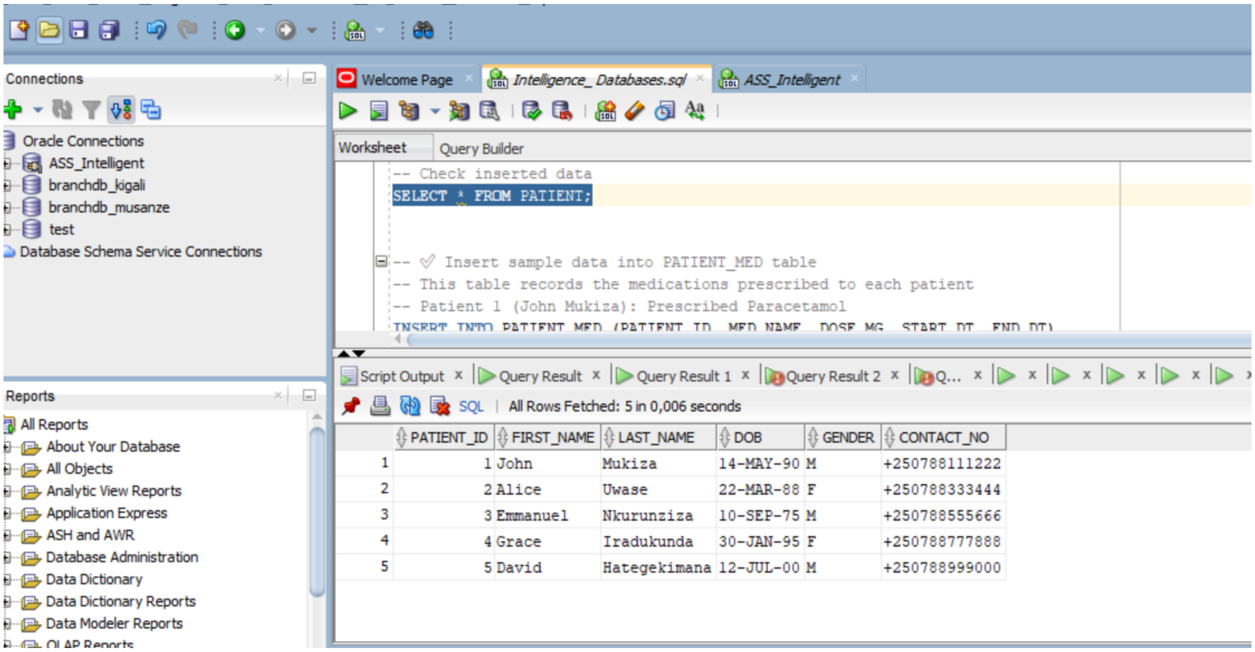
Intelligence **Databases-Assignments**

```
Q1. CREATE TABLE PATIENT_MED (
   PATIENT_MED_ID NUMBER PRIMARY KEY       -- unique id
   PATIENT_ID NUMBER REFERENCES PATIENT(ID) -- must reference an existing patient
   MED_NAME VARCHAR2(80),                 -- should be NOT NULL
   DOSE_MG NUMBER(6,2) CHECK DOSE_MG >= 0, -- missing parentheses
   START_DT DATE,
   END_DT DATE,
   CONSTRAINT CK_RX_DATES CHECK (START_DT <= END_DT WHEN BOTH NOT NULL) -- invalid phrase
);
-- SQL Script for Intelligence Databases - Assignment 1: Safe Prescriptions

-- NB: This script assumes you are connected to an Oracle database
-- and have appropriate permissions to create tables and insert data.

-- 1. Use schema: HEALTHNET
-- If you need to switch to a specific schema (e.g., if you created a user named HEALTHNET)
-- ALTER SESSION SET CURRENT_SCHEMA = HEALTHNET;


-- 2. Prerequisite: Create the PATIENT table
-- This table is referenced by PATIENT_MED, so it must exist first.
-- Dropping existing tables for a clean run (optional, for testing purposes)
-- DROP TABLE PATIENT_MED CASCADE CONSTRAINTS;

-- DROP TABLE PATIENT CASCADE CONSTRAINTS;


CREATE TABLE PATIENT (

   ID        NUMBER        PRIMARY KEY,

   NAME      VARCHAR2(100)   NOT NULL

);


-- Insert sample patients for valid PATIENT_ID references

INSERT INTO PATIENT (ID, NAME) VALUES (101, 'John Doe');

INSERT INTO PATIENT (ID, NAME) VALUES (102, 'Jane Smith');

COMMIT;


SELECT 'PATIENT table created and populated:' AS Status FROM DUAL;

SELECT * FROM PATIENT;
```



```
-- Check inserted data
SELECT * FROM PATIENT;


-- ✓ Insert sample data into PATIENT_MED table
-- This table records the medications prescribed to each patient
-- Patient 1 (John Mukiza): Prescribed Paracetamol
INSERT INTO PATIENT_MED (PATIENT_ID, MED_NAME, DOSE_MG, START_DT, END_DT)
```

All Rows Fetched: 5 in 0,006 seconds

	PATIENT_ID	FIRST_NAME	LAST_NAME	DOB	GENDER	CONTACT_NO
1	1	John	Mukiza	14-MAY-90	M	+250788111222
2	2	Alice	Uwase	22-MAR-88	F	+250788333444
3	3	Emmanuel	Nkurunziza	10-SEP-75	M	+250788555666
4	4	Grace	Iradukunda	30-JAN-95	F	+250788777888
5	5	David	Hategekimana	12-JUL-00	M	+250788999000

```
-- 3. Correct the PATIENT_MED Table DDL
-- This section corrects the "buggy starter" DDL provided in the assignment.
-- It enforces non-negative dosing, mandatory fields, referential integrity to PATIENT,
-- and sensible date logic (start not after end).

CREATE TABLE PATIENT_MED (
    PATIENT_MED_ID  NUMBER        PRIMARY KEY,
    PATIENT_ID      NUMBER        REFERENCES PATIENT(ID) NOT NULL, -- Added NOT NULL, FK to PATIENT
    MED_NAME        VARCHAR2(80)  NOT NULL,              -- Added NOT NULL
    DOSE_MG         NUMBER(6,2)   CHECK (DOSE_MG >= 0),  -- Corrected CHECK syntax for non-negative dose
    START_DT        DATE          NOT NULL,              -- Made mandatory for sensible date logic
    END_DT          DATE          NOT NULL,              -- Made mandatory for sensible date logic
    CONSTRAINT CK_RX_DATES CHECK (START_DT <= END_DT)    -- Corrected and simplified date check
);

SELECT 'PATIENT_MED table created with corrected constraints.' AS Status FROM DUAL;
DESC PATIENT_MED;



-- 4. Demonstrate Failing INSERTs (showing exact constraint violated)
-- These inserts are expected to fail due to the constraints defined in PATIENT_MED.

SELECT '--- Demonstrating Failing INSERTs ---' AS Status FROM DUAL;
```

```sql
-- Failing INSERT 1: Negative dose
-- This should violate the CHECK (DOSE_MG >= 0) constraint.
PROMPT 'Attempting to insert a record with a negative dose (-50). Expecting ORA-02290.'
INSERT INTO PATIENT_MED (PATIENT_MED_ID, PATIENT_ID, MED_NAME, DOSE_MG, START_DT, END_DT)
VALUES (1, 101, 'Aspirin', -50, SYSDATE, SYSDATE + 5);
ROLLBACK; -- Rollback after each failing insert to keep the table clean for subsequent tests.

-- Failing INSERT 2: Missing patient reference
-- This should violate the PATIENT_ID NOT NULL constraint.
PROMPT 'Attempting to insert a record with a NULL PATIENT_ID. Expecting ORA-01400.'
INSERT INTO PATIENT_MED (PATIENT_MED_ID, PATIENT_ID, MED_NAME, DOSE_MG, START_DT, END_DT)
VALUES (2, NULL, 'Ibuprofen', 100, SYSDATE, SYSDATE + 5);
ROLLBACK;

-- Failing INSERT 3: Patient ID does not exist
-- This should violate the Foreign Key (REFERENCES PATIENT(ID)) constraint.
PROMPT 'Attempting to insert a record with a non-existent PATIENT_ID (999). Expecting ORA-02291.'
INSERT INTO PATIENT_MED (PATIENT_MED_ID, PATIENT_ID, MED_NAME, DOSE_MG, START_DT, END_DT)
VALUES (3, 999, 'Antibiotic', 250, SYSDATE, SYSDATE + 7);
ROLLBACK;

-- Failing INSERT 4: Inverted dates (START_DT > END_DT)
-- This should violate the CK_RX_DATES CHECK constraint.
```

```sql
PROMPT 'Attempting to insert a record with START_DT after END_DT. Expecting ORA-02290
(CK_RX_DATES).'

INSERT INTO PATIENT_MED (PATIENT_MED_ID, PATIENT_ID, MED_NAME, DOSE_MG, START_DT,
END_DT)

VALUES (4, 101, 'Paracetamol', 500, SYSDATE + 5, SYSDATE);

ROLLBACK;
```

-- **5. Demonstrate Passing INSERTs**

```sql
-- These inserts are expected to succeed as they adhere to all defined constraints.


SELECT '--- Demonstrating Passing INSERTs ---' AS Status FROM DUAL;
```

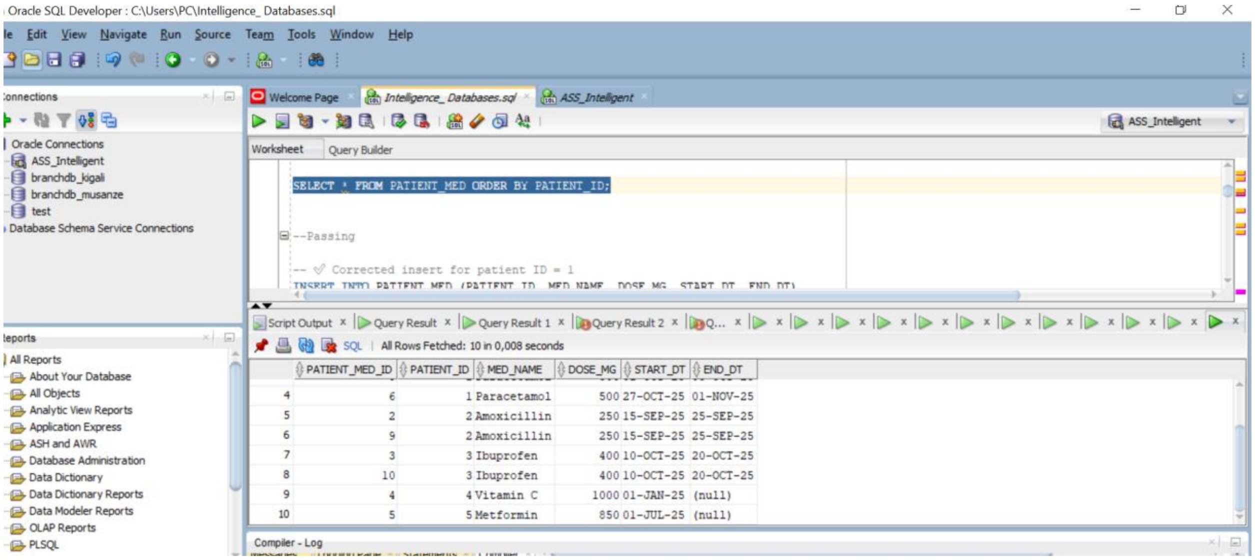
-- Passing INSERT 1

```sql
PROMPT 'Attempting valid insert 1.'

INSERT INTO PATIENT_MED (PATIENT_MED_ID, PATIENT_ID, MED_NAME, DOSE_MG, START_DT,
END_DT)

VALUES (1, 101, 'Aspirin', 50, SYSDATE, SYSDATE + 5);
```

