

## EXAMAN: ADVANCED DATABASE PROJECT-BASED EXAM

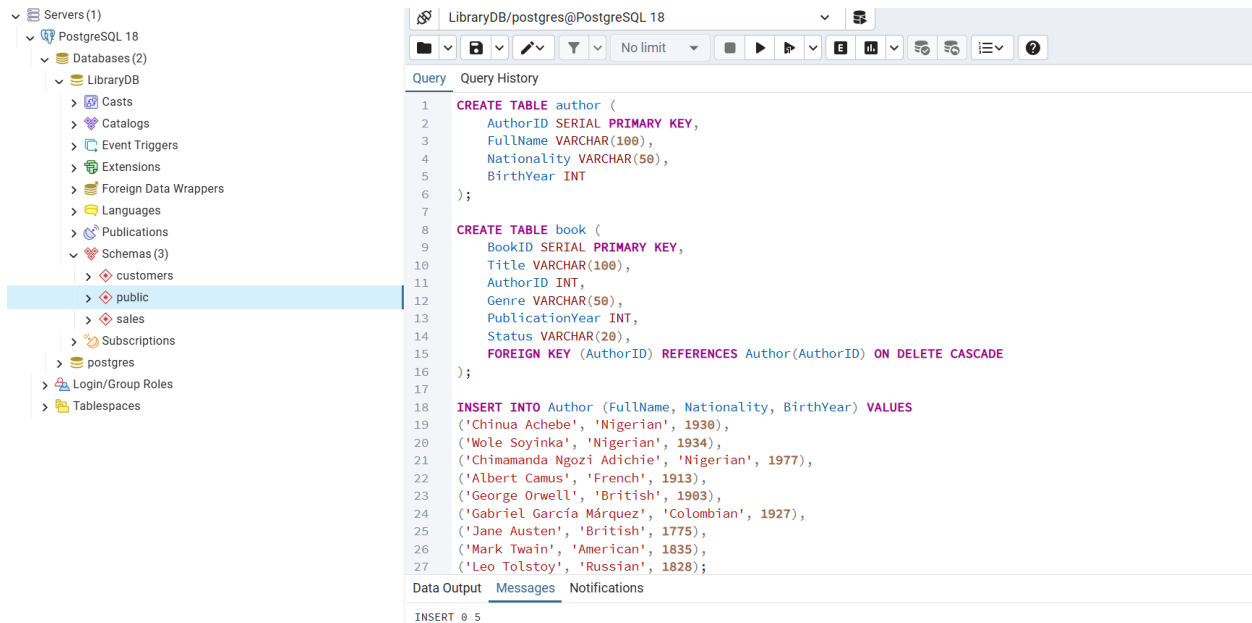
### STUDENT 14: LIBRARY & FINES

Module Code: DSM6235 School/Centre: African Centre of Excellence in Data Science

#### SECTION A:

#### WHAT TO DO

1. The table borrow is created in postgresql, LibraryDB, schema public, so my required to create Borrow\_A on Node\_A and Borrow\_B on Node\_B.



The screenshot shows the PostgreSQL IDE interface. On the left, the 'Servers' tree is expanded to show 'PostgreSQL 18' > 'Databases (2)' > 'LibraryDB' > 'Schemas (3)' > 'public'. The main editor displays the following SQL code:

```
1 CREATE TABLE author (  
2     AuthorID SERIAL PRIMARY KEY,  
3     FullName VARCHAR(100),  
4     Nationality VARCHAR(50),  
5     BirthYear INT  
6 );  
7  
8 CREATE TABLE book (  
9     BookID SERIAL PRIMARY KEY,  
10    Title VARCHAR(100),  
11    AuthorID INT,  
12    Genre VARCHAR(50),  
13    PublicationYear INT,  
14    Status VARCHAR(20),  
15    FOREIGN KEY (AuthorID) REFERENCES Author(AuthorID) ON DELETE CASCADE  
16 );  
17  
18 INSERT INTO Author (FullName, Nationality, BirthYear) VALUES  
19 ('Chinua Achebe', 'Nigerian', 1930),  
20 ('Wole Soyinka', 'Nigerian', 1934),  
21 ('Chimamanda Ngozi Adichie', 'Nigerian', 1977),  
22 ('Albert Camus', 'French', 1913),  
23 ('George Orwell', 'British', 1903),  
24 ('Gabriel Garcia Márquez', 'Colombian', 1927),  
25 ('Jane Austen', 'British', 1775),  
26 ('Mark Twain', 'American', 1835),  
27 ('Leo Tolstoy', 'Russian', 1828);
```

The bottom status bar shows 'INSERT 0 5'.

#### Creation of Borrow\_A

CREATE TABLE Borrow\_A AS

SELECT \* FROM borrow

WHERE borrow\_id <= 5;

#### Creation of Borrow\_A

CREATE TABLE Borrow\_B AS

SELECT \* FROM borrow

WHERE borrow\_id > 5;

2.

```

69
70 -- creation of Borrow_A and Borrow_B
71 CREATE TABLE Borrow_A AS
72 SELECT * FROM borrow
73 WHERE borrow_id <= 5;
74
75 CREATE TABLE Borrow_B AS
76 SELECT * FROM borrow_b
77 WHERE borrowid > 5;
78 -- inserting data into Borrow_A and Borrow_B
79
80 INSERT INTO Borrow_A (borrowid, bookid, memberid, borrowdate, returndate) VALUES
81 (1, 101, 201, '2025-10-01', '2025-10-10'),
82 (2, 102, 202, '2025-10-02', '2025-10-12'),
83 (3, 103, 203, '2025-10-03', '2025-10-13'),
84 (4, 104, 204, '2025-10-04', '2025-10-14'),
85 (5, 105, 205, '2025-10-05', '2025-10-15');
86
87 INSERT INTO Borrow_B (borrowid, bookid, memberid, borrowdate, returndate) VALUES
88 (6, 106, 206, '2025-10-06', '2025-10-16'),
89 (7, 107, 207, '2025-10-07', '2025-10-17'),
90 (8, 108, 208, '2025-10-08', '2025-10-18'),
91 (9, 109, 209, '2025-10-09', '2025-10-19'),
92 (10, 110, 210, '2025-10-10', '2025-10-20');
93
94 -- To confirm transaction.
95 COMMIT;

```

3. On Node\_A, create view Borrow\_ALL as UNION ALL of Borrow\_A and Borrow\_B@proj\_link.

```

110
111 CREATE EXTENSION IF NOT EXISTS postgres_fdw;
112
113 CREATE SERVER node_b_server
114 FOREIGN DATA WRAPPER postgres_fdw
115 OPTIONS (host 'node_b_host', dbname 'your_db', port '5432');
116
117 CREATE USER MAPPING FOR CURRENT_USER
118 SERVER node_b_server
119 OPTIONS (user 'postgres', password '123456');
120
121 CREATE FOREIGN TABLE borrow_b_fdw (
122     borrow_id INT,
123     book_id INT,
124     member_id INT,
125     borrow_date DATE,
126     return_date DATE
127 )
128 SERVER node_b_server
129 OPTIONS (schema_name 'public', table_name 'borrow_b');
130
131
132

