***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

ADD CONSTRAINT check\_service\_cost\_positive

CHECK (cost > 0);

/\*constraint for service date is reasonable (not in far future, optional)\*/

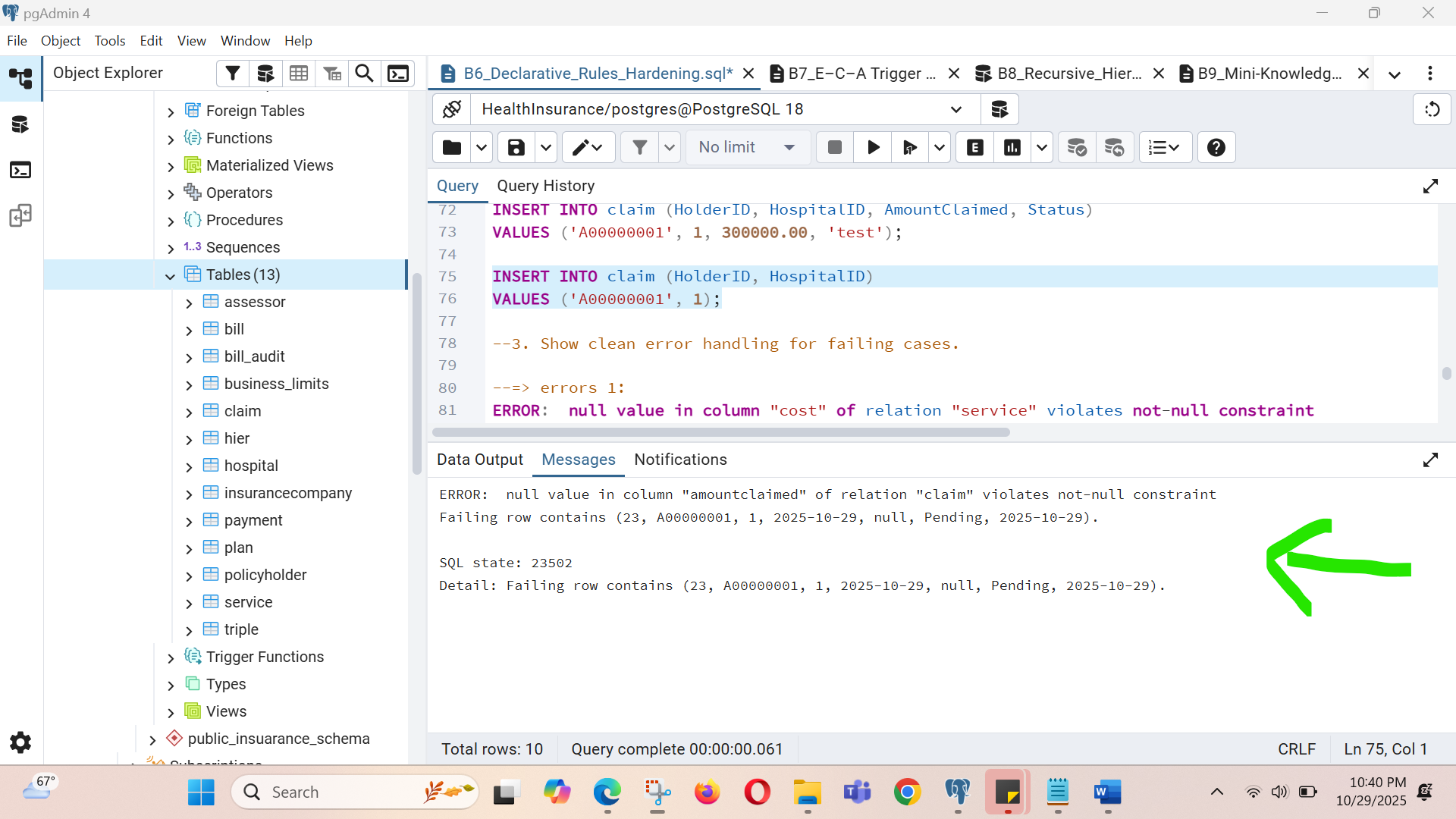
ALTER TABLE Service

ADD CONSTRAINT check\_service\_date\_valid