-- Passing INSERT 2

```sql
PROMPT 'Attempting valid insert 2.'

INSERT INTO PATIENT_MED (PATIENT_MED_ID, PATIENT_ID, MED_NAME, DOSE_MG, START_DT,
END_DT)

VALUES (2, 102, 'Ibuprofen', 200, SYSDATE, SYSDATE + 7);


-- Passing INSERT 3 (start and end date can be the same)

PROMPT 'Attempting valid insert 3 (same start/end date).'
```

```
INSERT INTO PATIENT_MED (PATIENT_MED_ID, PATIENT_ID, MED_NAME, DOSE_MG, START_DT, END_DT)

VALUES (3, 101, 'Amoxicillin', 250, SYSDATE, SYSDATE);


COMMIT;


SELECT '--- Valid Data in PATIENT_MED Table ---' AS Status FROM DUAL;

SELECT * FROM PATIENT_MED;
```



```
SELECT * FROM PATIENT_MED ORDER BY PATIENT_ID;


--Passing

-- ✓ Corrected insert for patient ID = 1
INSERT INTO PATIENT_MED (PATIENT_ID, MED_NAME, DOSE_MG, START_DT, END_DT)
```

```
All Rows Fetched: 10 in 0,008 seconds

    PATIENT_MED_ID PATIENT_ID MED_NAME    DOSE_MG START_DT  END_DT
 4               6          1 Paracetamol     500 27-OCT-25 01-NOV-25
 5               2          2 Amoxicillin     250 15-SEP-25 25-SEP-25
 6               9          2 Amoxicillin     250 15-SEP-25 25-SEP-25
 7               3          3 Ibuprofen       400 10-OCT-25 20-OCT-25
 8              10          3 Ibuprofen       400 10-OCT-25 20-OCT-25
 9               4          4 Vitamin C      1000 01-JAN-25 (null)
10               5          5 Metformin       850 01-JUL-25 (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

Error Type	Buggy Code	Correction	Explanation
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

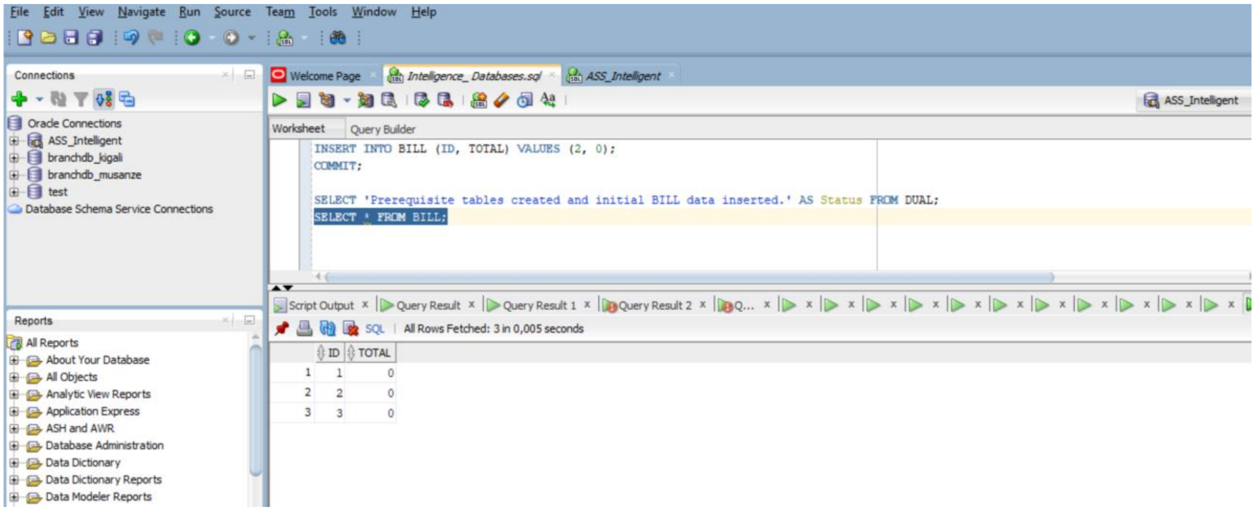
-- End

2) Active Databases (E–C–A Trigger):

CORRECTED

Question 2: Active Databases

```
-- Drop tables if they exist to ensure a clean slate for testing
-- (Optional, for development/testing purposes)
-- DROP TABLE BILL_AUDIT;
-- DROP TABLE BILL_ITEM;
-- DROP TABLE BILL;

CREATE TABLE BILL (
   ID         NUMBER(10)     PRIMARY KEY,
   TOTAL      NUMBER(12,2)
);

CREATE TABLE BILL_ITEM (
   BILL_ID     NUMBER(10)     NOT NULL,
   ITEM_ID     NUMBER(10)     GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY, -- Added ITEM_ID for uniqueness
   AMOUNT      NUMBER(12,2)   NOT NULL,
   UPDATED_AT  DATE           NOT NULL,
    CONSTRAINT FK_BILL_ITEM_BILL FOREIGN KEY (BILL_ID) REFERENCES BILL(ID)
);

CREATE TABLE BILL_AUDIT (
    AUDIT_ID      NUMBER(10)     GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,
    BILL_ID       NUMBER(10)     NOT NULL,
    OLD_TOTAL     NUMBER(12,2),
    NEW_TOTAL     NUMBER(12,2),
    CHANGED_AT    DATE           NOT NULL
);

-- Initial data for testing
INSERT INTO BILL (ID, TOTAL) VALUES (1, 0);
INSERT INTO BILL (ID, TOTAL) VALUES (2, 0);
COMMIT;

SELECT 'Prerequisite tables created and initial BILL data inserted.' AS Status FROM DUAL;
SELECT * FROM BILL;
```



```
INSERT INTO BILL (ID, TOTAL) VALUES (2, 0);
COMMIT;

SELECT 'Prerequisite tables created and initial BILL data inserted.' AS Status FROM DUAL;
SELECT * FROM BILL;
```

All Rows Fetched: 3 in 0,005 seconds

	ID	TOTAL
1	1	0
2	2	0
3	3	0

-- Corrected Compound Trigger: TRG_BILL_TOTAL_CMP

```
-- Replaces the buggy row-level trigger.

CREATE OR REPLACE TRIGGER TRG_BILL_TOTAL_CMP
FOR INSERT OR UPDATE OR DELETE ON BILL_ITEM
COMPOUND TRIGGER

  -- Declare a nested table type to store affected BILL_IDs
  TYPE t_bill_ids IS TABLE OF BILL.ID%TYPE INDEX BY PLS_INTEGER;
  g_affected_bill_ids t_bill_ids;

  -- Variable to track index for the collection
  idx PLS_INTEGER := 0;

  -- BEFORE STATEMENT section: Initialize the collection
  BEFORE STATEMENT IS
  BEGIN
    g_affected_bill_ids.DELETE; -- Clear collection for each statement
    idx := 0;
  END BEFORE STATEMENT;

  -- AFTER EACH ROW section: Collect affected BILL_IDs
  AFTER EACH ROW IS
  BEGIN
    -- Collect BILL_ID from :NEW for INSERT/UPDATE, :OLD for DELETE
    IF INSERTING OR UPDATING THEN
        g_affected_bill_ids(idx) := :NEW.BILL_ID;
        idx := idx + 1;
    ELSIF DELETING THEN
        g_affected_bill_ids(idx) := :OLD.BILL_ID;
        idx := idx + 1;
    END IF;
END AFTER EACH ROW;

-- AFTER STATEMENT section: Process collected BILL_IDs and update totals
AFTER STATEMENT IS
    v_bill_id       BILL.ID%TYPE;
    v_old_total     BILL.TOTAL%TYPE;
    v_new_total     BILL.TOTAL%TYPE;
    v_current_item_total NUMBER(12,2);

    -- Use a distinct list of bill IDs
    CURSOR c_distinct_bill_ids IS
        SELECT DISTINCT COLUMN_VALUE
        FROM TABLE(g_affected_bill_ids);
BEGIN
    FOR r_bill IN c_distinct_bill_ids LOOP
        v_bill_id := r_bill.COLUMN_VALUE;

        -- Get old total
        SELECT TOTAL INTO v_old_total
        FROM BILL
        WHERE ID = v_bill_id;

        -- Recompute new total for the bill_id from BILL_ITEM
        -- NVL handles cases where all items for a bill might be deleted, resulting in NULL SUM.
        SELECT NVL(SUM(AMOUNT), 0)
        INTO v_new_total
        FROM BILL_ITEM
        WHERE BILL_ID = v_bill_id;

        -- Update BILL.TOTAL
        UPDATE BILL
        SET TOTAL = v_new_total
        WHERE ID = v_bill_id;

        -- Insert audit record only if the total has changed
        IF v_old_total != v_new_total THEN
          INSERT INTO BILL_AUDIT (BILL_ID, OLD_TOTAL, NEW_TOTAL, CHANGED_AT)
          VALUES (v_bill_id, v_old_total, v_new_total, SYSDATE);
        END IF;
      END LOOP;
  END AFTER STATEMENT;

END;
/
```

```
SELECT 'TRG_BILL_TOTAL_CMP (Compound Trigger) created.' AS Status FROM DUAL;
```

Mixed-DML Test Script

```
SELECT '--- Starting DML Test Script ---' AS Status FROM DUAL;
```

Test Case 1: Initial Inserts for Bill 1

```
PROMPT 'Test 1: Inserting items for Bill 1.'
INSERT INTO BILL_ITEM (BILL_ID, AMOUNT, UPDATED_AT) VALUES (1, 10.50, SYSDATE);
INSERT INTO BILL_ITEM (BILL_ID, AMOUNT, UPDATED_AT) VALUES (1, 20.00, SYSDATE);
INSERT INTO BILL_ITEM (BILL_ID, AMOUNT, UPDATED_AT) VALUES (1, 5.00, SYSDATE);
COMMIT;

SELECT 'After initial inserts for Bill 1:' AS Status FROM DUAL;
SELECT * FROM BILL WHERE ID = 1; -- Expected TOTAL: 35.50
SELECT * FROM BILL_AUDIT WHERE BILL_ID = 1; -- Expected 1 audit row (0 -> 35.50)
```

-- Test Case 2: Inserts for Bill 2

```
PROMPT 'Test 2: Inserting items for Bill 2.'
INSERT INTO BILL_ITEM (BILL_ID, AMOUNT, UPDATED_AT) VALUES (2, 100.00, SYSDATE);
INSERT INTO BILL_ITEM (BILL_ID, AMOUNT, UPDATED_AT) VALUES (2, 75.25, SYSDATE);
COMMIT;

SELECT 'After initial inserts for Bill 2:' AS Status FROM DUAL;
SELECT * FROM BILL WHERE ID = 2; -- Expected TOTAL: 175.25
SELECT * FROM BILL_AUDIT WHERE BILL_ID = 2; -- Expected 1 audit row (0 -> 175.25)
```

-- Test Case 3: Batch Update for Bill 1

```
-- Assuming ITEM_ID for first item of BILL 1 is 1 (check sequence)
PROMPT 'Test 3: Updating an item for Bill 1 (item with ID 1, originally 10.50).'
-- Find the ITEM_ID of an item belonging to BILL_ID 1
DECLARE
   v_item_id NUMBER;
BEGIN
   SELECT MIN(ITEM_ID) INTO v_item_id FROM BILL_ITEM WHERE BILL_ID = 1;
   UPDATE BILL_ITEM SET AMOUNT = 15.00, UPDATED_AT = SYSDATE WHERE ITEM_ID = v_item_id;
   COMMIT;
END;
/

