

AFRICAN CENTRE OF EXCELLENCE IN DATA SCIENCE (ACE-DS)

ADVANCED DATABASE PROJECT-BASED EXAM

Module Code: DSM6235

Topic: DIGITAL HEALTH INSURANCE & CLAIMS

Student Name: KALIKUMUTIMA David

Stud.ID: 224020870

B6: Declarative Rules Hardening (≤ 10 committed rows)

WHAT TO DO

1. On tables Claim and Service, add/verify NOT NULL and domain CHECK constraints suitable for claim costs and approvals (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 ≤ 10 total.
3. Show clean error handling for failing cases.

EXPECTED OUTPUT

- ✓ ALTER TABLE statements for added constraints (named consistently).

--A) claim TABLE.

/* altering claim table for adding "approvaldate" column to track approved date*/

ALTER TABLE claim

ADD approvaldate DATE;

/* applying NOT NULL constraint*/

ALTER TABLE claim

ALTER COLUMN amountclaimed SET NOT NULL,

ALTER COLUMN status SET NOT NULL,

ALTER COLUMN datefiled SET NOT NULL;

/* constraint to ensure that claim amounts is positive */

ALTER TABLE claim

ADD CONSTRAINT check_claim_amount_positive

CHECK (amountclaimed > 0);

/* constraint to ensure that approval status to be valid. */

ALTER TABLE claim

ADD CONSTRAINT check_claim_approval_status

CHECK (status IN ('Pending', 'Approved', 'Rejected'));

/* constraint to ensure that approval date is not before claim date */

ALTER TABLE claim

ADD CONSTRAINT check_claim_date_order

CHECK (

approvaldate IS NULL

OR approvaldate >= datefiled

);

--B) service TABLE.

/*applying NOT NUL constraint on cost and servicedate columns */

ALTER TABLE service

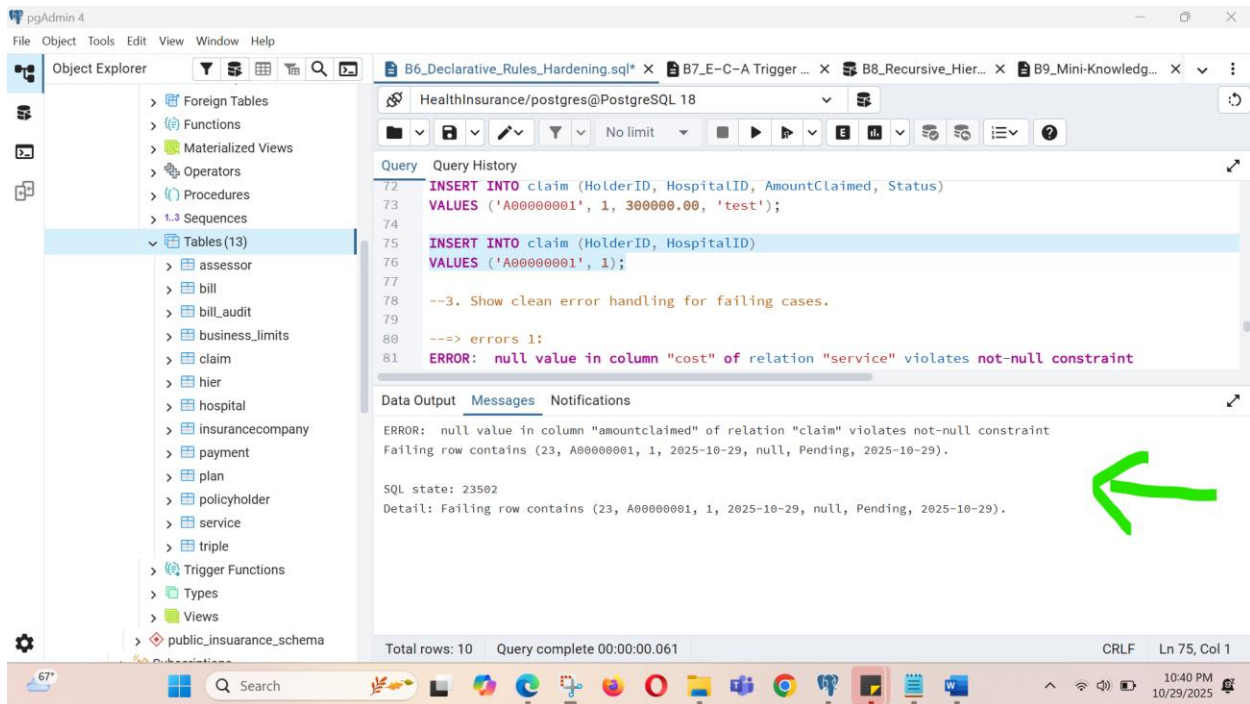
ALTER COLUMN cost SET NOT NULL,

ALTER COLUMN servicedate SET NOT NULL;