```

4. Validate with COUNT(\*) and a checksum on a key column (e.g., SUM(MOD(primary\_key,97))) :results must match fragments vs Borrow\_ALL.

```

130
131
132
133
134
135
136
137
138
139
140 SELECT
141     'Borrow_A' AS fragment, COUNT(*) AS total_rows FROM borrow_a
142 UNION ALL
143 SELECT 'Borrow_B' AS fragment, COUNT(*) AS total_rows FROM borrow_b;
144
145 SELECT COUNT(*) AS total_rows FROM borrow_b;
146
147 -- Checksum
148 SELECT 'Borrow_A' AS fragment, SUM(MOD(borrowid,97)) AS checksum FROM borrow_a
149 UNION ALL
150 SELECT 'Borrow_B' AS fragment, SUM(MOD(borrowid,97)) AS checksum FROM borrow_b;
151
152 SELECT SUM(MOD(borrowid,97)) AS checksum FROM borrow_all;

```

checksum	bigint
1	65

A2.

1.

```

152 SELECT SUM(MOD(borrowid,9)) AS checksum FROM borrow_all;
153
154 -- On Node_A : Create a foreign server pointing to Node_B
155 CREATE EXTENSION IF NOT EXISTS postgres_fdw;
156 CREATE SERVER proj_link
157 FOREIGN DATA WRAPPER postgres_fdw
158 OPTIONS (
159     host 'NODE_B_HOST_OR_IP', -- Replace with Node_B host or IP
160     dbname 'node_b_db', -- Node_B database name
161     port '5432'
162 );
163 -- Create a user mapping for authentication
164 CREATE USER MAPPING FOR CURRENT_USER
165 SERVER proj_link
166 OPTIONS (
167     user 'postgres', -- Node_B username
168     password 'your_password' -- Node_B password
169 );
170
171 CREATE FOREIGN TABLE borrow_b_fdw (
172     borrow_id INT,
173     book_id INT,
174     member_id INT,
175     borrow_date DATE,
176
177 );

```

Data Output Messages Notifications

2 Run remote SELECT on Book@proj\_link showing up to 5 sample rows.

```

169 );
170
171 CREATE FOREIGN TABLE borrow_b_fdw (
172     borrow_id INT,
173     book_id INT,
174     member_id INT,
175     borrow_date DATE,
176     return_date DATE
177 )
178 SERVER proj_link
179 OPTIONS (
180     schema_name 'public', -- Node-B schema
181     table_name 'borrow_b' -- Node-B table
182 );
183
184 CREATE FOREIGN TABLE book_fdw (
185     book_id INT,
186     title TEXT,
187     author_id INT,
188     published_year INT
189 )
190 SERVER proj_link
191 OPTIONS (
192     schema_name 'public',
193     table_name 'book'
194 );
195
196 SELECT *
197 FROM book_fdw

```

3. Run a distributed join: local Borrow\_A (or base Borrow) joined with remote Member@proj\_link returning between 3 and 10 rows total; include selective predicates to stay within the row budget.

```

206 OPTIONS (
207     schema_name 'public',
208     table_name 'member'
209 );
210 SELECT
211     b.borrowid,
212     b.bookid,
213     b.borrowdate,
214     m.name AS member_name,
215     m.membership_date
216 FROM borrow_a b -- local fragment
217 JOIN member_fdw m -- remote fragment
218     ON b.memberid = m.member_id
219 WHERE b.borrowid <= 5 -- selective predicate to limit rows
220 ORDER BY b.borrowid;
221
222
223
224
225

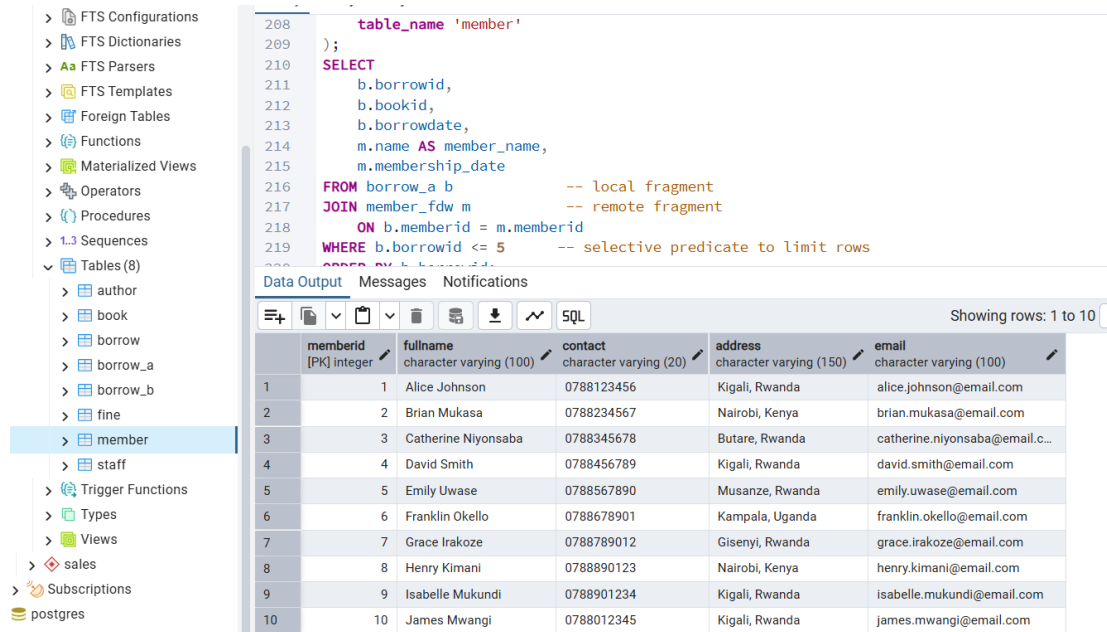
```

Data Output Messages Notifications

**No data output. Execute a query to get output.**

A3:

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



The screenshot shows a database IDE with a left sidebar containing a tree view of database objects. The 'member' table is selected. The main pane displays a SQL query:

```

208      table_name 'member'
209    );
210    SELECT
211      b.borrowid,
212      b.bookid,
213      b.borrowdate,
214      m.name AS member_name,
215      m.membership_date
216    FROM borrow_a b           -- local fragment
217   JOIN member_fdw m         -- remote fragment
218     ON b.memberid = m.memberid
219    WHERE b.borrowid <= 5     -- selective predicate to limit rows