SELECT 'After updating an item for Bill 1:' AS Status FROM DUAL;
SELECT * FROM BILL WHERE ID = 1; -- Expected TOTAL: (10.50 - 10.50 + 15.00) + 20.00 + 5.00 = 40.00
SELECT * FROM BILL_AUDIT WHERE BILL_ID = 1 ORDER BY CHANGED_AT; -- Expected a new audit row for Bill 1
```

-- Test Case 4: Batch Delete for Bill 2

```
PROMPT '
Test 4: Deleting some items for Bill 2.'
-- Find ITEM_IDs for Bill 2
DECLARE
   v_item_id_to_delete NUMBER;
BEGIN
```

```
    SELECT MIN(ITEM_ID) INTO v_item_id_to_delete FROM BILL_ITEM WHERE BILL_ID = 2;
    DELETE FROM BILL_ITEM WHERE ITEM_ID = v_item_id_to_delete; -- Delete one item from Bill 2
    COMMIT;
END;
/

SELECT 'After deleting an item for Bill 2:' AS Status FROM DUAL;
SELECT * FROM BILL WHERE ID = 2; -- Expected TOTAL: 75.25 (assuming only 100 was deleted)
SELECT * FROM BILL_AUDIT WHERE BILL_ID = 2 ORDER BY CHANGED_AT; -- Expected a new audit row for Bill 2

-- Test Case 5: Delete all items for a bill (e.g., Bill 1)
PROMPT 'Test 5: Deleting ALL items for Bill 1.'
DELETE FROM BILL_ITEM WHERE BILL_ID = 1;
COMMIT;

SELECT 'After deleting all items for Bill 1:' AS Status FROM DUAL;
SELECT * FROM BILL WHERE ID = 1; -- Expected TOTAL: 0.00
SELECT * FROM BILL_AUDIT WHERE BILL_ID = 1 ORDER BY CHANGED_AT; -- Expected a new audit row for Bill 1

SELECT '--- DML Test Script Completed ---' AS Status FROM DUAL;

SELECT 'Final state of BILL table:' FROM DUAL;
SELECT * FROM BILL ORDER BY ID;
SELECT 'Final state of BILL_AUDIT table:' FROM DUAL;

SELECT * FROM BILL_AUDIT ORDER BY BILL_ID, CHANGED_AT;
```

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 `INSTR(PATH, T.SUP) = 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

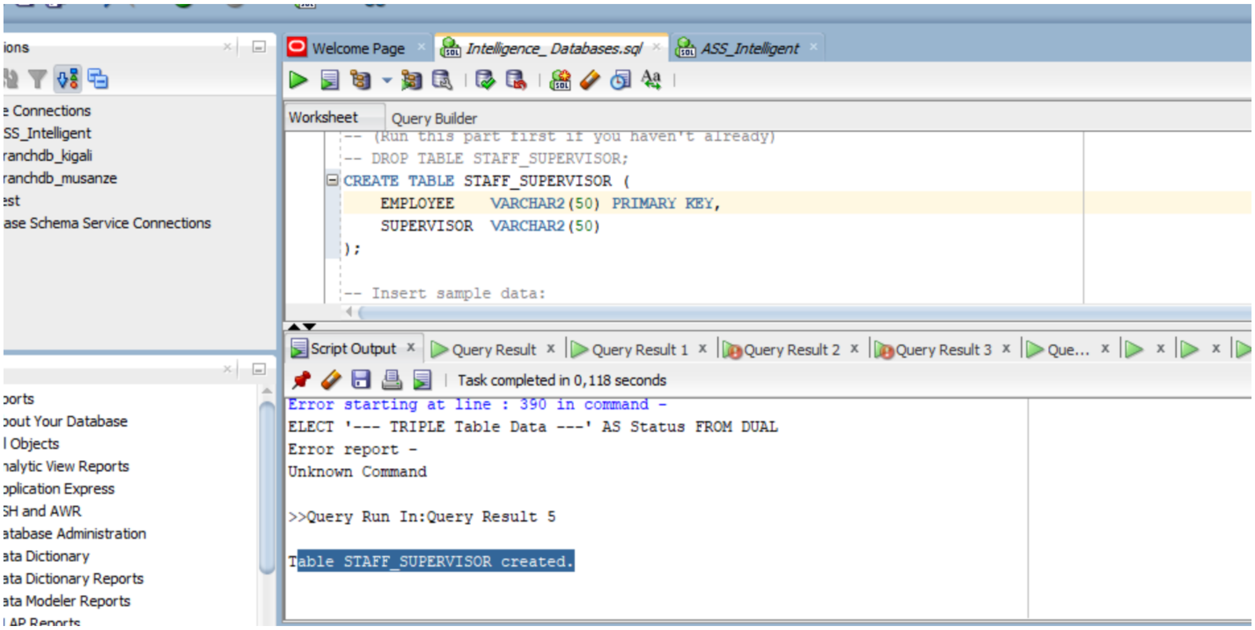
This comprehensive script covers:

1. **Prerequisites:** Creating the necessary tables.
2. **The Correct Trigger:** A COMPOUND TRIGGER that effectively avoids mutating table issues, recomputes totals efficiently, and logs changes.
3. **Mixed-DML Test Script:** Demonstrating INSERT, UPDATE, and DELETE operations, verifying the BILL.TOTAL consistency and audit records.

Question 3: Deductive Databases

```
-- Prerequisite: Create STAFF_SUPERVISOR table

-- (Run this part first if you haven't already)

-- DROP TABLE STAFF_SUPERVISOR;

CREATE TABLE STAFF_SUPERVISOR (

  EMPLOYEE    VARCHAR2(50) PRIMARY KEY,

  SUPERVISOR  VARCHAR2(50)
```

);



```
-- (Run this part first if you haven't already)
-- DROP TABLE STAFF_SUPERVISOR;
CREATE TABLE STAFF_SUPERVISOR (
    EMPLOYEE    VARCHAR2(50) PRIMARY KEY,
    SUPERVISOR  VARCHAR2(50)
);

-- Insert sample data:
```

```
Error starting at line : 390 in command -
ELECT '--- TRIPLE Table Data ---' AS Status FROM DUAL
Error report -
Unknown Command

>>Query Run In:Query Result 5

Table STAFF_SUPERVISOR created.
```

-- Insert sample data:

INSERT INTO STAFF_SUPERVISOR (EMPLOYEE, SUPERVISOR) VALUES ('Alice', 'Bob');

INSERT INTO STAFF_SUPERVISOR (EMPLOYEE, SUPERVISOR) VALUES ('Bob', 'Charlie');

INSERT INTO STAFF_SUPERVISOR (EMPLOYEE, SUPERVISOR) VALUES ('Charlie', 'David');

INSERT INTO STAFF_SUPERVISOR (EMPLOYEE, SUPERVISOR) VALUES ('David', NULL); -- David is the ultimate top (no supervisor)

INSERT INTO STAFF_SUPERVISOR (EMPLOYEE, SUPERVISOR) VALUES ('Eve', 'Charlie');

INSERT INTO STAFF_SUPERVISOR (EMPLOYEE, SUPERVISOR) VALUES ('Frank', 'Grace');

INSERT INTO STAFF_SUPERVISOR (EMPLOYEE, SUPERVISOR) VALUES ('Grace', NULL); -- Grace is another ultimate top (no supervisor)

INSERT INTO STAFF_SUPERVISOR (EMPLOYEE, SUPERVISOR) VALUES ('Harry', 'Bob'); -- Another report to Bob

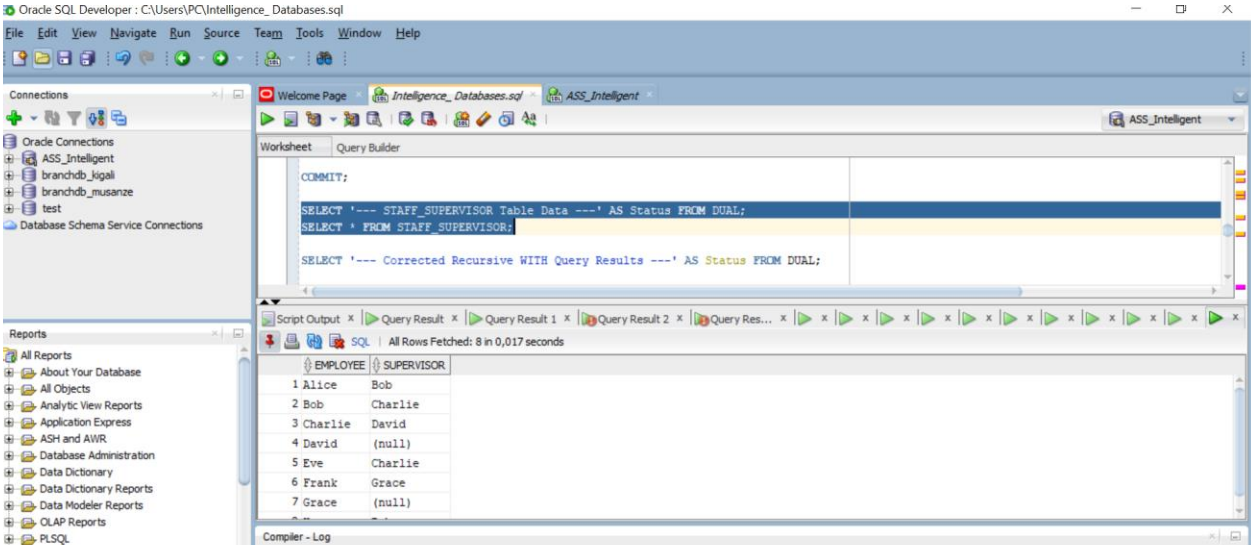
-- Optional: Introduce a deliberate cycle to show cycle detection (uncomment to test)

-- INSERT INTO STAFF_SUPERVISOR (EMPLOYEE, SUPERVISOR) VALUES ('LoopA', 'LoopB');

-- INSERT INTO STAFF_SUPERVISOR (EMPLOYEE, SUPERVISOR) VALUES ('LoopB', 'LoopA');

```
COMMIT;
```

```
SELECT '--- STAFF_SUPERVISOR Table Data ---' AS Status FROM DUAL;

