

### A1: Fragment & Recombine Main Fact ( $\leq 10$ 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 fragmentation

The screenshot shows a database query editor with a toolbar at the top and a main area divided into two panes: 'Query' and 'Scratch Pad'. The 'Query' pane contains the following SQL code:

```

35     vehicle_id INT,
36     start_time TIMESTAMP,
37     end_time TIMESTAMP
38 );
39
40 -- Insert range fragment data (assignment_id <= 5)
41 INSERT INTO Assignment_A
42 SELECT *
43 FROM Assignment
44 WHERE assignment_id <= 5;
45 SELECT * FROM Assignment_A

```

The 'Scratch Pad' pane is empty. Below the editor is a 'Data Output' section with tabs for 'Data Output', 'Messages', and 'Notifications'. It shows a table with the following data:

	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	2025-10-01 08:35:00	2025-10-01 09:00:00
2		2	1	2	2025-10-01 08:40:00	2025-10-01 09:15:00
3		3	2	3	2025-10-02 10:10:00	2025-10-02 10:45:00
4		4	3	4	2025-10-03 14:25:00	2025-10-03 15:00:00
5		5	4	5	2025-10-03 18:45:00	2025-10-03 19:05:00

Assignment\_B created from Assignment table using range fragmentation

The screenshot shows a database query editor with a toolbar at the top and a main area divided into two panes: 'Query' and 'Scratch Pad'. The 'Query' pane contains the following SQL code:

```

38
39 -- Insert fragment data (assignment_id > 5)
40 INSERT INTO Assignment_B
41 SELECT *
42 FROM Assignment
43 WHERE assignment_id > 5;
44 SELECT * FROM Assignment_B

```

The 'Scratch Pad' pane is empty. Below the editor is a 'Data Output' section with tabs for 'Data Output', 'Messages', and 'Notifications'. It shows a table with the following data:

	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	2025-10-04 09:05:00	2025-10-04 09:50:00
2		7	5	8	2025-10-04 09:10:00	2025-10-04 09:40:00
3		8	6	9	2025-10-05 11:15:00	2025-10-05 11:55:00
4		9	2	10	2025-10-02 10:05:00	2025-10-02 10:25:00
5		10	4	6	2025-10-03 18:50:00	2025-10-03 19:25:00

2. Insert a TOTAL of  $\leq 10$  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

```

45  SELECT * FROM Assignment_A
46  INSERT INTO Assignment_A (assignment_id, incident_id, responder_id, vehicle_id, start_time, end_time)
47  VALUES
48  (6, 5, 7, 6, '2025-10-04 09:05:00', '2025-10-04 09:50:00'),
49  (7, 5, 8, 7, '2025-10-04 09:10:00', '2025-10-04 09:40:00'),
50  (8, 6, 9, 8, '2025-10-05 11:15:00', '2025-10-05 11:55:00'),
51  (9, 2, 10, 9, '2025-10-02 10:05:00', '2025-10-02 10:25:00'),
52  (10, 4, 6, 10, '2025-10-03 18:50:00', '2025-10-03 19:25:00');

```

Data Output Messages Notifications

assignment_id	incident_id	responder_id	vehicle_id	start_time	end_time
1	1	1	1	2025-10-01 08:35:00	2025-10-01 09:00:00
2	2	1	2	2025-10-01 08:40:00	2025-10-01 09:15:00
3	3	2	3	2025-10-02 10:10:00	2025-10-02 10:45:00
4	4	3	4	2025-10-03 14:25:00	2025-10-03 15:00:00
5	5	4	5	2025-10-03 18:45:00	2025-10-03 19:05:00
6	6	5	7	2025-10-04 09:05:00	2025-10-04 09:50:00
7	7	5	8	2025-10-04 09:10:00	2025-10-04 09:40:00
8	8	6	9	2025-10-05 11:15:00	2025-10-05 11:55:00

After inserting rows into Assignment\_b

```

44  SELECT * FROM Assignment_B
45  INSERT INTO Assignment_B (assignment_id, incident_id, responder_id, vehicle_id, start_time, end_time)
46  VALUES
47  (1, 1, 1, 1, '2025-10-01 08:35:00', '2025-10-01 09:00:00'),
48  (2, 1, 2, 2, '2025-10-01 08:40:00', '2025-10-01 09:15:00'),
49  (3, 2, 3, 3, '2025-10-02 10:10:00', '2025-10-02 10:45:00'),
50  (4, 3, 4, 4, '2025-10-03 14:25:00', '2025-10-03 15:00:00'),