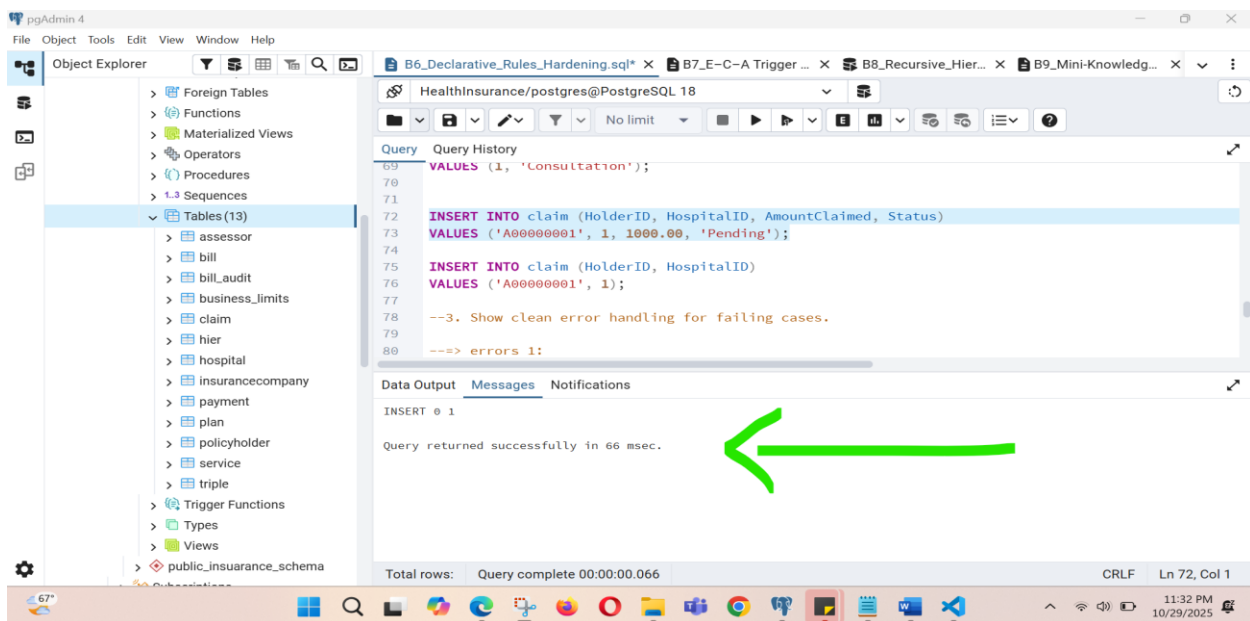
/*constraint for positive service cost*/

- ✓ Script with test INSERTs and captured ORA- errors for failing cases.

Screenshot for the errors for failing cases inset with NULL values



Screenshot for the successful inserted records without NULL values



Screenshot for the cost ≥ 0 constraint.

The screenshot shows the pgAdmin 4 interface. On the left, the 'Object Explorer' pane shows the 'public_insurance_schema' expanded, with 'Tables (13)' selected. The main query editor displays a SQL script with the following content:

```
64
65 SELECT * FROM Service;
66 SELECT * FROM policyholder;
67
68 INSERT INTO Service (claimid, description, cost)
69 VALUES (1, 'Consultation', -2000);
70
71 INSERT INTO Service(ClaimID, Description, Cost, ServiceDate) VALUES
72 (1, 'Consultation', -100.00, '2025-01-10');
73
74
75 INSERT INTO claim (HolderID, HospitalID, AmountClaimed, Status)
```

The 'Data Output' pane shows an error message:

```
ERROR: new row for relation "service" violates check constraint "check_service_cost_positive"
Failing row contains (52, 1, Consultation, -100.00, 2025-01-10).

SQL state: 23514
Detail: Failing row contains (52, 1, Consultation, -100.00, 2025-01-10).
```

A green arrow points to the error message. The status bar at the bottom indicates 'Total rows: 12' and 'Query complete 00:00:00.089'.