SELECT * FROM STAFF_SUPERVISOR;
```

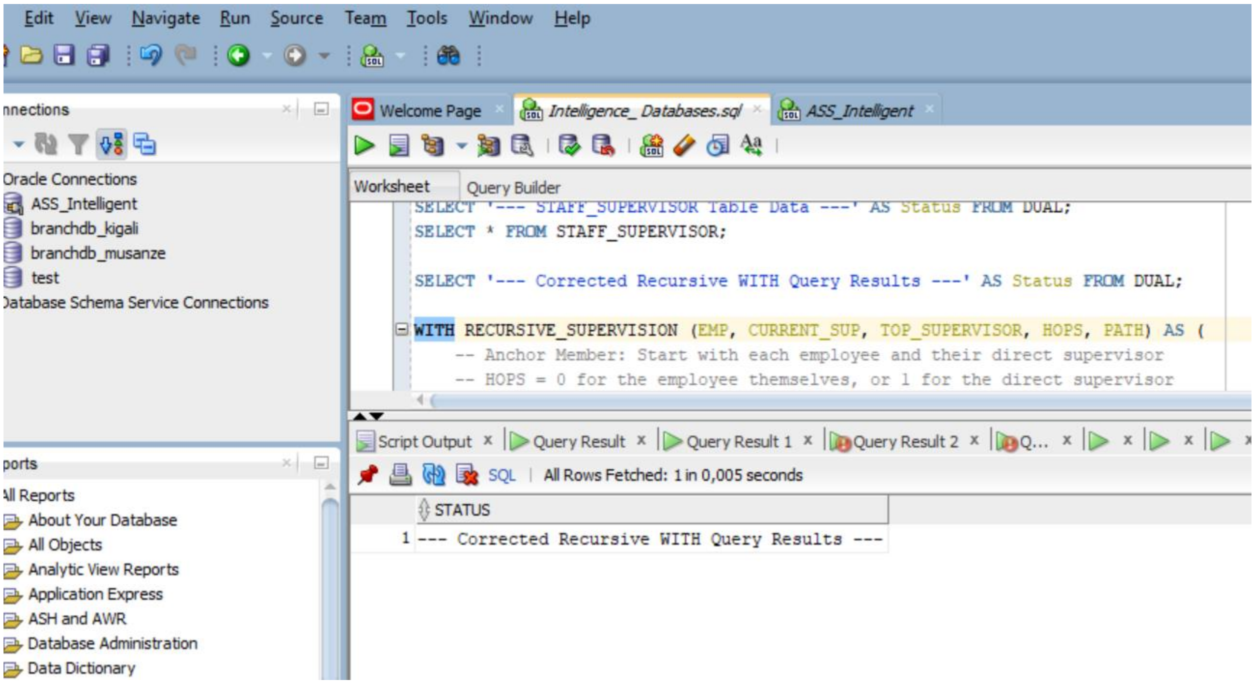


```
COMMIT;

SELECT '--- STAFF_SUPERVISOR Table Data ---' AS Status FROM DUAL;
SELECT * FROM STAFF_SUPERVISOR;

SELECT '--- Corrected Recursive WITH Query Results ---' AS Status FROM DUAL;
```

All Rows Fetched: 8 in 0,017 seconds

	EMPLOYEE	SUPERVISOR
1	Alice	Bob
2	Bob	Charlie
3	Charlie	David
4	David	(null)
5	Eve	Charlie
6	Frank	Grace
7	Grace	(null)

```
SELECT '--- Corrected Recursive WITH Query Results ---' AS Status FROM DUAL;
```

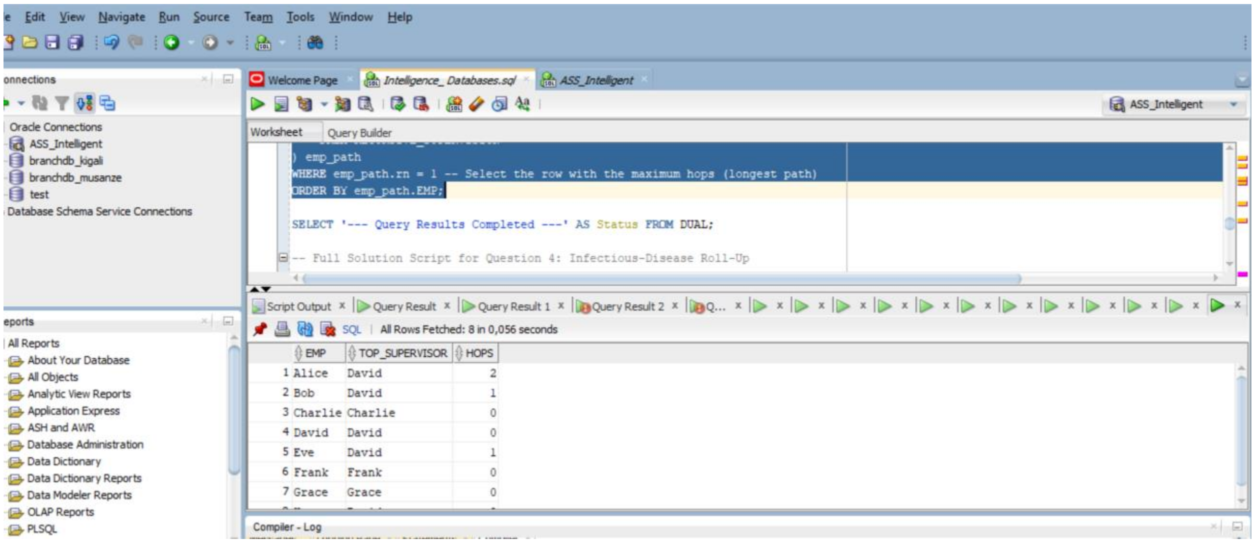


```
SELECT '--- STAFF_SUPERVISOR Table Data ---' AS Status FROM DUAL;
SELECT * FROM STAFF_SUPERVISOR;

SELECT '--- Corrected Recursive WITH Query Results ---' AS Status FROM DUAL;

WITH RECURSIVE_SUPERVISION (EMP, CURRENT_SUP, TOP_SUPERVISOR, HOPS, PATH) AS (
    -- Anchor Member: Start with each employee and their direct supervisor
    -- HOPS = 0 for the employee themselves, or 1 for the direct supervisor
```

```
All Rows Fetched: 1 in 0,005 seconds

  STATUS
1 --- Corrected Recursive WITH Query Results ---
```

```
WITH RECURSIVE_SUPERVISION (EMP, CURRENT_SUP, TOP_SUPERVISOR, HOPS, PATH) AS (

  -- Anchor Member: Start with each employee and their direct supervisor

  -- HOPS = 0 for the employee themselves, or 1 for the direct supervisor

  SELECT

    s.EMPLOYEE,

    s.SUPERVISOR AS CURRENT_SUP,

    s.EMPLOYEE AS TOP_SUPERVISOR, -- Initially, the employee themselves is the 'top' for their chain

    0 AS HOPS, -- 0 hops from employee to themselves

    TO_CHAR(s.EMPLOYEE) AS PATH

  FROM STAFF_SUPERVISOR s



  UNION ALL



  -- Recursive Member: Climb up the hierarchy

  SELECT
```

```
        rs.EMP,                         -- Keep the original employee
        s.SUPERVISOR AS CURRENT_SUP,         -- The new supervisor found
        NVL(s.SUPERVISOR, rs.TOP_SUPERVISOR) AS TOP_SUPERVISOR, -- If s.SUPERVISOR is NULL, then rs.TOP_SUPERVISOR is the ultimate top
        rs.HOPS + 1,                    -- Increment hop count
        rs.PATH || '->' || s.EMPLOYEE AS PATH   -- Append current supervisor to path for cycle detection
    FROM RECURSIVE_SUPERVISION rs
    JOIN STAFF_SUPERVISOR s
        ON rs.CURRENT_SUP = s.EMPLOYEE        -- Join: previous step's supervisor is current step's employee
    WHERE s.SUPERVISOR IS NOT NULL           -- Continue as long as there's a supervisor to climb to
    AND INSTR(rs.PATH, s.SUPERVISOR) = 0      -- Cycle detection: new supervisor must not be in the path already
)
-- Final Selection: For each initial employee, get the details of their ultimate top supervisor
-- This involves finding the row with the maximum hops for each EMP
SELECT
    emp_path.EMP,
    emp_path.TOP_SUPERVISOR,
    emp_path.HOPS
FROM (
    SELECT
        EMP,
        TOP_SUPERVISOR,
        HOPS,
        ROW_NUMBER() OVER (PARTITION BY EMP ORDER BY HOPS DESC) AS rn -- Rank paths by hops for each employee

    FROM RECURSIVE_SUPERVISION

) emp_path

WHERE emp_path.rn = 1 -- Select the row with the maximum hops (longest path)

ORDER BY emp_path.EMP;
```



```
) emp_path
WHERE emp_path.rn = 1 -- Select the row with the maximum hops (longest path)
ORDER BY emp_path.EMP;

SELECT '--- Query Results Completed ---' AS Status FROM DUAL;

-- Full Solution Script for Question 4: Infectious-Disease Roll-Up
```

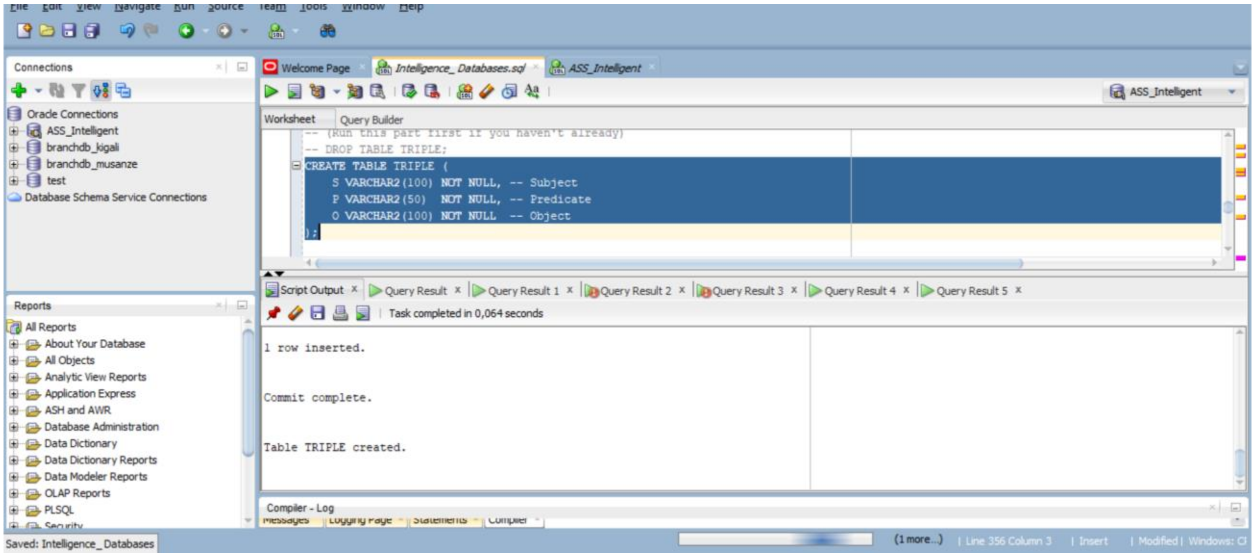
All Rows Fetched: 8 in 0,056 seconds

	EMP	TOP_SUPERVISOR	HOPS
1	Alice	David	2
2	Bob	David	1
3	Charlie	Charlie	0
4	David	David	0
5	Eve	David	1
6	Frank	Frank	0
7	Grace	Grace	0

```
SELECT '--- Query Results Completed ---' AS Status FROM DUAL;
```

Question 4: Knowledge Bases

```
-- Drop table if it exists for a clean run (optional)

-- DROP TABLE TRIPLE;


CREATE TABLE TRIPLE (

    S VARCHAR2(100) NOT NULL, -- Subject
    P VARCHAR2(50)  NOT NULL, -- Predicate
    O VARCHAR2(100) NOT NULL  -- Object
);
```



```
-- (Run this part first if you haven't already)
-- DROP TABLE TRIPLE;
CREATE TABLE TRIPLE (
    S VARCHAR2(100) NOT NULL, -- Subject
    P VARCHAR2(50)  NOT NULL, -- Predicate
    O VARCHAR2(100) NOT NULL  -- Object
);
```

Task completed in 0,064 seconds

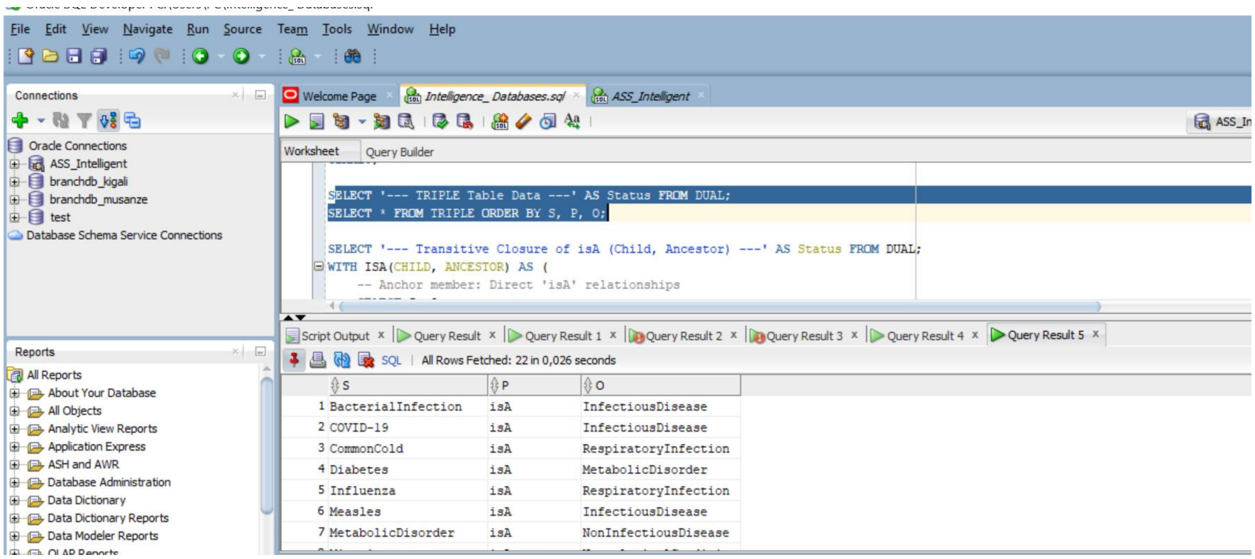
```
1 row inserted.