```

Data Output Messages Notifications

assignment_id	incident_id	responder_id	vehicle_id	start_time	end_time
1	6	5	7	2025-10-04 09:05:00	2025-10-04 09:50:00
2	7	5	8	2025-10-04 09:10:00	2025-10-04 09:40:00
3	8	6	9	2025-10-05 11:15:00	2025-10-05 11:55:00
4	9	2	10	2025-10-02 10:05:00	2025-10-02 10:25:00
5	10	4	6	2025-10-03 18:50:00	2025-10-03 19:25:00
6	1	1	1	2025-10-01 08:35:00	2025-10-01 09:00:00
7	2	1	2	2025-10-01 08:40:00	2025-10-01 09:15:00
8	3	2	3	2025-10-02 10:10:00	2025-10-02 10:45:00
9	4	3	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

```

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'
);

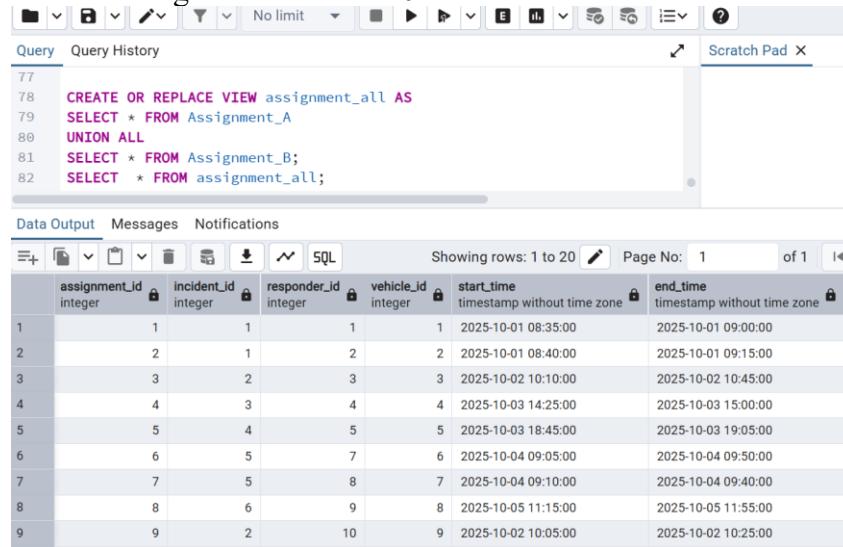
-- Step 3: Create user mapping
CREATE USER MAPPING FOR postgres
SERVER proj_link_1
OPTIONS (
    user 'postgres',
    password 'postgres'
);

-- 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.,  
 $\text{SUM}(\text{MOD}(\text{primary\_key}, 97))$ ) :results must match fragments vs Assignment\_ALL.

After creating view there is 20 row in it



The screenshot shows a PostgreSQL client interface. The top part is the Query tab, which contains the following SQL code:

```

77
78 CREATE OR REPLACE VIEW assignment_all AS
79 SELECT * FROM Assignment_A
80 UNION ALL
81 SELECT * FROM Assignment_B;
82 SELECT * FROM assignment_all;

```

The bottom part is the Data Output tab, which displays the results of the query. The results are presented in a table with the following columns:

	assignment_id	incident_id	responder_id	vehicle_id	start_time	end_time
1		1	1	1	2025-10-01 08:35:00	2025-10-01 09:00:00
2		2	1	2	2025-10-01 08:40:00	2025-10-01 09:15:00
3		3	2	3	2025-10-02 10:10:00	2025-10-02 10:45:00
4		4	3	4	2025-10-03 14:25:00	2025-10-03 15:00:00
5		5	4	5	2025-10-03 18:45:00	2025-10-03 19:05:00
6		6	5	6	2025-10-04 09:05:00	2025-10-04 09:50:00
7		7	6	7	2025-10-04 09:10:00	2025-10-04 09:40:00
8		8	7	8	2025-10-05 11:15:00	2025-10-05 11:55:00
9		9	8	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

	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

	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

	total_all	bigint
1	20	

```

92 -- Compute checksum for each fragment
93 SELECT SUM(MOD(assignment_id, 97)) AS checksum_a FROM assignment_a;
94 SELECT SUM(MOD(assignment_id, 97)) AS checksum_b FROM assignment_b;
95
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

	checksum_a	bigint
1	55	

```

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
100
Data Output Messages Notifications
SQL Showing rows: 1 to 1 Page No: 1
checksum_b bigint
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
100
Data Output Messages Notifications
SQL Showing rows: 1 to 1 Page No: 1
checksum_all bigint
1 110

```

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

- From Node\_A, create database link 'proj\_link' to Node\_B.

The following are screenshots of linking node\_A to Node\_B

```

Query History
53
54 CREATE EXTENSION IF NOT EXISTS postgres_fdw;
55 DROP SERVER IF EXISTS proj_link_1 CASCADE;
56 CREATE SERVER proj_link_1
57 FOREIGN DATA WRAPPER postgres_fdw
58 OPTIONS (
59   host 'localhost',
60   dbname 'node_b',
61   port '5432'
62 );
63
64 -- Step 3: Create user mapping
65 CREATE USER MAPPING FOR postgres
66 SERVER proj_link_1
67 OPTIONS (
68   user 'postgres',
69   password 'postgres'
70 );
71
72 -- Step 4: Import tables from BranchDB_A
73 IMPORT FOREIGN SCHEMA public
74 LIMIT TO (Assignment_B)
75 FROM SERVER proj_link_1

```

Total rows: 1 Query complete 00:00:00.468

- Run remote SELECT on Incident@proj\_link showing up to 5 sample rows.

The screenshot shows a pgAdmin interface with multiple tabs at the top: ranch\_A.sql\*, branchdb\_a/postg..., branchdb\_b/postg..., syntax.sql\*, NODE\_A\_DB\*, and NODE\_B\_DB\*. The main window displays a SQL query:

```

76 INTO public;
77
78 CREATE OR REPLACE VIEW assignment_all AS
79   SELECT * FROM Assignment_A
80   UNION ALL
81
82   SELECT * FROM Assignment_B
83   LIMIT 5;
84
85
86
87

```

The results are shown in a Data Output tab:

	assignment_id	incident_id	responder_id	vehicle_id	start_time	end_time
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

Total rows: 5 Query complete 00:00:00.184 CRLF Ln 8

- 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.

