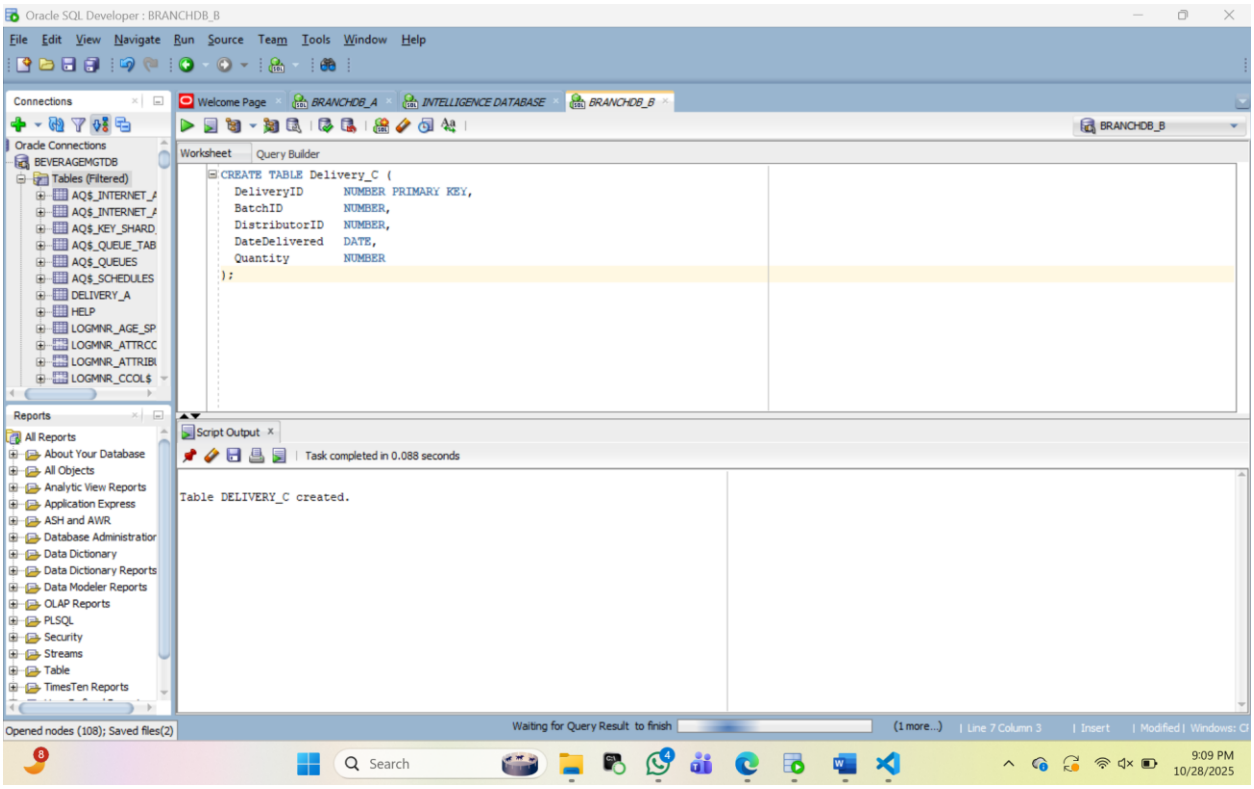
FINAL EXAMIN

A1. Fragment & Recombine Main Fact (≤10 rows)

```
CREATE TABLE Delivery_A (
  DeliveryID      NUMBER PRIMARY KEY,
  BatchID         NUMBER,
  DistributorID   NUMBER,
  DateDelivered   DATE,
  Quantity        NUMBER
);
-- On Node_B
CREATE TABLE Delivery_B (
  DeliveryID      NUMBER PRIMARY KEY,
  BatchID         NUMBER,
  DistributorID   NUMBER,
  DateDelivered   DATE,
  Quantity        NUMBER
);
```



2. Insert ≤10 Rows Total (e.g., 5 in each)

-- Node_A sample inserts

INSERT INTO Delivery_A VALUES (1001, 201, 301, SYSDATE, 500);

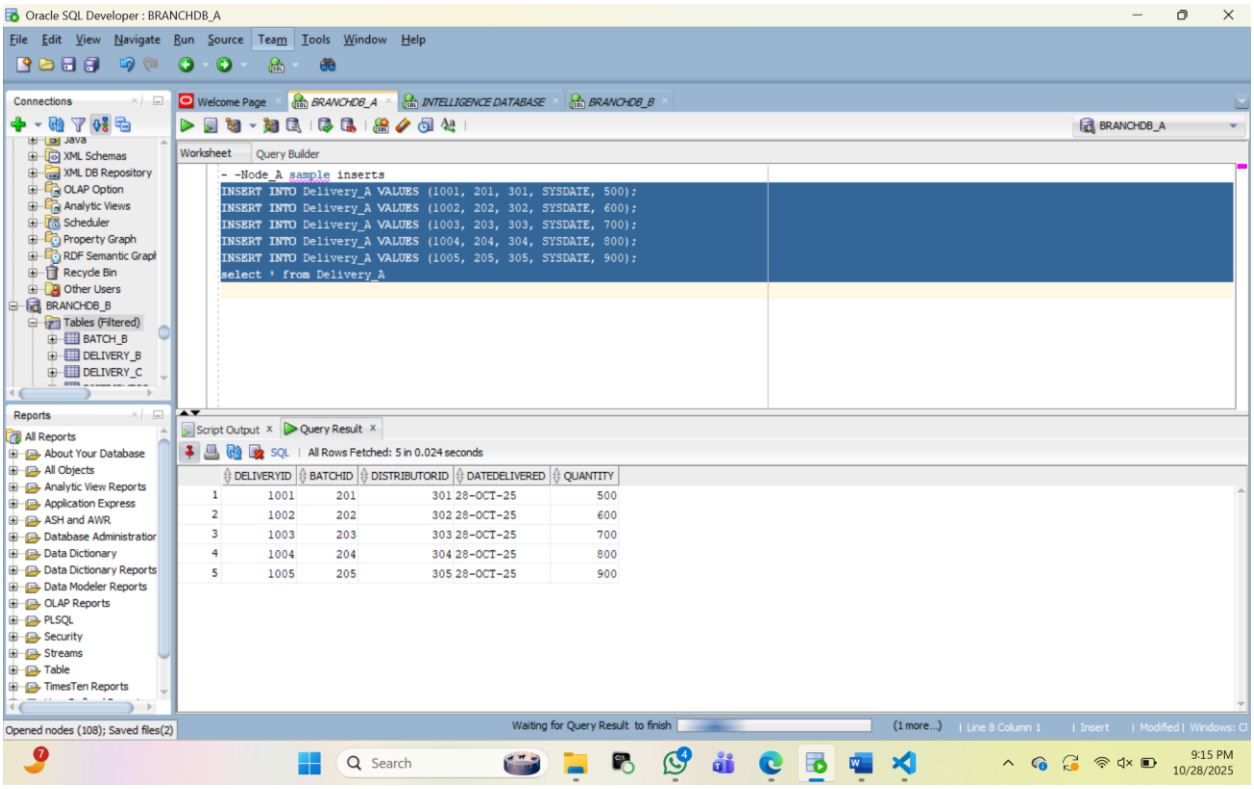
INSERT INTO Delivery_A VALUES (1002, 202, 302, SYSDATE, 600);

INSERT INTO Delivery_A VALUES (1003, 203, 303, SYSDATE, 700);

INSERT INTO Delivery_A VALUES (1004, 204, 304, SYSDATE, 800);

INSERT INTO Delivery_A VALUES (1005, 205, 305, SYSDATE, 900);

select * from Delivery_A



-- Node_B sample inserts

INSERT INTO Delivery_B VALUES (1006, 206, 306, SYSDATE, 300);

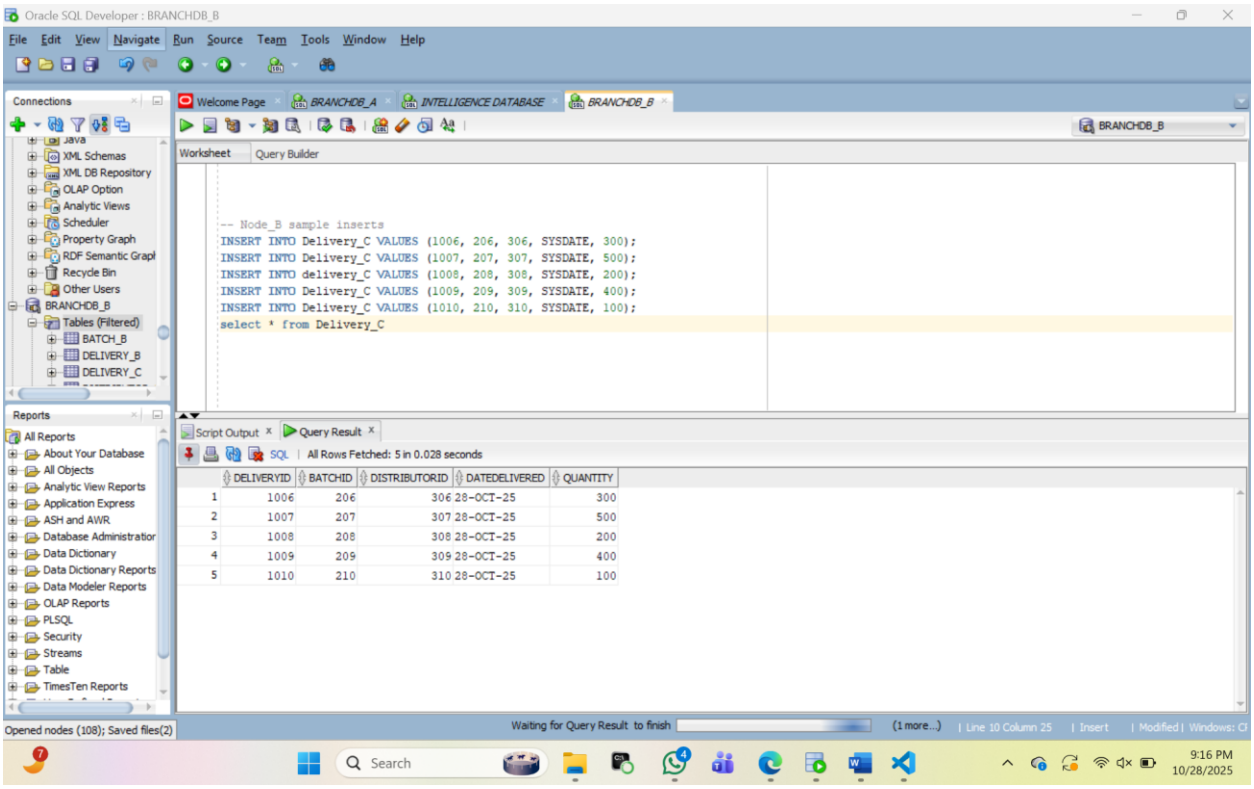
INSERT INTO Delivery_B VALUES (1007, 207, 307, SYSDATE, 500);

INSERT INTO Delivery_B VALUES (1008, 208, 308, SYSDATE, 200);

INSERT INTO Delivery_B VALUES (1009, 209, 309, SYSDATE, 400);

