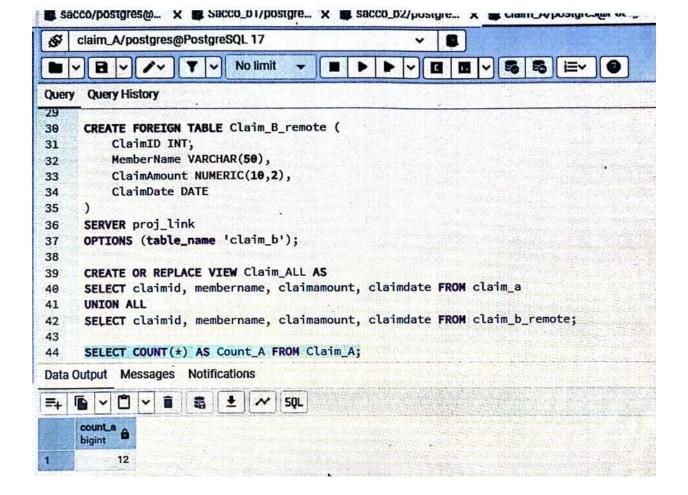
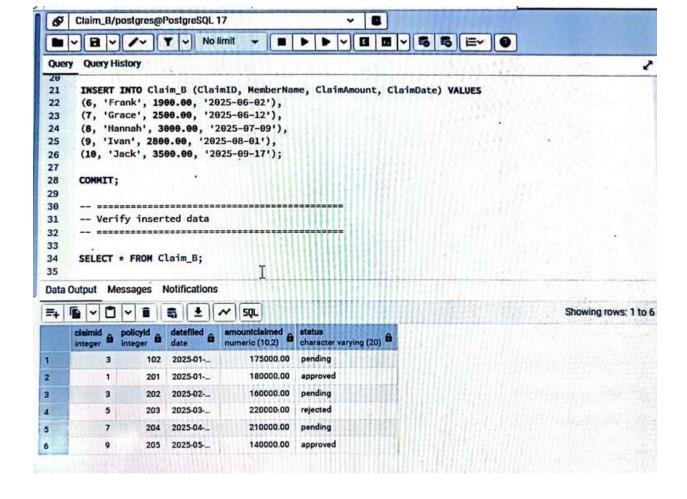
STUDENT 31: SACCO INSURANCE & MEMBER EXTENSION

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,
```



```
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;
```



```
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');
```

itput Messages Notifications

```
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.
CREATE FOREIGN TABLE Member_remote (
4
     memberid INT,
15
       membername VARCHAR(50),
16
       joindate DATE,
        address VARCHAR(100)
17
18
   SERVER proj_link
19
20
   OPTIONS (table_name 'member');
21
)ata Output Messages Notifications
CREATE FOREIGN TABLE
Query returned successfully in 828 msec.
8 CREATE TABLE Officer (
9
      OfficerID INT PRIMARY KEY,
       OfficerName VARCHAR(50),
10
11
       Department VARCHAR(50),
12
       MemberID INT
13 );
14
Data Output Messages Notifications
CREATE TABLE
```

Query returned successfully in 250 msec.

```
CREATE FOREIGN TABLE Officer_remote (
    officerid INT,
    officername VARCHAR(50),
    department VARCHAR(50),
    memberid INT

10

11

SERVER proj_link

OPTIONS (table_name 'officer');

Data Output Messages Notifications

CREATE FOREIGN TABLE

Query returned successfully in 124 msec.
```

3.

```
14 SELECT
15
      a.claimid,
       a.membername,
16
17
       a.claimamount,
     o.officername,
18
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

3. Capture execution plans with DBMS XPLAN and AUTOTRAC

- A4. Two-Phase Commit & Recovery (2 rows)
- 1. Clean run: insert one local and one remote row

```
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
           -- Insert into remote Payment@proj_link (this will NOT execute)
   1
           -- INSERT INTO Payment@proj_link (...) VALUES (...);
           COMMIT; -- will never reach here
       EXCEPTION
          WHEN OTHERS THEN
              DBMS_OUTPUT.PUT_LINE('Simulated failure, distributed transaction incomplete: ' || SQLERRM);
       END;
3. Query pending 2PC transactions
   2
        -- On Node_A
        SELECT LOCAL_TRAN_ID, STATE, START_TIME
   3
        FROM DBA 2PC PENDING;
   5
```

4. Repeat clean run to verify no pending transactions

```
1 -- Insert another local + remote row
 2 DECLARE
 3 v BEGIN
         INSERT INTO Claim_A (ClaimID, MemberName, ClaimAmount, ClaimDate)
 4
         VALUES (13, 'Nina', 3000, SYSDATE);
 5
 6
         INSERT INTO Payment@proj_link (PaymentID, ClaimID, Amount, PaymentDate)
 7
         VALUES (102, 13, 3000, SYSDATE);
 8
 9
10
        COMMIT;
11
12
         DBMS_OUTPUT.PUT_LINE('Clean distributed transaction committed successfully.');
13
    END;
14
     /
15
16
     -- Verify no pending transactions
     SELECT * FROM DBA_2PC_PENDING;
17
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
claimid a policyid a datefiled a amountclaimed a status numeric (10,2) a character varying (20)
```

