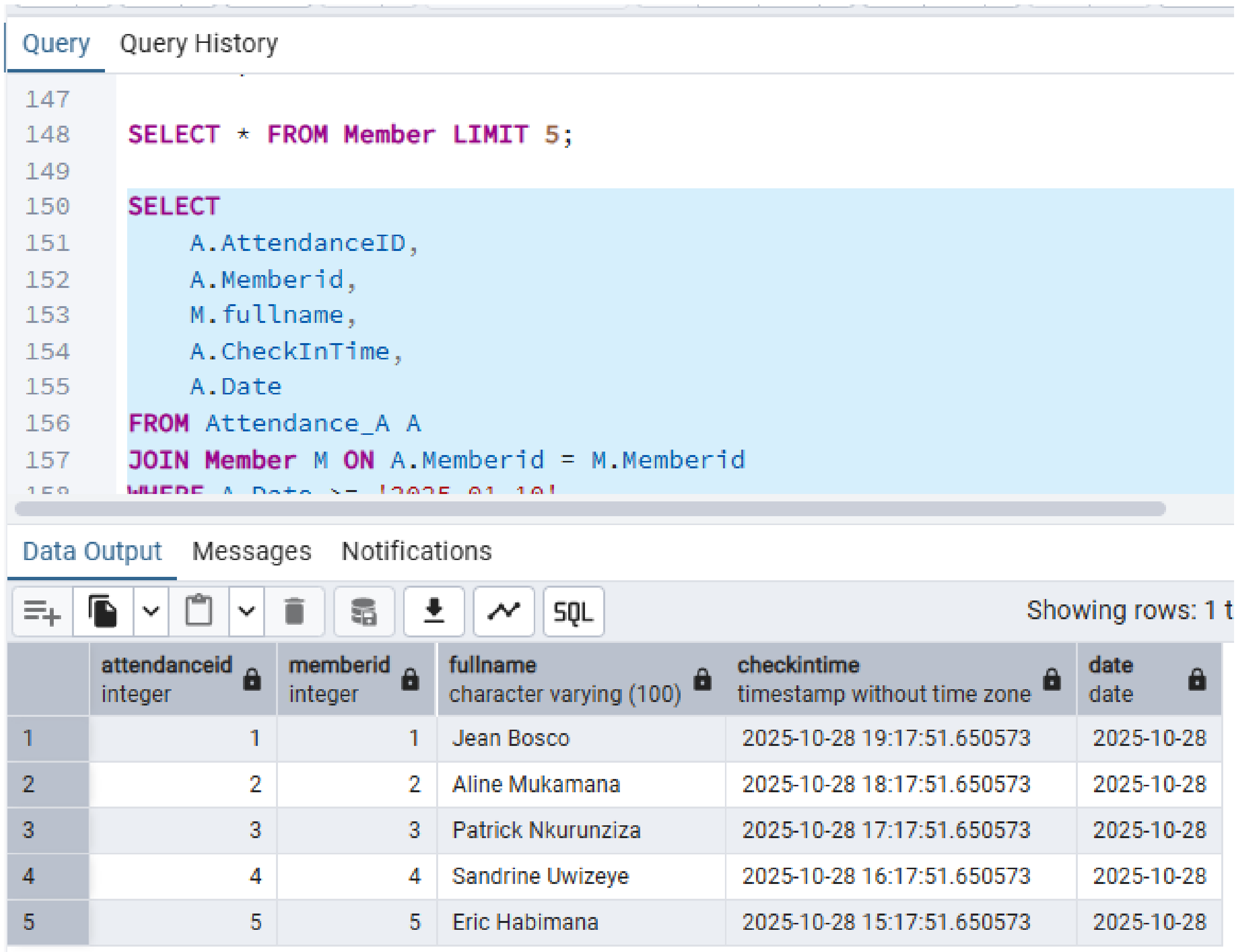
```
SELECT * FROM Member LIMIT 5;
```

Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1 of 1

	memberid [PK] integer	fullname character varying (100)	gender character varying (10)	phone character varying (20)	email character varying (100)	joindate date	address_pool text	city text
1	11	Jean Bosco	Male	0788000001	jean@example.com	2025-01-10	Gasabo, Kigali	Kigali
2	12	Aline Mukamana	Female	0788000002	aline@example.com	2025-02-15	Gasabo, Kigali	Kigali
3	13	Patrick Nkurunziza	Male	0788000003	patrick@example.com	2025-03-20	Gasabo, Kigali	Kigali
4	14	Sandrine Uwizeye	Female	0788000004	sandrine@example.com	2025-04-25	Gasabo, Kigali	Kigali
5	1	Jean Bosco	Male	0788000001	jean@example.com	2025-01-10	Gasabo, Kigali	Kigali

Run Your Distributed Join



Query Query History

```
SELECT * FROM Member LIMIT 5;

SELECT
    A.AttendanceID,
    A.Memberid,
    M.fullname,
    A.CheckInTime,
    A.Date
FROM Attendance_A A
JOIN Member M ON A.Memberid = M.Memberid
WHERE A.Date >= '2025-01-10'
```

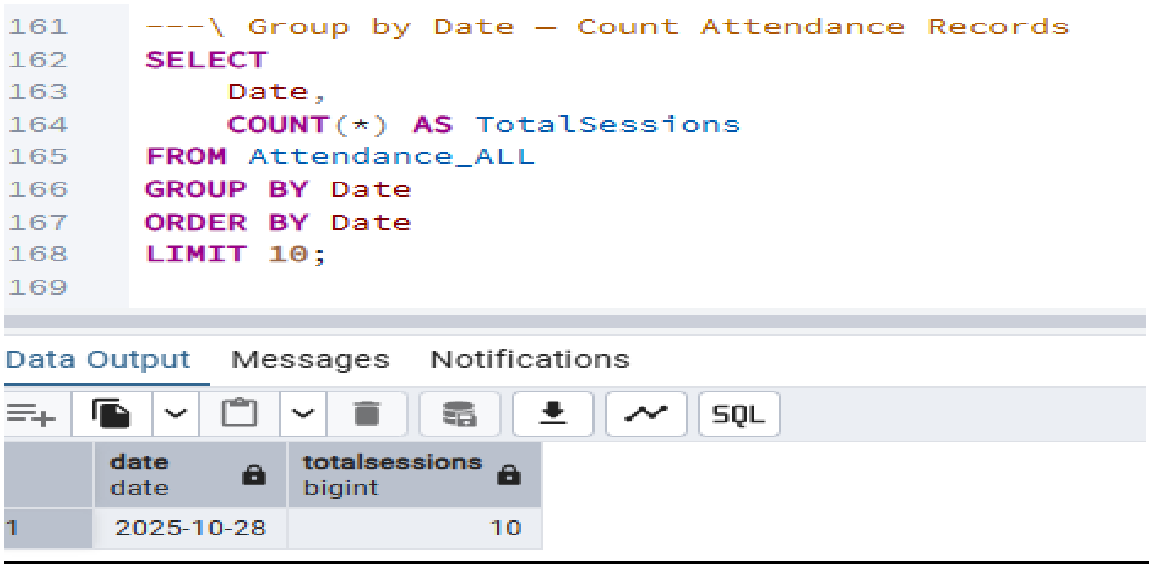
Data Output Messages Notifications

Showing rows: 1 t

	attendanceid integer	memberid integer	fullname character varying (100)	checkintime timestamp without time zone	date date
1	1	1	Jean Bosco	2025-10-28 19:17:51.650573	2025-10-28
2	2	2	Aline Mukamana	2025-10-28 18:17:51.650573	2025-10-28
3	3	3	Patrick Nkurunziza	2025-10-28 17:17:51.650573	2025-10-28
4	4	4	Sandrine Uwizeye	2025-10-28 16:17:51.650573	2025-10-28
5	5	5	Eric Habimana	2025-10-28 15:17:51.650573	2025-10-28

A3 :Parallel vs Serial Aggregation (≤10 rows data

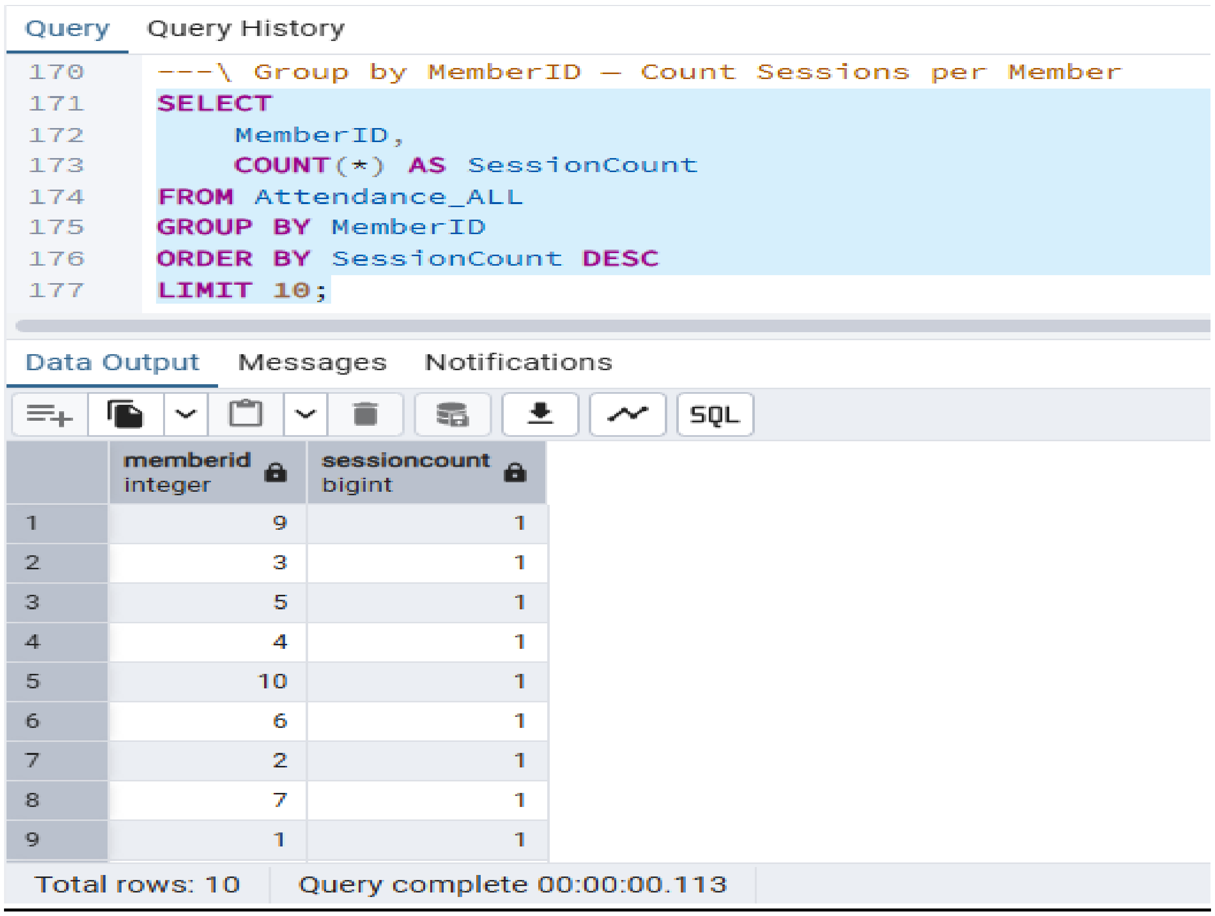
A3.a: Run a SERIAL aggregation on Attendance_ALL over the small dataset (e.g., totals by a domain column). Ensure result has 3–10 groups/rows.



```
---\ Group by Date — Count Attendance Records
SELECT
    Date,
    COUNT(*) AS TotalSessions
FROM Attendance_ALL
GROUP BY Date
ORDER BY Date
LIMIT 10;
```

Data Output Messages Notifications

	date date	totalsessions bigint
1	2025-10-28	10



Query Query History

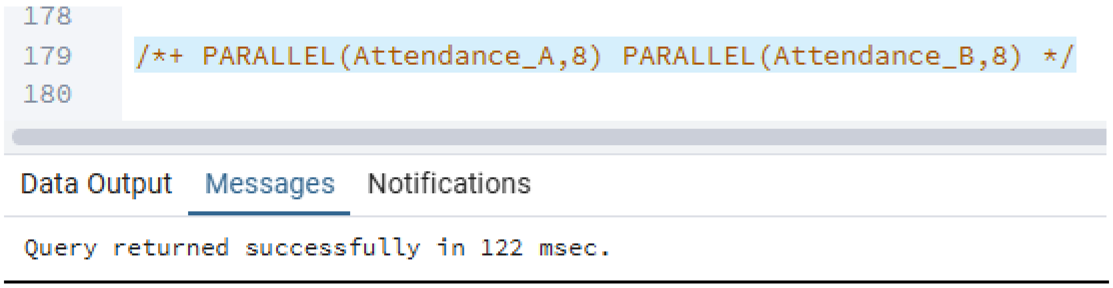
```
---\ Group by MemberID — Count Sessions per Member
SELECT
    MemberID,
    COUNT(*) AS SessionCount
FROM Attendance_ALL
GROUP BY MemberID
ORDER BY SessionCount DESC
LIMIT 10;
```

Data Output Messages Notifications

	memberid integer	sessioncount bigint
1	9	1
2	3	1
3	5	1
4	4	1
5	10	1
6	6	1
7	2	1
8	7	1
9	1	1

Total rows: 10 Query complete 00:00:00.113

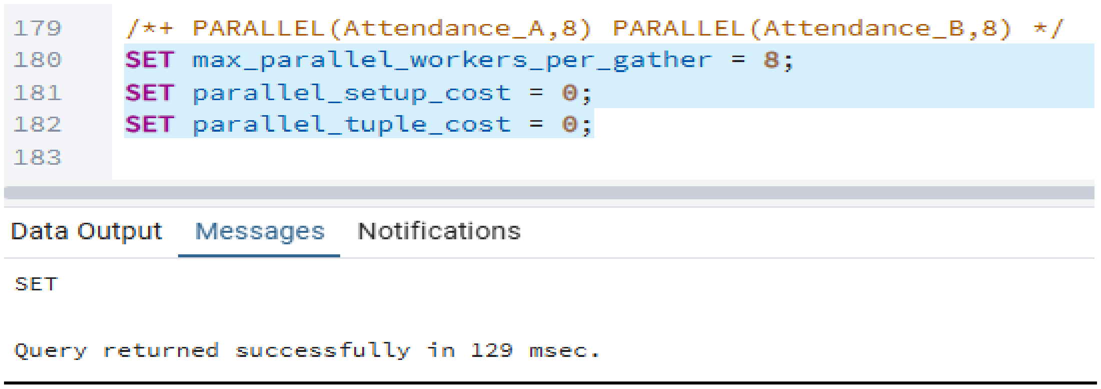
<u>A3.c: Run the same aggregation with /*+ PARALLEL(Attendance_A,8) PARALLEL(Attendance_B,8) */ to force a parallel plan despite small size.</u>



```
/*+ PARALLEL(Attendance_A,8) PARALLEL(Attendance_B,8) */
```

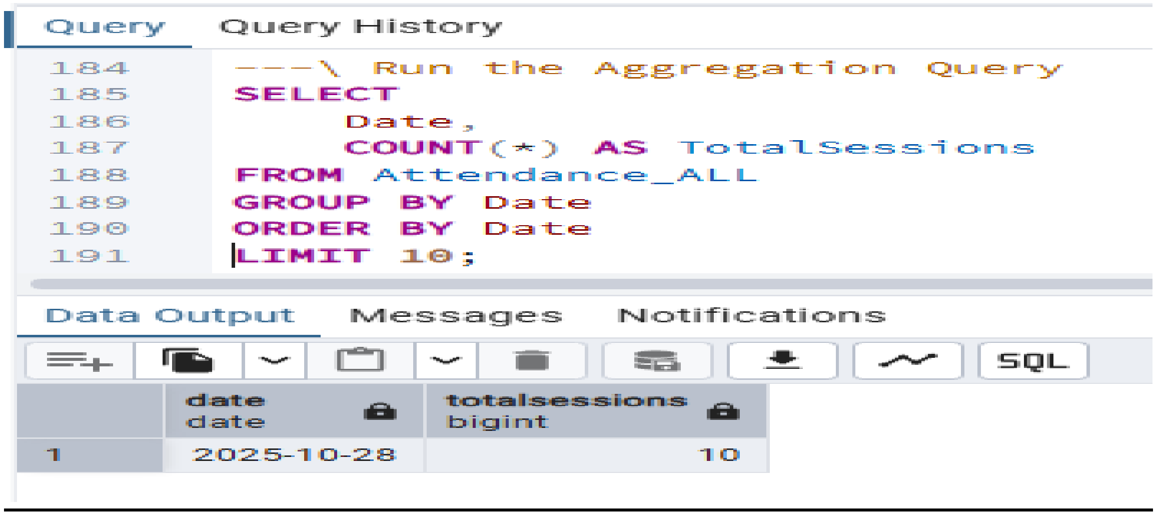
```
Query returned successfully in 122 msec.
```

Encourage Parallelism in PostgreSQL



```
/*+ PARALLEL(Attendance_A,8) PARALLEL(Attendance_B,8) */
SET max_parallel_workers_per_gather = 8;
SET parallel_setup_cost = 0;
SET parallel_tuple_cost = 0;
```

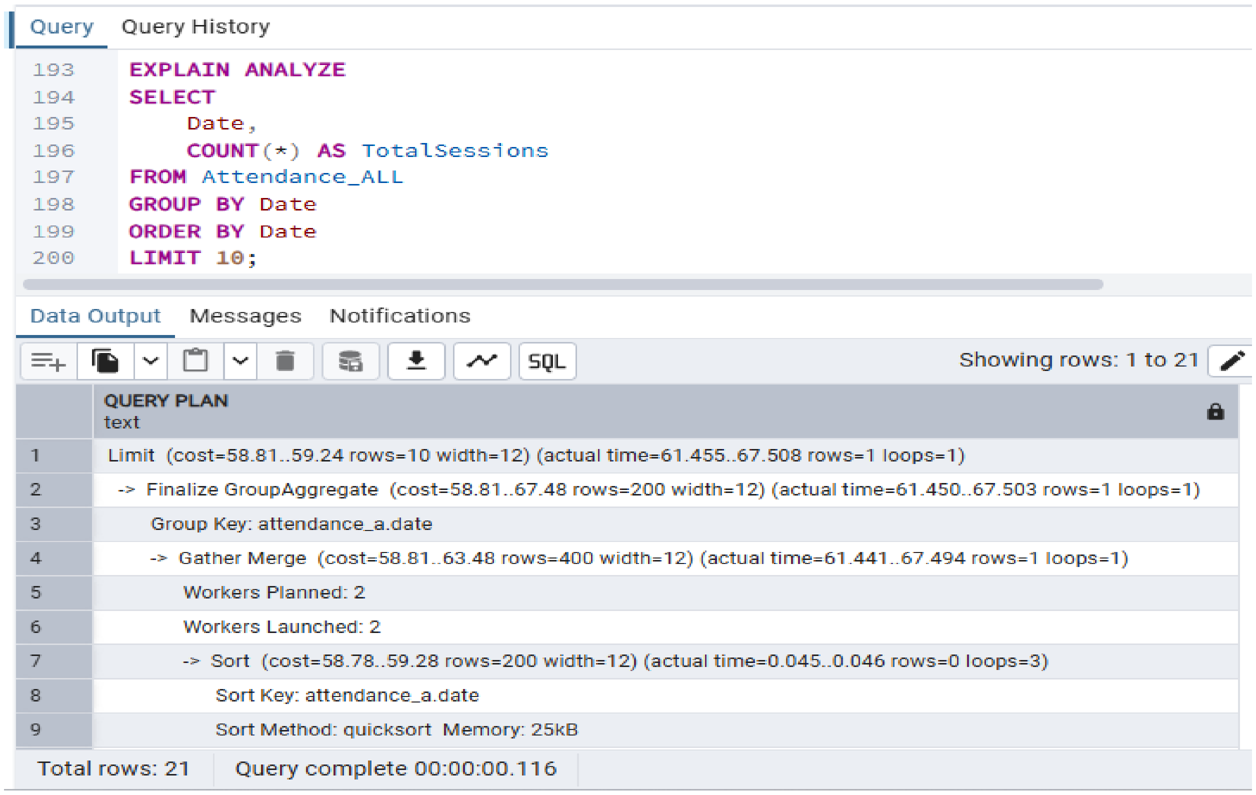
```
SET

Query returned successfully in 129 msec.
```



```
---\ Run the Aggregation Query
SELECT
        Date,
        COUNT(*) AS TotalSessions
FROM Attendance_ALL
GROUP BY Date
ORDER BY Date
LIMIT 10;
```

	date date	totalsessions bigint
1	2025-10-28	10

Check the Execution Plan



Query Query History

```
EXPLAIN ANALYZE
SELECT
    Date,
    COUNT(*) AS TotalSessions
FROM Attendance_ALL
GROUP BY Date
ORDER BY Date
LIMIT 10;
```

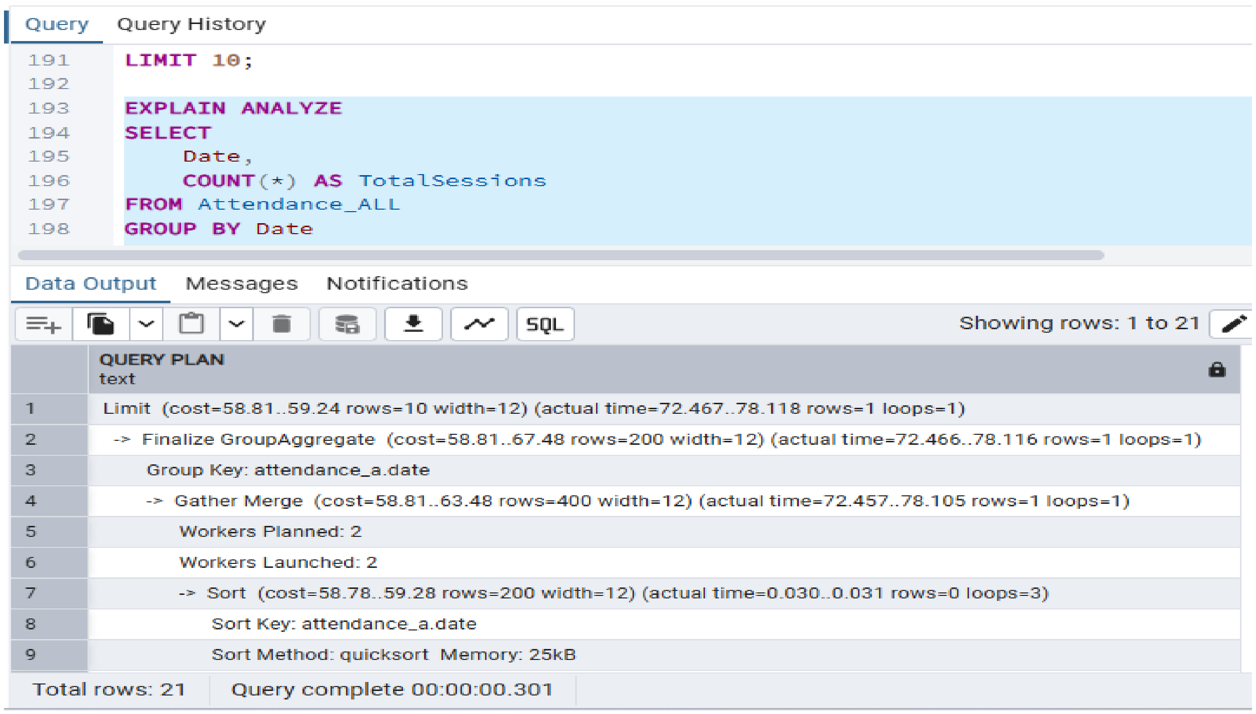
Data Output Messages Notifications

Showing rows: 1 to 21

```
QUERY PLAN
text
1 Limit  (cost=58.81..59.24 rows=10 width=12) (actual time=61.455..67.508 rows=1 loops=1)
2   -> Finalize GroupAggregate  (cost=58.81..67.48 rows=200 width=12) (actual time=61.450..67.503 rows=1 loops=1)
3         Group Key: attendance_a.date
4         -> Gather Merge  (cost=58.81..63.48 rows=400 width=12) (actual time=61.441..67.494 rows=1 loops=1)
5               Workers Planned: 2
6               Workers Launched: 2
7               -> Sort  (cost=58.78..59.28 rows=200 width=12) (actual time=0.045..0.046 rows=0 loops=3)
8                     Sort Key: attendance_a.date
9                     Sort Method: quicksort  Memory: 25kB
```

Total rows: 21   Query complete 00:00:00.116

PostgreSQL Equivalent to Oracle's DBMS_XPLAN and AUTOTRACE



Query Query History

```
LIMIT 10;

EXPLAIN ANALYZE
SELECT
    Date,
    COUNT(*) AS TotalSessions
FROM Attendance_ALL
GROUP BY Date
```

Data Output Messages Notifications

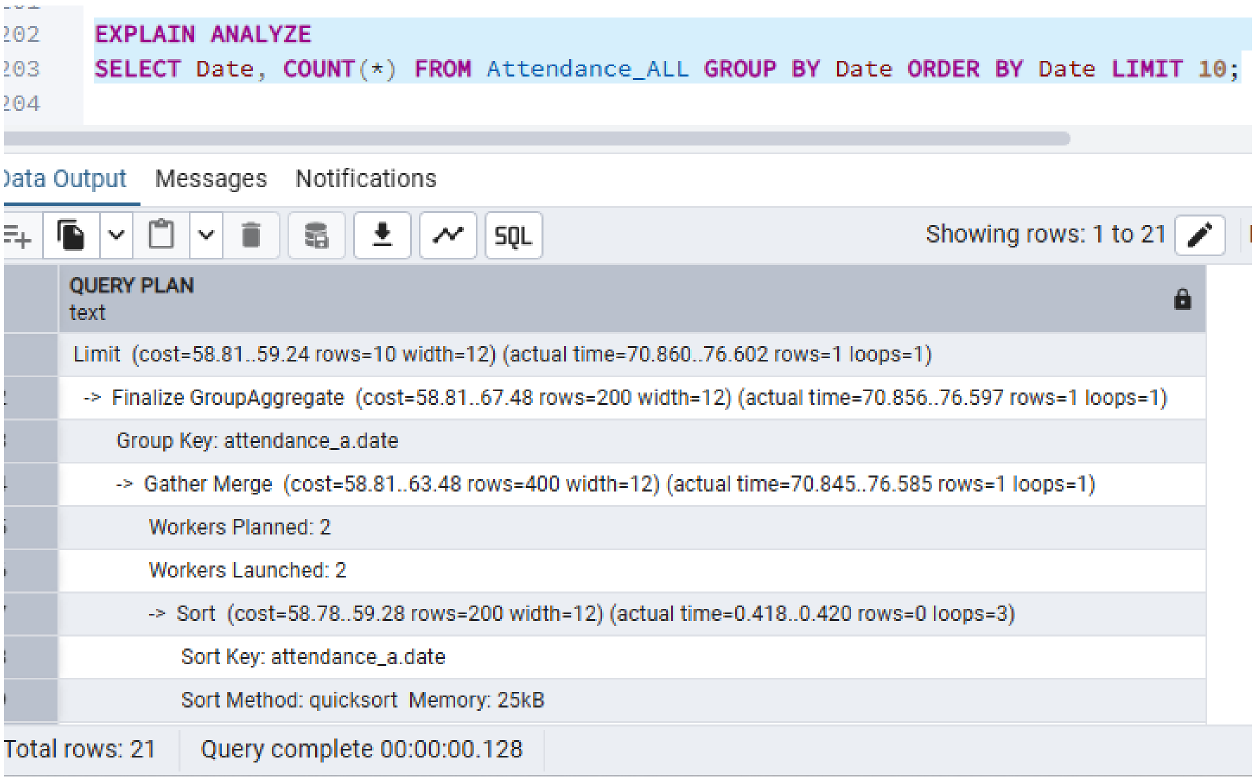
Showing rows: 1 to 21

```
QUERY PLAN
text
1 Limit  (cost=58.81..59.24 rows=10 width=12) (actual time=72.467..78.118 rows=1 loops=1)
2   -> Finalize GroupAggregate  (cost=58.81..67.48 rows=200 width=12) (actual time=72.466..78.116 rows=1 loops=1)
3         Group Key: attendance_a.date
4         -> Gather Merge  (cost=58.81..63.48 rows=400 width=12) (actual time=72.457..78.105 rows=1 loops=1)
5               Workers Planned: 2
6               Workers Launched: 2
7               -> Sort  (cost=58.78..59.28 rows=200 width=12) (actual time=0.030..0.031 rows=0 loops=3)
8                     Sort Key: attendance_a.date
9                     Sort Method: quicksort  Memory: 25kB
```

Total rows: 21   Query complete 00:00:00.301

A3.d: Produce a 2-row comparison table (serial vs parallel) with plan notes.