INSERT INTO Delivery_B VALUES (1010, 210, 310, SYSDATE, 100);

select * from Delivery_b



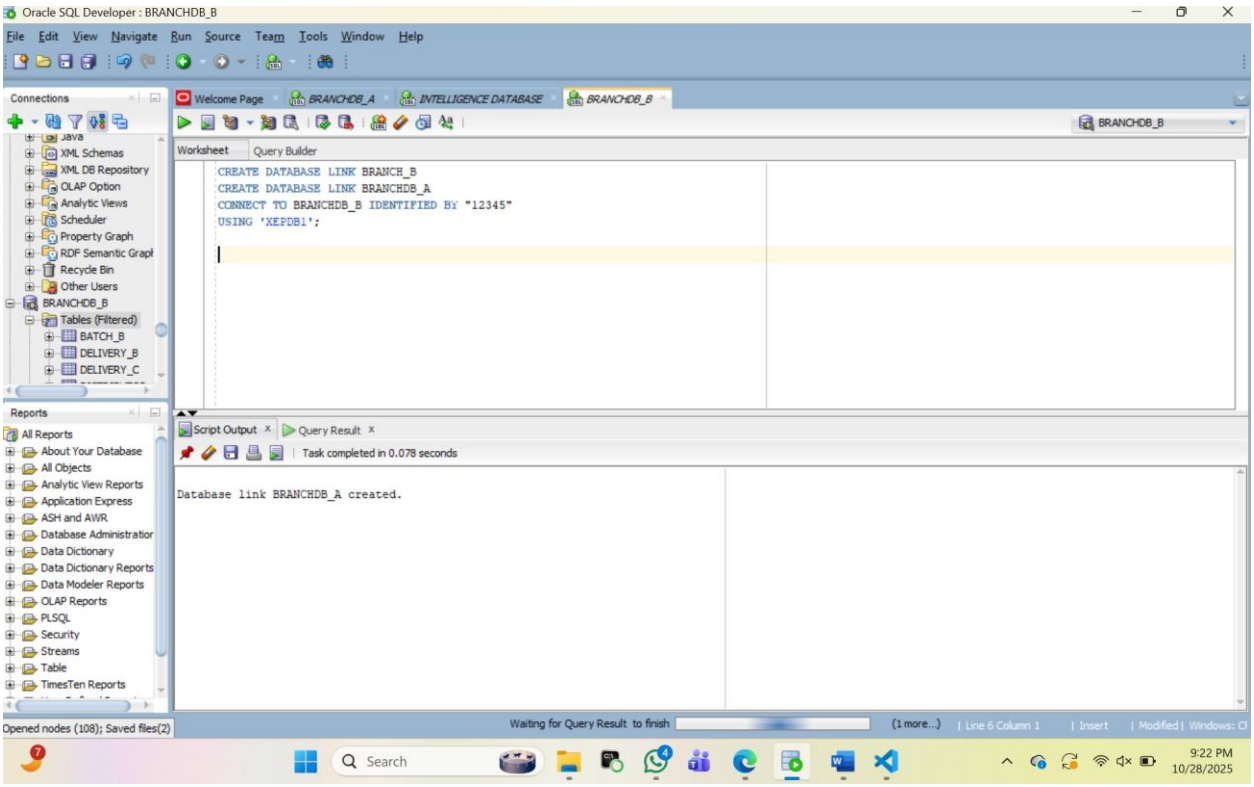
-3. Create a Database Link from Node_A to Node_B

CREATE DATABASE LINK BRANCH_B

CREATE DATABASE LINK BRANCHDB_A

CONNECT TO BRANCHDB_B IDENTIFIED BY "12345"

USING 'XEPDB1';



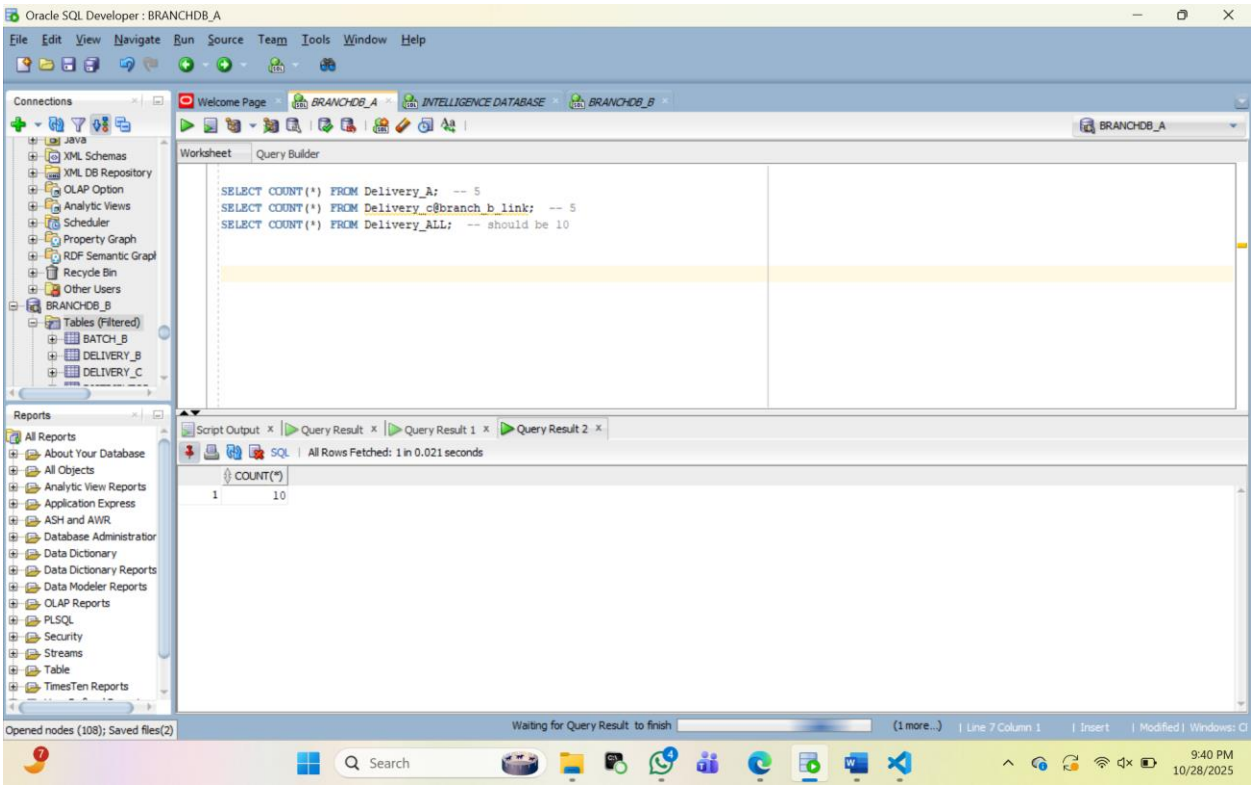
-- Validate with COUNT and Checksum

--Count Check

SELECT COUNT(*) FROM Delivery_A; -- 5

SELECT COUNT(*) FROM Delivery_B@proj_link; -- 5

SELECT COUNT(*) FROM Delivery_ALL; -- should be 10

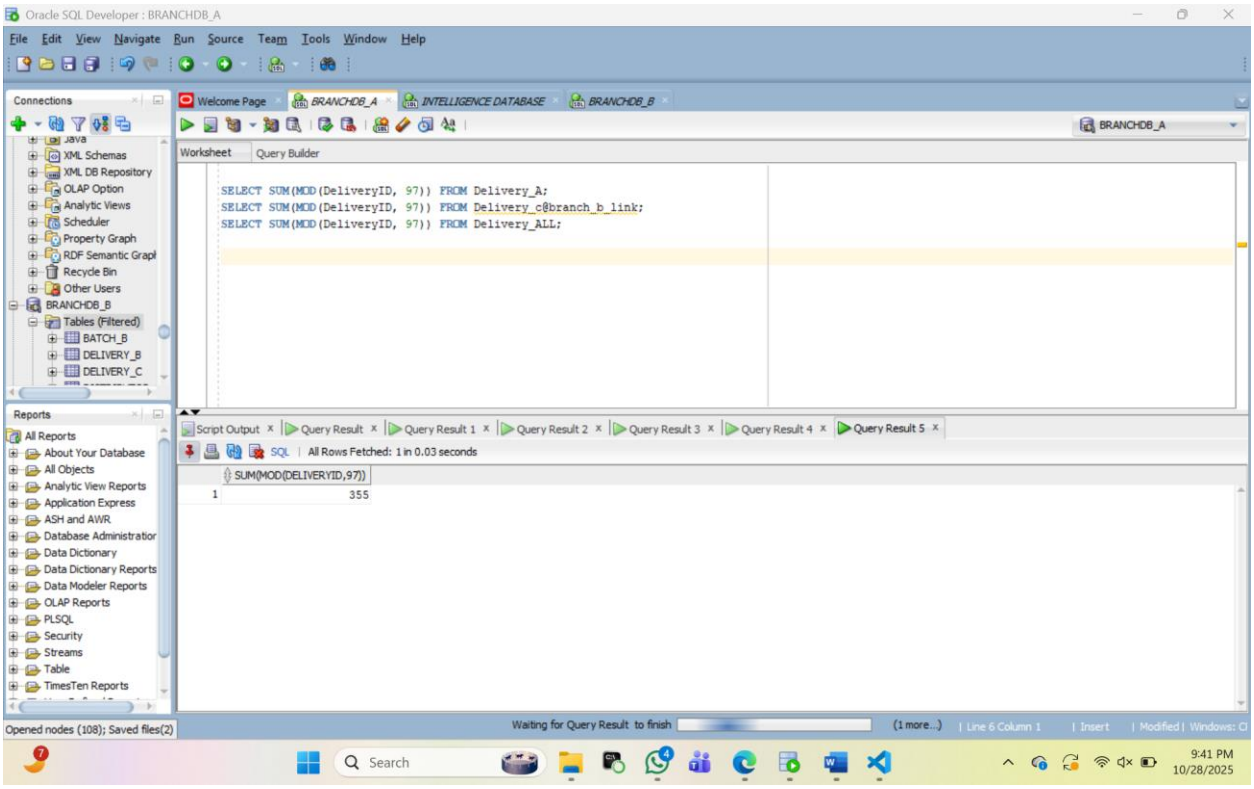


--Checksum Validation

SELECT SUM(MOD(DeliveryID, 97)) FROM Delivery_A;

SELECT SUM(MOD(DeliveryID, 97)) FROM Delivery_B@proj_link;

SELECT SUM(MOD(DeliveryID, 97)) FROM Delivery_ALL;

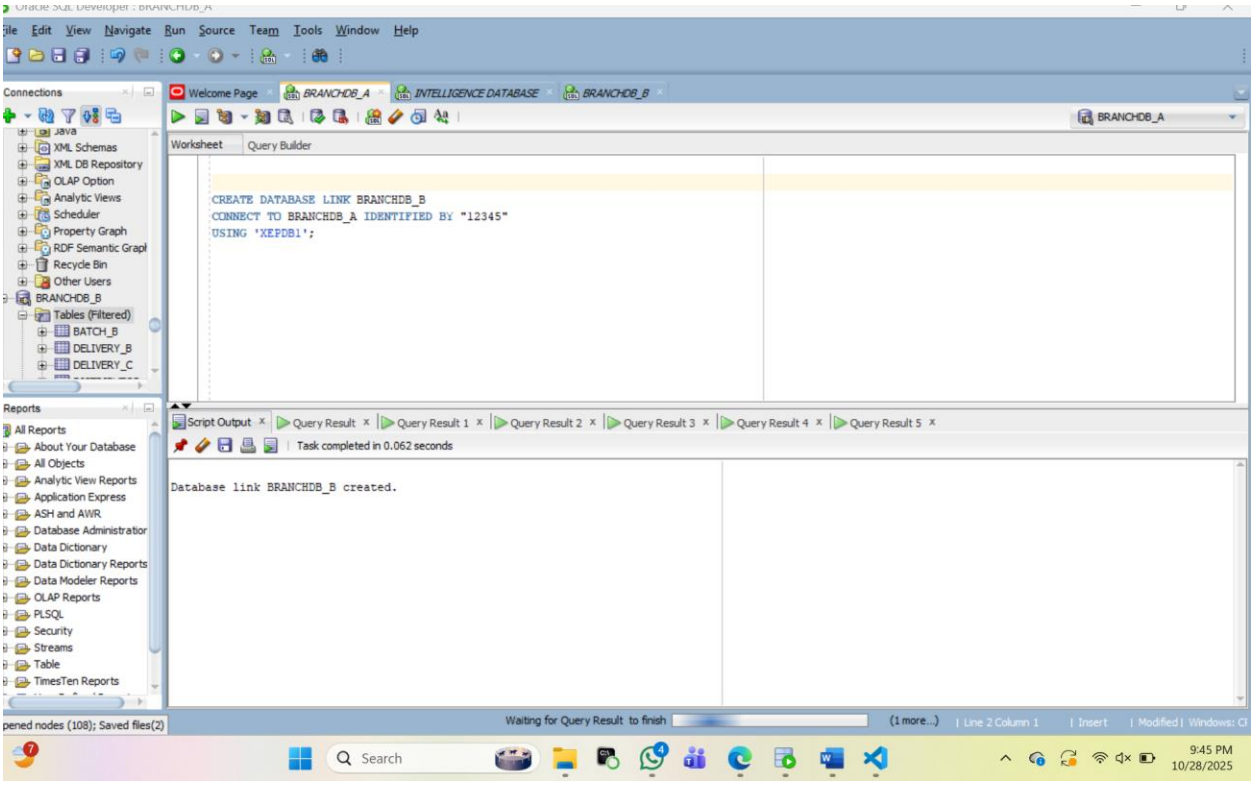


A2.1. Create Database Link from Node_B to Node_A

CREATE DATABASE LINK BRANCHDB_B

CONNECT TO BRANCHDB_A IDENTIFIED BY "12345"

