

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 truck 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
    );
```

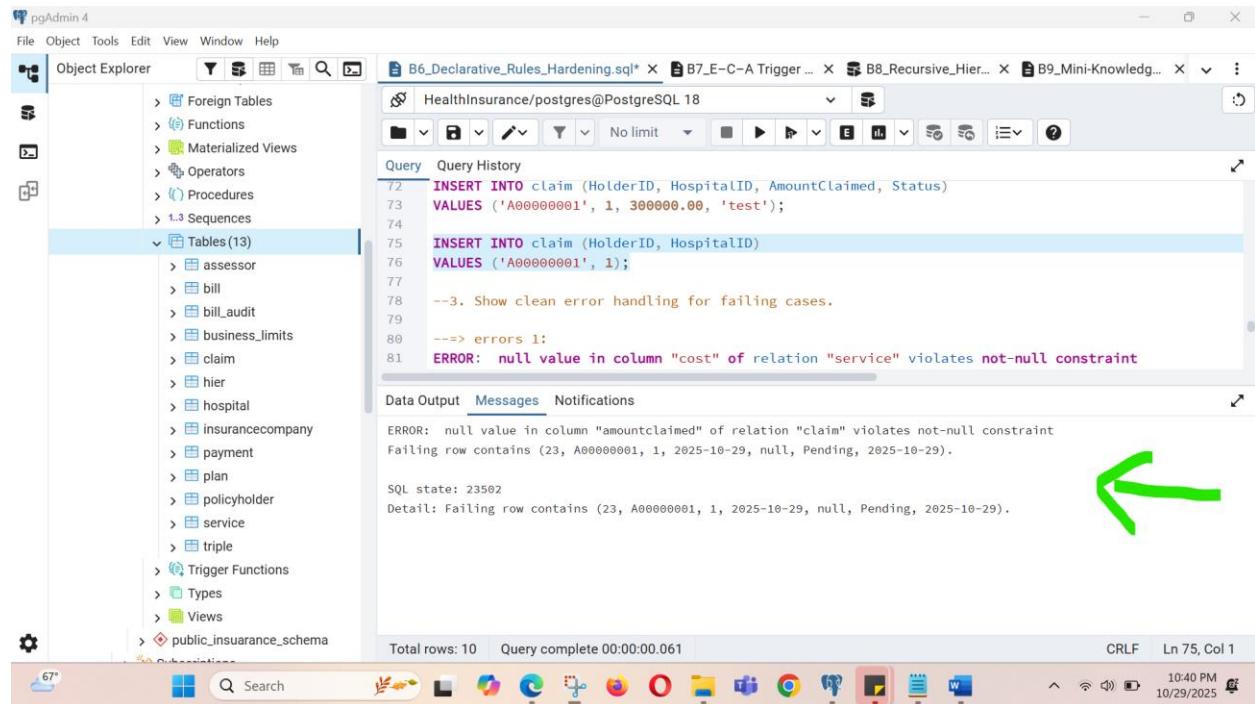
--B) service TABLE.

```
/*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*/
ALTER TABLE service
```

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

Screenshot for the errors for failing cases inset with NULL values



The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** Shows a tree view of database objects including Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables (13). The Tables node is expanded.
- Query Editor:** Contains the following SQL code:


```

    INSERT INTO claim (HolderID, HospitalID, AmountClaimed, Status)
    VALUES ('A00000001', 1, 30000.00, 'test');

    INSERT INTO claim (HolderID, HospitalID)
    VALUES ('A00000001', 1);