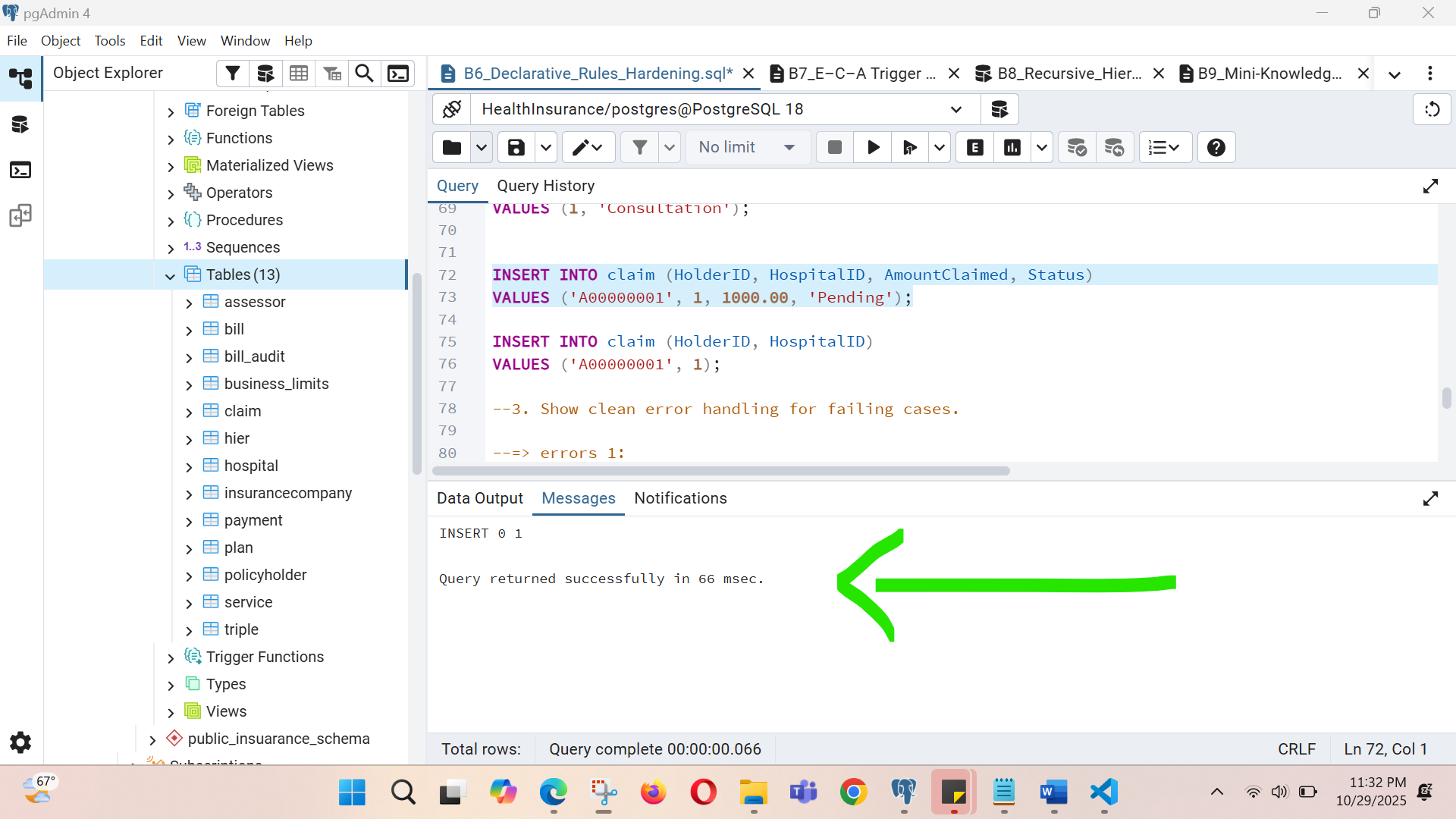
CHECK (servicedate <= CURRENT\_DATE + INTERVAL '30 days');

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

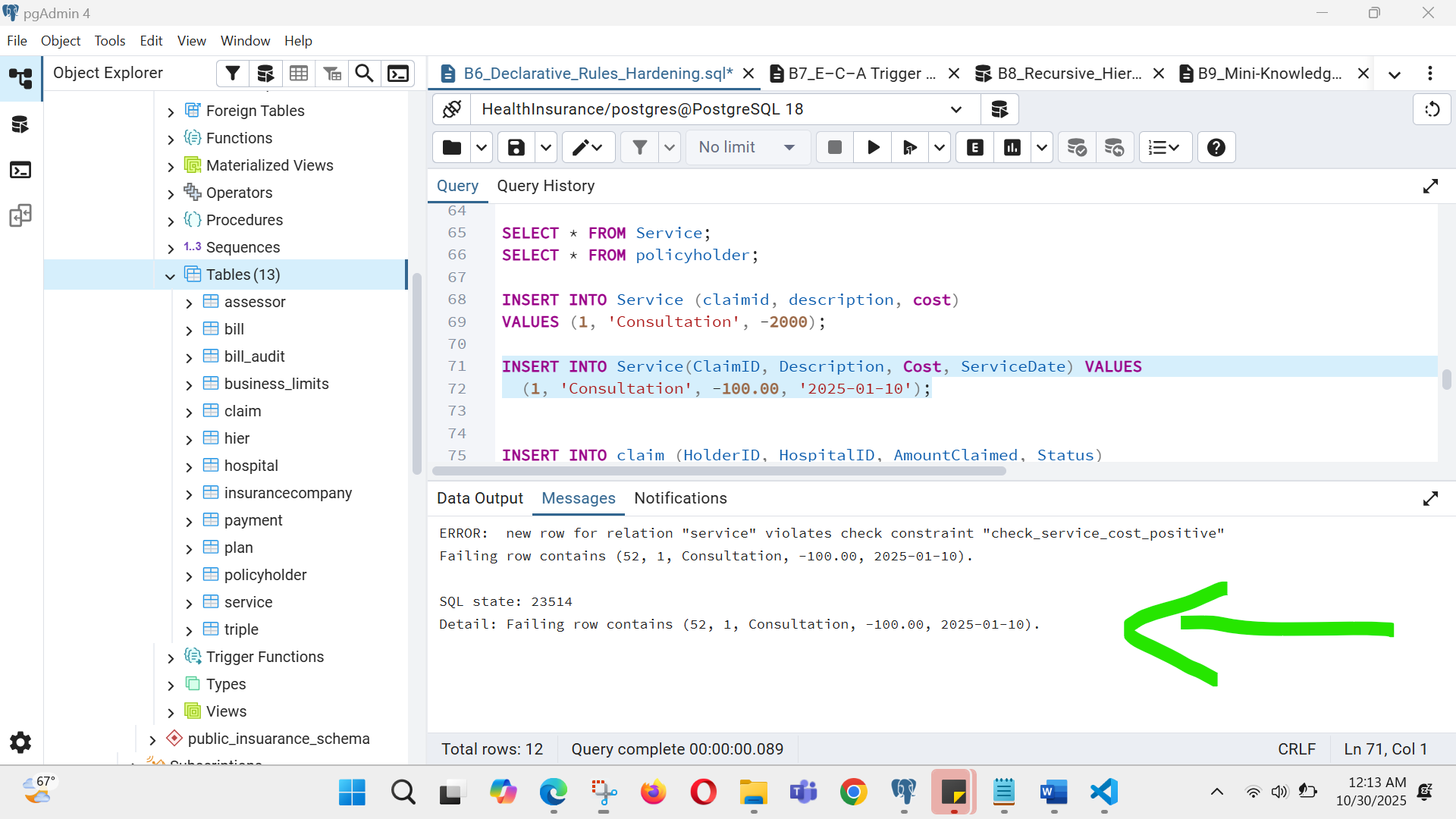
Screenshot for the errors for failing cases inset with NULL values