```

Below the query, the 'Data Output' tab shows the results of the query. The table has 10 rows and 5 columns: memberid, fullname, contact, address, and email.

	memberid [PK] integer	fullname character varying (100)	contact character varying (20)	address character varying (150)	email character varying (100)
1	1	Alice Johnson	0788123456	Kigali, Rwanda	alice.johnson@email.com
2	2	Brian Mukasa	0788234567	Nairobi, Kenya	brian.mukasa@email.com
3	3	Catherine Niyonsaba	0788345678	Butare, Rwanda	catherine.niyonsaba@email.c...
4	4	David Smith	0788456789	Kigali, Rwanda	david.smith@email.com
5	5	Emily Uwase	0788567890	Musanze, Rwanda	emily.uwase@email.com
6	6	Franklin Okello	0788678901	Kampala, Uganda	franklin.okello@email.com
7	7	Grace Irakoze	0788789012	Gisenyi, Rwanda	grace.irakoze@email.com
8	8	Henry Kimani	0788890123	Nairobi, Kenya	henry.kimani@email.com
9	9	Isabelle Mukundi	0788901234	Kigali, Rwanda	isabelle.mukundi@email.com
10	10	James Mwangi	0788012345	Kigali, Rwanda	james.mwangi@email.com

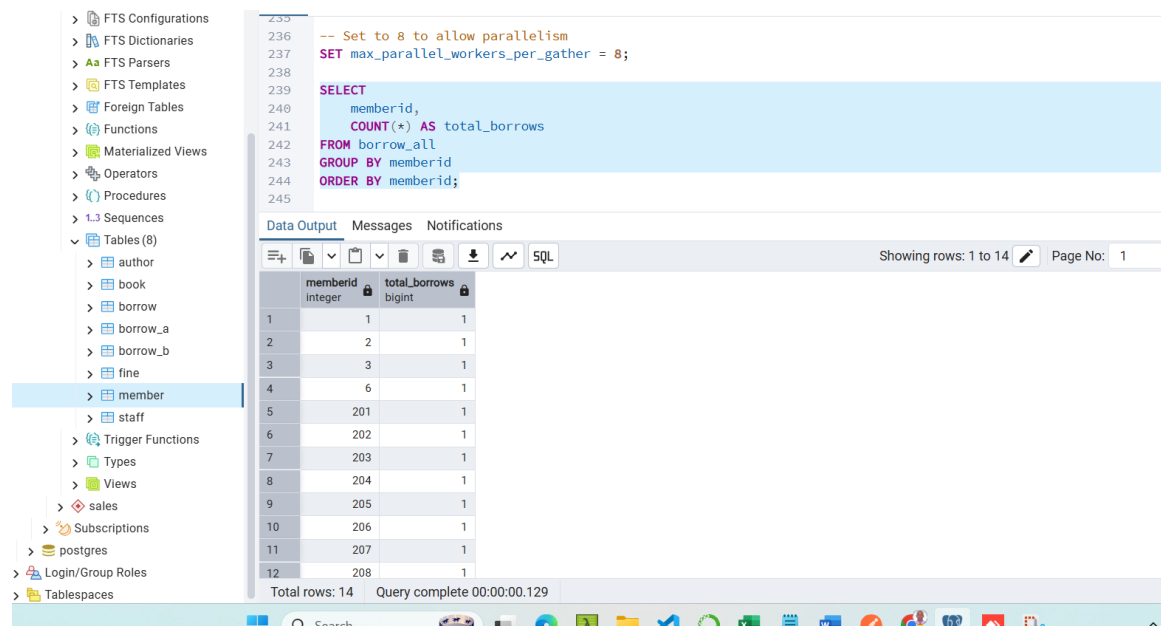
2. Run the same aggregation with `/*+ PARALLEL(Borrow_A,8) PARALLEL(Borrow_B,8) */` to force a parallel plan despite small size.

-- Check parallel settings

SHOW max\_parallel\_workers\_per\_gather; (2)

-- Set to 8 to allow parallelism

SET max\_parallel\_workers\_per\_gather = 8;



The screenshot shows a database IDE with a left sidebar containing a tree view of database objects. The 'member' table is selected. The main pane displays a SQL query:

```

235
236 -- Set to 8 to allow parallelism
237 SET max_parallel_workers_per_gather = 8;
238
239 SELECT
240   memberid,
241   COUNT(*) AS total_borrows
242 FROM borrow_all
243 GROUP BY memberid
244 ORDER BY memberid;
245

```

Below the query, the 'Data Output' tab shows the results of the query. The table has 12 rows and 2 columns: memberid and total\_borrows.

memberid integer	total_borrows bigint
1	1
2	1
3	1
4	6
5	201
6	202
7	203
8	204
9	205
10	206
11	207
12	208

Total rows: 14 Query complete 00:00:00.129

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

```

246
247 -- Enable AUTOTRACE with execution statistics
248 -- Run query (serial aggregation)
249 SELECT memberid, COUNT(*) AS total_borrows
250 FROM borrow_all
251 GROUP BY memberid;
252
253 -- Display execution plan
254 SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY());
255

```

memberid	total_borrows
1	202
2	210
3	3
4	204
5	209
6	205
7	201
8	6
9	2

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

```

255
256 -- Force serial execution
257 SET max_parallel_workers_per_gather = 0;
258
259 EXPLAIN ANALYZE
260 SELECT memberid, COUNT(*) AS total_borrows
261 FROM borrow_all
262 GROUP BY memberid;
263

```

QUERY PLAN
1 HashAggregate (cost=78.00..80.00 rows=200 width=12) (actual time=0.022..0.024 rows=14.00 loops=1)
2 Group Key: borrow_a.memberid
3 Batches: 1 Memory Usage: 32kB
4 Buffers: shared hit=2
5 -> Append (cost=0.00..63.50 rows=2900 width=4) (actual time=0.010..0.015 rows=14.00 loops=1)
6 Buffers: shared hit=2
7 -> Seq Scan on borrow_a (cost=0.00..24.50 rows=1450 width=4) (actual time=0.009..0.010 rows=9.00 loops=1)
8 Buffers: shared hit=1


A4.

```

264
265 A4 :
266 BEGIN
267 -- 1. Insert a local row into Borrow_A (Node_A)
268 INSERT INTO borrow_a (borrowid, bookid, memberid, borrowdate, returndate)
269 VALUES (11, 111, 211, TO_DATE('2025-10-28', 'YYYY-MM-DD'), TO_DATE('2025-11-07', 'YYYY-MM-DD'));
270
271 -- 2. Insert a remote row into Fine table on Node_B via proj_link
272 INSERT INTO fine@proj_link (fine_id, borrowid, amount, paid_status)
273 VALUES (101, 11, 500, 'N');
274
275 -- 3. Commit both inserts
276 COMMIT;
277
278 DBMS_OUTPUT.PUT_LINE('Local and remote rows inserted successfully.');
```

Query returned successfully in 100 msec.

