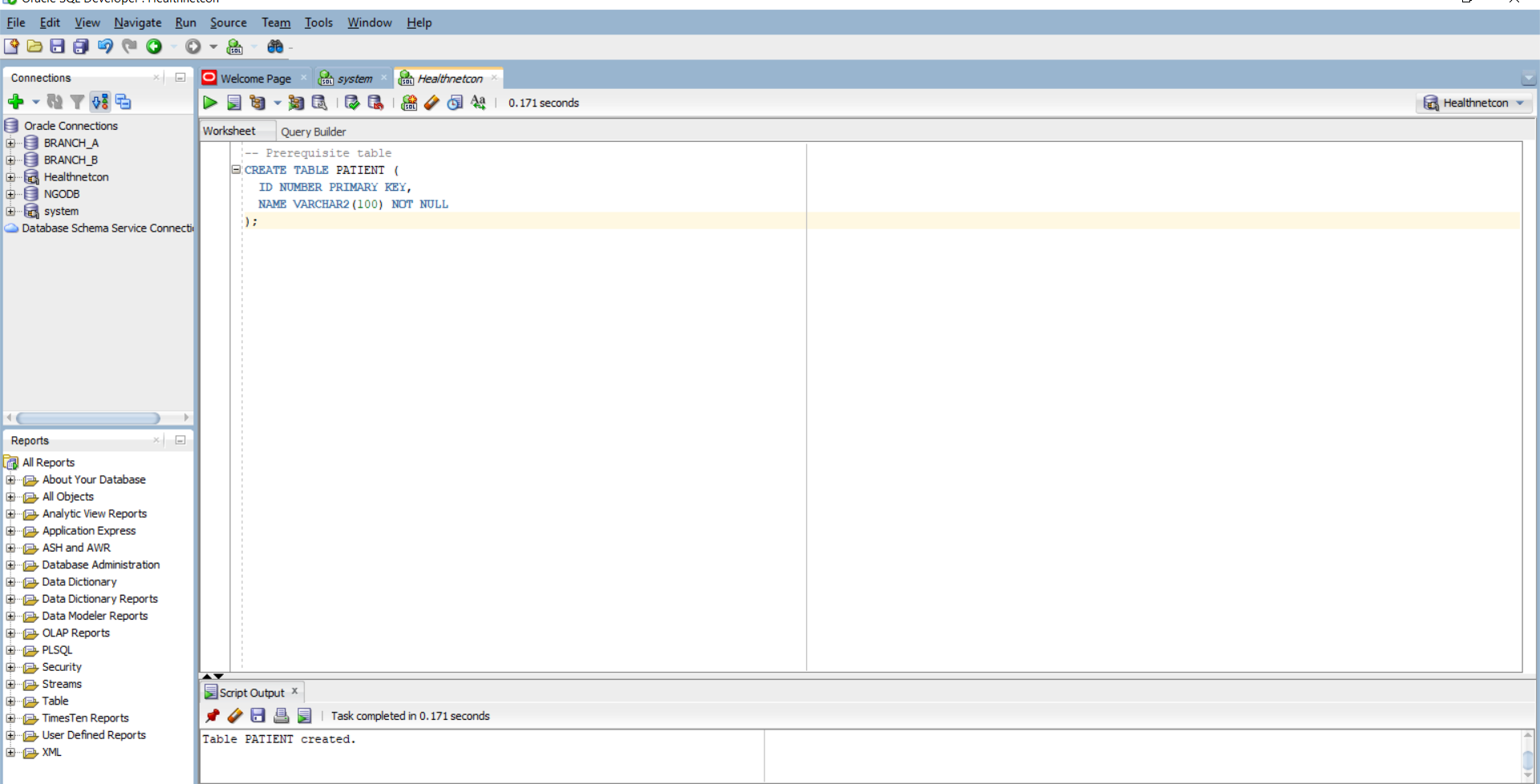
1.

-- Prerequisite table

CREATE TABLE PATIENT (

ID NUMBER PRIMARY KEY,

NAME VARCHAR2(100) NOT NULL

);

-- Corrected PATIENT\_MED table

CREATE TABLE PATIENT\_MED (

PATIENT\_MED\_ID NUMBER PRIMARY KEY, -- unique id

PATIENT\_ID NUMBER NOT NULL REFERENCES PATIENT(ID), -- must reference an existing patient

MED\_NAME VARCHAR2(80) NOT NULL, -- mandatory field

DOSE\_MG NUMBER(6,2) CHECK (DOSE\_MG >= 0), -- non-negative dose

START\_DT DATE,

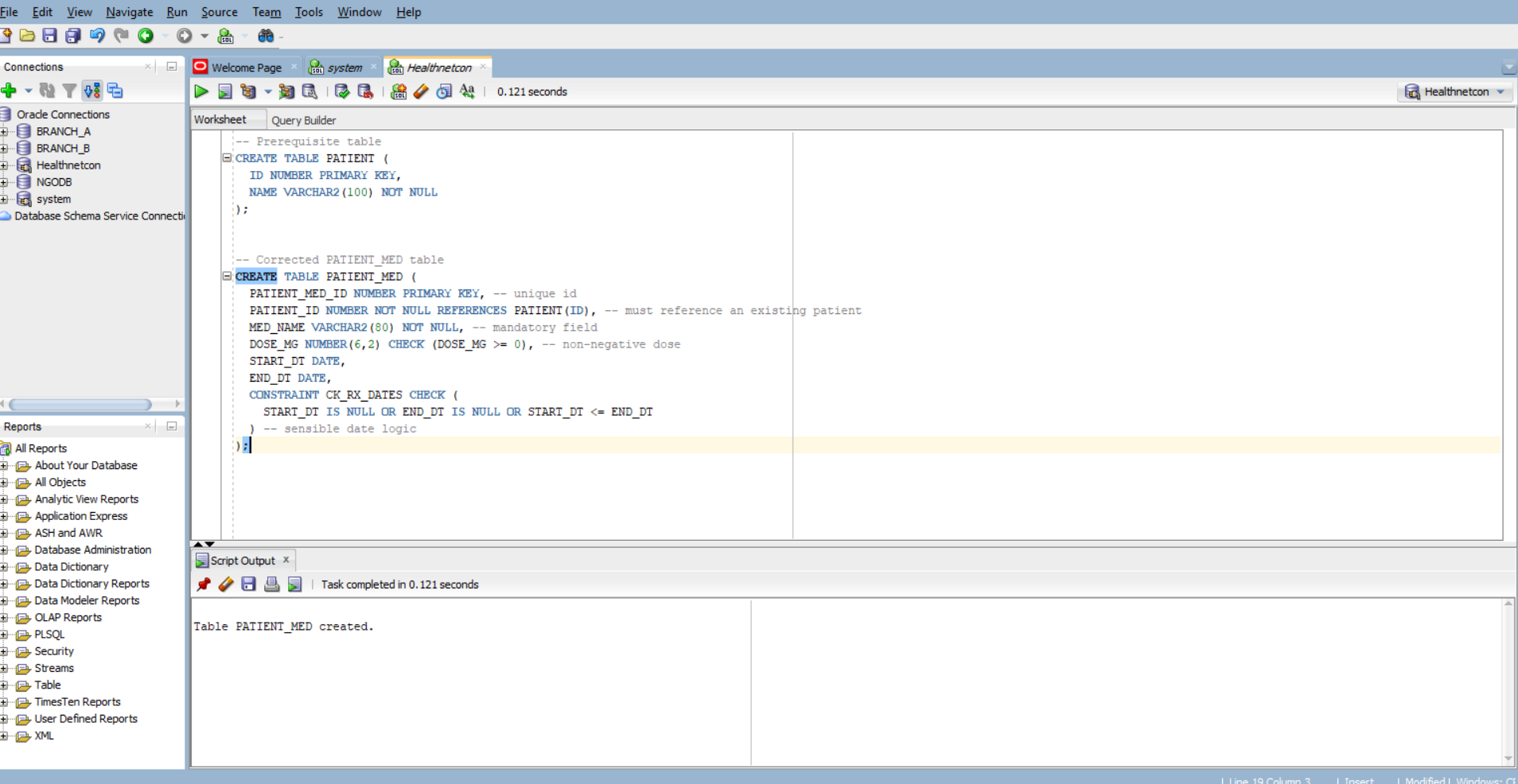
END\_DT DATE,

CONSTRAINT CK\_RX\_DATES CHECK (

START\_DT IS NULL OR END\_DT IS NULL OR START\_DT <= END\_DT

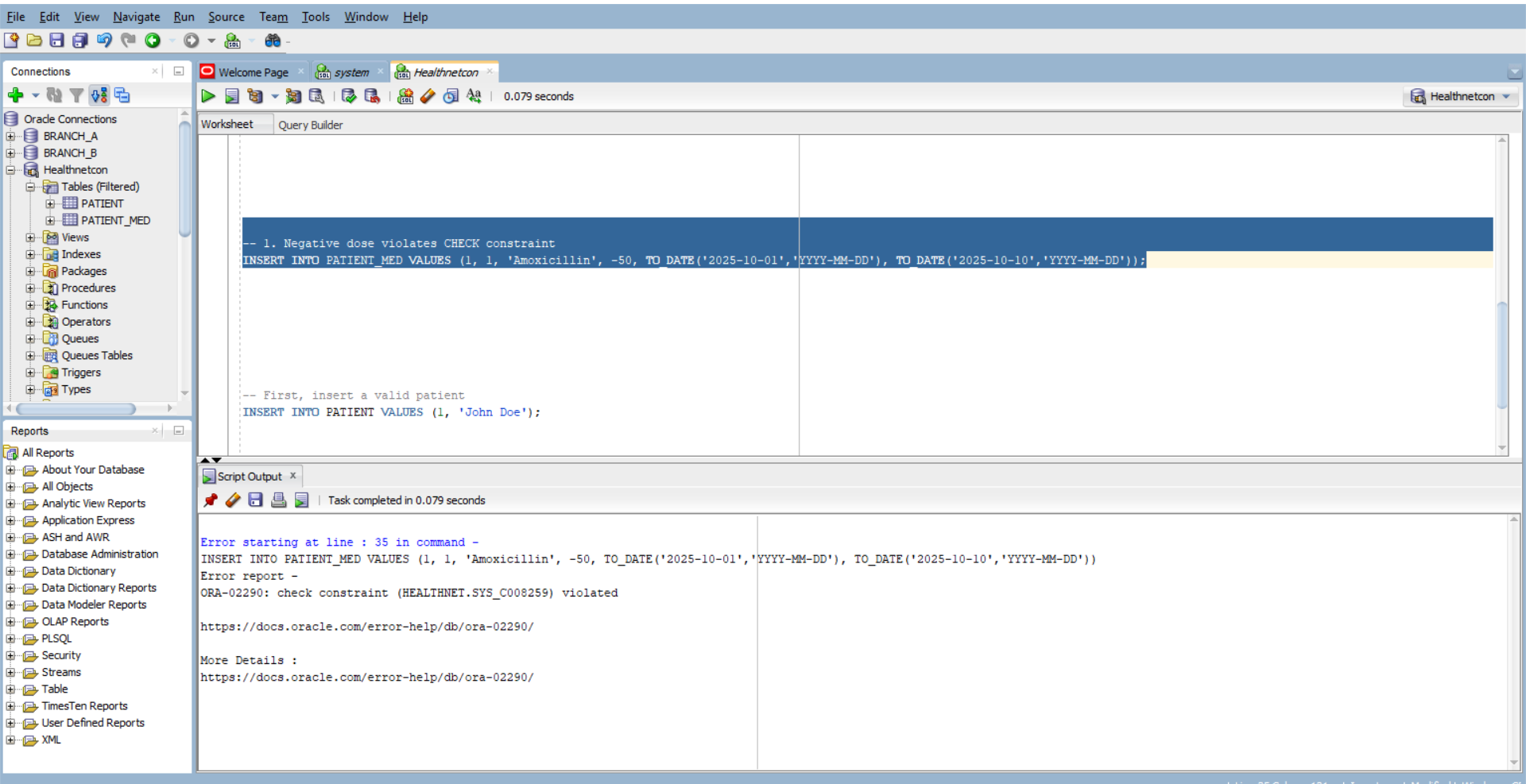
) -- sensible date logic

);