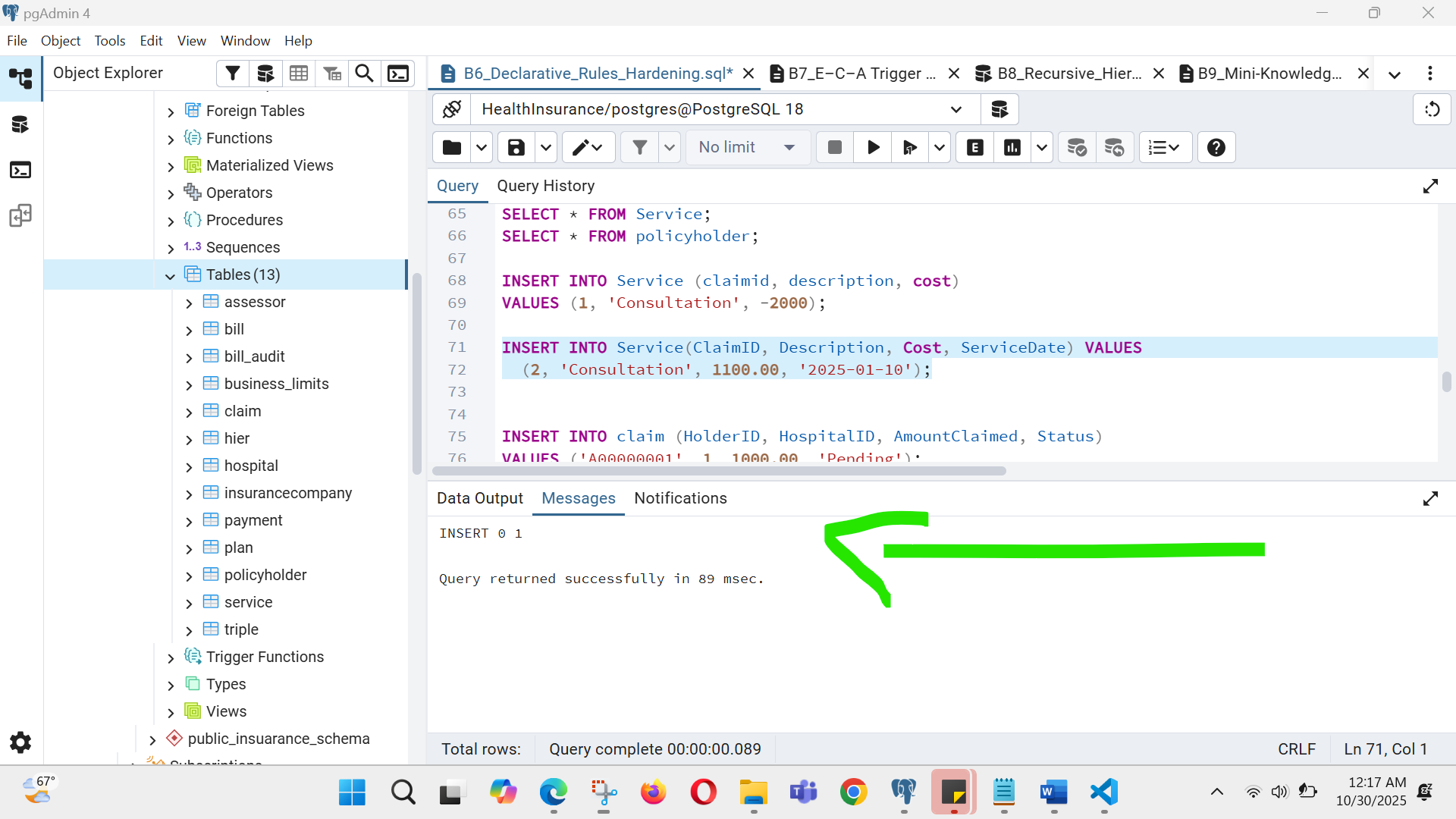
Screenshot for the successful inserted records without NULL values