2. Induce 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  $\leq 10$  committed row budget.



The screenshot shows the SQL Developer interface. On the left, the 'Tables (8)' folder is expanded, and the 'member' table is selected. The main window displays a SQL script with the following content:

```

283 /
284 -- =====
285 BEGIN;
286
287 -- Local insert
288 INSERT INTO borrow_a (borrowid, book_id, memberid, borrowdate, returndate)
289 VALUES (12, 112, 212, '2025-10-28', '2025-11-07');
290
291 -- Remote insert (simulate failure: table does not exist or foreign server disabled)
292 INSERT INTO fine_fdw (fine_id, borrow_id, amount, paid_status)
293 VALUES (102, 12, 600, 'N');
294
295 -- Commit transaction (will fail if fine_fdw insert fails)
296 COMMIT;
297

```

Below the script, the 'Messages' tab shows the following error:

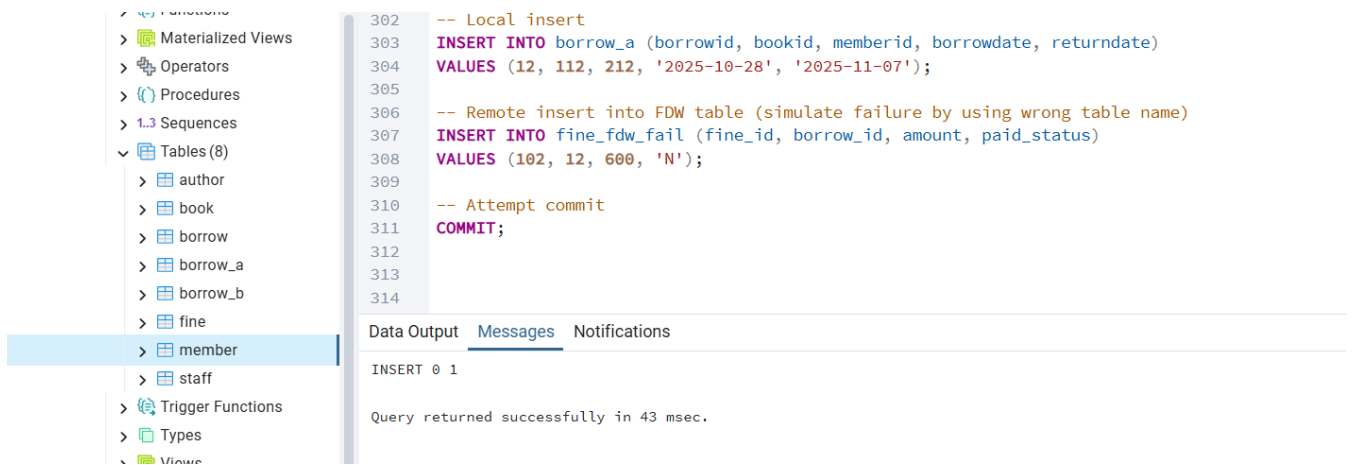
```

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

SQL state: 25P02

```

3. Query DBA\_2PC\_PENDING; then issue COMMIT FORCE or ROLLBACK FORCE; re-verify consistency on both nodes.



The screenshot shows the SQL Developer interface. On the left, the 'Tables (8)' folder is expanded, and the 'member' table is selected. The main window displays a SQL script with the following content:

```

302 -- Local insert
303 INSERT INTO borrow_a (borrowid, bookid, memberid, borrowdate, returndate)
304 VALUES (12, 112, 212, '2025-10-28', '2025-11-07');
305
306 -- Remote insert into FDW table (simulate failure by using wrong table name)
307 INSERT INTO fine_fdw_fail (fine_id, borrow_id, amount, paid_status)
308 VALUES (102, 12, 600, 'N');
309
310 -- Attempt commit
311 COMMIT;
312
313
314

```

Below the script, the 'Messages' tab shows the following output:

```

INSERT 0 1

Query returned successfully in 43 msec.

```

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



The screenshot shows the SQL Developer interface. On the left, the 'Columns (5)' folder is expanded, and the 'fineid' column is selected. The main window displays a SQL script with the following content:

```

332 BEGIN;
333
334 -- Local insert
335 INSERT INTO borrow_a (borrowid, bookid, memberid, borrowdate, returndate)
336 VALUES (13, 113, 213, '2025-10-28', '2025-11-08');
337
338 -- Remote insert via FDW
339 INSERT INTO fine_fdw (fineid, borrowid, amount, paymentdate, status)
340 VALUES (103, 13, 700, '2025-11-08', 'N');
341
342 -- Commit both inserts
343 COMMIT;
344
345

```

Below the script, the 'Messages' tab shows the following output:

```

WARNING: there is no transaction in progress
COMMIT

Query returned successfully in 44 msec.

```

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

1. Open Session 1 on Node\_A: UPDATE a single row in Borrow or Fine and keep the transaction open.

```
344 -- =====
345 BEGIN;
346 UPDATE borrow_a
347 SET return_date = '2025-11-10'
348 WHERE borrowid = 1;
349
350
```

Data Output Messages Notifications

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

SQL state: 25P02

```
345 BEGIN;
346 UPDATE borrow_a
347 SET return_date = '2025-11-10'
348 WHERE borrowid = 1;
349
350 SELECT * FROM borrow_a WHERE borrow_id = 1;
```

Data Output Messages Notifications

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

SQL state: 25P02

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

```
353 BEGIN;
354
355 -- Update a row in Borrow_A and keep transaction open
356 UPDATE borrow_a
357 SET return_date = '2025-11-10'
358 WHERE borrow_id = 1;
359 -- Do NOT COMMIT yet
```

Data Output Messages Notifications

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

3. Query lock views (DBA\_BLOCKERS/DBA\_WAITERS/V\$LOCK) from Node\_A to show the waiting session.

```
366
367 -- Identify which sessions are waiting and who blocks them
368 SELECT
369 blocked_locks.pid AS blocked_pid, blocked_activity.username AS blocked_user, blocked_activity.query AS blocked_query, blocking_locks
370 FROM pg_locks blocked_locks
371 JOIN pg_stat_activity blocked_activity ON blocked_locks.pid = blocked_activity.pid
372 JOIN pg_locks blocking_locks
373 ON blocked_locks.locktype = blocking_locks.locktype
374 AND blocked_locks.database IS NOT DISTINCT FROM blocking_locks.database
375 AND blocked_locks.relation IS NOT DISTINCT FROM blocking_locks.relation
376 AND blocked_locks.page IS NOT DISTINCT FROM blocking_locks.page
377 AND blocked_locks.tuple IS NOT DISTINCT FROM blocking_locks.tuple
378 AND blocking_locks.granted = true
379 WHERE blocked_locks.granted = false;
380
```