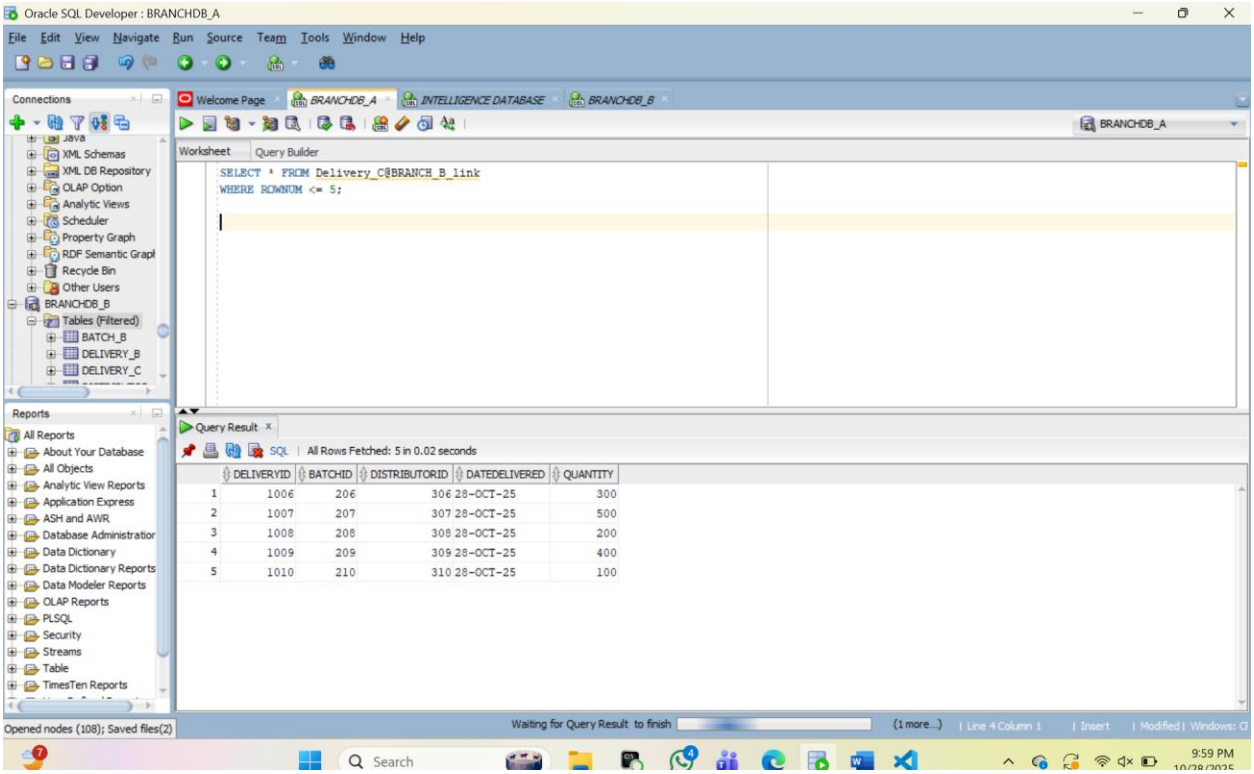
USING 'XEPDB1';



2.Run a Remote SELECT (Doctor@proj_link → Distributor@proj_link)

SELECT * FROM Delivery_C@BRANCH_B_link

WHERE ROWNUM <= 5;



--3. Run a Distributed Join

SELECT D.DeliveryID, D.DateDelivered, I.TotalAmount

FROM Delivery_A D

JOIN Invoice@proj_link I ON D.DeliveryID = I.DeliveryID

WHERE ROWNUM <= 10;

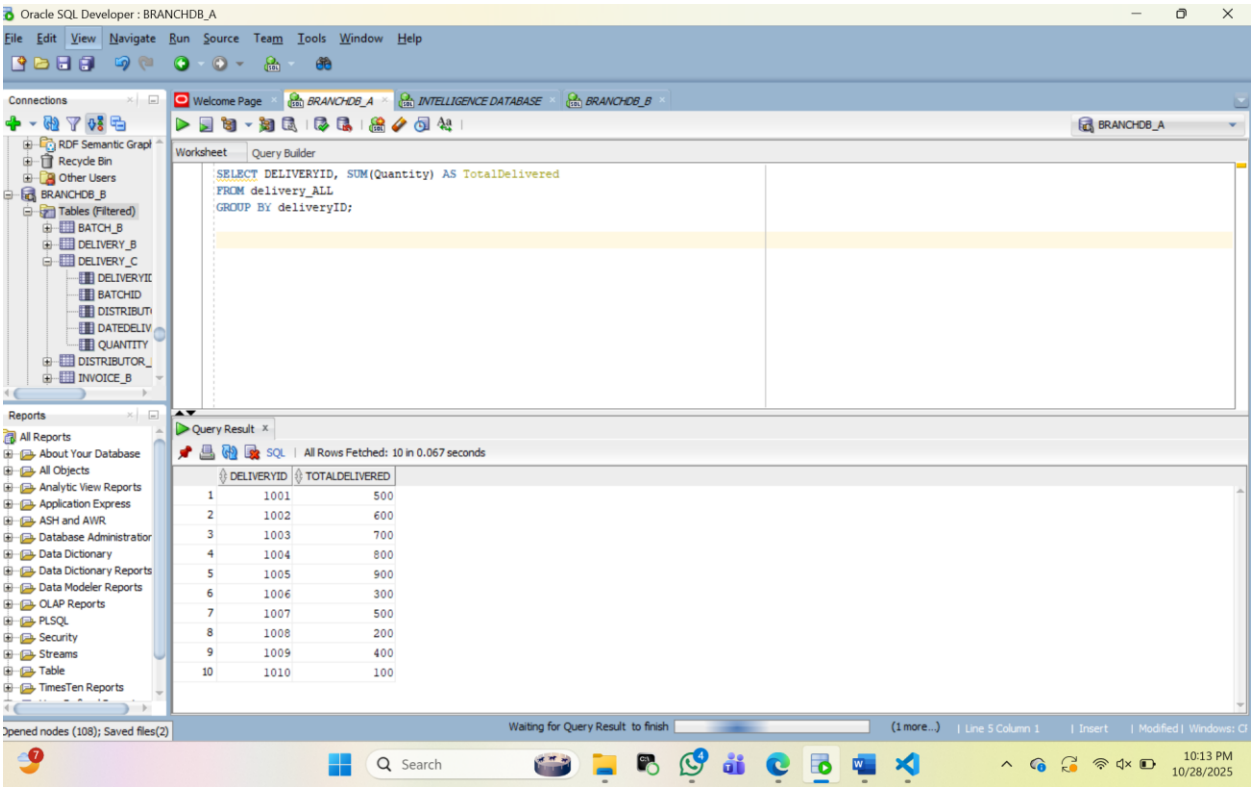
-A3: Parallel vs Serial Aggregation (≤10 rows data)

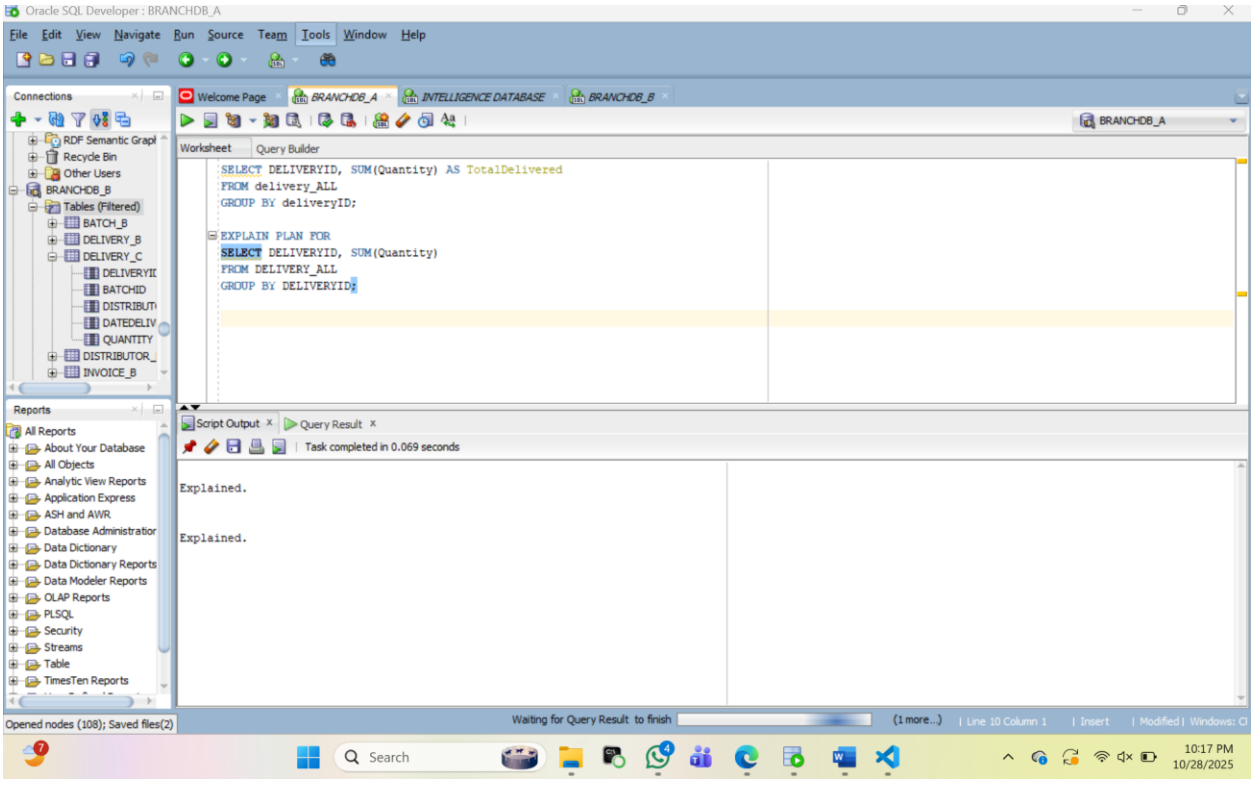
--1. SERIAL Aggregation (No Hint)

SELECT DELIVERYID, SUM(Quantity) AS TotalDelivered

FROM delivery_ALL

GROUP BY deliveryID;





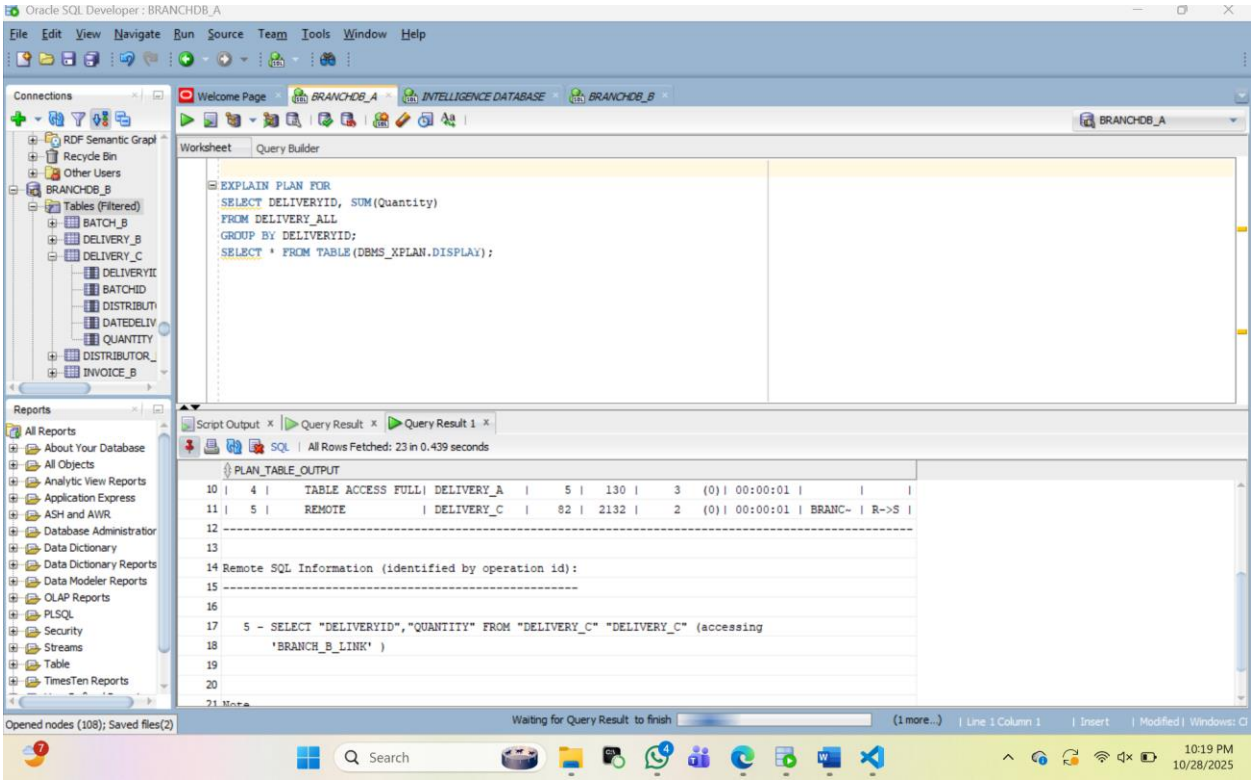
EXPLAIN PLAN FOR

SELECT DELIVERYID, SUM(Quantity)