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?
         SELECT s.sid, s.serial#, l.type, l.id1, l.id2, l.lmode, l.request, l.block
    3
     4 FROM v$lock l
         JOIN v$session s ON l.sid = s.sid
     5
         WHERE l.id1 IN (
     6
     7
              SELECT object id FROM dba objects WHERE object name = 'CLAIM A'
        );
    8
    9
         -- Optional: use DBA_BLOCKERS / DBA_WAITERS
    10
       SELECT * FROM dba_blockers;
    11
         SELECT * FROM dba_waiters;
    12
   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

```
CREATE TABLE Claim (
 1
 2
          ClaimID SERIAL PRIMARY KEY,
 3
          MemberID INT NOT NULL,
          ClaimAmount NUMERIC(10,2) NOT NULL CHECK (ClaimAmount > 0),
 4
 5
          ClaimDate DATE NOT NULL,
          Status VARCHAR(20) NOT NULL CHECK (Status IN ('Pending', 'Approved', 'Rejected')),
 6
 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,
         ClaimID INT NOT NULL,
11
         PaymentAmount NUMERIC(10,2) NOT NULL CHECK (PaymentAmount > 0),
13
         PaymentDate DATE NOT NULL,
         FOREIGN KEY (ClaimID) REFERENCES Claim(ClaimID)
14
    );
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 $$
2
     BEGIN
3 4
        BEGIN
 4
             INSERT INTO Claim (MemberID, ClaimAmount, ClaimDate, Status) VALUES
             (3, -100, '2025-10-20', 'Pending'); -- Negative amount
 5
        EXCEPTION WHEN OTHERS THEN
 6
 7
             RAISE NOTICE 'Claim failed: Negative amount';
        END;
 8
 9
10 🗸
         BEGIN
             INSERT INTO Claim (MemberID, ClaimAmount, ClaimDate, Status) VALUES
11
             (4, 2000, '2025-10-25', 'Unknown'); -- Invalid status
12
13
         EXCEPTION WHEN OTHERS THEN
14
             RAISE NOTICE 'Claim failed: Invalid status';
15
         END;
16
    END $$;
17
18
Data Output Messages Notifications
ERROR: unterminated dollar-quoted string at or near "$$;"
LINE 1: END SS:
  INSERT INTO Payment (ClaimID, PaymentAmount, PaymentDate) VALUES
  (1, 5000, '2025-10-05'),
  (2, 1200, '2025-10-16');
Output Messages Notifications
IR: insert or update on table "payment" violates foreign key constraint "payment_claimid_fkey"
(claimid)=(1) is not present in table "claim".
      DO $$
  2
 4 ~
            INSERT INTO Payment (ClaimID, PaymentAmount, PaymentDate) VALUES
  6
            (1, -500, '2025-10-06'); -- Negative payment
        EXCEPTION WHEN OTHERS THEN
  7
```

```
8
            RAISE NOTICE 'Payment failed: Negative amount';
       END;
9
10
        BEGIN
11 🗸
        INSERT INTO Payment (ClaimID, PaymentAmount, PaymentDate) VALUES
12
13:
            (2, 1500, '2025-10-10'); -- Amount exceeds claim
        EXCEPTION WHEN OTHERS THEN
14
15
            RAISE NOTICE 'Payment failed: Amount exceeds claim';
         END;
16
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
    CREATE TABLE claim_audit (
2
3
        bef_total NUMERIC(12,2),
         aft_total NUMERIC(12,2),
4
         changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
5
         key_col VARCHAR(64)
6
7
     );
8
9
10
Data Output Messages Notifications
CREATE TABLE
```

Query returned successfully in 4 secs 287 msec.

2: Implement Statement-Level AFTER Trigger on Payment

```
query query motory
 1
      ALTER TABLE claim ADD COLUMN total_payment NUMERIC(12,2) DEFAULT 0;
 2
 3
 4
  5
 Data Output Messages Notifications
 ALTER TABLE
 Query returned successfully in 4 secs 83 msec.
2. Create the trigger function:
    CREATE OR REPLACE FUNCTION update_claim_totals()
3
    RETURNS TRIGGER AS $$
4
   DECLARE
       before_total NUMERIC(12,2);
5
6
      after_total NUMERIC(12,2);
7 - BEGIN
В
       -- Compute total BEFORE change
9
      SELECT COALESCE(SUM(paymentamount),0) INTO before_total FROM payment;
9
     -- Recompute total payments per claim
1
  UPDATE claim c
SET total_payment = (
3
       SELECT COALESCE(SUM(p.paymentamount),0)
4
5
          FROM payment p
           WHERE p.claimid = c.claimid
8
   );
7
ata Output Messages Notifications
PDATE 0
```

