AFRICAN CENTRE OF EXCELLENCE IN DATA SCIENCE

COLLEGE OF BUSINESS & ECONOMICS

UNIVERSITY OF RWANDA (UR GIKONDO CAMPUS)

COLLEGE OF BUSINESS AND ECONOMICS (CBE)

NAMES: KABWALI MASUDI Dischon

REGISTRATION NUMBER: 223027551

SCHOOL: ACE-DS, COHORT 6 PROGRAM:

MSc IN DATA SCIENCE IN DATA MINING

ACADEMIC YEAR: 2024-2025

SPECIALIZATION MODULE: ADVANCED DATABASED TECHNOLOGY

ASSIGNMENT 4: INTELLIGENT DATABASES

Use case study: Intelligent databases

Introduction

This assignment explores five key domains of Intelligent Databases, each focusing on a different aspect of database logic, integrity, and reasoning. The solutions are implemented in PostgreSQL 16 (with PostGIS for spatial tasks) and demonstrate the use of constraints, triggers, recursive queries, ontological reasoning, and spatial operations.

The assignment includes the following components:

task1_safe_prescriptions.sql: implements the HEALTHNET schema with patient and patient_med tables. Declarative constraints ensure data integrity through NOT NULL, CHECK, and foreign key validations, preventing invalid prescriptions. The script includes both failing and passing insert examples. task2_bill_totals_trigger.sql: defines billing tables and a trigger mechanism that maintains consistent BILL.TOTAL values. A safe session-level approach ensures that each bill's total is recalculated only once per statement, while all updates are logged in an audit table. task3_supervision_chain.sql: uses a recursive common table expression (CTE) to trace employee supervision hierarchies. It determines each employee's top supervisor and the number of hierarchical hops, including logic to prevent cycles. task4_infectious_rollup.sql: models ontological relationships through a triples table and computes the transitive closure of isA relationships. The query identifies all patients diagnosed with diseases that ultimately classify as Infectious Diseases. task5_spatial_clinics.sql: demonstrates spatial database reasoning using PostGIS. It creates a spatially indexed clinic table, then queries clinics within a 1 km radius of an ambulance's location and identifies the three nearest clinics with precise distance calculations.

In PostgreSQL, a database serves as a container for multiple schemas, each acting as a logical namespace that organizes database objects such as tables, views, indexes, synonyms, functions, and sequences. This layered structure enhances clarity, security, and scalability by separating data logically within the same database. For instance, i might maintain operational tables in the public schema, logging data in an audit schema, and analytical summaries in an analytics schema, all coexisting independently. Schemas also prevent naming conflicts, allowing identical table names (public.ticket and analytics.ticket) without interference. By managing access at the schema level and customizing the search_path (SET search_path TO analytics, public;), PostgreSQL enables

precise control over visibility and permissions, supporting clean multi-tenant, modular, and enterprise-grade database architectures.

Create and connect to a database called hospital

```
postgres=# CREATE DATABASE hospital;
CREATE DATABASE
postgres=# \c hospital;
You are now connected to database "hospital" as user "postgres".
```

Task1: Safe prescriptions

Create database objects (tables) in schema healthment.

```
hospital=# CREATE SCHEMA IF NOT EXISTS healthnet;
CREATE SCHEMA
hospital=# SET search_path = healthnet;
SET
hospital=# -- Patients table
hospital=# CREATE TABLE IF NOT EXISTS patient (
hospital(#     id INT PRIMARY KEY,
hospital(#     name VARCHAR(100) NOT NULL
hospital(# );
CREATE TABLE
hospital=# -- Patient medications table with explicit constraints
hospital=# CREATE TABLE IF NOT EXISTS patient_med (
hospital(#     patient_med_id INT PRIMARY KEY,
hospital(#     patient_id INT NOT NULL REFERENCES patient(id),
hospital(#     med_name VARCHAR(80) NOT NULL,
hospital(#     dose_mg NUMERIC(6,2) NOT NULL CONSTRAINT ck_dose_mg CHECK (dose_mg >= 0),
hospital(#     start_dt DATE NOT NULL,
hospital(#     end_dt DATE NOT NULL,
hospital(#     CONSTRAINT ck_rx_dates CHECK (start_dt <= end_dt)
hospital(# );
CREATE TABLE
hospital=#
```