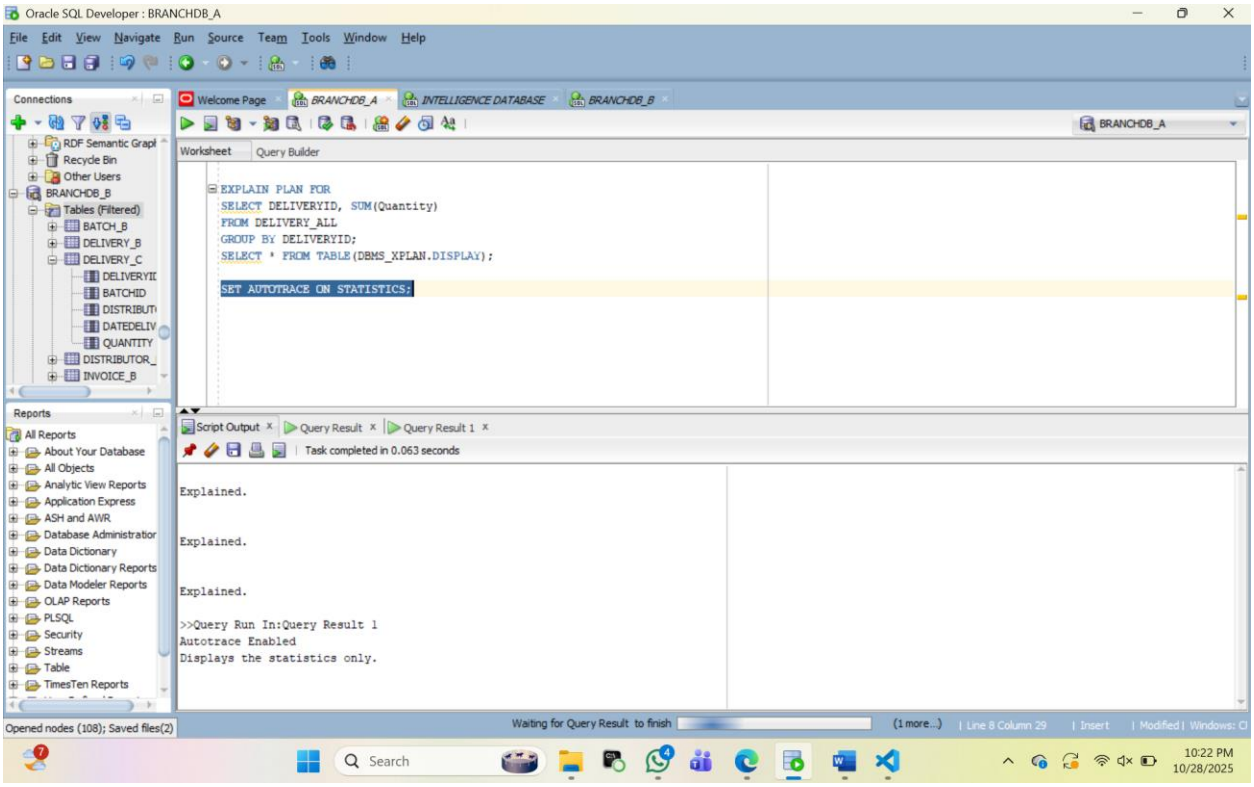
FROM DELIVERY_ALL

GROUP BY DELIVERYID;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

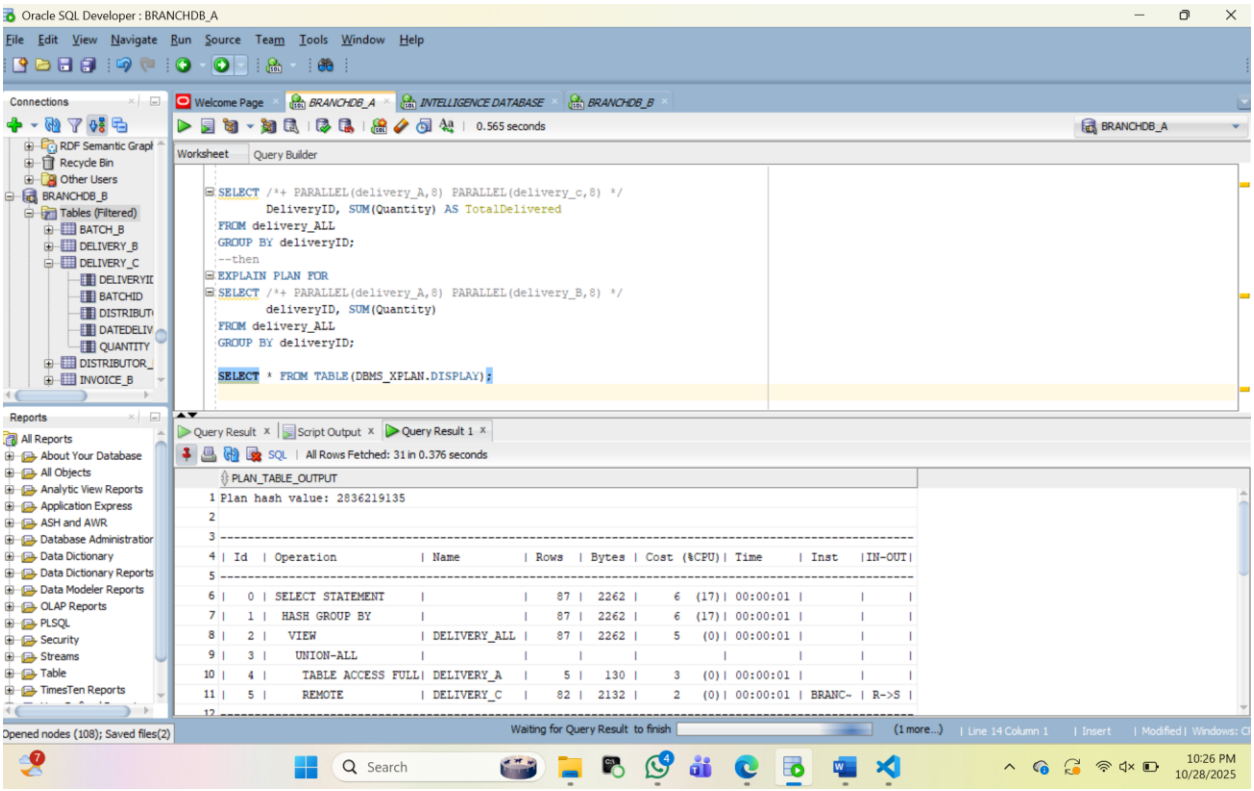


--Enable AUTOTRACE (in SQL*Plus or SQL Developer):



