Names: Ngizwenayo Edison
Reg no: 217029434
Module: Advanced Databases Technology

Question 1. On Rules (Declarative Constraints): Safe Prescriptions

-- Prerequisite table

```
CREATE TABLE PATIENT (
  ID NUMBER PRIMARY KEY,
  NAME VARCHAR(100) NOT NULL
);
```

1. **Enhanced DDL definition to store patient medication information**

```
CREATE TABLE patient_med (
    patient_med_id SERIAL PRIMARY KEY, -- unique ID (auto-incremented)
    patient_id INTEGER NOT NULL REFERENCES patient(id), -- must reference an existing patient
    med_name VARCHAR(80) NOT NULL, -- mandatory field for medication name
    dose_mg NUMERIC(6,2) CHECK (dose_mg >= 0), -- dose must be non-negative
    start_dt DATE, -- medication start date
    end_dt DATE,   -- medication end date
    -- Ensure start date is before or equal to end date when both are provided
    CONSTRAINT ck_rx_dates CHECK (start_dt IS NULL OR end_dt IS NULL OR start_dt <= end_dt)
);
```

Below is the result that show that table was created

patient_med_id [PK] integer	patient_id integer	med_name character varying (80)	dose_mg numeric (6,2)	start_dt date	end_dt date

2. **Negative dose violates CHECK constraint**

```
INSERT INTO patient_med
```

(patient_med_id, patient_id, med_name, dose_mg, start_dt, end_dt)

VALUES (1, 1, 'Amoxicillin', -50, TO_DATE('2025-10-01', 'YYYY-MM-DD'), TO_DATE('2025-10-10', 'YYYY-MM-DD'));

Below is error message that pop up when the query is executed

```
ERROR:  Failing row contains (1, 1, Amoxicillin, -50.00, 2025-10-01, 2025-10-10).new row for relation "patient_med" violates check constraint
"patient_med_dose_mg_check"

ERROR:  new row for relation "patient_med" violates check constraint "patient_med_dose_mg_check"
SQL state: 23514
Detail: Failing row contains (1, 1, Amoxicillin, -50.00, 2025-10-01, 2025-10-10).
```

3. **Inverted dates violate CK_RX_DATES constraint**

-- The start date (2025-10-15) is after the end date (2025-10-10),

-- so this will fail the CHECK constraint: start_dt <= end_dt

INSERT INTO patient_med

(patient_med_id, patient_id, med_name, dose_mg, start_dt, end_dt)

VALUES (2, 1, 'Ibuprofen', 200, TO_DATE('2025-10-15', 'YYYY-MM-DD'), TO_DATE('2025-10-10', 'YYYY-MM-DD'));

Below is error message that pop up when the query is executed

```
ERROR:  Failing row contains (2, 1, Ibuprofen, 200.00, 2025-10-15, 2025-10-10).new row for relation "patient_med" violates check constraint "ck_rx_dates"

ERROR:  new row for relation "patient_med" violates check constraint "ck_rx_dates"
SQL state: 23514
Detail: Failing row contains (2, 1, Ibuprofen, 200.00, 2025-10-15, 2025-10-10).
```

4. **Create valid prescription records**

– 1. First of all , insert a valid patient

INSERT INTO PATIENT VALUES (1, 'Eddy Kayinamura');

-- 2. Insert into PATIENT_MED

INSERT INTO PATIENT_MED VALUES (3, 1, 'Paracetamol', 500, TO_DATE('2025-09-01','YYYY-MM-DD'), TO_DATE('2025-10-05','YYYY-MM-DD'));

-- 2. Valid prescription with NULL dates

INSERT INTO PATIENT_MED VALUES (3,1, 'Cetirizine', 10, NULL, NULL);

Below is the result of insert statement

patient_med_id [PK] integer	patient_id integer	med_name character varying (80)	dose_mg numeric (6,2)	start_dt date	end_dt date
2	1	Ibuprofen	200.00	2025-10-15	2025-10-30
3	1	Cetirizine	10.00	[null]	[null]

