

STUDENT 31: SACCO INSURANCE & MEMBER EXTENSION

A1.

```
-- Create table Claim_A
CREATE TABLE Claim_A (
    ClaimID INT PRIMARY KEY,
    MemberName VARCHAR(50),
    ClaimAmount NUMERIC(10,2),
    ClaimDate DATE
);
INSERT INTO Claim_A VALUES
(1, 'Alice', 2500.00, '2025-01-15'),
(2, 'Ben', 1800.00, '2025-02-05'),
(3, 'Carine', 2200.00, '2025-03-02'),
(4, 'David', 3100.00, '2025-04-10'),
(5, 'Eva', 2700.00, '2025-05-11');
COMMIT;

-- Install extension (only once)
CREATE EXTENSION IF NOT EXISTS postgres_fdw;

-- Create a server link to Node_B
CREATE SERVER proj_link
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'localhost', dbname 'node_b', port '5432');

-- Create user mapping (adjust username/password)
CREATE USER MAPPING FOR CURRENT_USER
SERVER proj_link
OPTIONS (user 'postgres', password '1234');

CREATE FOREIGN TABLE Claim_B_remote (
    ClaimID INT,
```

sacco/postgres@... X sacco_01/postgre... X sacco_02/postgre... X claim_A/postgres@...

claim_A/postgres@PostgreSQL 17

No limit

Query Query History

```
29
30 CREATE FOREIGN TABLE Claim_B_remote (
31     ClaimID INT;
32     MemberName VARCHAR(50),
33     ClaimAmount NUMERIC(10,2),
34     ClaimDate DATE
35 )
36 SERVER proj_link
37 OPTIONS (table_name 'claim_b');
38
39 CREATE OR REPLACE VIEW Claim_ALL AS
40 SELECT claimid, membername, claimamount, claimdate FROM claim_a
41 UNION ALL
42 SELECT claimid, membername, claimamount, claimdate FROM claim_b_remote;
43
44 SELECT COUNT(*) AS Count_A FROM Claim_A;
```

Data Output Messages Notifications

count_a
bigint

1	12
---	----

```

SELECT COUNT(*) AS Count_A FROM Claim_A;
SELECT COUNT(*) AS Count_B FROM Claim_B_remote;
SELECT COUNT(*) AS Count_All FROM Claim_ALL;

SELECT SUM(MOD(ClaimID, 97)) AS checksum_A FROM Claim_A;

SELECT SUM(MOD(ClaimID, 97)) AS checksum_All FROM Claim_ALL;

```

```

CREATE TABLE IF NOT EXISTS Claim_B (
    ClaimID INT PRIMARY KEY,
    MemberName VARCHAR(50) NOT NULL,
    ClaimAmount NUMERIC(10,2) CHECK (ClaimAmount >= 0),
    ClaimDate DATE NOT NULL
);

-- =====
-- Insert sample data into Claim_B
-- =====

INSERT INTO Claim_B (ClaimID, MemberName, ClaimAmount, ClaimDate) VALUES
(6, 'Frank', 1900.00, '2025-06-02'),
(7, 'Grace', 2500.00, '2025-06-12'),

COMMIT;

-- =====
-- Verify inserted data
-- =====

SELECT * FROM Claim_B;

SELECT COUNT(*) AS Count_B FROM Claim_B_remote;

SELECT SUM(MOD(ClaimID, 97)) AS checksum_B FROM Claim_B_remote;

```

Claim_B/postgres@PostgreSQL 17

Query Query History

```

20
21 INSERT INTO Claim_B (ClaimID, MemberName, ClaimAmount, ClaimDate) VALUES
22 (6, 'Frank', 1900.00, '2025-06-02'),
23 (7, 'Grace', 2500.00, '2025-06-12'),
24 (8, 'Hannah', 3000.00, '2025-07-09'),
25 (9, 'Ivan', 2800.00, '2025-08-01'),
26 (10, 'Jack', 3500.00, '2025-09-17');
27
28 COMMIT;
29
30 -- =====
31 -- Verify inserted data
32 -- =====
33
34 SELECT * FROM Claim_B;
35

