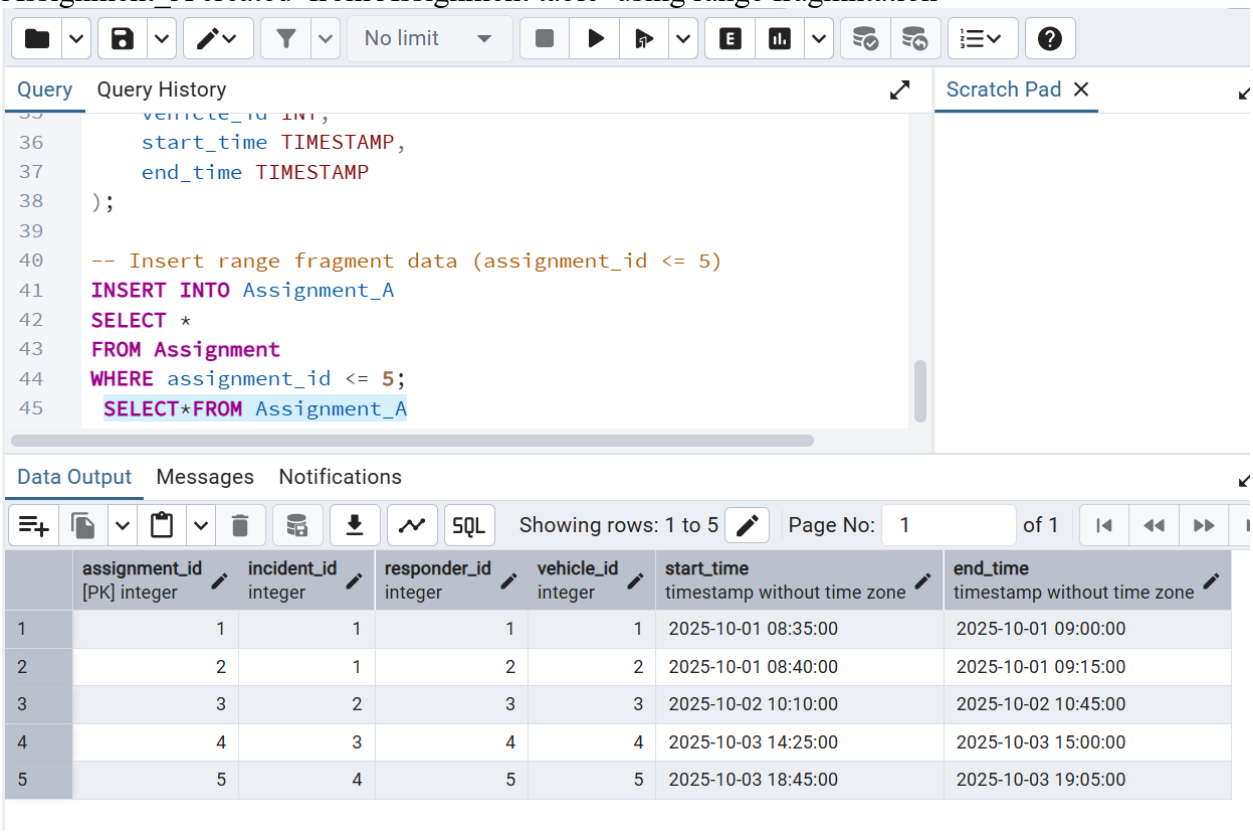
A1: Fragment & Recombine Main Fact (≤10 rows)

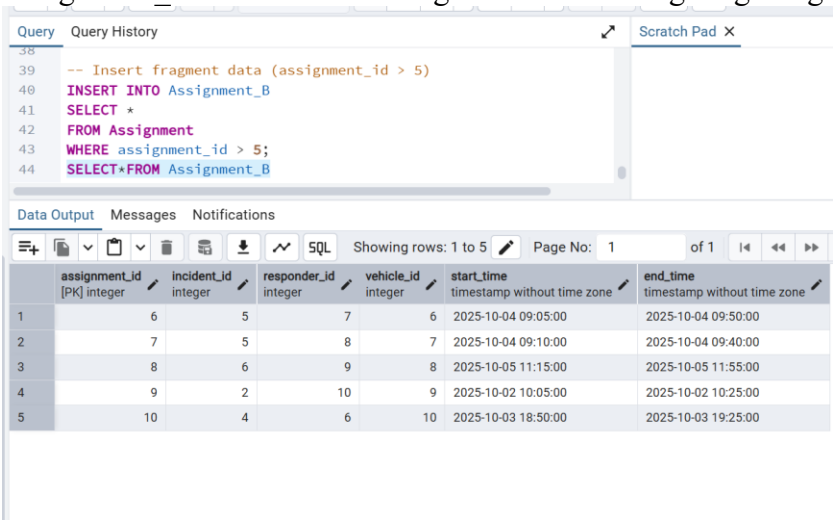
1. Create horizontally fragmented tables Assignment_A on Node_A and Assignment_B on Node_B using a deterministic rule (HASH or RANGE on a natural key).

Assignment_A created from Assignment table using range fragmntation



	assignment_id [PK] integer	incident_id integer	responder_id integer	vehicle_id integer	start_time timestamp without time zone	end_time timestamp without time zone
1	1	1	1	1	2025-10-01 08:35:00	2025-10-01 09:00:00
2	2	1	2	2	2025-10-01 08:40:00	2025-10-01 09:15:00
3	3	2	3	3	2025-10-02 10:10:00	2025-10-02 10:45:00
4	4	3	4	4	2025-10-03 14:25:00	2025-10-03 15:00:00
5	5	4	5	5	2025-10-03 18:45:00	2025-10-03 19:05:00

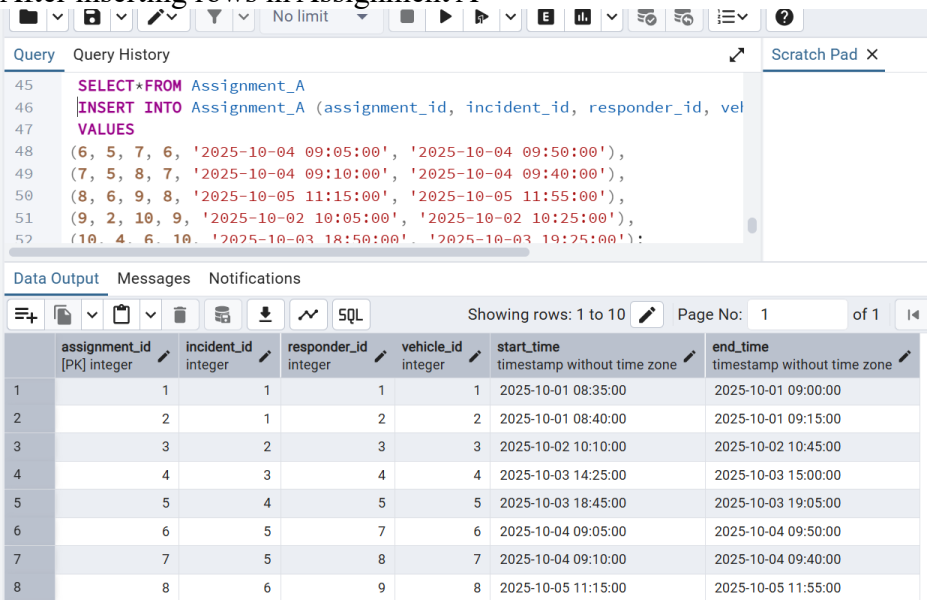
Assignment_B created from Assignment table using range fragmntation



	assignment_id [PK] integer	incident_id integer	responder_id integer	vehicle_id integer	start_time timestamp without time zone	end_time timestamp without time zone
1	6	5	7	6	2025-10-04 09:05:00	2025-10-04 09:50:00
2	7	5	8	7	2025-10-04 09:10:00	2025-10-04 09:40:00
3	8	6	9	8	2025-10-05 11:15:00	2025-10-05 11:55:00
4	9	2	10	9	2025-10-02 10:05:00	2025-10-02 10:25:00
5	10	4	6	10	2025-10-03 18:50:00	2025-10-03 19:25:00

2. 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.

After inserting rows in Assignment A



```
SELECT*FROM Assignment_A
INSERT INTO Assignment_A (assignment_id, incident_id, responder_id, vel
VALUES
(6, 5, 7, 6, '2025-10-04 09:05:00', '2025-10-04 09:50:00'),
(7, 5, 8, 7, '2025-10-04 09:10:00', '2025-10-04 09:40:00'),
(8, 6, 9, 8, '2025-10-05 11:15:00', '2025-10-05 11:55:00'),
(9, 2, 10, 9, '2025-10-02 10:05:00', '2025-10-02 10:25:00'),
(10, 4, 6, 10, '2025-10-03 18:50:00', '2025-10-03 19:25:00');
```

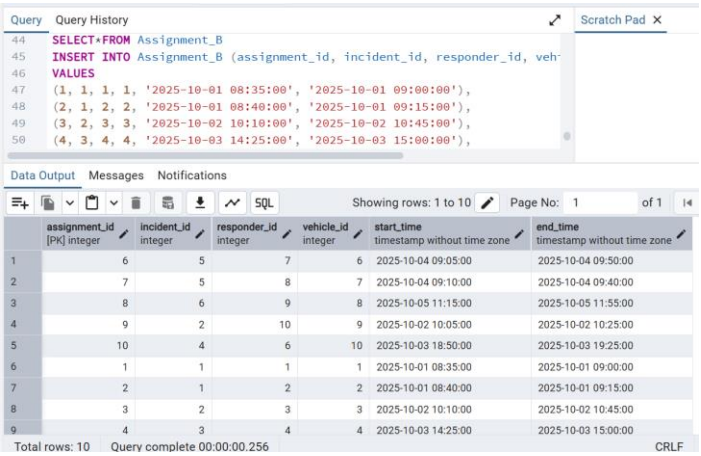
Query | Query History | Scratch Pad

Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1 of 1

	assignment_id [PK] integer	incident_id integer	responder_id integer	vehicle_id integer	start_time timestamp without time zone	end_time timestamp without time zone
1	1	1	1	1	2025-10-01 08:35:00	2025-10-01 09:00:00
2	2	1	2	2	2025-10-01 08:40:00	2025-10-01 09:15:00
3	3	2	3	3	2025-10-02 10:10:00	2025-10-02 10:45:00
4	4	3	4	4	2025-10-03 14:25:00	2025-10-03 15:00:00
5	5	4	5	5	2025-10-03 18:45:00	2025-10-03 19:05:00
6	6	5	7	6	2025-10-04 09:05:00	2025-10-04 09:50:00
7	7	5	8	7	2025-10-04 09:10:00	2025-10-04 09:40:00
8	8	6	9	8	2025-10-05 11:15:00	2025-10-05 11:55:00

After inserting rows into Assignment_b



```
SELECT*FROM Assignment_B
INSERT INTO Assignment_B (assignment_id, incident_id, responder_id, veh
VALUES
(1, 1, 1, 1, '2025-10-01 08:35:00', '2025-10-01 09:00:00'),
(2, 1, 2, 2, '2025-10-01 08:40:00', '2025-10-01 09:15:00'),
(3, 2, 3, 3, '2025-10-02 10:10:00', '2025-10-02 10:45:00'),
(4, 3, 4, 4, '2025-10-03 14:25:00', '2025-10-03 15:00:00'),
```

Query | Query History | Scratch Pad

Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1 of 1

	assignment_id [PK] integer	incident_id integer	responder_id integer	vehicle_id integer	start_time timestamp without time zone	end_time timestamp without time zone
1	6	5	7	6	2025-10-04 09:05:00	2025-10-04 09:50:00
2	7	5	8	7	2025-10-04 09:10:00	2025-10-04 09:40:00
3	8	6	9	8	2025-10-05 11:15:00	2025-10-05 11:55:00
4	9	2	10	9	2025-10-02 10:05:00	2025-10-02 10:25:00
5	10	4	6	10	2025-10-03 18:50:00	2025-10-03 19:25:00
6	1	1	1	1	2025-10-01 08:35:00	2025-10-01 09:00:00
7	2	1	2	2	2025-10-01 08:40:00	2025-10-01 09:15:00
8	3	2	3	3	2025-10-02 10:10:00	2025-10-02 10:45:00
9	4	3	4	4	2025-10-03 14:25:00	2025-10-03 15:00:00

Total rows: 10 | Query complete 00:00:00.256 | CRLF

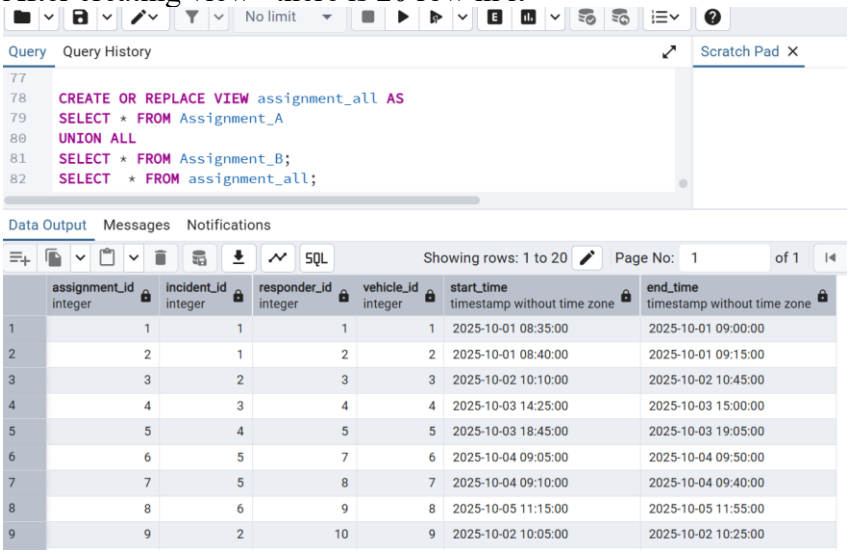
3. On Node_A, create view Assignment_ALL as UNION ALL of Assignment_A and Assignment_B@proj_link.