Data Output Messages Notifications

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

SQL state: 25P02

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

-- Node\_B schema

CREATE SCHEMA node\_b;

-- Node\_B Borrow table

```
CREATE TABLE node_b.borrow_a (  
    borrow_id INT PRIMARY KEY,  
    book_id INT,  
    member_id INT,  
    borrow_date DATE,  
    return_date DATE  
);
```

-- Insert sample row (reuse  $\leq 10$  rows)

INSERT INTO node\_b.borrow\_a VALUES (1, 101, 201, '2025-10-01', '2025-10-10');

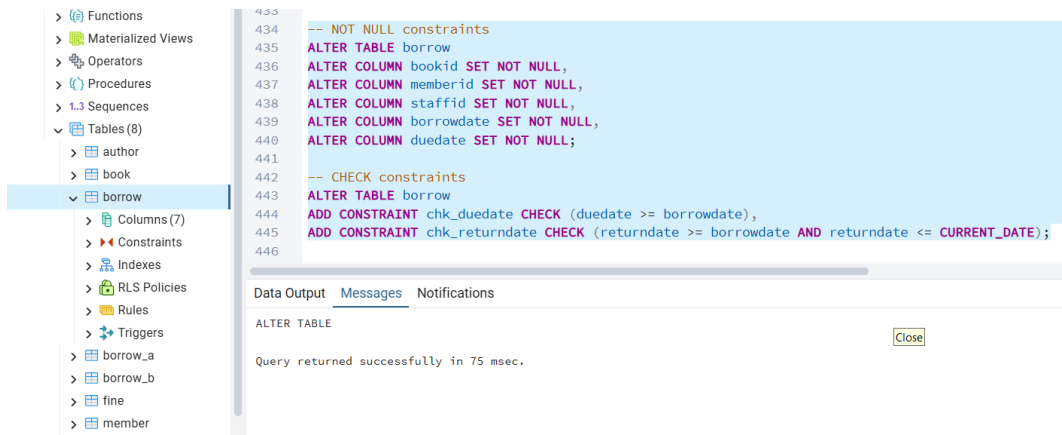
-- Create foreign table to simulate FDW

```
CREATE FOREIGN TABLE borrow_a_fdw (  
    borrow_id INT,  
    book_id INT,  
    member_id INT,  
    borrow_date DATE,  
    return_date DATE  
)  
SERVER postgres_fdw  
OPTIONS (schema_name 'node_b', table_name 'borrow_a');
```

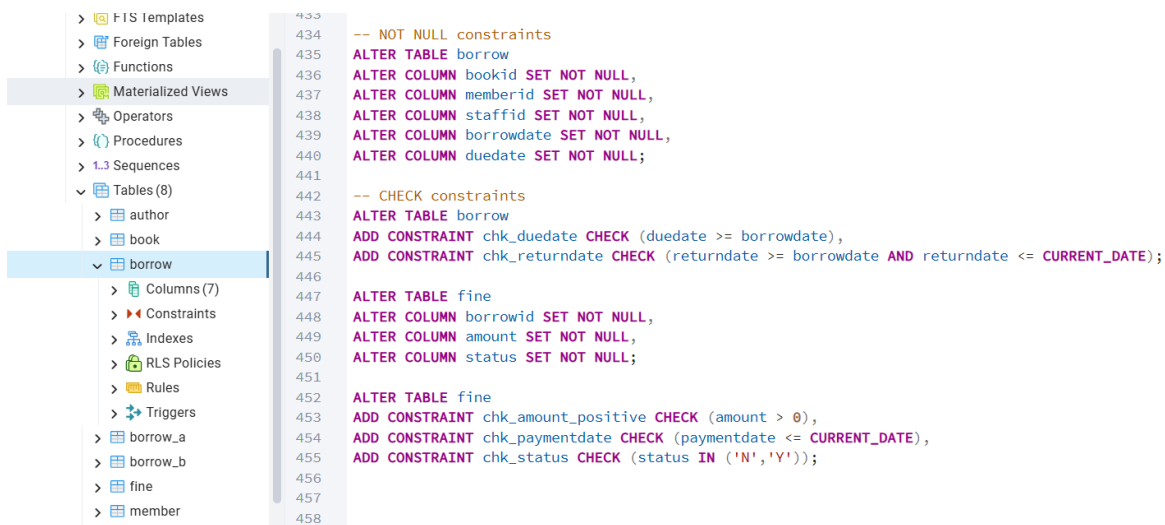


B6.

1. On tables Borrow and Fine, add/verify NOT NULL and domain CHECK constraints suitable for borrows, returns, and fines (e.g., positive amounts, valid statuses, date order).



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.



3. Show clean error handling for failing cases.

```

457 -- =====
458
459 DO $$
460 BEGIN
461 -- Attempt to insert an invalid row (e.g., duedate < borrowdate)
462 INSERT INTO borrow (borrowid, bookid, memberid, staffid, borrowdate, duedate, returndate)
463 VALUES (20, 101, 201, 301, '2025-10-28', '2025-10-25', '2025-11-05');
464
465 EXCEPTION
466 WHEN check_violation THEN
467 RAISE NOTICE 'Error: Borrow row violates CHECK constraints. Please verify dates.';
468 WHEN not_null_violation THEN
469 RAISE NOTICE 'Error: Borrow row violates NOT NULL constraints.';
470 WHEN others THEN
471 RAISE NOTICE 'Unexpected error: %', SQLERRM;
472 END;
473 $$;
474
475
476
477
478

```

Data Output Messages Notifications

NOTICE: Error: Borrow row violates CHECK constraints. Please verify dates.  
DO

Query returned successfully in 185 msec.

B7:

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

```

475 CREATE TABLE borrow_audit (
476     bef_total NUMERIC, -- total rows before change
477     aft_total NUMERIC, -- total rows after change
478     changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- time of change
479     key_col VARCHAR(64) -- primary key of affected row or descriptive key
480 );
481

```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 75 msec.

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

```

481 -- ==== Trigger=====
482 CREATE OR REPLACE FUNCTION recompute_borrow_totals()
483 RETURNS TRIGGER AS $$
484 BEGIN
485 -- Update Borrow totals based on current Fine table
486 UPDATE borrow b
487 SET total_fines = COALESCE(ft.total_amount, 0)
488 FROM (
489     SELECT borrowid, SUM(amount) AS total_amount
490     FROM fine
491     GROUP BY borrowid
492 ) AS ft
493 WHERE b.borrowid = ft.borrowid;
494
495 -- For Borrow rows with no fines, set total_fines to 0
496 UPDATE borrow
497 SET total_fines = 0
498 WHERE borrowid NOT IN (SELECT DISTINCT borrowid FROM fine);
499
500 RETURN NULL; -- statement-level triggers return NULL
501 END;

```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 70 msec.