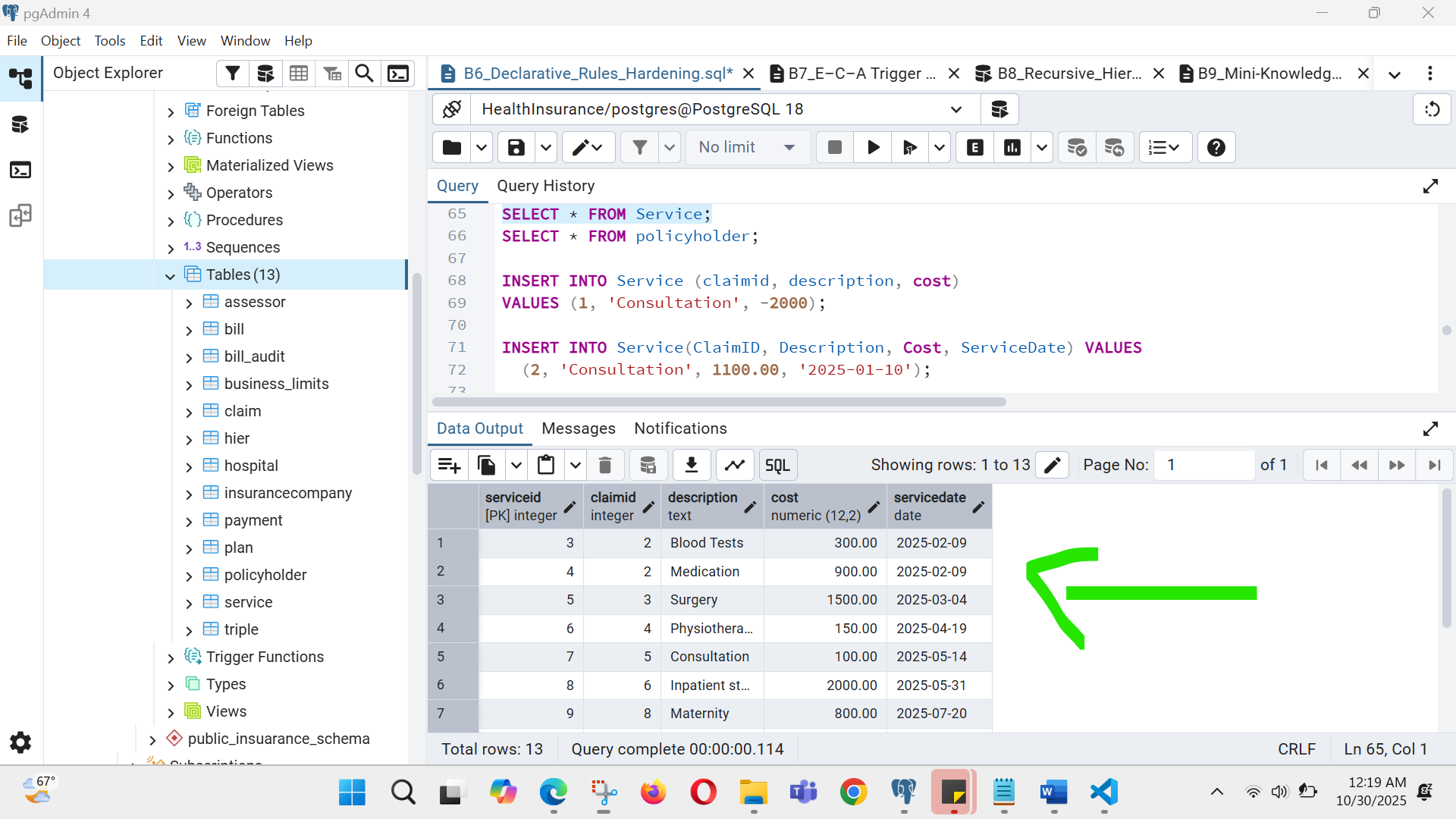


Screenshot for the cost > = 0 constraint.



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





**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 tatal 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

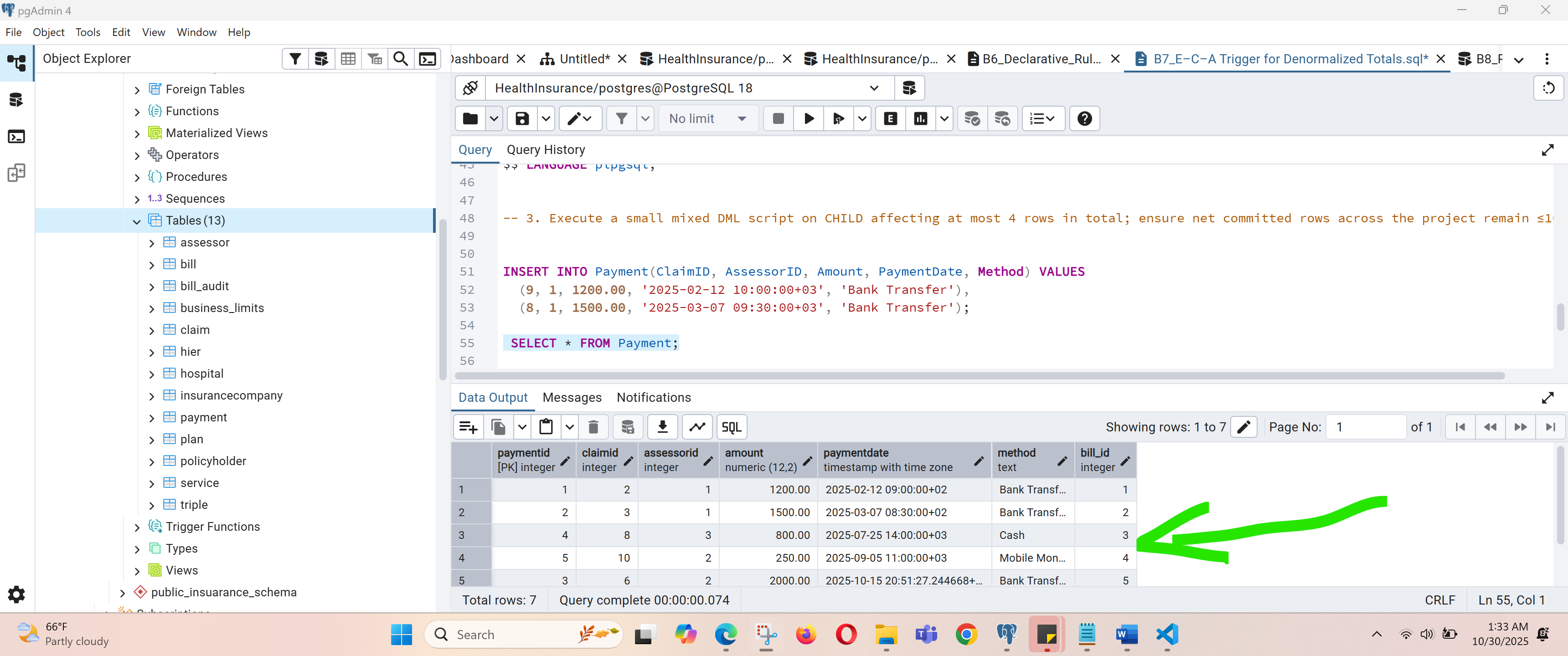
);

RETURN NULL;