Serial vs Parallel Aggregation Plan Comparison



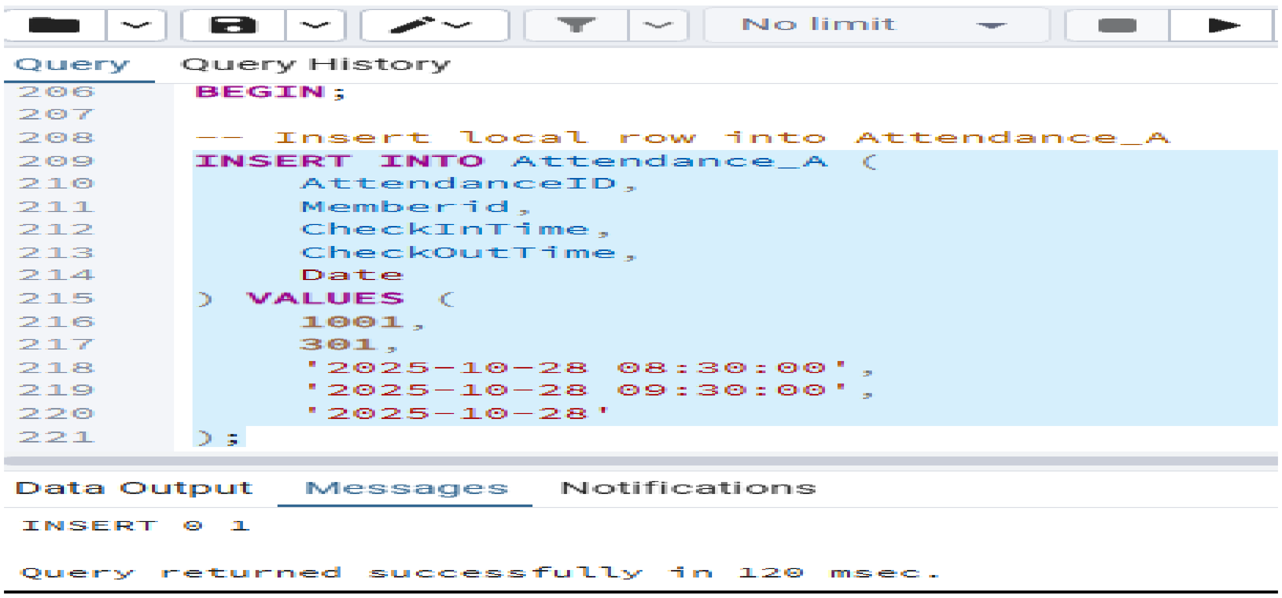
```
EXPLAIN ANALYZE
SELECT Date, COUNT(*) FROM Attendance_ALL GROUP BY Date ORDER BY Date LIMIT 10;
```

```
Limit  (cost=58.81..59.24 rows=10 width=12) (actual time=70.860..76.602 rows=1 loops=1)
  ->  Finalize GroupAggregate  (cost=58.81..67.48 rows=200 width=12) (actual time=70.856..76.597 rows=1 loops=1)
        Group Key: attendance_a.date
        ->  Gather Merge  (cost=58.81..63.48 rows=400 width=12) (actual time=70.845..76.585 rows=1 loops=1)
              Workers Planned: 2
              Workers Launched: 2
              ->  Sort  (cost=58.78..59.28 rows=200 width=12) (actual time=0.418..0.420 rows=0 loops=3)
                    Sort Key: attendance_a.date
                    Sort Method: quicksort  Memory: 25kB
```

Total rows: 21   Query complete 00:00:00.128

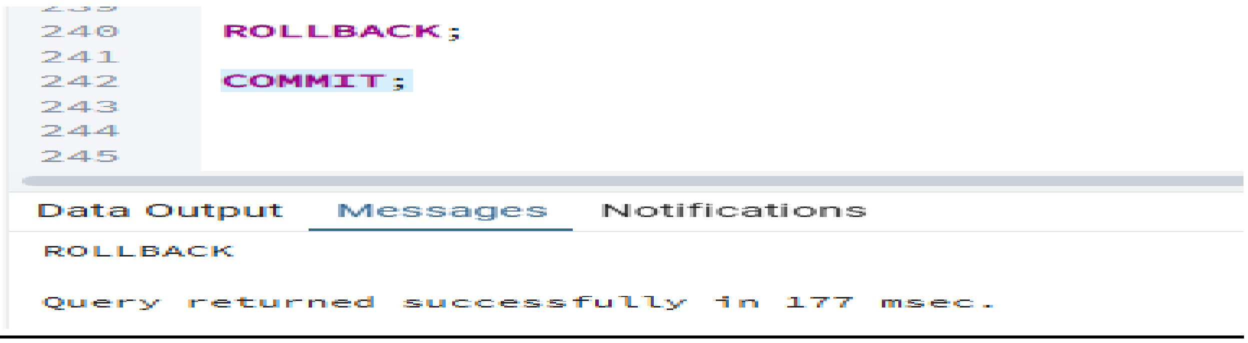
<u>A4 :Two-Phase Commit & Recovery (2 rows)</u>

<u>A4.a:</u> Write one PL/SQL block that inserts ONE local row (related to Attendance) on Node_A and ONE remote row into Payment@proj_link (or Subscription@proj_link); then COMMIT.



```
BEGIN;

-- Insert local row into Attendance_A
INSERT INTO Attendance_A (
    AttendanceID,
    Memberid,
    CheckInTime,
    CheckOutTime,
    Date
) VALUES (
    1001,
    301,
    '2025-10-28 08:30:00',
    '2025-10-28 09:30:00',
    '2025-10-28'
);
```

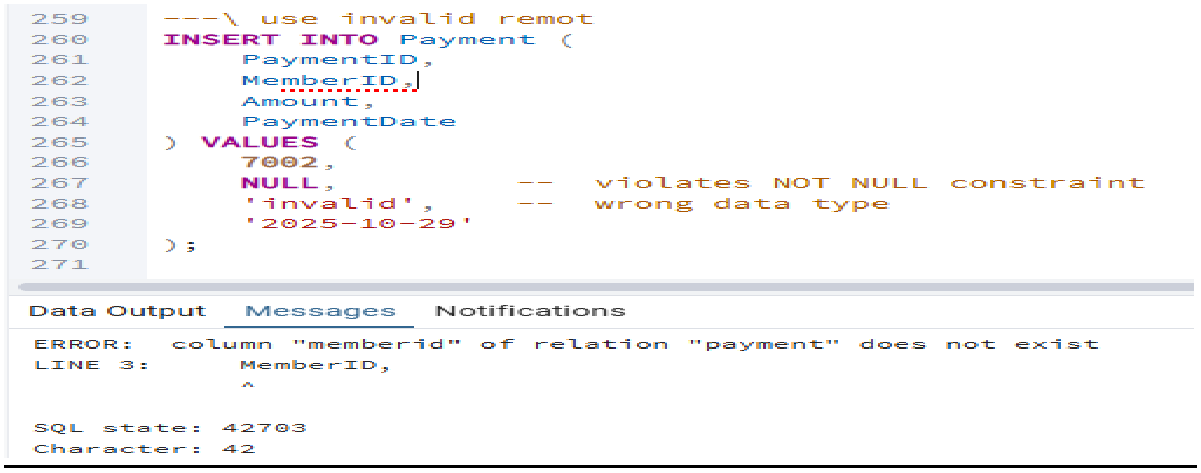
```
INSERT 0 1

Query returned successfully in 120 msec.
```



```
ROLLBACK;

COMMIT;
```

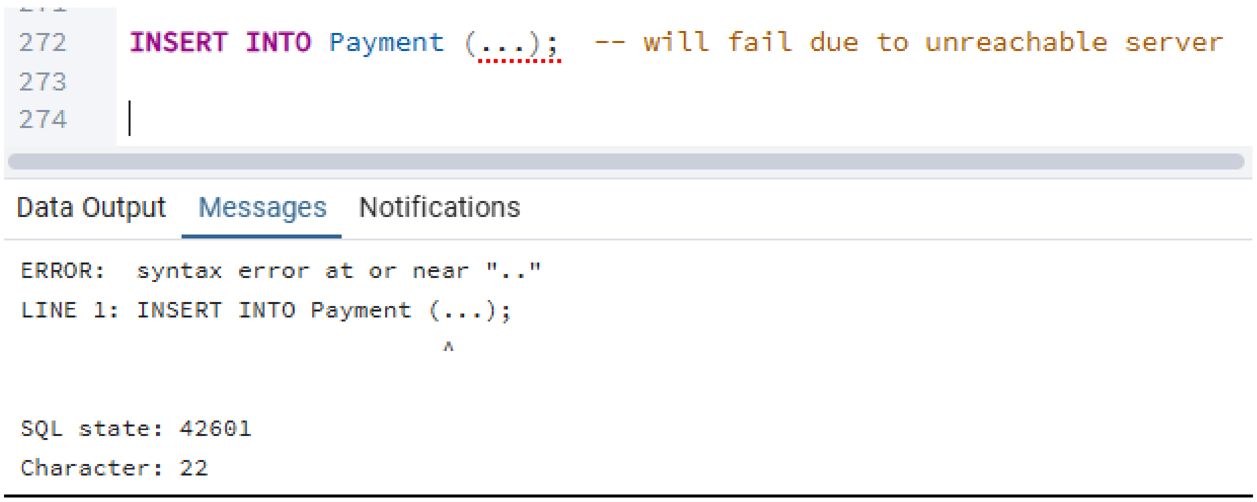
```
ROLLBACK

Query returned successfully in 177 msec.
```

A4.b: nduce a failure in a second run (e.g., disable the link between inserts) to create an in-doubt transaction; ensure any extra test rows are ROLLED BACK to keep within the ≤10 committed row budget.



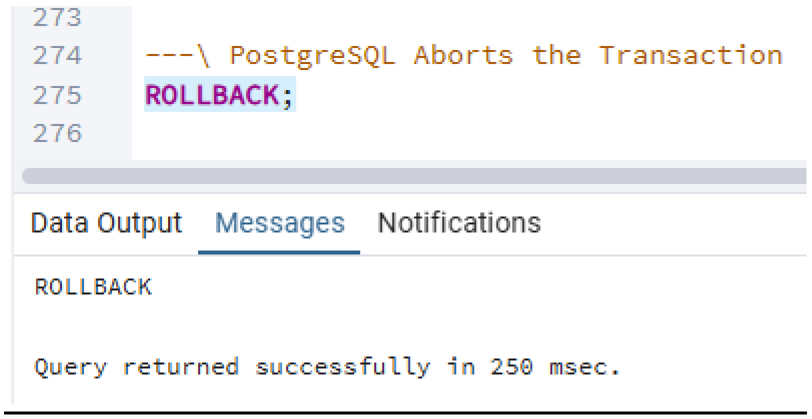
```
---\ use invalid remot
INSERT INTO Payment (
    PaymentID,
    MemberID,
    Amount,
    PaymentDate
) VALUES (
    7002,
    NULL,            -- violates NOT NULL constraint
    'invalid',       -- wrong data type
    '2025-10-29'
);
```

```
ERROR:  column "memberid" of relation "payment" does not exist
LINE 3:         MemberID,
                ^

SQL state: 42703
Character: 42
```



```
INSERT INTO Payment (...);  -- will fail due to unreachable server
```

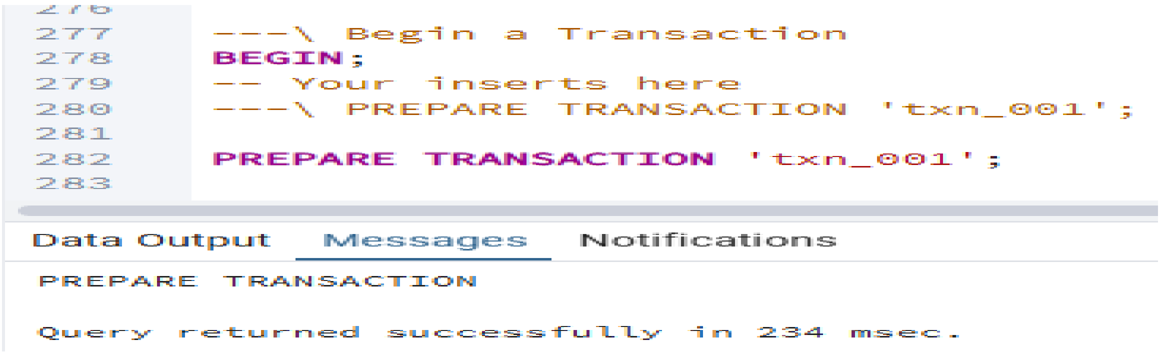
```
ERROR:  syntax error at or near ".."
LINE 1: INSERT INTO Payment (...);
                             ^


SQL state: 42601
Character: 22
```



```
---\ PostgreSQL Aborts the Transaction
ROLLBACK;
```

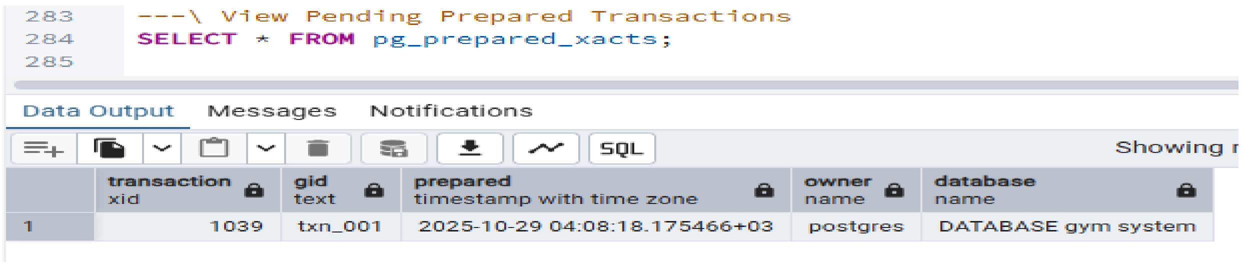
```
ROLLBACK

Query returned successfully in 250 msec.
```

A4.c: Query DBA_2PC_PENDING; then issue COMMIT FORCE or ROLLBACK FORCE; re-verify consistency on both nodes.



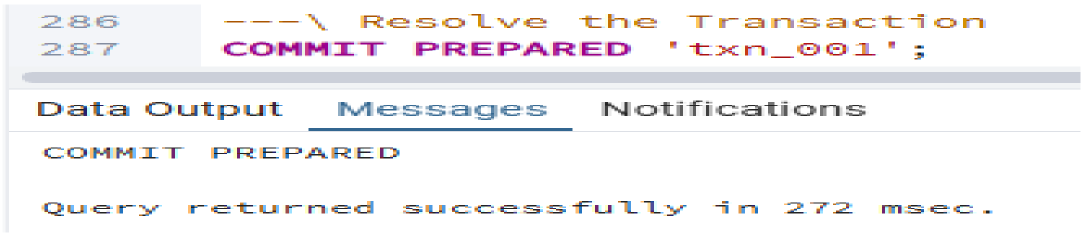
```
---\ Begin a Transaction
BEGIN;
-- Your inserts here
---\ PREPARE TRANSACTION 'txn_001';

PREPARE TRANSACTION 'txn_001';
```

```
PREPARE TRANSACTION

Query returned successfully in 234 msec.
```



```
---\ View Pending Prepared Transactions
SELECT * FROM pg_prepared_xacts;
```

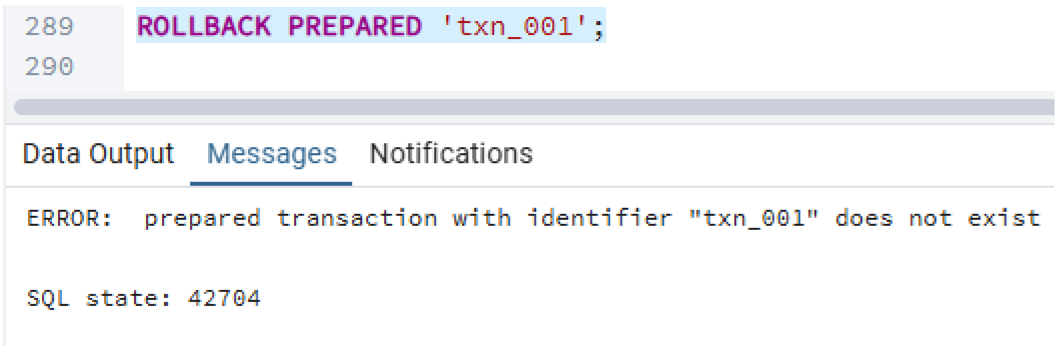
	transaction xid	gid text	prepared timestamp with time zone	owner name	database name
1	1039	txn_001	2025-10-29 04:08:18.175466+03	postgres	DATABASE gym system



```
---\ Resolve the Transaction
COMMIT PREPARED 'txn_001';
```

Data Output | Messages | Notifications

```
COMMIT PREPARED

Query returned successfully in 272 msec.
```

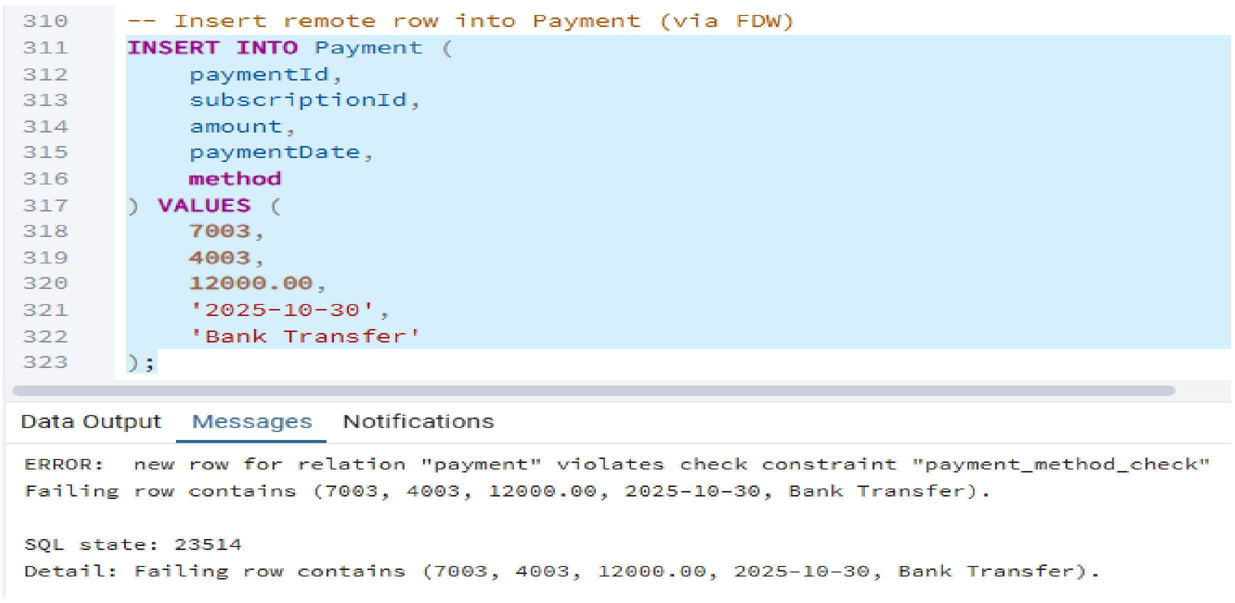


```
ROLLBACK PREPARED 'txn_001';
```

Data Output | Messages | Notifications

```
ERROR:  prepared transaction with identifier "txn_001" does not exist

SQL state: 42704
```

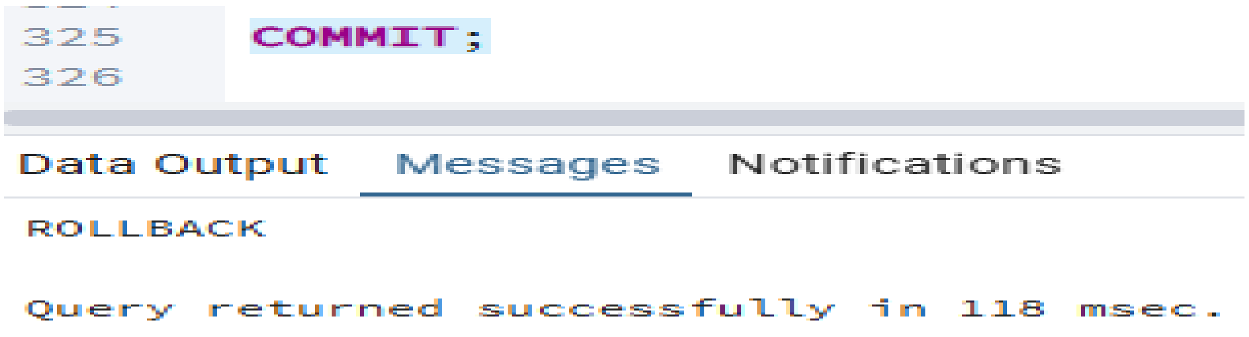
Repeat a clean run to show there are no pending transactions.



```
-- Insert remote row into Payment (via FDW)
INSERT INTO Payment (
    paymentId,
    subscriptionId,
    amount,
    paymentDate,
    method
) VALUES (
    7003,
    4003,
    12000.00,
    '2025-10-30',
    'Bank Transfer'
);
```

Data Output | Messages | Notifications

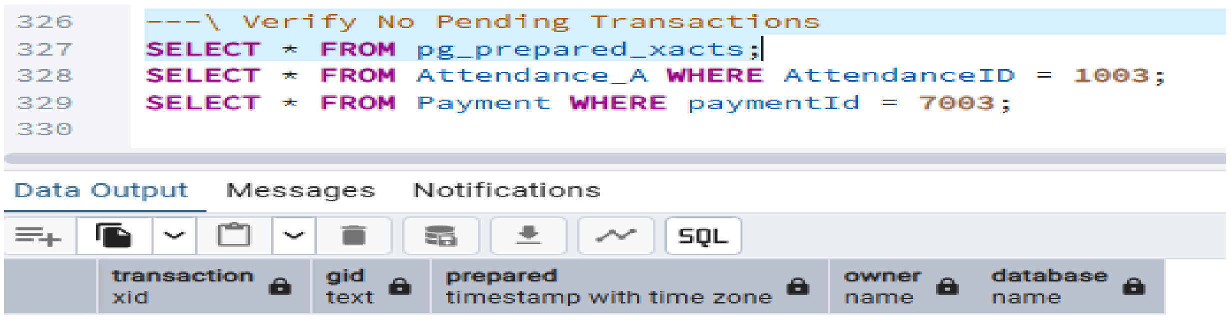
```
ERROR:  new row for relation "payment" violates check constraint "payment_method_check"
Failing row contains (7003, 4003, 12000.00, 2025-10-30, Bank Transfer).

SQL state: 23514
Detail: Failing row contains (7003, 4003, 12000.00, 2025-10-30, Bank Transfer).
```



```
COMMIT;
```

Data Output | Messages | Notifications

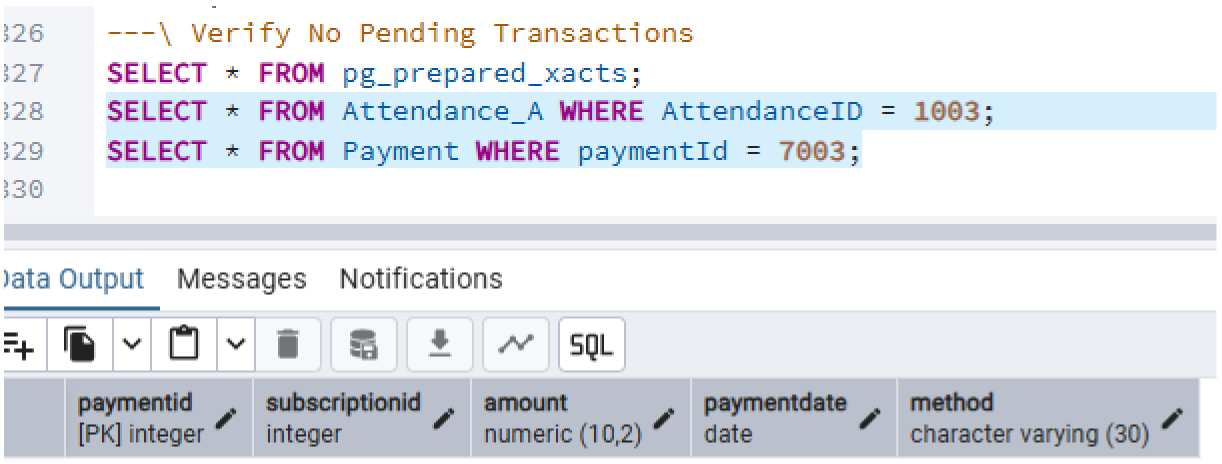
```
ROLLBACK

Query returned successfully in 118 msec.
```



```
---\ Verify No Pending Transactions
SELECT * FROM pg_prepared_xacts;
SELECT * FROM Attendance_A WHERE AttendanceID = 1003;
SELECT * FROM Payment WHERE paymentId = 7003;
```

Data Output | Messages | Notifications

transaction xid	gid text	prepared timestamp with time zone	owner name	database name