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

uery returned successfully in 274 msec.

```
CREATE TRIGGER trg_update_claim_totals

AFTER INSERT OR UPDATE OR DELETE ON payment

FOR EACH STATEMENT

EXECUTE FUNCTION update_claim_totals();

8

9
```

3: Test with Small Mixed DML

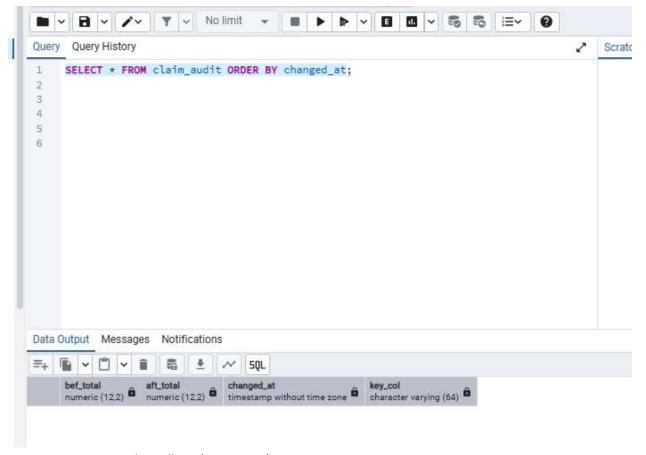
```
1
2 -- Passing INSERTs
    INSERT INTO payment (claimid, paymentamount, paymentdate) VALUES
4
    (1, 1000, '2025-10-10'),
5
    (2, 500, '2025-10-15');
6
7
     -- UPDATE affecting 1 row
8
     UPDATE payment SET paymentamount = 1200 WHERE paymentid = 1;
9
10
     -- DELETE affecting 1 row
11
 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



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

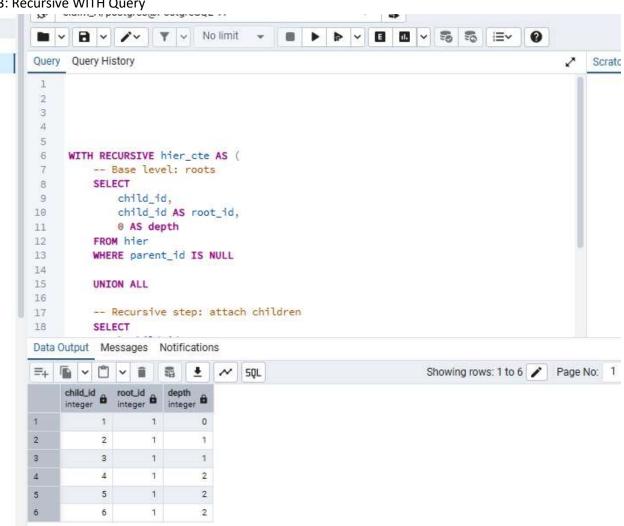
1: Create Hierarchy Table

```
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
                  -- level 2
8
     (1, 2),
9
     (1, 3),
10
     (2, 4),
                  -- level 3
11
     (2, 5),
     (3, 6); -- level 3
12
13
Data Output Messages Notifications
INSERT 0 6
Query returned successfully in 120 msec.
```

3: Recursive WITH Query



4: Join to Claim for Rollups