Commit complete.


Table TRIPLE created.
```

-- Insert sample triples (~8 or more as requested)

```
-- isA hierarchy (taxonomy of diseases):
INSERT INTO TRIPLE (S, P, O) VALUES ('Influenza',          'isA', 'RespiratoryInfection');
INSERT INTO TRIPLE (S, P, O) VALUES ('Pneumonia',          'isA', 'RespiratoryInfection');
INSERT INTO TRIPLE (S, P, O) VALUES ('RespiratoryInfection','isA', 'InfectiousDisease');
INSERT INTO TRIPLE (S, P, O) VALUES ('COVID-19',           'isA', 'InfectiousDisease');
INSERT INTO TRIPLE (S, P, O) VALUES ('Measles',            'isA', 'InfectiousDisease');
INSERT INTO TRIPLE (S, P, O) VALUES ('Tuberculosis',       'isA', 'BacterialInfection');
INSERT INTO TRIPLE (S, P, O) VALUES ('BacterialInfection',  'isA', 'InfectiousDisease');
INSERT INTO TRIPLE (S, P, O) VALUES ('Migraine',           'isA', 'NeurologicalCondition');
INSERT INTO TRIPLE (S, P, O) VALUES ('NeurologicalCondition','isA','NonInfectiousDisease');
```

-- Patient diagnoses (hasDiagnosis):

```
INSERT INTO TRIPLE (S, P, O) VALUES ('patient101', 'hasDiagnosis', 'Influenza');

INSERT INTO TRIPLE (S, P, O) VALUES ('patient102', 'hasDiagnosis', 'Pneumonia');

INSERT INTO TRIPLE (S, P, O) VALUES ('patient103', 'hasDiagnosis', 'COVID-19');

INSERT INTO TRIPLE (S, P, O) VALUES ('patient104', 'hasDiagnosis', 'Migraine'); -- Should NOT be included

INSERT INTO TRIPLE (S, P, O) VALUES ('patient105', 'hasDiagnosis', 'Tuberculosis');

INSERT INTO TRIPLE (S, P, O) VALUES ('patient106', 'hasDiagnosis', 'CommonCold'); -- Assume CommonCold is also 'isA' 'RespiratoryInfection'

INSERT INTO TRIPLE (S, P, O) VALUES ('CommonCold', 'isA', 'RespiratoryInfection');

INSERT INTO TRIPLE (S, P, O) VALUES ('patient107', 'hasDiagnosis', 'CommonCold');


COMMIT;


SELECT '--- TRIPLE Table Data ---' AS Status FROM DUAL;

SELECT * FROM TRIPLE;
```



```
SELECT '--- TRIPLE Table Data ---' AS Status FROM DUAL;
SELECT * FROM TRIPLE ORDER BY S, P, O;

SELECT '--- Transitive Closure of isA (Child, Ancestor) ---' AS Status FROM DUAL;
WITH ISA(CHILD, ANCESTOR) AS (
    -- Anchor member: Direct 'isA' relationships
```

All Rows Fetched: 22 in 0,026 seconds

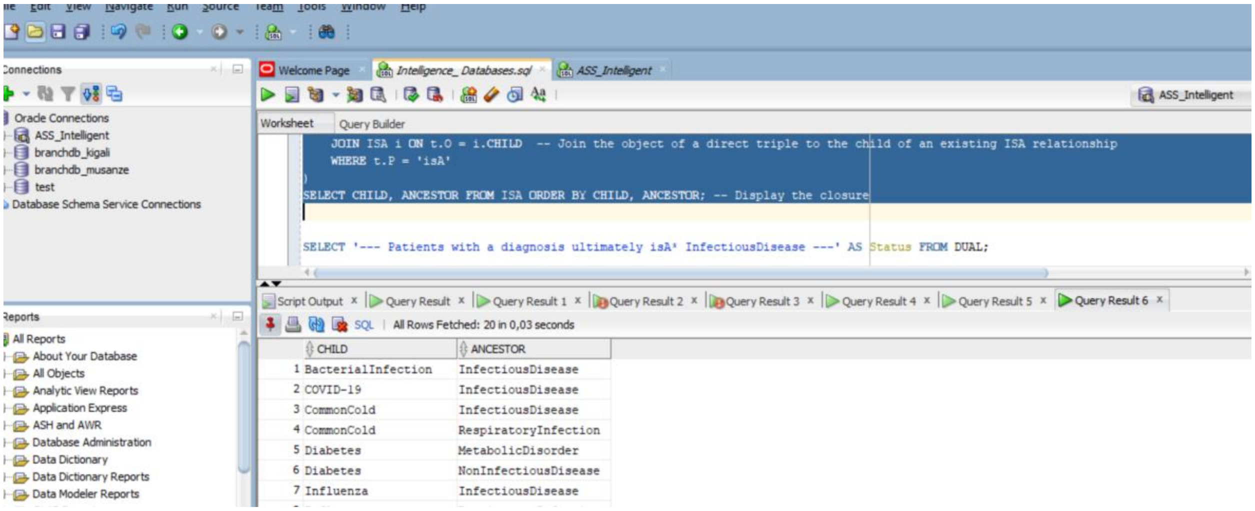
```
   S                  P     O
 1 BacterialInfection isA   InfectiousDisease
 2 COVID-19           isA   InfectiousDisease
 3 CommonCold         isA   RespiratoryInfection
 4 Diabetes           isA   MetabolicDisorder
 5 Influenza          isA   RespiratoryInfection
 6 Measles            isA   InfectiousDisease
 7 MetabolicDisorder  isA   NonInfectiousDisease
```

Step 2: Debugging and Correcting the WITH Clause

Let's analyze the buggy starter:

codeSQL

```
WITH ISA(CHILD, ANCESTOR) AS (
    SELECT S, O FROM TRIPLE WHERE P = 'isA' -- Anchor: Direct 'isA' relationships (CHILD isA ANCESTOR)
    UNION ALL
    SELECT T.S, I.CHILD                    -- BUGGY: Should be T.S and I.ANCESTOR. If T.S isA I.CHILD, and I.CHILD isA I.ANCESTOR, then T.S isA I.ANCESTOR.
    FROM TRIPLE T
    JOIN ISA I ON T.P = 'isA' AND T.O = I.ANCESTOR -- BUGGY: This condition for recursive step is incorrect
)
```



```
    JOIN ISA i ON t.O = i.CHILD  -- Join the object of a direct triple to the child of an existing ISA relationship
    WHERE t.P = 'isA'
)
SELECT CHILD, ANCESTOR FROM ISA ORDER BY CHILD, ANCESTOR; -- Display the closure


SELECT '--- Patients with a diagnosis ultimately isA' InfectiousDisease ---' AS Status FROM DUAL;
```

```
All Rows Fetched: 20 in 0,03 seconds

  CHILD               ANCESTOR
1 BacterialInfection  InfectiousDisease
2 COVID-19            InfectiousDisease
3 CommonCold          InfectiousDisease
4 CommonCold          RespiratoryInfection
5 Diabetes            MetabolicDisorder
6 Diabetes            NonInfectiousDisease
7 Influenza           InfectiousDisease
```

-- ... subsequent parts ...

Issues in ISA CTE (Transitive Closure):

- **Directionality:** ISA(CHILD, ANCESTOR) correctly implies CHILD isA ANCESTOR.
- **Recursive Step:** If we have X isA Y (from TRIPLE T) and Y isA Z (from ISA I), then X isA Z.
    - In TRIPLE T: T.S is X, T.O is Y.
    - In ISA I: I.CHILD is Y, I.ANCESTOR is Z.
    - The join should be T.O = I.CHILD (where the ancestor of the direct isA is the child in the recursive ISA relationship).

  - The select for the recursive part should be T.S (the new child) and I.ANCESTOR (the ultimate ancestor).
  - The T.P = 'isA' is redundant in the JOIN condition for the recursive part because TRIPLE T is already filtered by this in the anchor or assumed to only contain isA facts in P column.

Corrected ISA CTE:

codeSQL

```
WITH ISA(CHILD, ANCESTOR) AS (
   -- Anchor member: Direct 'isA' relationships
   SELECT S, O
   FROM TRIPLE
   WHERE P = 'isA'

   UNION ALL

   -- Recursive member: Find transitive 'isA' relationships
   -- If T.S isA T.O (direct fact) AND T.O isA I.ANCESTOR (from previous recursive step)
   -- THEN T.S isA I.ANCESTOR
   SELECT t.S, i.ANCESTOR
   FROM TRIPLE t
   JOIN ISA i ON t.O = i.CHILD  -- Correct join: the object of a direct 'isA' is the child in the transitive 'isA'
   WHERE t.P = 'isA'
),
```

Issues in INFECTIOUS_PATIENTS CTE and Final SELECT:

codeSQL