--2. PARALLEL Aggregation (With Hint)

```
SELECT /*+ PARALLEL(delivery_A,8) PARALLEL(delivery_c,8) */
      DeliveryID, SUM(Quantity) AS TotalDelivered
FROM delivery_ALL
GROUP BY deliveryID;
--then
EXPLAIN PLAN FOR
SELECT /*+ PARALLEL(delivery_A,8) PARALLEL(delivery_B,8) */
      deliveryID, SUM(Quantity)
FROM delivery_ALL
GROUP BY deliveryID;


SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```



--A4.1: PL/SQL Block for 2PC (One Local + One Remote Insert)

BEGIN

-- Local insert on Node_A

INSERT INTO Delivery_A (DeliveryID, BatchID, DistributorID, DateDelivered, Quantity)

VALUES (1011, 211, 311, SYSDATE, 1500);

-- Remote insert on Node_B via DB link

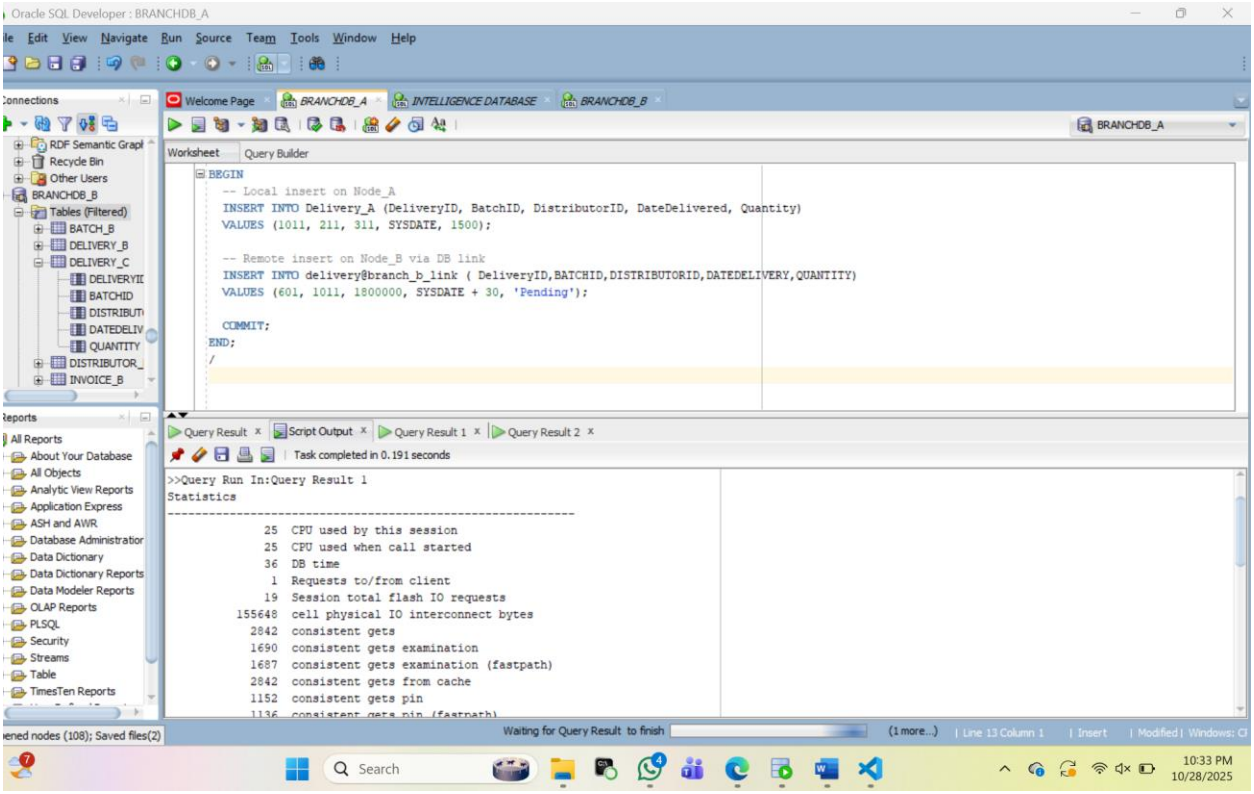
INSERT INTO delivery@branch_b_link (
DeliveryID,BATCHID,DISTRIBUTORID,DATEDELIVERY,QUANTITY)

VALUES (601, 1011, 1800000, SYSDATE + 30, 'Pending');

COMMIT;

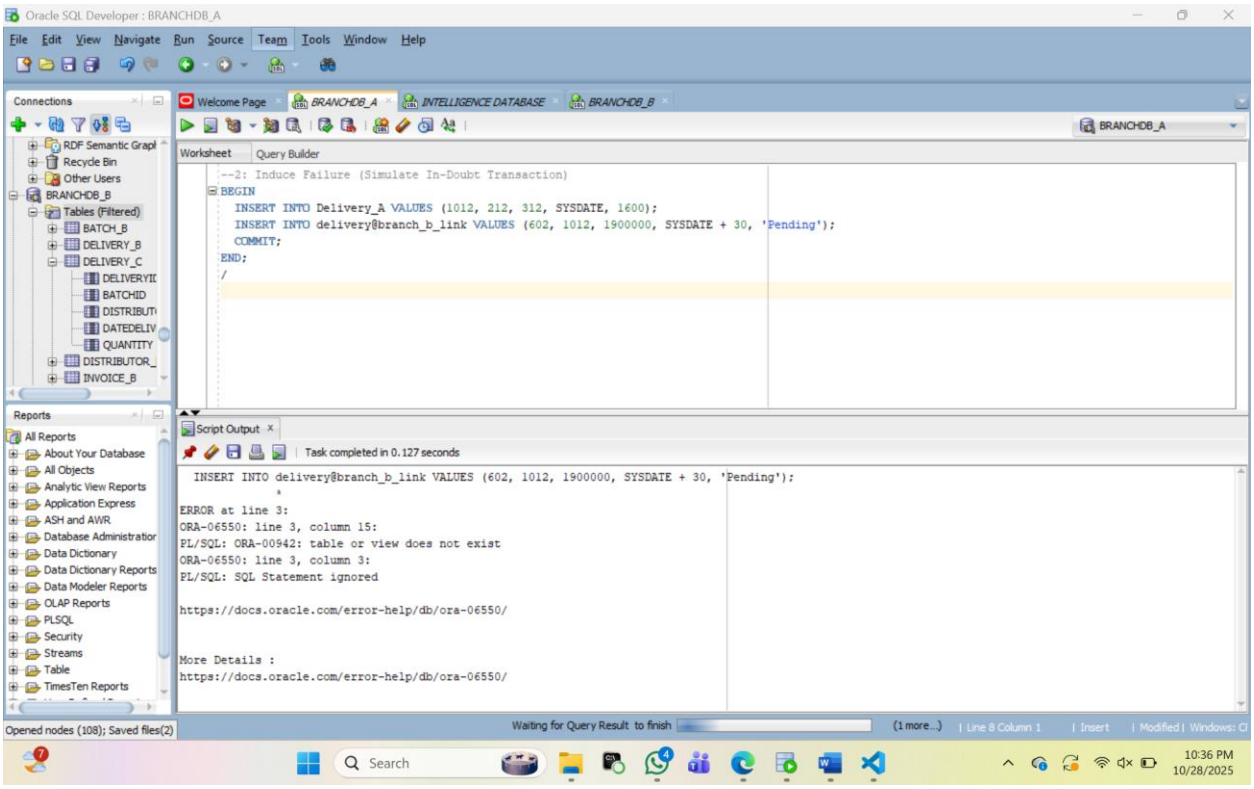
END;

/



--2: Induce Failure (Simulate In-Doubt Transaction)

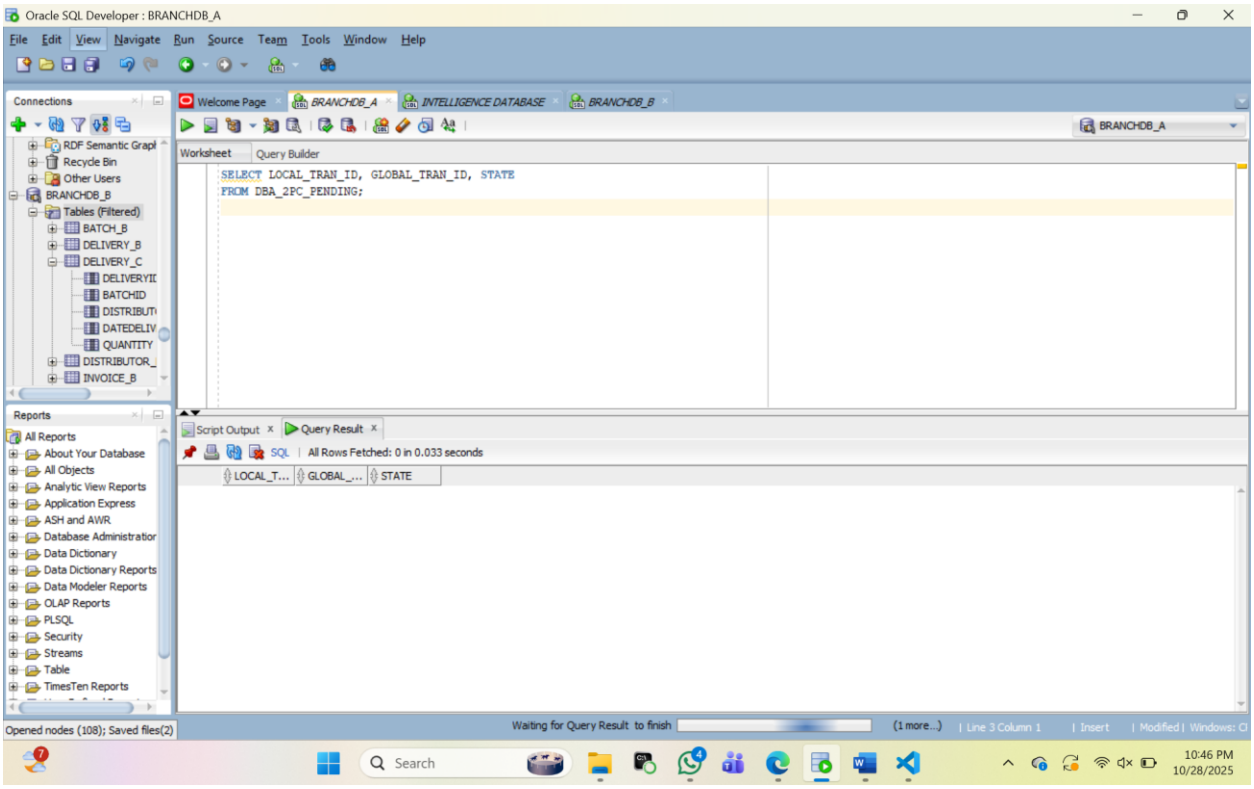
```
BEGIN
  INSERT INTO Delivery_A VALUES (1012, 212, 312, SYSDATE, 1600);
  INSERT INTO delivery@branch_b_link VALUES (602, 1012, 1900000, SYSDATE + 30, 'Pending');
  COMMIT;
END;
/
```



3.It return no rows beacause the transaction went through: Query DBA_2PC_PENDING

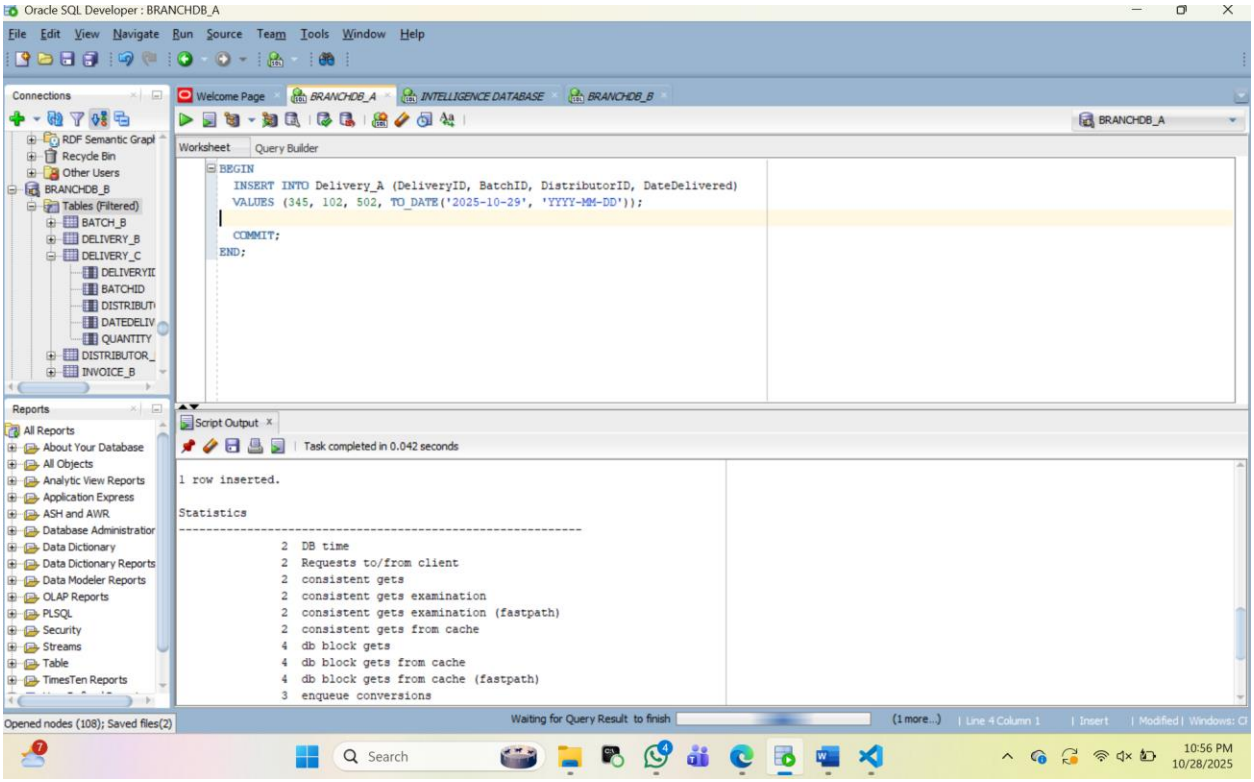
SELECT LOCAL_TRAN_ID, GLOBAL_TRAN_ID, STATE

FROM DBA_2PC_PENDING;



4. No need to roll back since no transaction is hanging

5. inserted row



A5: Distributed Lock Conflict & Diagnosis (no extra rows)

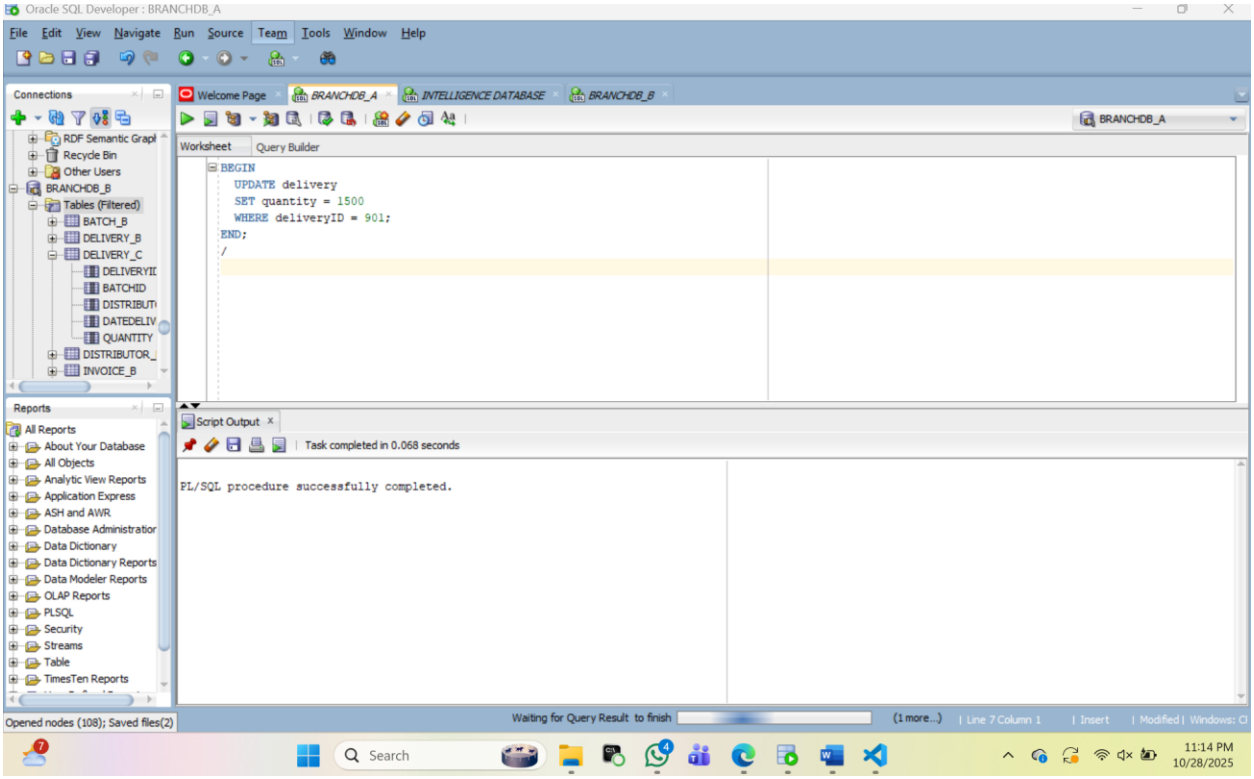
```
BEGIN
  UPDATE delivery
  SET quantity= 600
  WHERE deliverID =1002 ;

  -- You can keep the transaction open by not committing
END;
/
```

2.