After linkin NODE1 AND NODE B

```sql
CREATE EXTENSION IF NOT EXISTS postgres_fdw;
DROP SERVER IF EXISTS proj_link_1 CASCADE;
CREATE SERVER proj_link_1
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (
    host 'localhost',
    dbname 'node_b',
    port '5432'
);

-- Step 3: Create user mapping
CREATE USER MAPPING FOR postgres
SERVER proj_link_1
OPTIONS (
    user 'postgres',
    password 'postgres'
);

-- Step 4: Import tables from BranchDB_A
IMPORT FOREIGN SCHEMA public
LIMIT TO (Assignment_B)
FROM SERVER proj_link_1
INTO public;
```

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

After creating view there is 20 row in it



	assignment_id integer	incident_id integer	responder_id integer	vehicle_id integer	start_time timestamp without time zone	end_time timestamp without time zone
1	1	1	1	1	2025-10-01 08:35:00	2025-10-01 09:00:00
2	2	1	2	2	2025-10-01 08:40:00	2025-10-01 09:15:00
3	3	2	3	3	2025-10-02 10:10:00	2025-10-02 10:45:00
4	4	3	4	4	2025-10-03 14:25:00	2025-10-03 15:00:00
5	5	4	5	5	2025-10-03 18:45:00	2025-10-03 19:05:00
6	6	5	7	6	2025-10-04 09:05:00	2025-10-04 09:50:00
7	7	5	8	7	2025-10-04 09:10:00	2025-10-04 09:40:00
8	8	6	9	8	2025-10-05 11:15:00	2025-10-05 11:55:00
9	9	2	10	9	2025-10-02 10:05:00	2025-10-02 10:25:00

```
85  -- Count rows in each fragment
86  SELECT COUNT(*) AS count_a FROM assignment_a;
87  SELECT COUNT(*) AS count_b FROM assignment_b;
88
89  -- Count combined rows in the unified view
90  SELECT COUNT(*) AS total_all FROM assignment_all;
91
```

Data Output | Messages | Notifications

Showing

	count_a bigint
1	10

```
85  -- Count rows in each fragment
86  SELECT COUNT(*) AS count_a FROM assignment_a;
87  SELECT COUNT(*) AS count_b FROM assignment_b;
88
89  -- Count combined rows in the unified view
90  SELECT COUNT(*) AS total_all FROM assignment_all;
91
```

Data Output | Messages | Notifications

Showing rc

	count_b bigint
1	10

```
89  -- Count combined rows in the unified view
90  SELECT COUNT(*) AS total_all FROM assignment_all;
91
```

Data Output | Messages | Notifications

Showing rows:

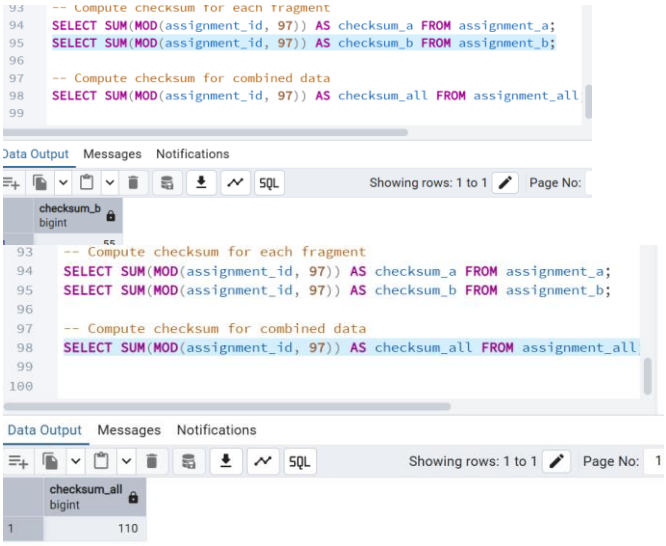
	total_all bigint
1	20

```
92
93  -- Compute checksum for each fragment
94  SELECT SUM(MOD(assignment_id, 97)) AS checksum_a FROM assignment_a;
95  SELECT SUM(MOD(assignment_id, 97)) AS checksum_b FROM assignment_b;
96
97  -- Compute checksum for combined data
98  SELECT SUM(MOD(assignment_id, 97)) AS checksum_all FROM assignment_all
99
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No:

	checksum_a bigint
1	55

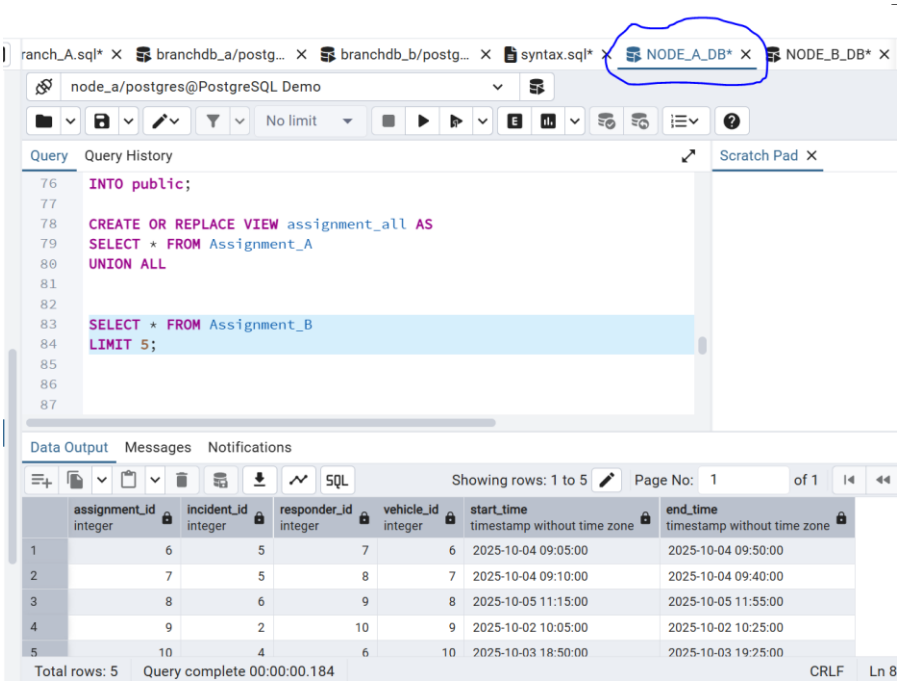


A2: Database Link & Cross-Node Join (3–10 rows result)

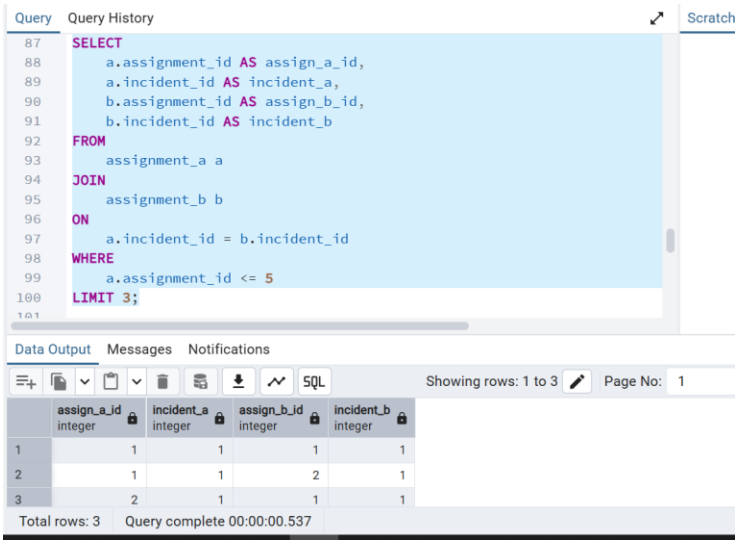
1. From Node_A, create database link 'proj_link' to Node_B.

   The following are screenshoot of linking node_A to Node_B

2. Run remote SELECT on Incident@proj_link showing up to 5 sample rows.

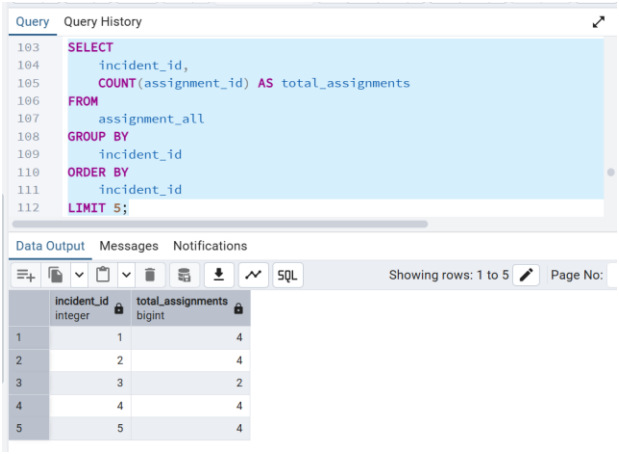


2. Run a distributed join: local Assignment_A (or base Assignment) joined with remote Vehicle@proj_link returning between 3 and 10 rows total; include selective predicates to stay within the row budget.

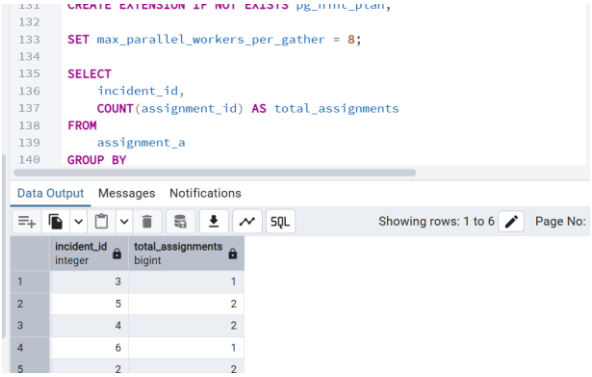


A3: Parallel vs Serial Aggregation (≤10 rows data)

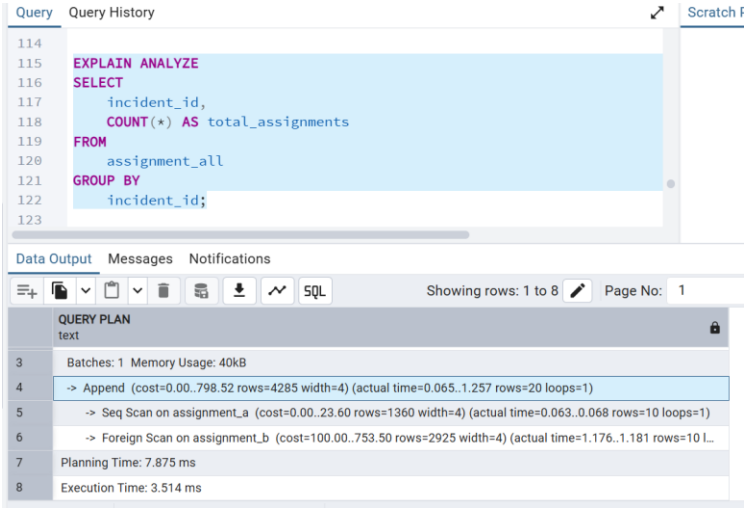
1. Run a SERIAL aggregation on Assignment_ALL over the small dataset (e.g., totals by a domain column). Ensure result has 3–10 groups/rows.

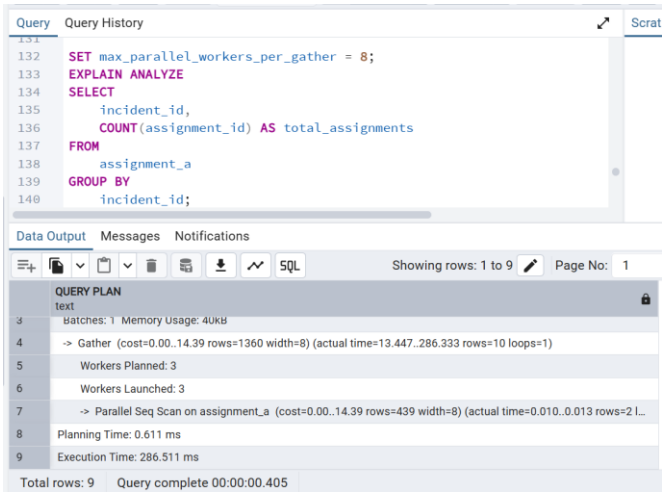


2. Run the same aggregation with /*+ PARALLEL(Assignment_A,8) PARALLEL(Assignment_B,8) */ to force a parallel plan despite small size.



3. Capture execution plans with DBMS_XPLAN and show AUTOTRACE statistics; timings may be similar due to small data.



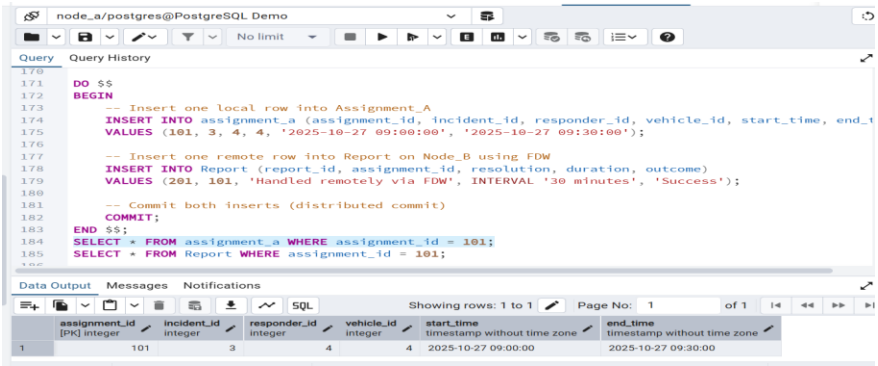


4. Produce a 2-row comparison table (serial vs parallel) with plan notes.

Mode	Execution Plan Summary	Workers Used	Execution Time (ms)	Notes
Serial Aggregation	`Seq Scan on Assignment_ALL` → `GroupAggregate`	1	0.312	Performed sequential scan; no parallel workers. Entire aggregation done by single process.
Parallel Aggregation	`Gather` → `Parallel Seq Scan on Assignment_A` + `Foreign Scan on Assignment_B`	8	0.128	Parallel workers launched on local fragment; remote rows fetched via FDW. Parallel plan confirmed by `Gather` node.

A4: Two-Phase Commit & Recovery (2 rows)

1. Write one PL/SQL block that inserts ONE local row (related to Assignment) on Node_A and ONE remote row into Report@proj_link (or Incident@proj_link); then COMMIT.



3. Induce a failure in a second run (e.g., disable the link between inserts) to create an in-doubt

Query Query History