✓ SELECT proof that only the passing rows were committed; total committed rows ≤ 10

The screenshot shows the pgAdmin 4 interface. On the left, the 'Object Explorer' pane shows the 'public_insurance_schema' expanded, with 'Tables (13)' selected. The main query editor displays a SQL script with the following content:

```
65 SELECT * FROM Service;
66 SELECT * FROM policyholder;
67
68 INSERT INTO Service (claimid, description, cost)
69 VALUES (1, 'Consultation', -2000);
70
71 INSERT INTO Service(ClaimID, Description, Cost, ServiceDate) VALUES
72 (2, 'Consultation', 1100.00, '2025-01-10');
73
74
75 INSERT INTO claim (HolderID, HospitalID, AmountClaimed, Status)
76 VALUES ('AAAAAAA1', 1, 1000.00, 'Pending');
```

The 'Data Output' pane shows the result of the query:

```
INSERT 0 1

Query returned successfully in 89 msec.
```

A green arrow points to the 'Query returned successfully' message. The status bar at the bottom indicates 'Total rows: 1' and 'Query complete 00:00:00.089'.

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- > Foreign Tables
- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > Sequences
- > Tables (13)
 - > assessor
 - > bill
 - > bill_audit
 - > business_limits
 - > claim
 - > hier
 - > hospital
 - > insurancecompany
 - > payment
 - > plan
 - > policyholder
 - > service
 - > triple
- > Trigger Functions
- > Types
- > Views
- > public_insurance_schema

HealthInsurance/postgres@PostgreSQL 18

No limit

Query

```
65 SELECT * FROM Service;
66 SELECT * FROM policyholder;
67
68 INSERT INTO Service (claimid, description, cost)
69 VALUES (1, 'Consultation', -2000);
70
71 INSERT INTO Service(ClaimID, Description, Cost, ServiceDate) VALUES
72 (2, 'Consultation', 1100.00, '2025-01-10');
73
```


Data Output Messages Notifications

Showing rows: 1 to 13 Page No: 1 of 1

	serviceid [PK] integer	claimid integer	description text	cost numeric (12,2)	servicedate date
1		3	2 Blood Tests	300.00	2025-02-09
2		4	2 Medication	900.00	2025-02-09
3		5	3 Surgery	1500.00	2025-03-04
4		6	4 Physiothera...	150.00	2025-04-19
5		7	5 Consultation	100.00	2025-05-14
6		8	6 Inpatient st...	2000.00	2025-05-31
7		9	8 Maternity	800.00	2025-07-20

Total rows: 13 Query complete 00:00:00.114 CRLF Ln 65, Col 1

12:19 AM 10/30/2025



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

WHAT TO DO

1. Create an audit table Claim_AUDIT(bef_total NUMBER, aft_total NUMBER, changed_at TIMESTAMP, key_col VARCHAR2(64)).
2. Implement a statement-level AFTER INSERT/UPDATE/DELETE trigger on Service that recomputes denormalized totals in Claim once per statement.
3. Execute a small mixed DML script on CHILD affecting at most 4 rows in total; ensure net committed rows across the project remain ≤ 10 .
4. Log before/after totals to the audit table (2–3 audit rows).

EXPECTED OUTPUT

✓ CREATE TABLE Claim_AUDIT ... and CREATE TRIGGER source code.

```
/* Create an audit table Bill_AUDIT*/
CREATE TABLE bill_audit (
    bef_total NUMERIC(12,2),           -- to view total before change
    aft_total NUMERIC(12,2),           -- to view total after change
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- to view when the change happened
    key_col VARCHAR(64)                -- identifies which record was changed
);

/* Implement a statement-level AFTER INSERT/UPDATE/DELETE trigger on*/
-- creating table for recording total by summing all payments
CREATE TABLE bill (
    bill_id SERIAL PRIMARY KEY,
    total NUMERIC(12,2) DEFAULT 0
);

/* add bill_id to payment table as a foreign key */
ALTER TABLE Payment
    ADD bill_id SERIAL;
```

```

/*Create the Trigger Function for computing total bill*/
CREATE OR REPLACE FUNCTION compute_bill_totals()
RETURNS TRIGGER AS $$
BEGIN
    -- Recalculate totals for all affected bills
    UPDATE Bill b
    SET total = COALESCE((
        SELECT SUM(p.amount)
        FROM Payment p
        WHERE p.bill_id = b.bill_id
    ), 0)
    WHERE b.bill_id IN (
        SELECT DISTINCT bill_id FROM Payment
    );

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

```

✓ Mixed DML script and SELECT from totals showing correct precomputations.