```
BEGIN
  UPDATE delivery
  SET quantity = 1500
  WHERE deliveryID = 901;
```

END;

/

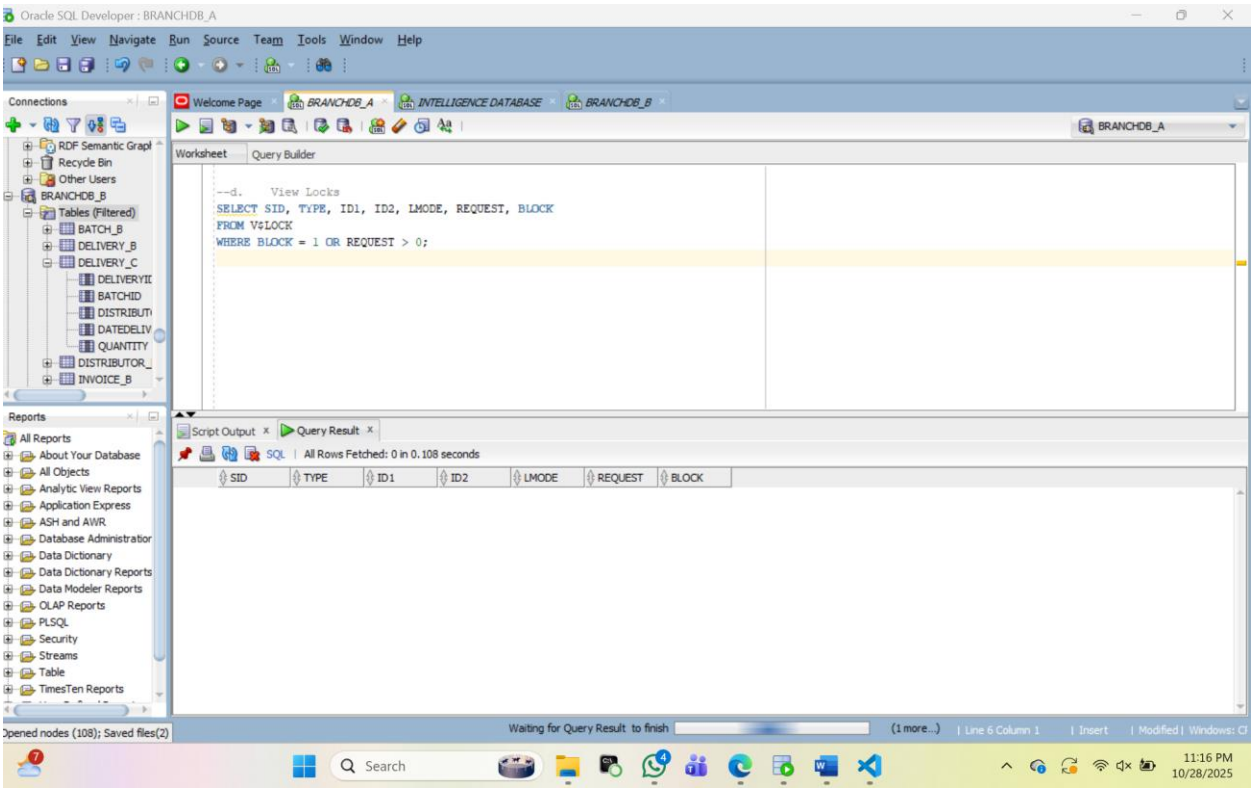


3. View Locks

SELECT SID, TYPE, ID1, ID2, LMODE, REQUEST, BLOCK

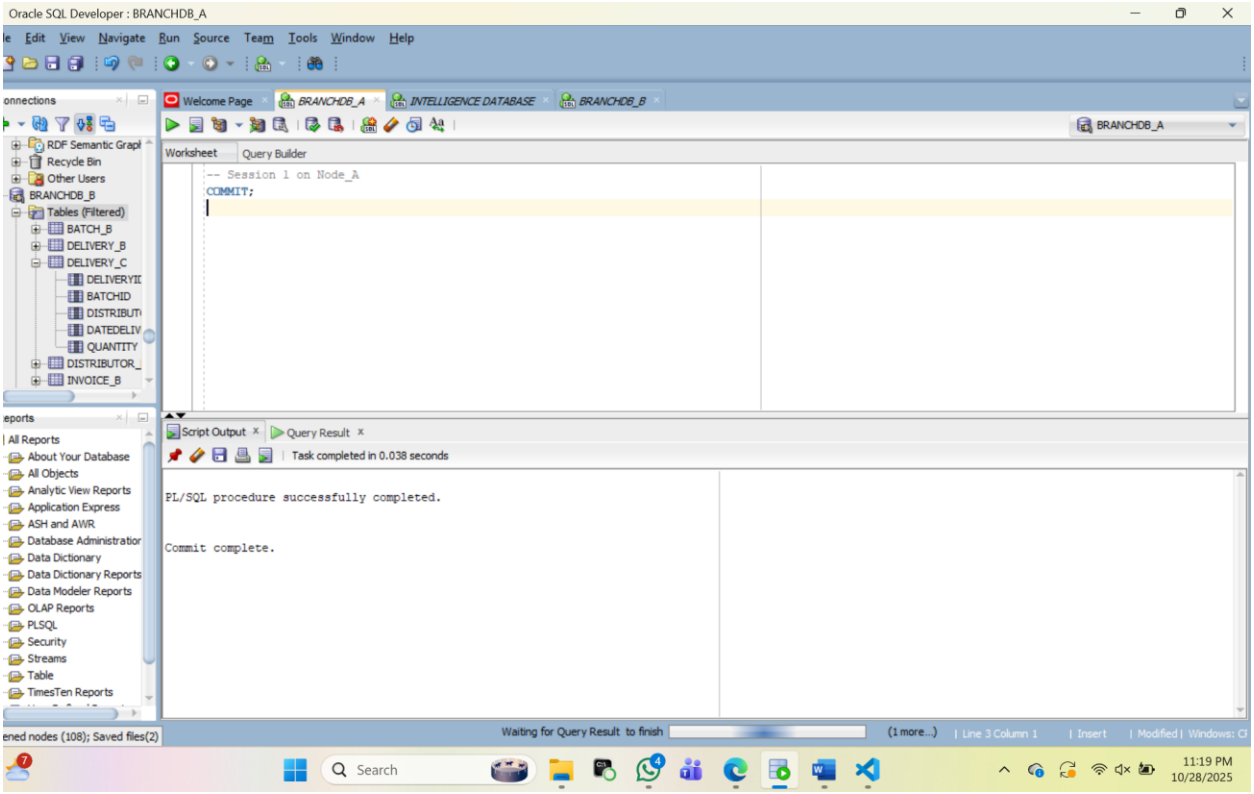
FROM V$LOCK

WHERE BLOCK = 1 OR REQUEST > 0;



4. Release the lock

COMMIT;



SECTION B:

B6: Declarative Rules Hardening (≤10 committed rows)

1. Add/Verify Constraints

Add constraints to Project table:

```
ALTER TABLE Payment
MODIFY (
  PaymentID NUMBER NOT NULL,
  InvoiceID NUMBER NOT NULL,
  Amount NUMBER NOT NULL,
  PaymentDate DATE NOT NULL,
  Method VARCHAR2(20) NOT NULL
);


-- Ensure positive amount and valid method
```

ALTER TABLE Payment ADD CONSTRAINT chk_payment_amount_positive

CHECK (Amount > 0);

ALTER TABLE Payment ADD CONSTRAINT chk_payment_method_valid

CHECK (Method IN ('CASH', 'CARD', 'TRANSFER'));

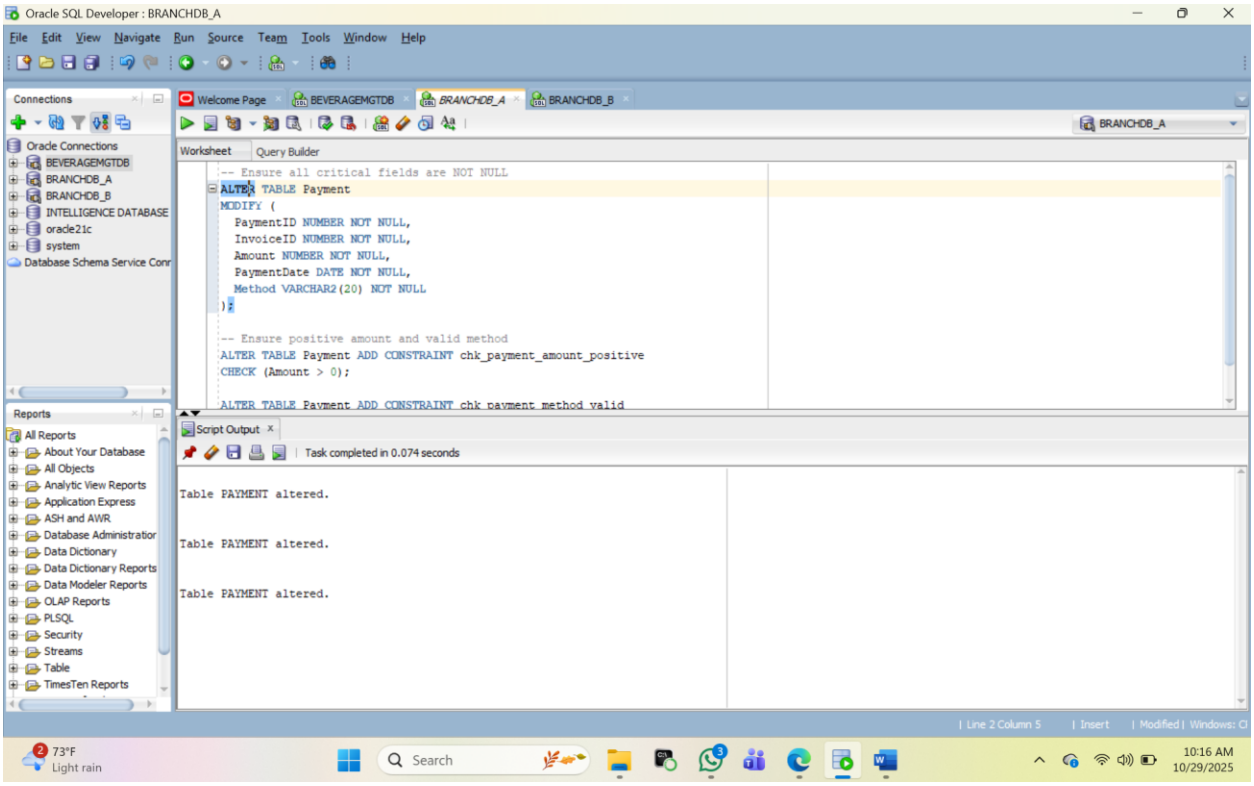
-Alter table invoice

ALTER TABLE Invoice ADD CONSTRAINT chk_invoice_amount_positive

CHECK (TotalAmount > 0);

ALTER TABLE Invoice ADD CONSTRAINT chk_invoice_status_valid

CHECK (Status IN ('PENDING', 'PAID', 'OVERDUE')) NOVALIDATE;



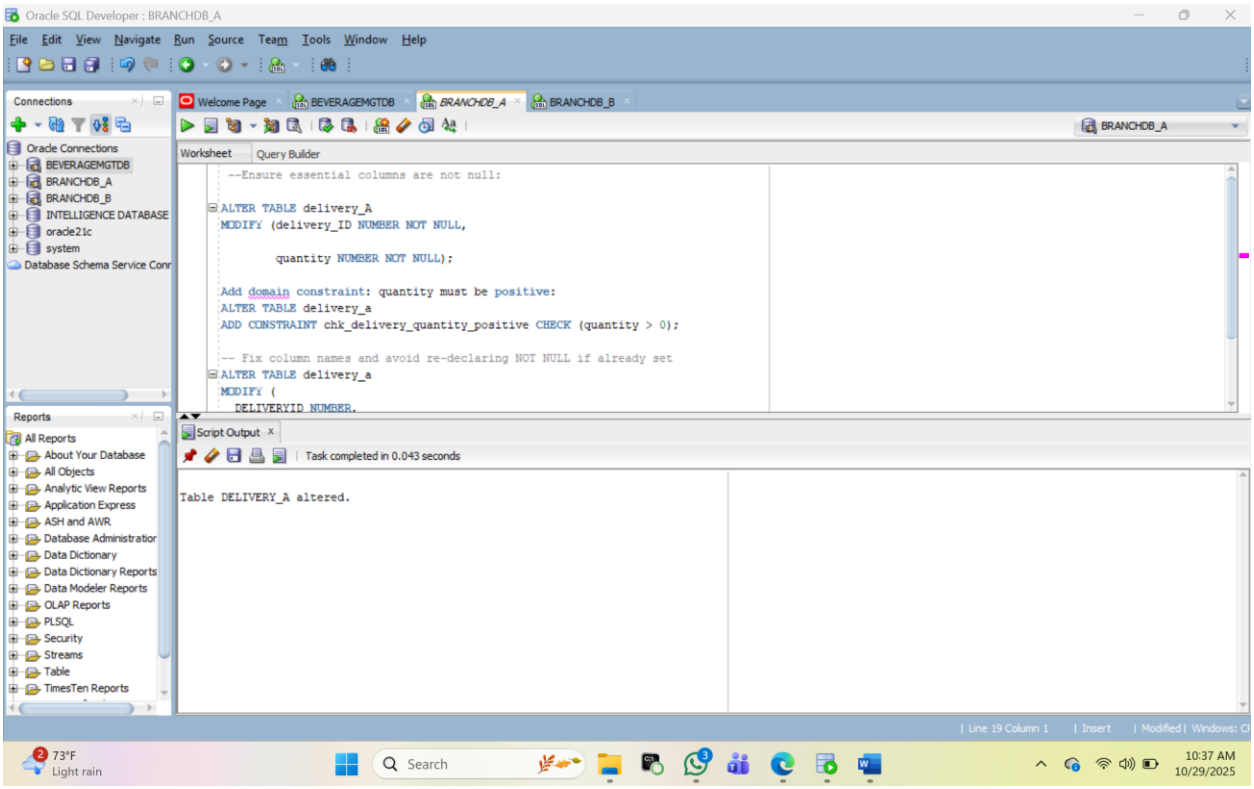
--Ensure essential columns are not null:

ALTER TABLE delivery_A