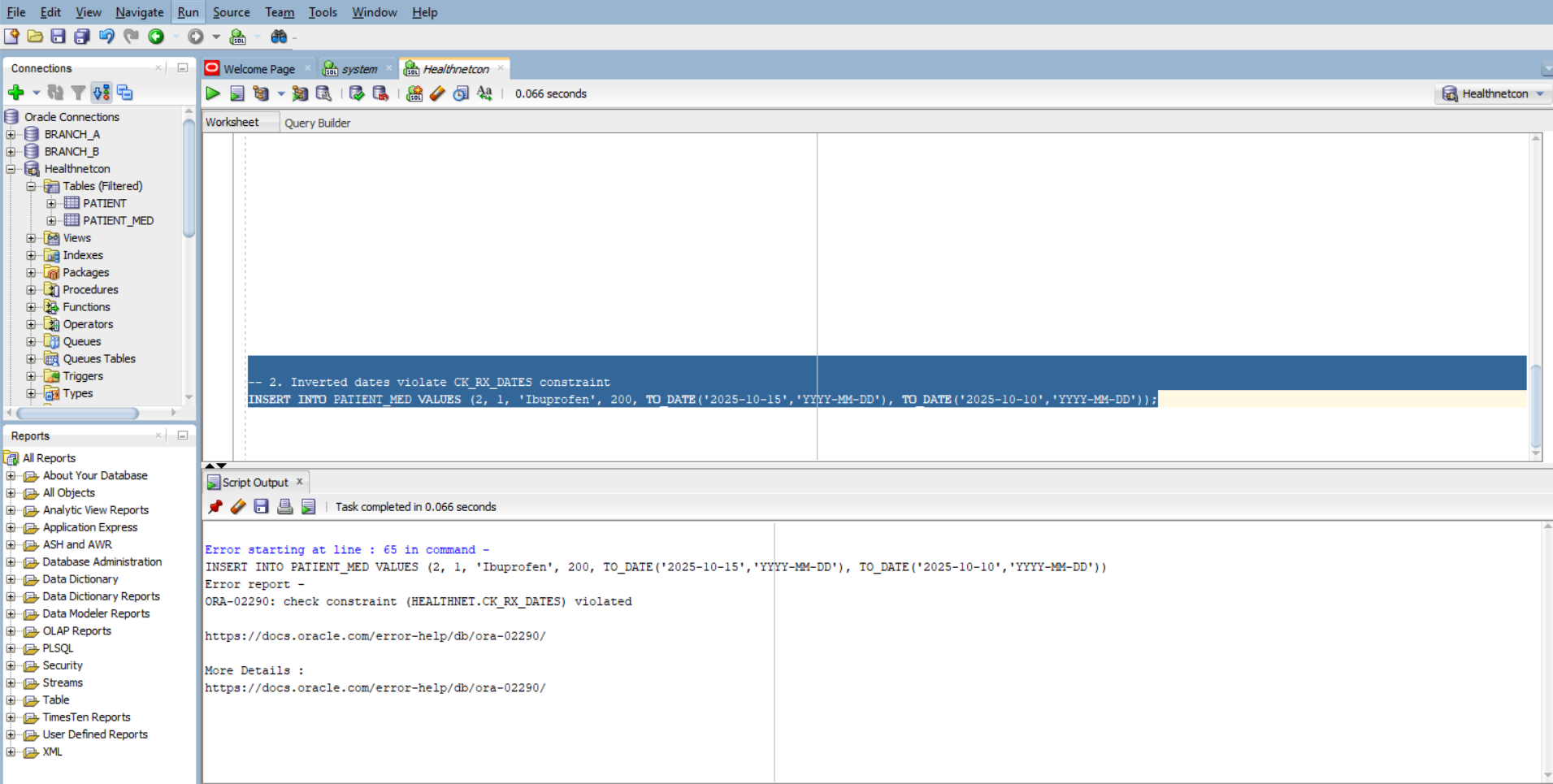
-- 1. Negative dose violates CHECK constraint

INSERT INTO PATIENT\_MED VALUES (1, 1, 'Amoxicillin', -50, TO\_DATE('2025-10-01','YYYY-MM-DD'), TO\_DATE('2025-10-10','YYYY-MM-DD'));



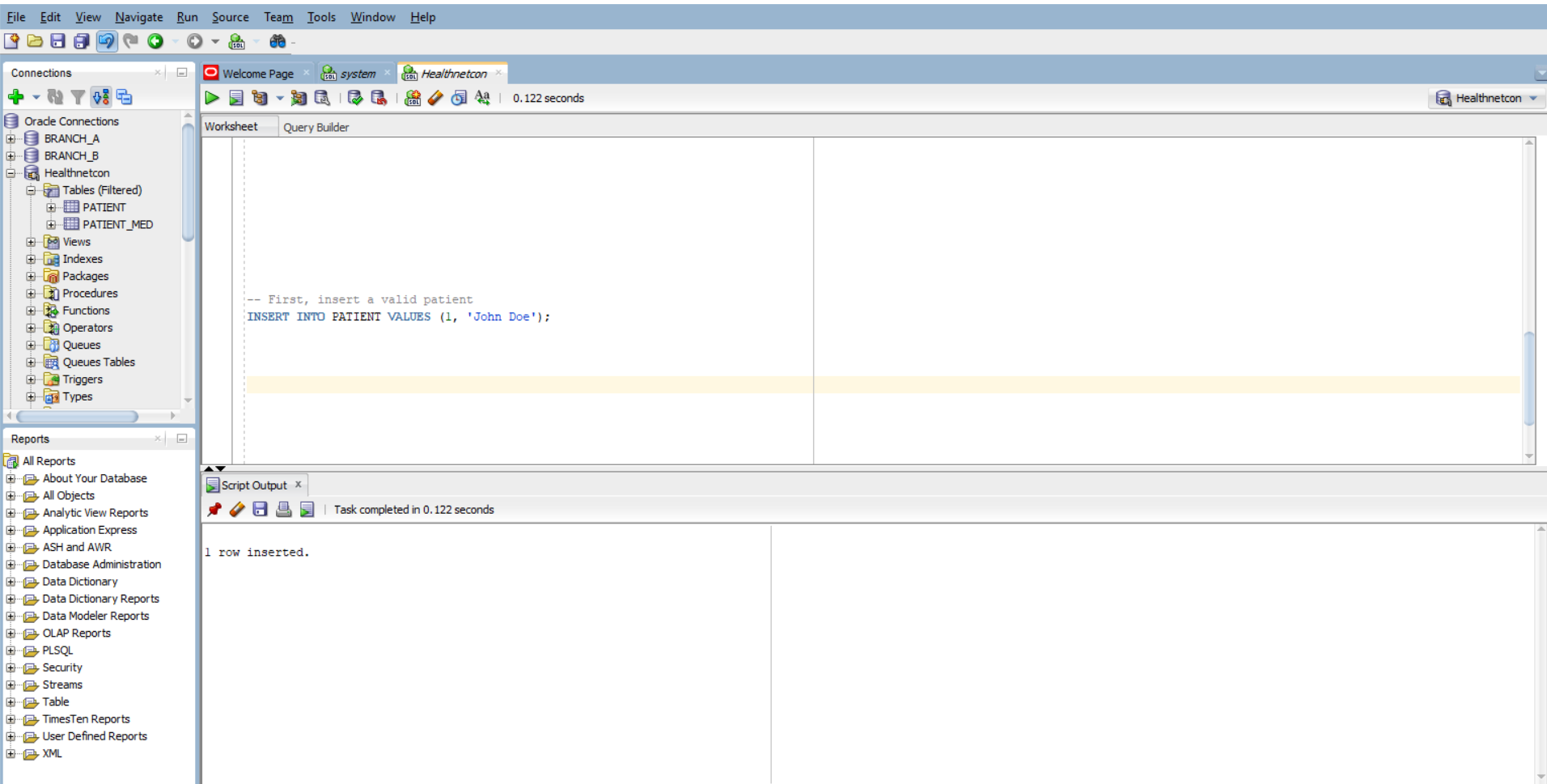
-- 2. Inverted dates violate CK\_RX\_DATES constraint

INSERT INTO PATIENT\_MED VALUES (2, 1, 'Ibuprofen', 200, TO\_DATE('2025-10-15','YYYY-MM-DD'), TO\_DATE('2025-10-10','YYYY-MM-DD'));



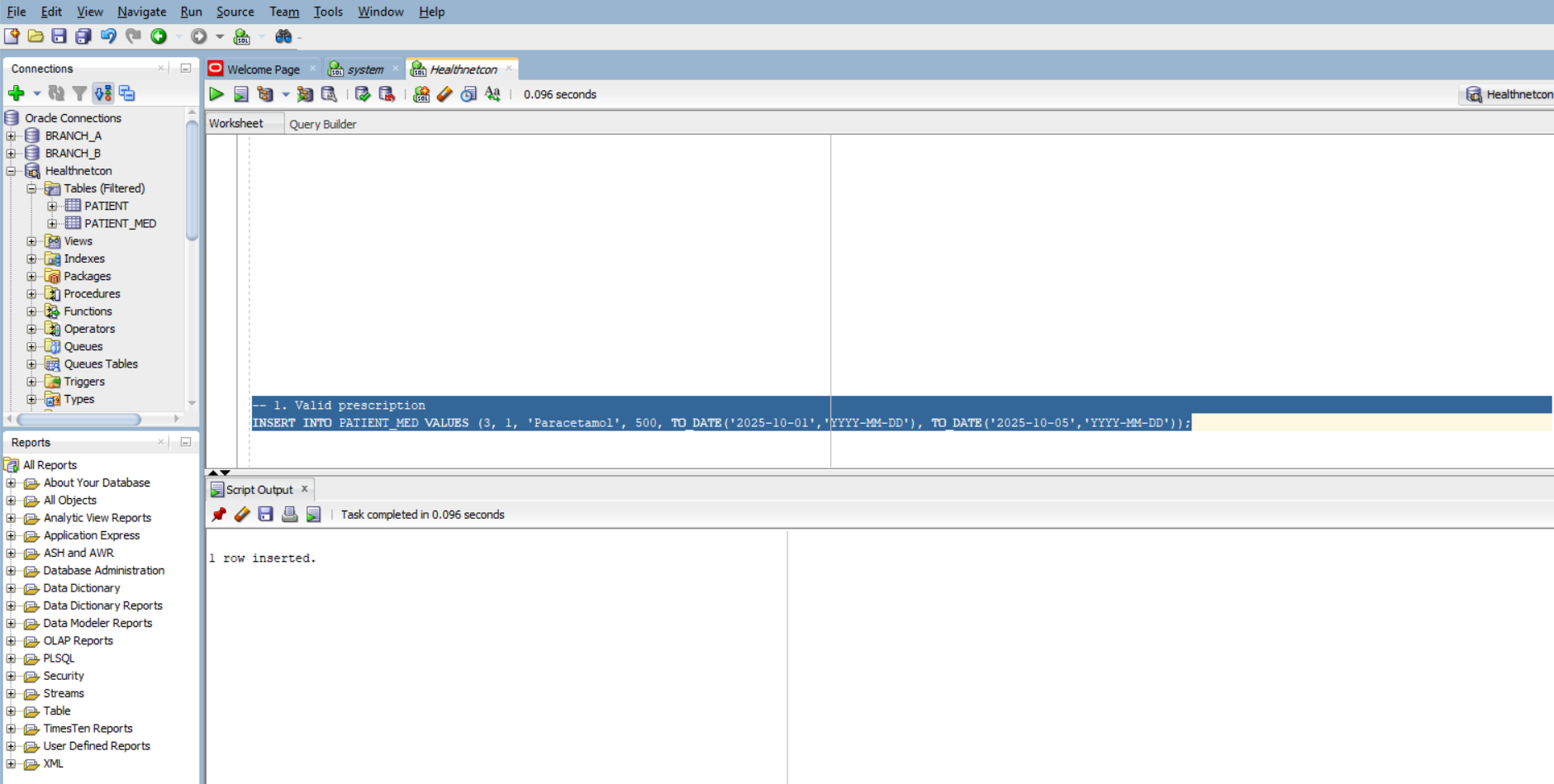
-- First, insert a valid patient

INSERT INTO PATIENT VALUES (1, 'John Doe');