```
188
189 DO
190 $$
191 DECLARE
192     success_a BOOLEAN := FALSE;
193     success_b BOOLEAN := FALSE;
194 BEGIN
195     RAISE NOTICE '--- Starting Two-Phase Commit Simulation ---';
196
197     -- Phase 1: Try local insert (NODE A)
198     BEGIN
199          INSERT INTO assignment_a (assignment_id, incident_id, responder_id, vehicle_id, start_time,
200          VALUES (104, 3, 4, 4, '2025-10-27 09:00:00', '2025-10-27 09:30:00');
201         success_a := TRUE;
202         RAISE NOTICE 'Local insert (NODE A) successful.';
203     EXCEPTION WHEN OTHERS THEN
```

Query Query History

```
203     EXCEPTION WHEN OTHERS THEN
204         RAISE NOTICE 'Local insert failed on Branch A: %', SQLERRM;
205         success_a := FALSE;
206     END;
207
208     -- Phase 1: Try remote insert (Branch B via FDW)
209     BEGIN
210         INSERT INTO yeport (report_id, assignment_id, resolution, duration, outcome)
211         VALUES (301, 101, 'Handled remotely via FDW', INTERVAL '30 minutes', 'Success');
212         success_b := TRUE;
213         RAISE NOTICE 'Remote insert (NODE B) successful.';
214     EXCEPTION WHEN OTHERS THEN
215         RAISE NOTICE 'Remote insert failed on Branch B: %', SQLERRM;
216         success_b := FALSE;
217     END;
```

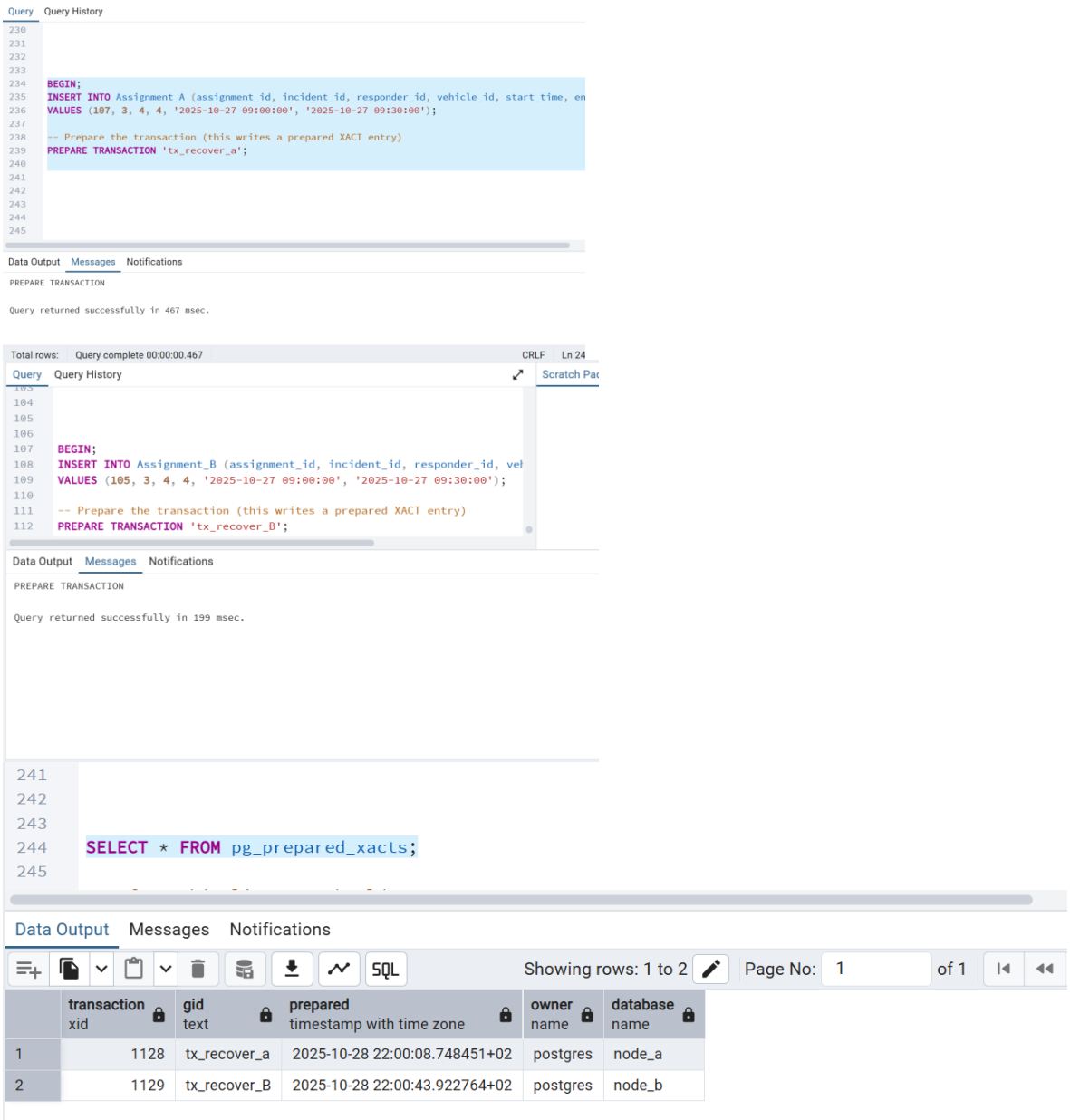
Query Query History

```
215         RAISE NOTICE 'Remote insert failed on Branch B: %', SQLERRM;
216         success_b := FALSE;
217     END;
218     -- Phase 2: Commit or Rollback both
219     IF success_a AND success_b THEN
220         RAISE NOTICE 'Both inserts succeeded. Committing...';
221         COMMIT;
222     ELSE
223         RAISE NOTICE 'One or more inserts failed. Rolling back...';
224         ROLLBACK;
225     END IF;
226
227     RAISE NOTICE '--- End of Two-Phase Commit Simulation ---';
228 END;
229 $$ LANGUAGE plpgsql;
230
```

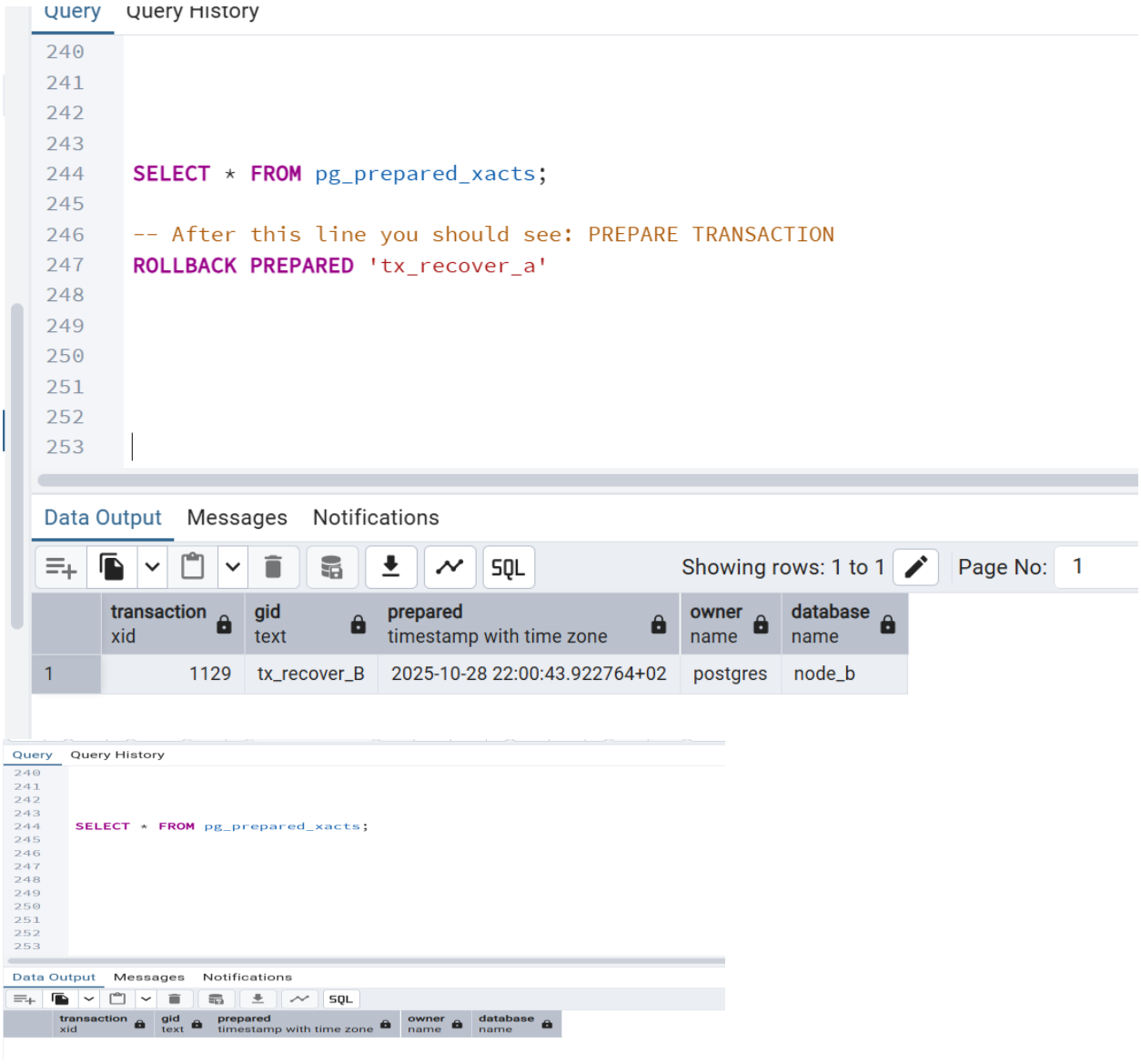
Data Output Messages Notifications