Passing inserts: insert sample data in created tables such as patient and patient_med.

```
hospital=# -- Sample data (passing inserts)
hospital=# INSERT INTO patient(id, name) VALUES (1, 'Alice') ON CONFLICT DO NOTHING;
INSERT 0 1
hospital=# INSERT INTO patient(id, name) VALUES (2, 'Bob') ON CONFLICT DO NOTHING;
INSERT 0 1
hospital=#
```

```
hospital=# INSERT INTO patient_med(patient_med_id, patient_id, med_name, dose_mg, start_dt, end_dt)
hospital-# VALUES (3, 1, 'Paracetamol', 500, DATE '2025-10-26', DATE '2025-10-30')
hospital-# ON CONFLICT DO NOTHING;
INSERT 0 1
hospital=# INSERT INTO patient_med(patient_med_id, patient_id, med_name, dose_mg, start_dt, end_dt)
hospital-# VALUES (4, 2, 'Amoxicillin', 250, DATE '2025-10-26', DATE '2025-11-02')
hospital-# ON CONFLICT DO NOTHING;
INSERT 0 1
hospital=#
```

Failing inserts (examples): these will raise CHECK or FK errors

```
hospital=# -- 1) Negative dose which violates ck_dose_mg
hospital=# INSERT INTO patient_med(patient_med_id, patient_id, med_name, dose_mg, start_dt, end_dt)
hospital-# VALUES (1, 1, 'Paracetamol', -50, DATE '2025-10-26', DATE '2025-10-30');
ERROR:  new row for relation "patient_med" violates check constraint "ck_dose_mg"
DETAIL:  Failing row contains (1, 1, Paracetamol, -50.00, 2025-10-26, 2025-10-30).
hospital=# -- 2) Inverted dates that violates ck_rx_dates
hospital=# INSERT INTO patient_med(patient_med_id, patient_id, med_name, dose_mg, start_dt, end_dt)
hospital-# VALUES (2, 1, 'Ibuprofen', 200, DATE '2025-11-01', DATE '2025-10-30');
ERROR:  new row for relation "patient_med" violates check constraint "ck_rx_dates"
DETAIL:  Failing row contains (2, 1, Ibuprofen, 200.00, 2025-11-01, 2025-10-30).hospital=# -- 3) Missing patient that violates FK
hospital=# INSERT INTO patient_med(patient_med_id, patient_id, med_name, dose_mg, start_dt, end_dt)
hospital-# VALUES (5, 99, 'Aspirin', 100, DATE '2025-10-26', DATE '2025-10-30');
ERROR:  insert or update on table "patient_med" violates foreign key constraint "patient_med_patient_id_fkey"
DETAIL:  Key (patient_id)=(99) is not present in table "patient".
hospital=#
```

	Types of constraint violated	***Constraint name***	***Error description***	***Reason for failure***
1	CHECK Constraint	ck_dose_mg	New row for relation "patient_med" violates check constraint "ck_dose_mg"	The inserted dose (-50) is negative, violating the rule that medication dose must be greater than zero.
2	CHECK Constrain	ck_rx_dates	New row for relation "patient_med" violates check constraint "ck_rx_dates"	The end date (2025-10-30) is earlier than the start date (2025-11-01), violating the rule that end_dt ≥ start_dt.
3	FOREIGN KEY Constraint	patient_med_patient_id_fkey	Insert or update on table "patient_med" violates foreign key constraint "patient_med_patient_id_fkey"	The specified patient_id (99) does not exist in the patient table, violating referential integrity.

Task 2: Bill totals trigger

```
hospital=# CREATE SCHEMA IF NOT EXISTS billing;
CREATE SCHEMA
hospital=# SET search_path = billing;
SET
hospital=#
```