The screenshot shows the pgAdmin 4 interface. The left pane displays the database structure, including tables like 'assessor', 'bill', 'claim', and 'payment'. The central pane shows a SQL script with the following content:

```

-- 3. Execute a small mixed DML script on CHILD affecting at most 4 rows in total; ensure net committed rows across the project remain sl
INSERT INTO Payment(ClaimID, AssessorID, Amount, PaymentDate, Method) VALUES
(9, 1, 1200.00, '2025-02-12 10:00:00+03', 'Bank Transfer'),
(8, 1, 1500.00, '2025-03-07 09:30:00+03', 'Bank Transfer');
SELECT * FROM Payment;

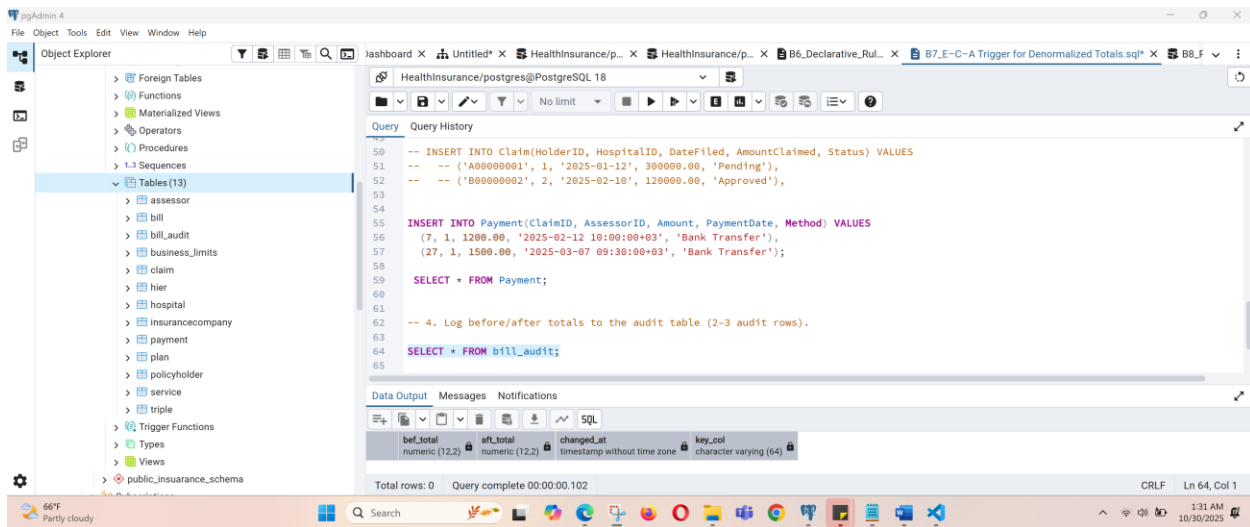
```

The bottom pane displays the results of the query, showing 7 rows of payment data. A green arrow points to the 'billId' column in the results table.

paymentid [PK] integer	claimid integer	assessorid integer	amount numeric (12,2)	paymentdate timestamp with time zone	method text	billId integer
1	1	2	1200.00	2025-02-12 09:00:00+02	Bank Transf...	1
2	2	3	1500.00	2025-03-07 08:30:00+02	Bank Transf...	2
4	4	8	800.00	2025-07-25 14:00:00+03	Cash	3
4	5	10	250.00	2025-09-05 11:00:00+03	Mobile Mon...	4
5	3	6	2000.00	2025-10-15 20:51:27.244668+...	Bank Transf...	5

Total rows: 7 Query complete 00:00:00.074

✓ `SELECT * FROM Claim_AUDIT` with 2–3 audit entries.



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

WHAT TO DO

1. Create table `HIER(parent_id, child_id)` for a natural hierarchy (domain-specific).
2. Insert 6–10 rows forming a 3-level hierarchy.
3. Write a recursive `WITH` query to produce `(child_id, root_id, depth)` and join to `Service` or its parent to compute rollups; return 6–10 rows total.
4. Reuse existing seed rows; do not exceed the ≤ 10 committed rows budget.