The screenshot shows a pgAdmin interface with a single tab labeled 'Scratch'. The main window displays a SQL query:

```

87
88   a.assignment_id AS assign_a_id,
89   a.incident_id AS incident_a,
90   b.assignment_id AS assign_b_id,
91   b.incident_id AS incident_b
92
93   FROM
94     assignment_a a
95   JOIN
96     assignment_b b
97   ON
98     a.incident_id = b.incident_id
99   WHERE
100    a.assignment_id <= 5
101   LIMIT 3;
102

```

The results are shown in a Data Output tab:

	assign_a_id	incident_a	assign_b_id	incident_b
1	1	1	1	1
2	1	1	2	1
3	2	1	1	1

Total rows: 3 Query complete 00:00:00.537

### A3: Parallel vs Serial Aggregation ( $\leq 10$ rows data)

- 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.

Query History

```

103
104     incident_id,
105     COUNT(assignment_id) AS total_assignments
106
107     FROM
108     assignment_all
109     GROUP BY
110         incident_id
111     ORDER BY
112         incident_id
113     LIMIT 5;

```

Data Output Messages Notifications

Showing rows: 1 to 5 Page No:

incident_id	total_assignments
1	4
2	4
3	2
4	4
5	4

2. Run the same aggregation with /\*+ PARALLEL(Assignment\_A,8)

PARALLEL(Assignment\_B,8) \*/ to force a parallel plan despite small size.

```

131 CREATE EXTENSION IF NOT EXISTS pg_parallel_plan;
132
133 SET max_parallel_workers_per_gather = 8;
134
135 SELECT
136     incident_id,
137     COUNT(assignment_id) AS total_assignments
138
139     FROM
140     assignment_a
141     GROUP BY
142

```

Data Output Messages Notifications

Showing rows: 1 to 6 Page No:

incident_id	total_assignments
1	3
2	5
3	4
4	6
5	2

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

Query History

```

114
115 EXPLAIN ANALYZE
116 SELECT
117     incident_id,
118     COUNT(*) AS total_assignments
119
120     FROM
121     assignment_all
122     GROUP BY
123         incident_id;

```

Scratch F

Data Output Messages Notifications

Showing rows: 1 to 8 Page No: 1

QUERY PLAN

text

Batches: 1 Memory Usage: 40kB

3     -> Append (cost=0.00..798.52 rows=4285 width=4) (actual time=0.065..1.257 rows=20 loops=1)

4      -> Seq Scan on assignment\_a (cost=0.00..23.60 rows=1360 width=4) (actual time=0.063..0.068 rows=10 loops=1)

5      -> Foreign Scan on assignment\_b (cost=100.00..753.50 rows=2925 width=4) (actual time=1.176..1.181 rows=10 loops=1)

6

Planning Time: 7.875 ms

Execution Time: 3.514 ms

```

131
132 SET max_parallel_workers_per_gather = 8;
133 EXPLAIN ANALYZE
134
135     SELECT
136         incident_id,
137         COUNT(assignment_id) AS total_assignments
138     FROM
139         assignment_a
140     GROUP BY
141         incident_id;

```

Showing rows: 1 to 9 Page No: 1

QUERY PLAN

text

Batches: 1 Memory Usage: 40KB

→ Gather (cost=0.00..14.39 rows=1360 width=8) (actual time=13.447..286.333 rows=10 loops=1)

Workers Planned: 3

Workers Launched: 3

→ Parallel Seq Scan on assignment\_a (cost=0.00..14.39 rows=439 width=8) (actual time=0.010..0.013 rows=1 loops=1)

Planning Time: 0.611 ms

Execution Time: 286.511 ms

Total rows: 9 Query complete 00:00:00.405

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.

```

170
171 DO $$$
172 BEGIN
173     -- Insert one local row into Assignment_A
174     INSERT INTO assignment_a (assignment_id, incident_id, responder_id, vehicle_id, start_time, end_time)
175     VALUES (101, 3, 4, 4, '2025-10-27 09:00:00', '2025-10-27 09:30:00');
176
177     -- Insert one remote row into Report on Node_B using FDW
178     INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
179     VALUES (201, 101, 'Handled remotely via FDW', INTERVAL '30 minutes', 'Success');
180
181     -- Commit both inserts (distributed commit)
182     COMMIT;
183 END $$;
184
185 SELECT * FROM assignment_a WHERE assignment_id = 101;
186 SELECT * FROM Report WHERE assignment_id = 101;
187

```

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

assignment_id	incident_id	responder_id	vehicle_id	start_time	end_time
101	3	4	4	2025-10-27 09:00:00	2025-10-27 09:30:00