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

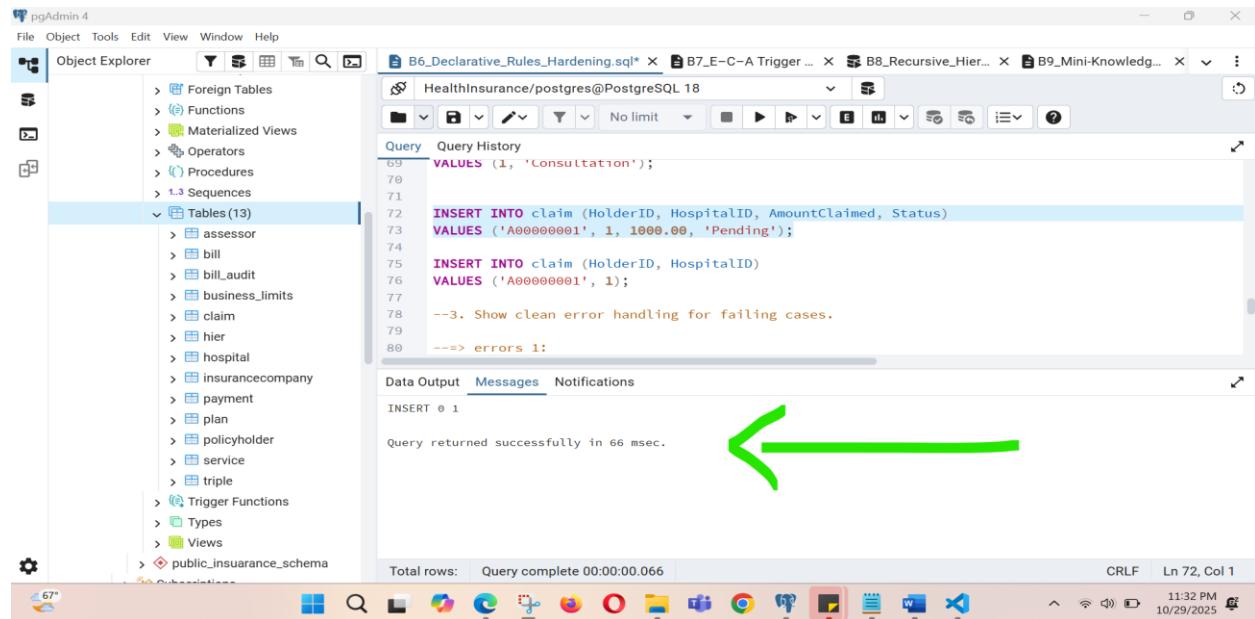
    --> errors 1:
    ERROR: null value in column "cost" of relation "service" violates not-null constraint
  
```
- Data Output:** Displays the error message:

ERROR: null value in column "amountclaimed" of relation "claim" violates not-null constraint
 Failing row contains (23, A00000001, 1, 2025-10-29, null, Pending, 2025-10-29).
- Messages:** Displays the error message:

SQL state: 23502
 Detail: Failing row contains (23, A00000001, 1, 2025-10-29, null, Pending, 2025-10-29).
- System Bar:** Shows the status bar with "Total rows: 10" and "Query complete 00:00:00.061".
- Bottom Icons:** Includes standard Windows taskbar icons for search, file explorer, and browser.

A green arrow points from the error message in the Messages tab to the error message in the Data Output tab.

Screenshot for the successful inserted records without NULL values



The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** Shows a tree view of database objects including Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables (13). The Tables node is expanded.
- Query Editor:** Contains the following SQL code:


```

    VALUES (1, 'Consultation');

    INSERT INTO claim (HolderID, HospitalID, AmountClaimed, Status)
    VALUES ('A00000001', 1, 1000.00, 'Pending');

    INSERT INTO claim (HolderID, HospitalID)
    VALUES ('A00000001', 1);