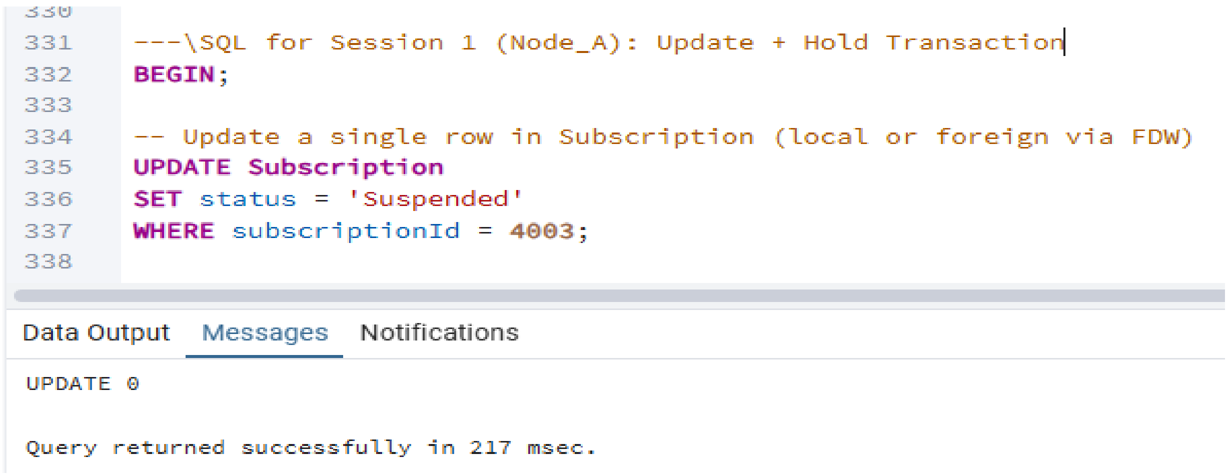


```
---\ Verify No Pending Transactions
SELECT * FROM pg_prepared_xacts;
SELECT * FROM Attendance_A WHERE AttendanceID = 1003;
SELECT * FROM Payment WHERE paymentId = 7003;
```

Data Output | Messages | Notifications

paymentid [PK] integer	subscriptionid integer	amount numeric (10,2)	paymentdate date	method character varying (30)

A5 a:Distributed Lock Conflict & Diagnosis (no extra rows)

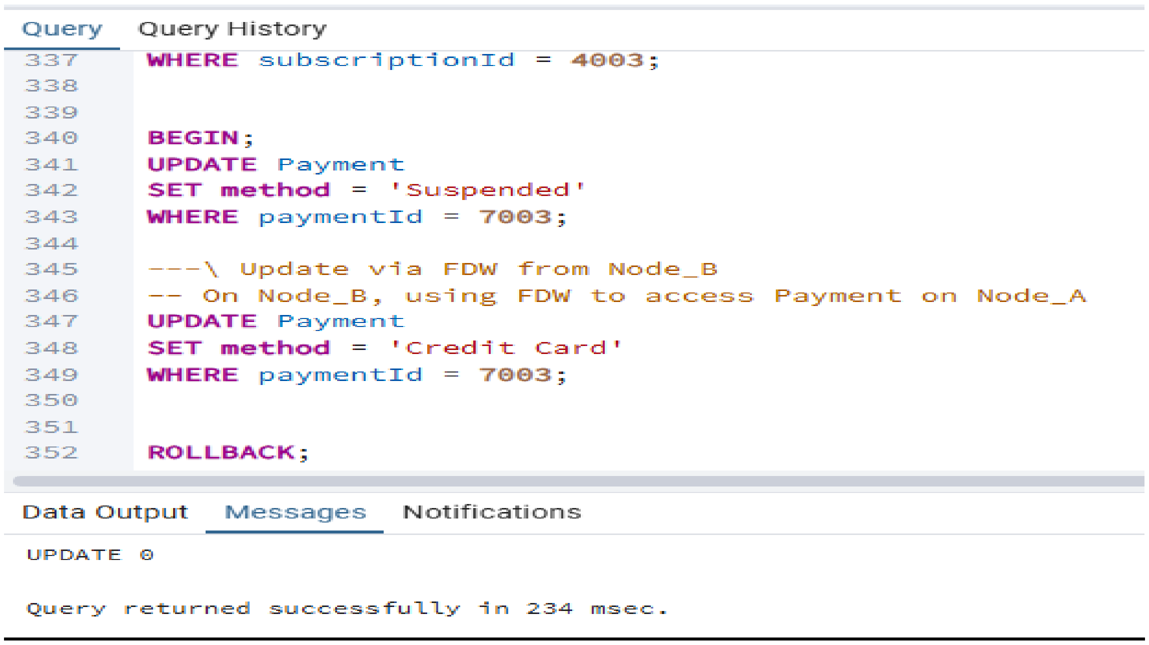


```
---\SQL for Session 1 (Node_A): Update + Hold Transaction
BEGIN;

-- Update a single row in Subscription (local or foreign via FDW)
UPDATE Subscription
SET status = 'Suspended'
WHERE subscriptionId = 4003;
```

Data Output | Messages | Notifications

```
UPDATE 0

Query returned successfully in 217 msec.
```

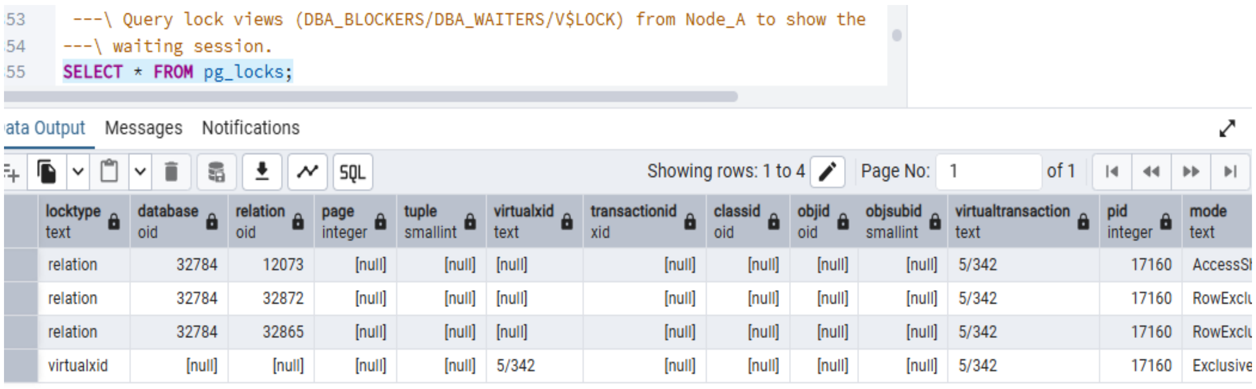
A5.b: Open Session 2 from Node_B via Subscription@proj_link or Payment@proj_link to UPDATE the same logical row.



```
WHERE subscriptionId = 4003;


BEGIN;
UPDATE Payment
SET method = 'Suspended'
WHERE paymentId = 7003;

---\ Update via FDW from Node_B
-- On Node_B, using FDW to access Payment on Node_A
UPDATE Payment
SET method = 'Credit Card'
WHERE paymentId = 7003;


ROLLBACK;
```

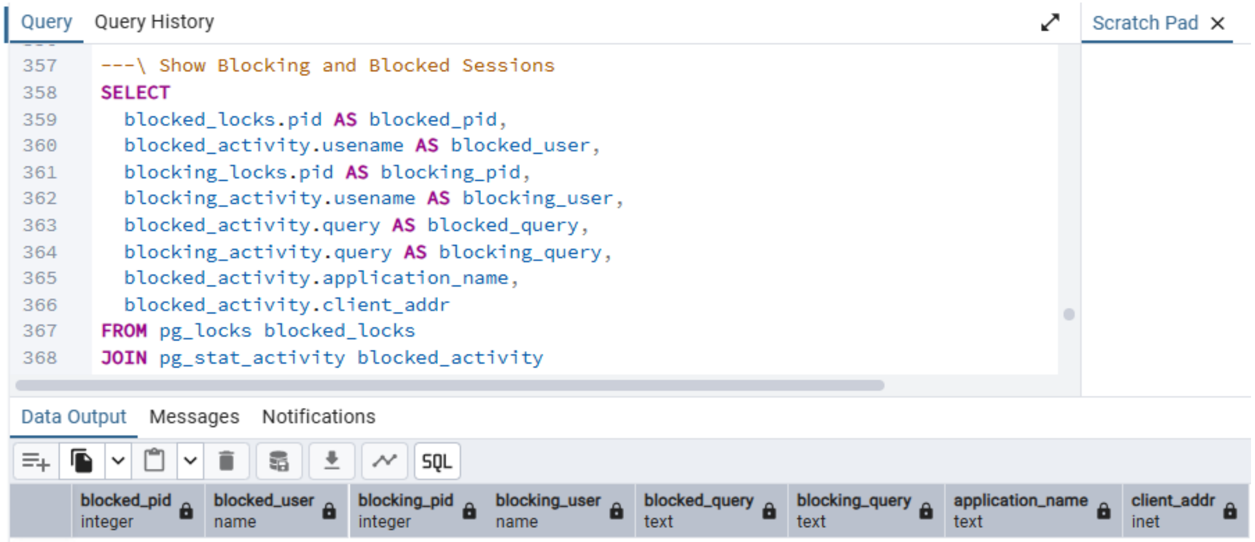
```
UPDATE 0

Query returned successfully in 234 msec.
```

A5.C: Query lock views (DBA_BLOCKERS/DBA_WAITERS/V$LOCK) from Node_A to show the waiting session.



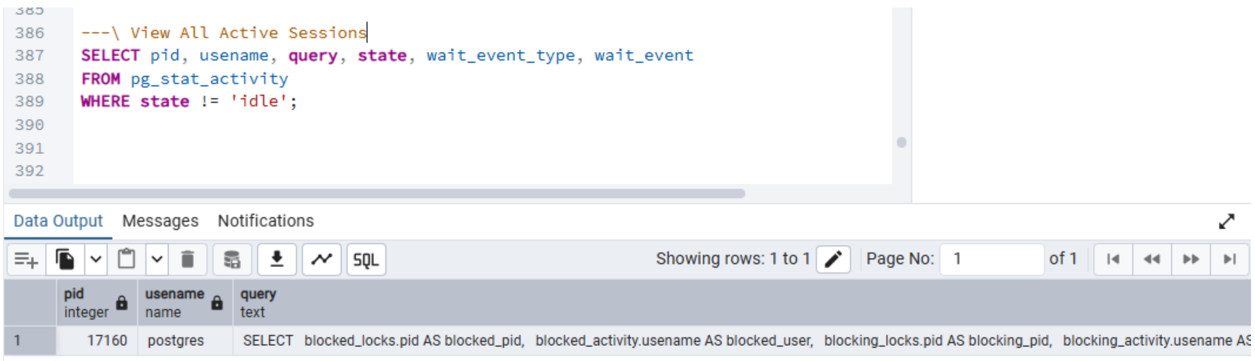
```
 ---\ Query lock views (DBA_BLOCKERS/DBA_WAITERS/V$LOCK) from Node_A to show the
---\ waiting session.
SELECT * FROM pg_locks;
```

locktype text	database oid	relation oid	page integer	tuple smallint	virtualxid text	transactionid xid	classid oid	objid oid	objsubid smallint	virtualtransaction text	pid integer	mode text
relation	32784	12073	[null]	[null]	[null]	[null]	[null]	[null]	[null]	5/342	17160	AccessSh
relation	32784	32872	[null]	[null]	[null]	[null]	[null]	[null]	[null]	5/342	17160	RowExclu
relation	32784	32865	[null]	[null]	[null]	[null]	[null]	[null]	[null]	5/342	17160	RowExclu
virtualxid	[null]	[null]	[null]	[null]	5/342	[null]	[null]	[null]	[null]	5/342	17160	Exclusive



```sql
---\ Show Blocking and Blocked Sessions
SELECT
  blocked_locks.pid AS blocked_pid,
  blocked_activity.usename AS blocked_user,
  blocking_locks.pid AS blocking_pid,
  blocking_activity.usename AS blocking_user,
  blocked_activity.query AS blocked_query,
  blocking_activity.query AS blocking_query,
  blocked_activity.application_name,
  blocked_activity.client_addr
FROM pg_locks blocked_locks
JOIN pg_stat_activity blocked_activity
```

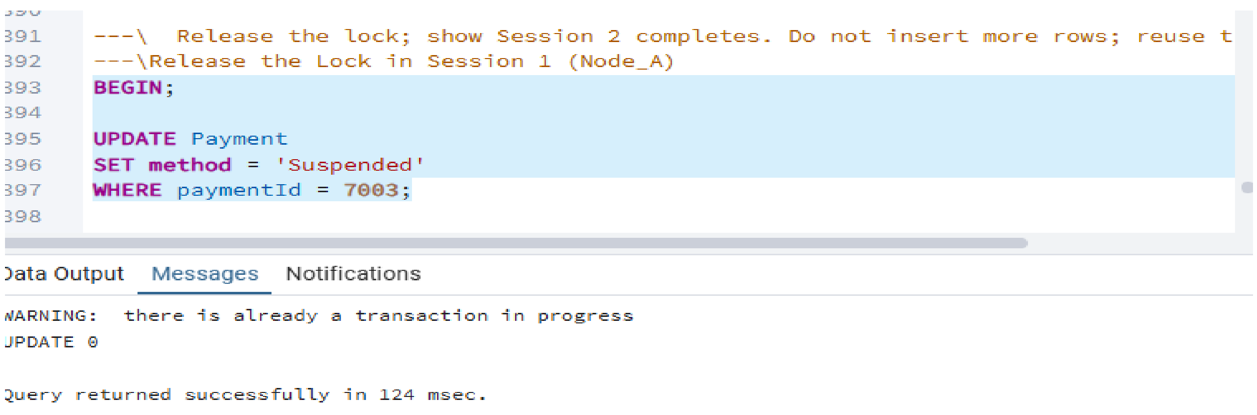
	blocked_pid integer	blocked_user name	blocking_pid integer	blocking_user name	blocked_query text	blocking_query text	application_name text	client_addr inet



```sql
---\ View All Active Sessions
SELECT pid, usename, query, state, wait_event_type, wait_event
FROM pg_stat_activity
WHERE state != 'idle';
```

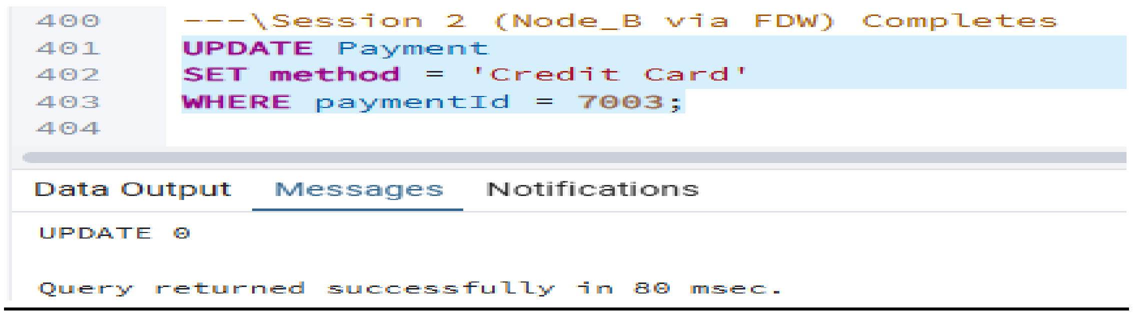
	pid integer	usename name	query text
1	17160	postgres	SELECT blocked_locks.pid AS blocked_pid, blocked_activity.usename AS blocked_user, blocking_locks.pid AS blocking_pid, blocking_activity.usename AS

A5.D: Release the lock; show Session 2 completes. Do not insert more rows; reuse the existing ≤10.



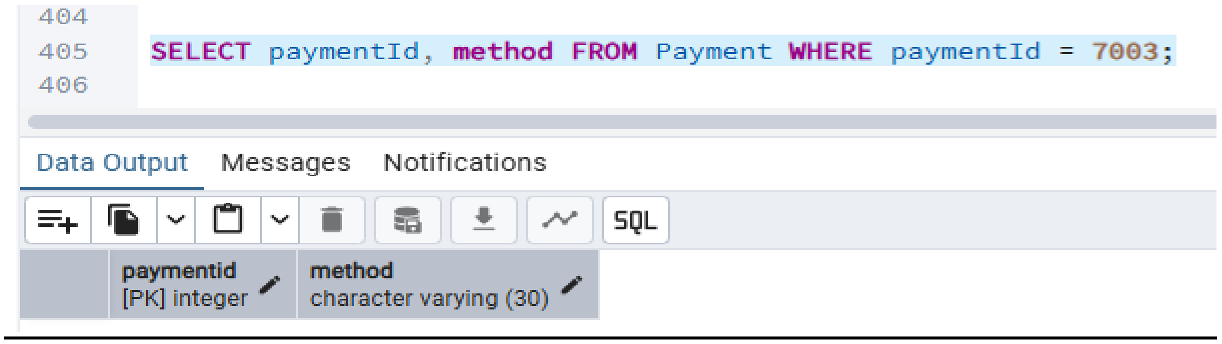
```sql
---\  Release the lock; show Session 2 completes. Do not insert more rows; reuse t
---\Release the Lock in Session 1 (Node_A)
BEGIN;

UPDATE Payment
SET method = 'Suspended'
WHERE paymentId = 7003;
```

```
WARNING:  there is already a transaction in progress
UPDATE 0

Query returned successfully in 124 msec.
```



```
---\Session 2 (Node_B via FDW) Completes
UPDATE Payment
SET method = 'Credit Card'
WHERE paymentId = 7003;
```

```
UPDATE 0

Query returned successfully in 80 msec.
```

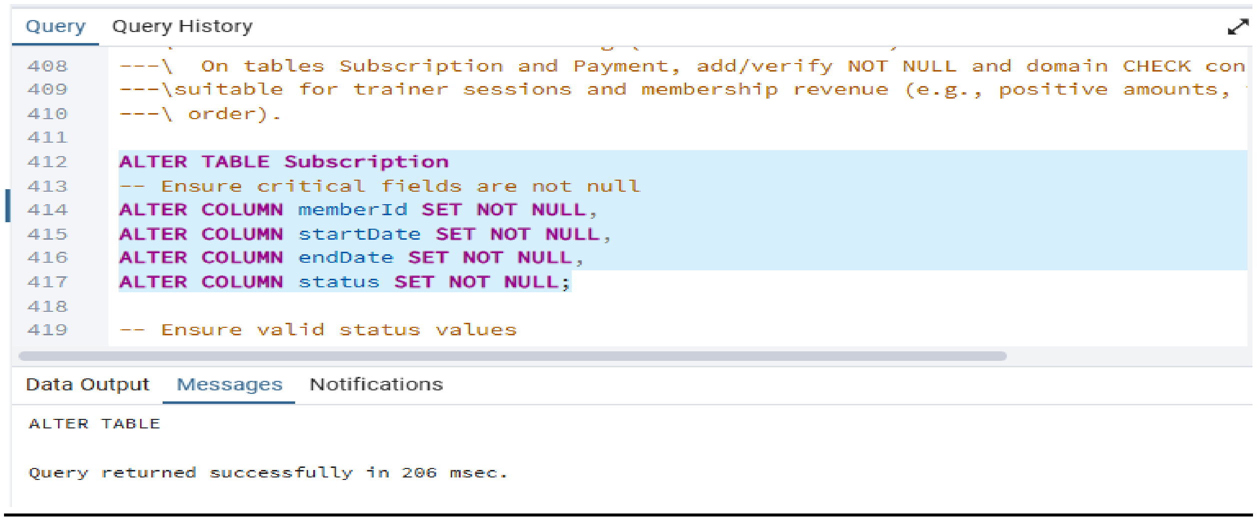


```
SELECT paymentId, method FROM Payment WHERE paymentId = 7003;
```

paymentid [PK] integer	method character varying (30)

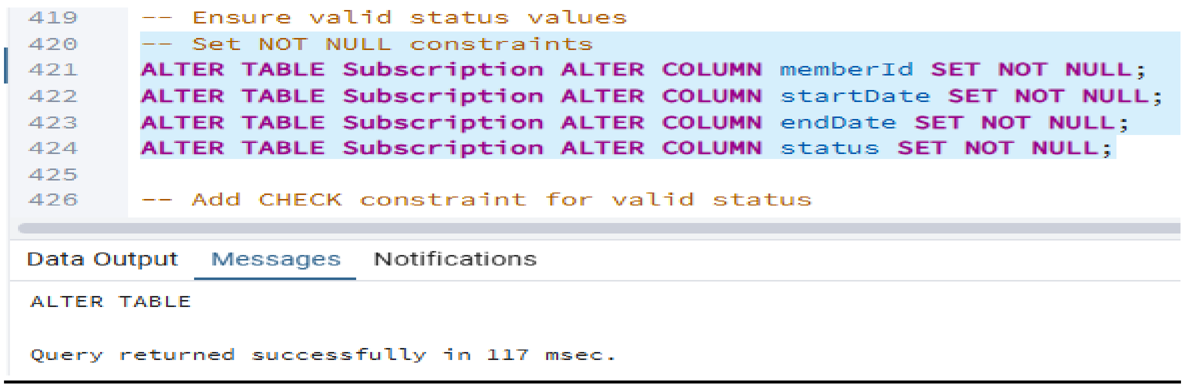
B6 :Declarative Rules Hardening (≤10 committed rows)

B6.a: On tables Subscription and Payment, add/verify NOT NULL and domain CHECK constraints suitable for trainer sessions and membership revenue (e.g., positive amounts, valid statuses, date order).



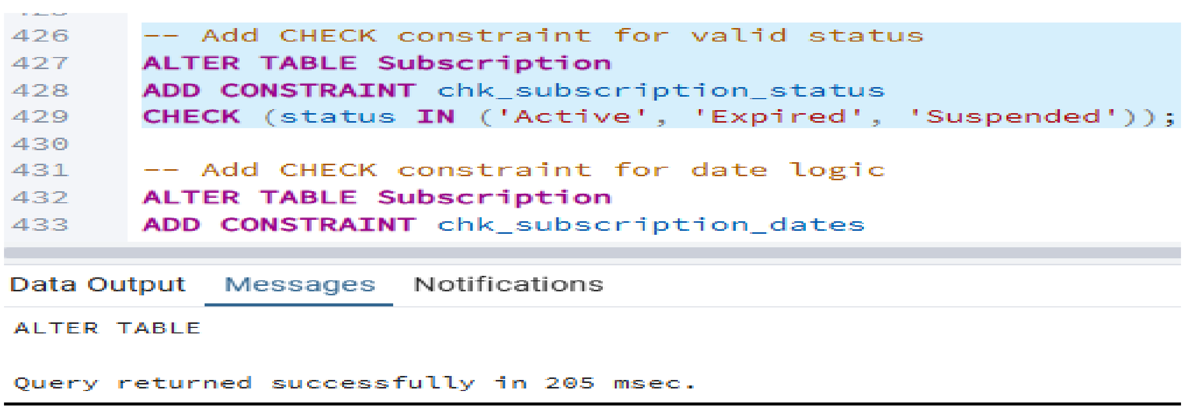
```
---\  On tables Subscription and Payment, add/verify NOT NULL and domain CHECK con
---\suitable for trainer sessions and membership revenue (e.g., positive amounts,
---\ order).

ALTER TABLE Subscription
-- Ensure critical fields are not null
ALTER COLUMN memberId SET NOT NULL,
ALTER COLUMN startDate SET NOT NULL,
ALTER COLUMN endDate SET NOT NULL,
ALTER COLUMN status SET NOT NULL;

-- Ensure valid status values
```

```
ALTER TABLE

Query returned successfully in 206 msec.
```



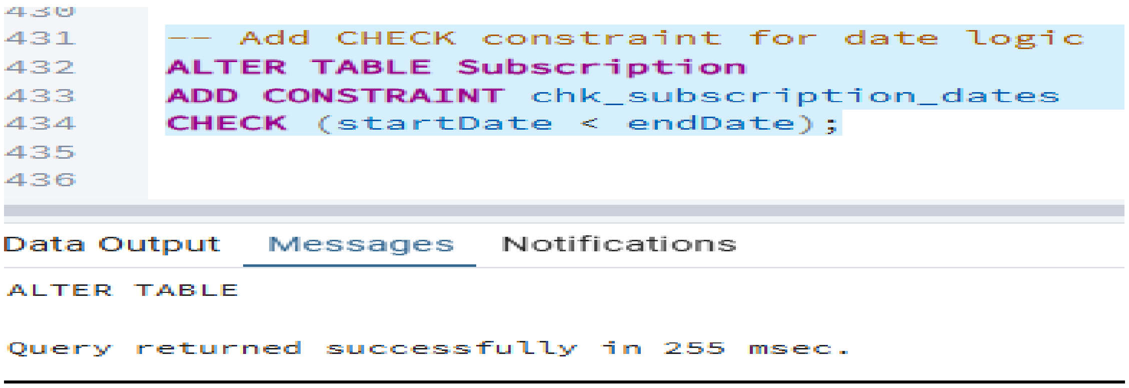
```sql
-- Ensure valid status values
-- Set NOT NULL constraints
ALTER TABLE Subscription ALTER COLUMN memberId SET NOT NULL;
ALTER TABLE Subscription ALTER COLUMN startDate SET NOT NULL;
ALTER TABLE Subscription ALTER COLUMN endDate SET NOT NULL;
ALTER TABLE Subscription ALTER COLUMN status SET NOT NULL;

-- Add CHECK constraint for valid status
```

```
ALTER TABLE

Query returned successfully in 117 msec.
```



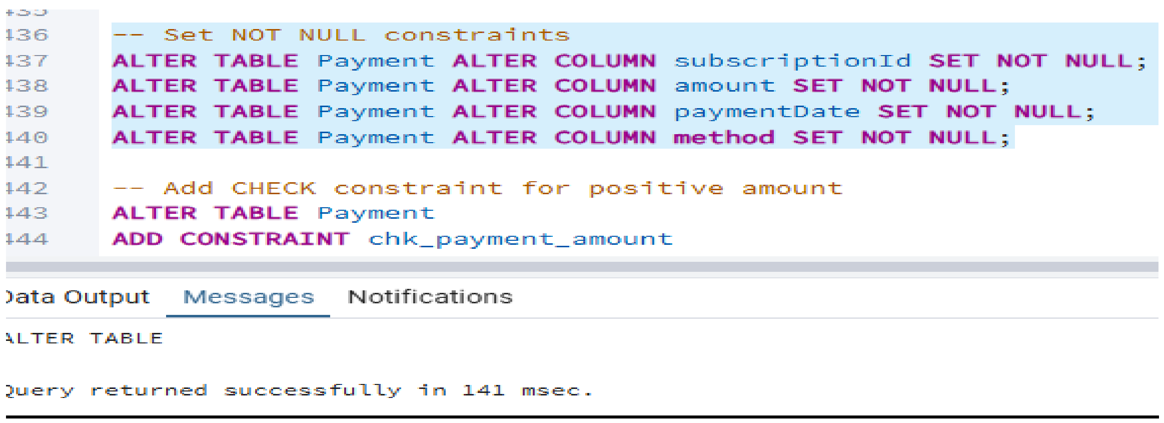
```sql
-- Add CHECK constraint for valid status
ALTER TABLE Subscription
ADD CONSTRAINT chk_subscription_status
CHECK (status IN ('Active', 'Expired', 'Suspended'));

-- Add CHECK constraint for date logic
ALTER TABLE Subscription
ADD CONSTRAINT chk_subscription_dates
```

```
ALTER TABLE

Query returned successfully in 205 msec.
```