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

```

Query 188
188
189 DO
190 $$*
191
192 DECLARE
193   success_a BOOLEAN := FALSE;
194   success_b BOOLEAN := FALSE;
195
196 BEGIN
197   RAISE NOTICE '--- Starting Two-Phase Commit Simulation ---';
198
199   -- Phase 1: Try local insert (NODE A)
200   BEGIN
201     INSERT INTO assignment_a (assignment_id, incident_id, responder_id, vehicle_id, start_time,
202       VALUES (104, 3, 4, 4, '2025-10-27 09:00:00', '2025-10-27 09:30:00');
203     success_a := TRUE;
204     RAISE NOTICE 'Local insert (NODE A) successful.';
205   EXCEPTION WHEN OTHERS THEN
206     RAISE NOTICE 'Local insert failed on Branch A: %', SQLERRM;
207     success_a := FALSE;
208   END;
209
210   -- Phase 1: Try remote insert (Branch B via FDW)
211   BEGIN
212     INSERT INTO yreport (report_id, assignment_id, resolution, duration, outcome)
213       VALUES (301, 101, 'Handled remotely via FDW', INTERVAL '30 minutes', 'Success');
214     success_b := TRUE;
215     RAISE NOTICE 'Remote insert (NODE B) successful.';
216   EXCEPTION WHEN OTHERS THEN
217     RAISE NOTICE 'Remote insert failed on Branch B: %', SQLERRM;
218     success_b := FALSE;
219   END;
220
221   -- Phase 2: Commit or Rollback both
222   IF success_a AND success_b THEN
223     RAISE NOTICE 'Both inserts succeeded. Committing...';
224     COMMIT;
225   ELSE
226     RAISE NOTICE 'One or more inserts failed. Rolling back...';
227     ROLLBACK;
228   END IF;
229
230   RAISE NOTICE '--- End of Two-Phase Commit Simulation ---';
231
232 $$ LANGUAGE plpgsql;

```

Data Output    Messages    Notifications

```

NOTICE: --- Starting Two-Phase Commit Simulation ---
NOTICE: Local insert (NODE A) successful.
NOTICE: Remote insert failed on Branch B: relation "yreport" 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.

```

236
237
238
239
240
241
242
243
244
245

BEGIN;
INSERT INTO Assignment_A (assignment_id, incident_id, responder_id, vehicle_id, start_time, end_time)
VALUES (107, 3, 4, 4, '2025-10-27 09:00:00', '2025-10-27 09:30:00');

-- Prepare the transaction (this writes a prepared XACT entry)
PREPARE TRANSACTION 'tx_recover_a';

Data Output Messages Notifications
PREPARE TRANSACTION

Query returned successfully in 467 msec.

Total rows: 1 Query complete 00:00:00.467 CRLF Ln 24
Query History Scratch Pad
100
104
105
106
107
108
109
110
111
112

BEGIN;
INSERT INTO Assignment_B (assignment_id, incident_id, responder_id, vehicle_id, start_time, end_time)
VALUES (105, 3, 4, 4, '2025-10-27 09:00:00', '2025-10-27 09:30:00');

-- Prepare the transaction (this writes a prepared XACT entry)
PREPARE TRANSACTION 'tx_recover_B';

Data Output Messages Notifications
PREPARE TRANSACTION

Query returned successfully in 199 msec.

241
242
243
244
245

SELECT * FROM pg_prepared_xacts;

Data Output Messages Notifications
SQL Showing rows: 1 to 2 Page No: 1 of 1 < <<
transaction_xid | gid | prepared_timestamp with time zone | owner_name | database_name
1 | 1128 | tx_recover_a | 2025-10-28 22:00:08.748451+02 | postgres | node_a
2 | 1129 | tx_recover_B | 2025-10-28 22:00:43.922764+02 | postgres | node_b

```

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

Query    Query History

```

240
241
242
243
244 SELECT * FROM pg_prepared_xacts;
245
246 -- After this line you should see: PREPARE TRANSACTION
247 ROLLBACK PREPARED 'tx_recover_a'
248
249
250
251
252
253

```

Data Output    Messages    Notifications

Showing rows: 1 to 1    Page No: 1

	transaction xid	gid	prepared timestamp with time zone	owner name	database name
1	1129	tx_recover_B	2025-10-28 22:00:43.922764+02	postgres	node_b

Query    Query History

```

240
241
242
243
244 SELECT * FROM pg_prepared_xacts;
245
246
247
248
249
250
251
252
253

```

Data Output    Messages    Notifications

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

#### 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.

The screenshot shows the pgAdmin 4 interface with multiple tabs at the top: syntax.sql\*, NODE\_A\_DB\*, NODE\_B\_DB\*, node\_a/postgres@PostgreSQL Demo\*, and node\_b/postgres@PostgreSQL Demo\*. The active tab is 'node\_a/postgres@PostgreSQL Demo'. The query editor contains the following SQL code:

```
1 BEGIN;
2 UPDATE Assignment_A
3 SET incident_id = 30
4 WHERE responder_id = 1;
```

The 'Data Output' tab shows the result of the update:

```
UPDATE 1
```

Query returned successfully in 124 msec.

2. Open Session 2 from Node\_B via Incident@proj\_link or Report@proj\_link to UPDATE the same logical row.

The screenshot shows the pgAdmin 4 interface with the same tabs as the previous screenshot. The active tab is 'node\_b/postgres@PostgreSQL Demo'. The query editor contains the same SQL code as the first screenshot:

```
1 BEGIN;
2 UPDATE Assignment_A
3 SET incident_id = 31
4 WHERE responder_id = 1;
```

The 'Data Output' tab is selected and displays:

```
No data to display
```

A message in the center of the window says: "Waiting for the query to complete..."

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.

Query    Query History    Scratch Pad