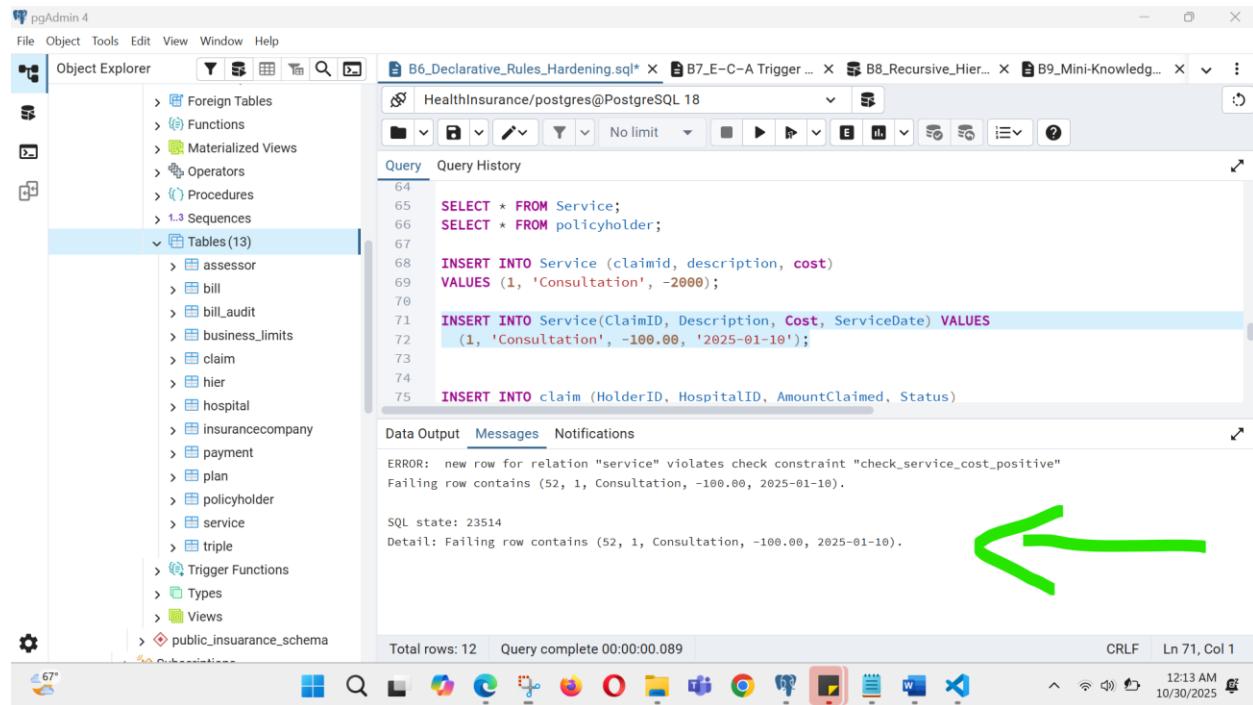
    --3. Show clean error handling for failing cases.

    --> errors 1:
  
```
- Data Output:** Displays the execution results:

INSERT 0 1
 Query returned successfully in 66 msec.
- System Bar:** Shows the status bar with "Total rows: 0" and "Query complete 00:00:00.066".
- Bottom Icons:** Includes standard Windows taskbar icons for search, file explorer, and browser.

A green arrow points from the "Query returned successfully in 66 msec." message in the Data Output tab to the same message in the System Bar.

Screenshot for the cost ≥ 0 constraint.



The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** Shows the schema structure, including a folder for `Tables (13)` which contains `assessor`, `bill`, `bill_audit`, `business_limits`, `claim`, `hier`, `hospital`, `insurancecompany`, `payment`, `plan`, `policyholder`, `service`, `triple`, `Trigger Functions`, `Types`, and `Views`.
- Query Editor:** Contains the following SQL code:

```

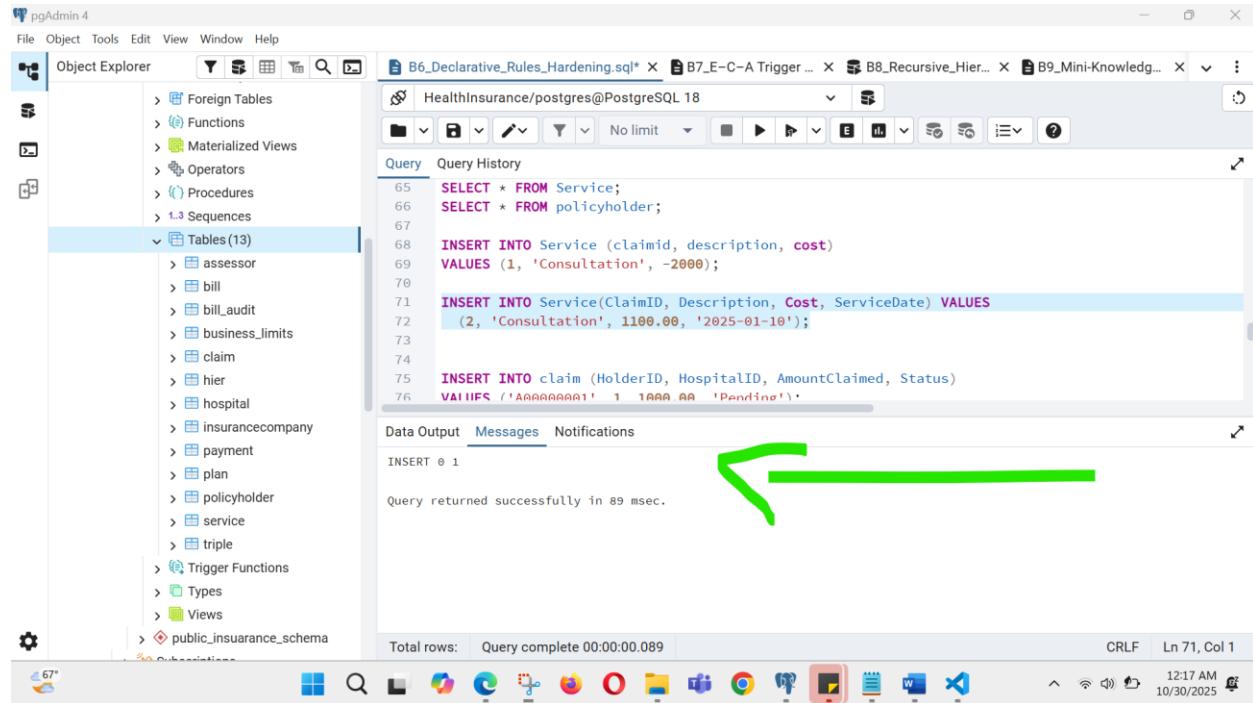
64
65   SELECT * FROM Service;
66   SELECT * FROM policyholder;
67
68   INSERT INTO Service (claimid, description, cost)
69     VALUES (1, 'Consultation', -2000);
70
71   INSERT INTO Service(ClaimID, Description, Cost, ServiceDate) VALUES
72     (1, 'Consultation', -100.00, '2025-01-10');
73
74
75   INSERT INTO claim (HolderID, HospitalID, AmountClaimed, Status)
    
```
- Data Output:** Displays 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).
    
```
- System Bar:** Shows the taskbar with various application icons and system status.

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



The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** Shows the schema structure, including a folder for `Tables (13)` which contains `assessor`, `bill`, `bill_audit`, `business_limits`, `claim`, `hier`, `hospital`, `insurancecompany`, `payment`, `plan`, `policyholder`, `service`, `triple`, `Trigger Functions`, `Types`, and `Views`.
- Query Editor:** Contains the following SQL code:

```

65
66   SELECT * FROM Service;
67   SELECT * FROM policyholder;
68
69   INSERT INTO Service (claimid, description, cost)
70     VALUES (1, 'Consultation', -2000);
71
72   INSERT INTO Service(ClaimID, Description, Cost, ServiceDate) VALUES
73     (2, 'Consultation', 1100.00, '2025-01-10');
74
75   INSERT INTO claim (HolderID, HospitalID, AmountClaimed, Status)
    
```
- Data Output:** Displays the result of the last query:

```

Data Output Messages Notifications
INSERT 0 1

Query returned successfully in 89 msec.
    
```
- System Bar:** Shows the taskbar with various application icons and system status.