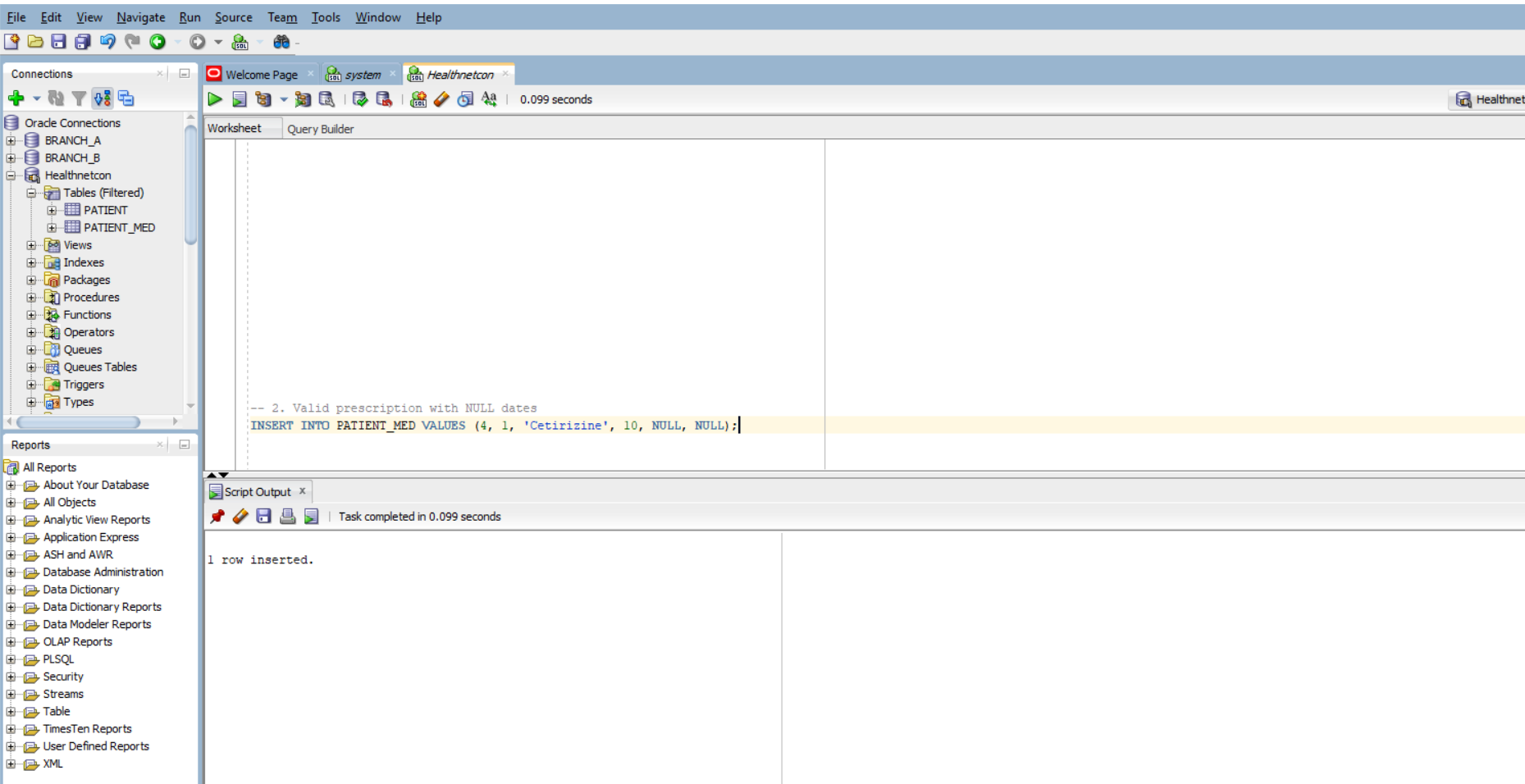
-- 1. Valid prescription

INSERT INTO PATIENT\_MED VALUES (3, 1, 'Paracetamol', 500, TO\_DATE('2025-10-01','YYYY-MM-DD'), TO\_DATE('2025-10-05','YYYY-MM-DD'));

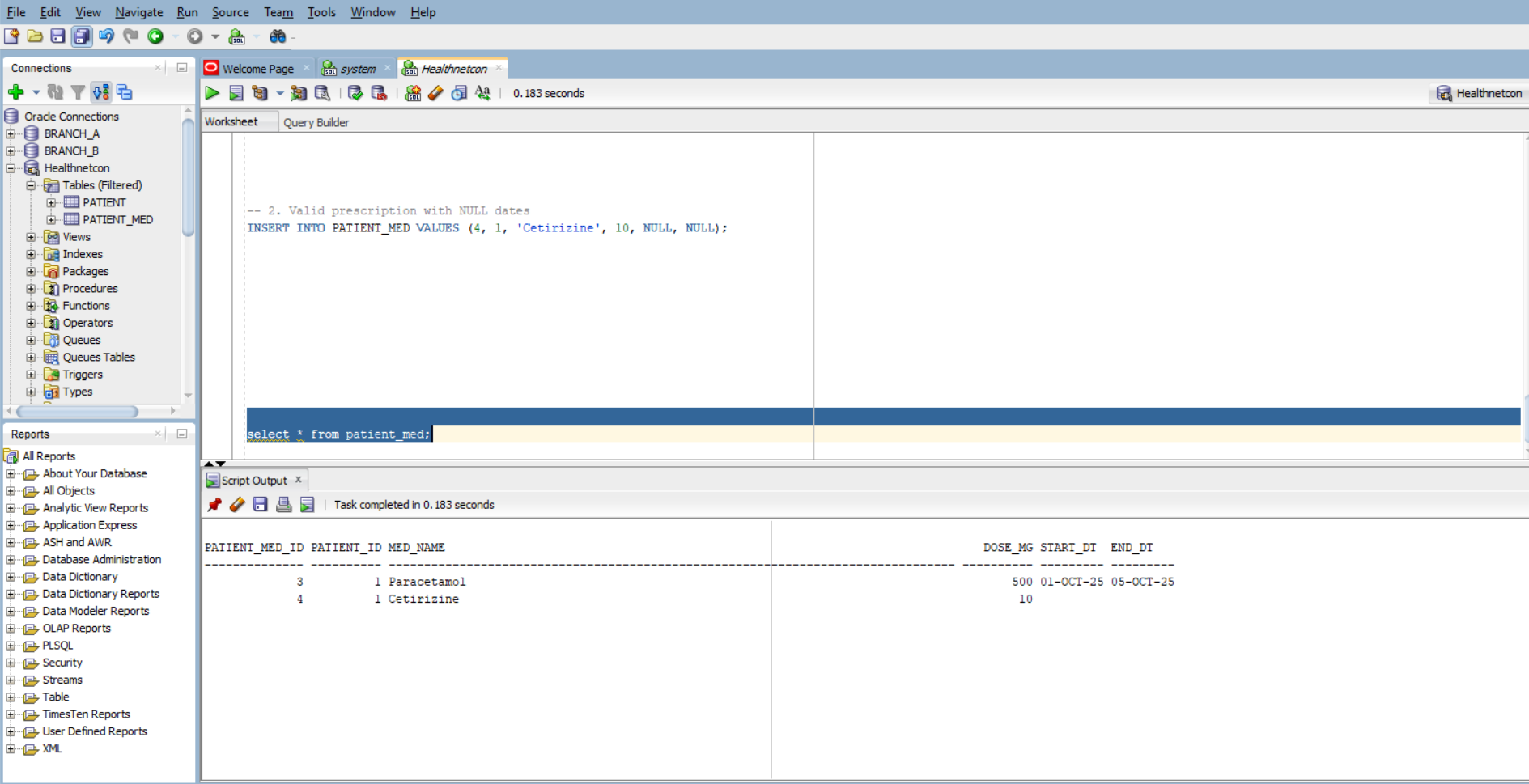


-- 2. Valid prescription with NULL dates

INSERT INTO PATIENT\_MED VALUES (4, 1, 'Cetirizine', 10, NULL, NULL);



select \* from patient\_med;



| **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 |
| **2. -- Main bill table**  **CREATE TABLE BILL (**  **ID NUMBER PRIMARY KEY,**  **TOTAL NUMBER(12,2)**  **);**  **-- Items linked to bills**  **CREATE TABLE BILL\_ITEM (**  **BILL\_ID NUMBER,**  **AMOUNT NUMBER(12,2),**  **UPDATED\_AT DATE,**  **CONSTRAINT FK\_BILL\_ITEM\_BILL FOREIGN KEY (BILL\_ID) REFERENCES BILL(ID)**  **);**  **-- Audit log for changes**  **CREATE TABLE BILL\_AUDIT (**  **BILL\_ID NUMBER,**  **OLD\_TOTAL NUMBER(12,2),**  **NEW\_TOTAL NUMBER(12,2),**  **CHANGED\_AT DATE**  **);** |  |  |  |

Correct Compound Trigger: TRG\_BILL\_TOTAL\_CMP: it updates BILL.TOTAL once per statement and logs changes into BILL\_AUDIT, avoiding mutating-table errors and redundant updates.

CREATE OR REPLACE TRIGGER TRG\_BILL\_TOTAL\_STMT

AFTER INSERT OR UPDATE OR DELETE ON BILL\_ITEM

DECLARE

TYPE bill\_id\_table IS TABLE OF BILL\_ITEM.BILL\_ID%TYPE INDEX BY PLS\_INTEGER;

v\_bill\_ids bill\_id\_table;

v\_index PLS\_INTEGER := 0;

BEGIN

-- Collect affected BILL\_IDs

FOR r IN (

SELECT DISTINCT BILL\_ID FROM BILL\_ITEM

WHERE BILL\_ID IS NOT NULL

) LOOP

v\_index := v\_index + 1;

v\_bill\_ids(v\_index) := r.BILL\_ID;

END LOOP;

-- Recompute totals and insert audit rows

FOR i IN 1 .. v\_index LOOP

DECLARE

v\_old\_total BILL.TOTAL%TYPE;

v\_new\_total BILL.TOTAL%TYPE;

BEGIN

SELECT TOTAL INTO v\_old\_total FROM BILL WHERE ID = v\_bill\_ids(i);

SELECT NVL(SUM(AMOUNT), 0) INTO v\_new\_total FROM BILL\_ITEM WHERE BILL\_ID = v\_bill\_ids(i);

UPDATE BILL SET TOTAL = v\_new\_total WHERE ID = v\_bill\_ids(i);

INSERT INTO BILL\_AUDIT (BILL\_ID, OLD\_TOTAL, NEW\_TOTAL, CHANGED\_AT)