```
MODIFY (delivery_ID NUMBER NOT NULL,

      quantity NUMBER NOT NULL);
```

Add domain constraint: quantity must be positive:

```
ALTER TABLE delivery_a
ADD CONSTRAINT chk_delivery_quantity_positive CHECK (quantity > 0);

-- Fix column names and avoid re-declaring NOT NULL if already set
ALTER TABLE delivery_a
MODIFY (
  DELIVERYID NUMBER,

quantity NUMBER
);
```



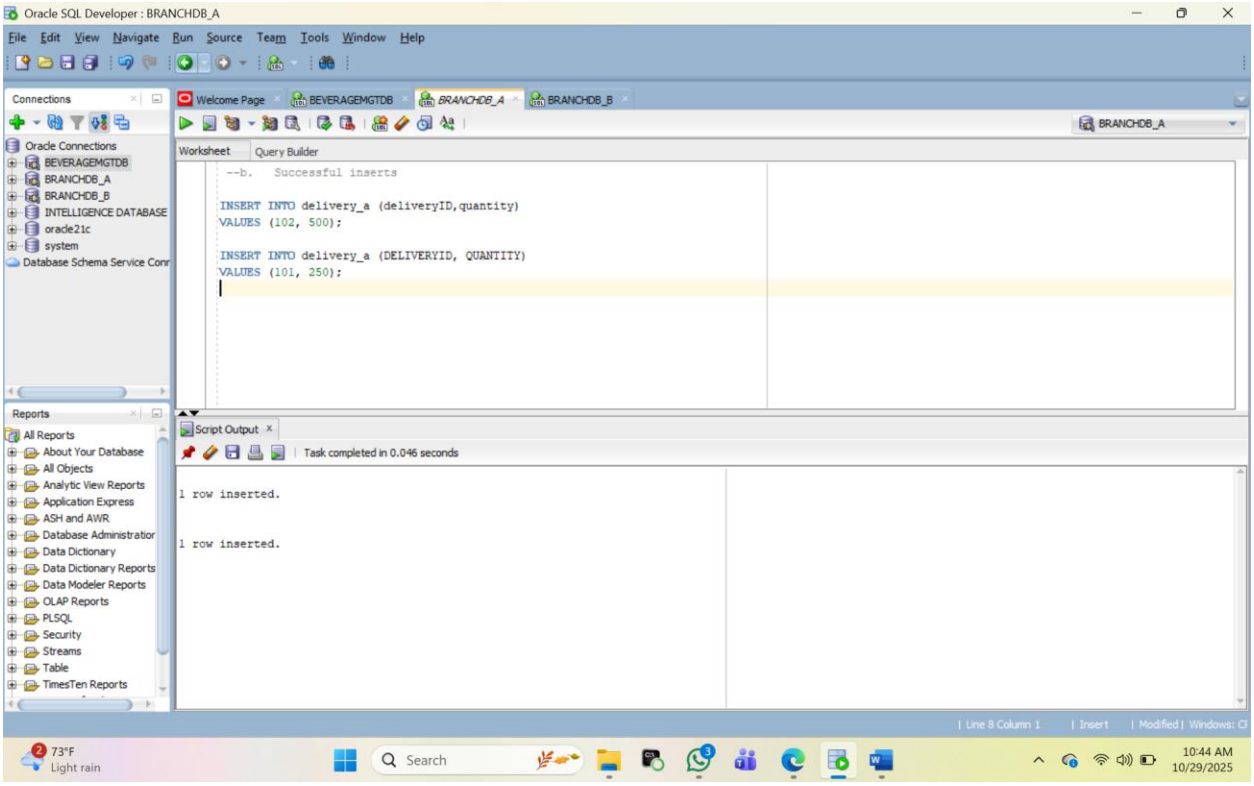
2. Successful inserts

INSERT INTO delivery_a (deliveryID,quantity)

VALUES (102, 500);

INSERT INTO delivery_a (DELIVERYID, QUANTITY)

VALUES (101, 250);



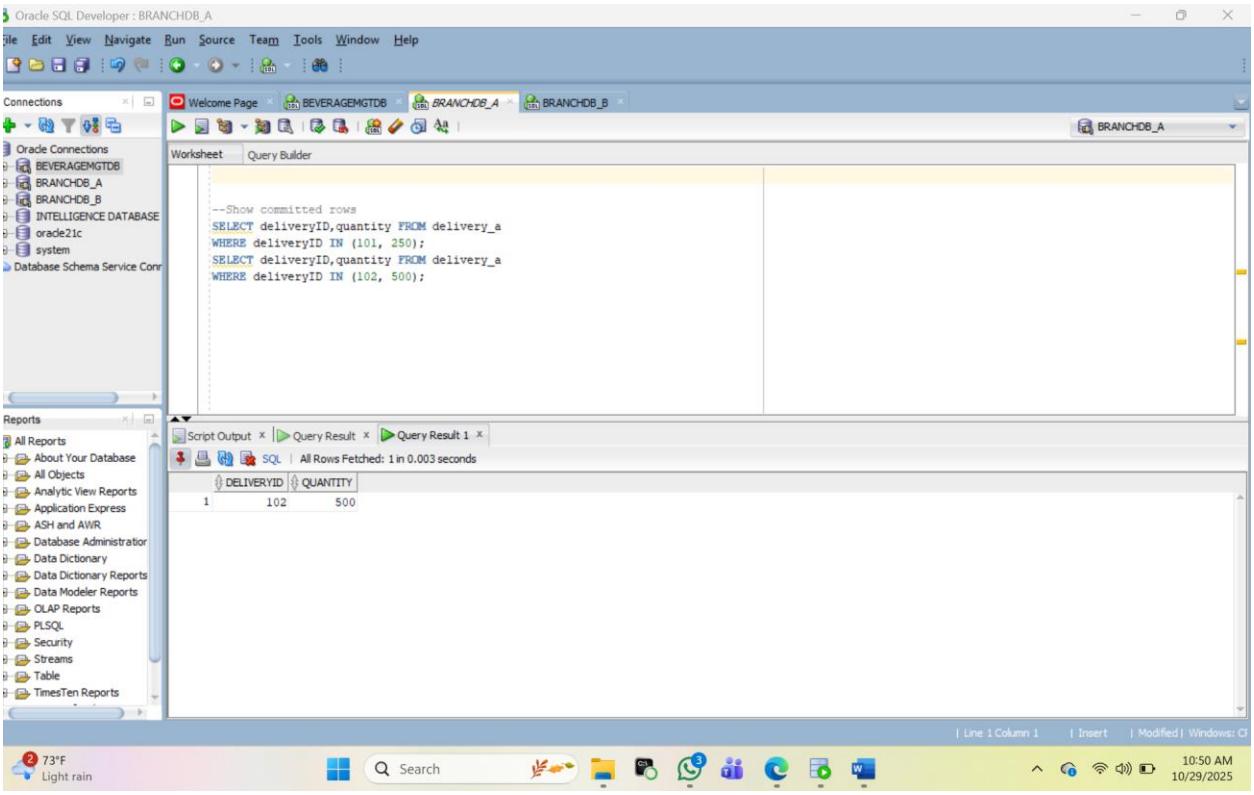
-Show committed rows

SELECT deliveryID,quantity FROM delivery_a

WHERE deliveryID IN (101, 250);

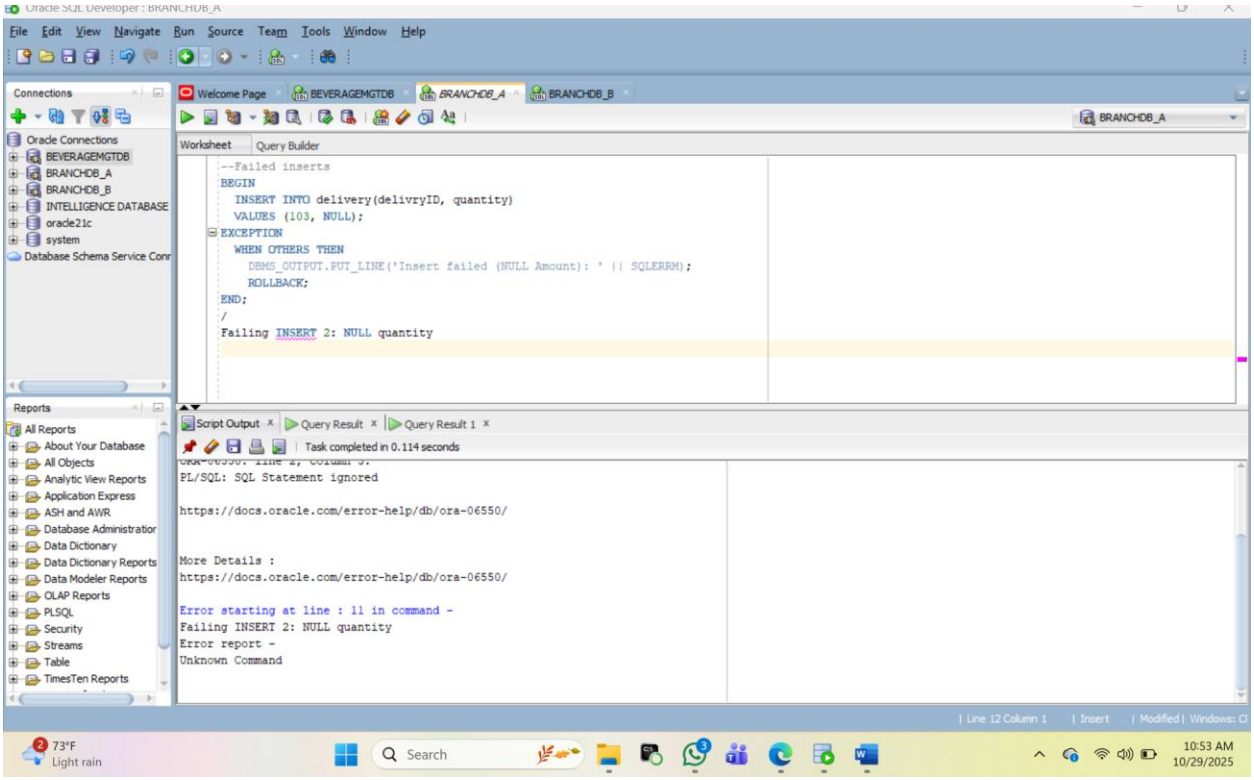
SELECT deliveryID,quantity FROM delivery_a

WHERE deliveryID IN (102, 500);



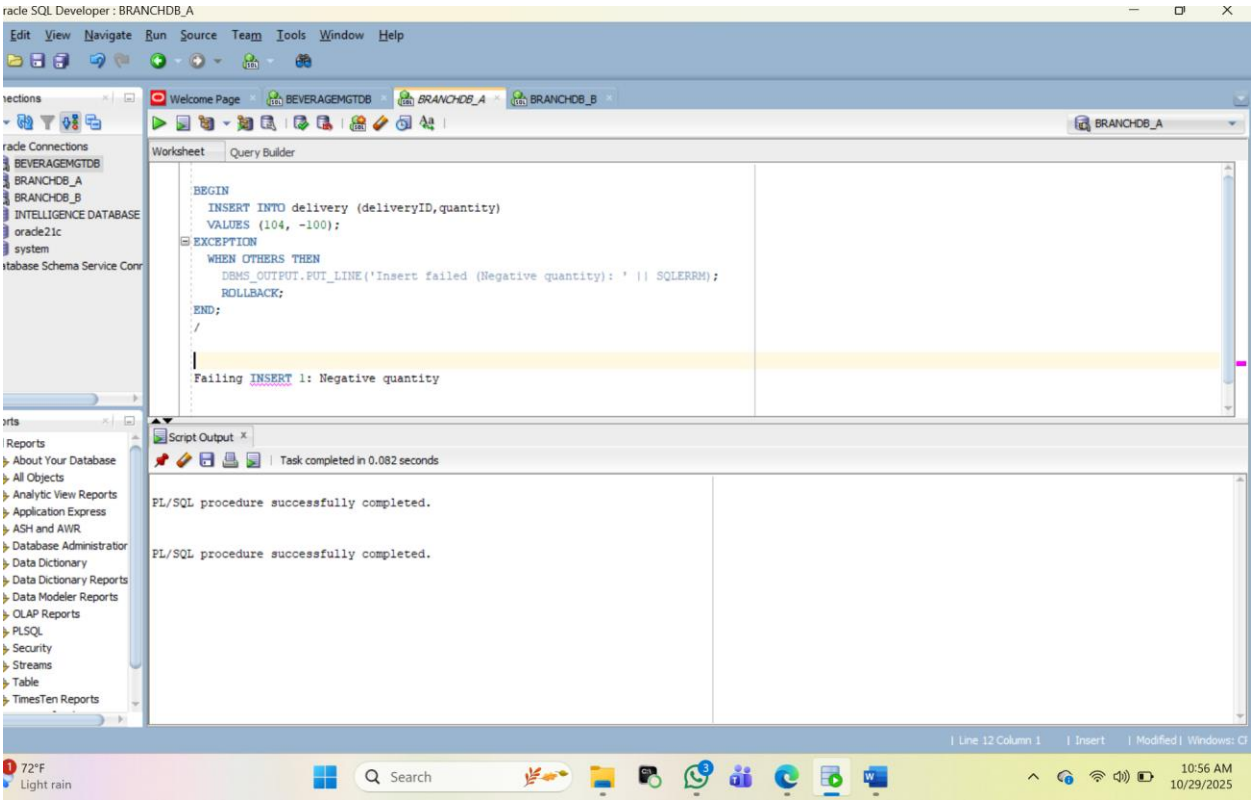