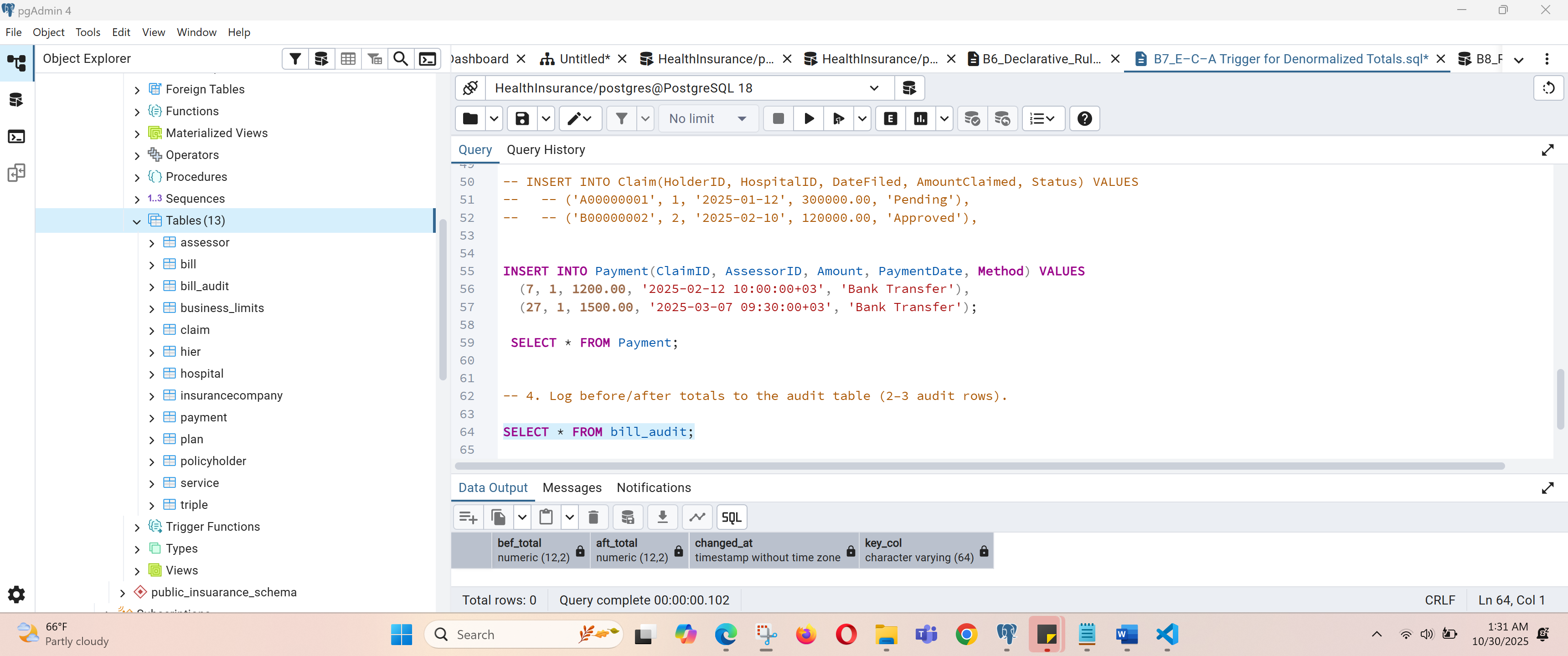
END;

$$ LANGUAGE plpgsql;

* + Mixed DML script and SELECT from totals showing correct precomputations.



* + 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'),

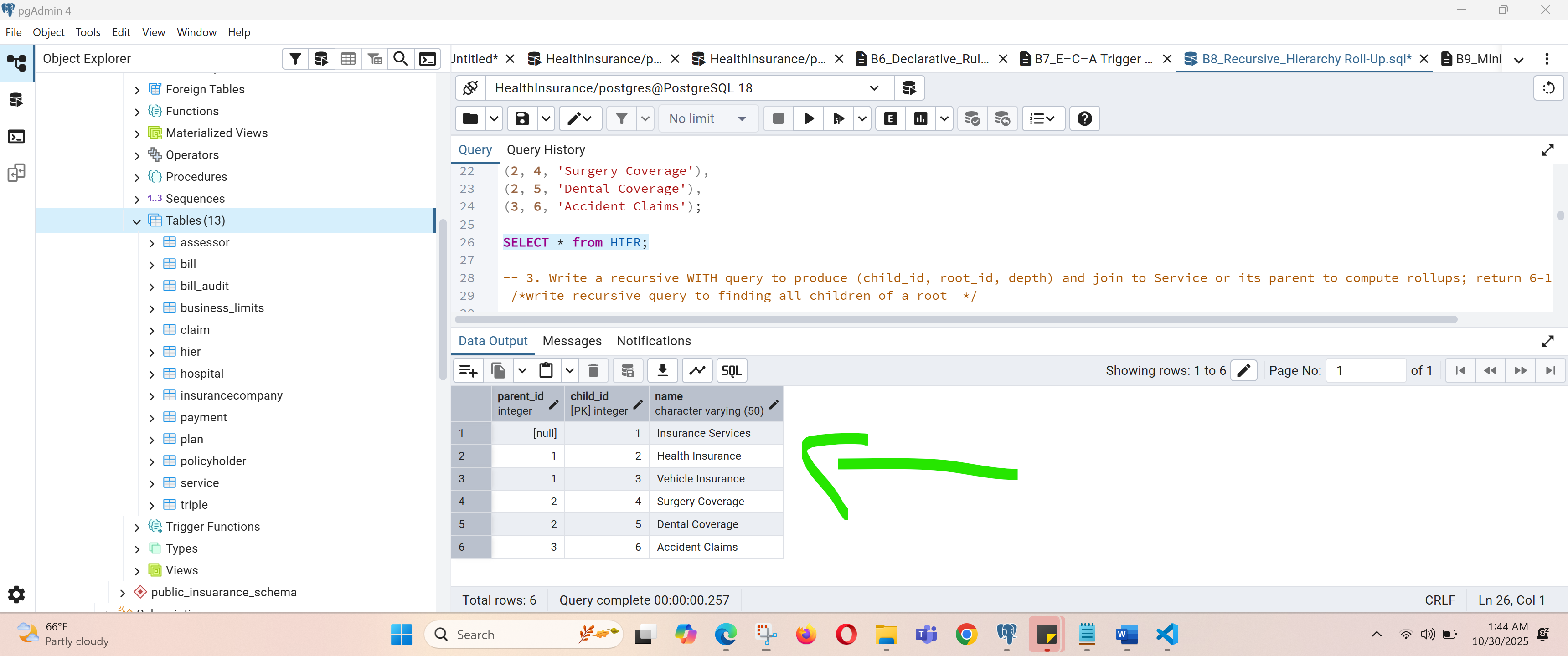
(2, 4, 'Surgery Coverage'),

(2, 5, 'Dental Coverage'),

(3, 6, 'Accident Claims');