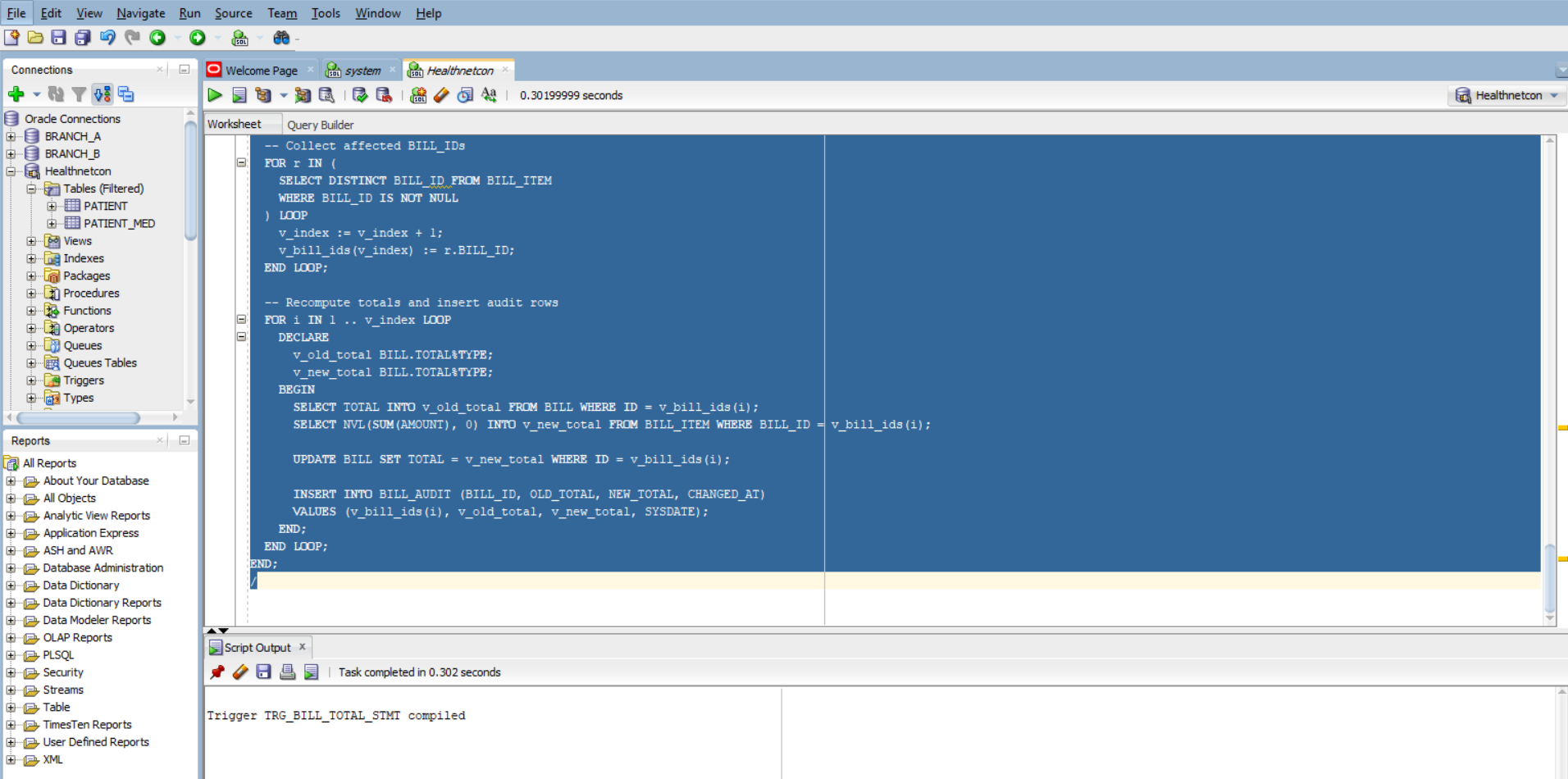
VALUES (v\_bill\_ids(i), v\_old\_total, v\_new\_total, SYSDATE);

END;

END LOOP;

END;

/

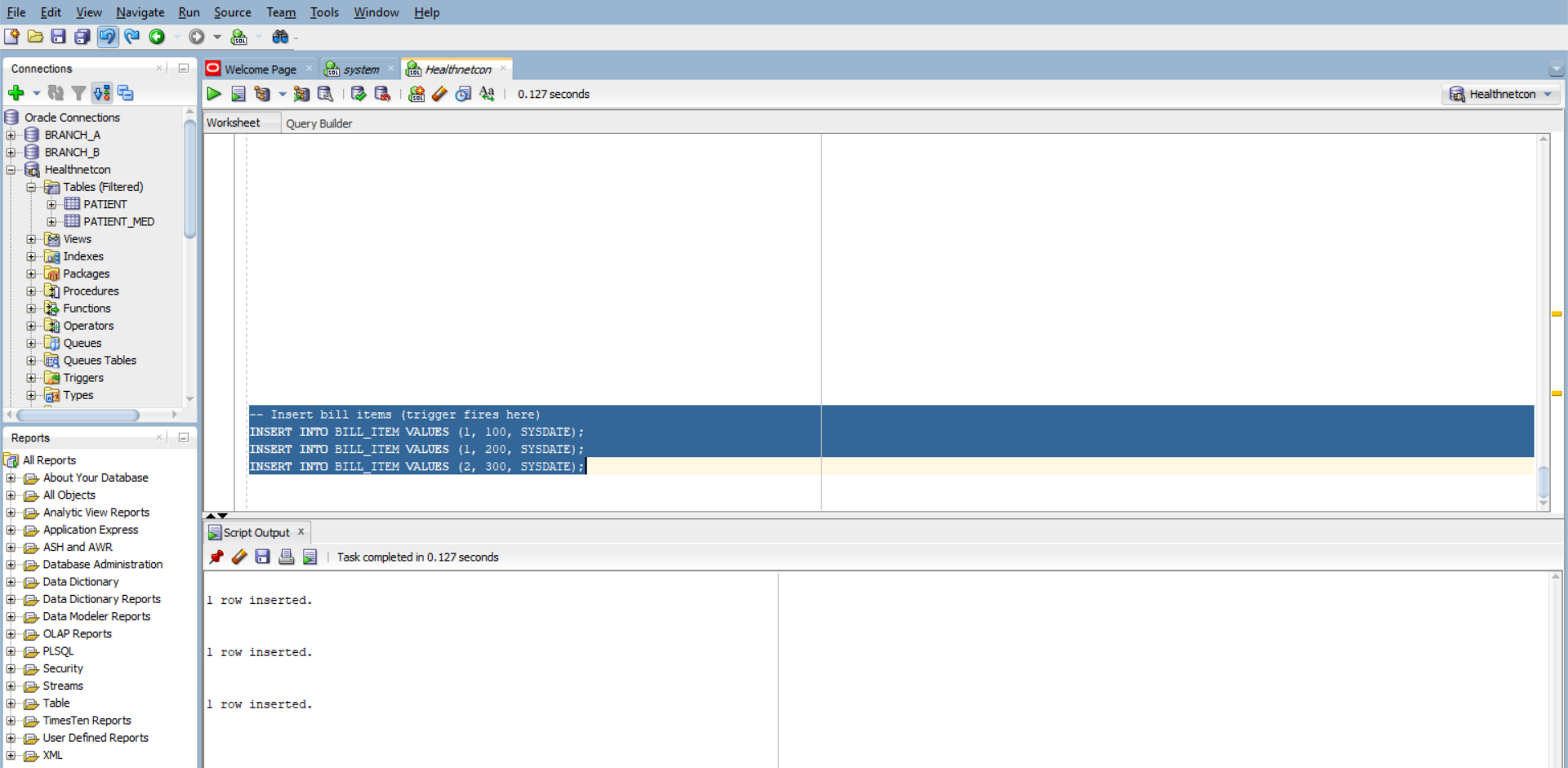


-- Insert bill items (trigger fires here)

INSERT INTO BILL\_ITEM VALUES (1, 100, SYSDATE);

INSERT INTO BILL\_ITEM VALUES (1, 200, SYSDATE);

INSERT INTO BILL\_ITEM VALUES (2, 300, SYSDATE);

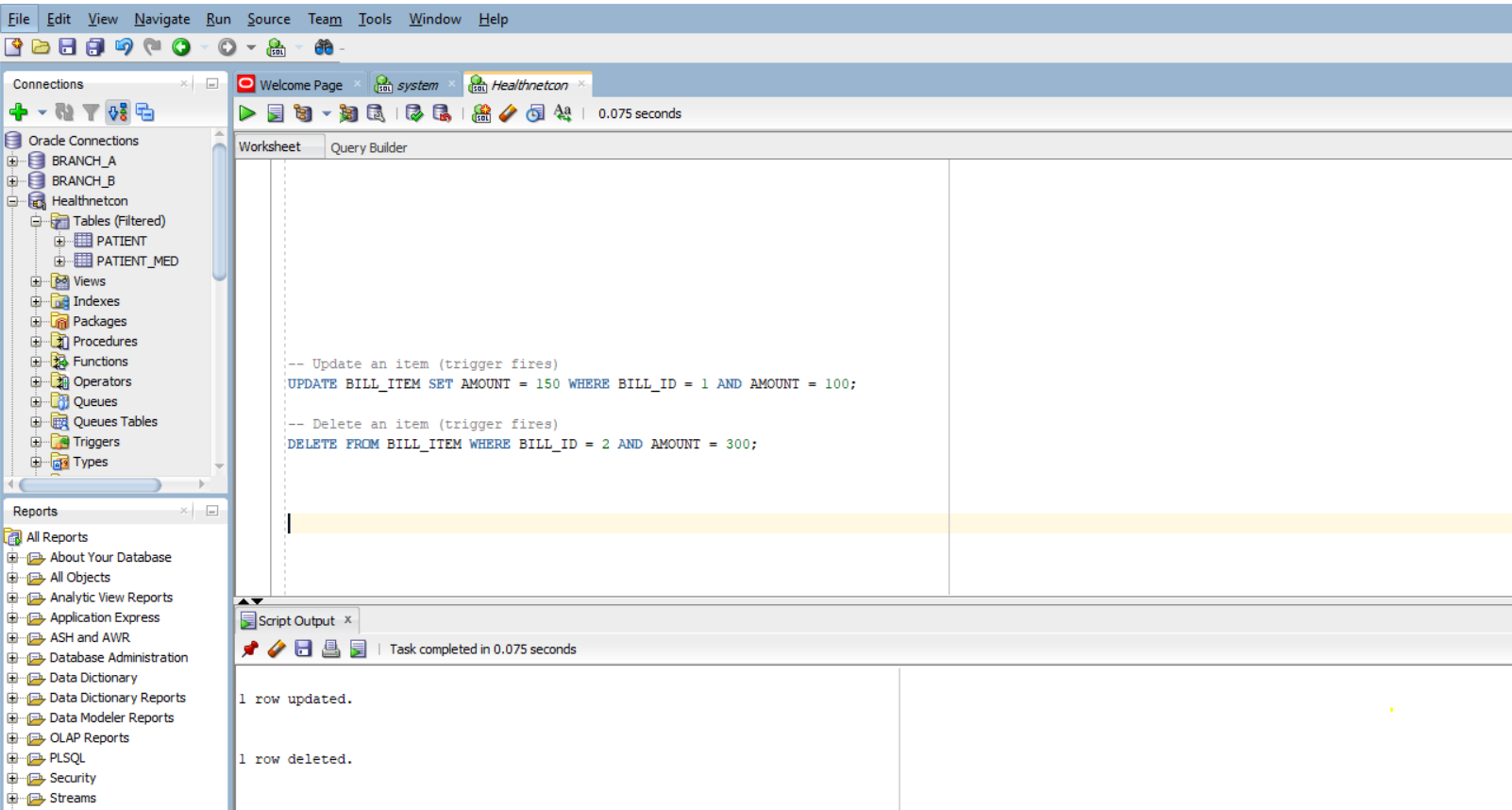


-- Update an item (trigger fires)

UPDATE BILL\_ITEM SET AMOUNT = 150 WHERE BILL\_ID = 1 AND AMOUNT = 100;