Error Type	Buggy Code	Correction	Explanation
Missing commas	No commas between column definitions	Added commas between each column definition	SQL requires commas to separate columns in a CREATE TABLE statement
Missing NOT NULL	MED_NAME VARCHAR2(80)	MED_NAME VARCHAR2(80) NOT NULL	Ensures MED_NAME is mandatory
Malformed CHECK clause	DOSE_MG NUMBER(6,2) CHECK DOSE_MG >= 0	DOSE_MG NUMBER(6,2) CHECK (DOSE_MG >= 0)	CHECK constraints must be enclosed in parentheses
Invalid date logic	CHECK (START_DT <= END_DT WHEN BOTH NOT NULL)	CHECK (START_DT IS NULL OR END_DT IS NULL OR START_DT <= END_DT)	SQL doesn't support "WHEN BOTH NOT NULL"; use logical OR to allow NULLs
Missing NOT NULL on FK	PATIENT_ID NUMBER REFERENCES PATIENT(ID)	PATIENT_ID NUMBER NOT NULL REFERENCES PATIENT(ID)	Ensures foreign key is mandatory

Question 2. Active Databases (E–C–A Trigger): Bill Totals That Stay Correct

1. **Define DDL definition for bill, bill_item and bill_audit** table

-- **Main bill table**

```
CREATE TABLE bill (
   id SERIAL PRIMARY KEY,        -- Unique bill ID, auto-incremented
   total NUMERIC(12,2)           -- Total bill amount with two decimal places
```

```
);

-- Bill items table (linked to bills)

CREATE TABLE bill_item (

    bill_id INTEGER NOT NULL,     -- References id in bill table

   amount NUMERIC(12,2),         -- Amount for the specific item

    updated_at DATE,              -- Last update date for the item

    CONSTRAINT fk_bill_item_bill

        FOREIGN KEY (bill_id) REFERENCES bill(id)

    ON DELETE CASCADE          -- optional: delete items if the bill is deleted

);
```

-- Audit log table to capture historical changes in bill totals

```
CREATE TABLE bill_audit (

    bill_id INTEGER NOT NULL,     -- References bill.id

    old_total NUMERIC(12,2),      -- Previous total before change

    new_total NUMERIC(12,2),      -- Updated total after change

    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- when the change
occurred

    CONSTRAINT fk_bill_audit_bill FOREIGN KEY (bill_id) REFERENCES bill(id)

);
```

2. **Insert into bill item table to fire a trigger**

Since bill_item table has bill_id that is referencing to id from bill table, so we will need to into bill table

```
-- Insert initial bills (totals will be updated later by trigger)

INSERT INTO bill (total) VALUES (100);  -- This will get id = 1

INSERT INTO bill (total) VALUES (200);  -- This will get id = 2
```

-- Insert bill items (trigger fires here)

-- Insert rows into bill_item with current date

INSERT INTO bill_item (bill_id, amount, updated_at) VALUES (1, 500, CURRENT_DATE);

INSERT INTO bill_item (bill_id, amount, updated_at) VALUES (1, 300, CURRENT_DATE);

INSERT INTO bill_item (bill_id, amount, updated_at) VALUES (2, 600, CURRENT_DATE);

Below is the query results from bill_item table to check if insert were successful

bill_id integer	amount numeric (12,2)	updated_at date
1	500.00	2025-10-31
1	300.00	2025-10-31
2	600.00	2025-10-31