```

1  SELECT
2      a.pid AS waiting_pid,
3      a.username AS waiting_user,
4      a.query AS waiting_query,
5      a.state AS waiting_state,
6      l.locktype,
7      l.mode AS lock_mode,
8      l.granted AS lock_granted,
9      t.relname AS table_name,
10     b.pid AS blocking_pid,

```

Data Output    Messages    Notifications

waiting_pid	waiting_user	waiting_query	waiting_state	locktype	lock_mode
20832	postgres	UPDATE public.assignment_a SET incident_id = 31 WHERE ((responder_id = 1))	active	text	transactionid
20832	postgres	UPDATE public.assignment_a SET incident_id = 31 WHERE ((responder_id = 1))	active	text	transactionid
20832	postgres	UPDATE public.assignment_a SET incident_id = 31 WHERE ((responder_id = 1))	active	text	transactionid

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

Query    Query History    Scratch Pad

```

1 BEGIN;
2 UPDATE Assignment_A
3 SET incident_id = 30
4 WHERE responder_id = 1;
5
6
7 COMMIT;
8
9
10
11

```

Data Output    Messages    Notifications

Query returned successfully in 110 msec.

node\_b/postgres@PostgreSQL Demo    Scratch Pad

Query    Query History    Scratch Pad

```

1 BEGIN;
2 UPDATE Assignment_A
3 SET incident_id = 31
4 WHERE responder_id = 1;
5
6
7
8
9
10
11

```

Data Output    Messages    Notifications

ERROR: could not serialize access due to concurrent update  
 CONTEXT: remote SQL command: UPDATE public.assignment\_a SET incident\_id = 31 WHERE ((responder\_id = 1))  
 SQL state: 40001

246  
 247  
 248    SELECT \* FROM Assignment\_A WHERE incident\_id = 30;  
 249  
 250  
 251  
 252  
 253

Data Output    Messages    Notifications

assignment_id	incident_id	responder_id	vehicle_id	start_time	end_time
1	30	1	1	2025-10-01 08:35:00	2025-10-01 09:00:00

## B6: Declarative Rules Hardening ( $\leq 10$ 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  $\leq 10$  total.

Query History

```

261
262
263
264
265
266
267
268
269
270 |
271 INSERT INTO Incident (incident_id, incident_type, location, district, severity, reported_time, sta
272 VALUES
273 (7, 'Fire Outbreak', 'Kigali', 'Kigali', 'High', '2025-10-06 08:00:00', 'Pending'),
274 (8, 'Flood', 'Rwamagana', 'Eastern', 'Medium', '2025-10-06 09:30:00', 'Pending');
275
276

```

Data Output Messages Notifications

INSERT 0 2

Query returned successfully in 603 msec.

Query History

```

283 EXCEPTION WHEN OTHERS THEN
284     RAISE NOTICE 'Expected failure: NULL incident_type';
285     ROLLBACK;
286 END;
287
288 -- Fails: invalid severity
289 BEGIN
290     INSERT INTO Incident (incident_id, incident_type, location, district, severity, reported_time)
291     VALUES (10, 'Medical Emergency', 'Musanze', 'Western', 'Critical', '2025-10-06 11:00:00', 'P
292 EXCEPTION WHEN OTHERS THEN
293     RAISE NOTICE 'Expected failure: invalid severity';
294     ROLLBACK;
295 END;
296 END $$;
297
298

```

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

302
303
304
305 |
306
307 **INSERT INTO** Report (report\_id, assignment\_id, resolution, duration, outcome)
308 **VALUES**
309 (11, 1, 'Resolved fire', INTERVAL '30 minutes', 'Success'),
310 (12, 2, 'Assisted flood victims', INTERVAL '45 minutes', 'Success');
311

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

```

327   EXCEPTION WHEN OTHERS THEN
328     RAISE NOTICE 'Expected failure: negative duration';
329     ROLLBACK;
330   END;
331
332   -- Fails: invalid outcome
333   BEGIN
334     INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
335       VALUES (14, 4, 'Invalid outcome', INTERVAL '20 minutes', 'Unknown');
336   EXCEPTION WHEN OTHERS THEN
337     RAISE NOTICE 'Expected failure: invalid outcome';
338     ROLLBACK;
339   END;
340 END $$;
341
342

```

Data Output Messages Notifications

NOTICE: Expected failure: negative duration  
NOTICE: Expected failure: invalid outcome  
DO

Query returned successfully in 91 msec.

Total rows: Query complete 00:00:00.091 CRLF Ln 342, Col 1

### 3. Show clean error handling for failing cases.