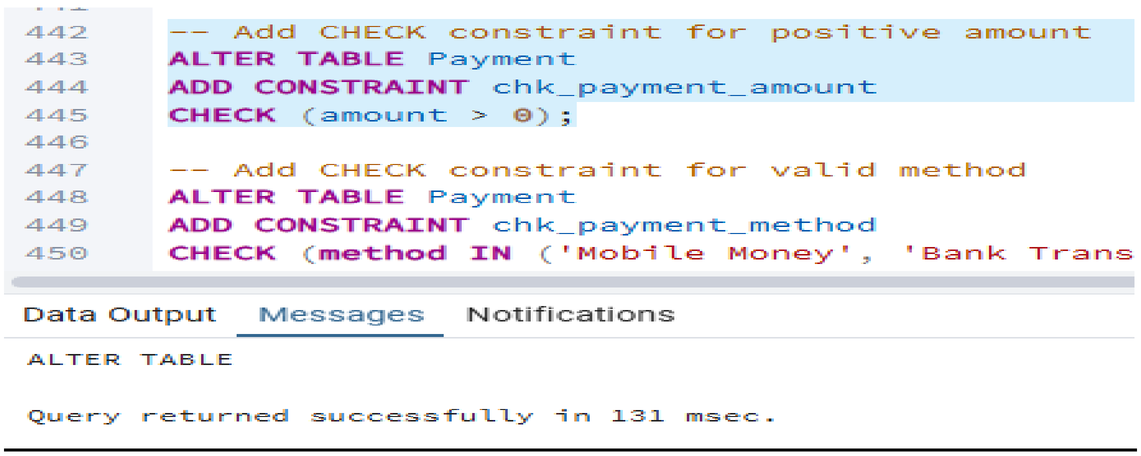
```sql
-- Add CHECK constraint for date logic
ALTER TABLE Subscription
ADD CONSTRAINT chk_subscription_dates
CHECK (startDate < endDate);
```

```
ALTER TABLE

Query returned successfully in 255 msec.
```



```sql
-- Set NOT NULL constraints
ALTER TABLE Payment ALTER COLUMN subscriptionId SET NOT NULL;
ALTER TABLE Payment ALTER COLUMN amount SET NOT NULL;
ALTER TABLE Payment ALTER COLUMN paymentDate SET NOT NULL;
ALTER TABLE Payment ALTER COLUMN method SET NOT NULL;

-- Add CHECK constraint for positive amount
ALTER TABLE Payment
ADD CONSTRAINT chk_payment_amount
```

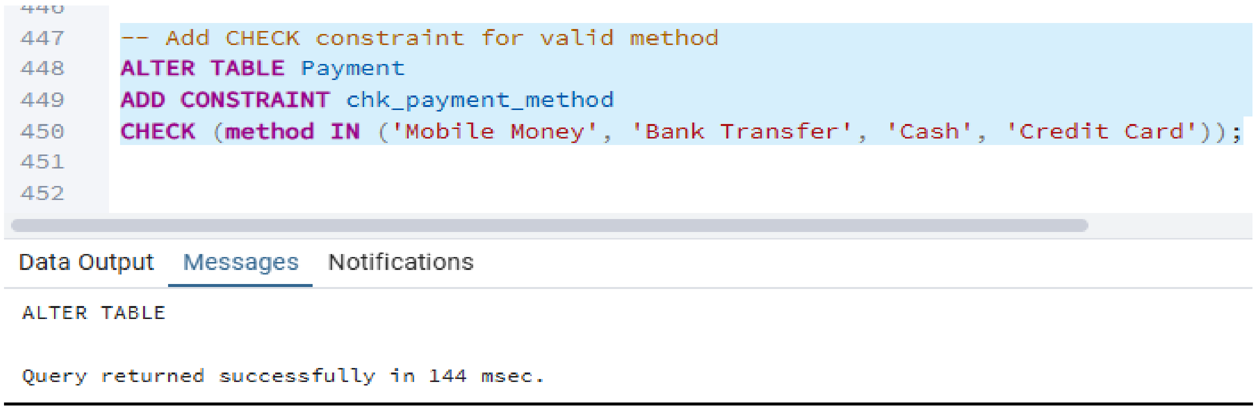
```
LTER TABLE

uery returned successfully in 141 msec.
```



```sql
-- Add CHECK constraint for positive amount
ALTER TABLE Payment
ADD CONSTRAINT chk_payment_amount
CHECK (amount > 0);

-- Add CHECK constraint for valid method
ALTER TABLE Payment
ADD CONSTRAINT chk_payment_method
CHECK (method IN ('Mobile Money', 'Bank Trans
```

Data Output | Messages | Notifications

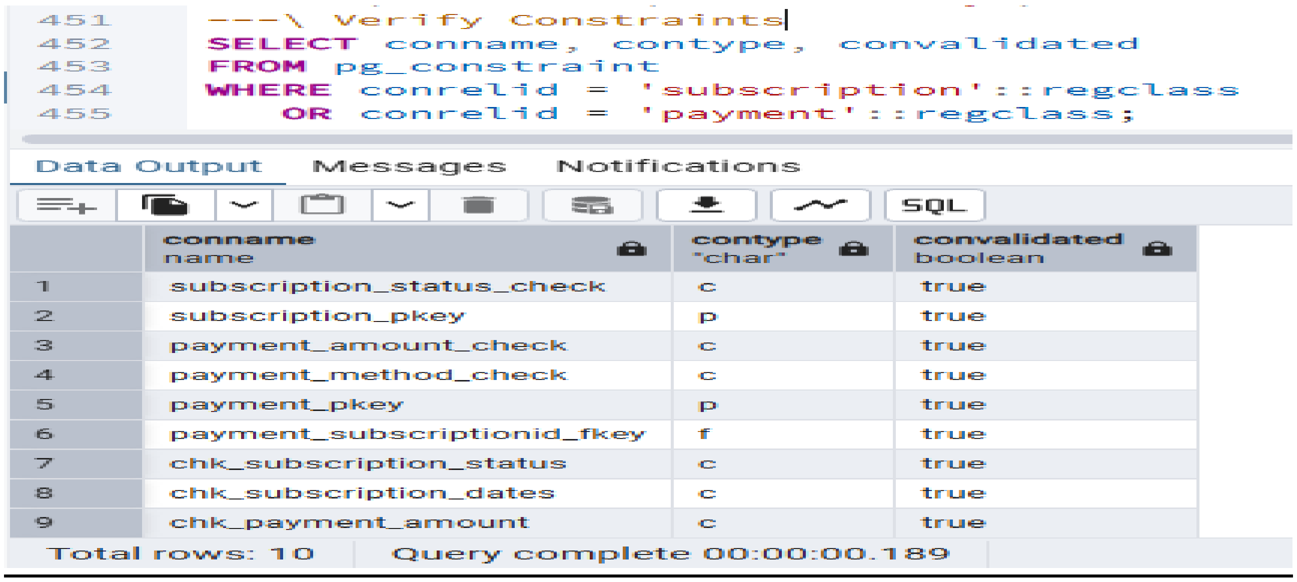
```
ALTER TABLE

Query returned successfully in 131 msec.
```



```sql
-- Add CHECK constraint for valid method
ALTER TABLE Payment
ADD CONSTRAINT chk_payment_method
CHECK (method IN ('Mobile Money', 'Bank Transfer', 'Cash', 'Credit Card'));
```

Data Output | Messages | Notifications

```
ALTER TABLE

Query returned successfully in 144 msec.
```



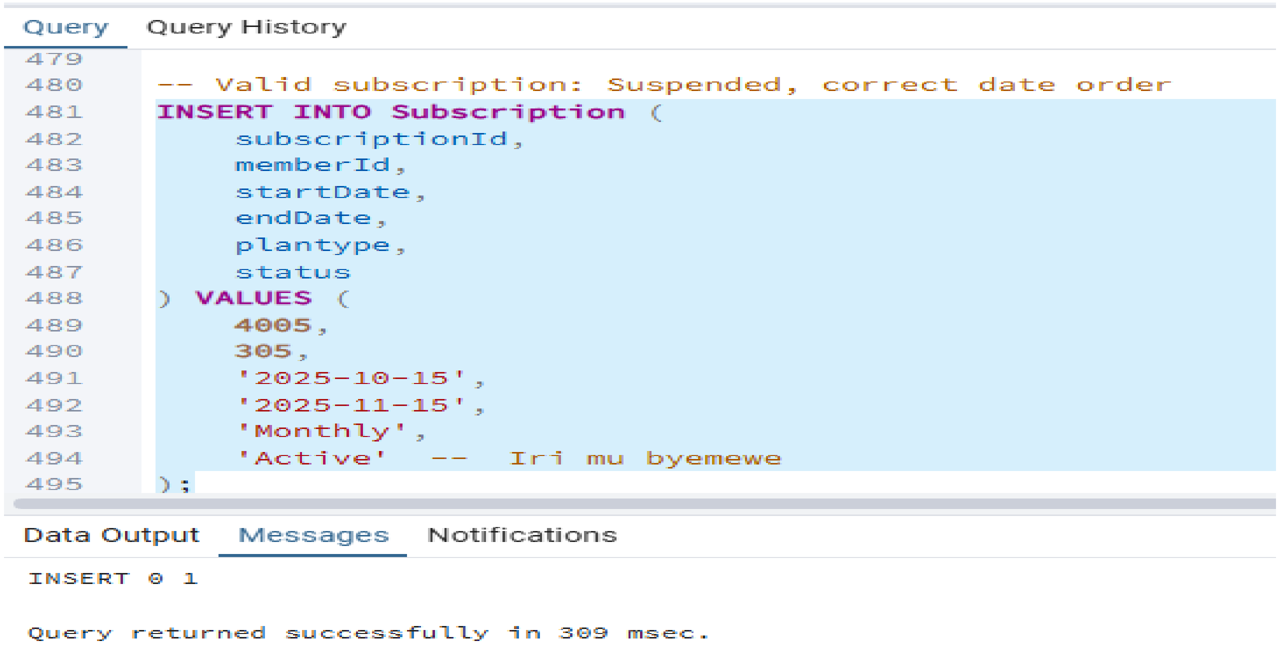
```sql
---\ Verify Constraints
SELECT conname, contype, convalidated
FROM pg_constraint
WHERE conrelid = 'subscription'::regclass
   OR conrelid = 'payment'::regclass;
```

Data Output | Messages | Notifications

	conname name	contype "char"	convalidated boolean
1	subscription_status_check	c	true
2	subscription_pkey	p	true
3	payment_amount_check	c	true
4	payment_method_check	c	true
5	payment_pkey	p	true
6	payment_subscriptionid_fkey	f	true
7	chk_subscription_status	c	true
8	chk_subscription_dates	c	true
9	chk_payment_amount	c	true

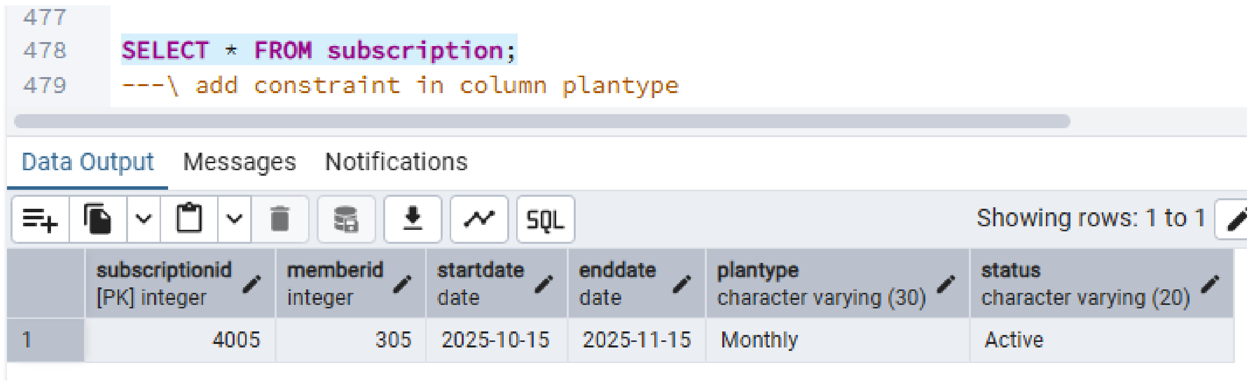
Total rows: 10   Query complete 00:00:00.189

B6.C: Prepare 2 failing and 2 passing INSERTs per table to validate rules, but wrap failing ones in a block and ROLLBACK so committed rows stay within ≤10 total.


```
-- Valid subscription: Suspended, correct date order
INSERT INTO Subscription (
    subscriptionId,
    memberId,
    startDate,
    endDate,
    plantype,
    status
) VALUES (
    4005,
    305,
    '2025-10-15',
    '2025-11-15',
    'Monthly',
    'Active'  --  Iri mu byemewe
);
```

```
INSERT 0 1

Query returned successfully in 309 msec.
```

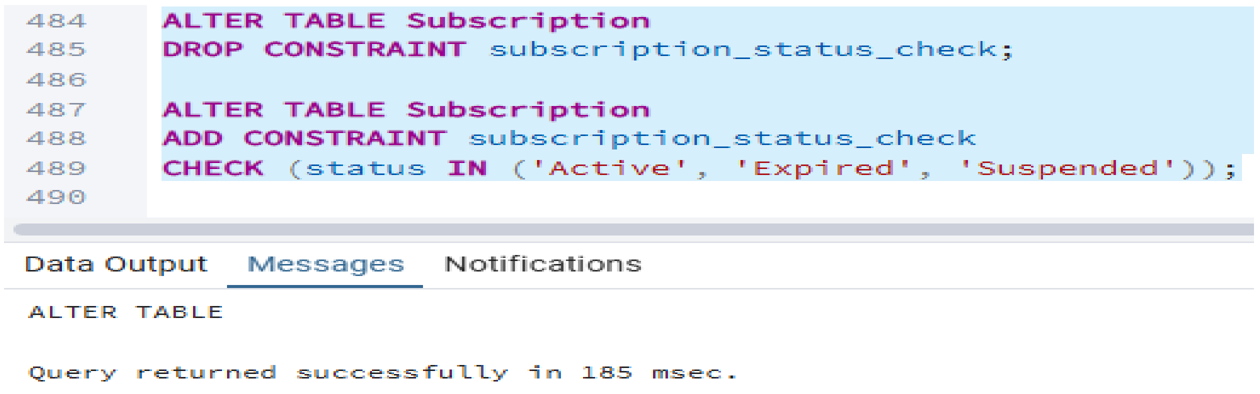
Check subscription


```
SELECT * FROM subscription;
---\ add constraint in column plantype
```

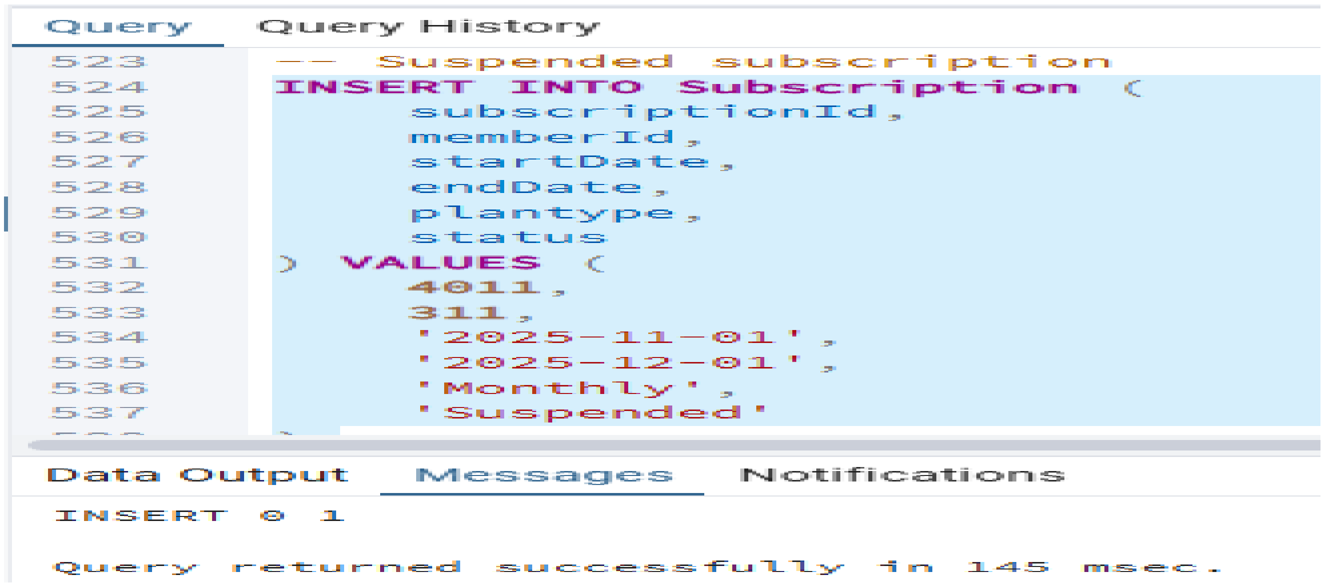
Showing rows: 1 to 1

	subscriptionid [PK] integer	memberid integer	startdate date	enddate date	plantype character varying (30)	status character varying (20)
1	4005	305	2025-10-15	2025-11-15	Monthly	Active

Add suspended in suscription


```
ALTER TABLE Subscription
DROP CONSTRAINT subscription_status_check;

ALTER TABLE Subscription
ADD CONSTRAINT subscription_status_check
CHECK (status IN ('Active', 'Expired', 'Suspended'));
```

```
ALTER TABLE

Query returned successfully in 185 msec.
```

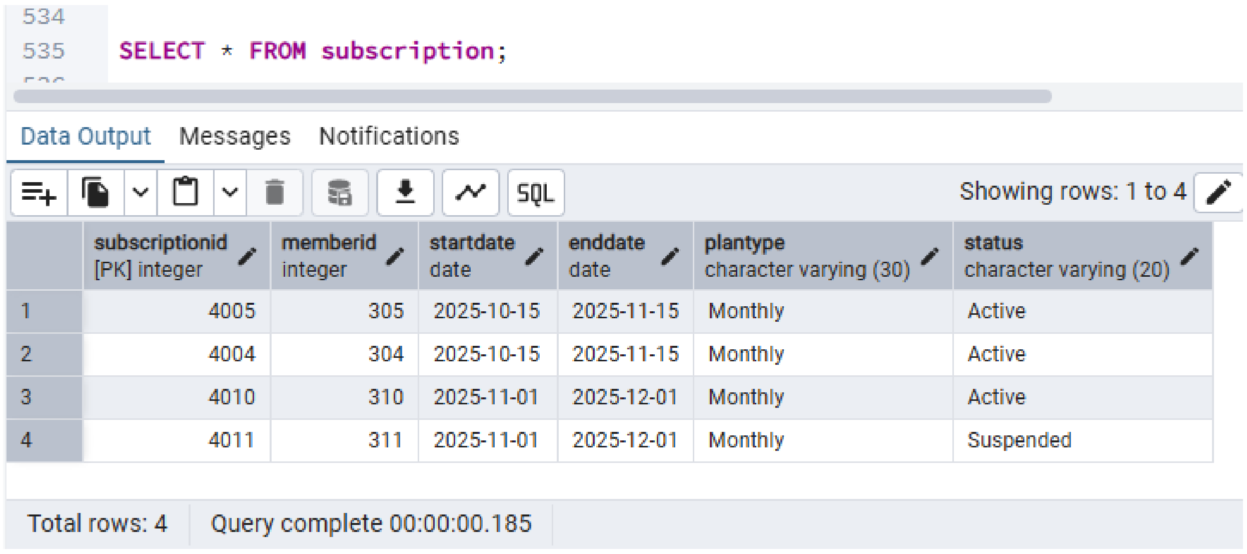


Query | Query History

```
-- Suspended subscription
INSERT INTO Subscription (
        subscriptionId,
        memberId,
        startDate,
        endDate,
        plantype,
        status
) VALUES (
        4011,
        311,
        '2025-11-01',
        '2025-12-01',
        'Monthly',
        'Suspended'
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 145 msec.
```



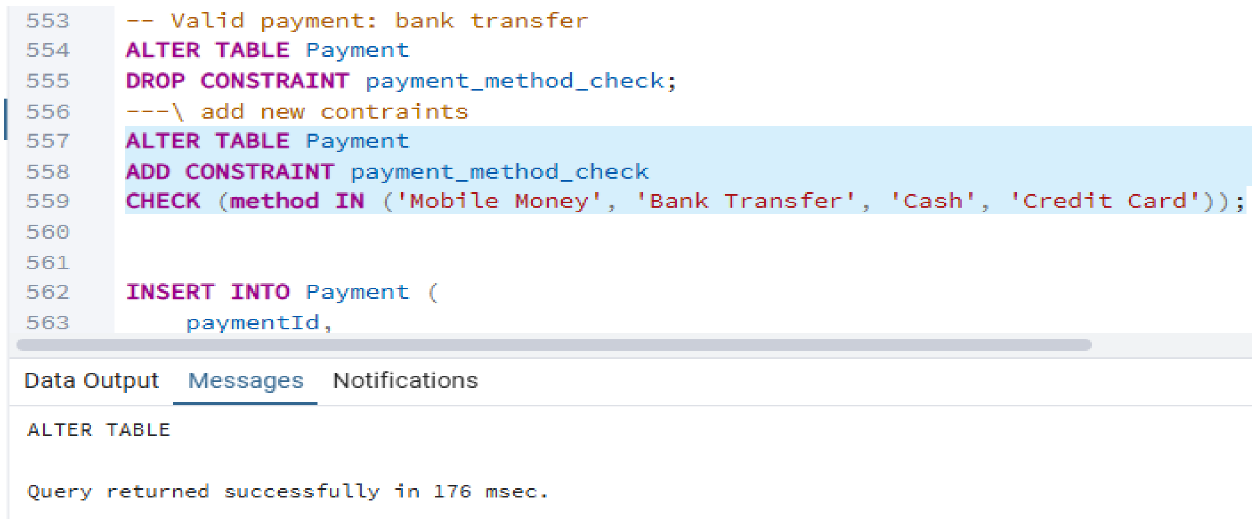
```
SELECT * FROM subscription;
```

Data Output | Messages | Notifications

Showing rows: 1 to 4

	subscriptionid [PK] integer	memberid integer	startdate date	enddate date	plantype character varying (30)	status character varying (20)
1	4005	305	2025-10-15	2025-11-15	Monthly	Active
2	4004	304	2025-10-15	2025-11-15	Monthly	Active
3	4010	310	2025-11-01	2025-12-01	Monthly	Active
4	4011	311	2025-11-01	2025-12-01	Monthly	Suspended

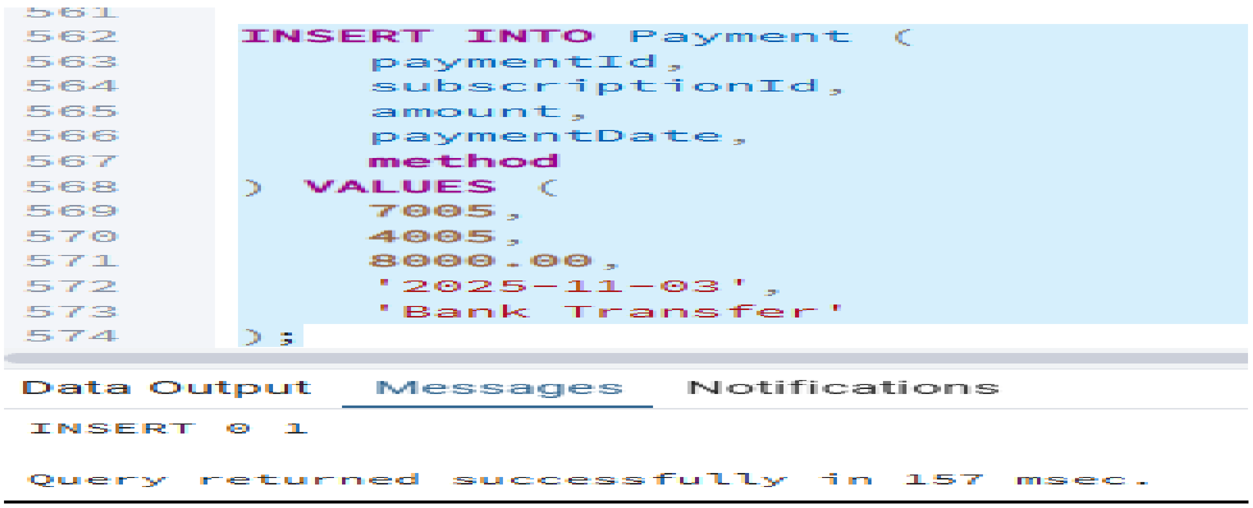
Total rows: 4 | Query complete 00:00:00.185



```
-- Valid payment: bank transfer
ALTER TABLE Payment
DROP CONSTRAINT payment_method_check;
---\ add new contraints
ALTER TABLE Payment
ADD CONSTRAINT payment_method_check
CHECK (method IN ('Mobile Money', 'Bank Transfer', 'Cash', 'Credit Card'));


INSERT INTO Payment (
    paymentId,
```

Data Output | Messages | Notifications

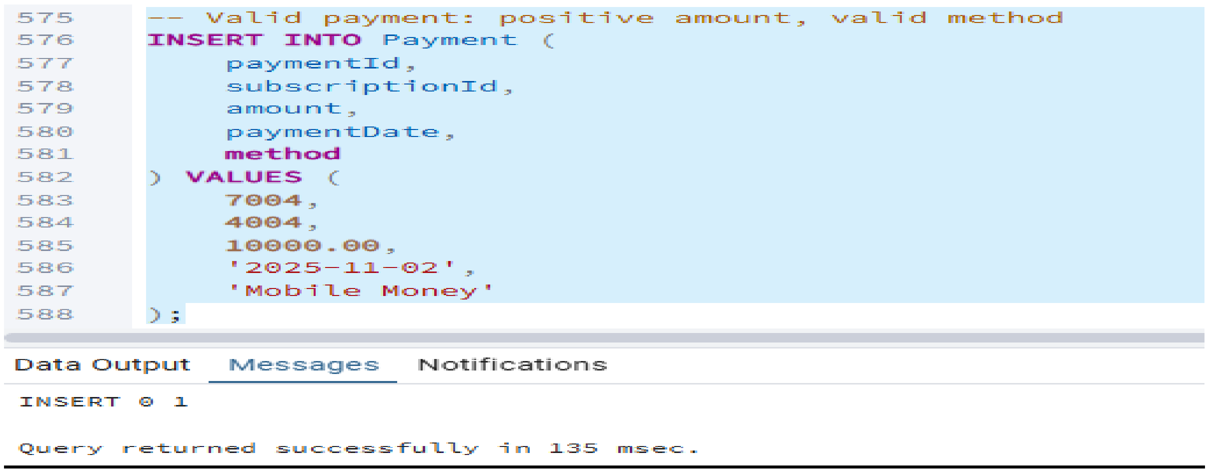
```
ALTER TABLE

Query returned successfully in 176 msec.
```



```sql
INSERT INTO Payment (
        paymentId,
        subscriptionId,
        amount,
        paymentDate,
        method
) VALUES (
        7005,
        4005,
        8000.00,
        '2025-11-03',
        'Bank Transfer'
);
```