SELECT \* from HIER;

Screenshot for HIER table records



* + 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

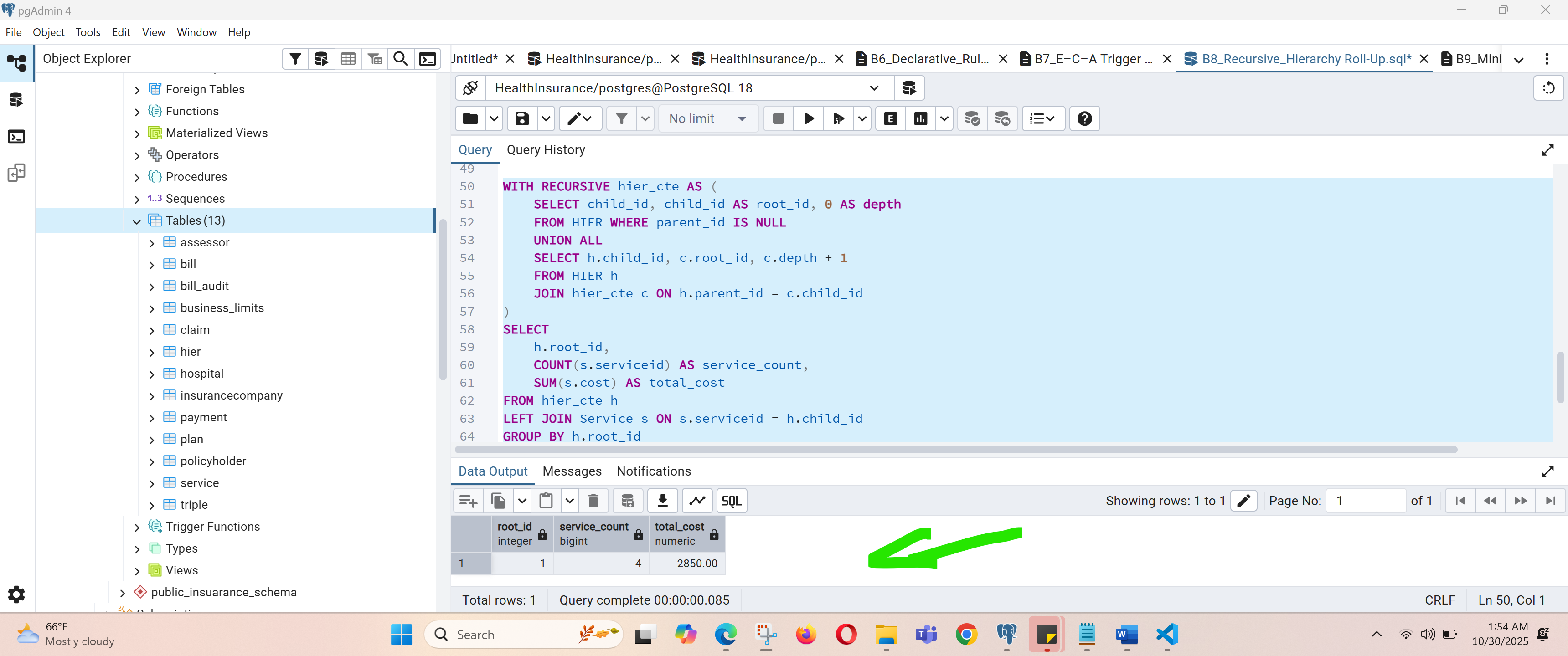
);

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;

.

..

.

.

.

.

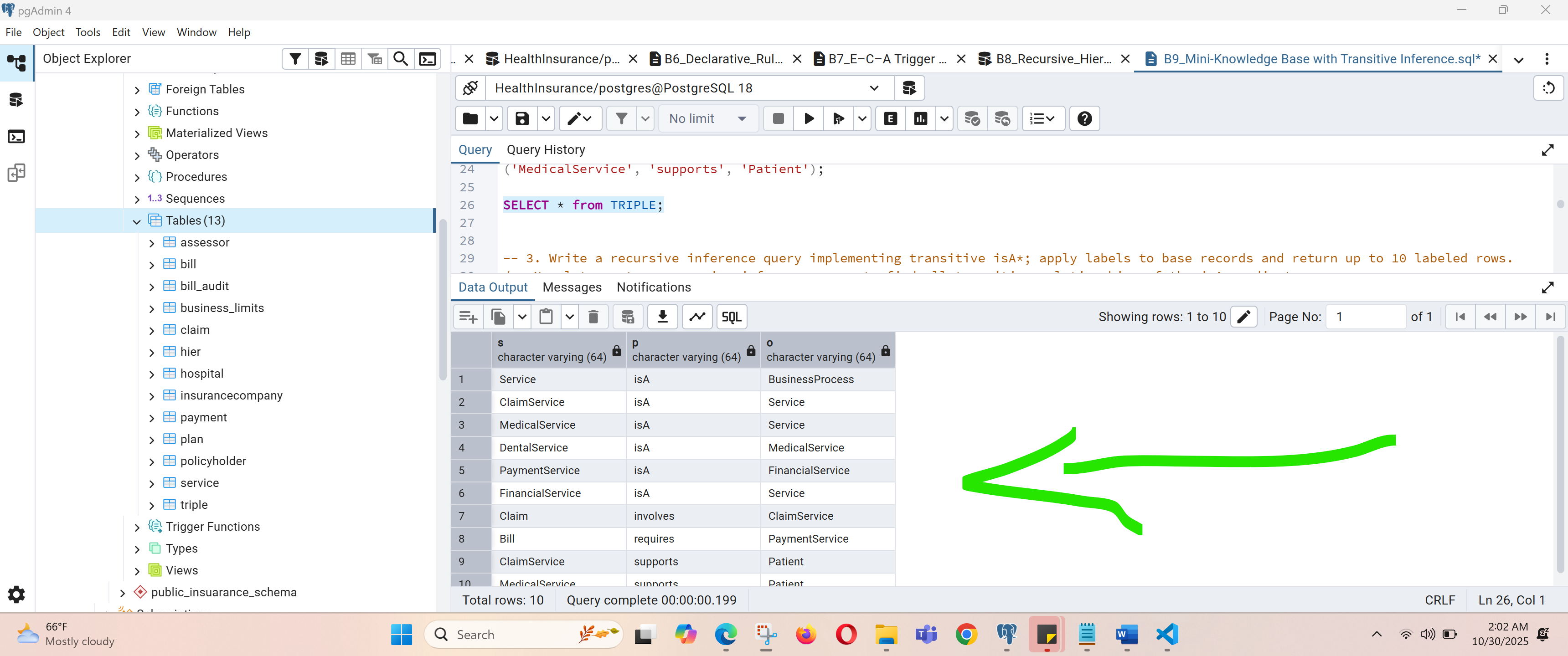
.