```

Data Output Messages Notifications

Showing rows: 1 to 6

	claimid integer	policyid integer	datefiled date	amountclaimed numeric (10,2)	status character varying (20)
1	3	102	2025-01-...	175000.00	pending
2	1	201	2025-01-...	180000.00	approved
3	3	202	2025-02-...	160000.00	pending
4	5	203	2025-03-...	220000.00	rejected
5	7	204	2025-04-...	210000.00	pending
6	9	205	2025-05-...	140000.00	approved

```

SERVER proj_link
OPTIONS (table_name 'claim_b');

CREATE OR REPLACE VIEW Claim_ALL AS
SELECT claimid, membername, claimamount, claimdate FROM claim_a
UNION ALL
SELECT claimid, membername, claimamount, claimdate FROM claim_b_remote;

```

A2.

```

-- Step 1: Enable the FDW extension
CREATE EXTENSION IF NOT EXISTS postgres_fdw;

-- Step 2: Create a server link named 'proj_link'
CREATE SERVER proj_link
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'localhost', dbname 'node_b', port '5432');

-- Step 3: Create user mapping for access
CREATE USER MAPPING FOR CURRENT_USER
SERVER proj_link
OPTIONS (user 'postgres', password 'yourpassword');

```

Output Messages Notifications

user mapping for "postgres" already exists for server "proj_link"

```
1
2 CREATE TABLE Member (
3     MemberID INT PRIMARY KEY,
4     MemberName VARCHAR(50),
5     JoinDate DATE,
6     Address VARCHAR(100)
7 );
8
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 217 msec.

```
13 CREATE FOREIGN TABLE Member_remote (
14     memberid INT,
15     membername VARCHAR(50),
16     joindate DATE,
17     address VARCHAR(100)
18 )
19 SERVER proj_link
20 OPTIONS (table_name 'member');
21
```

Data Output Messages Notifications

CREATE FOREIGN TABLE

Query returned successfully in 828 msec.

```
8 CREATE TABLE Officer (
9     OfficerID INT PRIMARY KEY,
10    OfficerName VARCHAR(50),
11    Department VARCHAR(50),
12    MemberID INT
13 );
14
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 250 msec.

```

5 CREATE FOREIGN TABLE Officer_remote (
6     officerid INT,
7     officername VARCHAR(50),
8     department VARCHAR(50),
9     memberid INT
10 )
11 SERVER proj_link
12 OPTIONS (table_name 'officer');
13

```

Data Output Messages Notifications

CREATE FOREIGN TABLE

Query returned successfully in 124 msec.

3.

```

14 SELECT
15     a.claimid,
16     a.membername,
17     a.claimamount,
18     o.officername,
19     o.department
20 FROM claim_a a
21 JOIN officer_remote o
22     ON a.claimid = o.memberid -- adjust join key as needed
23 WHERE a.claimamount > 1500
24 LIMIT 10;
25

```

Data Output Messages Notifications

ERROR: column a.membername does not exist
 LINE 3: a.membername,

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

1.

```

CREATE OR REPLACE VIEW Claim_ALL AS
SELECT * FROM Claim_A
UNION ALL
SELECT * FROM Claim_B;

```

```

-- SERIAL aggregation (normal)
SELECT MemberName,
       SUM(ClaimAmount) AS TotalAmount,
       COUNT(*) AS NumClaims
FROM Claim_ALL
GROUP BY MemberName
ORDER BY MemberName;

```


2. Run the same aggregation with **parallel hint**

```
-- PARALLEL aggregation (forced)
SELECT /*+ PARALLEL(Claim_A,8) PARALLEL(Claim_B,8) */
      MemberName,
      SUM(ClaimAmount) AS TotalAmount,
      COUNT(*) AS NumClaims
FROM Claim_ALL
GROUP BY MemberName
ORDER BY MemberName;
```

3. Capture execution plans with DBMS_XPLAN and AUTOTRAC

```
SET AUTOTRACE ON STATISTICS

-- Run serial version
SELECT MemberName, SUM(ClaimAmount), COUNT(*)
FROM Claim_ALL
GROUP BY MemberName;

-- Run parallel version
SELECT /*+ PARALLEL(Claim_A,8) PARALLEL(Claim_B,8) */
      MemberName, SUM(ClaimAmount), COUNT(*)
FROM Claim_ALL
GROUP BY MemberName;

SET AUTOTRACE OFF;
```

a Output Messages Notifications

ORA-00904: syntax error at or near "ON"

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST'));
```

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

1. Clean run: insert one local and one remote row