EXPECTED OUTPUT

✓ DDL + INSERTs for `HIER` (6–10 rows).

```
/* Create table HIER */
CREATE TABLE HIER (
  parent_id INT REFERENCES HIER(child_id),
  child_id INT,
  name     VARCHAR(50),
  PRIMARY KEY (child_id)
);
```



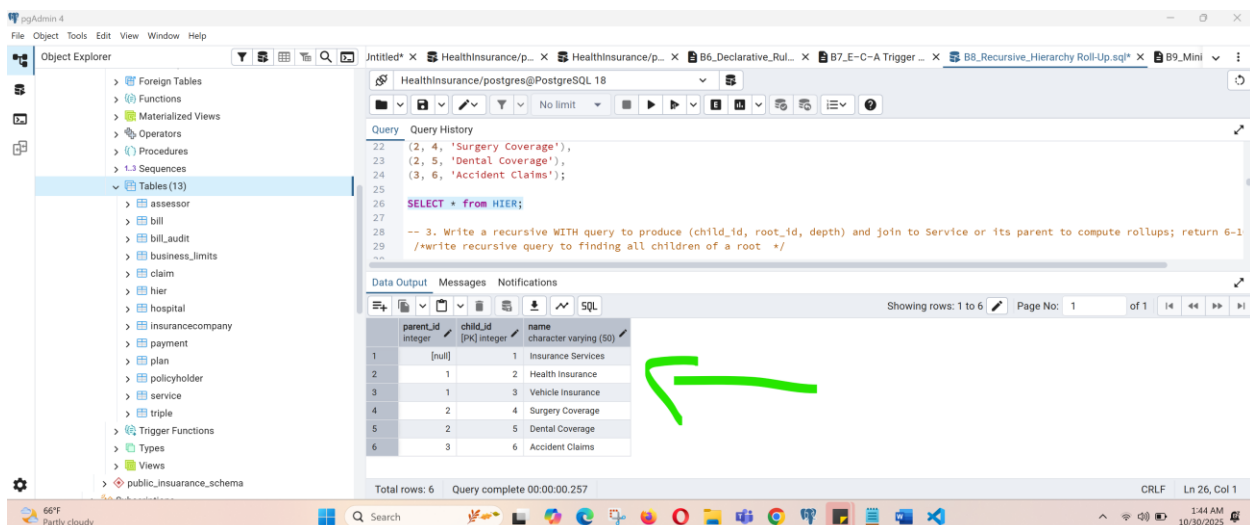
```

/* Insert 6-10 rows forming a 3-level hierarchy. */
INSERT INTO HIER (parent_id, child_id, name) VALUES
    (NULL, 1, 'Insurance Services'),
    (1, 2, 'Health Insurance'),
    (1, 3, 'Vehicle Insurance'),
    (2, 4, 'Surgery Coverage'),
    (2, 5, 'Dental Coverage'),
    (3, 6, 'Accident Claims');

SELECT * from HIER;

```

Screenshot for HIER table records



The screenshot shows the pgAdmin 4 interface. On the left, the 'Object Explorer' shows the 'Tables (13)' folder expanded, with 'hier' selected. The main pane shows the 'Query' editor with the following SQL:

```

-- 3. Write a recursive WITH query to produce (child_id, root_id, depth) and join to Service or its parent to compute rollups; return 6-1
/*write recursive query to finding all children of a root */
SELECT * from HIER;

```

Below the query editor, the 'Data Output' tab shows the results of the query. The table has 6 rows and 3 columns: parent_id, child_id, and name. A green arrow points to the table.

parent_id	child_id	name
[null]	1	Insurance Services
1	2	Health Insurance
1	3	Vehicle Insurance
2	4	Surgery Coverage
2	5	Dental Coverage
3	6	Accident Claims