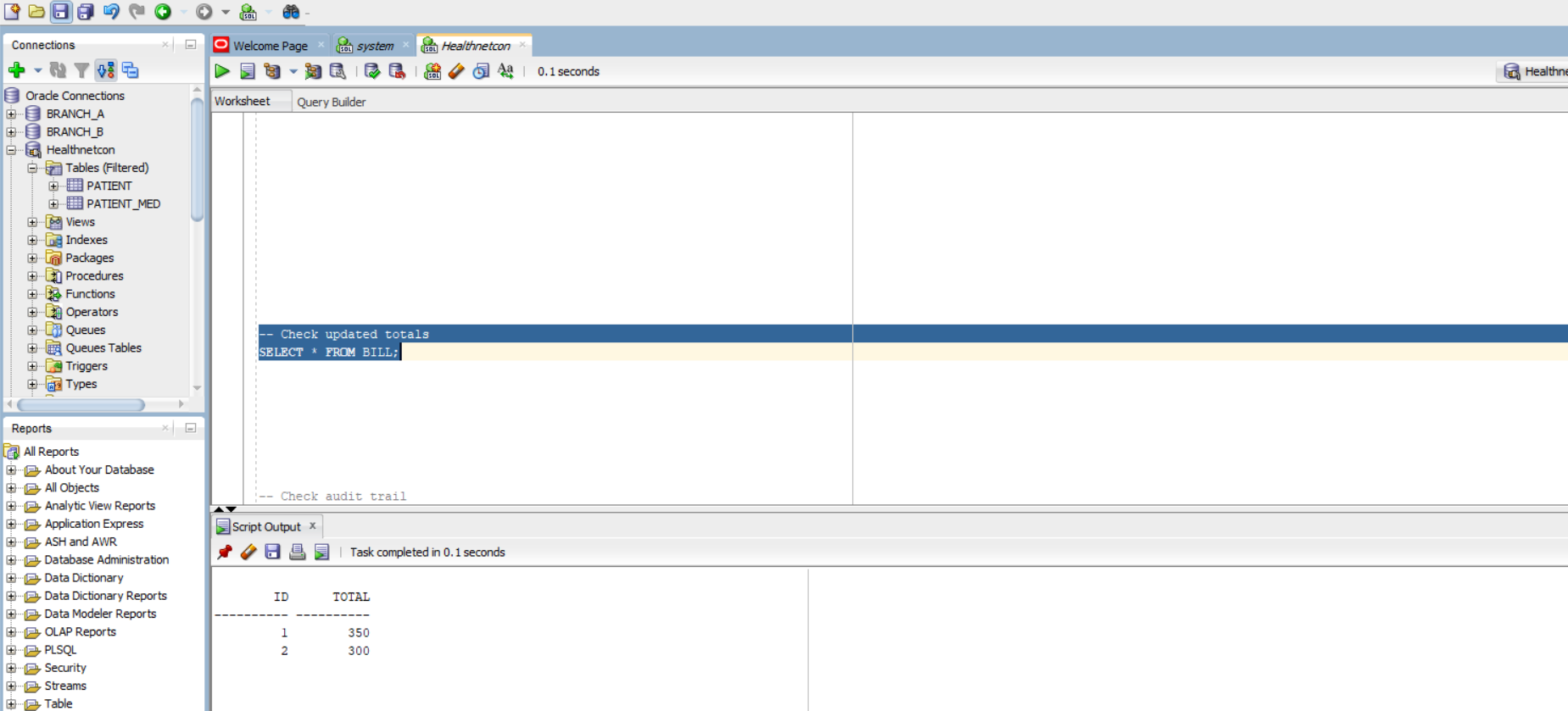
-- Delete an item (trigger fires)

DELETE FROM BILL\_ITEM WHERE BILL\_ID = 2 AND AMOUNT = 300;



-- Check updated totals

SELECT \* FROM BILL;



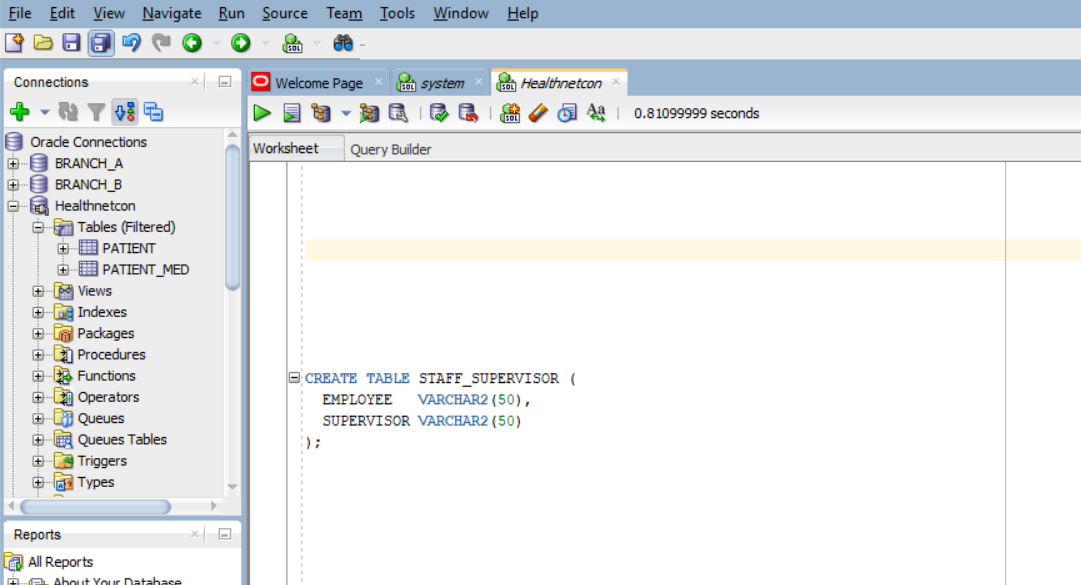
-- Check audit trail

SELECT \* FROM BILL\_AUDIT ORDER BY CHANGED\_AT;

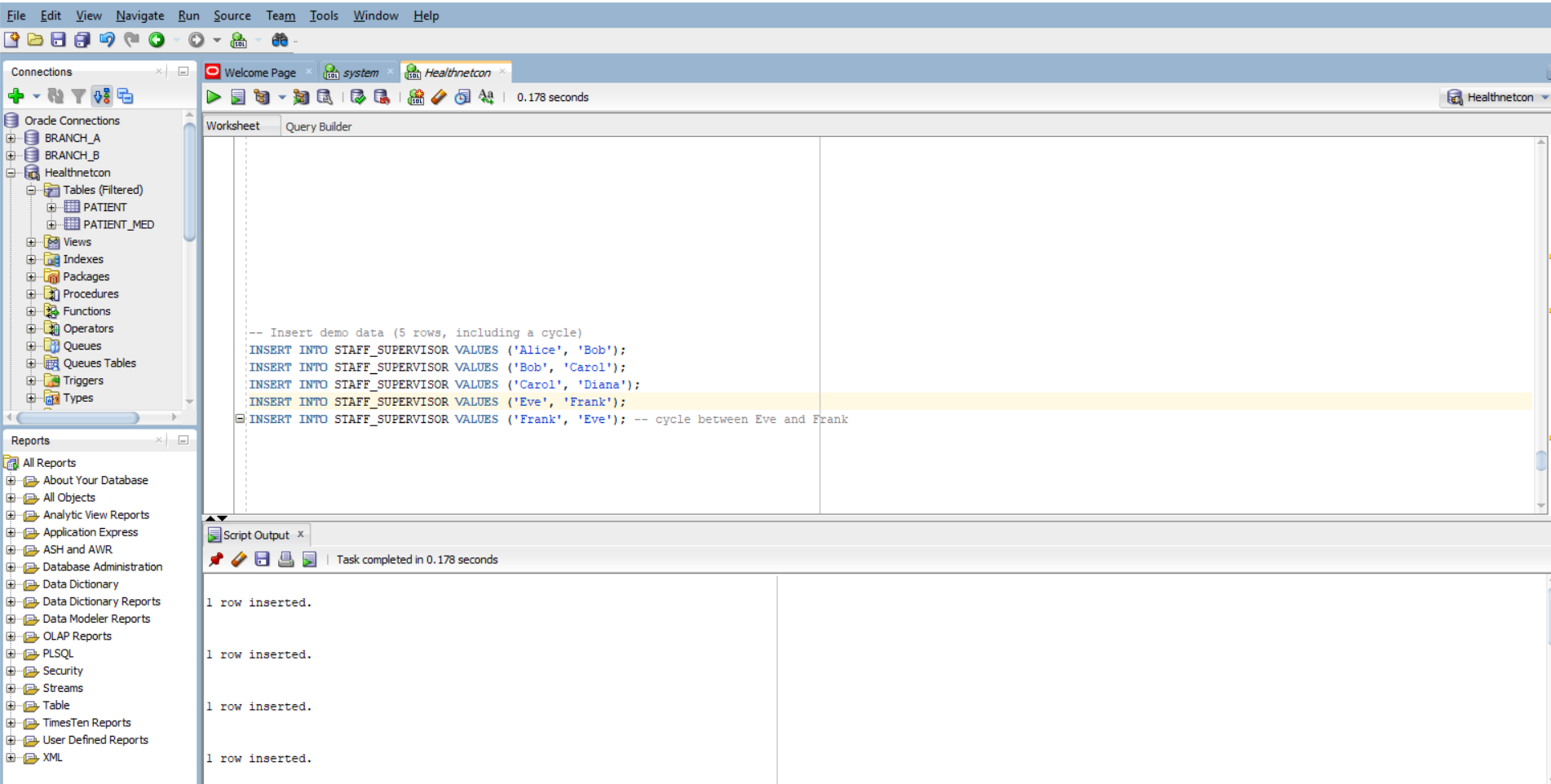


* BILL.TOTAL for ID 1 should reflect the sum of its items (e.g., 150 + 200 = 350).
* BILL.TOTAL for ID 2 should be 0 after deletion.
* BILL\_AUDIT should show old and new totals for each change.

3.



Inserting rows



-- Corrected recursive query

WITH SUPERS (EMP, SUP, HOPS, PATH) AS (

-- Anchor: start with direct supervision, hop count = 1

SELECT EMPLOYEE, SUPERVISOR, 1, EMPLOYEE || '>' || SUPERVISOR

FROM STAFF\_SUPERVISOR

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 INSTR(T.PATH, T.SUP) = 0 -- cycle guard

)

-- Final selection: top supervisor per employee

SELECT EMP, SUP AS TOP\_SUPERVISOR, HOPS

FROM (

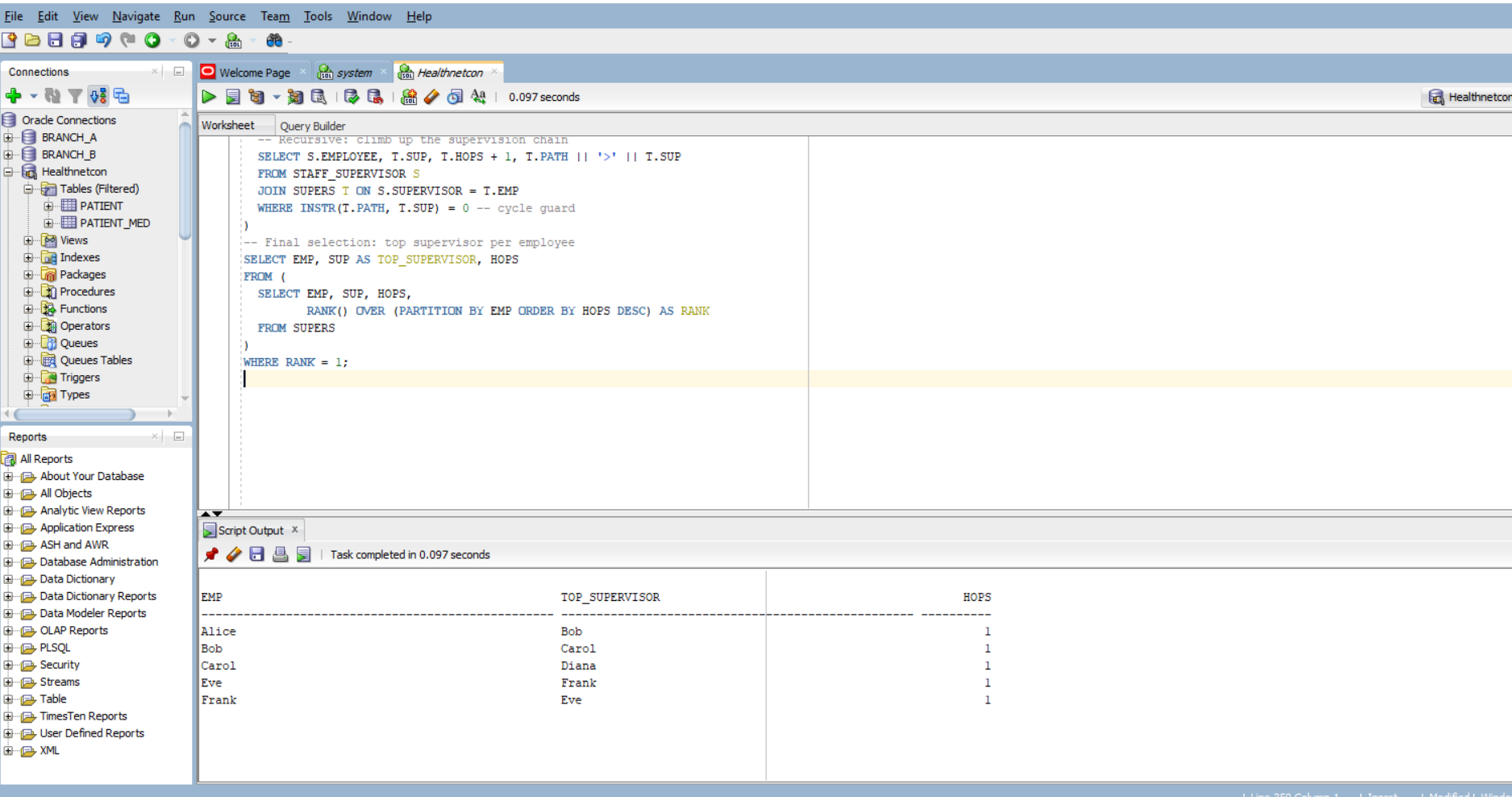
SELECT EMP, SUP, HOPS,

RANK() OVER (PARTITION BY EMP ORDER BY HOPS DESC) AS RANK

FROM SUPERS

)