.

.

.

Screenshot for inserted data into TRIPLE table



/\*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'

)

FROM isa\_chain

ORDER BY child, superclass

LIMIT 10;.

..

.

.

.

.

.

.

.

.

--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

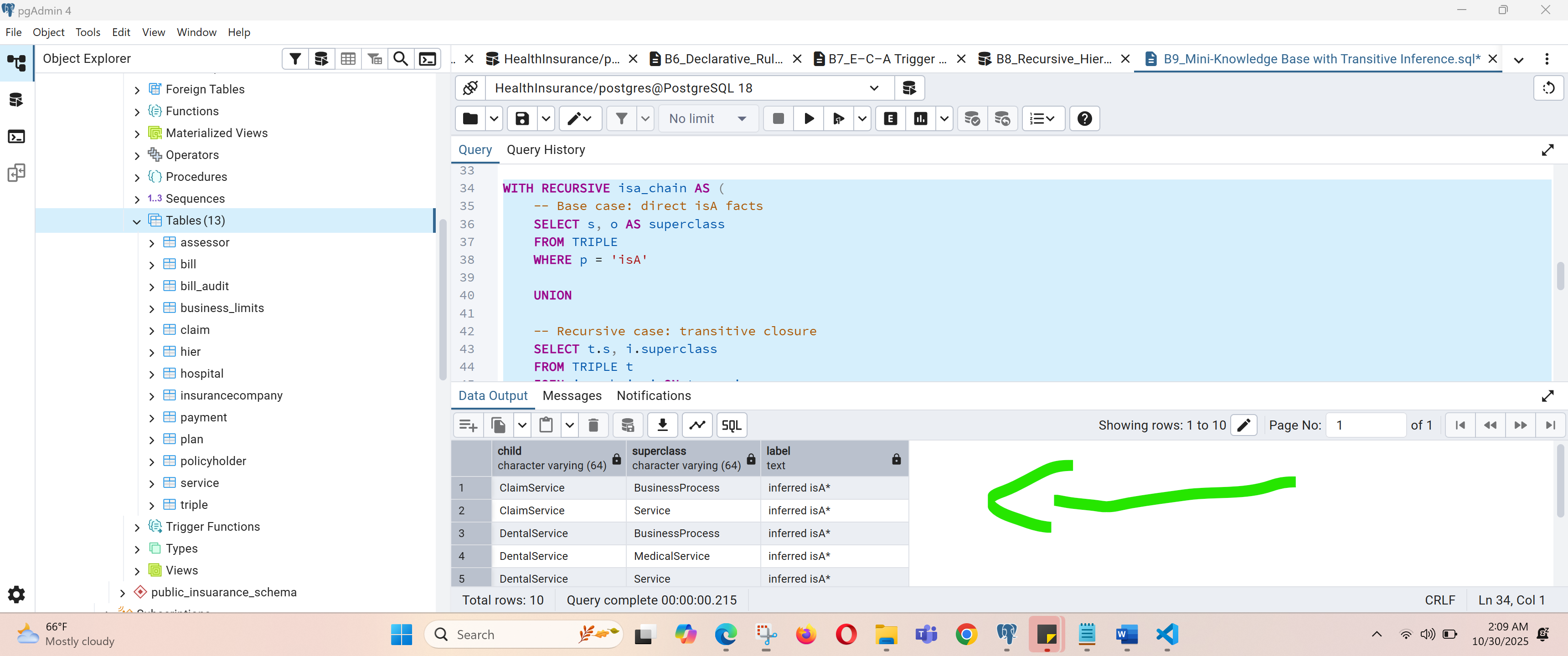
LIMIT 10;

.

.

* + Inference SELECT (with recursive part) and sample labeled output (≤10 rows).
  + Grouping counts proving inferred labels are consistent.
  + Inference SELECT (with recursive part) and sample labeled output (≤10 rows).

Screenshot result of recursive inference query implemented.