Data Output   Messages   Notifications

```
INSERT 0 1

Query returned successfully in 157 msec.
```

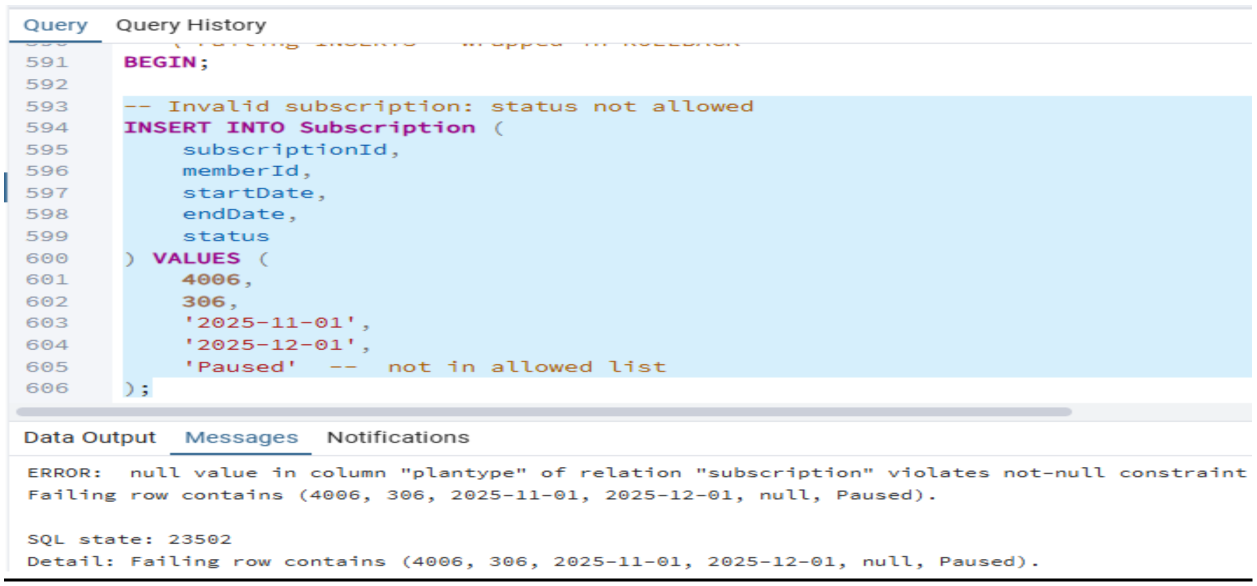


```sql
-- Valid payment: positive amount, valid method
INSERT INTO Payment (
        paymentId,
        subscriptionId,
        amount,
        paymentDate,
        method
) VALUES (
        7004,
        4004,
        10000.00,
        '2025-11-02',
        'Mobile Money'
);
```

Data Output   Messages   Notifications

```
INSERT 0 1

Query returned successfully in 135 msec.
```

<u>Failing INSERTs — Wrapped in ROLLBACK</u>

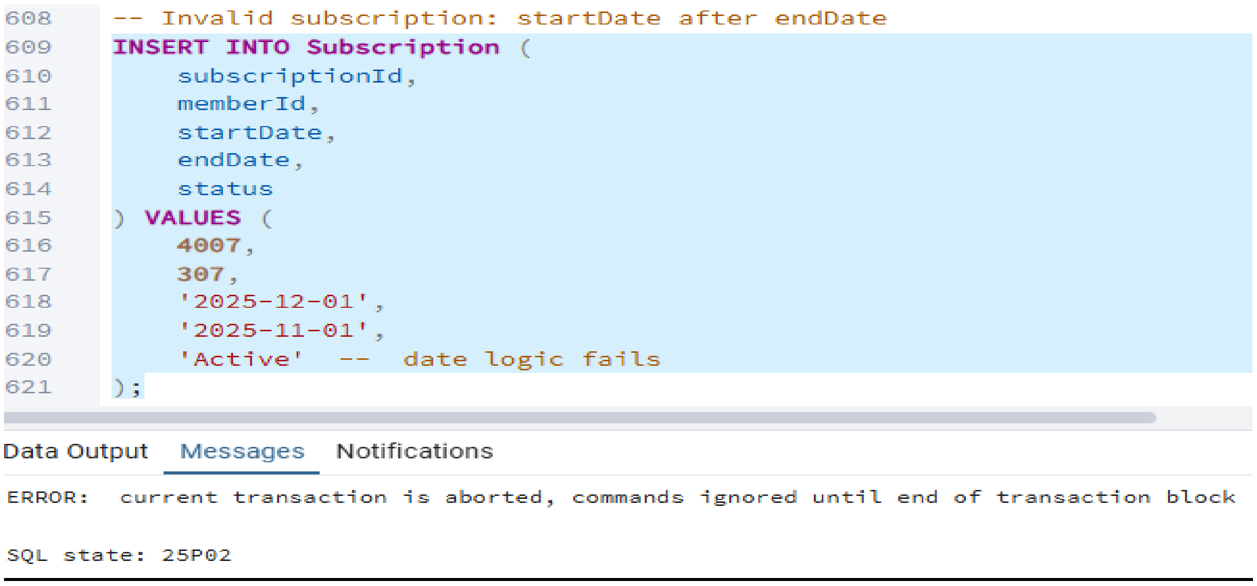


Query   Query History

```sql
BEGIN;

-- Invalid subscription: status not allowed
INSERT INTO Subscription (
    subscriptionId,
    memberId,
    startDate,
    endDate,
    status
) VALUES (
    4006,
    306,
    '2025-11-01',
    '2025-12-01',
    'Paused'  --  not in allowed list
);
```

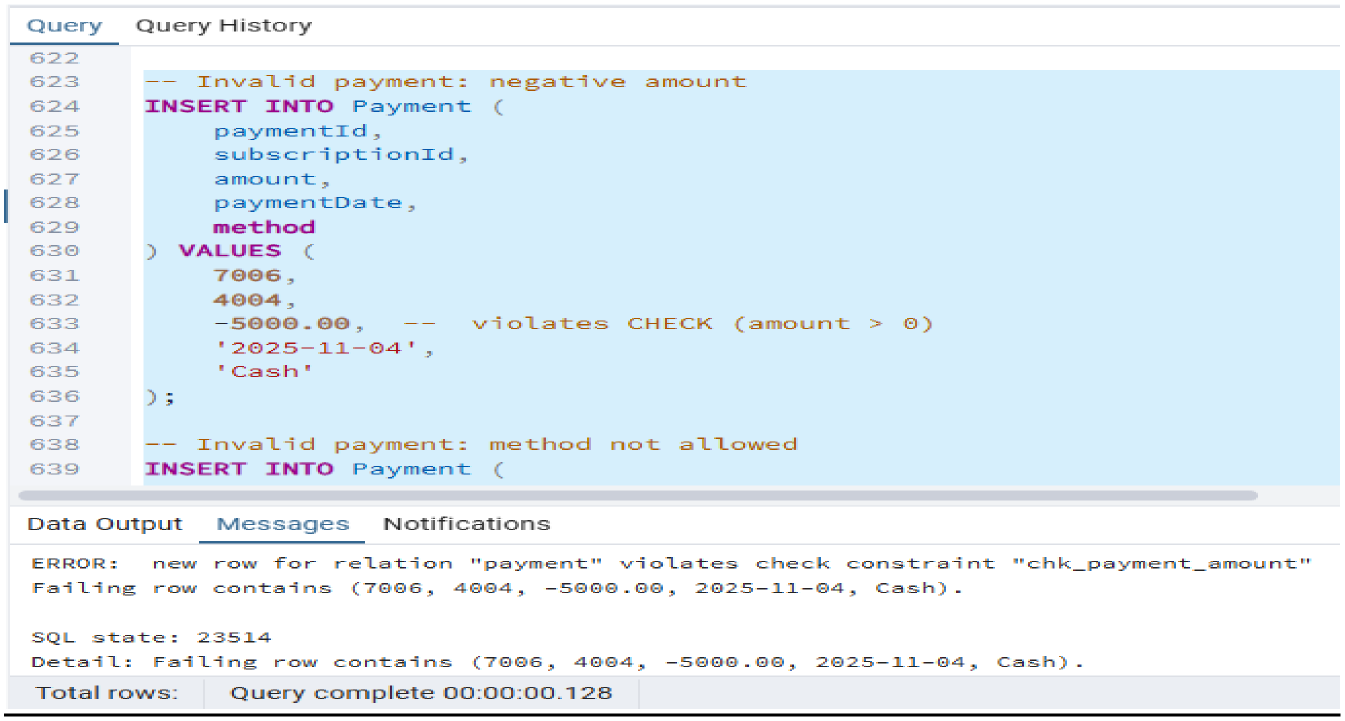
Data Output   Messages   Notifications

```
ERROR:  null value in column "plantype" of relation "subscription" violates not-null constraint
Failing row contains (4006, 306, 2025-11-01, 2025-12-01, null, Paused).

SQL state: 23502
Detail: Failing row contains (4006, 306, 2025-11-01, 2025-12-01, null, Paused).
```



```sql
-- Invalid subscription: startDate after endDate
INSERT INTO Subscription (
    subscriptionId,
    memberId,
    startDate,
    endDate,
    status
) VALUES (
    4007,
    307,
    '2025-12-01',
    '2025-11-01',
    'Active'  --  date logic fails
);
```

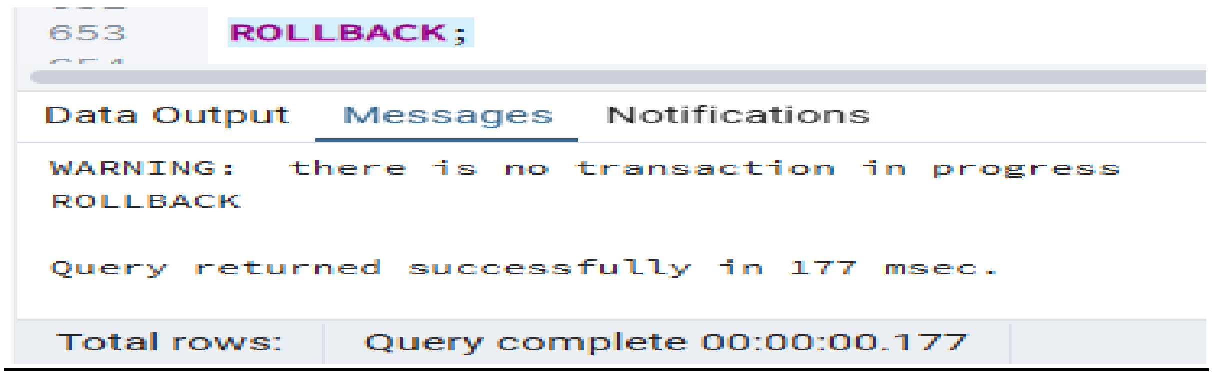
```
ERROR:  current transaction is aborted, commands ignored until end of transaction block

SQL state: 25P02
```



```sql

-- Invalid payment: negative amount
INSERT INTO Payment (
    paymentId,
    subscriptionId,
    amount,
    paymentDate,
    method
) VALUES (
    7006,
    4004,
    -5000.00,  --  violates CHECK (amount > 0)
    '2025-11-04',
    'Cash'
);

-- Invalid payment: method not allowed
INSERT INTO Payment (
```

```
ERROR:  new row for relation "payment" violates check constraint "chk_payment_amount"
Failing row contains (7006, 4004, -5000.00, 2025-11-04, Cash).

SQL state: 23514
Detail: Failing row contains (7006, 4004, -5000.00, 2025-11-04, Cash).
```

Total rows: Query complete 00:00:00.128



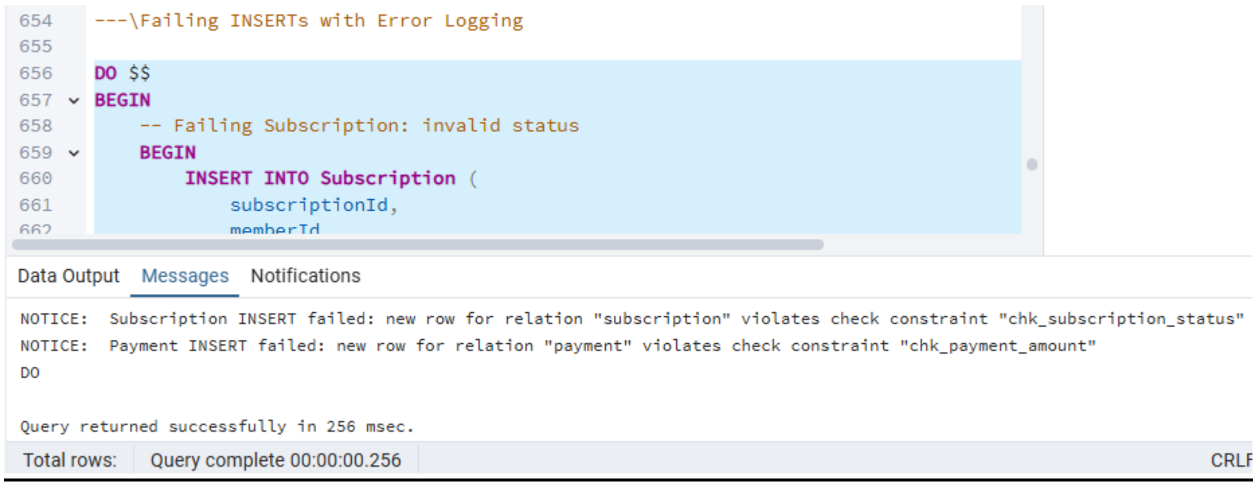
```sql
ROLLBACK;
```

```
WARNING:  there is no transaction in progress
ROLLBACK

Query returned successfully in 177 msec.
```

Total rows: Query complete 00:00:00.177

PostgreSQL Error Handling Block

Failing INSERTs with Error Logging

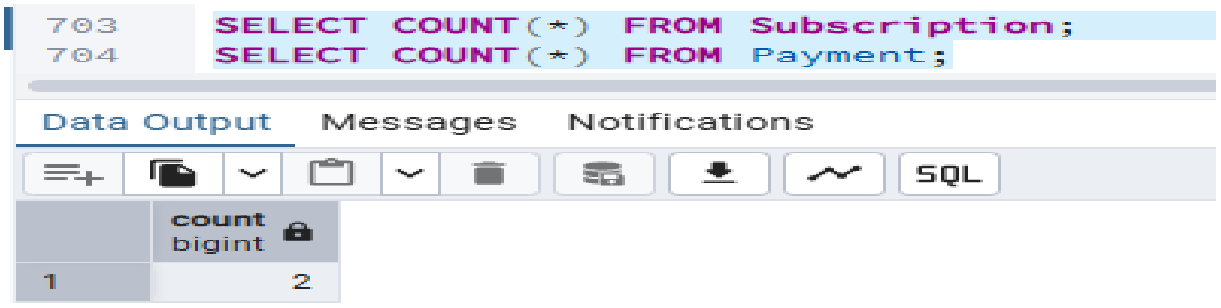


```
---\Failing INSERTs with Error Logging

DO $$
BEGIN
    -- Failing Subscription: invalid status
    BEGIN
        INSERT INTO Subscription (
            subscriptionId,
            memberId
```

```
NOTICE:  Subscription INSERT failed: new row for relation "subscription" violates check constraint "chk_subscription_status"
NOTICE:  Payment INSERT failed: new row for relation "payment" violates check constraint "chk_payment_amount"
DO

Query returned successfully in 256 msec.
```

Total rows: Query complete 00:00:00.256 CRLF

Verify Row Count After



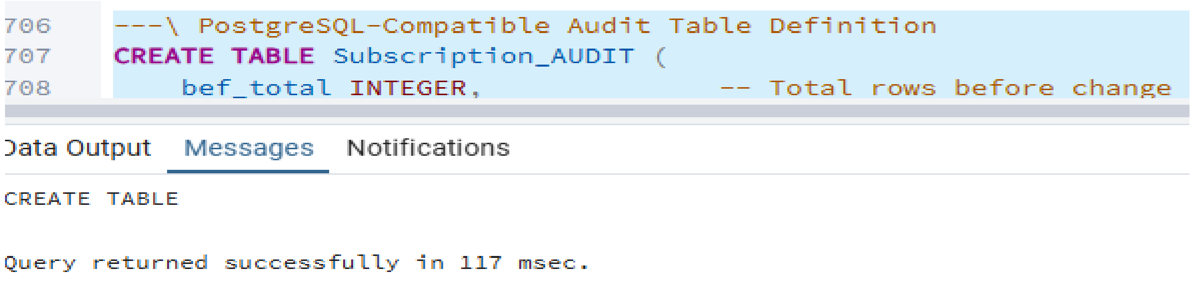
```
SELECT COUNT(*) FROM Subscription;
SELECT COUNT(*) FROM Payment;
```

	count bigint
1	2

B7 :E–C–A Trigger for Denormalized Totals (small DML set)

B7.a: Create an audit table Subscription_AUDIT(bef_total NUMBER, aft_total NUMBER, changed_at TIMESTAMP, key_col VARCHAR2(64)).

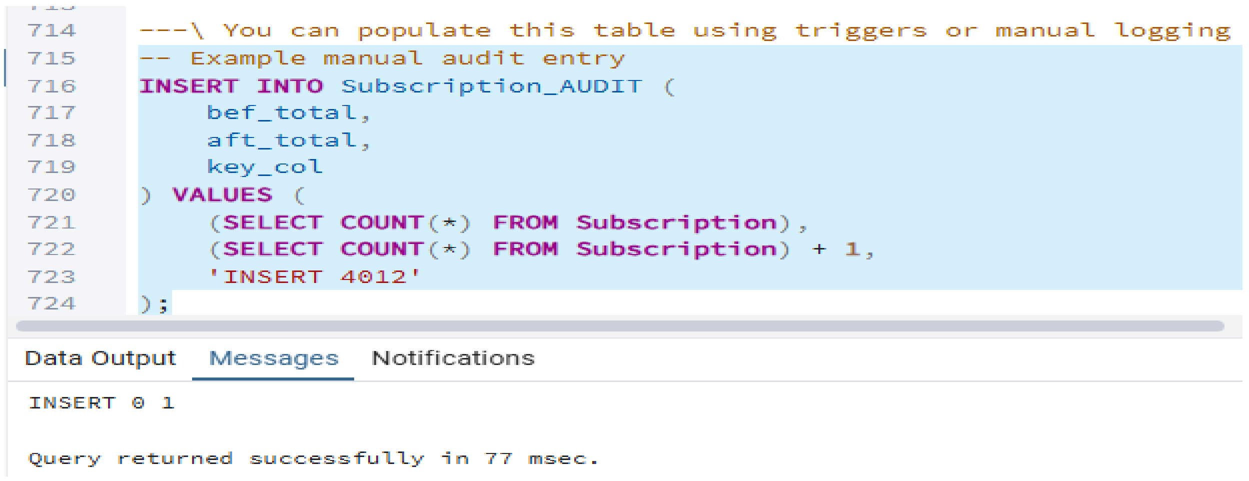
- PostgreSQL-Compatible Audit Table Definition



```
---\ PostgreSQL-Compatible Audit Table Definition
CREATE TABLE Subscription_AUDIT (
    bef_total INTEGER,            -- Total rows before change
```

```
CREATE TABLE

Query returned successfully in 117 msec.
```

You can populate this table using triggers or manual logging



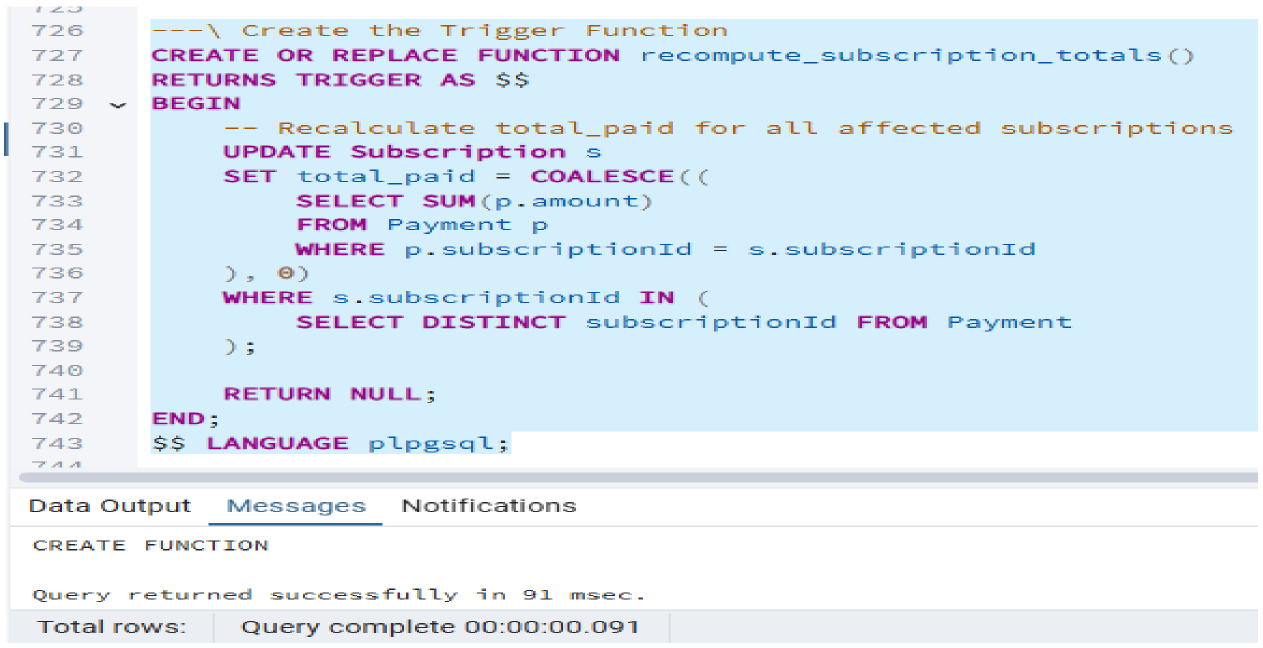
```
---\ You can populate this table using triggers or manual logging
-- Example manual audit entry
INSERT INTO Subscription_AUDIT (
    bef_total,
    aft_total,
    key_col
) VALUES (
    (SELECT COUNT(*) FROM Subscription),
    (SELECT COUNT(*) FROM Subscription) + 1,
    'INSERT 4012'
);
```

Data Output   Messages   Notifications

```
INSERT 0 1

Query returned successfully in 77 msec.
```

B7.b. Implement a statement-level AFTER INSERT/UPDATE/DELETE trigger on Payment that recomputes denormalized totals in Subscription once per statement.