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

B6_Declarative_Rules_Hardening.sql* B7_E-C-A Trigger ... B8_Recursive_Hier... B9_Mini-Knowledg...

HealthInsurance/postgres@PostgreSQL 18

No limit

Query History

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

Data Output Messages Notifications

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

servic eid [PK] integer	claim id integer	description text	cost numeric (12,2)	servicedate date
1	3	2	300.00	2025-02-09
2	4	2	900.00	2025-02-09
3	5	3	1500.00	2025-03-04
4	6	4	150.00	2025-04-19
5	7	5	100.00	2025-05-14
6	8	6	2000.00	2025-05-31
7	9	8	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 viewwhen 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 $$

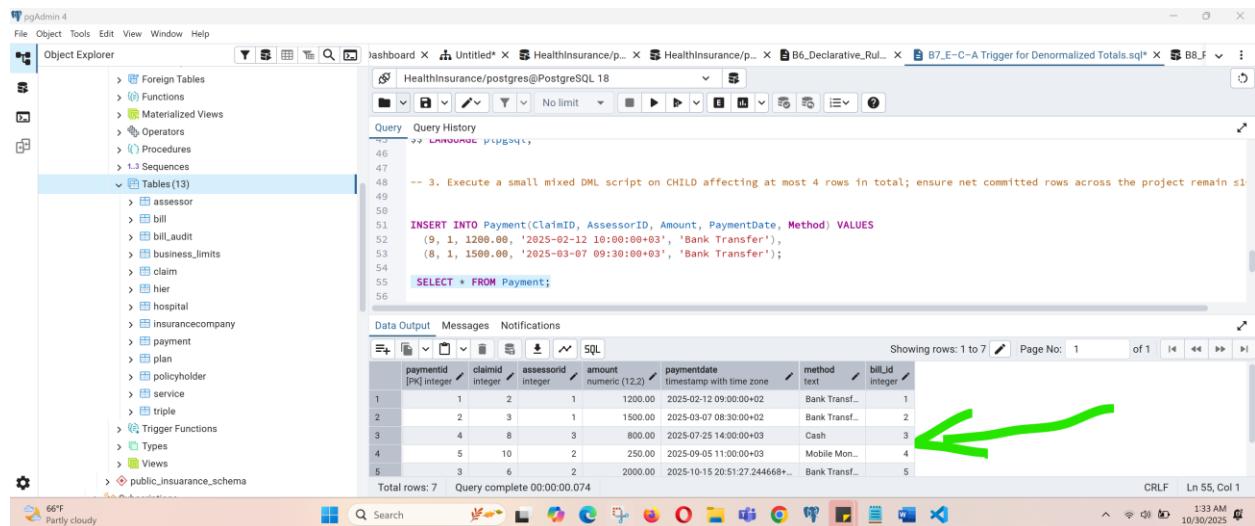
BEGIN
    -- Recalculate totals for all affected bills
    UPDATE Bill b
    SET total = COALESCE(
        SELECT SUM(p.amount)
        FROM Payment p
        WHERE p.bill_id = b.bill_id
    ), o)
    WHERE b.bill_id IN (
        SELECT DISTINCT bill_id FROM Payment
    );

    RETURN NULL;
END;

$$ LANGUAGE plpgsql;

```

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



The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** Shows the schema structure with 13 tables listed under "Tables".
- Query Editor:** Contains the following SQL code:


```
-- 3. Execute a small mixed DML script on CHILD affecting at most 4 rows in total; ensure net committed rows across the project remain stable
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;
```
- Data Output:** Displays the results of the query in a table format. A green arrow points to the last row of the table, which is highlighted in yellow.

paymentid	claimid	assessorid	amount	paymentdate	method	bill_id
1	1	2	1200.00	2025-02-12 09:00:00+02	Bank Trans...	1
2	2	3	1500.00	2025-03-07 08:30:00+02	Bank Trans...	2
3	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+03	Bank Trans...	5

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

```

File Object Tools Edit View Window Help
Object Explorer
Dashboard X Untitled X HealthInsurance/p... X HealthInsurance/p... X B6_Declarative_Rul... X B7.E-C-A Trigger for Denormalized Totals.sql* X B8.F
Query Query History
50 -- INSERT INTO Claim(HolderID, HospitalID, DateFiled, AmountClaimed, Status) VALUES
51 --   ('A00000001', 1, '2025-01-12', 300000.00, 'Pending'),
52 --   ('B00000002', 2, '2025-02-10', 120000.00, 'Approved'),
53
54
55 INSERT INTO Payment(ClaimID, AssessorID, Amount, PaymentDate, Method) VALUES
56   (7, 1, 1200.00, '2025-02-12 10:00:00+03', 'Bank Transfer'),
57   (27, 1, 1500.00, '2025-03-07 09:30:00+03', 'Bank Transfer');
58
59 SELECT * FROM Payment;
60
61
62 -- 4. Log before/after totals to the audit table (2-3 audit rows).
63
64 SELECT * FROM bill_audit;
65

```

Total rows: 0 Query complete 00:00:00.102 CRLF Ln 64, Col 1

66°F Partly cloudy Search Home Taskbar icons

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 with the following details:

- Object Explorer:** Shows the database schema with various objects like Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables.
- Query Editor:** Displays the SQL code used to insert data into the HIER table.
- Data Output:** Shows the results of the query `SELECT * from HIER;` which returns 6 rows of data.

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

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

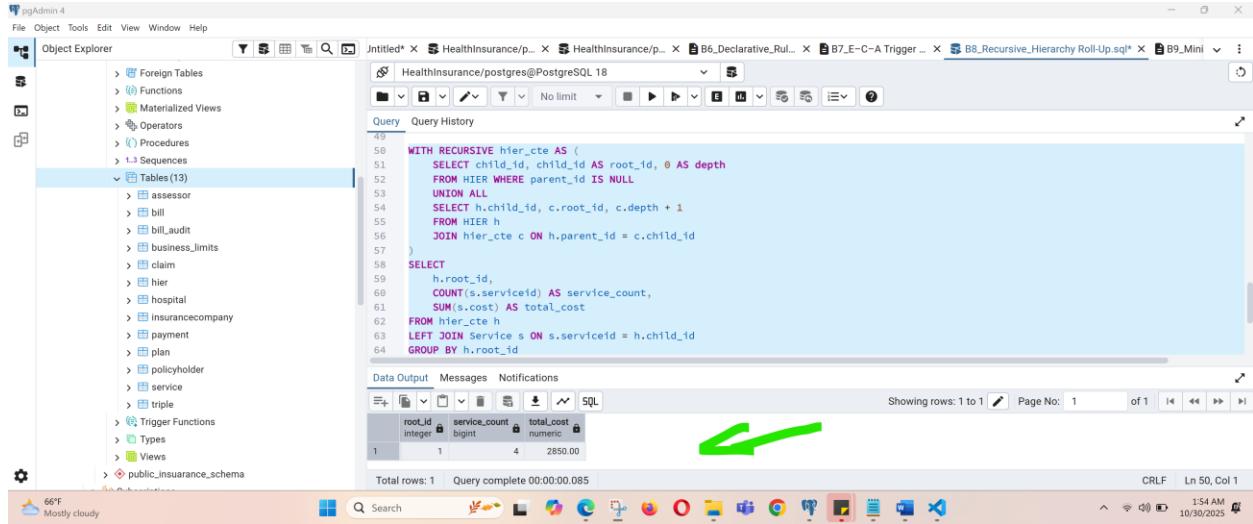
```