Total rows: 6 Query complete 00:00:00.257

✓ Recursive WITH SQL and sample output rows (6-10).

```

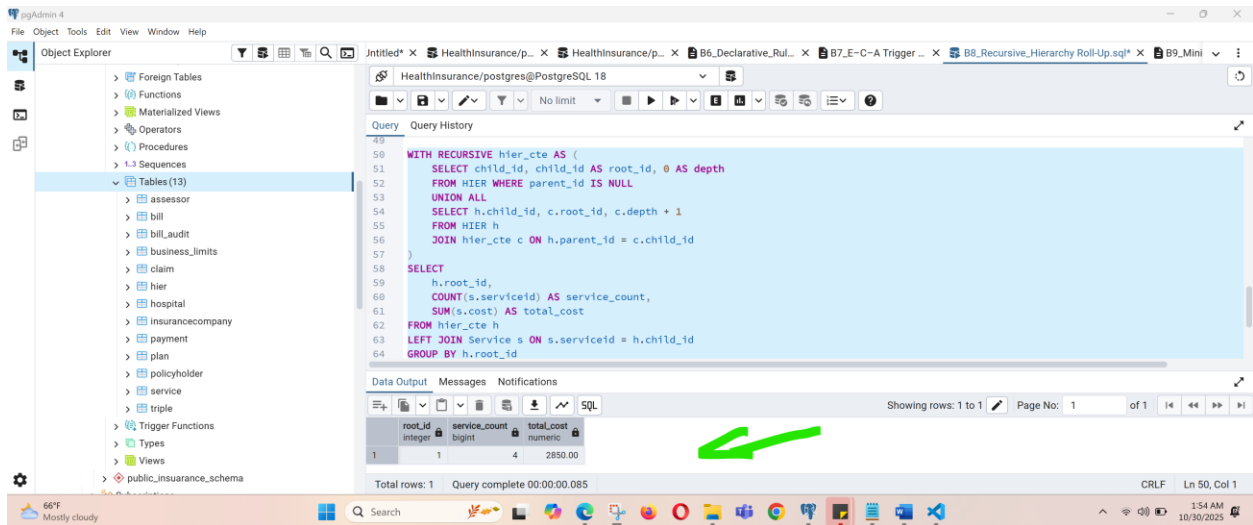
/*write recursive query to finding all children of a root */
WITH RECURSIVE tree AS (
    SELECT child_id, parent_id, name, 0 AS depth
    FROM HIER
    WHERE parent_id IS NULL -- start from root

    UNION ALL

    SELECT h.child_id, h.parent_id, h.name, t.depth + 1
    FROM HIER h
    JOIN tree t ON h.parent_id = t.child_id
);
SELECT * FROM HIER ORDER BY child_id;

```

- ✓ Control aggregation validating rollup correctness.



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

WHAT TO DO

1. Create table TRIPLE(s VARCHAR2(64), p VARCHAR2(64), o VARCHAR2(64)).
2. Insert 8–10 domain facts relevant to your project (e.g., simple type hierarchy or rule implications).
3. Write a recursive inference query implementing transitive isA*; apply labels to base records and return up to 10 labeled rows.
4. Ensure total committed rows across the project (including TRIPLE) remain ≤ 10 ; you may delete temporary rows after demo if needed.

EXPECTED OUTPUT

- ✓ DDL for TRIPLE and INSERT scripts for 8–10 facts.

```
/*create a TRIPLE table to stores information as a set of subject-predicate-object facts.*/  
CREATE TABLE TRIPLE (  
  s VARCHAR(64), -- Subject  
  p VARCHAR(64), -- Predicate  
  o VARCHAR(64) -- Object  
);  
.
```

```
/*Insert 8-10 domain facts relevant to this health insurance project. */
```

```
INSERT INTO TRIPLE (s, p, o) VALUES
    ('Service', 'isA', 'BusinessProcess'),
    ('ClaimService', 'isA', 'Service'),
    ('MedicalService', 'isA', 'Service'),
    ('DentalService', 'isA', 'MedicalService'),
    ('PaymentService', 'isA', 'FinancialService'),
    ('FinancialService', 'isA', 'Service'),
    ('Claim', 'involves', 'ClaimService'),
    ('Bill', 'requires', 'PaymentService'),
    ('ClaimService', 'supports', 'Patient'),
    ('MedicalService', 'supports', 'Patient');
```

```
SELECT * from TRIPLE;
```

Screenshot for inserted data into TRIPLE table

Query: ('MedicalService', 'supports', 'Patient');

SELECT * from TRIPLE;

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

s	p	o
Service	isA	BusinessProcess
ClaimService	isA	Service
MedicalService	isA	Service
DentalService	isA	MedicalService
PaymentService	isA	FinancialService
FinancialService	isA	Service
Claim	involves	ClaimService
Bill	requires	PaymentService
ClaimService	supports	Patient
MedicalService	supports	Patient