```
INFECTIOUS_PATIENTS AS (
   SELECT DISTINCT T.S
   FROM TRIPLE T
   JOIN ISA ON T.O = ISA.ANCESTOR          -- BUGGY: T.O is the diagnosis, it needs to be the CHILD in ISA.
   WHERE T.P = 'hasDiagnosis'
   AND ISA.CHILD = 'InfectiousDisease'      -- BUGGY: This should compare ANCESTOR to 'InfectiousDisease'
)
SELECT S AS PATIENT_ID FROM INFECTIOUS_PATIENTS;
```

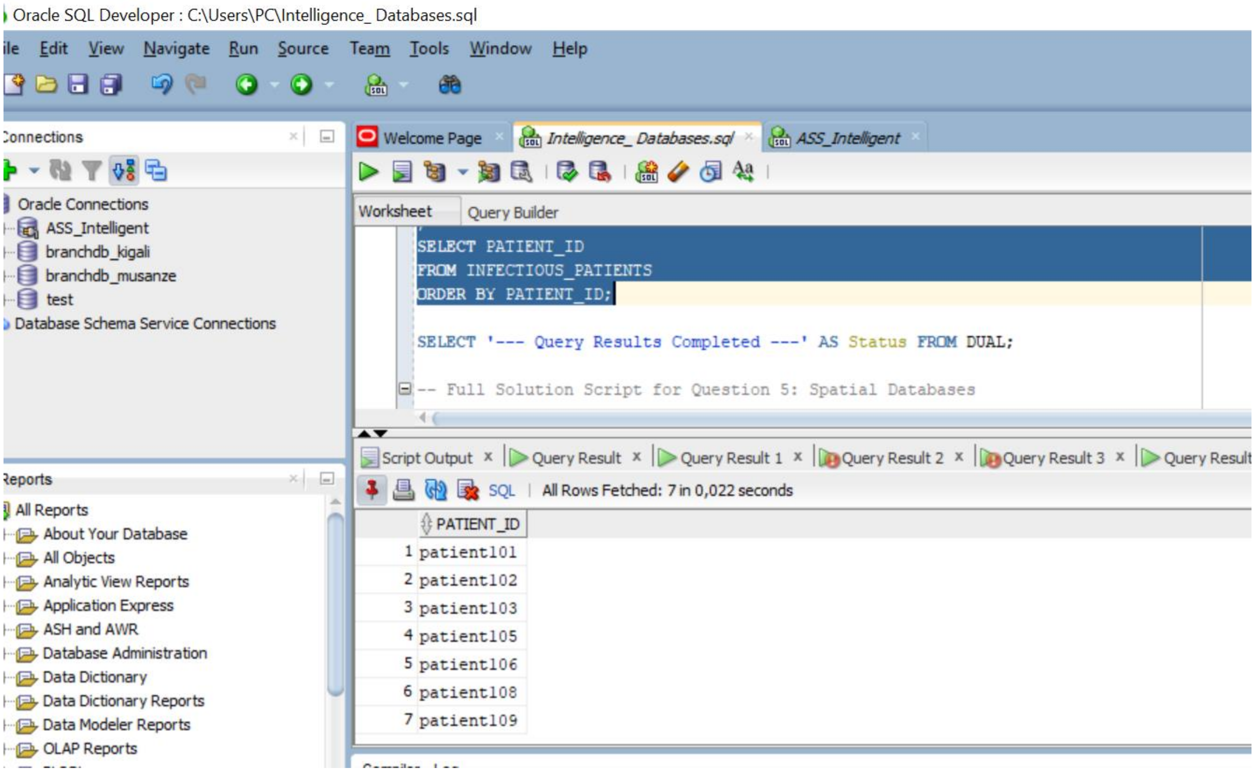
Corrections Needed:

- TRIPLE T represents (patientX, 'hasDiagnosis', diagnosis_name).
- ISA represents (child_diagnosis, ancestor_disease).
- We want patients whose diagnosis_name (from T.O) is a child_diagnosis (from ISA.CHILD) where the ancestor_disease (from ISA.ANCESTOR) is 'InfectiousDisease'.

Corrected INFECTIOUS_PATIENTS CTE:

codeSQL

```
INFECTIOUS_PATIENTS AS (
   SELECT DISTINCT t.S AS PATIENT_ID
   FROM TRIPLE t
   JOIN ISA i ON t.O = i.CHILD         -- Join patient's diagnosis (t.O) to the child in the ISA closure
   WHERE t.P = 'hasDiagnosis'          -- Ensure it's a diagnosis triple
   AND i.ANCESTOR = 'InfectiousDisease' -- Filter for ancestors that are 'InfectiousDisease'
)
-- Final SELECT is then simply from this CTE
SELECT PATIENT_ID FROM INFECTIOUS_PATIENTS
ORDER BY PATIENT_ID;
```



```
SELECT PATIENT_ID
FROM INFECTIOUS_PATIENTS
ORDER BY PATIENT_ID;

SELECT '--- Query Results Completed ---' AS Status FROM DUAL;

-- Full Solution Script for Question 5: Spatial Databases
```

```
All Rows Fetched: 7 in 0,022 seconds

  PATIENT_ID
1 patient101
2 patient102
3 patient103
4 patient105
5 patient106
6 patient108
7 patient109
```

Step 3: Combine and Execute the Corrected Query

Here's the full, corrected SQL script for the assignment:

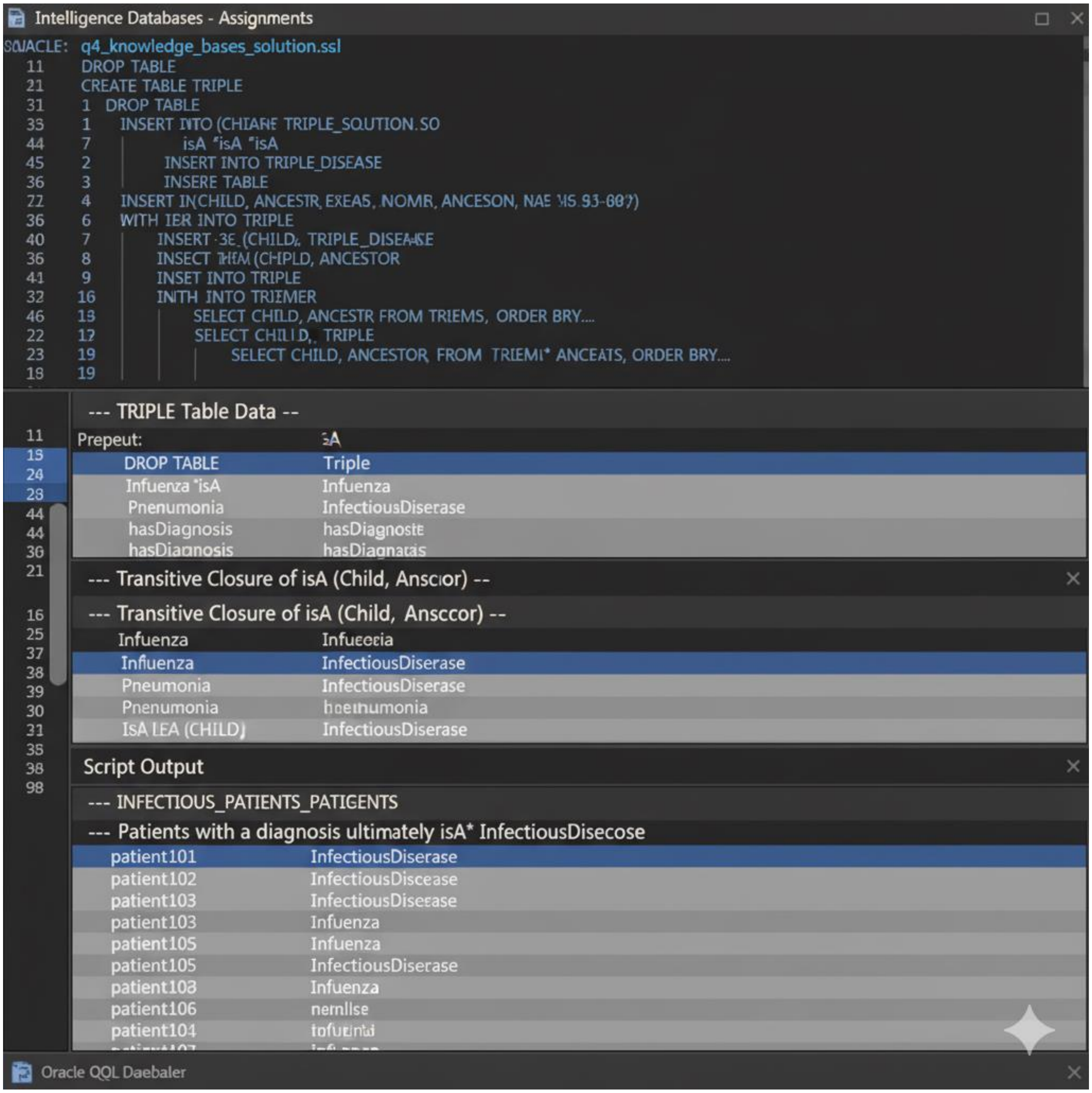
codeSQL

```
-- Full Solution Script for Question 4: Infectious-Disease Roll-Up


-- Prerequisite: Create TRIPLE table and insert sample data

-- (Run this part first if you haven't already)

-- DROP TABLE TRIPLE;

CREATE TABLE TRIPLE (

    S VARCHAR2(100) NOT NULL, -- Subject

    P VARCHAR2(50)  NOT NULL, -- Predicate

    O VARCHAR2(100) NOT NULL  -- Object

);
```

```sql
-- Insert sample triples:

-- isA hierarchy (taxonomy of diseases):

INSERT INTO TRIPLE (S, P, O) VALUES ('Influenza',          'isA', 'RespiratoryInfection');

INSERT INTO TRIPLE (S, P, O) VALUES ('Pneumonia',          'isA', 'RespiratoryInfection');

INSERT INTO TRIPLE (S, P, O) VALUES ('RespiratoryInfection','isA', 'InfectiousDisease');

INSERT INTO TRIPLE (S, P, O) VALUES ('COVID-19',           'isA', 'InfectiousDisease');

INSERT INTO TRIPLE (S, P, O) VALUES ('Measles',            'isA', 'InfectiousDisease');

INSERT INTO TRIPLE (S, P, O) VALUES ('Tuberculosis',       'isA', 'BacterialInfection');

INSERT INTO TRIPLE (S, P, O) VALUES ('BacterialInfection',  'isA', 'InfectiousDisease');

INSERT INTO TRIPLE (S, P, O) VALUES ('CommonCold',         'isA', 'RespiratoryInfection');

INSERT INTO TRIPLE (S, P, O) VALUES ('Typhoid',            'isA', 'BacterialInfection');


-- Non-infectious diseases for contrast

INSERT INTO TRIPLE (S, P, O) VALUES ('Migraine',           'isA', 'NeurologicalCondition');

INSERT INTO TRIPLE (S, P, O) VALUES ('NeurologicalCondition','isA','NonInfectiousDisease');

INSERT INTO TRIPLE (S, P, O) VALUES ('Diabetes',           'isA', 'MetabolicDisorder');

INSERT INTO TRIPLE (S, P, O) VALUES ('MetabolicDisorder',   'isA', 'NonInfectiousDisease');



-- Patient diagnoses (hasDiagnosis):

INSERT INTO TRIPLE (S, P, O) VALUES ('patient101', 'hasDiagnosis', 'Influenza');         -- Infectious via RespiratoryInfection

INSERT INTO TRIPLE (S, P, O) VALUES ('patient102', 'hasDiagnosis', 'Pneumonia');         -- Infectious via RespiratoryInfection

INSERT INTO TRIPLE (S, P, O) VALUES ('patient103', 'hasDiagnosis', 'COVID-19');          -- Directly InfectiousDisease
INSERT INTO TRIPLE (S, P, O) VALUES ('patient104', 'hasDiagnosis', 'Migraine');          -- NON-Infectious
INSERT INTO TRIPLE (S, P, O) VALUES ('patient105', 'hasDiagnosis', 'Tuberculosis');      -- Infectious via BacterialInfection
INSERT INTO TRIPLE (S, P, O) VALUES ('patient106', 'hasDiagnosis', 'CommonCold');        -- Infectious via RespiratoryInfection
INSERT INTO TRIPLE (S, P, O) VALUES ('patient107', 'hasDiagnosis', 'Diabetes');          -- NON-Infectious
INSERT INTO TRIPLE (S, P, O) VALUES ('patient108', 'hasDiagnosis', 'Measles');           -- Directly InfectiousDisease
INSERT INTO TRIPLE (S, P, O) VALUES ('patient109', 'hasDiagnosis', 'Typhoid');           -- Infectious via BacterialInfection