Create database objects in schema billing

```
hospital=# CREATE TABLE IF NOT EXISTS bill (
hospital(#     id INT PRIMARY KEY,
hospital(#     total NUMERIC(12,2) DEFAULT 0
hospital(# );
CREATE TABLE
hospital=# CREATE TABLE IF NOT EXISTS bill_item (
hospital(#     bill_item_id SERIAL PRIMARY KEY,
hospital(#     bill_id INT NOT NULL REFERENCES bill(id),
hospital(#     amount NUMERIC(12,2) NOT NULL,
hospital(#     updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
hospital(# );
CREATE TABLE
hospital=# CREATE TABLE IF NOT EXISTS bill_audit (
hospital(#     audit_id SERIAL PRIMARY KEY,
hospital(#     bill_id INT NOT NULL,
hospital(#     old_total NUMERIC(12,2),
hospital(#     new_total NUMERIC(12,2),
hospital(#     changed_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
hospital(# );
CREATE TABLE
hospital=# -- This temp table is session-local. ON COMMIT DELETE ROWS clears rows after commit.
hospital=# CREATE TEMP TABLE IF NOT EXISTS tmp_changed_bills (
hospital(#     bill_id INT PRIMARY KEY
hospital(# ) ON COMMIT DELETE ROWS;
CREATE TABLE
hospital=#
```

```
hospital=# -- Row-level trigger function: collect affected bill ids
hospital=# CREATE OR REPLACE FUNCTION billing.trg_collect_bill_changes()
hospital-# RETURNS TRIGGER AS $$
hospital$# BEGIN
hospital$#     IF (TG_OP = 'INSERT' OR TG_OP = 'UPDATE') THEN
hospital$#         -- NEW.bill_id exists
hospital$#         INSERT INTO tmp_changed_bills(bill_id) VALUES (NEW.bill_id)
hospital$#         ON CONFLICT DO NOTHING;
hospital$#     ELSIF (TG_OP = 'DELETE') THEN
hospital$#         INSERT INTO tmp_changed_bills(bill_id) VALUES (OLD.bill_id)
hospital$#         ON CONFLICT DO NOTHING;
hospital$#     END IF;
hospital$#     RETURN NULL;
hospital$# END;
hospital$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
hospital=#
hospital=# DROP TRIGGER IF EXISTS trg_bill_total_each_row ON bill_item;
NOTICE:  trigger "trg_bill_total_each_row" for relation "bill_item" does not exist, skipping
DROP TRIGGER
hospital=# CREATE TRIGGER trg_bill_total_each_row
hospital-# AFTER INSERT OR UPDATE OR DELETE ON bill_item
hospital-# FOR EACH ROW EXECUTE FUNCTION billing.trg_collect_bill_changes();
CREATE TRIGGER
hospital=#
```

```
hospital=# -- Statement-level trigger function: recompute totals once per affected bill and insert audit
hospital=# CREATE OR REPLACE FUNCTION billing.trg_update_bill_totals()
hospital-# RETURNS TRIGGER AS $$
hospital$# DECLARE
hospital$#     rec RECORD;
hospital$#     v_old_total NUMERIC(12,2);
hospital$#     v_new_total NUMERIC(12,2);
hospital$# BEGIN
hospital$#     -- If tmp_changed_bills is empty, do nothing
hospital$#     FOR rec IN SELECT bill_id FROM tmp_changed_bills LOOP
hospital$#         -- get old total (treat NULL as 0)
hospital$#         SELECT COALESCE(total,0) INTO v_old_total FROM bill WHERE id = rec.bill_id;
hospital$#         -- compute new total
hospital$#         SELECT COALESCE(SUM(amount),0) INTO v_new_total FROM bill_item WHERE bill_id = rec.bill_id;
hospital$#         -- update bill total
hospital$#         UPDATE bill SET total = v_new_total WHERE id = rec.bill_id;
hospital$#         -- insert audit row
hospital$#         INSERT INTO bill_audit(bill_id, old_total, new_total, changed_at)
hospital$#         VALUES (rec.bill_id, v_old_total, v_new_total, NOW());
hospital$#     END LOOP;
hospital$#
hospital$#     -- Cleanup (rows in tmp_changed_bills are cleared ON COMMIT; explicit delete keeps session tidy)
hospital$#     DELETE FROM tmp_changed_bills;
hospital$#     RETURN NULL;
hospital$# END;
hospital$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
hospital=#
hospital=# DROP TRIGGER IF EXISTS trg_bill_total_stmt ON bill_item;
NOTICE:  trigger "trg_bill_total_stmt" for relation "bill_item" does not exist, skipping
DROP TRIGGER
hospital=# CREATE TRIGGER trg_bill_total_stmt
hospital-# AFTER INSERT OR UPDATE OR DELETE ON bill_item
hospital-# FOR EACH STATEMENT EXECUTE FUNCTION billing.trg_update_bill_totals();
CREATE TRIGGER
hospital=#
```

Small mixed-DML (INSERT, UPDATE, and DELETE key words) test script:

Prepare test data

```
hospital=# INSERT INTO bill(id, total) VALUES (1,0) ON CONFLICT DO NOTHING;
INSERT 0 1
hospital=# INSERT INTO bill(id, total) VALUES (2,0) ON CONFLICT DO NOTHING;
INSERT 0 1
hospital=#
```

Batch: insert items for bill 1 and 2

```
hospital=# INSERT INTO bill_item(bill_id, amount) VALUES (1, 100), (1, 50), (2, 200);
INSERT 0 3
hospital=#
```

Update one item by changing the amount

```
hospital=# UPDATE bill_item SET amount = 75 WHERE bill_id = 1 AND amount = 50;
UPDATE 1
hospital=# select * from bill_item;
 bill_item_id | bill_id | amount |          updated_at
--------------+---------+--------+-------------------------------
            1 |       1 | 100.00 | 2025-10-30 22:26:37.409711+03
            3 |       2 | 200.00 | 2025-10-30 22:26:37.409711+03
            2 |       1 |  75.00 | 2025-10-30 22:26:37.409711+03
(3 rows)
```

Delete one item

```
hospital=# DELETE FROM bill_item WHERE bill_id = 2;
DELETE 1
hospital=# select * from bill_item;
 bill_item_id | bill_id | amount |          updated_at
--------------+---------+--------+-------------------------------
            1 |       1 | 100.00 | 2025-10-30 22:26:37.409711+03
            2 |       1 |  75.00 | 2025-10-30 22:26:37.409711+03
(2 rows)
```

Inspect results: bill totals and audit rows

```
hospital=# SELECT * FROM bill ORDER BY id;
 id | total
----+--------
  1 | 175.00
  2 |   0.00
(2 rows)


hospital=# SELECT * FROM bill_audit ORDER BY changed_at DESC;
 audit_id | bill_id | old_total | new_total |          changed_at
----------+---------+-----------+-----------+-------------------------------
        4 |       2 |    200.00 |      0.00 | 2025-10-30 22:30:00.774552+03
        3 |       1 |    150.00 |    175.00 | 2025-10-30 22:27:53.185893+03
        1 |       1 |      0.00 |    150.00 | 2025-10-30 22:26:37.409711+03
        2 |       2 |      0.00 |    200.00 | 2025-10-30 22:26:37.409711+03
(4 rows)
```

Task 3: Referral/supervision chain (recursive WITH)

Create schema org

```
hospital=# CREATE SCHEMA IF NOT EXISTS org;
CREATE SCHEMA
hospital=# SET search_path = org;
SET
hospital=#
```

Create database objects of schema org

```
hospital=# CREATE TABLE IF NOT EXISTS staff_supervisor (
hospital(#     employee TEXT PRIMARY KEY,
hospital(#     supervisor TEXT   -- NULL if top-level (no supervisor)
hospital(# );
CREATE TABLE
hospital=#
```

Sample data (5 – 6 rows, including a small cycle example)

```
hospital=# INSERT INTO staff_supervisor(employee, supervisor) VALUES
hospital-# ('Alice', 'Bob'),
hospital-# ('Bob', 'Carol'),
hospital-# ('Carol', 'Dana'),
hospital-# ('Eve', 'Bob'),
hospital-# ('Frank', NULL),
hospital-# ('LoopA', 'LoopB'),
hospital-# ('LoopB', 'LoopA') ON CONFLICT DO NOTHING;
INSERT 0 7
hospital=#
```

Recursive CTE that follows supervisors upward, tracks path to avoid cycles

```
HINT:  Perhaps you meant to reference the table alias "s".
hospital=# WITH RECURSIVE supers(emp, top_sup, hops, path) AS (
hospital(#     -- Anchor: direct supervisor relationship; hops = 1
hospital(#     SELECT employee AS emp,
hospital(#            supervisor AS top_sup,
hospital(#            1 AS hops,
hospital(#            ARRAY[employee, supervisor]::text[] AS path
hospital(#     FROM staff_supervisor
hospital(#     WHERE supervisor IS NOT NULL
hospital(#
hospital(#     UNION ALL
hospital(#
hospital(#     -- Recursive step: climb from current top_sup to its supervisor
hospital(#     SELECT s.emp,
hospital(#            ss.supervisor AS top_sup,
hospital(#            s.hops + 1 AS hops,
hospital(#            s.path || ss.supervisor
hospital(#     FROM supers s
hospital(#     JOIN staff_supervisor ss ON ss.employee = s.top_sup
hospital(#     -- avoid cycles: do not follow if supervisor already in path or supervisor IS NULL
hospital(#     WHERE ss.supervisor IS NOT NULL
hospital(#       AND NOT ss.supervisor = ANY(s.path)
hospital(# ),
hospital-# ranked AS (
hospital(#     SELECT emp, top_sup, hops,
hospital(#            ROW_NUMBER() OVER (PARTITION BY emp ORDER BY hops DESC) AS rn
hospital(#     FROM supers
hospital(# )
hospital-# SELECT emp AS employee,
hospital-#        top_sup AS top_supervisor,
hospital-#        hops
hospital-# FROM ranked
hospital-# WHERE rn = 1
hospital-# ORDER BY employee;
```