WHERE RANK = 1;



| **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 |

Diana

└── Carol

└── Bob

└── Alice

Eve ↔ Frank (cycle)

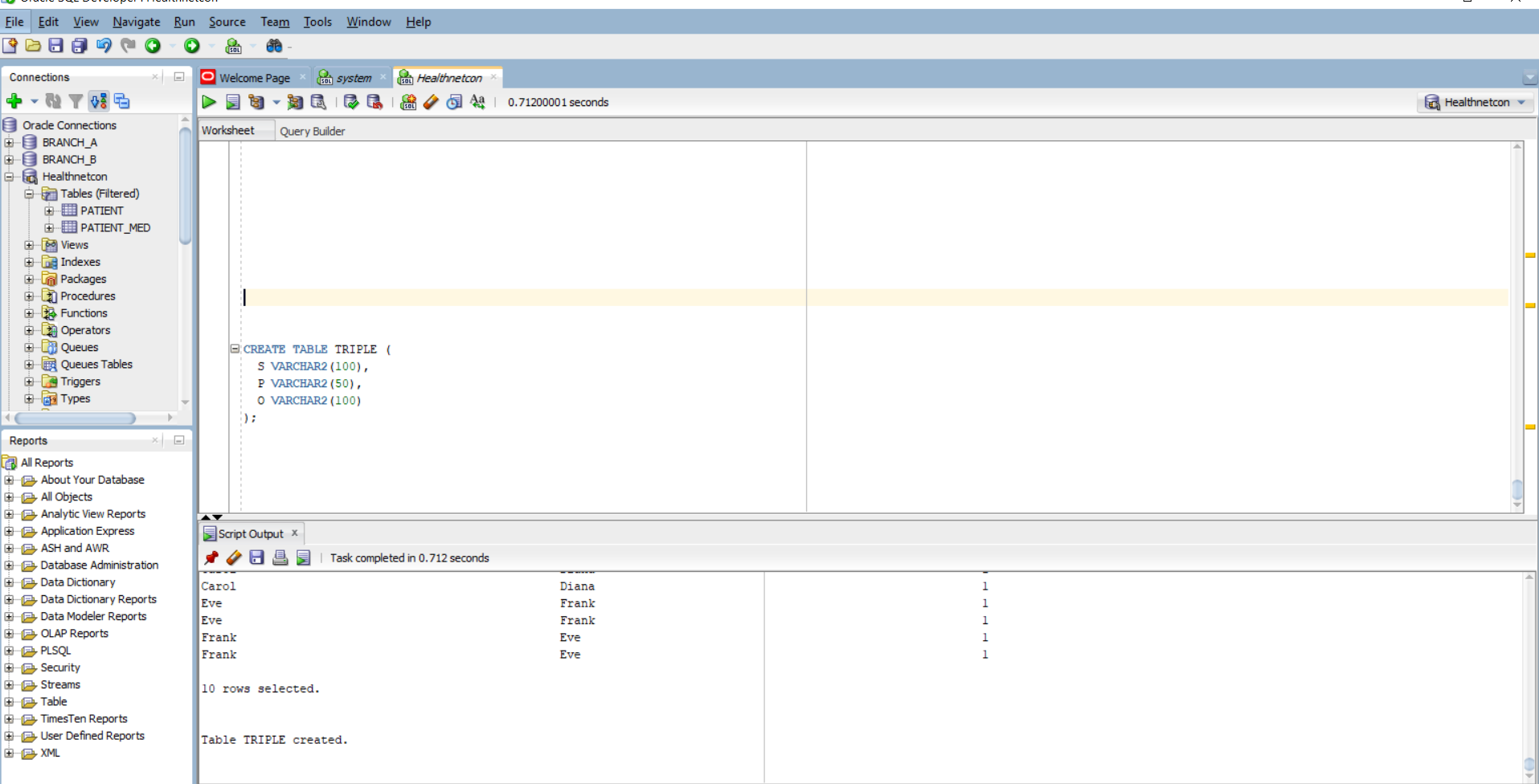
4.

CREATE TABLE TRIPLE (

S VARCHAR2(100),

P VARCHAR2(50),

O VARCHAR2(100)

);

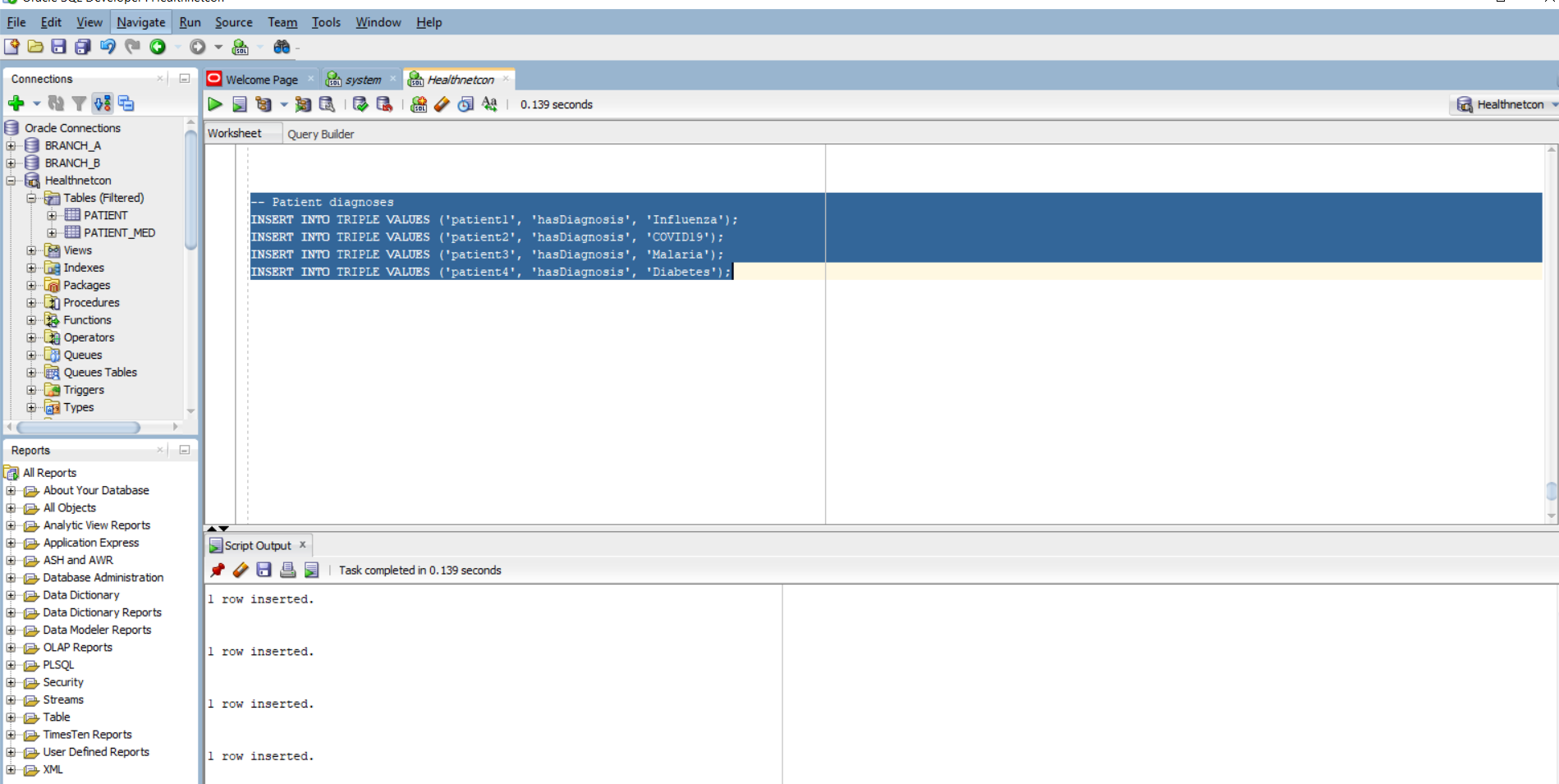
-- Patient diagnoses

INSERT INTO TRIPLE VALUES ('patient1', 'hasDiagnosis', 'Influenza');

INSERT INTO TRIPLE VALUES ('patient2', 'hasDiagnosis', 'COVID19');

INSERT INTO TRIPLE VALUES ('patient3', 'hasDiagnosis', 'Malaria');

INSERT INTO TRIPLE VALUES ('patient4', 'hasDiagnosis', 'Diabetes');



-- Taxonomy edges

INSERT INTO TRIPLE VALUES ('Influenza', 'isA', 'ViralInfection');

INSERT INTO TRIPLE VALUES ('COVID19', 'isA', 'ViralInfection');

INSERT INTO TRIPLE VALUES ('Malaria', 'isA', 'ParasiticInfection');

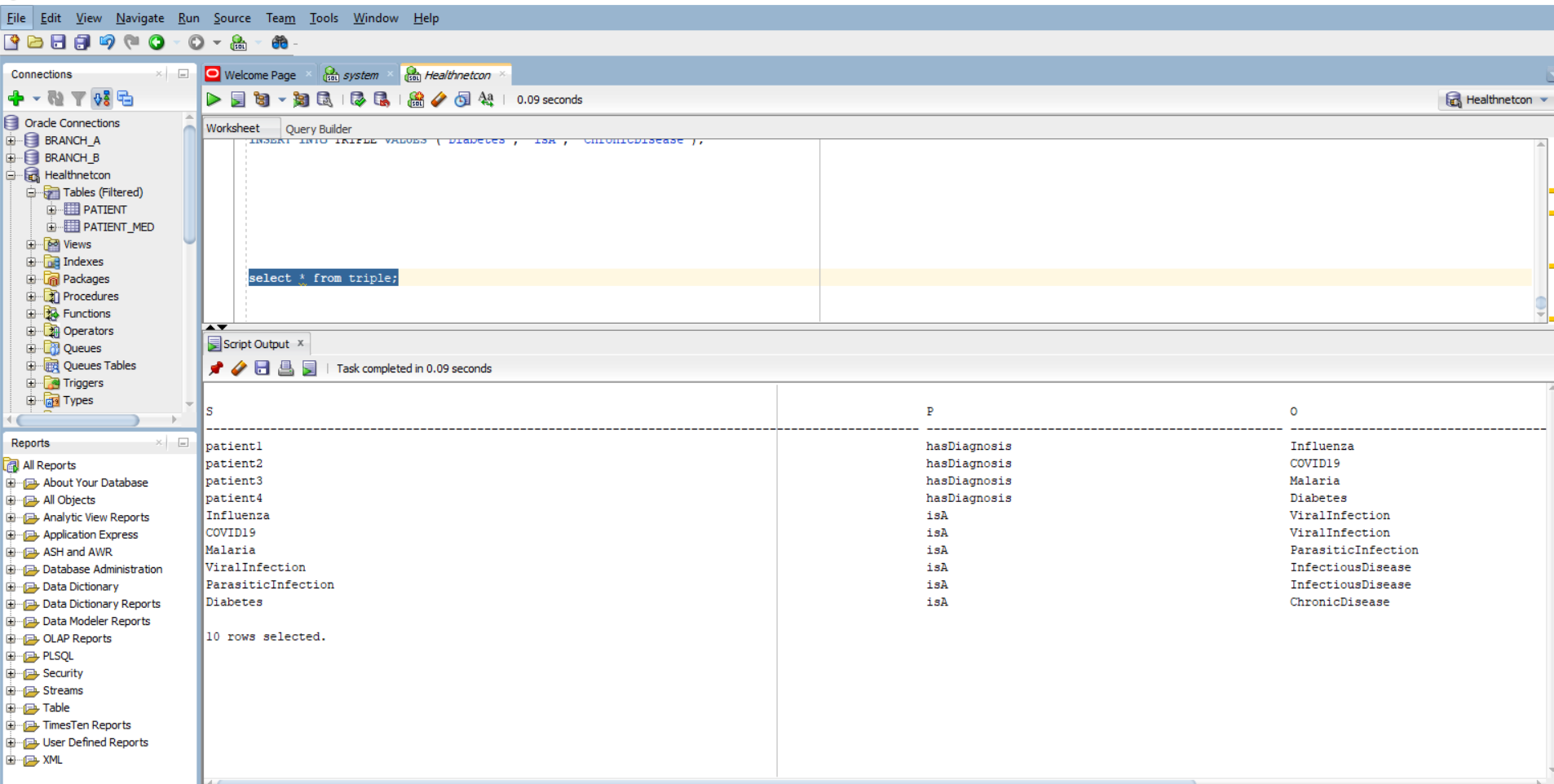
INSERT INTO TRIPLE VALUES ('ViralInfection', 'isA', 'InfectiousDisease');