3. Execute a small mixed DML script on CHILD affecting at most 4 rows in total; ensure net committed rows across the project remain  $\leq 10$ .

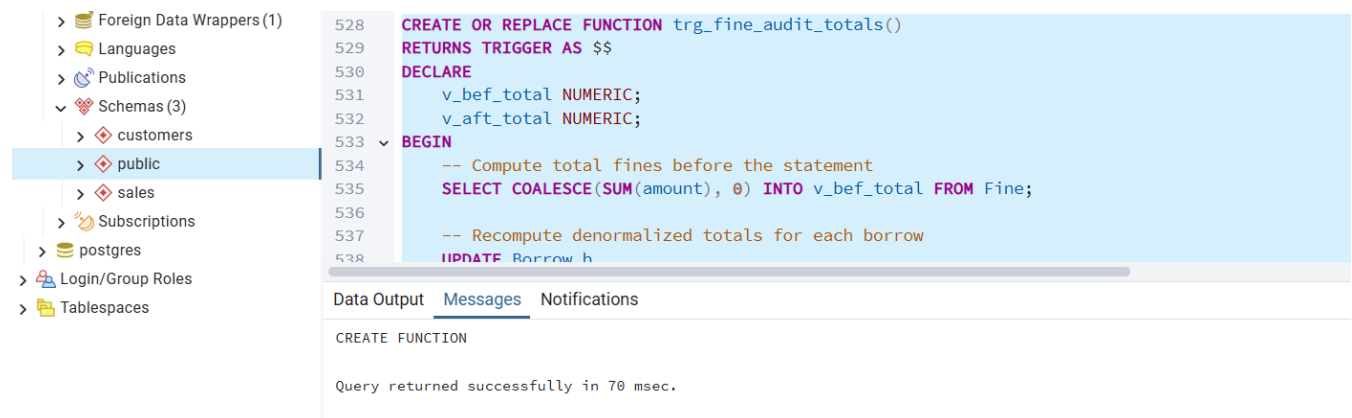


```
506 BEGIN;
507
508 -- 1 INSERT: add 2 new fines (2 rows)
509 INSERT INTO fine (fineid, borrowid, amount, paymentdate, status)
510 VALUES
511 (301, 1, 3000, '2025-10-25', 'N'),
512 (302, 2, 1500, '2025-10-27', 'N');
513
514 -- 2 UPDATE: modify 1 fine (1 row)
515 UPDATE fine
516 SET amount = 2000, status = 'Y', paymentdate = CURRENT_DATE
517 WHERE fineid = 302;
518
519 -- 3 DELETE: remove 1 fine (1 row)
520 DELETE FROM fine
521 WHERE fineid = 301;
522 COMMIT;
```

COMMIT

Query returned successfully in 94 msec.

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



```
528 CREATE OR REPLACE FUNCTION trg_fine_audit_totals()
529 RETURNS TRIGGER AS $$
530 DECLARE
531     v_bef_total NUMERIC;
532     v_aft_total NUMERIC;
533 BEGIN
534     -- Compute total fines before the statement
535     SELECT COALESCE(SUM(amount), 0) INTO v_bef_total FROM Fine;
536
537     -- Recompute denormalized totals for each borrow
538     UPDATE Borrow b
```

CREATE FUNCTION

Query returned successfully in 70 msec.

B8 :

```
CREATE TABLE HIER (
    parent_id VARCHAR(20),
    child_id VARCHAR(20),
    CONSTRAINT pk_hier PRIMARY KEY (parent_id, child_id)
);
```

- Publications
- Schemas (3)
  - customers
  - public
  - sales
- Subscriptions
- postgres
- Login/Group Roles
- Tablespaces

```

557 CREATE TABLE HIER (
558     parent_id VARCHAR(20),
559     child_id VARCHAR(20),
560     CONSTRAINT pk_hier PRIMARY KEY (parent_id, child_id)
561 );
562
563
564

```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 73 msec.

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

- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers (1)
- Languages
- Publications
- Schemas (3)
  - customers
  - public
  - sales
- Subscriptions
- postgres
- Login/Group Roles
- Tablespaces

```

557 CREATE TABLE HIER (
558     parent_id VARCHAR(20),
559     child_id VARCHAR(20),
560     CONSTRAINT pk_hier PRIMARY KEY (parent_id, child_id)
561 );
562
563 INSERT INTO HIER (parent_id, child_id) VALUES
564 ('BOOKS', 'FICTION'),
565 ('BOOKS', 'NONFICTION'),
566 ('FICTION', 'NOVEL'),
567 ('FICTION', 'SHORTSTORY'),
568 ('NONFICTION', 'BIOGRAPHY'),
569 ('NONFICTION', 'SCIENCE'),
570 ('SCIENCE', 'PHYSICS'),
571 ('SCIENCE', 'CHEMISTRY');
572
573
574
575

```

Data Output Messages Notifications

INSERT 0 8

Query returned successfully in 71 msec.

3. Write a recursive WITH query to produce (child\_id, root\_id, depth) and join to Borrow or its parent to compute rollups; return 6–10 rows total.

- FTS Parsers
- FTS Templates
- Foreign Tables
- Functions (5)
- Materialized Views
- Operators
- Procedures
- Sequences
- Tables (10)
  - author
  - book
  - borrow
  - borrow\_a
  - borrow\_audit
  - borrow\_b
  - fine
  - hier
  - member
  - staff
- Trigger Functions
- Types
- Views (1)
- sales
- Subscriptions
- postgres
- Login/Group Roles

```

571
572 WITH RECURSIVE hierarchy AS (
573     -- Base case: each node starts from its parent
574     SELECT
575         parent_id AS root_id,
576         child_id,
577         1 AS depth
578     FROM hier
579
580     UNION ALL
581
582     -- Recursive case: follow child-to-parent links
583     SELECT
584         h.root_id,
585         c.child_id,
586         h.depth + 1
587     FROM hier h
588         JOIN hier c ON c.parent_id = h.child_id
589 )
590
591

```

Data Output Messages Notifications

Showing rows: 1 to 16

Page No:

	root_id character varying (20)	child_id character varying (20)	depth integer
4	BOOKS	NOVEL	2
5	BOOKS	SCIENCE	2
6	BOOKS	BIOGRAPHY	2
7	BOOKS	CHEMISTRY	3
8	BOOKS	PHYSICS	3
9	FICTION	SHORTSTORY	1