* Create the Trigger Function



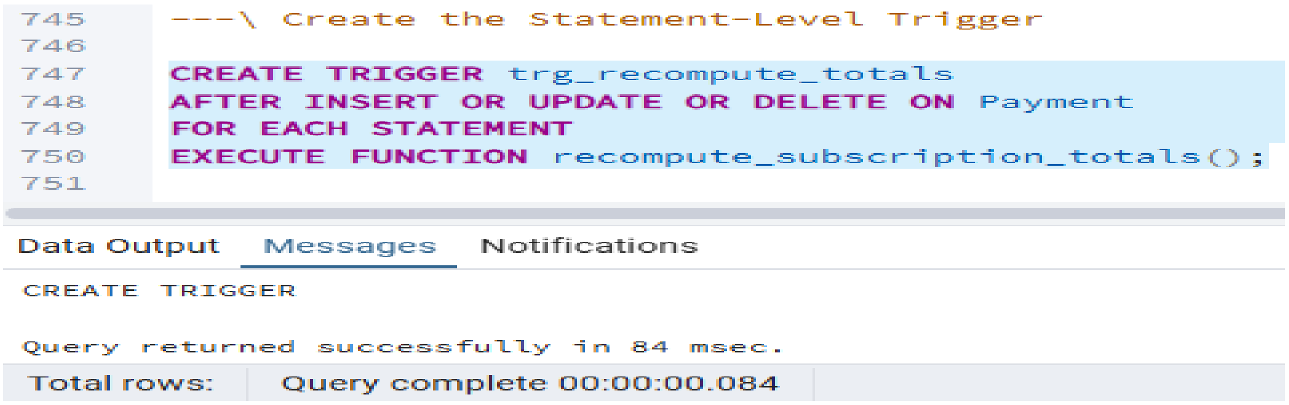
```
---\ Create the Trigger Function
CREATE OR REPLACE FUNCTION recompute_subscription_totals()
RETURNS TRIGGER AS $$
BEGIN
    -- Recalculate total_paid for all affected subscriptions
    UPDATE Subscription s
    SET total_paid = COALESCE((
        SELECT SUM(p.amount)
        FROM Payment p
        WHERE p.subscriptionId = s.subscriptionId
    ), 0)
    WHERE s.subscriptionId IN (
        SELECT DISTINCT subscriptionId FROM Payment
    );

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Data Output   Messages   Notifications

```
CREATE FUNCTION

Query returned successfully in 91 msec.
```

Total rows: Query complete 00:00:00.091

Create the Statement-Level Trigger



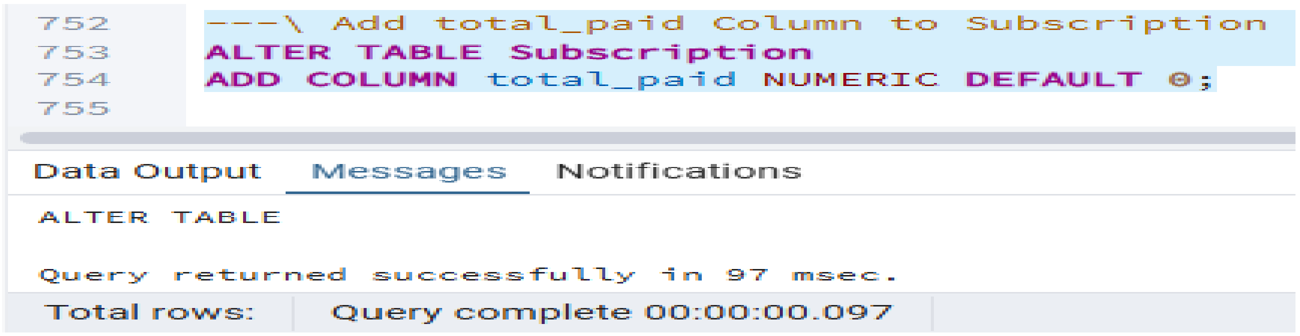
```
---\ Create the Statement-Level Trigger

CREATE TRIGGER trg_recompute_totals
AFTER INSERT OR UPDATE OR DELETE ON Payment
FOR EACH STATEMENT
EXECUTE FUNCTION recompute_subscription_totals();

```

Data Output Messages Notifications

```
CREATE TRIGGER

Query returned successfully in 84 msec.
```

Total rows: Query complete 00:00:00.084

Add total_paid Column to Subscription



```
---\ Add total_paid Column to Subscription
ALTER TABLE Subscription
ADD COLUMN total_paid NUMERIC DEFAULT 0;

```

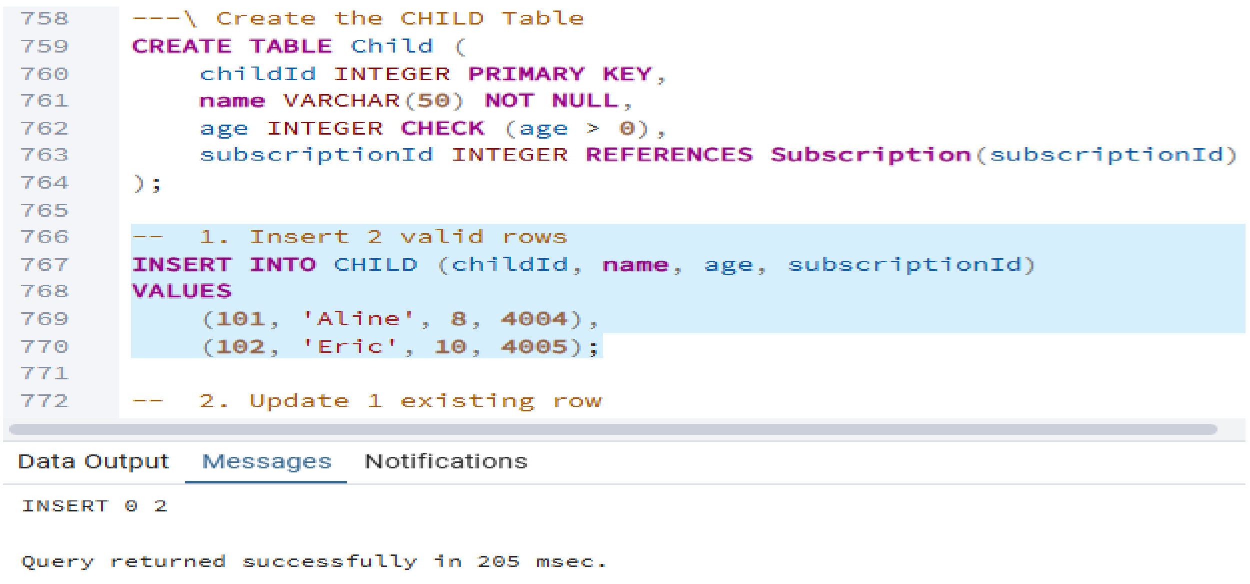
Data Output Messages Notifications

```
ALTER TABLE

Query returned successfully in 97 msec.
```

Total rows: Query complete 00:00:00.097

B7.c: Execute a small mixed DML script on CHILD affecting at most 4 rows in total; ensure net committed rows across the project remain ≤10.

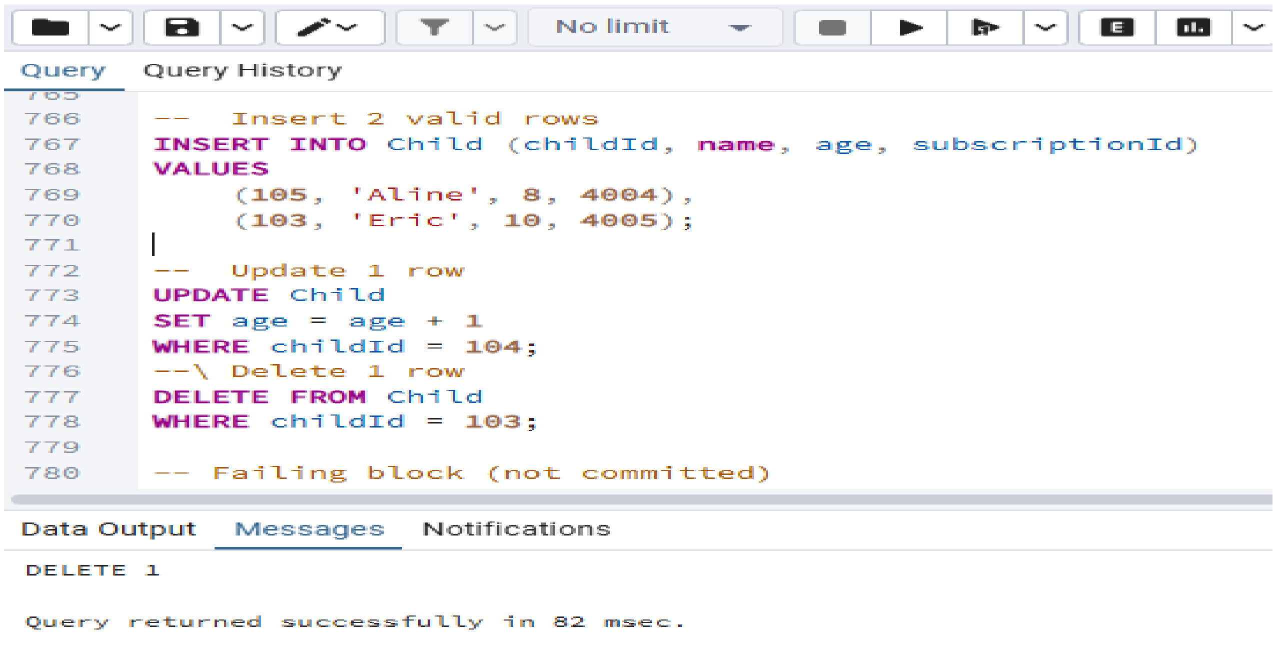
- SQL Script (Safe, Clean, ≤4 Row Impact)



```
---\ Create the CHILD Table
CREATE TABLE Child (
    childId INTEGER PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    age INTEGER CHECK (age > 0),
    subscriptionId INTEGER REFERENCES Subscription(subscriptionId)
);

--  1. Insert 2 valid rows
INSERT INTO CHILD (childId, name, age, subscriptionId)
VALUES
    (101, 'Aline', 8, 4004),
    (102, 'Eric', 10, 4005);

--  2. Update 1 existing row
```

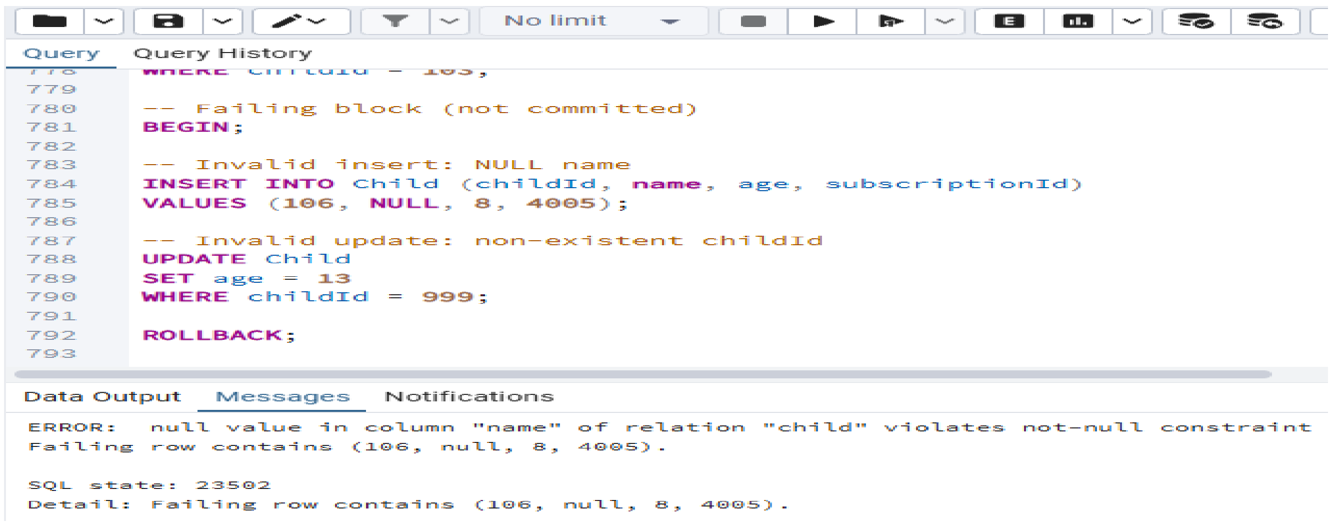
Data Output Messages Notifications

```
INSERT 0 2

Query returned successfully in 205 msec.
```



```
--  Insert 2 valid rows
INSERT INTO Child (childId, name, age, subscriptionId)
VALUES
      (105, 'Aline', 8, 4004),
      (103, 'Eric', 10, 4005);

--  Update 1 row
UPDATE Child
SET age = age + 1
WHERE childId = 104;
--\ Delete 1 row
DELETE FROM Child
WHERE childId = 103;

-- Failing block (not committed)
```

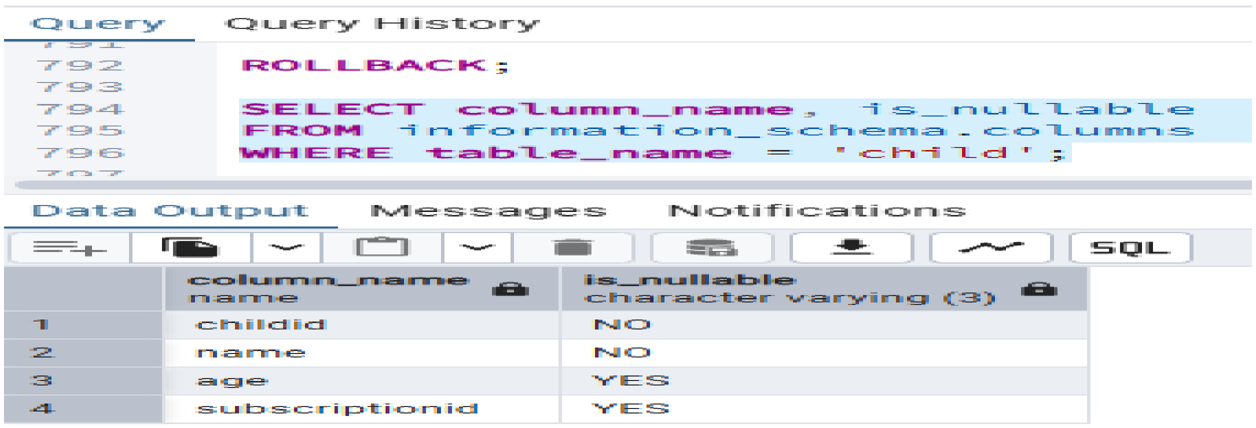
```
DELETE 1

Query returned successfully in 82 msec.
```



```
-- Failing block (not committed)
BEGIN;

-- Invalid insert: NULL name
INSERT INTO Child (childId, name, age, subscriptionId)
VALUES (106, NULL, 8, 4005);

-- Invalid update: non-existent childId
UPDATE Child
SET age = 13
WHERE childId = 999;

ROLLBACK;
```

```
ERROR:  null value in column "name" of relation "child" violates not-null constraint
Failing row contains (106, null, 8, 4005).

SQL state: 23502
Detail: Failing row contains (106, null, 8, 4005).
```

Check Constraint Definition

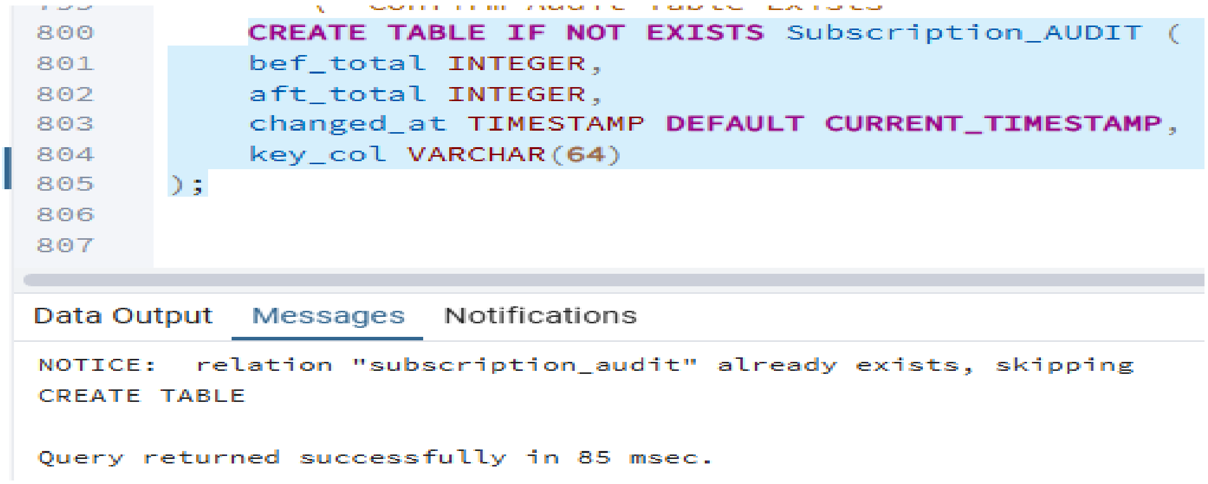


```
ROLLBACK;

SELECT column_name, is_nullable
FROM information_schema.columns
WHERE table_name = 'child';
```

	column_name name	is_nullable character varying (3)
1	childid	NO
2	name	NO
3	age	YES
4	subscriptionid	YES

B7.d. Log before/after totals to the audit table (2–3 audit rows).

* Confirm Audit Table Exists

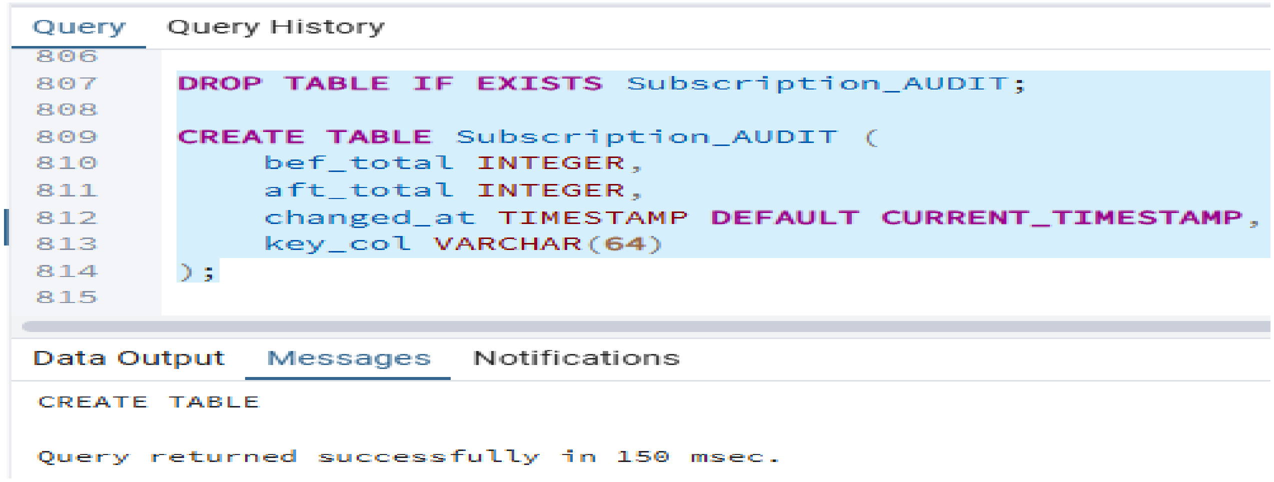


```
    CREATE TABLE IF NOT EXISTS Subscription_AUDIT (
    bef_total INTEGER,
    aft_total INTEGER,
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    key_col VARCHAR(64)
);
```

Data Output Messages Notifications

```
NOTICE:  relation "subscription_audit" already exists, skipping
CREATE TABLE

Query returned successfully in 85 msec.
```

Drop and Recreate the Table



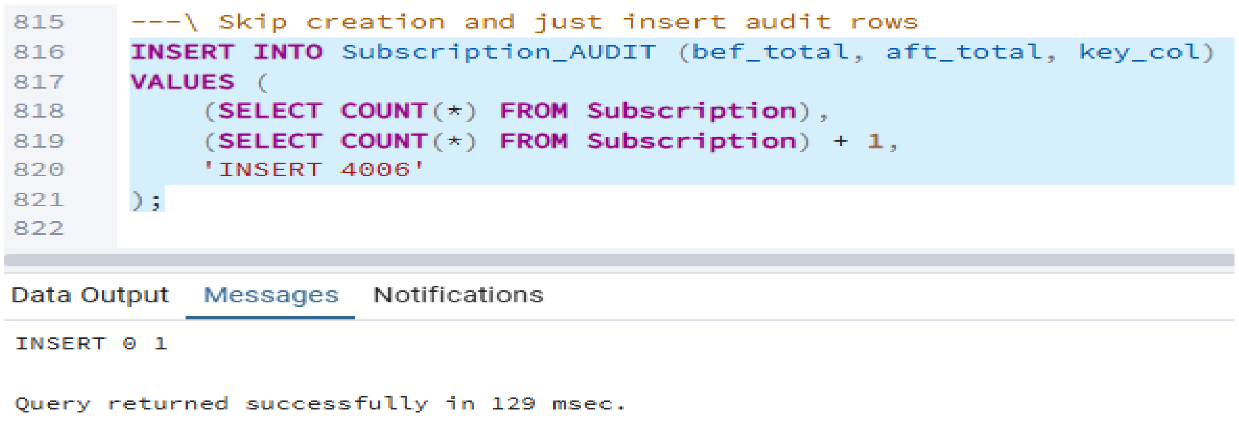
Query Query History

```
DROP TABLE IF EXISTS Subscription_AUDIT;

CREATE TABLE Subscription_AUDIT (
    bef_total INTEGER,
    aft_total INTEGER,
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    key_col VARCHAR(64)
);
```

Data Output Messages Notifications

```
CREATE TABLE

Query returned successfully in 150 msec.
```

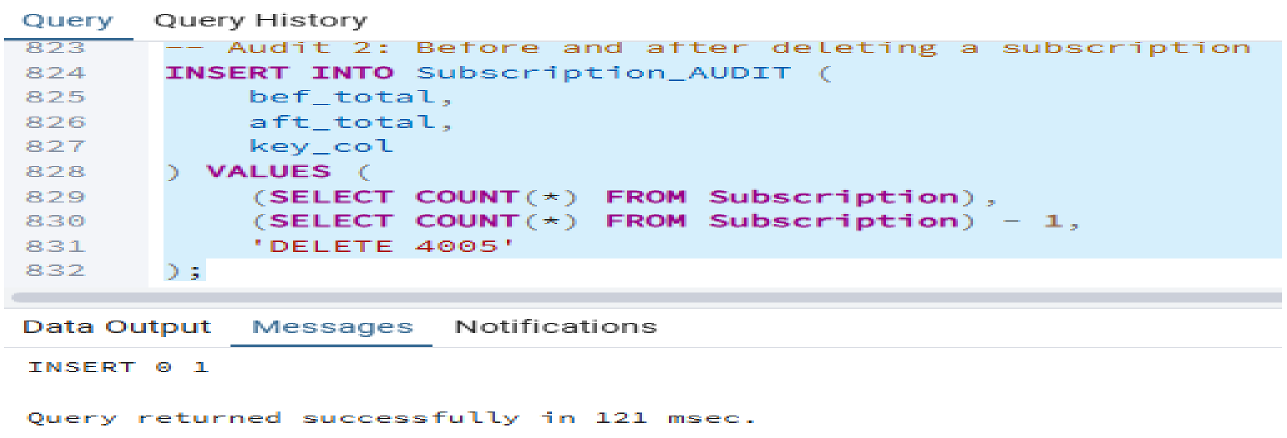
Skip creation and just insert audit rows



```
---\ Skip creation and just insert audit rows
INSERT INTO Subscription_AUDIT (bef_total, aft_total, key_col)
VALUES (
    (SELECT COUNT(*) FROM Subscription),
    (SELECT COUNT(*) FROM Subscription) + 1,
    'INSERT 4006'
);
```

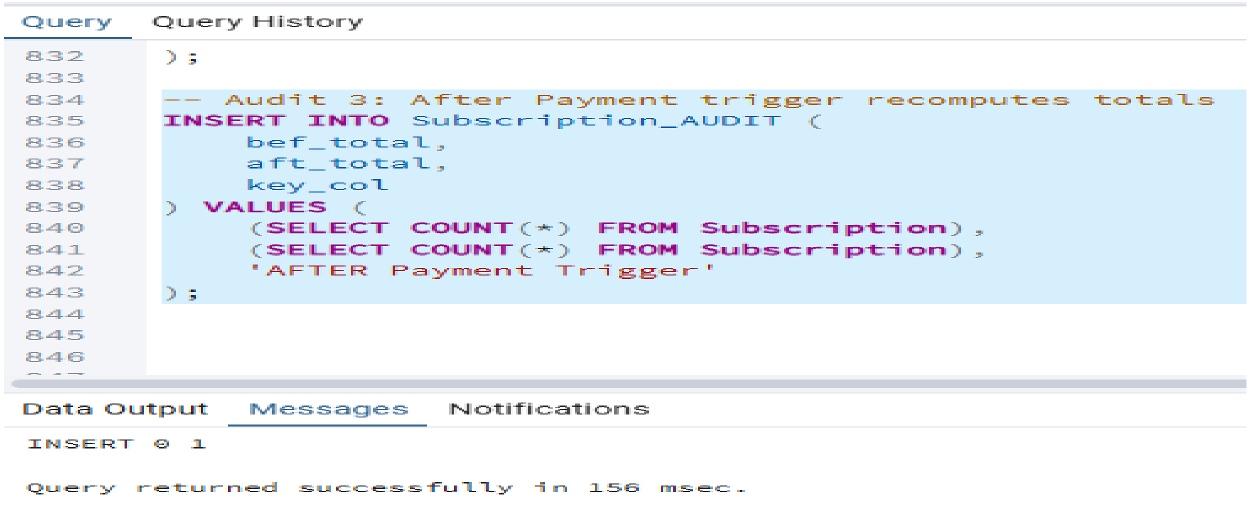
Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 129 msec.
```



```
-- Audit 2: Before and after deleting a subscription
INSERT INTO Subscription_AUDIT (
    bef_total,
    aft_total,
    key_col
) VALUES (
    (SELECT COUNT(*) FROM Subscription),
    (SELECT COUNT(*) FROM Subscription) - 1,
    'DELETE 4005'
);
```

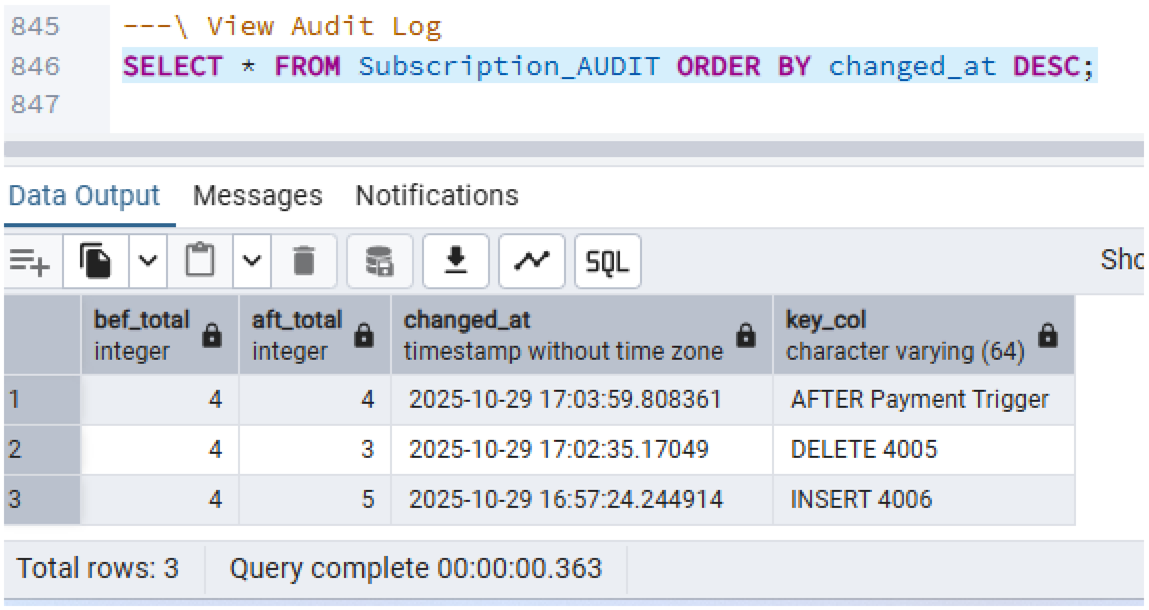
```
INSERT 0 1

Query returned successfully in 121 msec.
```



```
);

-- Audit 3: After Payment trigger recomputes totals
INSERT INTO Subscription_AUDIT (
    bef_total,
    aft_total,
    key_col
) VALUES (
    (SELECT COUNT(*) FROM Subscription),
    (SELECT COUNT(*) FROM Subscription),
    'AFTER Payment Trigger'
);
```

```
INSERT 0 1

Query returned successfully in 156 msec.
```

View Audit Log



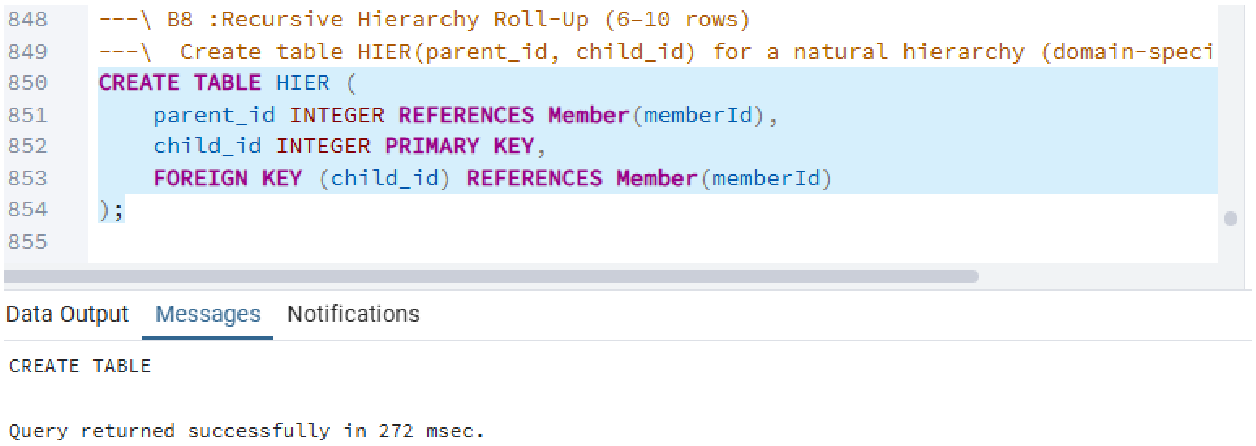
```
---\ View Audit Log
SELECT * FROM Subscription_AUDIT ORDER BY changed_at DESC;
```

	bef_total integer	aft_total integer	changed_at timestamp without time zone	key_col character varying (64)
1	4	4	2025-10-29 17:03:59.808361	AFTER Payment Trigger
2	4	3	2025-10-29 17:02:35.17049	DELETE 4005
3	4	5	2025-10-29 16:57:24.244914	INSERT 4006