```
Query Query History
     WITH RECURSIVE hier_cte AS (
 2
         SELECT
 3
             child_id,
             child_id AS root_id,
 4
 5
             0 AS depth
 8
         FROM hier
 7
         WHERE parent_id IS NULL
 8
 9
         UNION ALL
10
11
         SELECT
12
             h.child_id.
             cte.root_id,
13
14
             cte.depth + 1
15
         FROM hier h
         JOIN hier_cte cte ON h.parent_id = cte.child_id
16
17
18 SELECT
Query Query History
11
         SELECT
12
            h.child_id,
13
            cte.root_id,
             cte.depth + 1
14
15
         FROM hier h
         JOIN hier_cte cte ON h.parent_id = cte.child_id
16
    )
17
18 SELECT
        cte.child_id,
19
         cte.root_id,
20
         cte.depth,
21
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
     GROUP BY cte.child_id, cte.root_id, cte.depth
25
     ORDER BY cte.root_id, cte.depth;
26
27
```

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

1: Create TRIPLE Table

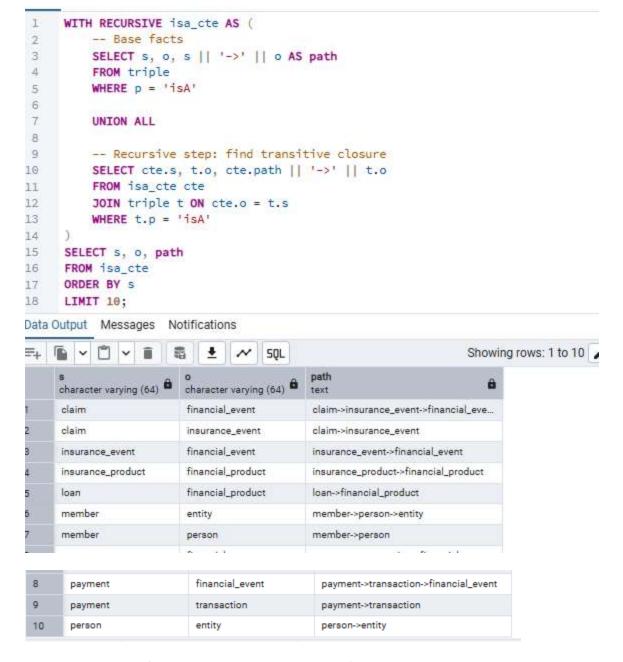
```
Query Query History
1
2
     CREATE TABLE triple (
3
         s VARCHAR(64),
4
        p VARCHAR(64),
5

    VARCHAR(64)

6
    );
7
8
9
Data Output Messages Notifications
CREATE TABLE
Query returned successfully in 166 msec.
```

2: Insert 8-10 Domain Facts

```
Query Query History
 1
     INSERT INTO triple (s, p, o) VALUES
 2
 3
     ('claim', 'isA', 'insurance_event'),
 4
     ('payment', 'isA', 'transaction'),
     ('insurance_event', 'isA', 'financial_event'),
 5
 6
   ('transaction', 'isA', 'financial_event'),
 7
     ('member', 'isA', 'person'),
     ('person', 'isA', 'entity'),
 8
     ('loan', 'isA', 'financial_product'),
9
10
     ('insurance_product', 'isA', 'financial_product');
11
12
13
Data Output Messages Notifications
INSERT 0 8
Query returned successfully in 157 msec.
```



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

```
Query Query History

1
2
3
4
DELETE FROM triple
5
WHERE s IN ('loan', 'insurance_product');
6

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
4
        rule_key VARCHAR(64) PRIMARY KEY,
5
        threshold NUMERIC(12,2),
        active CHAR(1) CHECK (active IN ('Y','N'))
6
7
    );
8
9
     -- Seed one active rule
     INSERT INTO business_limits (rule_key, threshold, active)
10
     VALUES ('max_payment_per_claim', 2000, 'Y');
11
12
13
Data Output Messages Notifications
INSERT 0 1
Query returned successfully in 158 msec.
```

```
Query Query History
                                                                                                 S
      CREATE OR REPLACE FUNCTION fn_should_alert(p_claimid INT, p_amount NUMERIC)
1
      RETURNS INT AS $$
 2
      DECLARE
         limit_val NUMERIC;
 4
         total_payment NUMERIC;
5
6 v BEGIN
7
          -- Read the active threshold
 8
        SELECT threshold INTO limit_val
9
        FROM business_limits
        WHERE rule_key = 'max_payment_per_claim' AND active = 'Y';
10
11
       -- Compute current total payments for this claim
SELECT COALESCE(SUM(paymentamount),0) INTO total_payment
FROM payment
12
13
14
        WHERE claimid = p_claimid;
15
16
          -- Check if inserting this payment exceeds the threshold
17
         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
                                                                                      2
     CREATE OR REPLACE FUNCTION trg_fn_payment_alert()
2 RETURNS TRIGGER AS $$
3 V BEGIN
4 +
         IF fn_should_alert(NEW.claimid, NEW.paymentamount) = 1 THEN
5
             RAISE EXCEPTION 'Payment exceeds business limit for claim %', NEW.claimid;
6
         END IF;
7
         RETURN NEW;
8 END;
9 $$ 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.
```

```
Query Query History
                                                                                 Z | |
  5 -- Failing inserts (would exceed threshold)
  6 DO $$
  7 BEGIN
         BEGIN
  8 🗸
             INSERT INTO payment (claimid, paymentamount, paymentdate) VALUES (1, 2000,
  9
 10
        EXCEPTION WHEN OTHERS THEN
             RAISE NOTICE 'Failed payment insert: %', SQLERRM;
 11
 12
         END;
 13
 14 🗸
        BEGIN
 15
              INSERT INTO payment (claimid, paymentamount, paymentdate) VALUES (2, 1500,
 16
        EXCEPTION WHEN OTHERS THEN
             RAISE NOTICE 'Failed payment insert: %', SQLERRM;
 17
        END;
 18
 19 END $$;
 20
 21
```

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".