Total rows: 16    Query complete 00:00:00.084

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

The screenshot shows a database IDE with a left sidebar containing a tree view of database objects. The 'Tables (10)' folder is expanded, showing tables: author, book, borrow, borrow\_a, borrow\_audit, borrow\_b, fine, hier, member, and staff. The main editor displays a SQL query starting with a recursive CTE named 'hier'. The query is as follows:

```

WITH RECURSIVE hier AS (
    SELECT parent_id AS root_id, child_id, 1 AS depth
    FROM hier
    UNION ALL
    SELECT h.root_id, c.child_id, h.depth + 1
    FROM hier h
    JOIN hier c ON h.child_id = c.parent_id
)
SELECT
    b.borrowid,
    b.bookid,
    h.root_id,
    h.depth,
    b.borrowdate,
    b.duedate
FROM borrow b
JOIN hierarchy h ON b.bookid = h.child_id
ORDER BY h.root_id, h.depth;

```

Below the query editor, the 'Data Output' tab is active, displaying an error message:

```

ERROR: recursive reference to query "hier" must not appear within its non-recursive term
LINE 3:     FROM hier
          ^
SQL state: 42P19

```

B9:

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

The screenshot shows a database IDE with a left sidebar containing a tree view of database objects. The 'Tables (10)' folder is expanded, showing tables: author, book, borrow, borrow\_a, borrow\_audit, borrow\_b, fine, hier, member, and staff. The main editor displays a SQL query that creates a table named 'triple' and inserts data into it. The query is as follows:

```

CREATE TABLE triple (
    s VARCHAR(64), -- subject
    p VARCHAR(64), -- predicate (relationship)
    o VARCHAR(64)  -- object
);

INSERT INTO triple (s, p, o) VALUES
('Book1', 'written_by', 'Author1'),
('Book1', 'belongs_to', 'Fiction'),
('Book2', 'written_by', 'Author2'),
('Book2', 'belongs_to', 'Science');

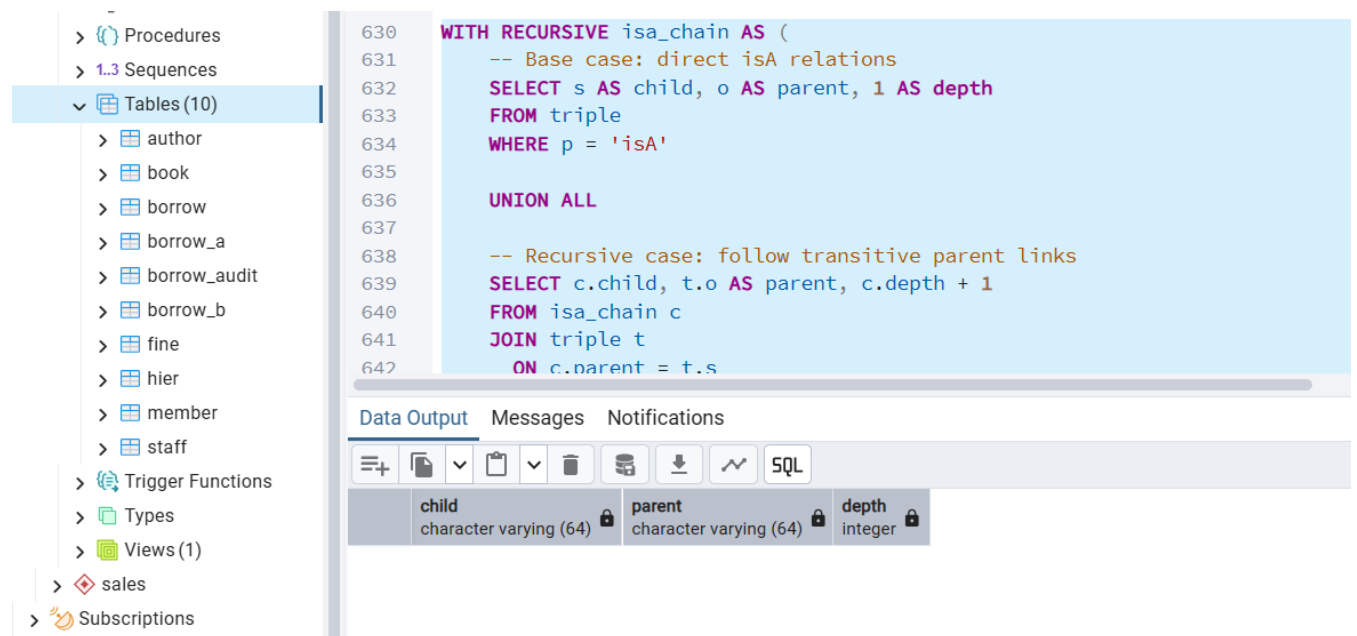
SELECT * FROM triple

```

Below the query editor, the 'Data Output' tab is active, displaying the results of the SELECT query:

	s character varying (64)	p character varying (64)	o character varying (64)
1	Book1	written_by	Author1
2	Book1	belongs_to	Fiction
3	Book2	written_by	Author2
4	Book2	belongs_to	Science