Output

```
 employee | top_supervisor | hops
----------+----------------+------
 Alice    | Dana           |    3
 Bob      | Dana           |    2
 Carol    | Dana           |    1
 Eve      | Dana           |    3
 LoopA    | LoopB          |    1
 LoopB    | LoopA          |    1
(6 rows)


hospital=#
```

Task 4: Infectious disease roll-up: triples and ontology

Create schema kb

```
hospital=# CREATE SCHEMA IF NOT EXISTS kb;
CREATE SCHEMA
hospital=# SET search_path = kb;
SET
hospital=#
```

Create database objects in the schema kb

```
hospital=# CREATE TABLE IF NOT EXISTS triple (
hospital(#     s TEXT,
hospital(#     p TEXT,
hospital(#     o TEXT
hospital(# );
CREATE TABLE
hospital=# -- Sample triples (~8 rows)
hospital=# INSERT INTO triple(s,p,o) VALUES
hospital-# ('patient1','hasDiagnosis','Influenza'),
hospital-# ('patient2','hasDiagnosis','CommonCold'),
hospital-# ('patient3','hasDiagnosis','COVID19'),
hospital-# ('Influenza','isA','ViralInfection'),
hospital-# ('CommonCold','isA','ViralInfection'),
hospital-# ('ViralInfection','isA','InfectiousDisease'),
hospital-# ('COVID19','isA','InfectiousDisease'),
hospital-# ('BacterialSepsis','isA','InfectiousDisease')
hospital-# ON CONFLICT DO NOTHING;
INSERT 0 8
hospital=#
```

Compute transitive closure of isA: ISA(child, ancestor)

```
hospital=# -- Compute transitive closure of isA: ISA(child, ancestor)
hospital=# WITH RECURSIVE isa(child, ancestor) AS (
hospital(#     -- anchor
hospital(#     SELECT s AS child, o AS ancestor FROM triple WHERE p = 'isA'
hospital(#     UNION
hospital(#     -- recursive: if child -> mid and mid -> ancestor, then child -> ancestor
hospital(#     SELECT i.child, t.o AS ancestor
hospital(#     FROM isa i
hospital(#     JOIN triple t ON t.p = 'isA' AND t.s = i.ancestor
hospital(# )
hospital-# -- Now find patients whose diagnosis isA* InfectiousDisease
hospital-# SELECT DISTINCT t.s AS patient_id
hospital-# FROM triple t
hospital-# JOIN isa ON t.o = isa.child
hospital-# WHERE t.p = 'hasDiagnosis' AND isa.ancestor = 'InfectiousDisease';
 patient_id
------------
 patient1
 patient2
 patient3
(3 rows)


hospital=#
```

Task 5: Spatial – clinics within 1km and nearest 3 with PostGIS

Create extension postgis

-- Requires PostGIS extension

```
CREATE EXTENSION IF NOT EXISTS postgis;

CREATE SCHEMA IF NOT EXISTS spatial;

SET search_path = spatial;

-- Clinics table: use geography for easy distance (meters)

CREATE TABLE IF NOT EXISTS clinic (

id SERIAL PRIMARY KEY,

name TEXT,

geom GEOGRAPHY(POINT,4326) -- lon/lat stored as geography

);

-- Spatial index

CREATE INDEX IF NOT EXISTS clinic_geom_gix ON clinic USING GIST (geom);

-- Sample clinics (lon, lat)

INSERT INTO clinic(name, geom) VALUES

('Clinic A', ST_SetSRID(ST_MakePoint(30.0605, -1.9565), 4326)::geography),

('Clinic B', ST_SetSRID(ST_MakePoint(30.0610, -1.9575), 4326)::geography),

('Clinic C', ST_SetSRID(ST_MakePoint(30.0580, -1.9580), 4326)::geography),

('Clinic D', ST_SetSRID(ST_MakePoint(30.0700, -1.9600), 4326)::geography),

('Clinic E', ST_SetSRID(ST_MakePoint(30.0595, -1.9550), 4326)::geography)

ON CONFLICT DO NOTHING;
```