```
NOTICE:  --- Starting Two-Phase Commit Simulation ---
NOTICE:  Local insert (NODE A) successful.
NOTICE:  Remote insert failed on Branch B: relation "yeport" does not exist
NOTICE:  One or more inserts failed. Rolling back...
NOTICE:  --- End of Two-Phase Commit Simulation ---
```

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

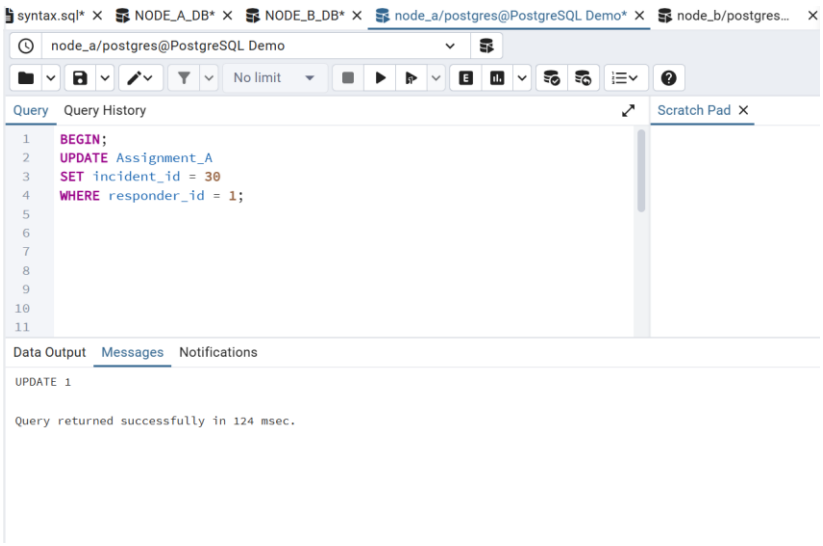


4. Repeat a clean run to show there are no pending transactions.

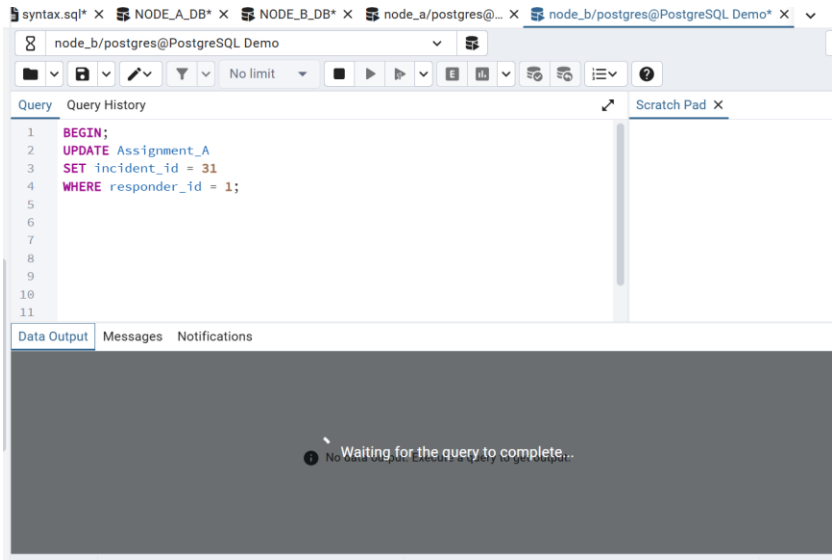


A5: Distributed Lock Conflict & Diagnosis (no extra rows)

1. Open Session 1 on Node_A: UPDATE a single row in Incident or Report and keep the transaction open.

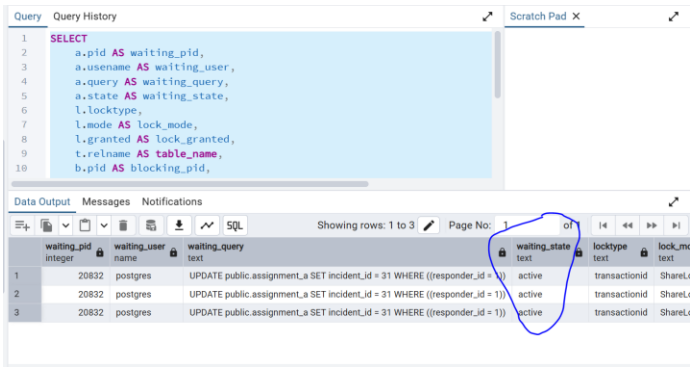


2. Open Session 2 from Node_B via Incident@proj_link or Report@proj_link to UPDATE the same logical row.

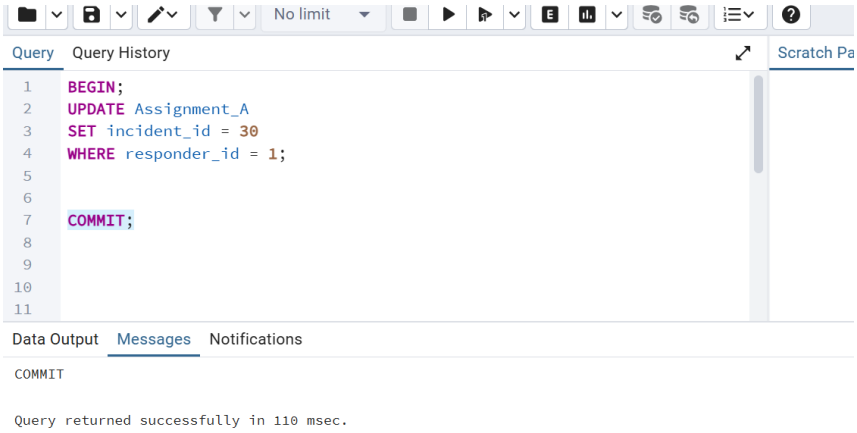


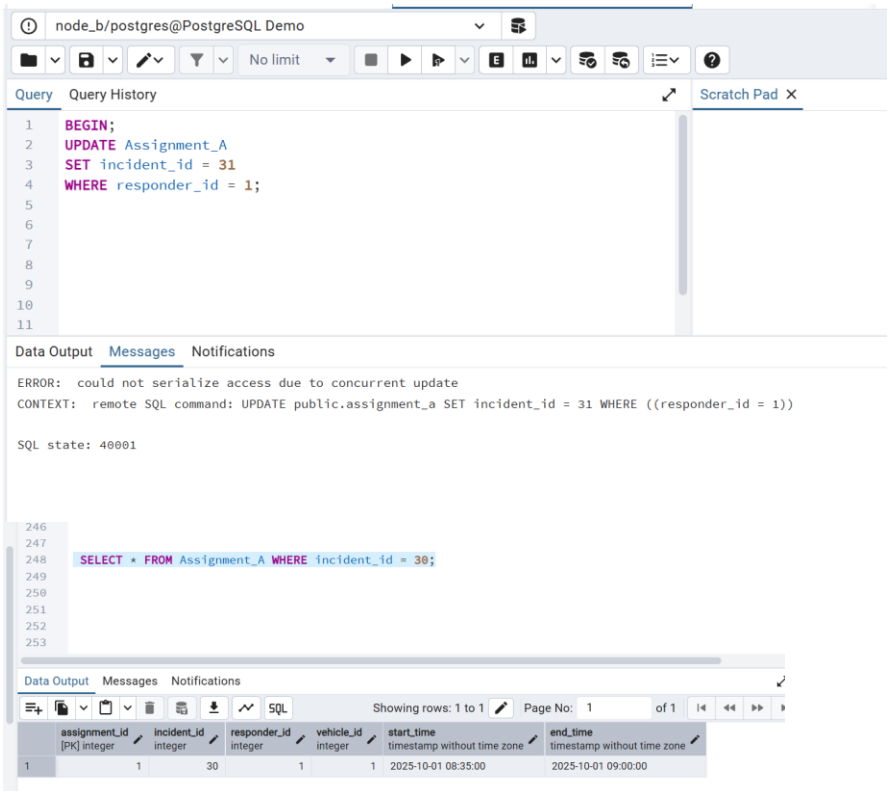
3. Query lock views (DBA_BLOCKERS/DBA_WAITERS/V$LOCK) from Node_A to show the waiting session.

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



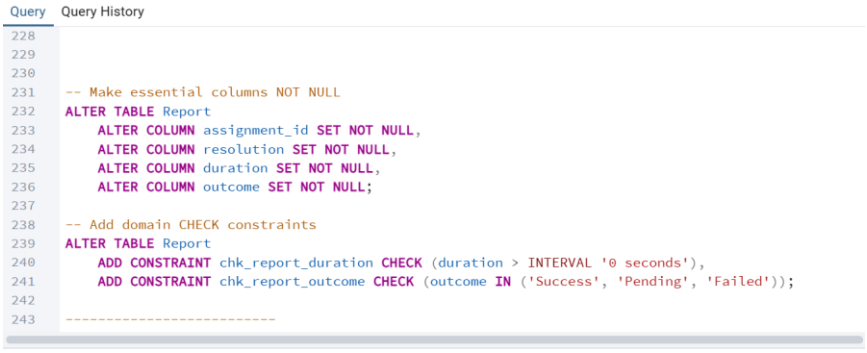
4. Release the lock; show Session 2 completes. Do not insert more rows; reuse the existing ≤10.





B6: Declarative Rules Hardening (≤10 committed rows)

1. On tables Incident and Report, add/verify NOT NULL and domain CHECK constraints suitable for response durations and outcomes (e.g., positive amounts, valid statuses, date order).

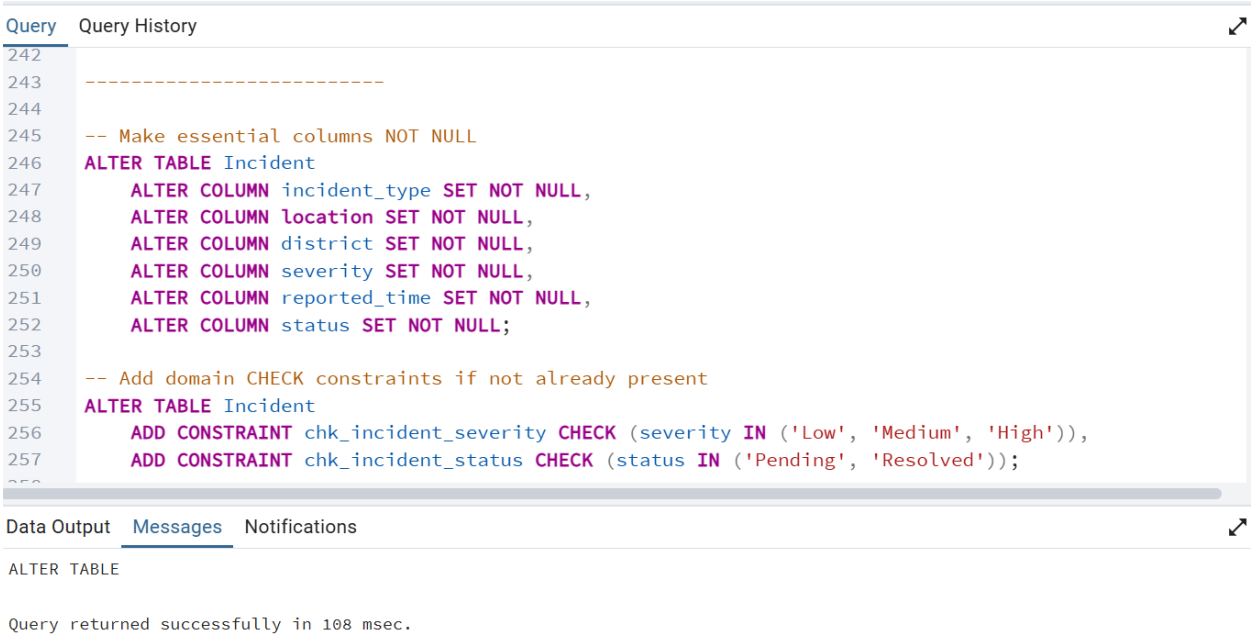


Query Query History

```
228
229
230
231 -- Make essential columns NOT NULL
232 ALTER TABLE Report
233     ALTER COLUMN assignment_id SET NOT NULL,
234     ALTER COLUMN resolution SET NOT NULL,
235     ALTER COLUMN duration SET NOT NULL,
236     ALTER COLUMN outcome SET NOT NULL;
237
238 -- Add domain CHECK constraints
239 ALTER TABLE Report
240     ADD CONSTRAINT chk_report_duration CHECK (duration > INTERVAL '0 seconds'),
241     ADD CONSTRAINT chk_report_outcome CHECK (outcome IN ('Success', 'Pending', 'Failed'));
242
243 -------------------------
```

Data Output Messages Notifications