/*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.



```

49 WITH RECURSIVE hier_cte AS (
50   SELECT child_id, child_id AS root_id, 0 AS depth
51   FROM HIER WHERE parent_id IS NULL
52   UNION ALL
53   SELECT h.child_id, c.root_id, c.depth + 1
54   FROM HIER h
55   JOIN hier_cte c ON h.parent_id = c.child_id
56 )
57 )
58 SELECT
59   h.root_id,
60   COUNT(s.serviceid) AS service_count,
61   SUM(s.cost) AS total_cost
62 FROM hier_cte h
63 LEFT JOIN Service s ON s.serviceid = h.child_id
64 GROUP BY h.root_id

```

root_id	service_count	total_cost
1	4	2850.00

Total rows: 1 Query complete 00:00:00.085

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

The screenshot shows the pgAdmin 4 interface. On the left is the Object Explorer, which lists various database objects like Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables (13). The 'Tables' node is expanded, showing tables such as assessor, bill, bill_audit, business_limits, claim, hier, hospital, Insurancecompany, payment, plan, policyholder, service, and triple. A green arrow points from the 'Tables' node towards the query results grid. The main area contains a query editor with the following SQL:

```
24 ('MedicalService', 'supports', 'Patient');  
25  
26 SELECT * from TRIPLE;  
27  
28 -- 3. Write a recursive inference query implementing transitive isA*; apply labels to base records and return up to 10 labeled rows.  
29
```

The results grid shows the following data:

#	s	p	o
1	Service	isA	BusinessProcess
2	ClaimService	isA	Service
3	MedicalService	isA	Service
4	DentalService	isA	MedicalService
5	PaymentService	isA	FinancialService
6	FinancialService	isA	Service
7	Claim	involves	ClaimService
8	Bill	requires	PaymentService
9	ClaimService	supports	Patient
10	MedicalService	supports	Patient

Total rows: 10 Query complete 00:00:00.199 CRLF Ln 26, Col 1

```
/*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 sidebar displays the Object Explorer with various database objects like Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables (13). The Tables section is currently selected. The main pane shows a SQL query editor with the following code:

```
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
)
```

Below the query, the Data Output tab is active, showing a table with the following data:

	child	superclass	label
1	ClaimService	BusinessProcess	inferred isA*
2	ClaimService	Service	inferred isA*
3	DentalService	BusinessProcess	inferred isA*
4	DentalService	MedicalService	inferred isA*
5	DentalService	Service	inferred isA*

A green highlighter has been used to draw attention to the first two rows of the table.

```
/* 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.

```

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;

```

superclass	num_children
BusinessProcess	6
FinancialService	1
MedicalService	1
Service	5