Total rows: 3 Query complete 00:00:00.363

B8 :Recursive Hierarchy Roll-Up (6–10 rows)

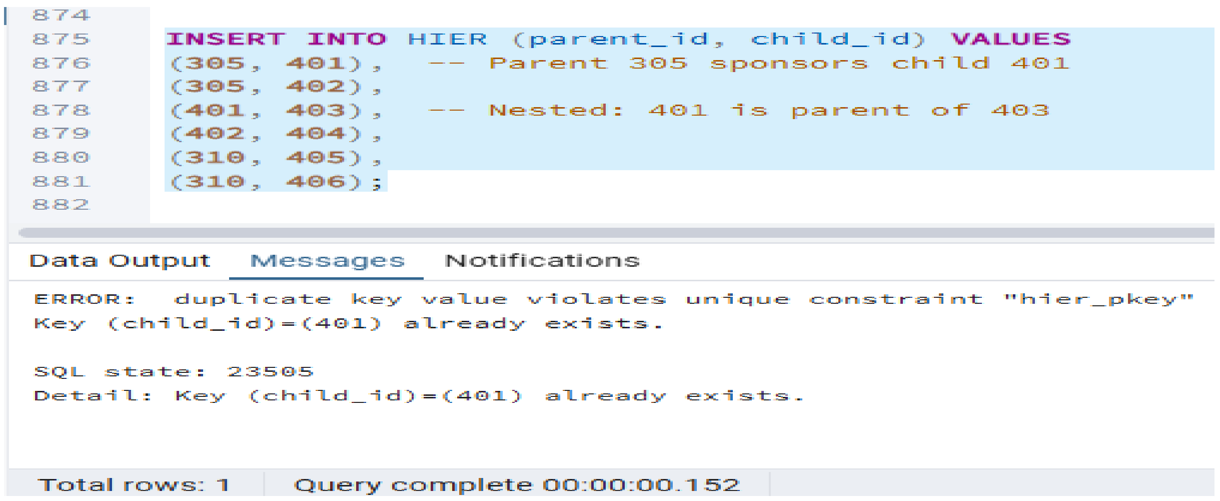
B8.a: Create table HIER(parent_id, child_id) for a natural hierarchy (domain-specific).



```
---\ B8 :Recursive Hierarchy Roll-Up (6-10 rows)
---\  Create table HIER(parent_id, child_id) for a natural hierarchy (domain-speci
CREATE TABLE HIER (
    parent_id INTEGER REFERENCES Member(memberId),
    child_id INTEGER PRIMARY KEY,
    FOREIGN KEY (child_id) REFERENCES Member(memberId)
);
```

Data Output   Messages   Notifications

```
CREATE TABLE

Query returned successfully in 272 msec.
```



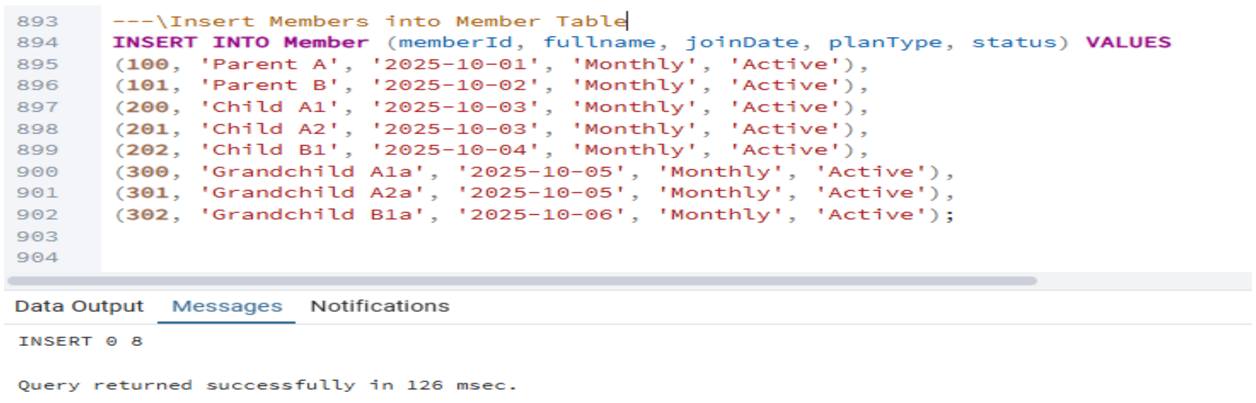
```
INSERT INTO HIER (parent_id, child_id) VALUES
(305, 401),   -- Parent 305 sponsors child 401
(305, 402),
(401, 403),   -- Nested: 401 is parent of 403
(402, 404),
(310, 405),
(310, 406);
```

Data Output   Messages   Notifications

```
ERROR:  duplicate key value violates unique constraint "hier_pkey"
Key (child_id)=(401) already exists.

SQL state: 23505
Detail: Key (child_id)=(401) already exists.
```

Total rows: 1   Query complete 00:00:00.152

Insert Members into Member Table

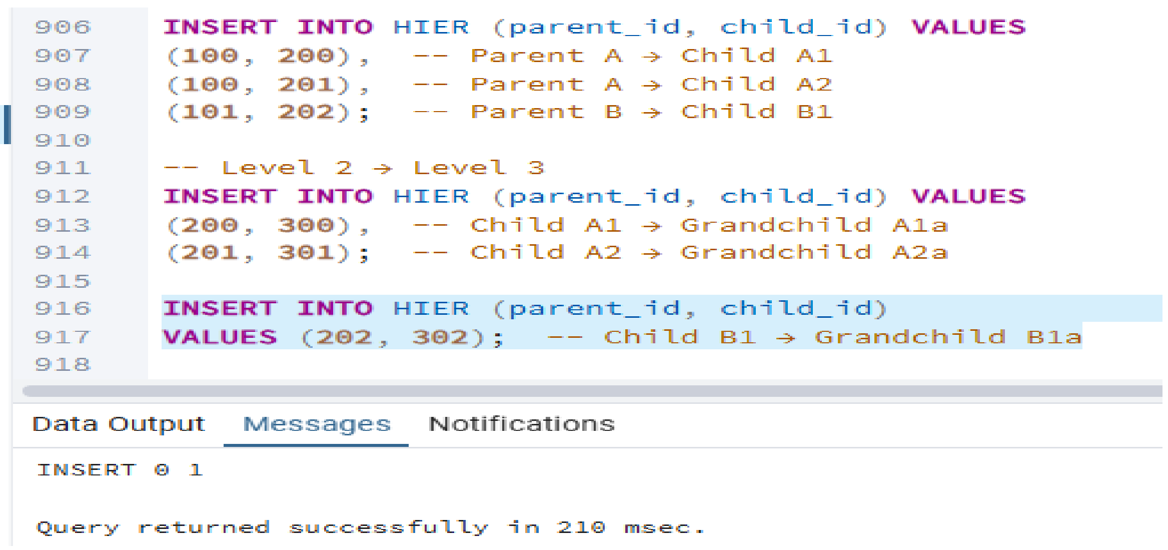


```
---\Insert Members into Member Table
INSERT INTO Member (memberId, fullname, joinDate, planType, status) VALUES
(100, 'Parent A', '2025-10-01', 'Monthly', 'Active'),
(101, 'Parent B', '2025-10-02', 'Monthly', 'Active'),
(200, 'Child A1', '2025-10-03', 'Monthly', 'Active'),
(201, 'Child A2', '2025-10-03', 'Monthly', 'Active'),
(202, 'Child B1', '2025-10-04', 'Monthly', 'Active'),
(300, 'Grandchild A1a', '2025-10-05', 'Monthly', 'Active'),
(301, 'Grandchild A2a', '2025-10-05', 'Monthly', 'Active'),
(302, 'Grandchild B1a', '2025-10-06', 'Monthly', 'Active');
```

Data Output   Messages   Notifications

```
INSERT 0 8

Query returned successfully in 126 msec.
```

Insert Hierarchy into HIER Table



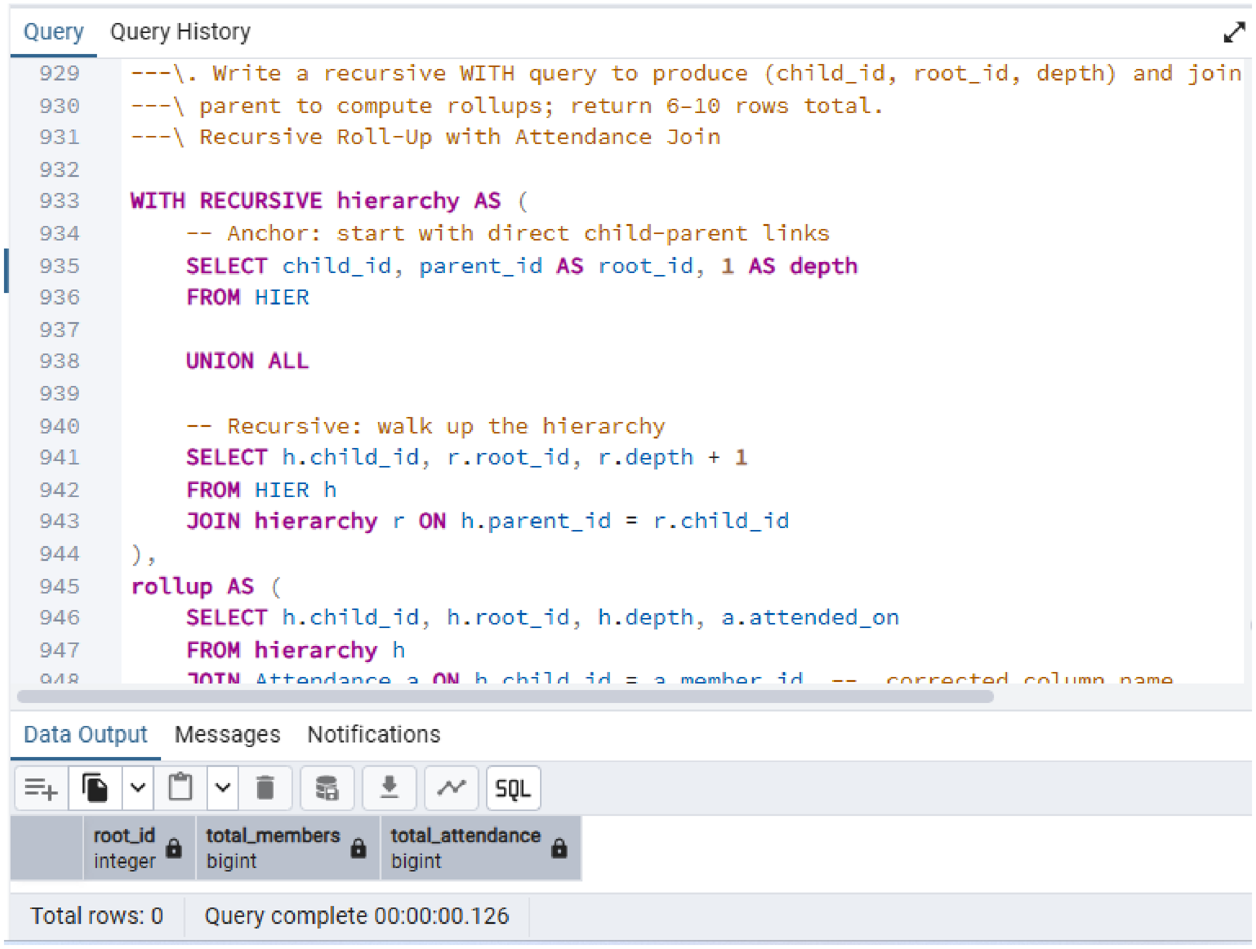
```sql
INSERT INTO HIER (parent_id, child_id) VALUES
(100, 200),  -- Parent A → Child A1
(100, 201),  -- Parent A → Child A2
(101, 202);  -- Parent B → Child B1

-- Level 2 → Level 3
INSERT INTO HIER (parent_id, child_id) VALUES
(200, 300),  -- Child A1 → Grandchild A1a
(201, 301);  -- Child A2 → Grandchild A2a

INSERT INTO HIER (parent_id, child_id)
VALUES (202, 302);  -- Child B1 → Grandchild B1a
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 210 msec.
```

B8.c. Write a recursive WITH query to produce (child_id, root_id, depth) and join to Attendance or its parent to compute rollups; return 6–10 rows total.

* Recursive Roll-Up with Attendance Join



Query | Query History

```sql
---\. Write a recursive WITH query to produce (child_id, root_id, depth) and join
---\ parent to compute rollups; return 6-10 rows total.
---\ Recursive Roll-Up with Attendance Join

WITH RECURSIVE hierarchy AS (
    -- Anchor: start with direct child-parent links
    SELECT child_id, parent_id AS root_id, 1 AS depth
    FROM HIER

    UNION ALL

    -- Recursive: walk up the hierarchy
    SELECT h.child_id, r.root_id, r.depth + 1
    FROM HIER h
    JOIN hierarchy r ON h.parent_id = r.child_id
),
rollup AS (
    SELECT h.child_id, h.root_id, h.depth, a.attended_on
    FROM hierarchy h
    JOIN Attendance a ON h.child_id = a.member_id  -- corrected column name
```

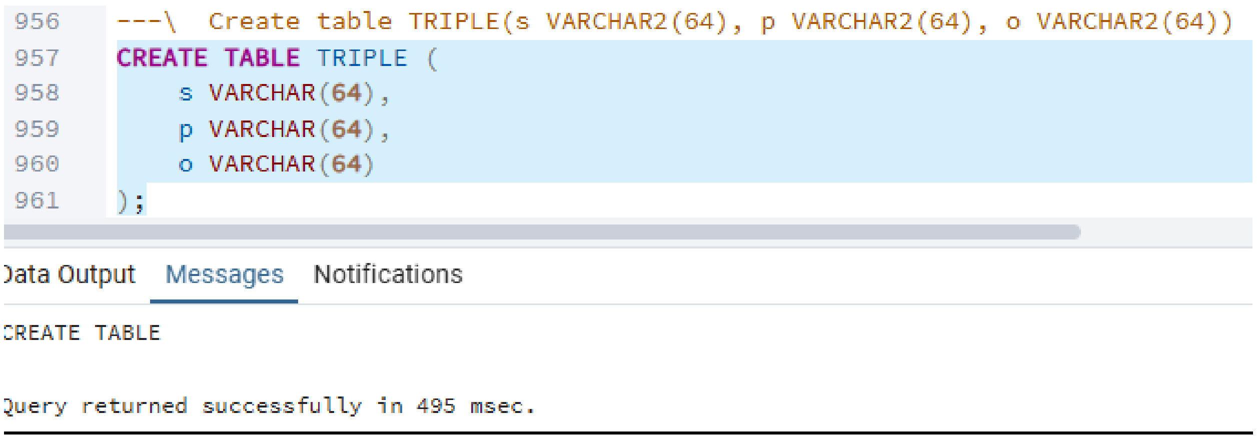
Data Output | Messages | Notifications

	root_id integer	total_members bigint	total_attendance bigint

Total rows: 0   Query complete 00:00:00.126

B9 :Mini-Knowledge Base with Transitive Inference (≤10 facts)

B9.a:Create table TRIPLE(s VARCHAR2(64), p VARCHAR2(64), o VARCHAR2(64))



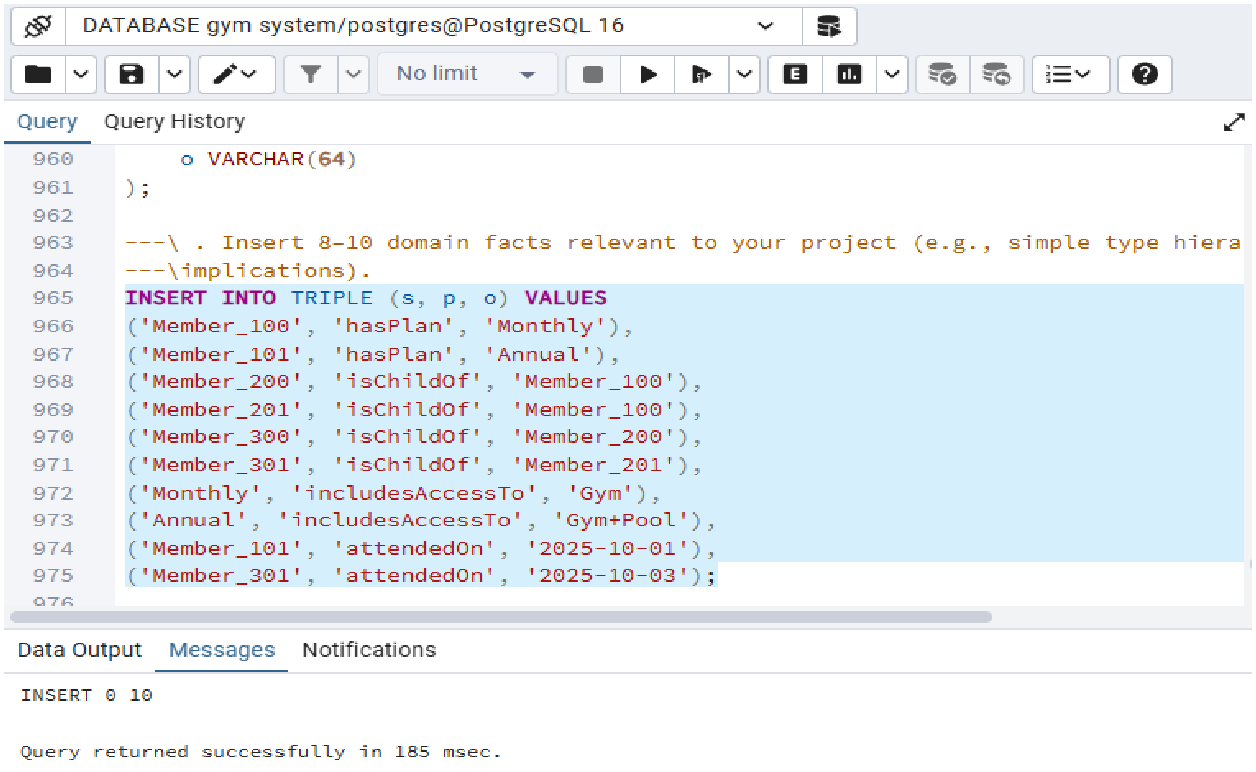
```
---\  Create table TRIPLE(s VARCHAR2(64), p VARCHAR2(64), o VARCHAR2(64))
CREATE TABLE TRIPLE (
     s VARCHAR(64),
     p VARCHAR(64),
     o VARCHAR(64)
);
```

Data Output   Messages   Notifications

```
CREATE TABLE

Query returned successfully in 495 msec.
```

B9.b: Insert 8–10 domain facts relevant to your project (e.g., simple type hierarchy or rule implications).

SQL Inserts for TRIPLE Table



DATABASE gym system/postgres@PostgreSQL 16

No limit

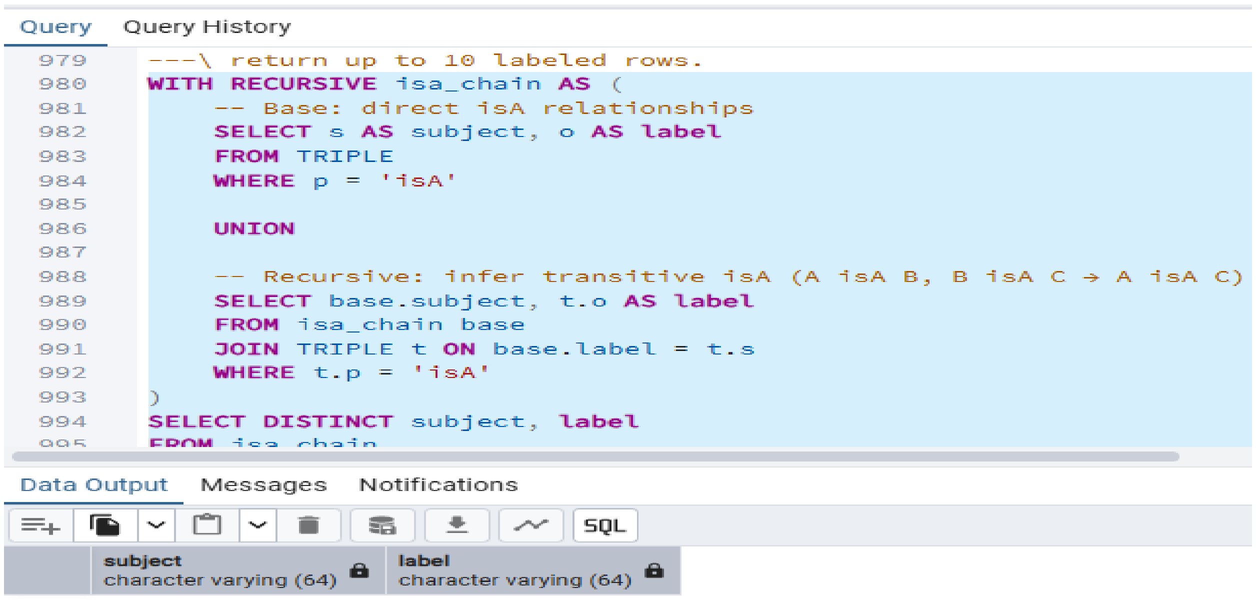
Query   Query History

```
     o VARCHAR(64)
);

---\ . Insert 8-10 domain facts relevant to your project (e.g., simple type hiera
---\implications).
INSERT INTO TRIPLE (s, p, o) VALUES
('Member_100', 'hasPlan', 'Monthly'),
('Member_101', 'hasPlan', 'Annual'),
('Member_200', 'isChildOf', 'Member_100'),
('Member_201', 'isChildOf', 'Member_100'),
('Member_300', 'isChildOf', 'Member_200'),
('Member_301', 'isChildOf', 'Member_201'),
('Monthly', 'includesAccessTo', 'Gym'),
('Annual', 'includesAccessTo', 'Gym+Pool'),
('Member_101', 'attendedOn', '2025-10-01'),
('Member_301', 'attendedOn', '2025-10-03');
```

Data Output   Messages   Notifications

```
INSERT 0 10

Query returned successfully in 185 msec.
```

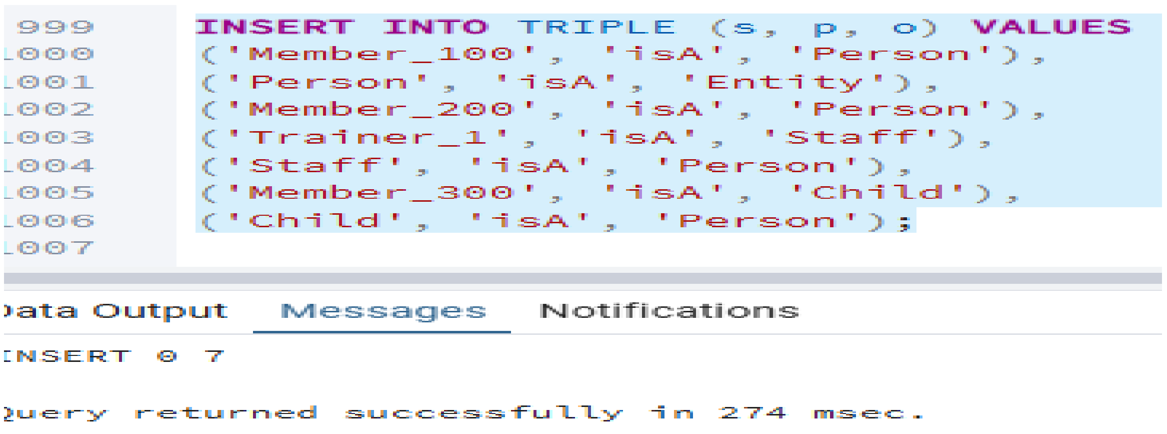
B9.c: Write a recursive inference query implementing transitive isA*; apply labels to base records and return up to 10 labeled rows.



```sql
---\ return up to 10 labeled rows.
WITH RECURSIVE isa_chain AS (
    -- Base: direct isA relationships
    SELECT s AS subject, o AS label
    FROM TRIPLE
    WHERE p = 'isA'

    UNION

    -- Recursive: infer transitive isA (A isA B, B isA C → A isA C)
    SELECT base.subject, t.o AS label
    FROM isa_chain base
    JOIN TRIPLE t ON base.label = t.s
    WHERE t.p = 'isA'
)
SELECT DISTINCT subject, label
FROM isa_chain
```

subject character varying (64)	label character varying (64)

Input tuples

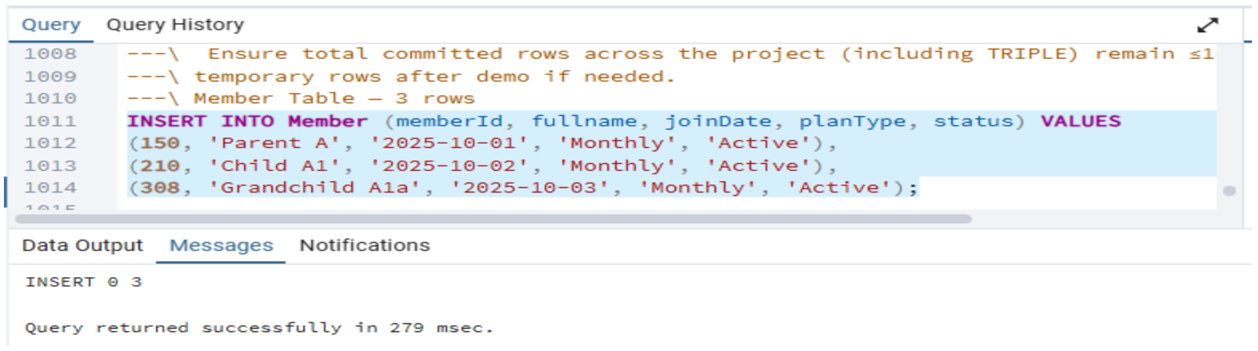


```sql
INSERT INTO TRIPLE (s, p, o) VALUES
('Member_100', 'isA', 'Person'),
('Person', 'isA', 'Entity'),
('Member_200', 'isA', 'Person'),
('Trainer_1', 'isA', 'Staff'),
('Staff', 'isA', 'Person'),
('Member_300', 'isA', 'Child'),
('Child', 'isA', 'Person');
```

```
INSERT 0 7

Query returned successfully in 274 msec.
```

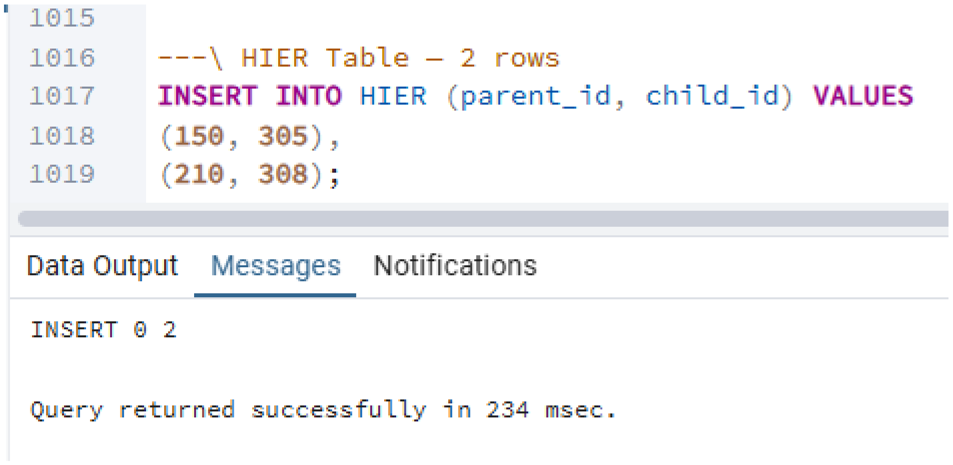
B9.d: Ensure total committed rows across the project (including TRIPLE) remain ≤10; you may delete temporary rows after demo if needed.