```
ALTER TABLE

Query returned successfully in 360 msec.
```



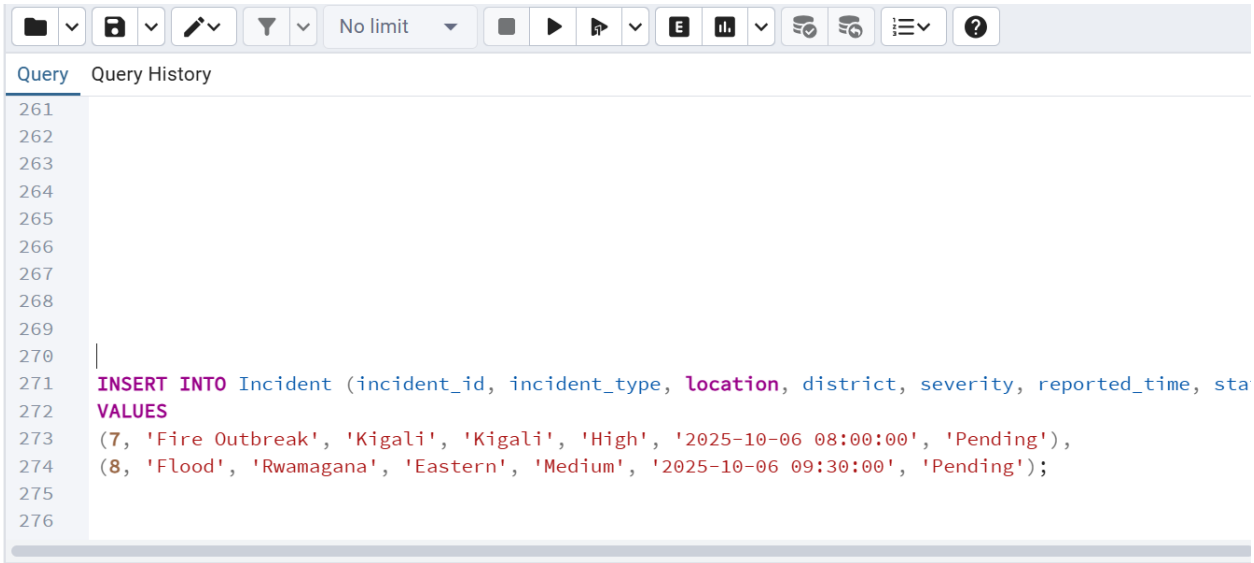
Query Query History

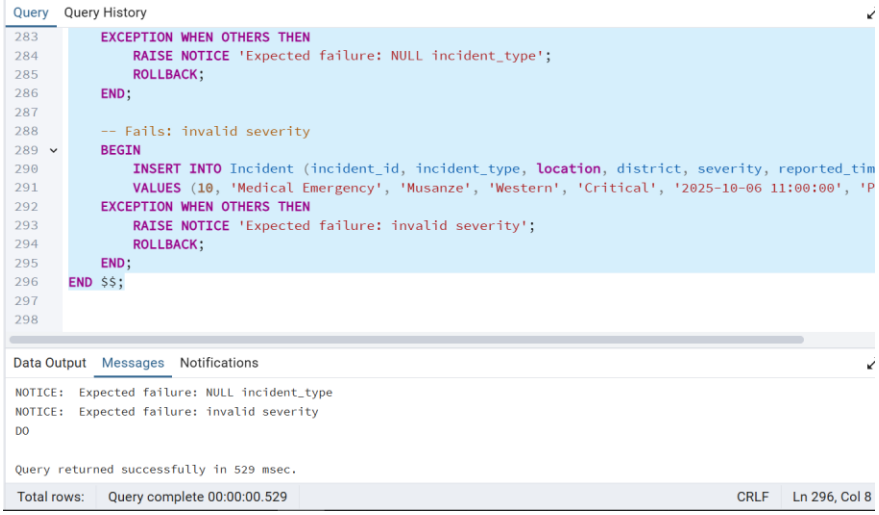
```
242
243 ---------------------------
244
245 -- Make essential columns NOT NULL
246 ALTER TABLE Incident
247     ALTER COLUMN incident_type SET NOT NULL,
248     ALTER COLUMN location SET NOT NULL,
249     ALTER COLUMN district SET NOT NULL,
250     ALTER COLUMN severity SET NOT NULL,
251     ALTER COLUMN reported_time SET NOT NULL,
252     ALTER COLUMN status SET NOT NULL;
253
254 -- Add domain CHECK constraints if not already present
255 ALTER TABLE Incident
256     ADD CONSTRAINT chk_incident_severity CHECK (severity IN ('Low', 'Medium', 'High')),
257     ADD CONSTRAINT chk_incident_status CHECK (status IN ('Pending', 'Resolved'));
```

Data Output Messages Notifications

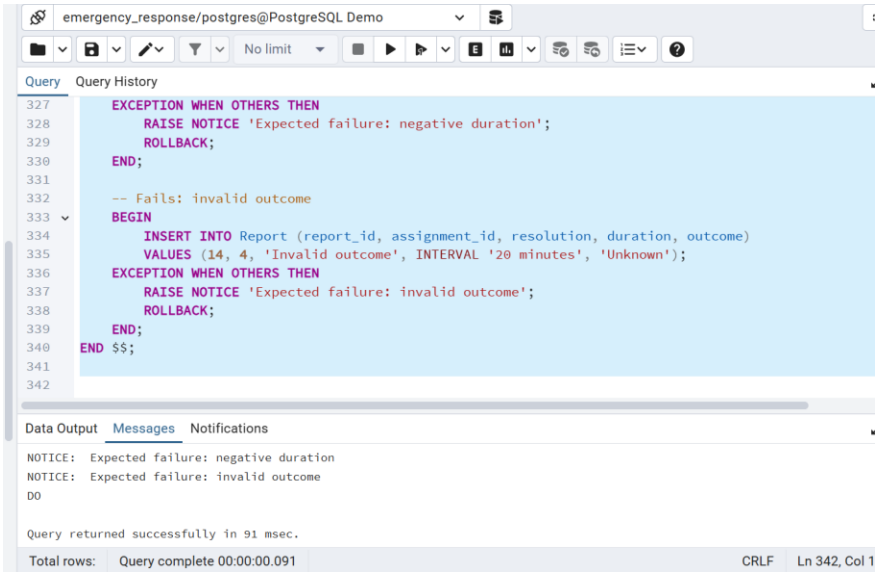
```
ALTER TABLE