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



The screenshot shows a database IDE with a left sidebar containing a tree view of database objects. The 'Tables (10)' folder is expanded, showing tables like 'author', 'book', 'borrow', 'borrow\_a', 'borrow\_audit', 'borrow\_b', 'fine', 'hier', 'member', 'staff', 'Trigger Functions', 'Types', 'Views (1)', 'sales', and 'Subscriptions'. The main editor displays an SQL query for a recursive inference query named 'isa\_chain'.

```

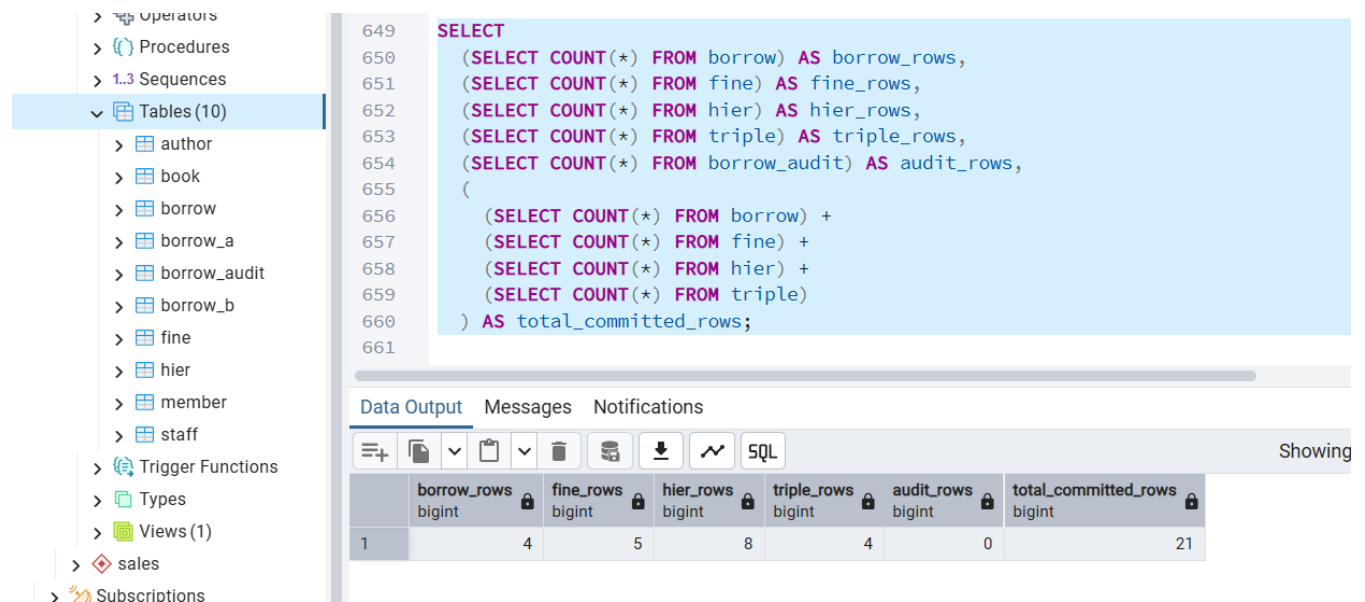
630 WITH RECURSIVE isa_chain AS (
631     -- Base case: direct isA relations
632     SELECT s AS child, o AS parent, 1 AS depth
633     FROM triple
634     WHERE p = 'isA'
635
636     UNION ALL
637
638     -- Recursive case: follow transitive parent links
639     SELECT c.child, t.o AS parent, c.depth + 1
640     FROM isa_chain c
641     JOIN triple t
642         ON c.parent = t.s

```

Below the query editor, the 'Data Output' tab is active, showing the schema of the query results:

child	parent	depth
character varying (64)	character varying (64)	integer

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



The screenshot shows the same database IDE with the same tree view. The main editor displays an SQL query to count committed rows across various tables.

```

649 SELECT
650     (SELECT COUNT(*) FROM borrow) AS borrow_rows,
651     (SELECT COUNT(*) FROM fine) AS fine_rows,
652     (SELECT COUNT(*) FROM hier) AS hier_rows,
653     (SELECT COUNT(*) FROM triple) AS triple_rows,
654     (SELECT COUNT(*) FROM borrow_audit) AS audit_rows,
655     (
656         (SELECT COUNT(*) FROM borrow) +
657         (SELECT COUNT(*) FROM fine) +
658         (SELECT COUNT(*) FROM hier) +
659         (SELECT COUNT(*) FROM triple)
660     ) AS total_committed_rows;
661

```

Below the query editor, the 'Data Output' tab is active, showing the results of the query:

	borrow_rows	fine_rows	hier_rows	triple_rows	audit_rows	total_committed_rows
	bigint	bigint	bigint	bigint	bigint	bigint
1	4	5	8	4	0	21

B10:

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

```

654 (SELECT COUNT(*) FROM borrow_audit) AS audit_rows,
655 (
656 (SELECT COUNT(*) FROM borrow) +
657 (SELECT COUNT(*) FROM fine) +
658 (SELECT COUNT(*) FROM hier) +
659 (SELECT COUNT(*) FROM triple)
660 ) AS total_committed_rows;
661
662 -- =====
663 CREATE TABLE business_limits (
664 rule_key VARCHAR(64),
665 threshold NUMERIC,
666 active CHAR(1) CHECK (active IN ('Y', 'N'))
667 );
668

```

- Implement function `fn_should_alert(...)` that reads `BUSINESS_LIMITS` and inspects current data in Fine or Borrow to decide a violation (return 1/0).

```

659 (SELECT COUNT(*) FROM triple)
660 ) AS total_committed_rows;
661
662 -- =====
663 CREATE TABLE business_limits (
664 rule_key VARCHAR(64),
665 threshold NUMERIC,
666 active CHAR(1) CHECK (active IN ('Y', 'N'))
667 );
668
669 CREATE OR REPLACE FUNCTION fn_should_alert(rule_key_input VARCHAR)
670 RETURNS INT AS $$
671 DECLARE
672 v_threshold NUMERIC;
673 v_max_amount NUMERIC;
674 BEGIN
675 -- Get the active threshold for the rule
676 SELECT threshold
677 INTO v_threshold
678 FROM business_limits
679 WHERE rule_key = rule_key_input
680 AND active = 'Y';
681

```

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

```

695 CREATE OR REPLACE FUNCTION trg_fine_business_limit()
696 RETURNS TRIGGER AS $$
697 BEGIN
698 -- Call the business rule function
699 IF fn_should_alert('MAX_FINE_AMOUNT') = 1 THEN
700 RAISE EXCEPTION 'Business rule violated: Fine amount exceeds threshold!';
701 END IF;
702
703 RETURN NEW; -- required for BEFORE triggers
704
705 END;
706 $$ LANGUAGE plpgsql;
707
708 -- Attach the trigger to Fine
709 CREATE TRIGGER trg_fine_check_limit
710 BEFORE INSERT OR UPDATE ON fine
711 FOR EACH ROW
712 EXECUTE FUNCTION trg_fine_business_limit();
713

```

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

- > Domains
- > FTS Configurations
- > FTS Dictionaries
- > FTS Parsers
- > FTS Templates
- > Foreign Tables
- ✓ Functions (6)
  - fn\_should\_alert(rule\_key\_input character varying)
  - postgres\_fdw\_disconnect(text)
  - postgres\_fdw\_disconnect\_all()
  - postgres\_fdw\_get\_connections(check\_conn boolean, OUT serv
  - postgres\_fdw\_handler()
  - postgres\_fdw\_validator(text[], oid)
- > Materialized Views
- > Operators
- > Procedures
- > 1..3 Sequences
- > Tables
- > Trigger Functions
- > Types
- > Views

```
707
708 -- Attach the trigger to Fine
709 CREATE TRIGGER trg_fine_check_limit
710 BEFORE INSERT OR UPDATE ON fine
711 FOR EACH ROW
712 EXECUTE FUNCTION trg_fine_business_limit();
713 -- =====
714
715 -- BEGIN a transaction for demonstration
716 BEGIN;
717
718
```

Data Output Messages Notifications

ERROR: syntax error at or near "RAISE"  
LINE 4: RAISE NOTICE 'PASSING CASE 1: inserted fine 3000';  
^

SQL state: 42601  
Character: 149