```
SET SERVEROUTPUT ON;

DECLARE
BEGIN
  -- Insert local row
  INSERT INTO Claim_A (ClaimID, MemberName, ClaimAmount, ClaimDate)
  VALUES (11, 'Laura', 3200, SYSDATE);

  -- Insert remote row
  INSERT INTO Payment@proj_link (PaymentID, ClaimID, Amount, PaymentDate)
  VALUES (101, 11, 3200, SYSDATE);

  -- Commit distributed transaction
  COMMIT;
```

```

-- Insert remote row
INSERT INTO Payment@proj_link (PaymentID, ClaimID, Amount, PaymentDate)
VALUES (101, 11, 3200, SYSDATE);

-- Commit distributed transaction
COMMIT;

DBMS_OUTPUT.PUT_LINE('Distributed transaction committed successfully.');
```

EXCEPTION

```

    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error occurred, transaction rolled back: ' || SQLERRM);
END;
```

ta Output Messages Notifications

ROR: syntax error at or near "ON"

NE 2: SET SERVEROUTPUT ON;

2. Induce a failure to create an in-doubt transaction

```

DECLARE
BEGIN
    -- Insert local row
    INSERT INTO Claim_A (ClaimID, MemberName, ClaimAmount, ClaimDate)
    VALUES (12, 'Mark', 2800, SYSDATE);

    -- Insert remote row (simulate failure)
    RAISE_APPLICATION_ERROR(-20001, 'Simulated failure after local insert');

    -- Insert into remote Payment@proj_link (this will NOT execute)
    -- INSERT INTO Payment@proj_link (...) VALUES (...);

    COMMIT; -- will never reach here
3      -- Insert remote row (simulate failure)
9      RAISE_APPLICATION_ERROR(-20001, 'Simulated failure after local insert');
9
1      -- Insert into remote Payment@proj_link (this will NOT execute)
2      -- INSERT INTO Payment@proj_link (...) VALUES (...);
3
4      COMMIT; -- will never reach here
5
EXCEPTION
5      WHEN OTHERS THEN
7          DBMS_OUTPUT.PUT_LINE('Simulated failure, distributed transaction incomplete: ' || SQLERRM);
3
END;
3
/
3
3
```

3. Query pending 2PC transactions

```

1
2      -- On Node_A
3      SELECT LOCAL_TRAN_ID, STATE, START_TIME
4      FROM DBA_2PC_PENDING;
5
```


4. Repeat clean run to verify no pending transactions

```

1  -- Insert another local + remote row
2  DECLARE
3  BEGIN
4      INSERT INTO Claim_A (ClaimID, MemberName, ClaimAmount, ClaimDate)
5      VALUES (13, 'Nina', 3000, SYSDATE);
6
7      INSERT INTO Payment@proj_link (PaymentID, ClaimID, Amount, PaymentDate)
8      VALUES (102, 13, 3000, SYSDATE);
9
10     COMMIT;
11
12     DBMS_OUTPUT.PUT_LINE('Clean distributed transaction committed successfully.');
```

```

13 END;
14 /
15
16 -- Verify no pending transactions
17 SELECT * FROM DBA_2PC_PENDING;
18
```

Data Output Messages Notifications

```
ERROR: syntax error at or near "INSERT"
LINE 4:     INSERT INTO Claim_A (ClaimID, MemberName, ClaimAmount, C...
          ^
```

A5.

1. Open Session 1 (Node_A)

```
1  -- Session 1 on Node_A
2  UPDATE Claim_A
3  SET ClaimAmount = ClaimAmount + 100
4  WHERE ClaimID = 1; -- pick an existing row
5
6  -- Keep transaction open (do not commit yet)
7  -- Check the row is locked
8  SELECT * FROM Claim_A WHERE ClaimID = 1 FOR UPDATE;
9
10
11
```

[Data Output](#) [Messages](#) [Notifications](#)

SQL

2. Open Session 2 (Node_B)

```
1  -- Session 2 (Node_B or Node_A using remote table)
2  UPDATE Claim_A@proj_link
3  SET ClaimAmount = ClaimAmount + 50
4  WHERE ClaimID = 1;
5
```

3. Check locks from Node A