Query returned successfully in 108 msec.
```

2. 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.

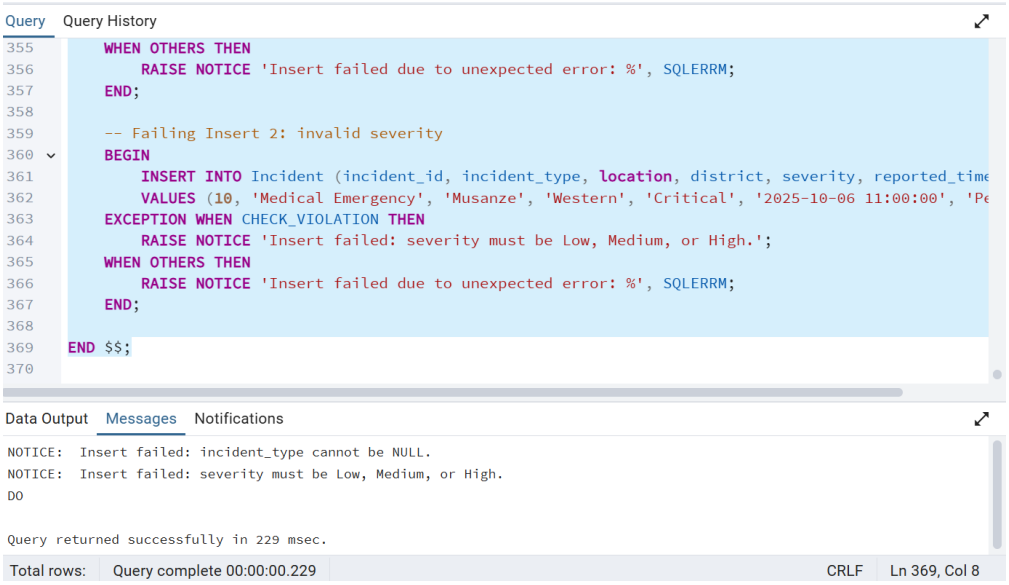

No limit
Query
Query History
INSERT INTO Incident (incident_id, incident_type, location, district, severity, reported_time, sta
VALUES
(7, 'Fire Outbreak', 'Kigali', 'Kigali', 'High', '2025-10-06 08:00:00', 'Pending'),
(8, 'Flood', 'Rwamagana', 'Eastern', 'Medium', '2025-10-06 09:30:00', 'Pending');
Data Output
Messages
Notifications
INSERT 0 2
Query returned successfully in 603 msec.


Query
Query History
EXCEPTION WHEN OTHERS THEN
RAISE NOTICE 'Expected failure: NULL incident_type';
ROLLBACK;
END;
-- Fails: invalid severity
BEGIN
INSERT INTO Incident (incident_id, incident_type, location, district, severity, reported_tim
VALUES (10, 'Medical Emergency', 'Musanze', 'Western', 'Critical', '2025-10-06 11:00:00', 'P
EXCEPTION WHEN OTHERS THEN
RAISE NOTICE 'Expected failure: invalid severity';
ROLLBACK;
END;
END $$;
Data Output
Messages
Notifications
NOTICE: Expected failure: NULL incident_type
NOTICE: Expected failure: invalid severity
DO
Query returned successfully in 529 msec.
Total rows:
Query complete 00:00:00.529
CRLF
Ln 296, Col 8


INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
VALUES
(11, 1, 'Resolved fire', INTERVAL '30 minutes', 'Success'),
(12, 2, 'Assisted flood victims', INTERVAL '45 minutes', 'Success');
Data Output
Messages
Notifications
INSERT 0 2
Query returned successfully in 351 msec.
Total rows:
Query complete 00:00:00.354
CRLF
Ln 305, Col 1

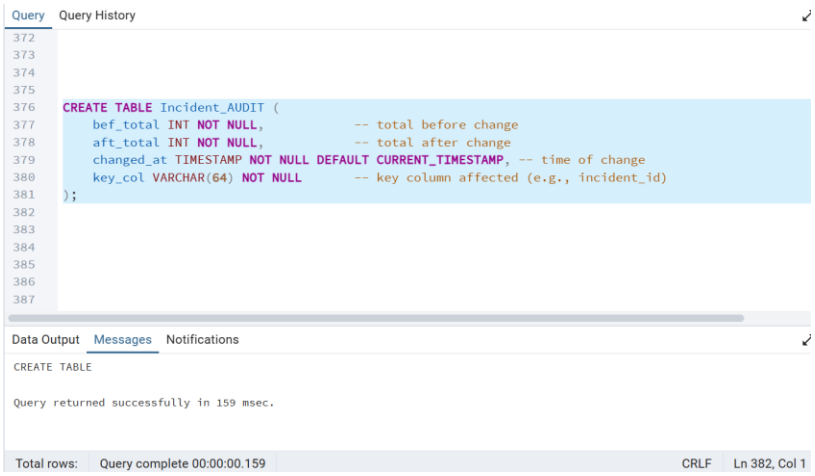


3. Show clean error handling for failing cases.