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*/

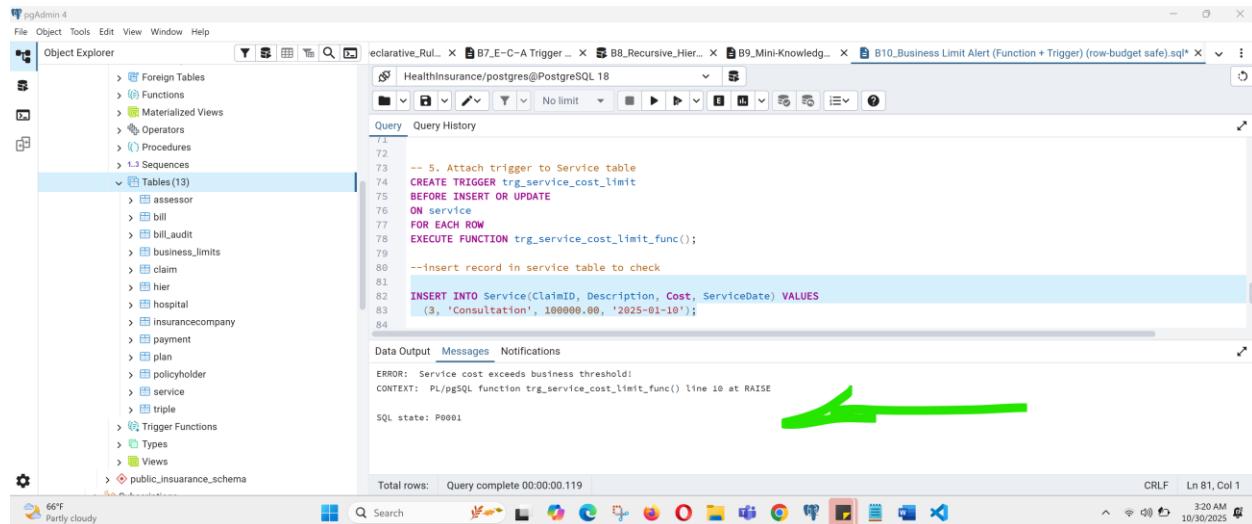
```

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 arrow when ever record with cost that is above business limit. (20000)



The screenshot shows the pgAdmin 4 interface with a query editor window. The query being run is:

```

-- 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');

```

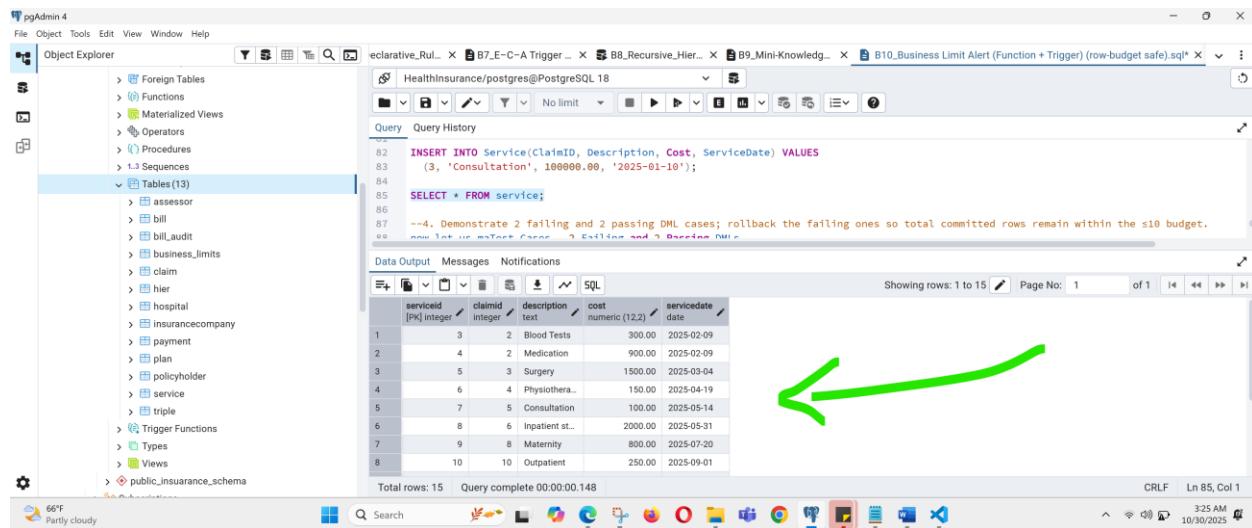
The output pane shows an error message:

ERROR: Service cost exceeds business threshold
CONTEXT: PL/pgSQL function `trg_service_cost_limit_func()` line 10 at RAISE

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 with a query editor window. The query being run is:

```

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

SELECT * FROM service;

```

The output pane shows the resulting data table:

	servicid [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
8	10	10	Outpatient	250.00	2025-09-01

A green arrow points to the data table.