```

-- Who is blocking and who is waiting?
SELECT s.sid, s.serial#, l.type, l.id1, l.id2, l.lmode, l.request, l.block
FROM v$lock l
JOIN v$session s ON l.sid = s.sid
WHERE l.id1 IN (
    SELECT object_id FROM dba_objects WHERE object_name = 'CLAIM_A'
);

-- Optional: use DBA_BLOCKERS / DBA_WAITERS
SELECT * FROM dba_blockers;
SELECT * FROM dba_waiters;

```

4. Release the lock

```

1
2 -- Who is blocking and who is waiting?
3 SELECT s.sid, s.serial#, l.type, l.id1, l.id2, l.lmode, l.request, l.block
4 FROM v$lock l
5 JOIN v$session s ON l.sid = s.sid
6 WHERE l.id1 IN (
7     SELECT object_id FROM dba_objects WHERE object_name = 'CLAIM_A'
8 );
9
10 -- Optional: use DBA_BLOCKERS / DBA_WAITERS
11 SELECT * FROM dba_blockers;
12 SELECT * FROM dba_waiters;
13 COMMIT; -- releases the row-level lock
14
15
16

```

Data Output Messages Notifications

WARNING: there is no transaction in progress
COMMIT

Query returned successfully in 299 msec.

```

-- Optional: use DBA_BLOCKERS / DBA_WAITERS
SELECT * FROM dba_blockers;
SELECT * FROM dba_waiters;
COMMIT; -- releases the row-level lock

-- Verify changes applied by both sessions
SELECT ClaimID, ClaimAmount FROM Claim_A WHERE ClaimID = 1;

```

a Output Messages Notifications

OR: column "claimamount" does not exist
E 2: SELECT ClaimID, ClaimAmount FROM Claim_A WHERE ClaimID = 1;

B6: Declarative Rules Hardening (≤10 committed rows)

1. On tables Claim and Payment, add/verify NOT NULL and domain CHECK constraints suitable

```

1 CREATE TABLE Claim (
2     ClaimID SERIAL PRIMARY KEY,
3     MemberID INT NOT NULL,
4     ClaimAmount NUMERIC(10,2) NOT NULL CHECK (ClaimAmount > 0),
5     ClaimDate DATE NOT NULL,
6     Status VARCHAR(20) NOT NULL CHECK (Status IN ('Pending', 'Approved', 'Rejected')),
7     FOREIGN KEY (MemberID) REFERENCES Member(MemberID)
8 );
9
10

```

Data Output [Messages](#) Notifications

ERROR: relation "claim" already exists

SQL state: 42P07

2.2

```

9 CREATE TABLE Payment (
10     PaymentID SERIAL PRIMARY KEY,
11     ClaimID INT NOT NULL,
12     PaymentAmount NUMERIC(10,2) NOT NULL CHECK (PaymentAmount > 0),
13     PaymentDate DATE NOT NULL,
14     FOREIGN KEY (ClaimID) REFERENCES Claim(ClaimID)
15 );
16
17

```

Data Output [Messages](#) Notifications

CREATE TABLE

Query returned successfully in 4 secs 721 msec.

2. Sample INSERTs

```

CREATE TABLE claim (
    claimid SERIAL PRIMARY KEY,
    memberid INT NOT NULL,
    claimamount NUMERIC(10,2) NOT NULL CHECK (claimamount > 0),
    claimdate DATE NOT NULL,
    status VARCHAR(20) NOT NULL CHECK (status IN ('Pending', 'Approved', 'Rejected'))
);

INSERT INTO claim (memberid, claimamount, claimdate, status) VALUES
(1, 5000, '2025-10-01', 'Pending'),
(2, 1200, '2025-10-15', 'Approved');

```

```

1 DO $$
2 BEGIN
3 BEGIN
4     INSERT INTO Claim (MemberID, ClaimAmount, ClaimDate, Status) VALUES
5         (3, -100, '2025-10-20', 'Pending'); -- Negative amount
6 EXCEPTION WHEN OTHERS THEN
7     RAISE NOTICE 'Claim failed: Negative amount';
8 END;
9
10 BEGIN
11     INSERT INTO Claim (MemberID, ClaimAmount, ClaimDate, Status) VALUES
12         (4, 2000, '2025-10-25', 'Unknown'); -- Invalid status
13 EXCEPTION WHEN OTHERS THEN
14     RAISE NOTICE 'Claim failed: Invalid status';
15 END;
16 END $$;
17
18

```

Data Output Messages Notifications

ERROR: unterminated dollar-quoted string at or near "\$\$;"
 LINE 1: END \$\$;

```

INSERT INTO Payment (ClaimID, PaymentAmount, PaymentDate) VALUES
(1, 5000, '2025-10-05'),
(2, 1200, '2025-10-16');

```

Output Messages Notifications

ER: insert or update on table "payment" violates foreign key constraint "payment_claimid_fkey" (claimid)=(1) is not present in table "claim".