-- Ambulance location (lon, lat) - WGS84

-- Using geography distances (meters). 1 km = 1000 meters.

```
WITH params AS (
SELECT ST_SetSRID(ST_MakePoint(30.0600, -1.9570), 4326)::geography AS amb_geog )
-- 1) Clinics within 1 km
SELECT c.id, c.name, ST_Distance(c.geom, p.amb_geog) AS meters
FROM clinic c, params p
WHERE ST_DWithin(c.geom, p.amb_geog, 1000) -- 1000 meters
ORDER BY meters;
-- 2) Nearest 3 clinics with distance in meters
WITH params AS (
SELECT ST_SetSRID(ST_MakePoint(30.0600, -1.9570), 4326)::geography AS amb_geog )
SELECT c.id, c.name, ST_Distance(c.geom, p.amb_geog) AS meters
FROM clinic c, params p
ORDER BY meters
LIMIT 3;
```

```
ERROR:  relation "clinic" does not exist
LINE 1: INSERT INTO clinic(name, geom) VALUES
                    ^
hospital=#
hospital=# -- Ambulance location (lon, lat) - WGS84
hospital=# -- Using geography distances (meters). 1 km = 1000 meters.
hospital=# WITH params AS (
hospital(#   SELECT ST_SetSRID(ST_MakePoint(30.0600, -1.9570), 4326)::geography AS amb_geog
hospital(# )
hospital-# -- 1) Clinics within 1 km
hospital-# SELECT c.id, c.name, ST_Distance(c.geom, p.amb_geog) AS meters
hospital-# FROM clinic c, params p
hospital-# WHERE ST_DWithin(c.geom, p.amb_geog, 1000)  -- 1000 meters
hospital-# ORDER BY meters;
ERROR:  type "geography" does not exist
LINE 2: ...ST_SetSRID(ST_MakePoint(30.0600, -1.9570), 4326)::geography ...
                                                             ^
hospital=#
hospital=# -- 2) Nearest 3 clinics with distance in meters
hospital=# WITH params AS (
hospital(#   SELECT ST_SetSRID(ST_MakePoint(30.0600, -1.9570), 4326)::geography AS amb_geog
hospital(# )
hospital-# SELECT c.id, c.name, ST_Distance(c.geom, p.amb_geog) AS meters
hospital-# FROM clinic c, params p
hospital-# ORDER BY meters
hospital-# LIMIT 3;
ERROR:  type "geography" does not exist
LINE 2: ...ST_SetSRID(ST_MakePoint(30.0600, -1.9570), 4326)::geography ...
                                                             ^
hospital=#
```

Note: *On my Windows 10 computer, the command CREATE EXTENSION IF NOT EXISTS postgis; is failing because the PostGIS extension is not installed or properly registered with my PostgreSQL 16 instance. This usually happens when the PostGIS bundle for Windows was not added through StackBuilder or the installer was for a different PostgreSQL version. As a result, PostgreSQL cannot locate the required PostGIS files (like postgis.control) in the share/extension directory, preventing the extension from being created. Reinstalling PostGIS using the correct PostGIS 3.x Bundle for PostgreSQL 16 typically resolves the issue.*

Conclusion

This assignment demonstrates how Intelligent Databases extend traditional database capabilities through embedded logic, automated reasoning, and adaptive behavior. Using PostgreSQL 16 and PostGIS, each task applied intelligence principles within a relational framework—ensuring data validity, consistency, and insight generation. Declarative constraints and triggers automated integrity enforcement and auditing, recursive queries enabled hierarchical reasoning, and ontological modeling supported semantic inferences. The integration of spatial logic via PostGIS

further illustrated real-world decision support, such as proximity-based clinic recommendations. Together, these components reflect the evolution of databases from passive data stores to active, knowledge-driven systems capable of enforcing rules, deducing relationships, and responding dynamically to context. Intelligent Databases thus bridge the gap between data management and artificial intelligence, paving the way for more autonomous, reliable, and insightful information systems.