```

355   WHEN OTHERS THEN
356     RAISE NOTICE 'Insert failed due to unexpected error: %', SQLERRM;
357   END;
358
359   -- Failing Insert 2: invalid severity
360   BEGIN
361     INSERT INTO Incident (incident_id, incident_type, location, district, severity, reported_time)
362       VALUES (10, 'Medical Emergency', 'Musanje', 'Western', 'Critical', '2025-10-06 11:00:00', 'P
363   EXCEPTION WHEN CHECK_VIOLATION THEN
364     RAISE NOTICE 'Insert failed: severity must be Low, Medium, or High.';
365   WHEN OTHERS THEN
366     RAISE NOTICE 'Insert failed due to unexpected error: %', SQLERRM;
367   END;
368
369 END $$;
370

```

Data Output Messages Notifications

NOTICE: Insert failed: incident\_type cannot be NULL.  
NOTICE: Insert failed: severity must be Low, Medium, or High.  
DO

Query returned successfully in 229 msec.

Total rows: Query complete 00:00:00.229 CRLF Ln 369, Col 8

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

- Create an audit table Incident\_AUDIT(bef\_total NUMBER, aft\_total NUMBER, changed\_at TIMESTAMP, key\_col VARCHAR2(64)).

Query    Query History

```

372
373
374
375
376   CREATE TABLE Incident_AUDIT (
377     bef_total INT NOT NULL,          -- total before change
378     aft_total INT NOT NULL,          -- total after change
379     changed_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, -- time of change
380     key_col VARCHAR(64) NOT NULL    -- key column affected (e.g., incident_id)
381   );
382
383
384
385
386
387

```

Data Output    Messages    Notifications

CREATE TABLE

Query returned successfully in 159 msec.

Total rows:    Query complete 00:00:00.159    CRLF    Ln 382, Col 1

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

Query    Query History

```

385   CREATE OR REPLACE FUNCTION update_incident_totals()
386   RETURNS TRIGGER AS $$ 
387   BEGIN
388     -- Update total reports per incident
389     UPDATE Incident i
390     SET total_reports = sub.count_reports
391     FROM (
392       SELECT assignment_id AS incident_id, COUNT(*) AS count_reports
393         FROM Report
394        GROUP BY assignment_id
395     ) AS sub
396     WHERE i.incident_id = sub.incident_id;
397
398     RETURN NULL; -- statement-level triggers return NULL
399   END;
400   $$ LANGUAGE plpgsql;

```

Data Output    Messages    Notifications

CREATE FUNCTION

Query returned successfully in 106 msec.

Query    Query History

```

401 CREATE TRIGGER trg_update_incident_totals
402 AFTER INSERT OR UPDATE OR DELETE
403 ON Report
404 FOR EACH STATEMENT
405 EXECUTE FUNCTION update_incident_totals();
406
407
408
409
410
411
412
413
414
415
416

```

Data Output    Messages    Notifications

CREATE TRIGGER

Query returned successfully in 97 msec.

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

1. Create table HIER(parent\_id, child\_id) for a natural hierarchy (domain-specific)

Query    Query History

```

477
478
479
480
481
482
483 CREATE TABLE HIER (
484     parent_id INT NOT NULL,
485     child_id INT PRIMARY KEY, -- each child is unique
486     CONSTRAINT fk_parent FOREIGN KEY (parent_id) REFERENCES HIER(child_id) ON DELETE CASCADE
487 );
488
489
490
491
492

```

Data Output    Messages    Notifications

CREATE TABLE

Query returned successfully in 158 msec.

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

Query History

```

483 CREATE TABLE HIER (
484     parent_id INT NULL,          -- allow NULL for root nodes
485     child_id INT PRIMARY KEY,
486     CONSTRAINT fk_parent FOREIGN KEY (parent_id) REFERENCES HIER(child_id) ON DELETE CASCADE
487 );
488
489 INSERT INTO HIER (parent_id, child_id) VALUES (NULL, 1); -- Root node
490
491
492 INSERT INTO HIER (parent_id, child_id) VALUES (1, 2);
493 INSERT INTO HIER (parent_id, child_id) VALUES (1, 3);
494 INSERT INTO HIER (parent_id, child_id) VALUES (1, 4);
495
496 INSERT INTO HIER (parent_id, child_id) VALUES (2, 5);
497 INSERT INTO HIER (parent_id, child_id) VALUES (2, 6);
498 INSERT INTO HIER (parent_id, child_id) VALUES (3, 7);

```

Data Output

```

INSERT 0 1

Query returned successfully in 147 msec.

Total rows: 7 Query complete 00:00:00.149 CRLF Ln 498, Col 54

```

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.

Query History

```

504 WITH RECURSIVE hier_cte AS (
505     -- Base case: root nodes (parent_id IS NULL)
506     SELECT
507         child_id,
508         child_id AS root_id,
509         1 AS depth
510     FROM HIER
511     WHERE parent_id IS NULL
512
513     UNION ALL
514
515     -- Recursive case: children
516     SELECT
517         h.child_id,
518         cte.root_id,
519         cte.depth + 1 AS depth

```

Data Output

	child_id	root_id	depth
1	1	1	1
2	2	1	2

```

Showing rows: 1 to 7 Page No: 1 of 1 CRLF Ln 526, Col 1
Total rows: 7 Query complete 00:00:00.127

```

Query History

```

529      WITH RECURSIVE hier_cte AS (
530          SELECT child_id, child_id AS root_id, 1 AS depth
531          FROM HIER
532         WHERE parent_id IS NULL
533
534        UNION ALL
535
536          SELECT h.child_id, cte.root_id, cte.depth + 1
537          FROM HIER h
538         JOIN hier_cte cte ON h.parent_id = cte.child_id
539      )
540      SELECT
541          cte.child_id,
542          cte.root_id,
543          cte.depth.

```

Data Output Messages Notifications

	child_id integer	root_id integer	depth integer	total_assignments bigint
1	1	1	1	2
2	2	1	2	2
3	3	1	2	1

Total rows: 3 Query complete 00:00:00.156 CRLF Ln 550, Col 1

4. Reuse existing seed rows; do not exceed the  $\leq 10$  committed rows budget.

Query History