```

1 DO $$
2 BEGIN
3 BEGIN
4 BEGIN
5     INSERT INTO Payment (ClaimID, PaymentAmount, PaymentDate) VALUES
6         (1, -500, '2025-10-06'); -- Negative payment
7 EXCEPTION WHEN OTHERS THEN
8     RAISE NOTICE 'Payment failed: Negative amount';
9 END;
10
11 BEGIN
12     INSERT INTO Payment (ClaimID, PaymentAmount, PaymentDate) VALUES
13         (2, 1500, '2025-10-10'); -- Amount exceeds claim
14 EXCEPTION WHEN OTHERS THEN
15     RAISE NOTICE 'Payment failed: Amount exceeds claim';
16 END;
17 END $$;
18

```

Data Output Messages Notifications

NOTICE: Payment failed: Negative amount
 NOTICE: Payment failed: Amount exceeds claim
 DO

Query returned successfully in 5 secs 127 msec.

3: Clean Error Handling
DO \$\$... EXCEPTION ... END \$\$

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

1: Create Audit Table

```
1  
2 CREATE TABLE claim_audit (  
3     bef_total NUMERIC(12,2),  
4     aft_total NUMERIC(12,2),  
5     changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
6     key_col VARCHAR(64)  
7 );  
8  
9  
10
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 4 secs 267 msec.

2: Implement Statement-Level AFTER Trigger on Payment

```
Query History
1
2 ALTER TABLE claim ADD COLUMN total_payment NUMERIC(12,2) DEFAULT 0;
3
4
5
```

Data Output Messages Notifications

ALTER TABLE

Query returned successfully in 4 secs 83 msec.

2. Create the trigger function:

```
1
2 CREATE OR REPLACE FUNCTION update_claim_totals()
3 RETURNS TRIGGER AS $$
4 DECLARE
5     before_total NUMERIC(12,2);
6     after_total NUMERIC(12,2);
7 BEGIN
8     -- Compute total BEFORE change
9     SELECT COALESCE(SUM(paymentamount),0) INTO before_total FROM payment;
10
11     -- Recompute total payments per claim
12     UPDATE claim c
13     SET total_payment = (
14         SELECT COALESCE(SUM(p.paymentamount),0)
15         FROM payment p
16         WHERE p.claimid = c.claimid
17     );
18
```

Data Output Messages Notifications

UPDATE 0

Query returned successfully in 274 msec.

3. Attach the **AFTER statement-level trigger** to Payment:


```
1
2
3     CREATE TRIGGER trg_update_claim_totals
4 AFTER INSERT OR UPDATE OR DELETE ON payment
5 FOR EACH STATEMENT
6 EXECUTE FUNCTION update_claim_totals();
7
8
9
```

3: Test with Small Mixed DML

```
1
2 -- Passing INSERTs
3 INSERT INTO payment (claimid, paymentamount, paymentdate) VALUES
4 (1, 1000, '2025-10-10'),
5 (2, 500, '2025-10-15');
6
7 -- UPDATE affecting 1 row
8 UPDATE payment SET paymentamount = 1200 WHERE paymentid = 1;
9
10 -- DELETE affecting 1 row
11 DELETE FROM payment WHERE paymentid = 2;
12
13
14
15
```

Data Output Messages Notifications

DELETE 0

Query returned successfully in 188 msec.