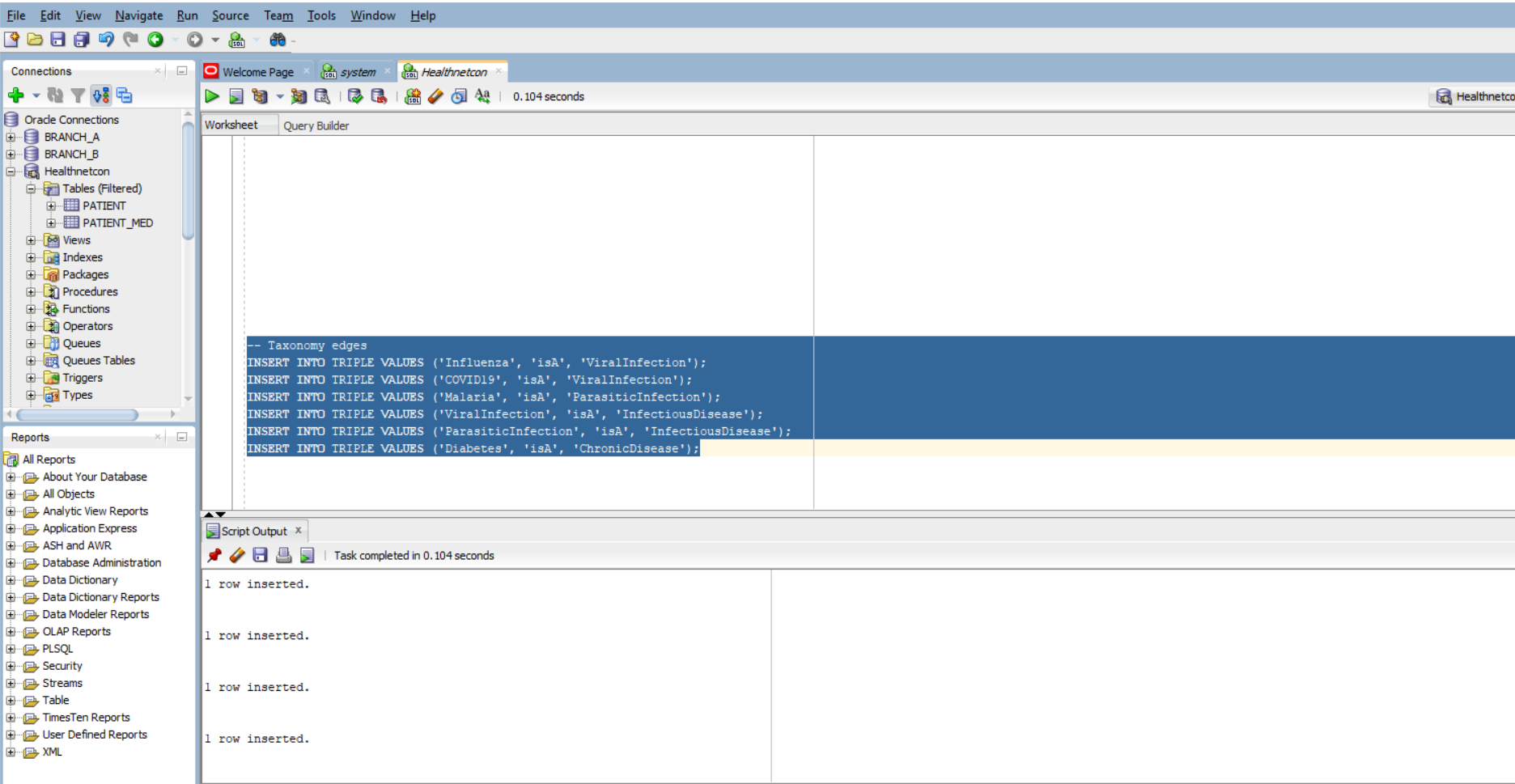
INSERT INTO TRIPLE VALUES ('ParasiticInfection', 'isA', 'InfectiousDisease');

INSERT INTO TRIPLE VALUES ('Diabetes', 'isA', 'ChronicDisease');

Check inserted rows;

select \* from triple;





WITH ISA(ANCESTOR, CHILD) AS (

-- Anchor: direct isA relationships

SELECT O, S FROM TRIPLE WHERE P = 'isA'

UNION ALL

-- Recursive: climb up the taxonomy

SELECT I.ANCESTOR, T.S

FROM TRIPLE T

JOIN ISA I ON T.P = 'isA' AND T.O = I.CHILD

),

INFECTIOUS\_PATIENTS AS (

SELECT DISTINCT T.S

FROM TRIPLE T

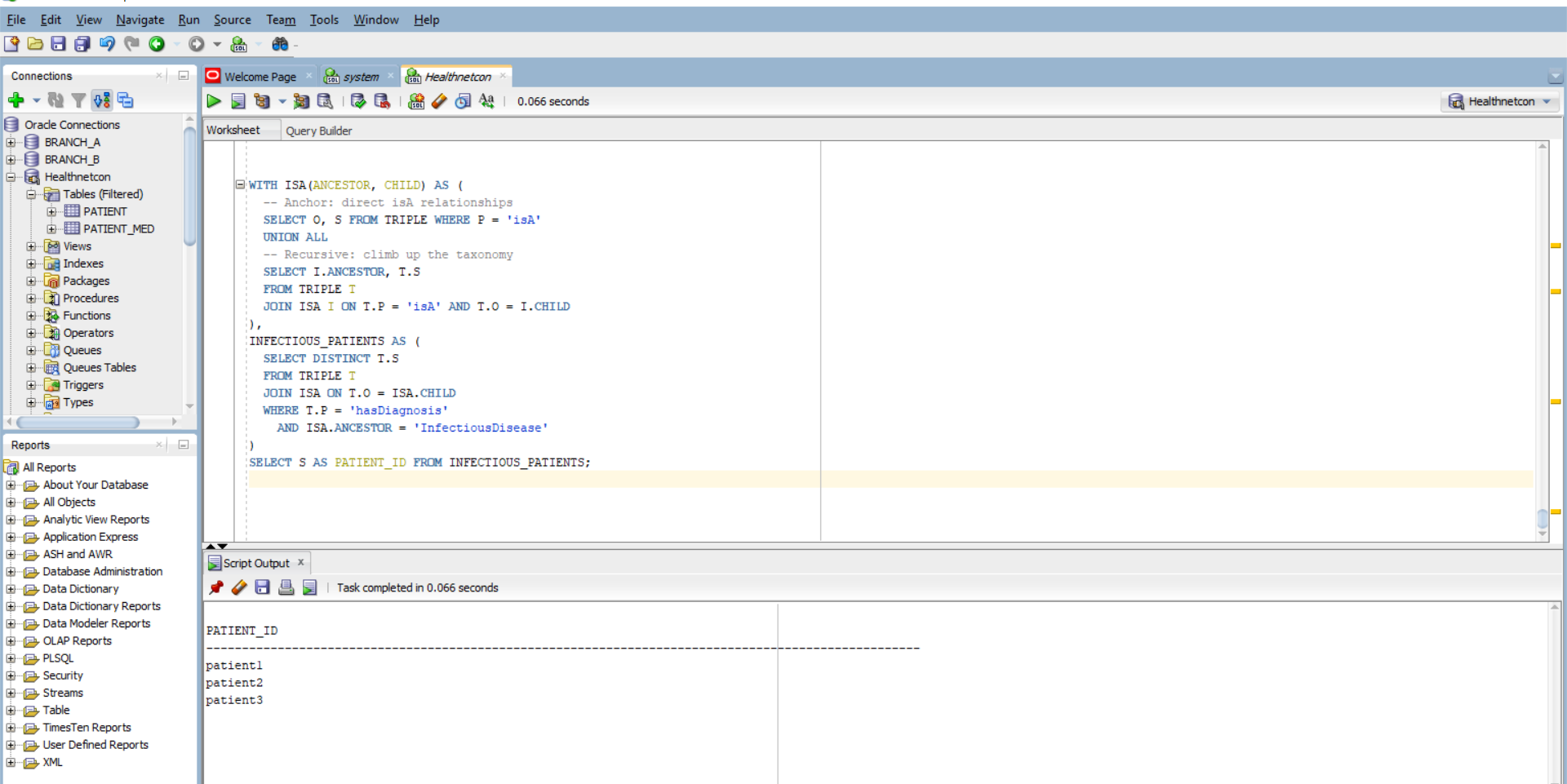
JOIN ISA ON T.O = ISA.CHILD

WHERE T.P = 'hasDiagnosis'

AND ISA.ANCESTOR = 'InfectiousDisease'

)

SELECT S AS PATIENT\_ID FROM INFECTIOUS\_PATIENTS;



 Represent facts in a flexible, searchable format

 Link concepts together (like diseases to categories)

 Enable reasoning and inference (e.g., if Influenza is an InfectiousDisease, then patient1 has an InfectiousDisease)

5.