3. **Define triggered function to update bill total log bill change history in bill_audit in responding to INSERT ,UPDATE and DELETE on bill_item**

```
CREATE OR REPLACE FUNCTION trg_bill_total_stmt()

RETURNS TRIGGER AS $$

DECLARE

    v_bill_id INTEGER;

    v_old_total NUMERIC(12,2);

    v_new_total NUMERIC(12,2);

BEGIN

    -- Loop over all distinct affected bills

    FOR v_bill_id IN

        SELECT DISTINCT bill_id
```

```
        FROM bill_item
        WHERE bill_id IS NOT NULL
    LOOP
        -- Get the current total
        SELECT total INTO v_old_total
        FROM bill
        WHERE id = v_bill_id;

        -- Compute new total (sum of amounts in bill_item)
        SELECT COALESCE(SUM(amount), 0) INTO v_new_total
        FROM bill_item
        WHERE bill_id = v_bill_id;

        -- Update the bill total
        UPDATE bill
        SET total = v_new_total
        WHERE id = v_bill_id;

        -- Insert an audit record
        INSERT INTO bill_audit (bill_id, old_total, new_total, changed_at)
        VALUES (v_bill_id, v_old_total, v_new_total, CURRENT_TIMESTAMP);
    END LOOP;

    RETURN NULL;  -- statement-level trigger
END;
```

```
$$ LANGUAGE plpgsql;


-- Attach trigger to BILL_ITEM

CREATE TRIGGER trg_bill_total_stmt

AFTER INSERT OR UPDATE OR DELETE ON bill_item

FOR EACH STATEMENT

EXECUTE FUNCTION trg_bill_total_stmt();
```

3.1 Insert bill items to trigger trg_bill_total_stmt function

```
INSERT INTO bill_item (bill_id, amount, updated_at)  VALUES (1, 500,
CURRENT_DATE);

INSERT INTO bill_item (bill_id, amount, updated_at)VALUES (1, 300,
CURRENT_DATE);

INSERT INTO bill_item (bill_id, amount, updated_at)

VALUES (2, 600, CURRENT_DATE);
```

3.2 . Update an existing bill item (trigger fires to recalc total and log audit)

```
UPDATE bill_item

SET amount = 600,

updated_at = CURRENT_TIMESTAMP   -- optional: refresh timestamp

WHERE bill_id = 1
```

3. 3. Delete a bill item (trigger fires again)

```
DELETE FROM bill_item

WHERE bill_id = 2

  AND amount = 300;
```

3.4. Retrieve the bill data and Bill audit to change if the update were made

id [PK] integer	total numeric (12,2)
2	600.00
1	1200.00

bill_id integer	old_total numeric (12,2)	new_total numeric (12,2)	changed_at timestamp without time zone
1	100.00	500.00	2025-10-31 09:38:24.71987
1	500.00	800.00	2025-10-31 09:38:24.71987
2	200.00	600.00	2025-10-31 09:38:24.71987
1	800.00	800.00	2025-10-31 09:38:24.71987
2	600.00	600.00	2025-10-31 09:59:05.954163
1	800.00	800.00	2025-10-31 09:59:05.954163
2	600.00	600.00	2025-10-31 10:00:56.13091
1	800.00	800.00	2025-10-31 10:00:56.13091
2	600.00	600.00	2025-10-31 10:08:07.802817
1	800.00	600.00	2025-10-31 10:08:07.802817
2	600.00	600.00	2025-10-31 10:08:40.913015
1	600.00	1200.00	2025-10-31 10:08:40.913015

From the above screenshot it is clear that any change made on bill_item will be logged in the bill audit table.

Question 3. Deductive Databases (Recursive WITH): Referral/Supervision Chain

Prerequisite

```
-- Table to store employees and their supervisors
CREATE TABLE staff_supervisor (
    employee VARCHAR(50) NOT NULL,   -- Employee name
    supervisor VARCHAR(50)           -- Supervisor name
);
```

3.1 Insert some example employees and their supervisors

```
INSERT INTO staff_supervisor (employee, supervisor)
```

VALUES ('Alice', 'John'),('Bob', 'John'),('Charlie', 'Alice'), ('Diana', 'Alice'),('Eve', 'Bob');