4: Verify Audit Logging

Query Query History

```
1 SELECT * FROM claim_audit ORDER BY changed_at;
2
3
4
5
6
```

Data Output Messages Notifications

bef_total	aft_total	changed_at	key_col
numeric (12,2)	numeric (12,2)	timestamp without time zone	character varying (64)

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

1: Create Hierarchy Table

```
1 CREATE TABLE hier (
2     parent_id INT,
3     child_id INT,
4     PRIMARY KEY (child_id)
5 );
6
7
8
9
10
11
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 237 msec.

2: Insert 6–10 Rows Forming a 3-Level Hierarchy

```

5
6 INSERT INTO hier (parent_id, child_id) VALUES
7 (NULL, 1),    -- root node
8 (1, 2),       -- level 2
9 (1, 3),
10 (2, 4),       -- level 3
11 (2, 5),
12 (3, 6);       -- level 3
13

```

Data Output Messages Notifications

INSERT 0 6

Query returned successfully in 120 msec.

3: Recursive WITH Query

The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, query execution, and settings. Below the toolbar, there are tabs for 'Query', 'Query History', and 'Scratchpad'. The 'Query' tab is active, displaying a SQL query that uses a recursive Common Table Expression (CTE) to calculate the depth of each node in a hierarchy.

```

1
2
3
4
5
6 WITH RECURSIVE hier_cte AS (
7   -- Base level: roots
8   SELECT
9     child_id,
10    child_id AS root_id,
11    0 AS depth
12   FROM hier
13   WHERE parent_id IS NULL
14
15   UNION ALL
16
17   -- Recursive step: attach children
18   SELECT

```

Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing the results of the query in a table format. The table has three columns: 'child_id', 'root_id', and 'depth', all of which are integers. The results show a hierarchy starting from a root node (1) and branching out to child nodes (2, 3, 4, 5, 6) with their respective depths.

	child_id integer	root_id integer	depth integer
1	1	1	0
2	2	1	1
3	3	1	1
4	4	1	2
5	5	1	2
6	6	1	2

At the bottom right of the 'Data Output' tab, it says 'Showing rows: 1 to 6' and 'Page No: 1'.

4: Join to Claim for Rollups

Query	Query History
1	WITH RECURSIVE hier_cte AS (
2	SELECT
3	child_id,
4	child_id AS root_id,
5	0 AS depth
6	FROM hier
7	WHERE parent_id IS NULL
8	
9	UNION ALL
10	
11	SELECT
12	h.child_id,
13	cte.root_id,
14	cte.depth + 1
15	FROM hier h
16	JOIN hier_cte cte ON h.parent_id = cte.child_id
17)
18	SELECT

Query	Query History
10	
11	SELECT
12	h.child_id,
13	cte.root_id,
14	cte.depth + 1
15	FROM hier h
16	JOIN hier_cte cte ON h.parent_id = cte.child_id
17)
18	SELECT
19	cte.child_id,
20	cte.root_id,
21	cte.depth,
22	COALESCE(SUM(cl.claimamount),0) AS total_claim
23	FROM hier_cte cte
24	LEFT JOIN claim cl ON cl.memberid = cte.child_id
25	GROUP BY cte.child_id, cte.root_id, cte.depth
26	ORDER BY cte.root_id, cte.depth;
27	

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

1: Create TRIPLE Table

Query

Query History

1

2

3

4

5

6

7

8

9

CREATE TABLE triple (
s VARCHAR(64),
p VARCHAR(64),
o VARCHAR(64)
);

Data Output

Messages

Notifications

CREATE TABLE

Query returned successfully in 166 msec.

2: Insert 8–10 Domain Facts

Query

Query History

1

2

3

4

5

6

7

8

9

10

11

12

13

INSERT INTO triple (s, p, o) VALUES
(
'claim', 'isA', 'insurance_event'),
'payment', 'isA', 'transaction'),
'insurance_event', 'isA', 'financial_event'),
'transaction', 'isA', 'financial_event'),
'member', 'isA', 'person'),
'person', 'isA', 'entity'),
'loan', 'isA', 'financial_product'),
'insurance_product', 'isA', 'financial_product');

Data Output

Messages

Notifications

INSERT 0 8

Query returned successfully in 157 msec.

3: Recursive Query for Transitive