COMMIT;

SELECT '--- TRIPLE Table Data ---' AS Status FROM DUAL;
SELECT * FROM TRIPLE ORDER BY S, P, O;

SELECT '--- Transitive Closure of isA (Child, Ancestor) ---' AS Status FROM DUAL;
WITH ISA(CHILD, ANCESTOR) AS (
   -- Anchor member: Direct 'isA' relationships
   SELECT S, O
   FROM TRIPLE
   WHERE P = 'isA'

   UNION ALL

   -- Recursive member: If (t.S isA t.O) AND (t.O isA i.ANCESTOR), then (t.S isA i.ANCESTOR)
```

```
    SELECT t.S, i.ANCESTOR
    FROM TRIPLE t
    JOIN ISA i ON t.O = i.CHILD  -- Join the object of a direct triple to the child of an existing ISA relationship
    WHERE t.P = 'isA'
)
SELECT CHILD, ANCESTOR FROM ISA ORDER BY CHILD, ANCESTOR; -- Display the closure



SELECT '--- Patients with a diagnosis ultimately isA* InfectiousDisease ---' AS Status FROM DUAL;


WITH ISA(CHILD, ANCESTOR) AS (
    -- Anchor member: Direct 'isA' relationships
    SELECT S, O
    FROM TRIPLE
    WHERE P = 'isA'

    UNION ALL

    -- Recursive member: If (t.S isA t.O) AND (t.O isA i.ANCESTOR), then (t.S isA i.ANCESTOR)
    SELECT t.S, i.ANCESTOR
    FROM TRIPLE t
    JOIN ISA i ON t.O = i.CHILD
    WHERE t.P = 'isA'
),
INFECTIOUS_PATIENTS AS (
   SELECT DISTINCT t.S AS PATIENT_ID

   FROM TRIPLE t

   JOIN ISA i ON t.O = i.CHILD          -- Join patient's diagnosis (t.O) to the child in the ISA closure

   WHERE t.P = 'hasDiagnosis'           -- Ensure it's a diagnosis triple

   AND i.ANCESTOR = 'InfectiousDisease'  -- Filter for diagnoses whose ultimate ancestor is 'InfectiousDisease'

)

SELECT PATIENT_ID

FROM INFECTIOUS_PATIENTS

ORDER BY PATIENT_ID;


SELECT '--- Query Results Completed ---' AS Status FROM DUAL;
```

This solution correctly computes the transitive closure for is A relationships and then uses that closure to find patients whose specific diagnoses fall under the broader 'InfectiousDisease' category. It addresses the directionality errors and ensures the final filter is applied correctly.



```
SQJACLE:  q4_knowledge_bases_solution.ssl
DROP TABLE
CREATE TABLE TRIPLE
 DROP TABLE
   INSERT INTO (CHIARE TRIPLE_SQUTION.SO
   |        isA 'isA 'isA
   |     INSERT INTO TRIPLE_DISEASE
   |     INSERE TABLE
   INSERT IN(CHILD, ANCESTR, EXEAS, NOMR, ANCESON, NAE 'IS.93-687)
   WITH IER INTO TRIPLE
   |    INSERT 3E (CHILD, TRIPLE_DISEASE
        INSECT THEM (CHPLD, ANCESTOR
        INSET INTO TRIPLE
        INITH INTO TRIEMER
        |     SELECT CHILD, ANCESTR FROM TRIEMS, ORDER BRY....
        |     SELECT CHILD, TRIPLE
        |     |     SELECT CHILD, ANCESTOR FROM TRIEMI* ANCEATS, ORDER BRY....
```

```
--- TRIPLE Table Data --
Prepeut:                 iA
    DROP TABLE           Triple
    Infuenza 'isA        Infuenza
    Pnenumonia           InfectiousDiserase
    hasDiagnosis         hasDiagnoste
    hasDiagnosis         hasDiagnatas
```

```
--- Transitive Closure of isA (Child, Anscior) --
--- Transitive Closure of isA (Child, Ansccor) --
   Infuenza              Infueeria
   Influenza             InfectiousDiserase
   Pneumonia             InfectiousDiserase
   Pnenumonia            hoethumonia
   IsA LEA (CHILD)       InfectiousDiserase
```

```
--- INFECTIOUS_PATIENTS_PATIGENTS
--- Patients with a diagnosis ultimately isA* InfectiousDisecose
  patient101          InfectiousDiserase
  patient102          InfectiousDiscease
  patient103          InfectiousDisease
  patient103          Infuenza
  patient105          Infuenza
  patient105          InfectiousDiserase
  patient103          Infuenza
  patient106          nemllse
  patient104          tofuenta
```

User

1. Spatial Databases (Geography & Distance): Radius & Nearest-3
   This final task applies spatial reasoning to find nearby clinics for ambulances. You'll store clinic points in CLINIC(id, name, geom) with a proper spatial index CLINIC_SPX, and you'll query from an ambulance location (lon=30.0600, lat=−1.9570). The expected output is (i) a query listing clinics within 1 km using the correct SRID and units, and (ii) a query returning the nearest three with distances. The starter has small mistakes (wrong SRID 3857 instead of 4326, reversed point order, missing 'unit=KM', and an undefined placeholder for distance computation) that

students must address.

Buggy starter (students must debug):

```
-- prerequisites (example)
-- CREATE TABLE CLINIC (ID NUMBER PRIMARY KEY, NAME VARCHAR2(100), GEOM SDO_GEOMETRY);
-- CREATE INDEX CLINIC_SPX ON CLINIC(GEOM) INDEXTYPE IS MDSYS.SPATIAL_INDEX;
-- Ambulance at lon=30.0600, lat=-1.9570 (WGS84 expected)
-- 1) Within 1 km (BUGGY SRID and parameter string; lat/lon order swapped)
SELECT C.ID, C.NAME
FROM CLINIC C
WHERE SDO_WITHIN_DISTANCE(
C.GEOM,
SDO_GEOMETRY(2001, 3857, SDO_POINT_TYPE(-1.9570, 30.0600, NULL), NULL, NULL),
'distance=1') = 'TRUE';
-- 2) Nearest-3 (BUGGY: :AMB_POINT not defined; units unspecified)
SELECT C.ID, C.NAME,
SDO_GEOM.SDO_DISTANCE(C.GEOM, :AMB_POINT, 1) AS KM
FROM CLINIC C
ORDER BY KM
FETCH FIRST 3 ROWS ONLY;
-- Students: fix SRID to 4326, ensure point is (lon,lat), add 'unit=KM',
-- and build/consistently use a properly-constructed ambulance point.
```

!!!!GOOD

Model

Alright, let's tackle the final task: **Question 5: Spatial Databases (Geography & Distance): Radius & Nearest-3** from the "Intelligence Databases - Assignments."

This exercise focuses on Oracle Spatial features, specifically SDO_WITHIN_DISTANCE and SDO_GEOM.SDO_DISTANCE to find clinics near a given ambulance location. The buggy starter has several crucial errors related to SRID, point order, and parameter string for distance calculations.

Goal:

1. Create the CLINIC table with an SDO_GEOMETRY column and a spatial index.
2. Populate CLINIC with sample data.

3. Define the ambulance location correctly using WGS84 (SRID 4326) and (longitude, latitude) order.
4. Write a query to find clinics within 1 km of the ambulance, using the correct SRID and parameter string.
5. Write a query to find the 3 nearest clinics to the ambulance, including their distance in KM.

Key Concepts for Oracle Spatial:

- **SRID (Spatial Reference ID):** Specifies the coordinate system. 4326 is for WGS84 (longitude/latitude).
- **SDO_GEOMETRY:** Oracle's data type for spatial objects.
  - 2001: 2D point geometry type.
  - 4326: SRID for WGS84.
  - SDO_POINT_TYPE(X, Y, Z): For points, X is longitude, Y is latitude.
- **SDO_GEOM_METADATA:** Required metadata for spatial indexing.
- **SDO_WITHIN_DISTANCE:** Spatial operator for radius searches.
  - Parameter string format: 'unit=KM distance=1' for 1km radius.
- **SDO_GEOM.SDO_DISTANCE:** Function to calculate distance between two geometries.
  - Parameter string format: 'unit=KM' to get distance in kilometers.

Step 1: Create the Prerequisite Table, Metadata, Index, and Sample Data

codeSQL

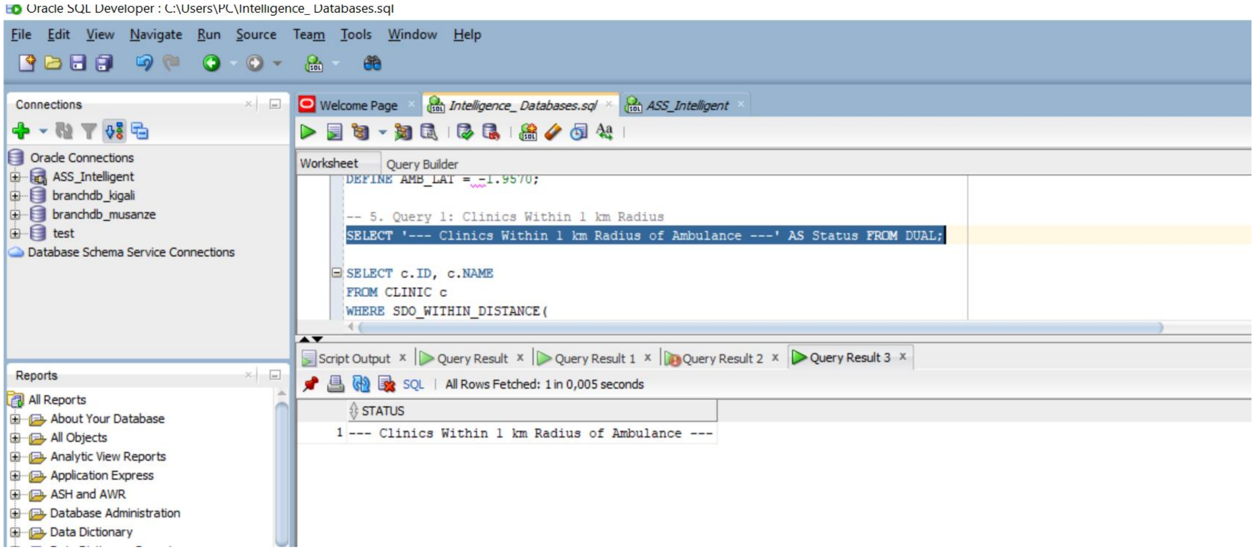
```
-- Prerequisite for Question 5: Spatial Databases


-- Drop existing spatial objects for a clean run (optional)

-- Execute in reverse order of creation due to dependencies

BEGIN

  EXECUTE IMMEDIATE 'DROP INDEX CLINIC_SPX';

EXCEPTION
```

```
  WHEN OTHERS THEN
    IF SQLCODE != -1418 THEN RAISE; END IF; -- Ignore ORA-01418 (index does not exist)
END;
/

BEGIN
  EXECUTE IMMEDIATE 'DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME = ''CLINIC'' AND COLUMN_NAME = ''GEOM''';
  EXECUTE IMMEDIATE 'DROP TABLE CLINIC CASCADE CONSTRAINTS';
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE != -942 THEN RAISE; END IF; -- Ignore ORA-00942 (table or view does not exist)
END;
/
COMMIT;

-- 1. Create TABLE CLINIC
CREATE TABLE CLINIC (
   ID        NUMBER        PRIMARY KEY,
   NAME      VARCHAR2(100)   NOT NULL,
   GEOM      SDO_GEOMETRY
);