3.2 Corrected recursive query for all bugs fixed.

– BUGGY were anchor hop count, join direction, and final selection are off

```
WITH RECURSIVE supers (emp, sup, hops, path) AS (
    -- Anchor: start with direct supervision, hop count = 1
    SELECT employee, supervisor, 1, employee || '>' || supervisor
    FROM staff_supervisor
    WHERE supervisor IS NOT NULL
    UNION ALL
    -- Recursive: climb up the supervision chain
    SELECT s.employee, t.sup, t.hops + 1, t.path || '>' || t.sup
    FROM staff_supervisor s
    JOIN supers t ON s.supervisor = t.emp
    WHERE POSITION(t.sup IN t.path) = 0  -- cycle guard
)
-- Final selection: top supervisor per employee
SELECT emp, sup AS top_supervisor, hops
FROM (
    SELECT emp, sup, hops,
        RANK() OVER (PARTITION BY emp ORDER BY hops DESC) AS rnk
    FROM supers
) sub
WHERE rnk = 1;
```

Below is the result of the query

emp character varying (50)	top_supervisor character varying (50)	hops integer
Alice	John	1
Bob	John	1
Charlie	Alice	1
Diana	Alice	1
Eve	Bob	1

Below is the table that shows bugs that were fixed

Bug	Fix
Anchor hop count was 0	Set to 1 to reflect first supervision step
Join direction was reversed	Corrected to climb up: S.SUPERVISOR = T.EMP
Cycle guard was naive	Improved with POSITION(t.sup IN t.path) = 0
Scalar subquery with MAX(HOPS or the **number of steps** it takes to reach an employee's **top supervisor** by following the chain of supervision)	Replaced with RANK() analytic function for clarity and correctness

Question 4. Knowledge Bases (Triples & Ontology): Infectious-Disease Roll-Up

4.1. Table to store subject-predicate-object triples

```
CREATE TABLE triple (
    s VARCHAR(100),  -- Subject
    p VARCHAR(50),   -- Predicate
    o VARCHAR(100)   -- Object
);
```

4.1 Insert patient diagnoses as triples

```
INSERT INTO triple (s, p, o)
VALUES ('patient1', 'hasDiagnosis', 'Influenza'),
```

('patient2', 'hasDiagnosis', 'COVID19'),

('patient3', 'hasDiagnosis', 'Malaria');

-- Taxonomy edges

INSERT INTO TRIPLE

VALUES ('Influenza', 'isA', 'ViralInfection'),

('COVID19', 'isA', 'ViralInfection'),

('Malaria', 'isA', 'ParasiticInfection'),

('ViralInfection', 'isA', 'InfectiousDisease'),

('ParasiticInfection', 'isA', 'InfectiousDisease'),

('Diabetes', 'isA', 'ChronicDisease');

Query triple to check is insert were successful

s character varying (100)	p character varying (50)	o character varying (100)
patient1	hasDiagnosis	Influenza
patient2	hasDiagnosis	COVID19
patient3	hasDiagnosis	Malaria
Influenza	isA	ViralInfection
COVID19	isA	ViralInfection
Malaria	isA	ParasiticInfection
ViralInfection	isA	InfectiousDisease
ParasiticInfection	isA	InfectiousDisease
Diabetes	isA	ChronicDisease

4.2 Recursive query to find patients diagnosed with an infectious disease

WITH RECURSIVE isa (ancestor, child) AS (

```
    -- Anchor: direct isA relationships from the triple table
    SELECT o, s FROM triple WHERE p = 'isA'
    UNION ALL
    -- Recursive: climb up the taxonomy (find indirect ancestors)
    SELECT i.ancestor, t.s FROM triple t
    JOIN isa i ON t.p = 'isA' AND t.o = i.child
),
infectious_patients AS (
    -- Select patients whose diagnosis is a descendant of InfectiousDisease
    SELECT DISTINCT t.s FROM triple t
    JOIN isa ON t.o = isa.child WHERE t.p = 'hasDiagnosis'
      AND isa.ancestor = 'InfectiousDisease'
)
-- Final result: list of patient IDs
SELECT s AS patient_id FROM infectious_patients;
```

Below is the query result

patient_id character varying (100)
patient1
patient2
patient3

From the above result, the recursive query organizes information in a flexible, searchable format that connects related concepts, such as linking specific diseases to their broader categories and supports logical reasoning and inference, allowing conclusions to be drawn automatically (for example, if Influenza is an

InfectiousDisease, then a patient diagnosed with Influenza can be inferred to have an InfectiousDisease.