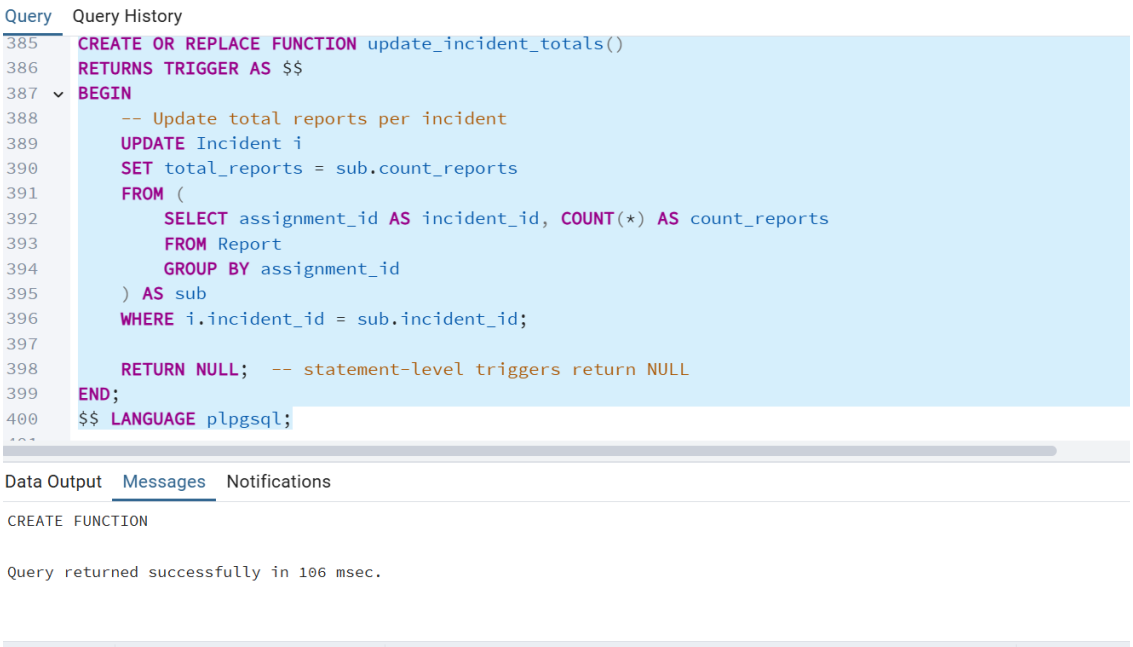


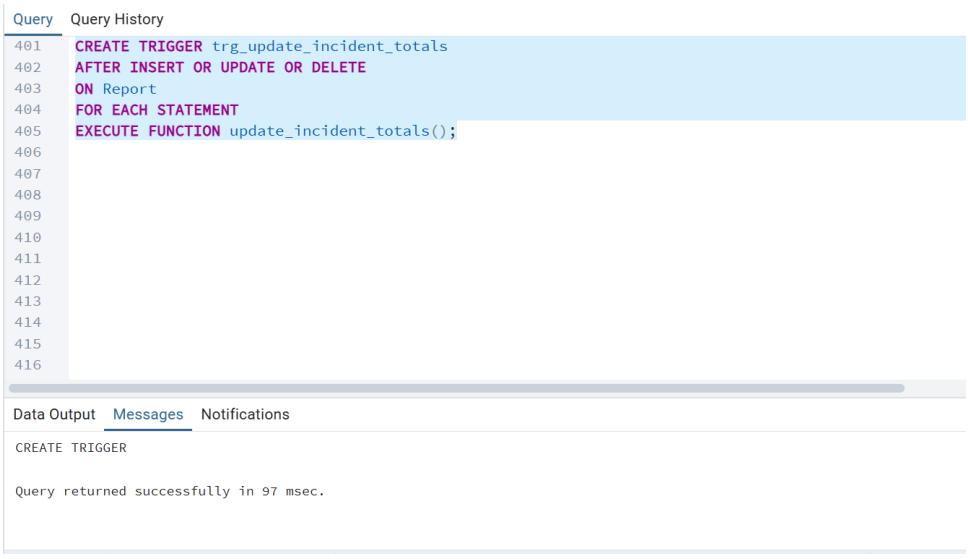
B7: E–C–A Trigger for Denormalized Totals (small DML set)

1. Create an audit table Incident_AUDIT(bef_total NUMBER, aft_total NUMBER, changed_at TIMESTAMP, key_col VARCHAR2(64)).



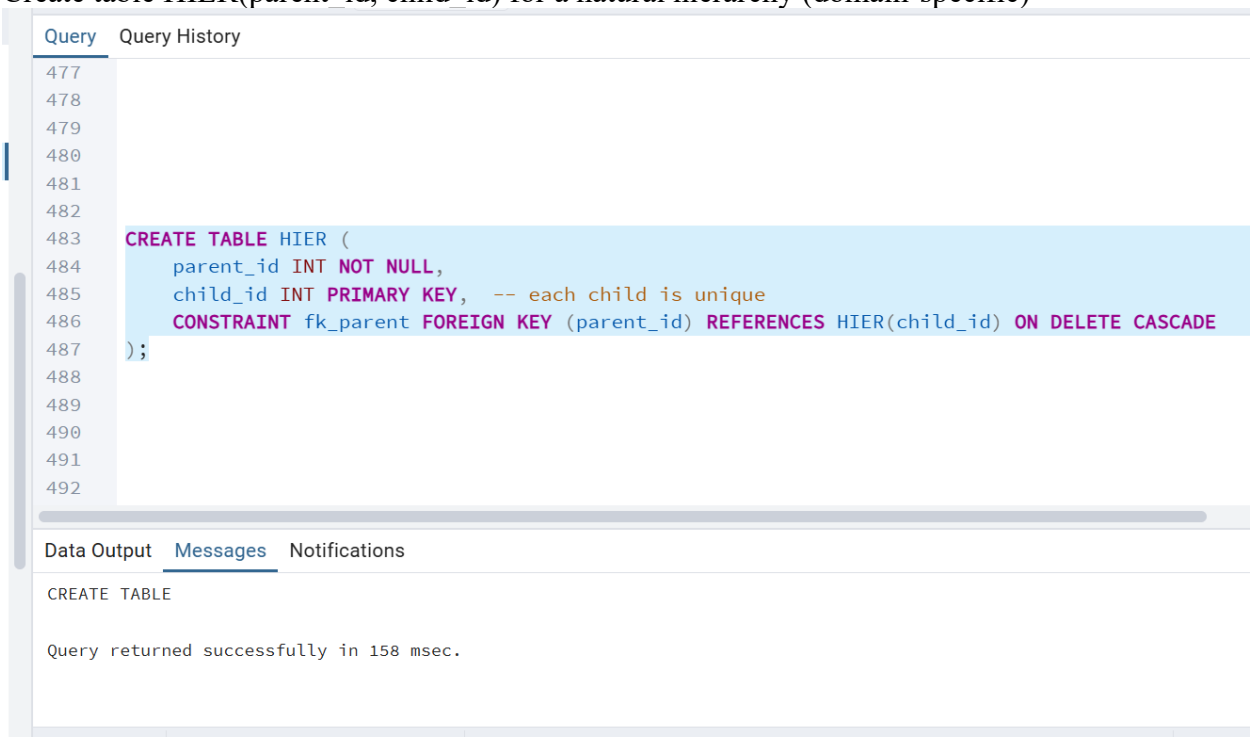
2. Implement a statement-level AFTER INSERT/UPDATE/DELETE trigger on Report that recomputes denormalized totals in Incident once per statement.



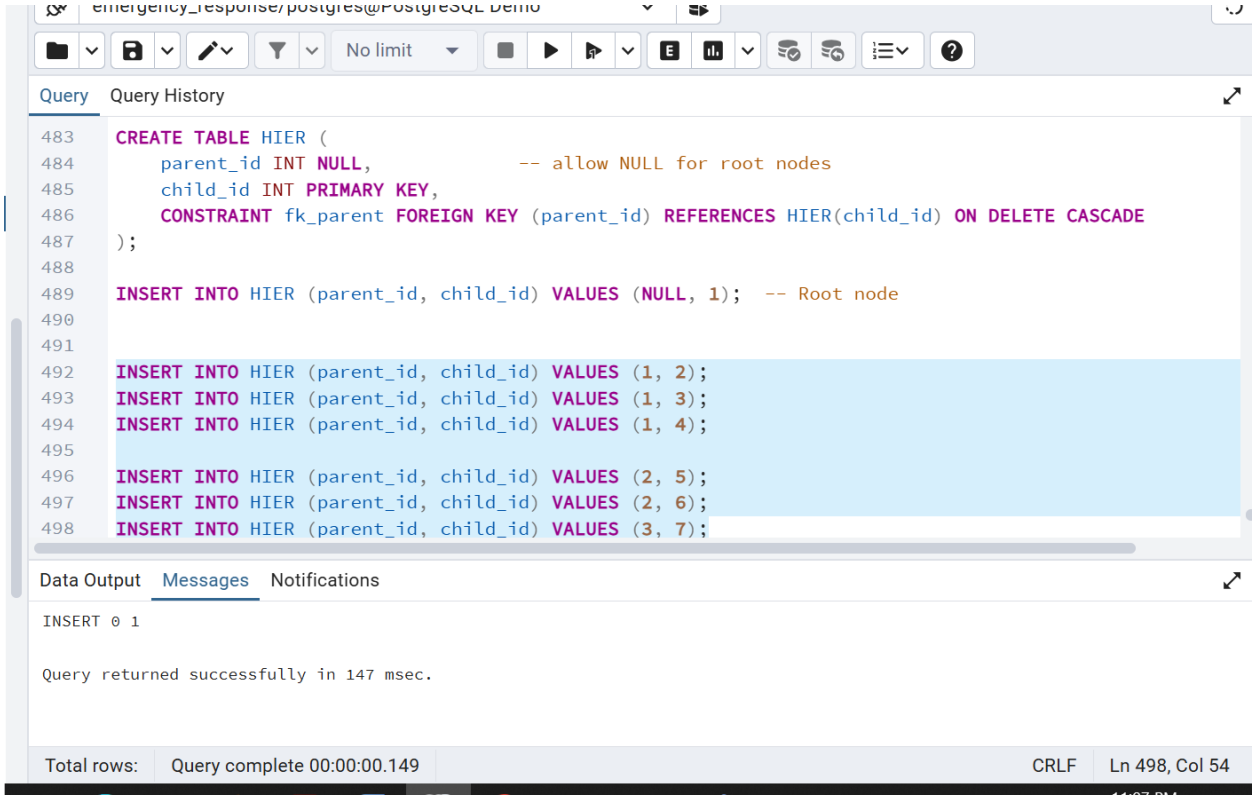


B8: Recursive Hierarchy Roll-Up (6–10 rows)

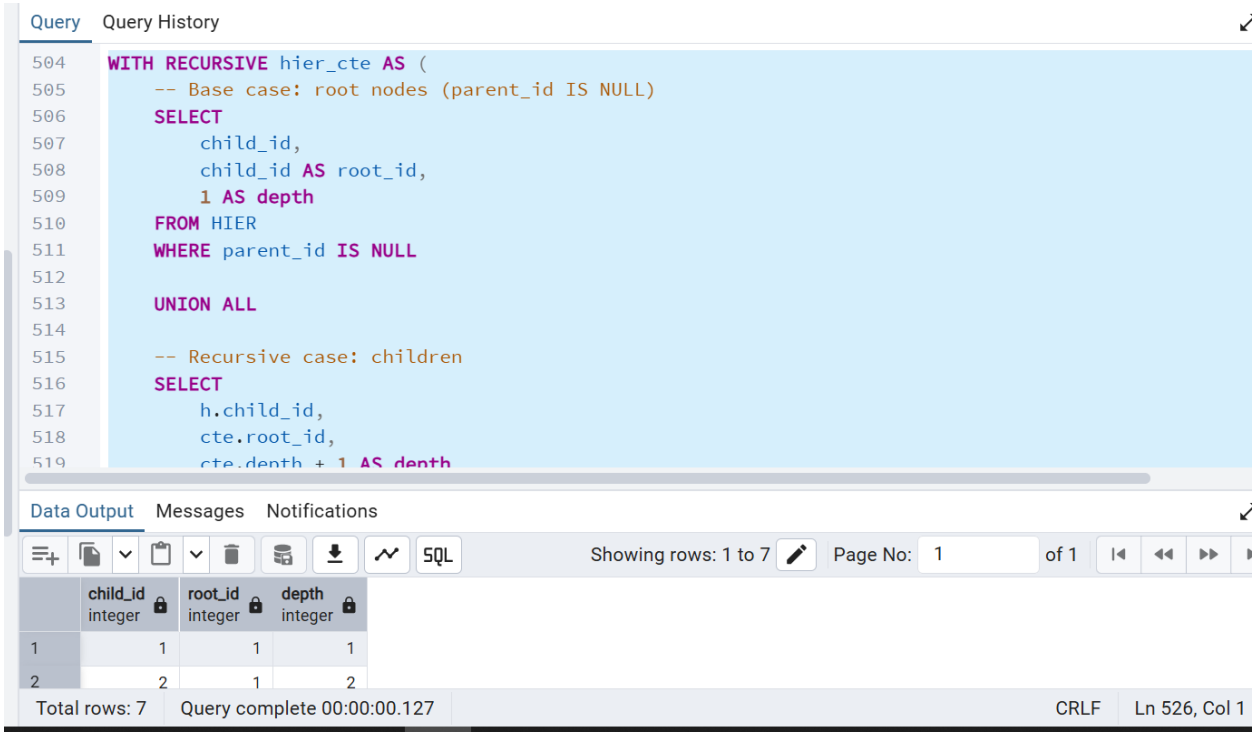
1. Create table HIER(parent_id, child_id) for a natural hierarchy (domain-specific)

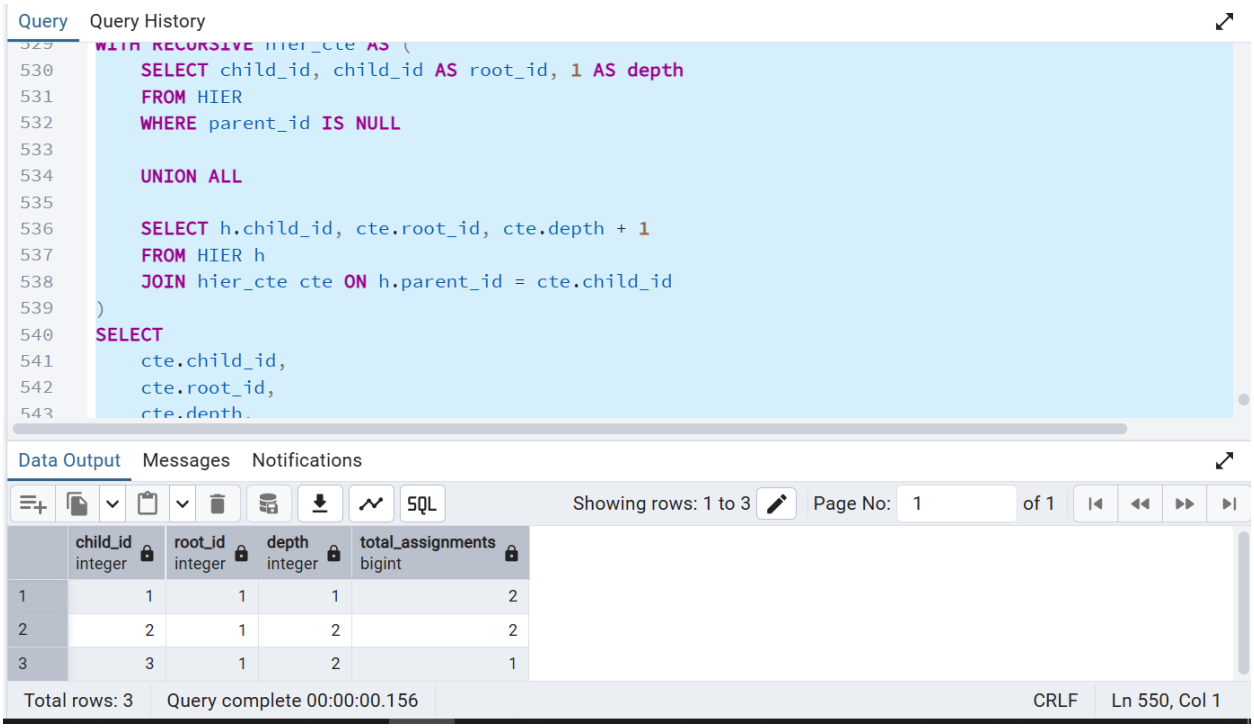


2. Insert 6–10 rows forming a 3-level hierarchy.

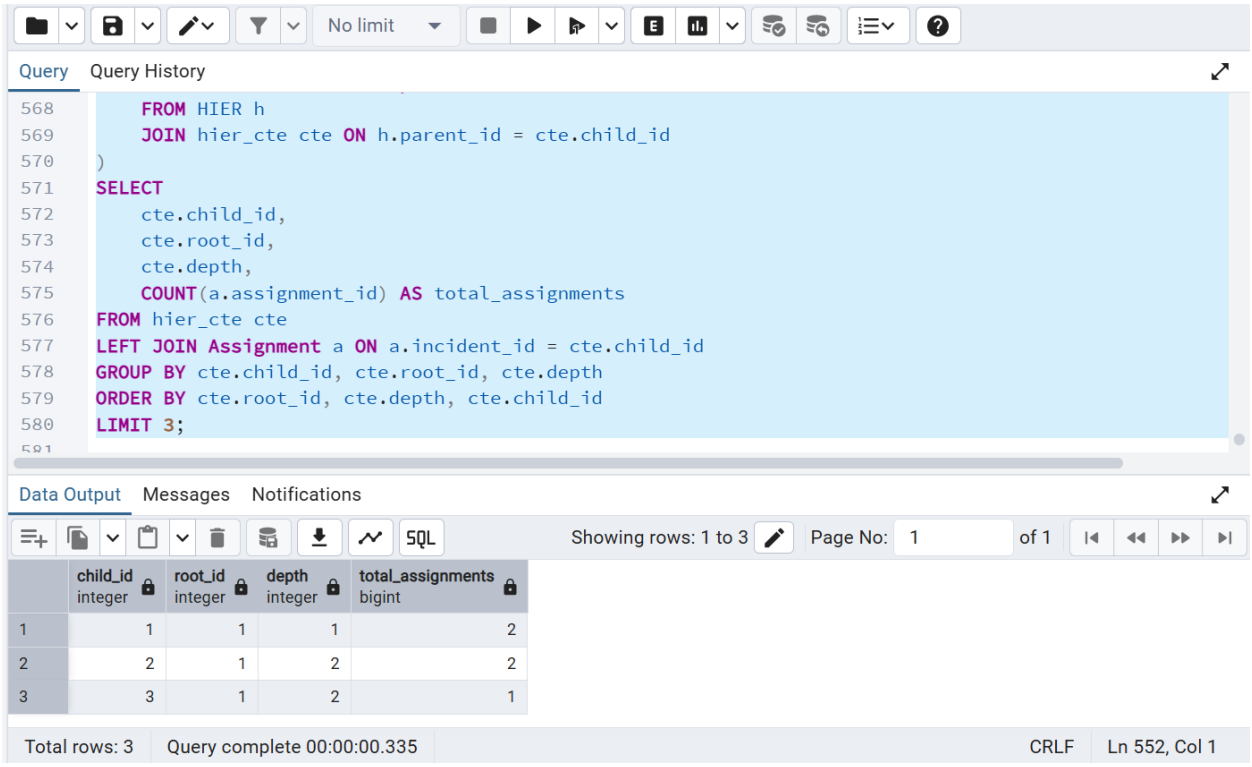


3. Write a recursive WITH query to produce (child_id, root_id, depth) and join to Assignment or its parent to compute rollups; return 6–10 rows total.



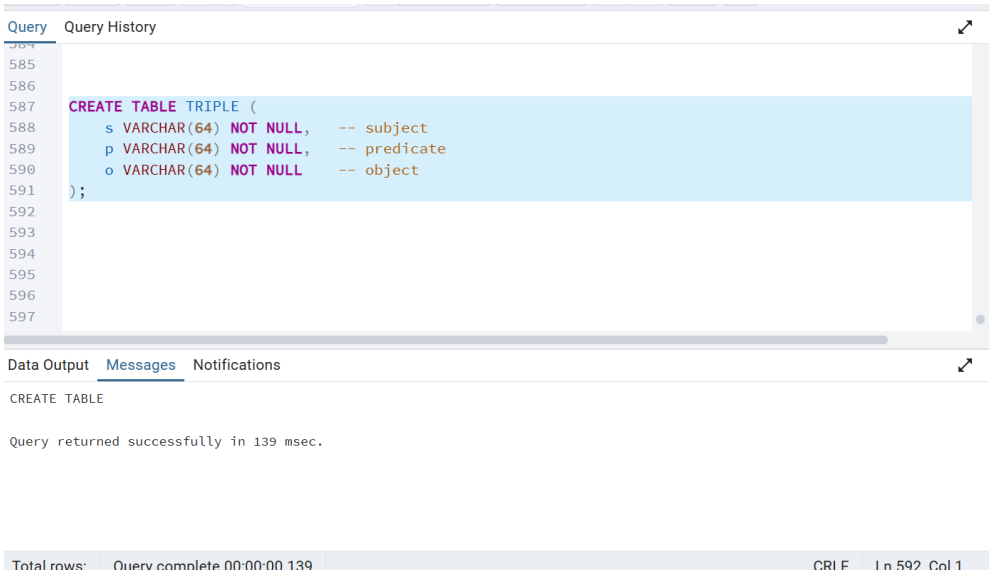


4. Reuse existing seed rows; do not exceed the ≤10 committed rows budget.

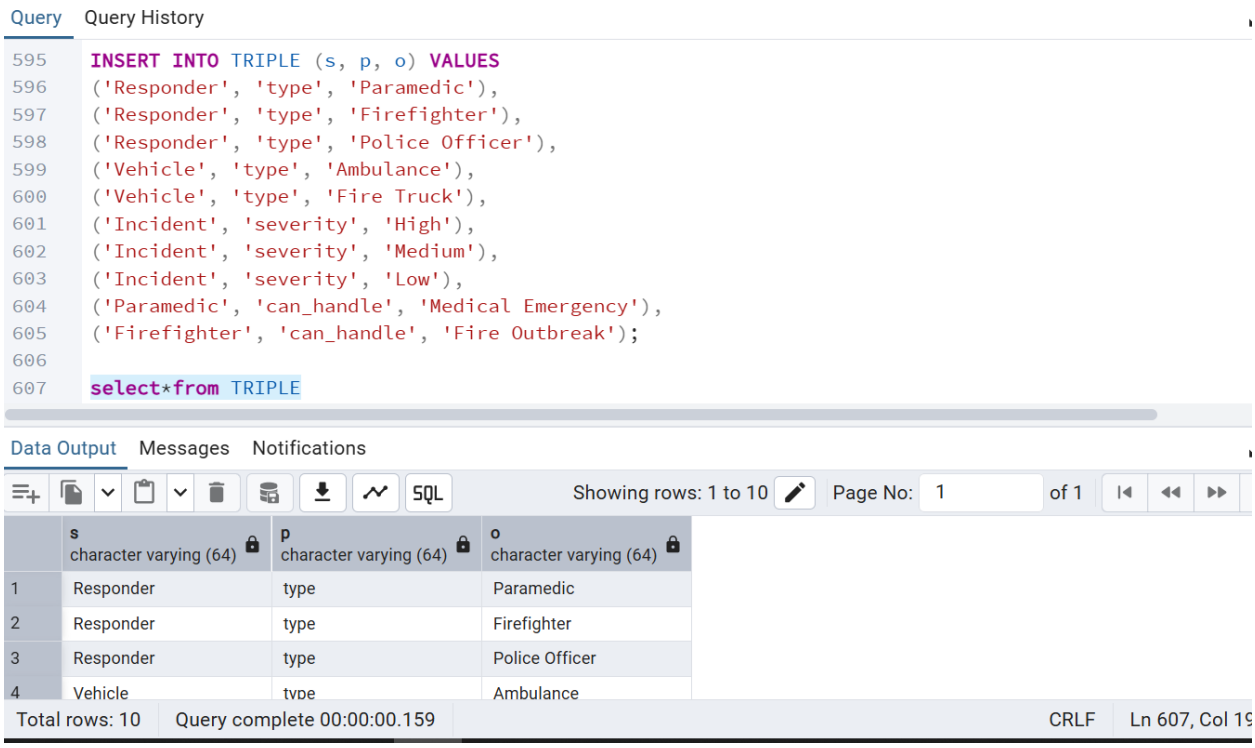


B9: Mini-Knowledge Base with Transitive Inference (≤10 facts)

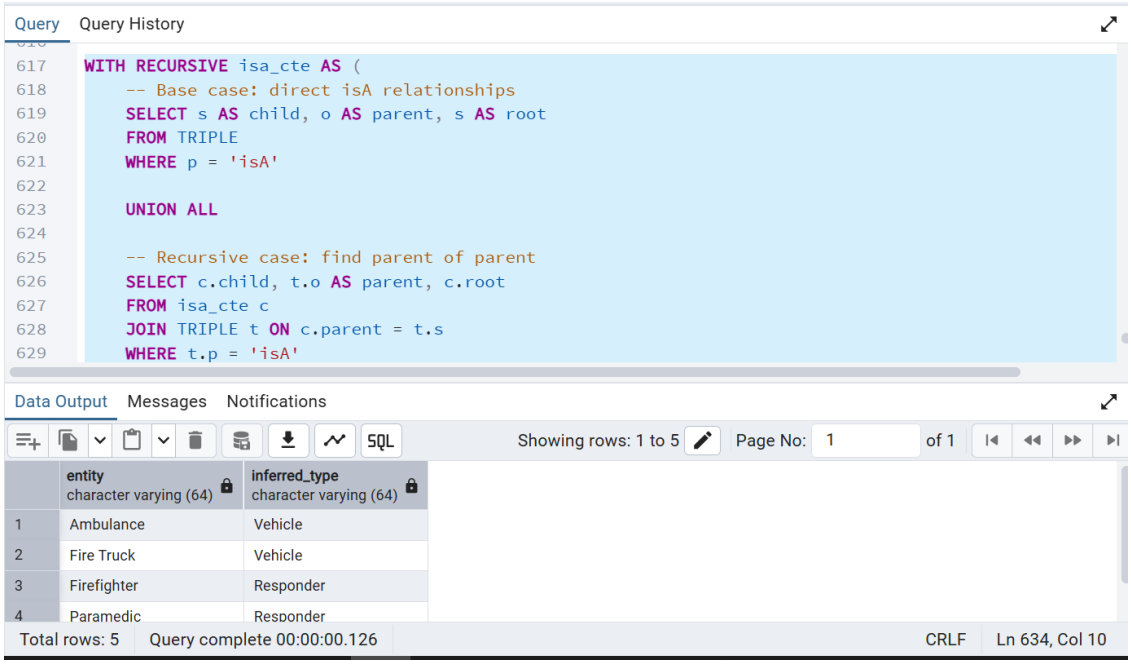
1. Create table TRIPLE(s VARCHAR2(64), p VARCHAR2(64), o VARCHAR2(64)).



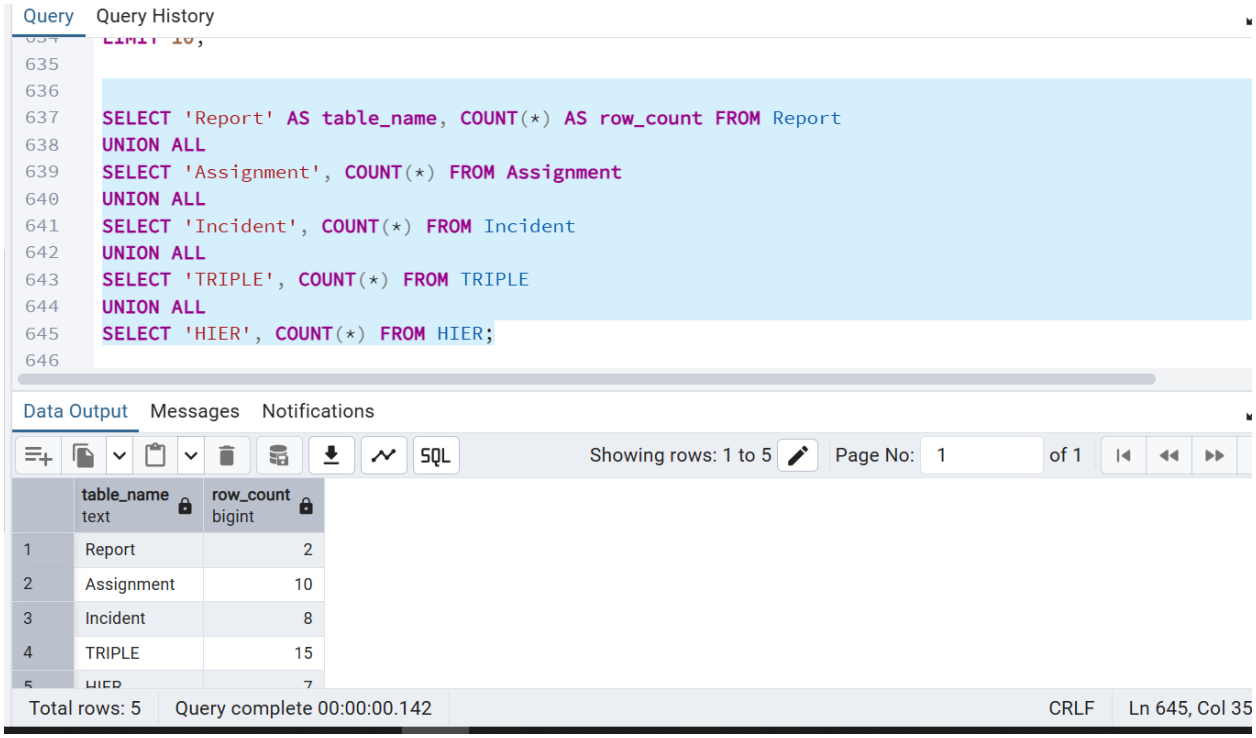
2. Insert 8–10 domain facts relevant to your project (e.g., simple type hierarchy or rule implications).



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



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



emergency_response/postgres@PostgreSQL Demo

No limit

Query | Query History