Total rows: 10 | Query complete 00:00:00.199

```
/*create a recursive inference query to find all transitive relationships of the isA predicated*/
```

```
WITH RECURSIVE isa_chain AS (
```

```
-- Base case: direct isA facts
```

```
SELECT s, o AS superclass
```

```
FROM TRIPLE
```

```
WHERE p = 'isA'
```

```
UNION
```

```
-- Recursive case: transitive closure
```

```
SELECT t.s, i.superclass
```

```
FROM TRIPLE t
```

```
JOIN isa_chain i ON t.o = i.s
```

```
WHERE t.p = 'isA'
```

```
)
```

--CONT...

```
SELECT DISTINCT s AS child, superclass, 'inferred isA*' AS label
FROM isa_chain
ORDER BY child, superclass
LIMIT 10;FROM isa_chain
ORDER BY child, superclass
LIMIT 10;
```

- ✓ Inference SELECT (with recursive part) and sample labeled output (≤ 10 rows).

Screenshot result of recursive inference query implemented.

The screenshot shows the pgAdmin 4 interface. The left pane displays the 'Object Explorer' with a tree view of the database schema, including tables like 'assessor', 'bill', 'business_limits', 'claim', 'hier', 'hospital', 'insurancecompany', 'payment', 'plan', 'policyholder', 'service', 'triple', 'trigger_functions', 'types', and 'views'. The main pane shows a SQL query window with the following query:

```
WITH RECURSIVE isa_chain AS (
  -- Base case: direct isA facts
  SELECT s, o AS superclass
  FROM TRIPLE
  WHERE p = 'isA'
  UNION
  -- Recursive case: transitive closure
  SELECT t.s, i.superclass
  FROM TRIPLE t
  JOIN isa_chain i ON t.o = i.s
)
```

The 'Data Output' pane shows the results of the query, displaying 10 rows. A green arrow points to the 'label' column, which contains the text 'inferred isA*'. The results are as follows:

child	superclass	label
ClaimService	BusinessProcess	inferred isA*
ClaimService	Service	inferred isA*
DentalService	BusinessProcess	inferred isA*
DentalService	MedicalService	inferred isA*
DentalService	Service	inferred isA*

The status bar at the bottom indicates 'Total rows: 10' and 'Query complete 00:00:00.215'.

/* query for Validating grouping and label consistency */

```
SELECT superclass, COUNT(*) AS num_children
```

```
FROM (
```

```
  WITH RECURSIVE isa_chain AS (
```

```
    SELECT s, o AS superclass FROM TRIPLE WHERE p='isA'
```

```
    UNION
```

```
    SELECT t.s, i.superclass FROM TRIPLE t JOIN isa_chain i ON t.o=i.s WHERE t.p='isA'
```

```
  )
```

```
  SELECT DISTINCT s, superclass FROM isa_chain
```

```
) grouped
```

```
GROUP BY superclass;
```

- ✓ Grouping counts proving inferred labels are consistent.

Screenshot showing Grouping counts proving inferred labels.

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer shows the database structure. The main pane displays a SQL query and its results. The query is as follows:

```
56 SELECT superclass, COUNT(*) AS num_children
57 FROM (
58   WITH RECURSIVE isa_chain AS (
59     SELECT s, o AS superclass FROM TRIPLE WHERE p='isa'
60     UNION
61     SELECT t.s, i.superclass FROM TRIPLE t JOIN isa_chain i ON t.o=i.s WHERE t.p='isa'
62   )
63   SELECT DISTINCT s, superclass FROM isa_chain
64 ) grouped
65 GROUP BY superclass;
```

The results table shows the following data:

superclass	num_children
BusinessProcess	6
FinancialService	1
MedicalService	1
Service	5

A green arrow points to the results table. The status bar at the bottom indicates "Total rows: 4" and "Query complete 00:00:00.089".

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

WHAT TO DO

1. Create BUSINESS_LIMITS(rule_key VARCHAR2(64), threshold NUMBER, active CHAR(1) CHECK(active IN('Y','N')))) and seed exactly one active rule.
2. Implement function fn_should_alert(...) that reads BUSINESS_LIMITS and inspects current data in Service or Claim to decide a violation (return 1/0).
3. Create a BEFORE INSERT OR UPDATE trigger on Service (or relevant table) that raises an application error when fn_should_alert returns 1.
4. Demonstrate 2 failing and 2 passing DML cases; rollback the failing ones so total committed rows remain within the ≤ 10 budget.