-- Create clinic table with spatial geometry

CREATE TABLE CLINIC (

ID NUMBER PRIMARY KEY,

NAME VARCHAR2(100),

GEOM SDO\_GEOMETRY

);

INSERT INTO USER\_SDO\_GEOM\_METADATA

(TABLE\_NAME, COLUMN\_NAME, DIMINFO, SRID)

VALUES (

'CLINIC',

'GEOM',

SDO\_DIM\_ARRAY(

SDO\_DIM\_ELEMENT('Longitude', 30.0, 31.0, 0.005),

SDO\_DIM\_ELEMENT('Latitude', -2.5, -1.5, 0.005)

),

4326

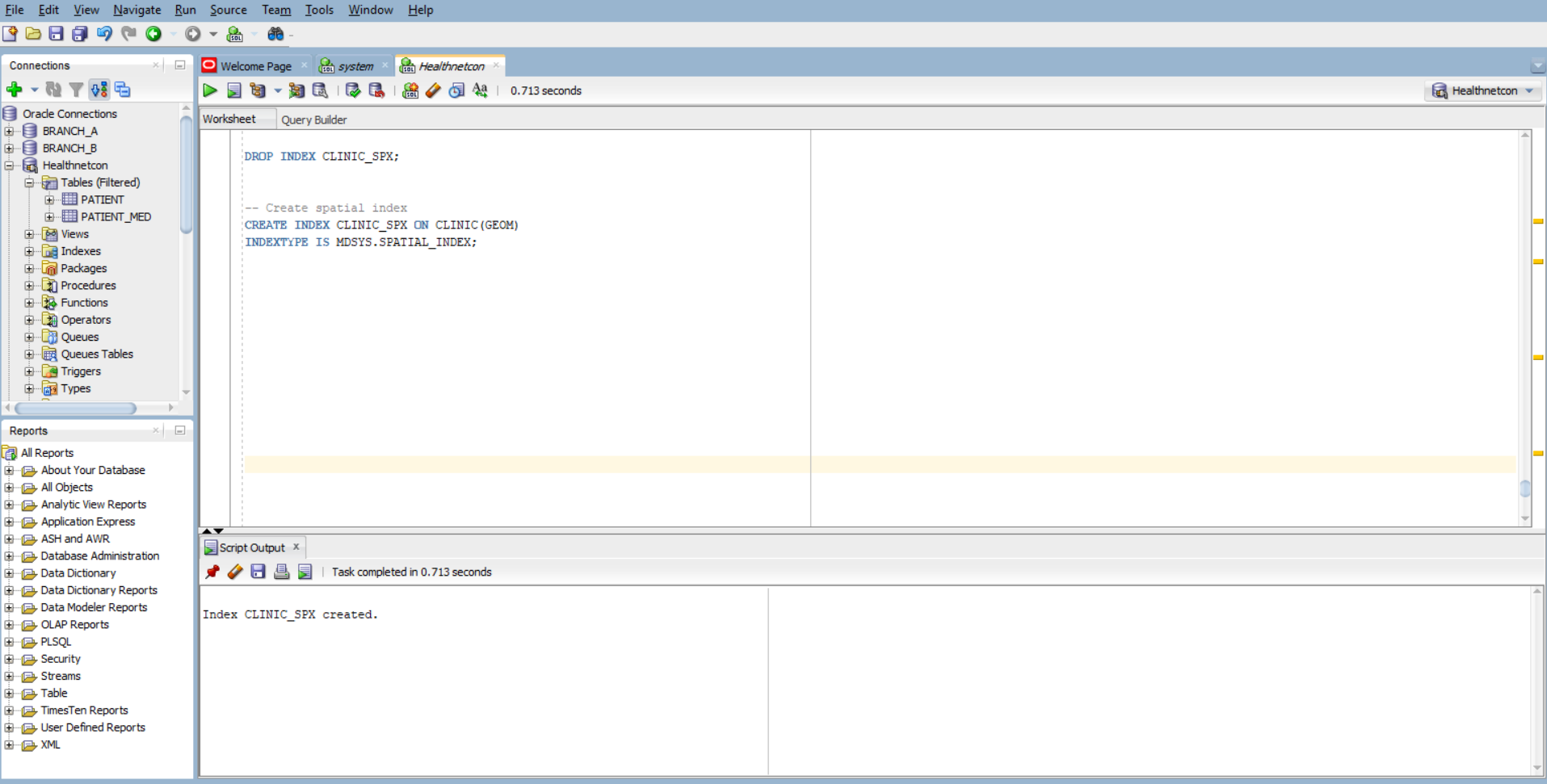
);



-- Create spatial index

CREATE INDEX CLINIC\_SPX ON CLINIC(GEOM)

INDEXTYPE IS MDSYS.SPATIAL\_INDEX;



-- Ambulance is at (30.0600, -1.9570)

INSERT INTO CLINIC VALUES (

1, 'Kigali Central Clinic',

SDO\_GEOMETRY(2001, 4326, SDO\_POINT\_TYPE(30.0610, -1.9575, NULL), NULL, NULL)

);

INSERT INTO CLINIC VALUES (

2, 'Nyamirambo Health Center',

SDO\_GEOMETRY(2001, 4326, SDO\_POINT\_TYPE(30.0595, -1.9560, NULL), NULL, NULL)

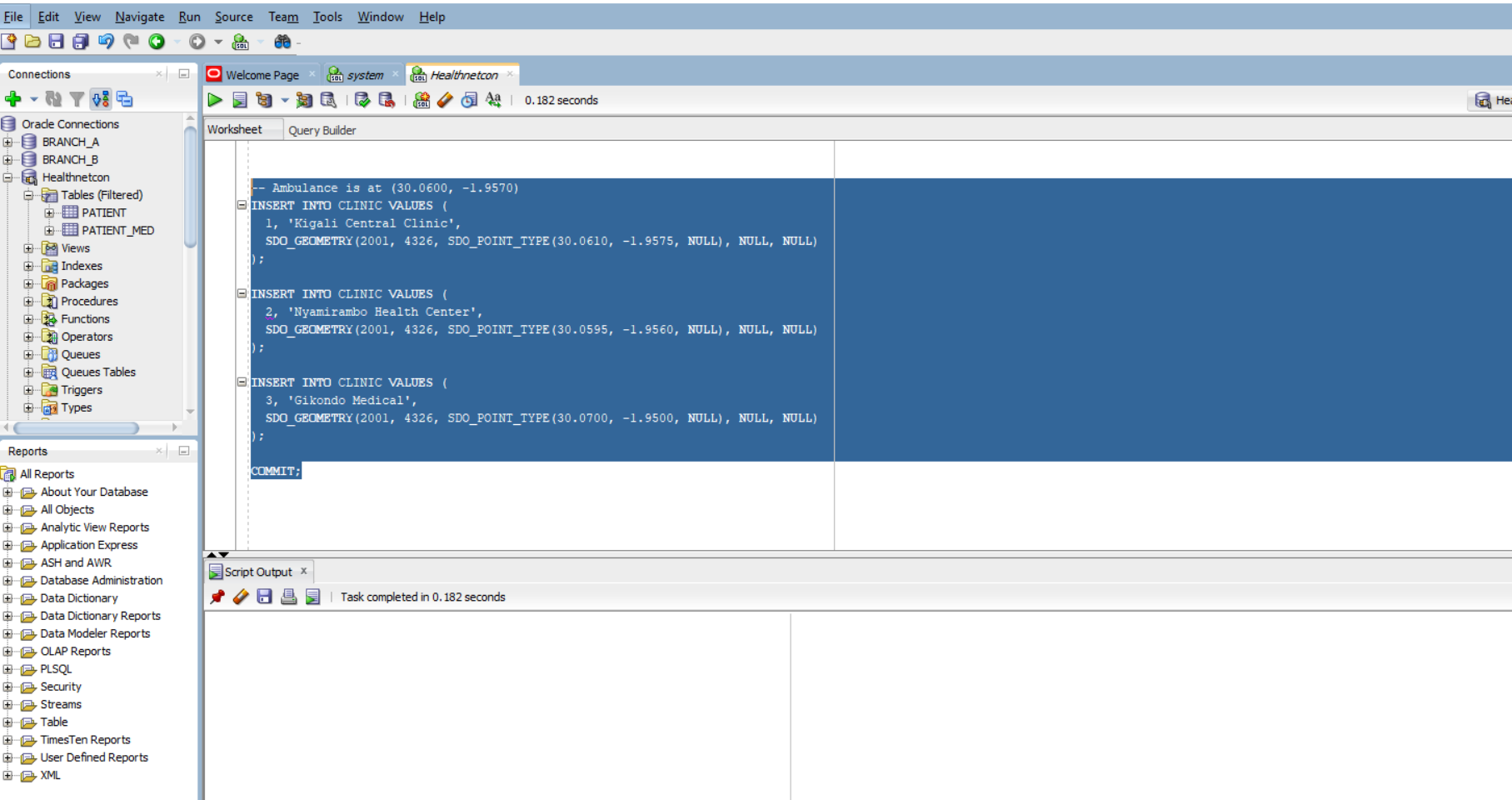
);

INSERT INTO CLINIC VALUES (

3, 'Gikondo Medical',

SDO\_GEOMETRY(2001, 4326, SDO\_POINT\_TYPE(30.0700, -1.9500, NULL), NULL, NULL)

);

COMMIT; 

SELECT C.ID, C.NAME

FROM CLINIC C

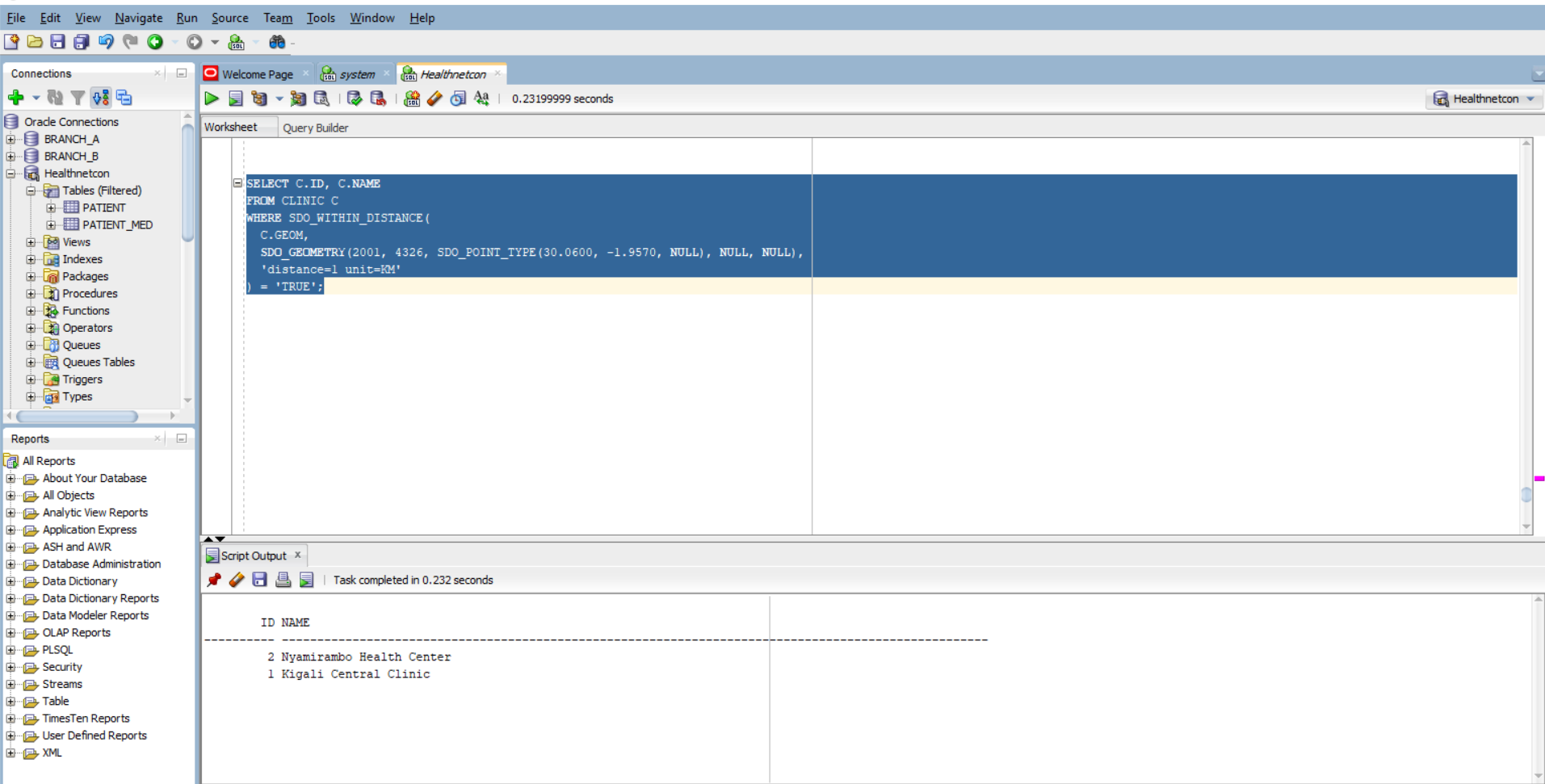
WHERE SDO\_WITHIN\_DISTANCE(

C.GEOM,

SDO\_GEOMETRY(2001, 4326, SDO\_POINT\_TYPE(30.0600, -1.9570, NULL), NULL, NULL),

'distance=1 unit=KM'

) = 'TRUE';



SELECT C.ID, C.NAME,

SDO\_GEOM.SDO\_DISTANCE(

C.GEOM,

SDO\_GEOMETRY(2001, 4326, SDO\_POINT\_TYPE(30.0600, -1.9570, NULL), NULL, NULL),

0.005,

'unit=KM'

) AS KM

FROM CLINIC C

ORDER BY KM

FETCH FIRST 3 ROWS ONLY;