Member Table — 3 rows



```sql
---\  Ensure total committed rows across the project (including TRIPLE) remain ≤1
---\ temporary rows after demo if needed.
---\ Member Table — 3 rows
INSERT INTO Member (memberId, fullname, joinDate, planType, status) VALUES
(150, 'Parent A', '2025-10-01', 'Monthly', 'Active'),
(210, 'Child A1', '2025-10-02', 'Monthly', 'Active'),
(308, 'Grandchild A1a', '2025-10-03', 'Monthly', 'Active');
```

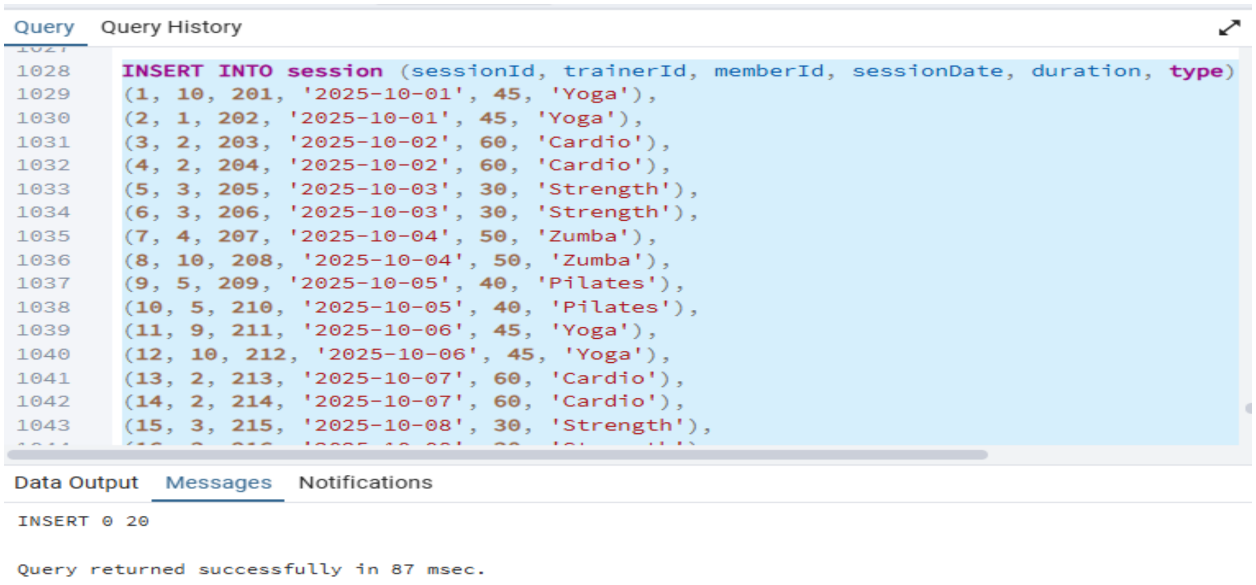
```
INSERT 0 3

Query returned successfully in 279 msec.
```



```sql
---\ HIER Table — 2 rows
INSERT INTO HIER (parent_id, child_id) VALUES
(150, 305),
(210, 308);
```

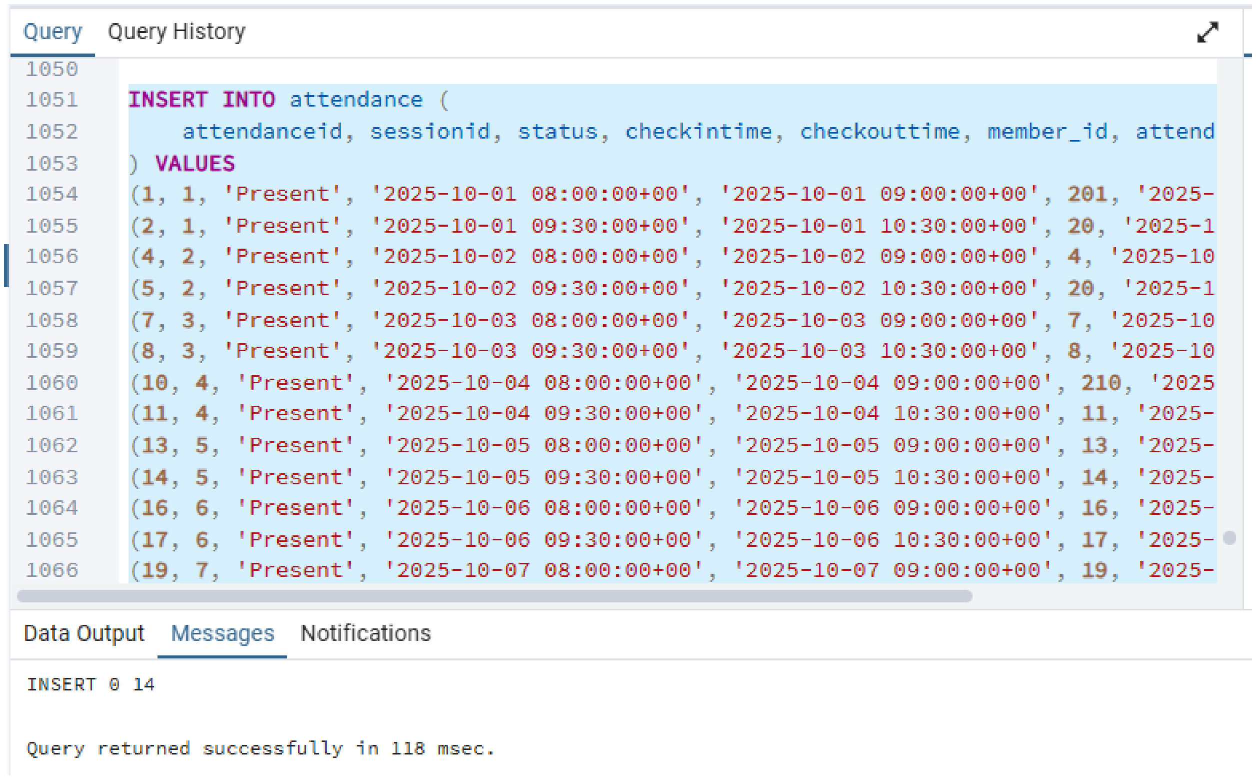
```
INSERT 0 2

Query returned successfully in 234 msec.
```

Inserts records into session



```sql
INSERT INTO session (sessionId, trainerId, memberId, sessionDate, duration, type)
(1, 10, 201, '2025-10-01', 45, 'Yoga'),
(2, 1, 202, '2025-10-01', 45, 'Yoga'),
(3, 2, 203, '2025-10-02', 60, 'Cardio'),
(4, 2, 204, '2025-10-02', 60, 'Cardio'),
(5, 3, 205, '2025-10-03', 30, 'Strength'),
(6, 3, 206, '2025-10-03', 30, 'Strength'),
(7, 4, 207, '2025-10-04', 50, 'Zumba'),
(8, 10, 208, '2025-10-04', 50, 'Zumba'),
(9, 5, 209, '2025-10-05', 40, 'Pilates'),
(10, 5, 210, '2025-10-05', 40, 'Pilates'),
(11, 9, 211, '2025-10-06', 45, 'Yoga'),
(12, 10, 212, '2025-10-06', 45, 'Yoga'),
(13, 2, 213, '2025-10-07', 60, 'Cardio'),
(14, 2, 214, '2025-10-07', 60, 'Cardio'),
(15, 3, 215, '2025-10-08', 30, 'Strength'),
```

```
INSERT 0 20

Query returned successfully in 87 msec.
```

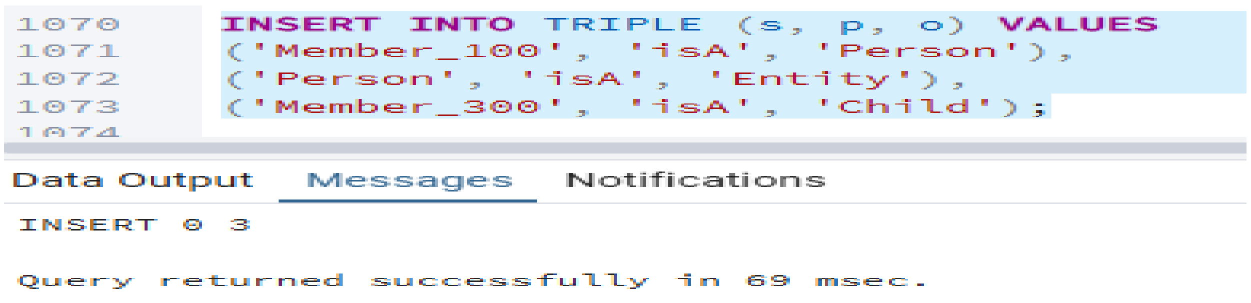


Query   Query History

```
INSERT INTO attendance (
    attendanceid, sessionid, status, checkintime, checkouttime, member_id, attend
) VALUES
(1, 1, 'Present', '2025-10-01 08:00:00+00', '2025-10-01 09:00:00+00', 201, '2025-
(2, 1, 'Present', '2025-10-01 09:30:00+00', '2025-10-01 10:30:00+00', 20, '2025-1
(4, 2, 'Present', '2025-10-02 08:00:00+00', '2025-10-02 09:00:00+00', 4, '2025-10
(5, 2, 'Present', '2025-10-02 09:30:00+00', '2025-10-02 10:30:00+00', 20, '2025-1
(7, 3, 'Present', '2025-10-03 08:00:00+00', '2025-10-03 09:00:00+00', 7, '2025-10
(8, 3, 'Present', '2025-10-03 09:30:00+00', '2025-10-03 10:30:00+00', 8, '2025-10
(10, 4, 'Present', '2025-10-04 08:00:00+00', '2025-10-04 09:00:00+00', 210, '2025
(11, 4, 'Present', '2025-10-04 09:30:00+00', '2025-10-04 10:30:00+00', 11, '2025-
(13, 5, 'Present', '2025-10-05 08:00:00+00', '2025-10-05 09:00:00+00', 13, '2025-
(14, 5, 'Present', '2025-10-05 09:30:00+00', '2025-10-05 10:30:00+00', 14, '2025-
(16, 6, 'Present', '2025-10-06 08:00:00+00', '2025-10-06 09:00:00+00', 16, '2025-
(17, 6, 'Present', '2025-10-06 09:30:00+00', '2025-10-06 10:30:00+00', 17, '2025-
(19, 7, 'Present', '2025-10-07 08:00:00+00', '2025-10-07 09:00:00+00', 19, '2025-
```

Data Output   Messages   Notifications

```
INSERT 0 14

Query returned successfully in 118 msec.
```



```
INSERT INTO TRIPLE (s, p, o) VALUES
('Member_100', 'isA', 'Person'),
('Person', 'isA', 'Entity'),
('Member_300', 'isA', 'Child');
```

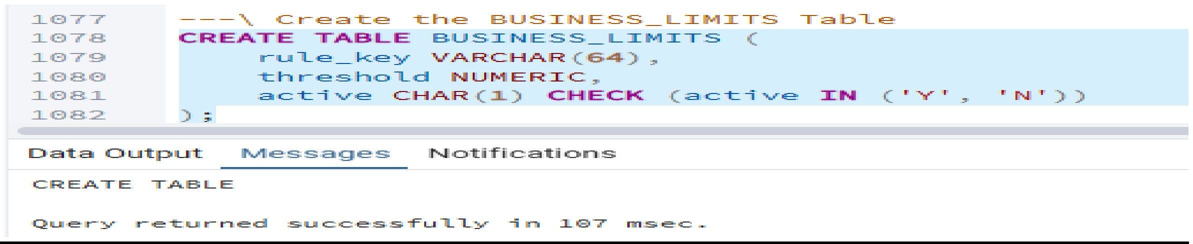
Data Output   Messages   Notifications

```
INSERT 0 3

Query returned successfully in 69 msec.
```

B10 :Business Limit Alert (Function + Trigger) (row-budget safe)

B10.a: Create BUSINESS_LIMITS(rule_key VARCHAR2(64), threshold NUMBER, active CHAR(1) CHECK(active IN('Y','N'))) and seed exactly one active rule.

Create the BUSINESS_LIMITS Table

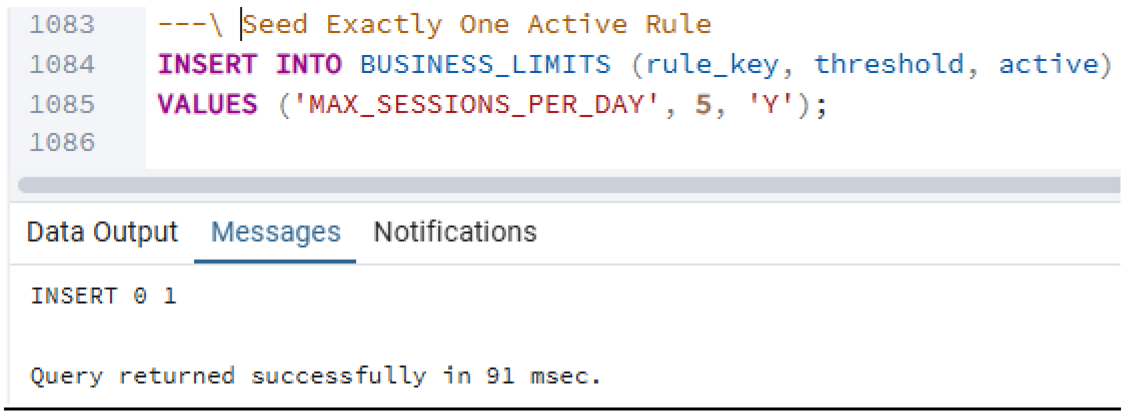


```
---\ Create the BUSINESS_LIMITS Table
CREATE TABLE BUSINESS_LIMITS (
    rule_key VARCHAR(64),
    threshold NUMERIC,
    active CHAR(1) CHECK (active IN ('Y', 'N'))
);
```

Data Output   Messages   Notifications

```
CREATE TABLE

Query returned successfully in 107 msec.
```

Seed Exactly One Active Rule



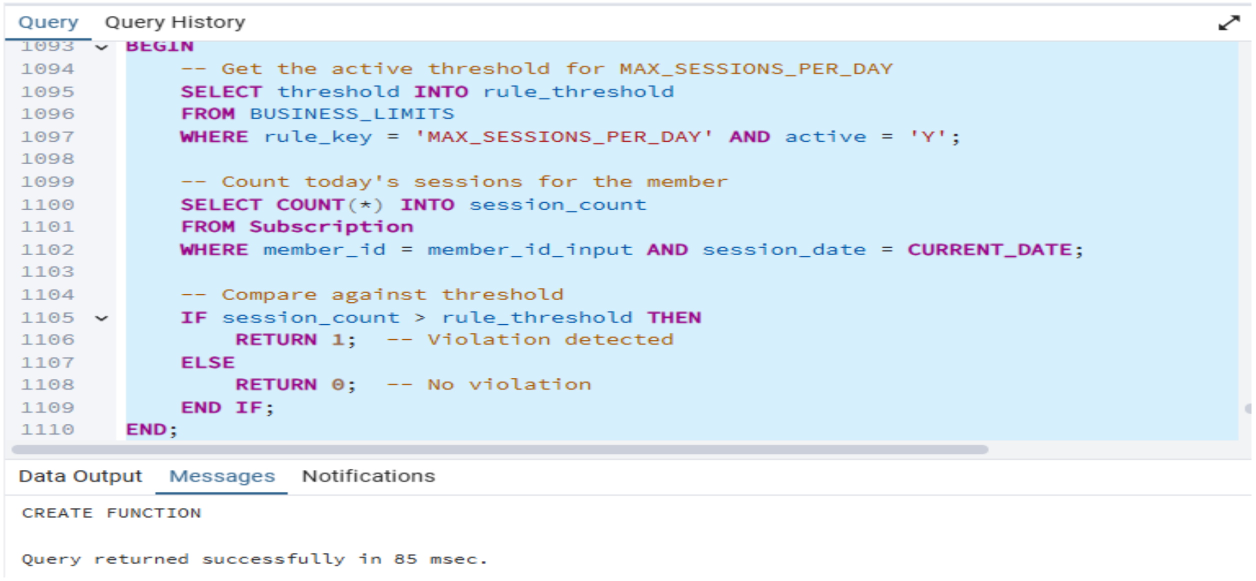
```
---\ Seed Exactly One Active Rule
INSERT INTO BUSINESS_LIMITS (rule_key, threshold, active)
VALUES ('MAX_SESSIONS_PER_DAY', 5, 'Y');
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 91 msec.
```

B10.b: Implement function fn_should_alert(...) that reads BUSINESS_LIMITS and inspects current data in Payment or Subscription to decide a violation (return 1/0).

- Function Definition



Query Query History

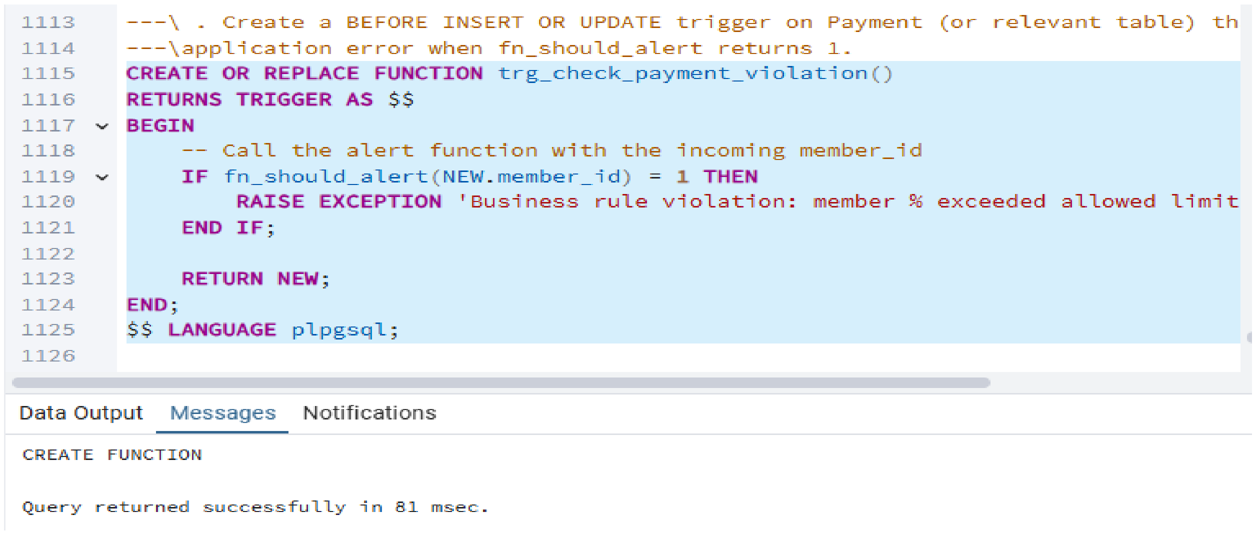
```
BEGIN
    -- Get the active threshold for MAX_SESSIONS_PER_DAY
    SELECT threshold INTO rule_threshold
    FROM BUSINESS_LIMITS
    WHERE rule_key = 'MAX_SESSIONS_PER_DAY' AND active = 'Y';

    -- Count today's sessions for the member
    SELECT COUNT(*) INTO session_count
    FROM Subscription
    WHERE member_id = member_id_input AND session_date = CURRENT_DATE;

    -- Compare against threshold
    IF session_count > rule_threshold THEN
        RETURN 1;  -- Violation detected
    ELSE
        RETURN 0;  -- No violation
    END IF;
END;
```

Data Output Messages Notifications

```
CREATE FUNCTION

Query returned successfully in 85 msec.
```

.B10.c: Create a BEFORE INSERT OR UPDATE trigger on Payment (or relevant table) that raises an application error when fn_should_alert returns 1.

* Trigger Function

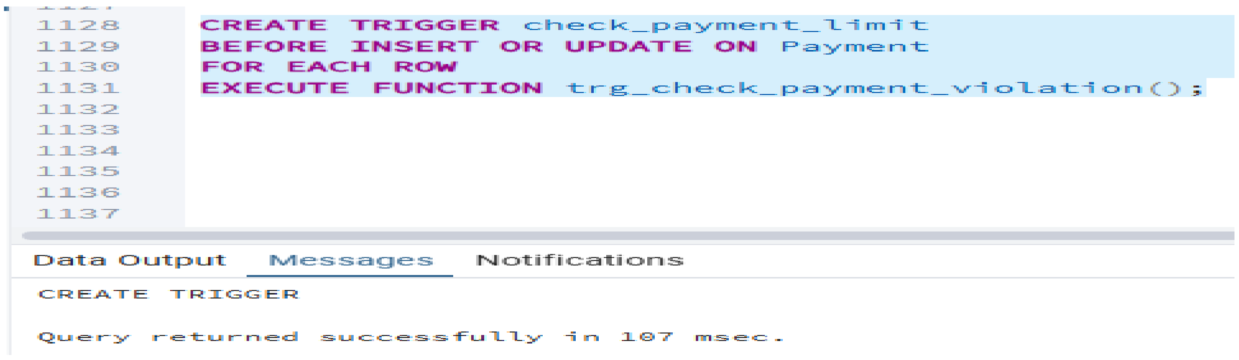


```
---\ . Create a BEFORE INSERT OR UPDATE trigger on Payment (or relevant table) th
---\application error when fn_should_alert returns 1.
CREATE OR REPLACE FUNCTION trg_check_payment_violation()
RETURNS TRIGGER AS $$
BEGIN
    -- Call the alert function with the incoming member_id
    IF fn_should_alert(NEW.member_id) = 1 THEN
        RAISE EXCEPTION 'Business rule violation: member % exceeded allowed limit
    END IF;

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

Data Output Messages Notifications

```
CREATE FUNCTION

Query returned successfully in 81 msec.
```

Trigger on Payment Table



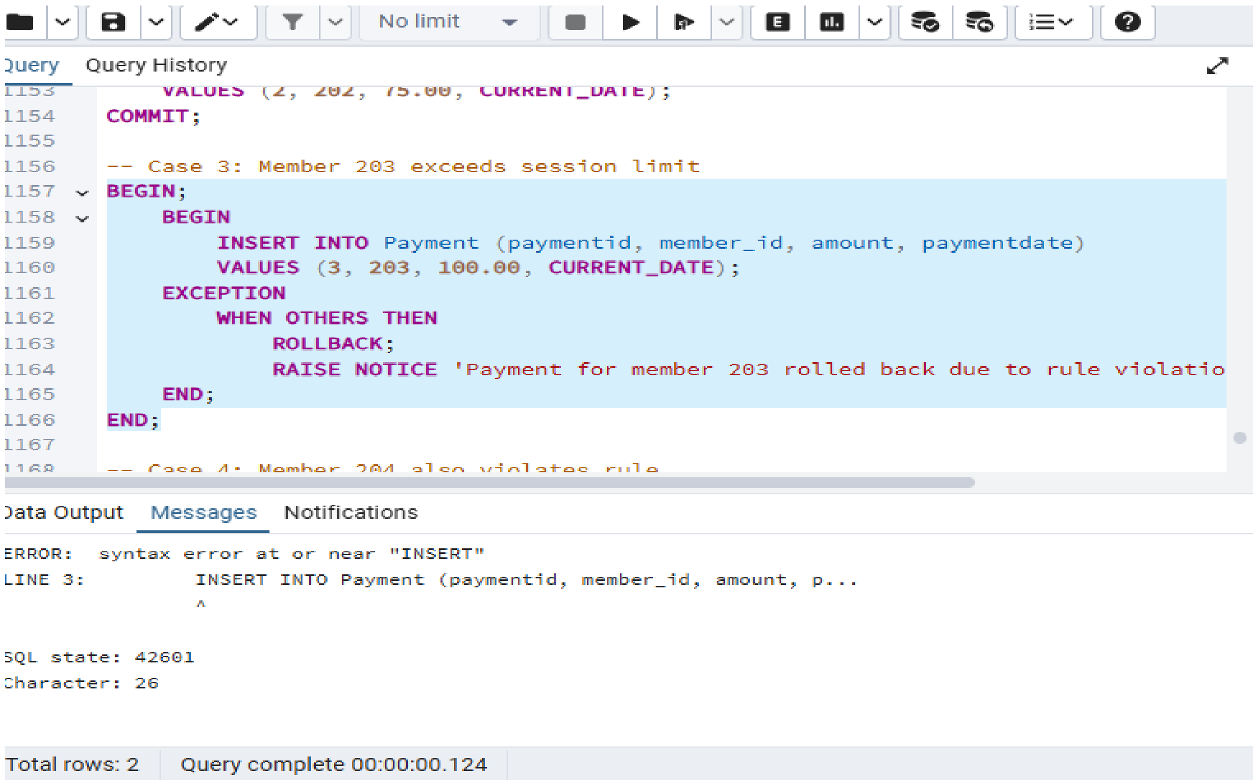
```
CREATE TRIGGER check_payment_limit
BEFORE INSERT OR UPDATE ON Payment
FOR EACH ROW
EXECUTE FUNCTION trg_check_payment_violation();
```

Data Output Messages Notifications

```
CREATE TRIGGER

Query returned successfully in 107 msec.
```

B10.d: . Demonstrate 2 failing and 2 passing DML cases; rollback the failing ones so total committed rows remain within the ≤10 budget.

* 2 Passing DML Cases



```
    VALUES (2, 202, 75.00, CURRENT_DATE);
COMMIT;

-- Case 3: Member 203 exceeds session limit
BEGIN;
    BEGIN
        INSERT INTO Payment (paymentid, member_id, amount, paymentdate)
        VALUES (3, 203, 100.00, CURRENT_DATE);
    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK;
            RAISE NOTICE 'Payment for member 203 rolled back due to rule violatio
    END;
END;

-- Case 4: Member 204 also violates rule
```

```
ERROR:  syntax error at or near "INSERT"
LINE 3:         INSERT INTO Payment (paymentid, member_id, amount, p...
                ^

SQL state: 42601
Character: 26
```

Total rows: 2   Query complete 00:00:00.124