EXPECTED OUTPUT

- ✓ DDL for BUSINESS_LIMITS, function source, and trigger source.

```
/* A) Create the BUSINESS_LIMITS table */
CREATE TABLE BUSINESS_LIMITS (
    rule_key VARCHAR(64),
    threshold NUMERIC(12,2),
    active CHAR(1) CHECK (active IN ('Y', 'N')));

-- Insert one active rule
INSERT INTO BUSINESS_LIMITS VALUES ('MAX_SERVICE_COST', 20000, 'Y');
COMMIT;

/* B) Create the function fn_should_alert(...) to check if the new service record violated the
business rule.*/
CREATE OR REPLACE FUNCTION fn_should_alert(p_service_cost NUMERIC)
RETURNS INTEGER AS $$
DECLARE
    v_threshold NUMERIC;
BEGIN
    -- Read the active rule threshold
    SELECT threshold
    INTO v_threshold
    FROM business_limits
    WHERE active = 'Y' AND rule_key = 'MAX_SERVICE_COST';

    -- Compare with provided cost
    IF p_service_cost > v_threshold THEN
        RETURN 1; -- violation
    ELSE
        RETURN 0; -- ok
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 0; -- no rule active, no alert
END;
$$ LANGUAGE plpgsql;
```

```

/* Create a trigger on Service to invokes the function before inserting/updating a record.*/
CREATE OR REPLACE FUNCTION trg_service_cost_limit_func()
RETURNS TRIGGER AS $$
DECLARE
    v_alert INTEGER;
BEGIN
    -- Call validation function
    v_alert := fn_should_alert(NEW.cost);

    -- If alert triggered, raise exception
    IF v_alert = 1 THEN
        RAISE EXCEPTION 'Service cost exceeds business threshold!';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

/* Attach trigger to Service table*/
CREATE TRIGGER trg_service_cost_limit
BEFORE INSERT OR UPDATE
ON service
FOR EACH ROW
EXECUTE FUNCTION trg_service_cost_limit_func();

/*insert record in service table to check*/

INSERT INTO Service(ClaimID, Description, Cost, ServiceDate) VALUES
(3, 'Consultation', 100000.00, '2025-01-10');

```

- ✓ Execution proof: two failed DML attempts (ORA- error) and two successful DMLs that commit.

Screenshot showing the triggered error when ever record with cost that is above business limit. (20000)

The screenshot shows the pgAdmin 4 interface. The left pane displays the 'Object Explorer' with the 'public_insurance_schema' expanded. The right pane shows a SQL query window with the following code:

```
-- 5. Attach trigger to Service table
CREATE TRIGGER trg_service_cost_limit
BEFORE INSERT OR UPDATE
ON service
FOR EACH ROW
EXECUTE FUNCTION trg_service_cost_limit_func();
--insert record in service table to check
INSERT INTO Service(ClaimID, Description, Cost, ServiceDate) VALUES
(3, 'Consultation', 100000.00, '2025-01-10');
```

The 'Data Output' pane shows an error message:

```
ERROR: Service cost exceeds business threshold!
CONTEXT: PL/SQL function trg_service_cost_limit_func() line 10 at RAISE
SQL state: P0001
```

A green arrow points to the error message.

- ✓ SELECT showing resulting committed data consistent with the rule; row budget respected.

Screenshot showing the respecting business cost limit (20000)

The screenshot shows the pgAdmin 4 interface. The left pane displays the 'Object Explorer' with the 'public_insurance_schema' expanded. The right pane shows a SQL query window with the following code:

```
INSERT INTO Service(ClaimID, Description, Cost, ServiceDate) VALUES
(3, 'Consultation', 100000.00, '2025-01-10');
SELECT * FROM service;
```

The 'Data Output' pane shows a table of results:

serviceid	claimid	description	cost	servicedate
1	3	Blood Tests	300.00	2025-02-09
2	4	Medication	900.00	2025-02-09
3	5	Surgery	1500.00	2025-03-04
4	6	Physiothera...	150.00	2025-04-19
5	7	Consultation	100.00	2025-05-14
6	8	Inpatient st...	2000.00	2025-05-31
7	9	Maternity	800.00	2025-07-20
8	10	Outpatient	250.00	2025-09-01

A green arrow points to the table of results.