```

568      FROM HIER h
569      JOIN hier_cte cte ON h.parent_id = cte.child_id
570    )
571    SELECT
572        cte.child_id,
573        cte.root_id,
574        cte.depth,
575        COUNT(a.assignment_id) AS total_assignments
576      FROM hier_cte cte
577      LEFT JOIN Assignment a ON a.incident_id = cte.child_id
578      GROUP BY cte.child_id, cte.root_id, cte.depth
579      ORDER BY cte.root_id, cte.depth, cte.child_id
580      LIMIT 3;

```

Data Output Messages Notifications

	child_id integer	root_id integer	depth integer	total_assignments bigint
1	1	1	1	2
2	2	1	2	2
3	3	1	2	1

Total rows: 3 Query complete 00:00:00.335 CRLF Ln 552, Col 1

B9: Mini-Knowledge Base with Transitive Inference ( $\leq 10$  facts)

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

Query History

```

584
585
586
587 CREATE TABLE TRIPLE (
588     s VARCHAR(64) NOT NULL,    -- subject
589     p VARCHAR(64) NOT NULL,    -- predicate
590     o VARCHAR(64) NOT NULL    -- object
591 );
592
593
594
595
596
597

```

Data Output Messages Notifications

```

CREATE TABLE

Query returned successfully in 139 msec.

Total rows: 0 Query complete 00:00:00.139 CRLF Ln 592 Col 1

```

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

Query History

```

595 INSERT INTO TRIPLE (s, p, o) VALUES
596 ('Responder', 'type', 'Paramedic'),
597 ('Responder', 'type', 'Firefighter'),
598 ('Responder', 'type', 'Police Officer'),
599 ('Vehicle', 'type', 'Ambulance'),
600 ('Vehicle', 'type', 'Fire Truck'),
601 ('Incident', 'severity', 'High'),
602 ('Incident', 'severity', 'Medium'),
603 ('Incident', 'severity', 'Low'),
604 ('Paramedic', 'can_handle', 'Medical Emergency'),
605 ('Firefighter', 'can_handle', 'Fire Outbreak');

606
607 select*from TRIPLE

```

Data Output Messages Notifications

	s character varying (64)	p character varying (64)	o character varying (64)
1	Responder	type	Paramedic
2	Responder	type	Firefighter
3	Responder	type	Police Officer
4	Vehicle	type	Ambulance

Total rows: 10 Query complete 00:00:00.159 CRLF Ln 607, Col 1

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

Query    Query History

```

617 WITH RECURSIVE isa_cte AS (
618   -- Base case: direct isA relationships
619   SELECT s AS child, o AS parent, s AS root
620   FROM TRIPLE
621   WHERE p = 'isA'
622
623   UNION ALL
624
625   -- Recursive case: find parent of parent
626   SELECT c.child, t.o AS parent, c.root
627   FROM isa_cte c
628   JOIN TRIPLE t ON c.parent = t.s
629   WHERE t.p = 'isA'

```

Data Output    Messages    Notifications

	entity character varying (64)	inferred_type character varying (64)
1	Ambulance	Vehicle
2	Fire Truck	Vehicle
3	Firefighter	Responder
4	Paramedic	Responder

Total rows: 5    Query complete 00:00:00.126    CRLF    Ln 634, Col 10

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

Query    Query History

```

635
636
637   SELECT 'Report' AS table_name, COUNT(*) AS row_count FROM Report
638   UNION ALL
639   SELECT 'Assignment', COUNT(*) FROM Assignment
640   UNION ALL
641   SELECT 'Incident', COUNT(*) FROM Incident
642   UNION ALL
643   SELECT 'TRIPLE', COUNT(*) FROM TRIPLE
644   UNION ALL
645   SELECT 'HIER', COUNT(*) FROM HIER;
646

```

Data Output    Messages    Notifications

	table_name text	row_count bigint
1	Report	2
2	Assignment	10
3	Incident	8
4	TRIPLE	15
5	HIER	7

Total rows: 5    Query complete 00:00:00.142    CRLF    Ln 645, Col 35

emergency\_response/postgres@PostgreSQL Demo

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;

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

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.

```

272
273
274
275
276
277
278
279
280
281
282
283

```

--1. Create BUSINESS\_LIMITS(rule\_key VARCHAR2(64), threshold NUMBER, active  
----CHECK(active IN('Y','N'))) and seed exactly one active rule.

```

CREATE TABLE BUSINESS_LIMITS (
    rule_key VARCHAR(64) PRIMARY KEY,      -- unique identifier for the rule
    threshold NUMERIC NOT NULL,           -- numeric threshold value
    active CHAR(1) NOT NULL CHECK (active IN ('Y','N')) -- rule status
);
INSERT INTO BUSINESS_LIMITS (rule_key, threshold, active)
VALUES ('MAX_ASSIGNMENTS_PER_RESPONDER', 5, 'Y');
SELECT * FROM BUSINESS_LIMITS

```

rule_key	threshold	active
[PK] character varying (64)	numeric	character (1)
1 MAX_ASSIGNMENTS_PER_RESPONDER	5	Y

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).