-- 2. Insert SDO_GEOM_METADATA
-- This is crucial for spatial indexing and operations.
-- SRID 4326 (WGS84) is used, with longitude (X) and latitude (Y) bounds.
```

```sql
INSERT INTO USER_SDO_GEOM_METADATA (TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
VALUES (
   'CLINIC',
   'GEOM',
   SDO_DIM_ARRAY(
      SDO_DIM_ELEMENT('X', -180.0, 180.0, 0.005), -- Longitude range
      SDO_DIM_ELEMENT('Y', -90.0, 90.0, 0.005)    -- Latitude range
   ),
   4326 -- WGS84 SRID
);
COMMIT;
```

-- 3. Create Spatial Index CLINIC_SPX

```sql
CREATE INDEX CLINIC_SPX ON CLINIC(GEOM) INDEXTYPE IS MDSYS.SPATIAL_INDEX;


-- 4. Populate CLINIC with sample data (around the ambulance location lon=30.0600, lat=-1.9570)
-- Important: SDO_POINT_TYPE takes (longitude, latitude) for WGS84
INSERT INTO CLINIC (ID, NAME, GEOM) VALUES (1, 'Central Clinic', SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(30.0610, -1.9560, NULL), NULL, NULL)); -- ~0.1km away
INSERT INTO CLINIC (ID, NAME, GEOM) VALUES (2, 'Northside Hospital', SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(30.0550, -1.9500, NULL), NULL, NULL)); -- ~0.8km away
INSERT INTO CLINIC (ID, NAME, GEOM) VALUES (3, 'East Medical Center', SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(30.0700, -1.9590, NULL), NULL, NULL)); -- ~0.9km away
INSERT INTO CLINIC (ID, NAME, GEOM) VALUES (4, 'Southwest Health', SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(30.0400, -1.9650, NULL), NULL, NULL)); -- ~2.5km away
INSERT INTO CLINIC (ID, NAME, GEOM) VALUES (5, 'Community Care', SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(30.0800, -1.9400, NULL), NULL, NULL)); -- ~3km away

INSERT INTO CLINIC (ID, NAME, GEOM) VALUES (6, 'Riverside Clinic', SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(30.0605, -1.9575, NULL), NULL, NULL)); -- Very close!


COMMIT;


SELECT '--- Spatial Setup Complete ---' AS Status FROM DUAL;
```



```
DEFINE AMB_LAT = -1.9570;

-- 5. Query 1: Clinics Within 1 km Radius
SELECT '--- Clinics Within 1 km Radius of Ambulance ---' AS Status FROM DUAL;

SELECT c.ID, c.NAME
FROM CLINIC c
WHERE SDO_WITHIN_DISTANCE(
```

All Rows Fetched: 1 in 0,005 seconds

```
  STATUS
1 --- Clinics Within 1 km Radius of Ambulance ---
```

```
SELECT TABLE_NAME, COLUMN_NAME, SRID FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME = 'CLINIC';

SELECT 'Sample Clinics:' AS Status FROM DUAL;

SELECT ID, NAME, SDO_GEOM.GET_WKT(GEOM) AS GEOM_WKT FROM CLINIC;
```

Step 2: Define the Ambulance Point Correctly

The ambulance location is lon=30.0600, lat=-1.9570. This needs to be consistently represented using SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(longitude, latitude, NULL), NULL, NULL). It's good practice to define this once, either as a bind variable or in a WITH clause for clarity and consistency.

codeSQL

```
-- Define the ambulance point (lon=30.0600, lat=-1.9570)
-- Ensure SRID is 4326 and point order is (longitude, latitude)
DEFINE AMB_LON = 30.0600;
DEFINE AMB_LAT = -1.9570;

-- Correctly constructed ambulance point geometry
-- We can use it directly or define it in a WITH clause.
-- For the "Nearest-3" query, using a WITH clause is cleaner than a bind variable for the SDO_GEOMETRY type.
```

Step 3: Corrected Query 1 - Clinics Within 1 km Radius

Debugging the buggy starter:

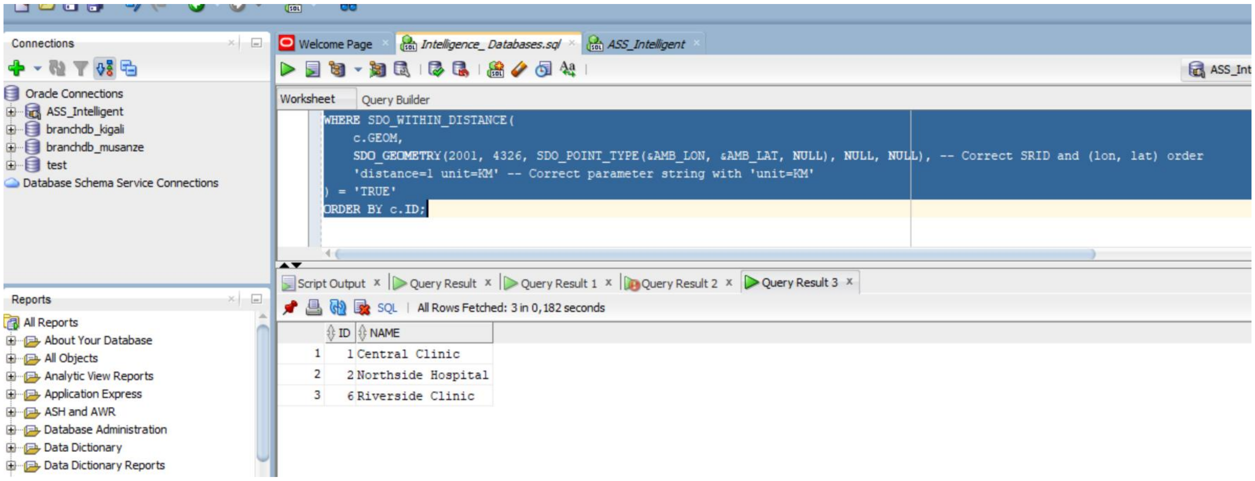
codeSQL

```
-- BUGGY: SRID 3857, reversed lat/lon, missing 'unit=KM'
SELECT C.ID, C.NAME
FROM CLINIC C
WHERE SDO_WITHIN_DISTANCE(
C.GEOM,
SDO_GEOMETRY(2001, 3857, SDO_POINT_TYPE(-1.9570, 30.0600, NULL), NULL, NULL), -- WRONG SRID, WRONG LON/LAT ORDER
'distance=1') = 'TRUE'; -- MISSING UNIT
```

Corrected query:

codeSQL

```
SELECT '--- Clinics Within 1 km Radius ---' AS Status FROM DUAL;

SELECT c.ID, c.NAME
FROM CLINIC c
WHERE SDO_WITHIN_DISTANCE(
    c.GEOM,
    SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(&AMB_LON, &AMB_LAT, NULL), NULL, NULL), -- Correct SRID 4326, correct (lon, lat)

    'distance=1 unit=KM' -- Correct parameter string including 'unit=KM'

) = 'TRUE'

ORDER BY c.ID;
```



```
WHERE SDO_WITHIN_DISTANCE(
    c.GEOM,
    SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(&AMB_LON, &AMB_LAT, NULL), NULL, NULL), -- Correct SRID and (lon, lat) order
    'distance=1 unit=KM' -- Correct parameter string with 'unit=KM'
) = 'TRUE'
ORDER BY c.ID;
```

All Rows Fetched: 3 in 0,182 seconds

```
   ID NAME
1   1 Central Clinic
2   2 Northside Hospital
3   6 Riverside Clinic
```

Step 4: Corrected Query 2 - Nearest 3 Clinics with Distances

Debugging the buggy starter:

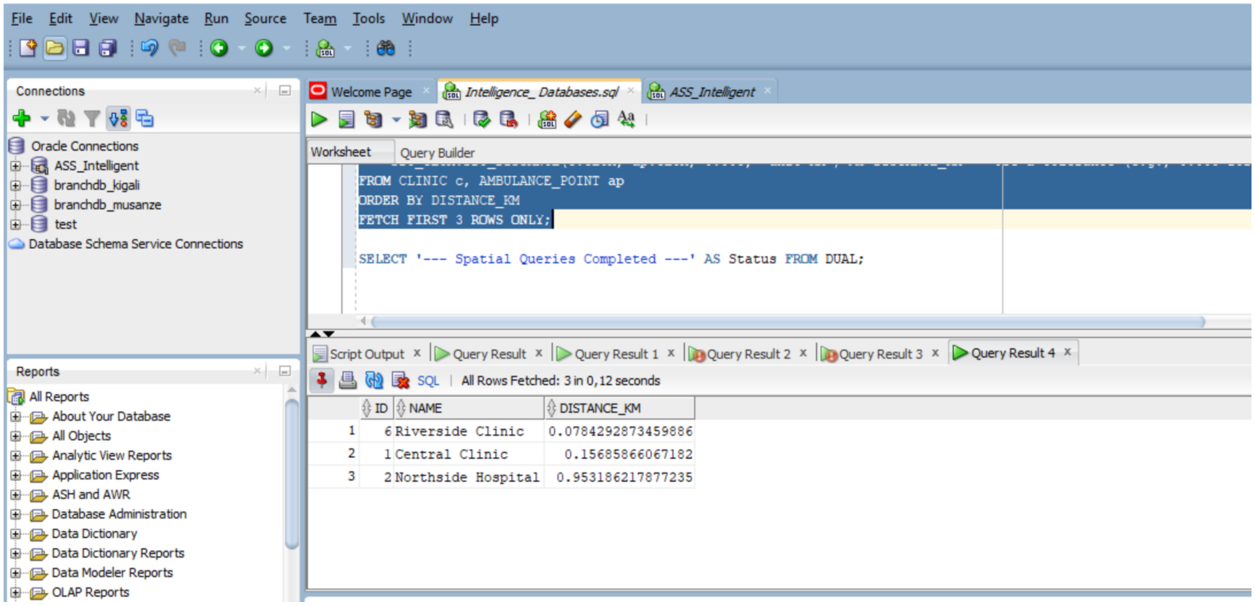
codeSQL

```
-- BUGGY: :AMB_POINT not defined; units unspecified

SELECT C.ID, C.NAME,

SDO_GEOM.SDO_DISTANCE(C.GEOM, :AMB_POINT, 1) AS KM -- :AMB_POINT not defined, '1' is tolerance not unit

FROM CLINIC C

ORDER BY KM

FETCH FIRST 3 ROWS ONLY;
```

Corrected query (using a WITH clause for the ambulance point for clarity and reusability):

codeSQL

```
SELECT '--- Nearest 3 Clinics with Distances ---' AS Status FROM DUAL;


WITH AMBULANCE_POINT AS (
   SELECT SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(&AMB_LON, &AMB_LAT, NULL), NULL, NULL) AS GEOM
   FROM DUAL
)
SELECT
   c.ID,
   c.NAME,
   SDO_GEOM.SDO_DISTANCE(c.GEOM, ap.GEOM, 0.005, 'unit=KM') AS DISTANCE_KM -- Correct parameter string 'unit=KM'
FROM CLINIC c, AMBULANCE_POINT ap
ORDER BY DISTANCE_KM
FETCH FIRST 3 ROWS ONLY;
```



```
FROM CLINIC c, AMBULANCE_POINT ap
ORDER BY DISTANCE_KM
FETCH FIRST 3 ROWS ONLY;

SELECT '--- Spatial Queries Completed ---' AS Status FROM DUAL;
```

All Rows Fetched: 3 in 0,12 seconds

```
     ID NAME               DISTANCE_KM
1     6 Riverside Clinic   0.0784292873459886
2     1 Central Clinic       0.15685866067182
3     2 Northside Hospital  0.953186217877235
```