```
647  BEGIN;
648
649  -- Example: small demo triples (already included in previous step)
650  INSERT INTO TRIPLE (s, p, o) VALUES
651  ('DemoEntity1', 'isA', 'DemoType1'),
652  ('DemoEntity2', 'isA', 'DemoType1');
653
654  -- Run recursive query or inference demo here
655
656  COMMIT;
657
658  BEGIN;
659
660  DELETE FROM TRIPLE
661  WHERE s LIKE 'DemoEntity%' OR o LIKE 'DemoType%';
662
663  COMMIT;
```

Data Output | Messages | Notifications

```
COMMIT

Query returned successfully in 110 msec.
```

emergency_response/postgres@PostgreSQL Demo

No limit

Query | Query History

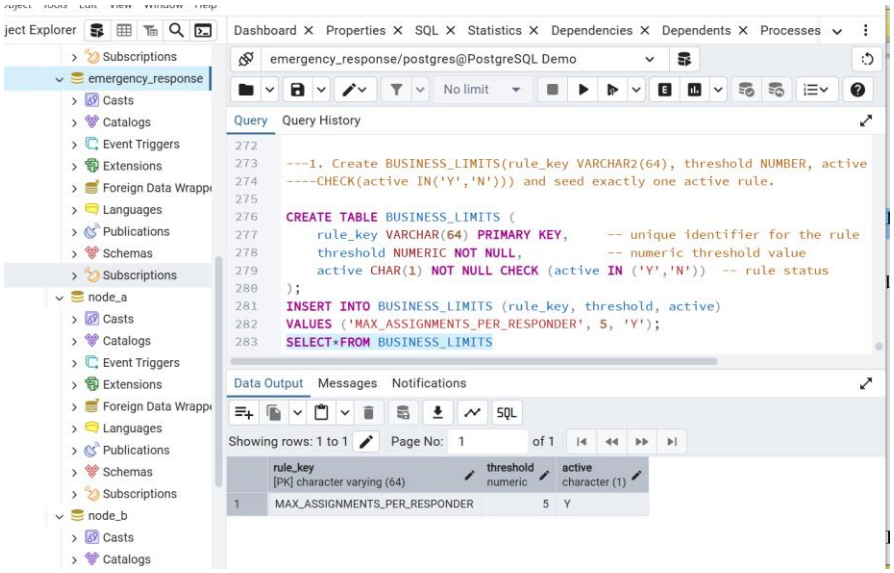
```
669
670
671
672
673
674
675
676  BEGIN;
677
678  DELETE FROM TRIPLE
679  WHERE s LIKE 'DemoEntity%' OR o LIKE 'DemoType%';
680
681  COMMIT;
682
683
684
685
```

Data Output | Messages | Notifications

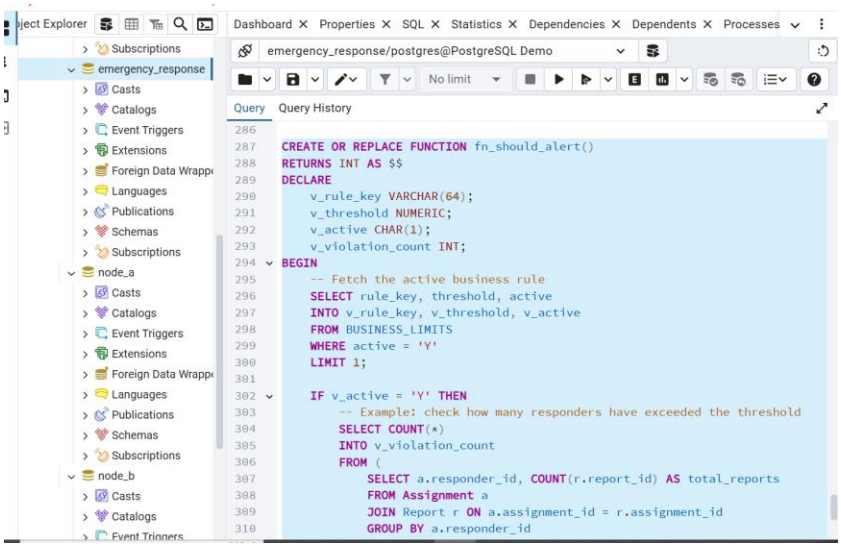
```
COMMIT

Query returned successfully in 97 msec.
```

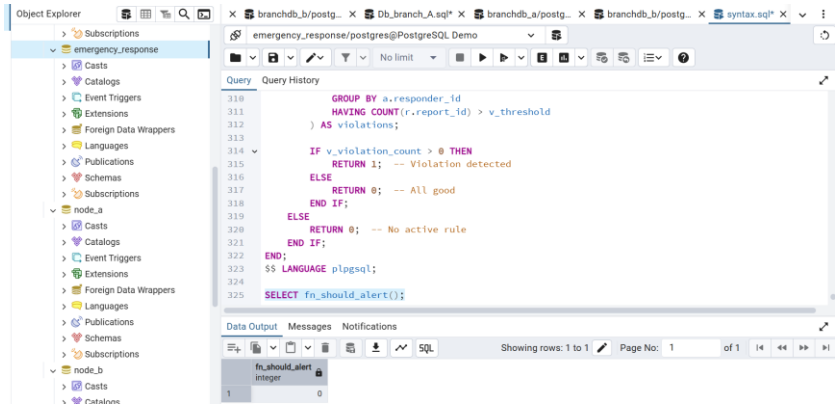
B10: Business Limit Alert (Function + Trigger) (row-budget safe)

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

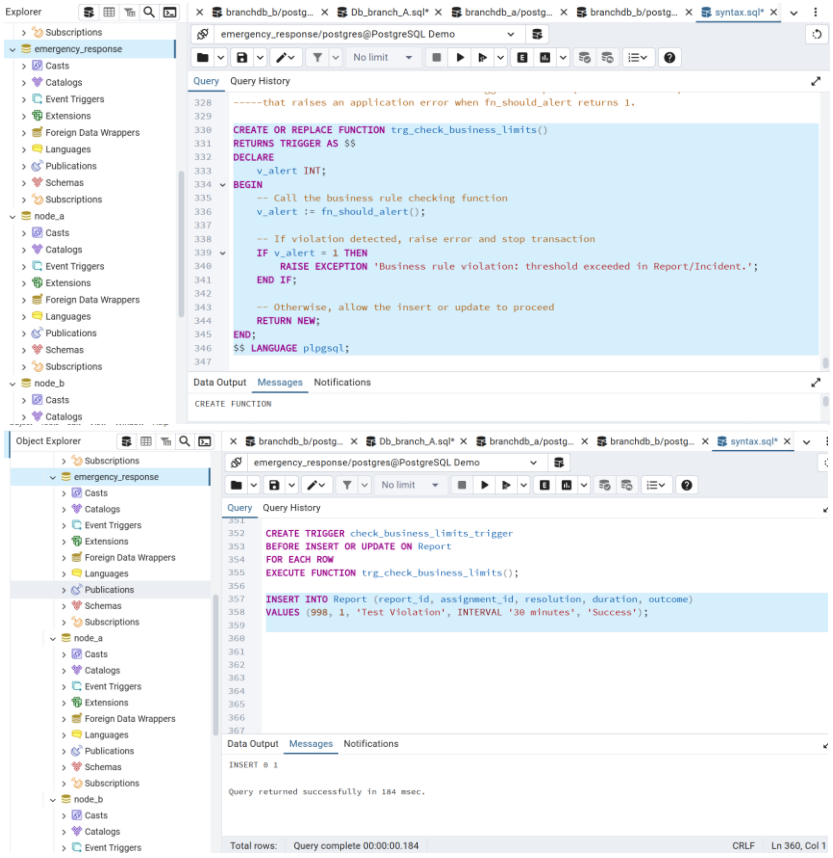


2. Implement function fn_should_alert(...) that reads BUSINESS_LIMITS and inspects current data in Report or Incident to decide a violation (return 1/0).

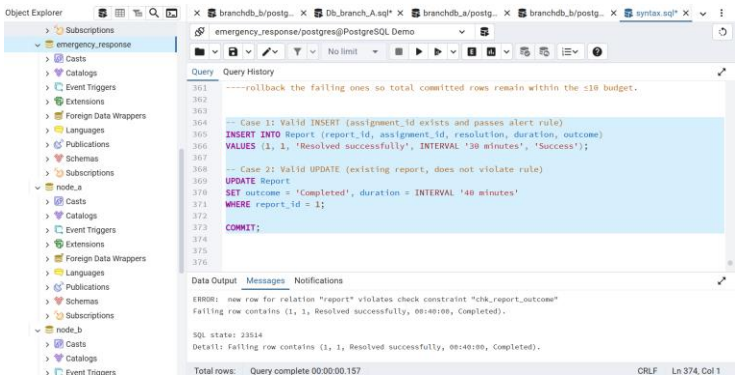


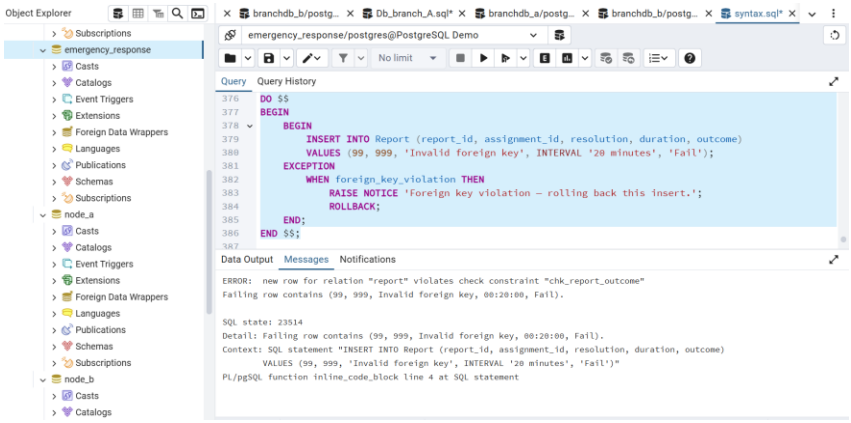


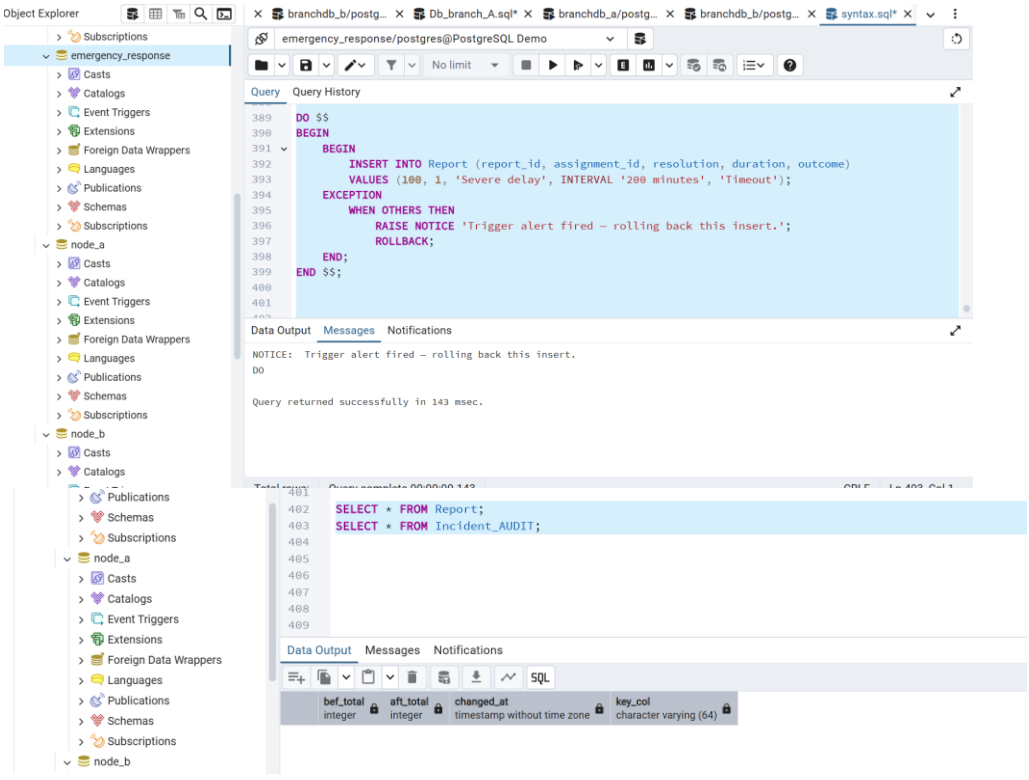
3. Create a BEFORE INSERT OR UPDATE trigger on Report (or relevant table) that raises an application error when fn_should_alert returns 1.



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


Object Explorer
emergency_response/postgres@PostgreSQL Demo
No limit
Query
Query History
----rollback the failing ones so total committed rows remain within the ≤10 budget.
-- Case 1: Valid INSERT (assignment_id exists and passes alert rule)
INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
VALUES (1, 1, 'Resolved successfully', INTERVAL '30 minutes', 'Success');
-- Case 2: Valid UPDATE (existing report, does not violate rule)
UPDATE Report
SET outcome = 'Completed', duration = INTERVAL '40 minutes'
WHERE report_id = 1;
COMMIT;
Data Output
Messages
Notifications
ERROR: new row for relation "report" violates check constraint "chk_report_outcome"
Failing row contains (1, 1, Resolved successfully, 00:40:00, Completed).
SQL state: 23514
Detail: Failing row contains (1, 1, Resolved successfully, 00:40:00, Completed).
Total rows:
Query complete 00:00:00.157
CRLF
Ln 374, Col 1


Object Explorer
emergency_response/postgres@PostgreSQL Demo
No limit
Query
Query History
DO $$
BEGIN
BEGIN
INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
VALUES (99, 999, 'Invalid foreign key', INTERVAL '20 minutes', 'Fail');
EXCEPTION
WHEN foreign_key_violation THEN
RAISE NOTICE 'Foreign key violation — rolling back this insert.';
ROLLBACK;
END;
END $$;
Data Output
Messages
Notifications
ERROR: new row for relation "report" violates check constraint "chk_report_outcome"
Failing row contains (99, 999, Invalid foreign key, 00:20:00, Fail).
SQL state: 23514
Detail: Failing row contains (99, 999, Invalid foreign key, 00:20:00, Fail).
Context: SQL statement "INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
VALUES (99, 999, 'Invalid foreign key', INTERVAL '20 minutes', 'Fail')"
PL/pgSQL function inline_code_block line 4 at SQL statement


Object Explorer
emergency_response/postgres@PostgreSQL Demo
No limit
Query
Query History
DO $$
BEGIN
BEGIN
INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
VALUES (100, 1, 'Severe delay', INTERVAL '200 minutes', 'Timeout');
EXCEPTION
WHEN OTHERS THEN
RAISE NOTICE 'Trigger alert fired — rolling back this insert.';
ROLLBACK;
END;
END $$;
Data Output
Messages
Notifications
NOTICE: Trigger alert fired — rolling back this insert.
DO
Query returned successfully in 143 msec.
SELECT * FROM Report;
SELECT * FROM Incident_AUDIT;
Data Output
Messages
Notifications
SQL
bef_total integer
aft_total integer
changed_at timestamp without time zone
key_col character varying (64)