```

286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310

```

```

CREATE OR REPLACE FUNCTION fn_should_alert()
RETURNS INT AS $$
DECLARE
    v_rule_key VARCHAR(64);
    v_threshold NUMERIC;
    v_active CHAR(1);
    vViolationCount INT;
BEGIN
    -- Fetch the active business rule
    SELECT rule_key, threshold, active
    INTO v_rule_key, v_threshold, v_active
    FROM BUSINESS_LIMITS
    WHERE active = 'Y'
    LIMIT 1;

    IF v_active = 'Y' THEN
        -- Example: check how many responders have exceeded the threshold
        SELECT COUNT(*)
        INTO vViolationCount
        FROM (
            SELECT a.responder_id, COUNT(r.report_id) AS total_reports
            FROM Assignment a
            JOIN Report r ON a.assignment_id = r.assignment_id
            GROUP BY a.responder_id
        ) t
        WHERE total_reports > v_threshold;
    END IF;
    RETURN vViolationCount;
END;
$$ LANGUAGE plpgsql;

```

The screenshot shows the pgAdmin interface with the Object Explorer on the left and a query editor on the right. The query editor contains the following SQL code:

```

310 GROUP BY a.responder_id
311 HAVING COUNT(r.report_id) > v_threshold
312
313 ) AS violations;
314
315 IF vViolationCount > 0 THEN
316   RETURN 1; -- Violation detected
317 ELSE
318   RETURN 0; -- All good
319 END IF;
320
321 ELSE
322   RETURN 0; -- No active rule
323 END IF;
324
325 $ LANGUAGE plpgsql;
326
327 SELECT fn_should_alert();

```

The Data Output tab shows the result of the function call:

fn_should_alert	integer
	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.

The screenshot shows the pgAdmin interface with the Object Explorer on the left and two query editors on the right. The top query editor contains the following SQL code for a function:

```

228 -----that raises an application error when fn_should_alert returns 1.
229
230 CREATE OR REPLACE FUNCTION trg_check_business_limits()
231 RETURNS TRIGGER AS $$$
232 DECLARE
233   vAlert INT;
234 BEGIN
235   -- Call the business rule checking function
236   vAlert := fn_should_alert();
237
238   -- If violation detected, raise error and stop transaction
239   IF vAlert = 1 THEN
240     RAISE EXCEPTION 'Business rule violation: threshold exceeded in Report/Incident.';
241   END IF;
242
243   -- Otherwise, allow the insert or update to proceed
244   RETURN NEW;
245
246 $$ LANGUAGE plpgsql;
247
248

```

The bottom query editor contains the following SQL code for a trigger:

```

352 CREATE TRIGGER check_business_limits_trigger
353 BEFORE INSERT OR UPDATE ON Report
354 FOR EACH ROW
355 EXECUTE FUNCTION trg_check_business_limits();
356
357 INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
358 VALUES (998, 1, 'Test Violation', INTERVAL '30 minutes', 'Success');
359
360
361
362
363
364
365
366
367

```

The Data Output tab shows the result of the insert operation:

Total rows:	Query complete 00:00:00.184
-------------	-----------------------------

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

Object Explorer

```

361     --rollback the failing ones so total committed rows remain within the $10 budget.
362
363
364     -- Case 1: Valid INSERT (assignment_id exists and passes alert rule)
365     INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
366         VALUES (1, 1, 'Resolved successfully', INTERVAL '30 minutes', 'Success');
367
368     -- Case 2: Valid UPDATE (existing report, does not violate rule)
369     UPDATE Report
370     SET outcome = 'Completed', duration = INTERVAL '40 minutes'
371     WHERE report_id = 1;
372
373     COMMIT;
374
375
376

```

Data Output Messages Notifications

ERROR: new row for relation "report" violates check constraint "chk\_report\_outcome"  
Failing row contains (1, 1, Resolved successfully, 00:30:00, Completed).

SQL state: 23514  
Detail: Failing row contains (1, 1, Resolved successfully, 00:40:00, Completed).

Total rows: 1 Query complete 00:00:00.157

Object Explorer

```

376 DO $$ BEGIN
377
378     BEGIN
379         INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
380             VALUES (99, 999, 'Invalid foreign key', INTERVAL '20 minutes', 'Fail');
381
382         EXCEPTION
383             WHEN foreign_key_violation THEN
384                 RAISE NOTICE 'Foreign key violation - rolling back this insert.';
385                 ROLLBACK;
386         END;
387     END $$;
388
389
390

```

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

```

389 DO $$ BEGIN
390
391     BEGIN
392         INSERT INTO Report (report_id, assignment_id, resolution, duration, outcome)
393             VALUES (100, 1, 'Severe delay', INTERVAL '200 minutes', 'Timeout');
394
395         EXCEPTION
396             WHEN OTHERS THEN
397                 RAISE NOTICE 'Trigger alert fired - rolling back this insert.';
398                 ROLLBACK;
399
400     END $$;
401
402
403

```

Data Output Messages Notifications

NOTICE: Trigger alert fired - rolling back this insert.  
DO

Query returned successfully in 143 msec.

Total rows: 1 Query complete 00:00:00.143

```

403
404
405
406
407
408
409

```

Data Output Messages Notifications

bef_total	aft_total	changed_at	key_col
integer	integer	timestamp without time zone	character varying (64)