/\* quey 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

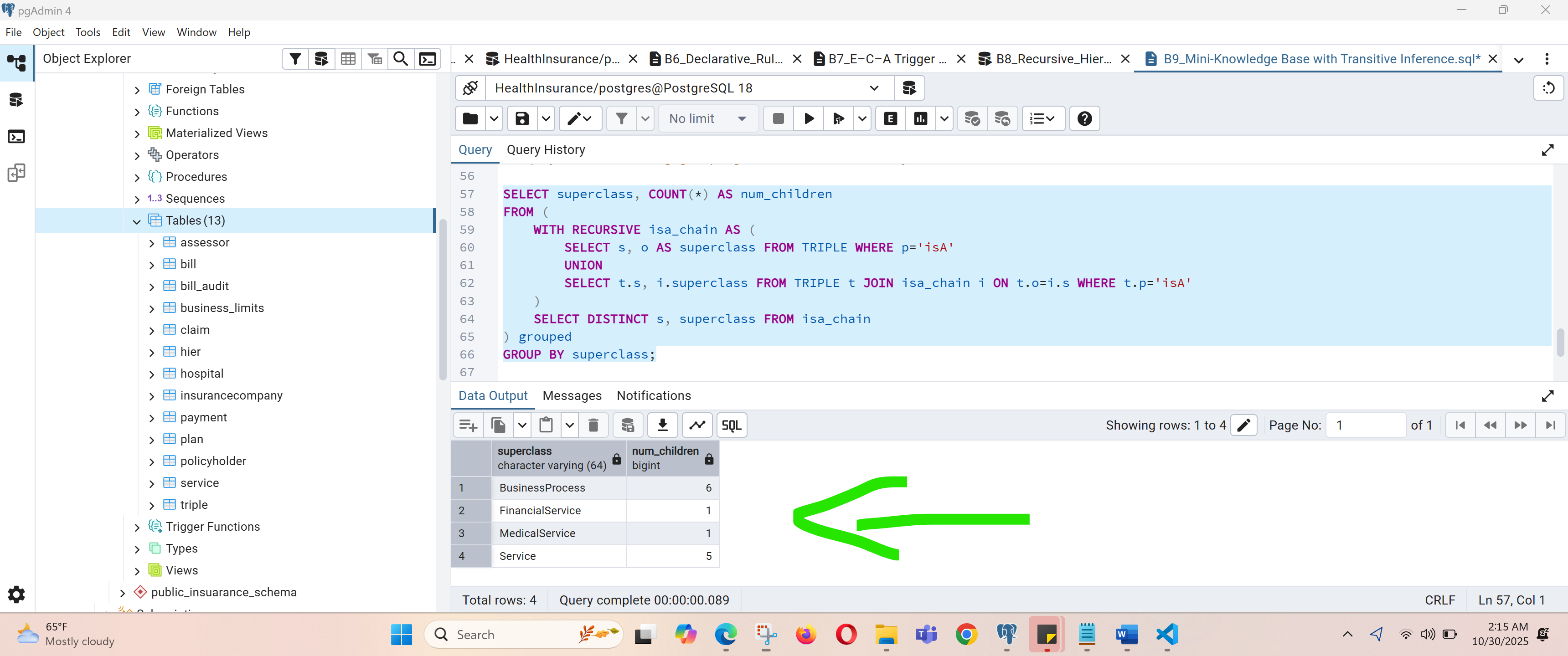
) grouped

GROUP BY superclass;

.

* + Grouping counts proving inferred labels are consistent.

Screenshot showing Grouping counts proving inferred labels.



**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;

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

/\* 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();

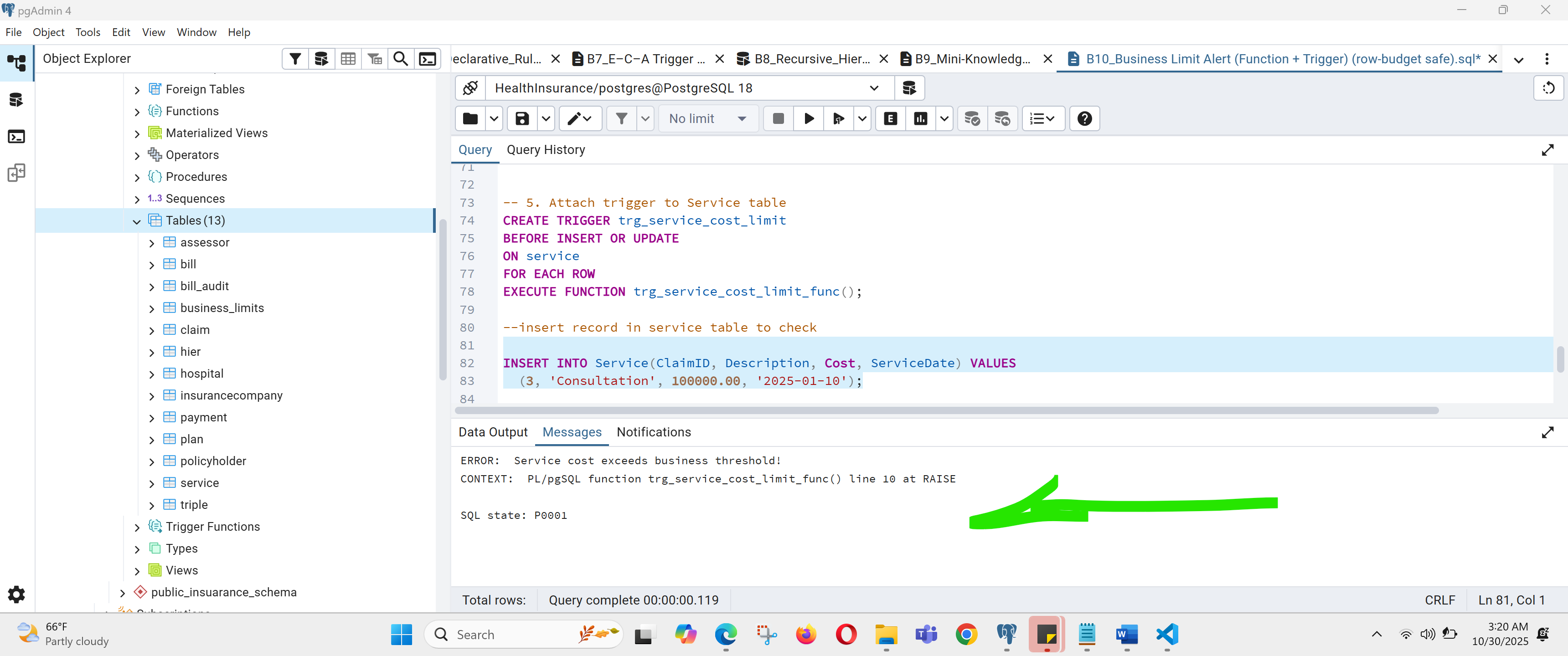
/\*insert record in service table to check\*/

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)



* + SELECT showing resulting committed data consistent with the rule; row budget respected.

Screenshot showing the respecting business cost limit (20000)