```

1  WITH RECURSIVE isa_cte AS (
2      -- Base facts
3      SELECT s, o, s || '->' || o AS path
4      FROM triple
5      WHERE p = 'isA'
6
7      UNION ALL
8
9      -- Recursive step: find transitive closure
10     SELECT cte.s, t.o, cte.path || '->' || t.o
11     FROM isa_cte cte
12     JOIN triple t ON cte.o = t.s
13     WHERE t.p = 'isA'
14 )
15 SELECT s, o, path
16 FROM isa_cte
17 ORDER BY s
18 LIMIT 10;

```

Data Output Messages Notifications

Showing rows: 1 to 10

	s character varying (64)	o character varying (64)	path text
1	claim	financial_event	claim->insurance_event->financial_eve...
2	claim	insurance_event	claim->insurance_event
3	insurance_event	financial_event	insurance_event->financial_event
4	insurance_product	financial_product	insurance_product->financial_product
5	loan	financial_product	loan->financial_product
6	member	entity	member->person->entity
7	member	person	member->person
8	payment	financial_event	payment->transaction->financial_event
9	payment	transaction	payment->transaction
10	person	entity	person->entity

4: Optional Cleanup (to stay under 10 committed rows)

Query

Query History

1

2

3

4

5

6

DELETE FROM triple

WHERE s IN ('loan','insurance_product');

Data Output

Messages

Notifications

DELETE 2

Query returned successfully in 291 msec.

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

1: Create business_limits Table

Query

Query History

1

2

3

4

5

6

7

8

9

10

11

12

13

CREATE TABLE business_limits (

rule_key VARCHAR(64) PRIMARY KEY,

threshold NUMERIC(12,2),

active CHAR(1) CHECK (active IN ('Y','N'))

);

-- Seed one active rule

INSERT INTO business_limits (rule_key, threshold, active)

VALUES ('max_payment_per_claim', 2000, 'Y');

Data Output

Messages

Notifications

INSERT 0 1

Query returned successfully in 158 msec.

2: Implement fn_should_alert Function

Query	Query History
1	CREATE OR REPLACE FUNCTION fn_should_alert(p_claimid INT, p_amount NUMERIC)
2	RETURNS INT AS \$\$
3	DECLARE
4	limit_val NUMERIC;
5	total_payment NUMERIC;
6	BEGIN
7	-- Read the active threshold
8	SELECT threshold INTO limit_val
9	FROM business_limits
10	WHERE rule_key = 'max_payment_per_claim' AND active = 'Y';
11	
12	-- Compute current total payments for this claim
13	SELECT COALESCE(SUM(paymentamount),0) INTO total_payment
14	FROM payment
15	WHERE claimid = p_claimid;
16	
17	-- Check if inserting this payment exceeds the threshold
18	IF (total_payment + p_amount) > limit_val THEN

Data Output	Messages	Notifications
CREATE FUNCTION		
Query returned successfully in 231 msec.		

3: BEFORE INSERT OR UPDATE Trigger on Payment

Query	Query History
1	CREATE OR REPLACE FUNCTION trg_fn_payment_alert()
2	RETURNS TRIGGER AS \$\$
3	BEGIN
4	IF fn_should_alert(NEW.claimid, NEW.paymentamount) = 1 THEN
5	RAISE EXCEPTION 'Payment exceeds business limit for claim %', NEW.claimid;
6	END IF ;
7	RETURN NEW;
8	END ;
9	\$\$ LANGUAGE plpgsql;
10	
11	CREATE TRIGGER payment_alert
12	BEFORE INSERT OR UPDATE ON payment
13	FOR EACH ROW
14	EXECUTE FUNCTION trg_fn_payment_alert();
15	

Data Output	Messages	Notifications
CREATE TRIGGER		
Query returned successfully in 125 msec.		

4: Demonstrate 2 Passing and 2 Failing DML Cases

```
5 -- Failing inserts (would exceed threshold)
6 DO $$
7 BEGIN
8     BEGIN
9         INSERT INTO payment (claimid, paymentamount, paymentdate) VALUES (1, 2000,
10        EXCEPTION WHEN OTHERS THEN
11            RAISE NOTICE 'Failed payment insert: %', SQLERRM;
12        END;
13
14     BEGIN
15         INSERT INTO payment (claimid, paymentamount, paymentdate) VALUES (2, 1500,
16        EXCEPTION WHEN OTHERS THEN
17            RAISE NOTICE 'Failed payment insert: %', SQLERRM;
18        END;
19 END $$;
```

Data Output Messages Notifications

ERROR: insert or update on table "payment" violates foreign key constraint "payment_claimid_fkey"
Key (claimid)=(1) is not present in table "claim".

SQL state: 23503

Detail: Key (claimid)=(1) is not present in table "claim".