```
--Failed inserts
BEGIN
  INSERT INTO delivery(delivryID, quantity)
  VALUES (103, NULL);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Insert failed (NULL Amount): ' || SQLERRM);
    ROLLBACK;
END;
/
```

Failing INSERT 2: NULL quantity



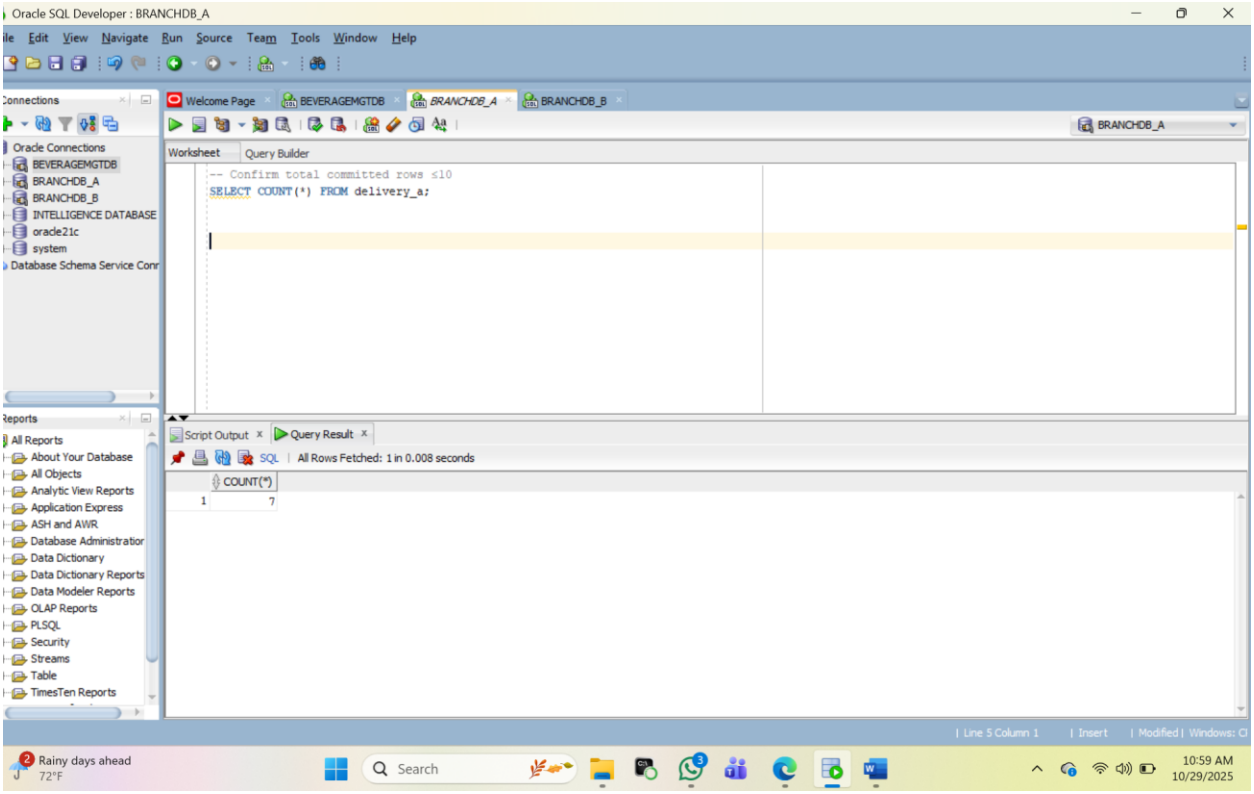

```
BEGIN
  INSERT INTO delivery (deliveryID,quantity)
  VALUES (104, -100);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Insert failed (Negative quantity): ' || SQLERRM);
    ROLLBACK;
END;
/
```

Failing INSERT 1: Negative quantity



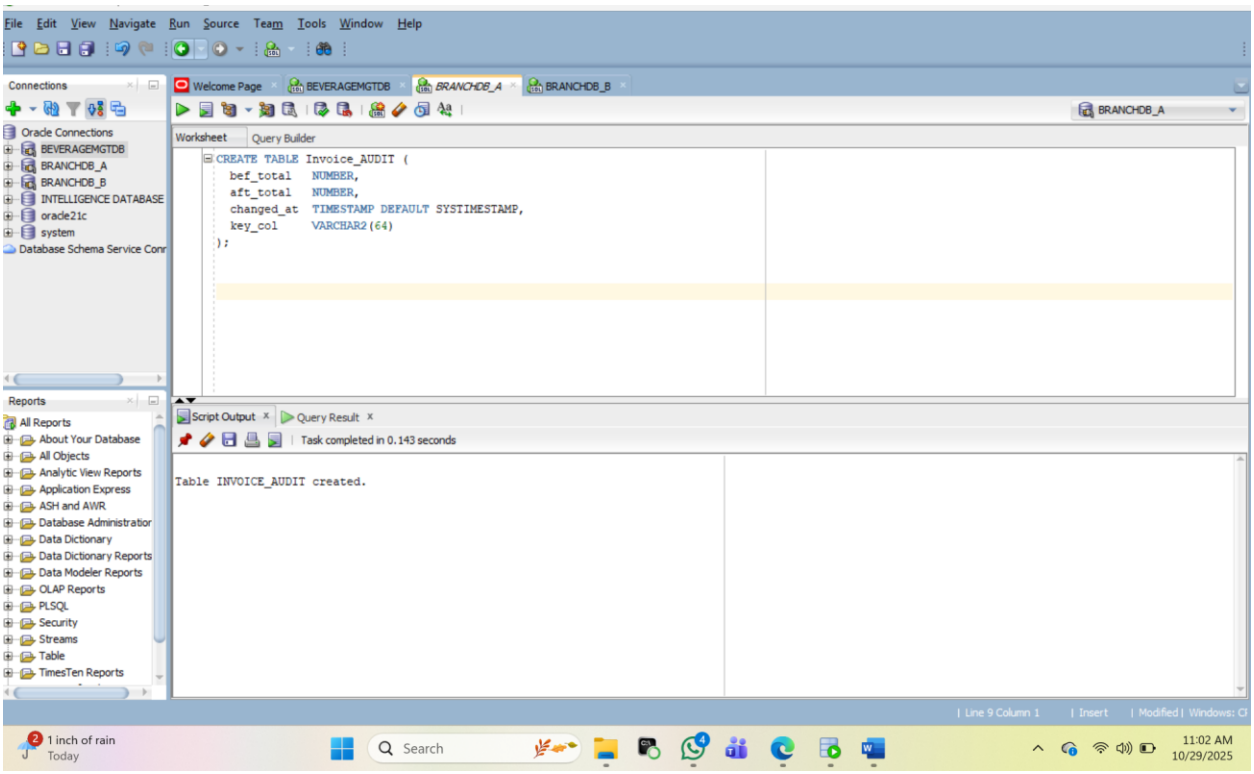
-- Confirm total committed rows ≤10

SELECT COUNT(*) FROM delivery_a;



B7 : E–C–A Trigger for Denormalized Totals

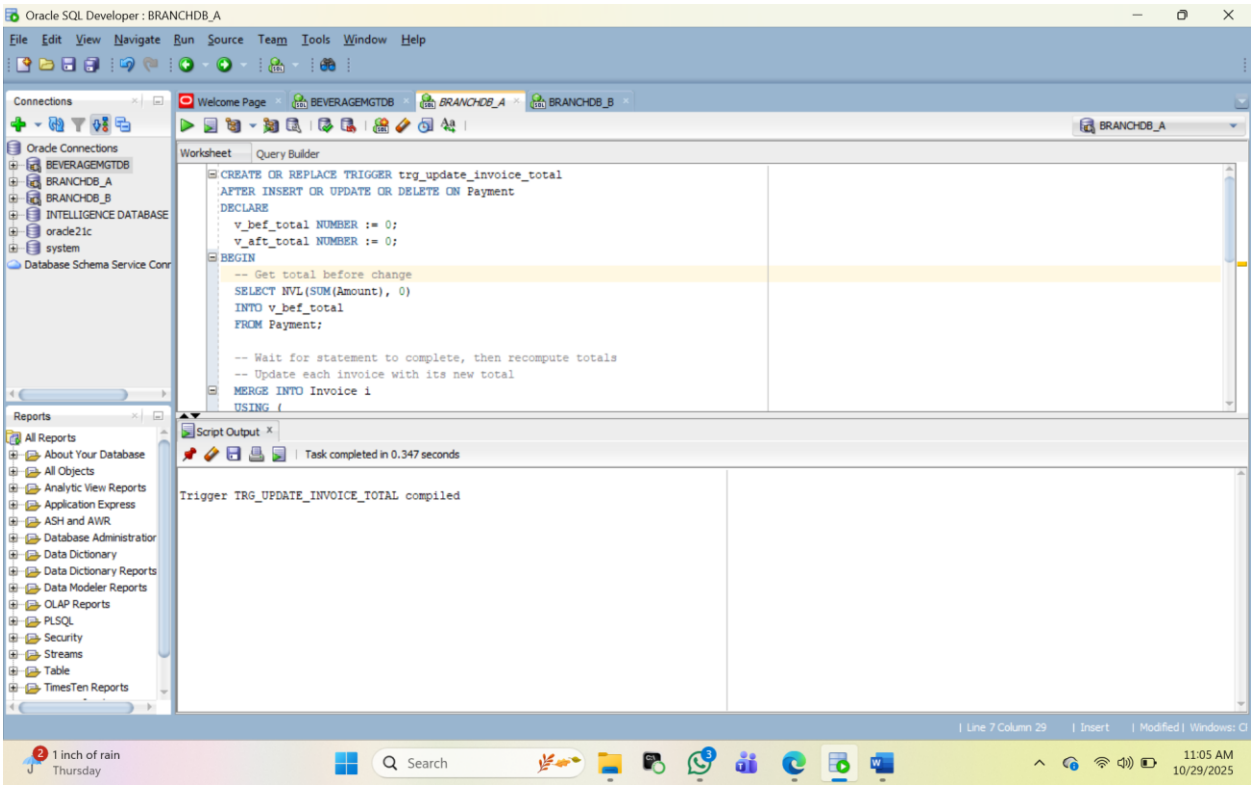
```
CREATE TABLE Invoice_AUDIT (
  bef_total   NUMBER,
  aft_total   NUMBER,
  changed_at  TIMESTAMP DEFAULT SYSTIMESTAMP,
  key_col     VARCHAR2(64)
);
```



2. Trigger: AFTER INSERT/UPDATE/DELETE on Payment

```
CREATE OR REPLACE TRIGGER trg_update_invoice_total
AFTER INSERT OR UPDATE OR DELETE ON Payment
DECLARE
  v_bef_total NUMBER := 0;
  v_aft_total NUMBER := 0;
BEGIN
  -- Get total before change
  SELECT NVL(SUM(Amount), 0)
  INTO v_bef_total
  FROM Payment;


  -- Wait for statement to complete, then recompute totals
  -- Update each invoice with its new total
```

```
 MERGE INTO Invoice i
 USING (
  SELECT InvoiceID, SUM(Amount) AS new_total
  FROM Payment
  GROUP BY InvoiceID
 ) p
 ON (i.InvoiceID = p.InvoiceID)
 WHEN MATCHED THEN
  UPDATE SET i.TotalAmount = p.new_total;

 -- Get total after change
 SELECT NVL(SUM(Amount), 0)
 INTO v_aft_total
 FROM Payment;

 -- Log to audit table
 INSERT INTO Invoice_AUDIT (bef_total, aft_total, key_col)
 VALUES (v_bef_total, v_aft_total, 'Payment');

END;
/
```



-- Insert 2 payments

INSERT INTO Payment(PaymentID, InvoiceID, Amount, PaymentDate, Method)

VALUES (3001, 1003, 2000, SYSDATE, 'CASH');

INSERT INTO Payment(PaymentID, InvoiceID, Amount, PaymentDate, Method)

VALUES (3002, 1003, 1000, SYSDATE, 'CARD');

-- Update 1 payment

UPDATE Payment

SET Amount = 2500

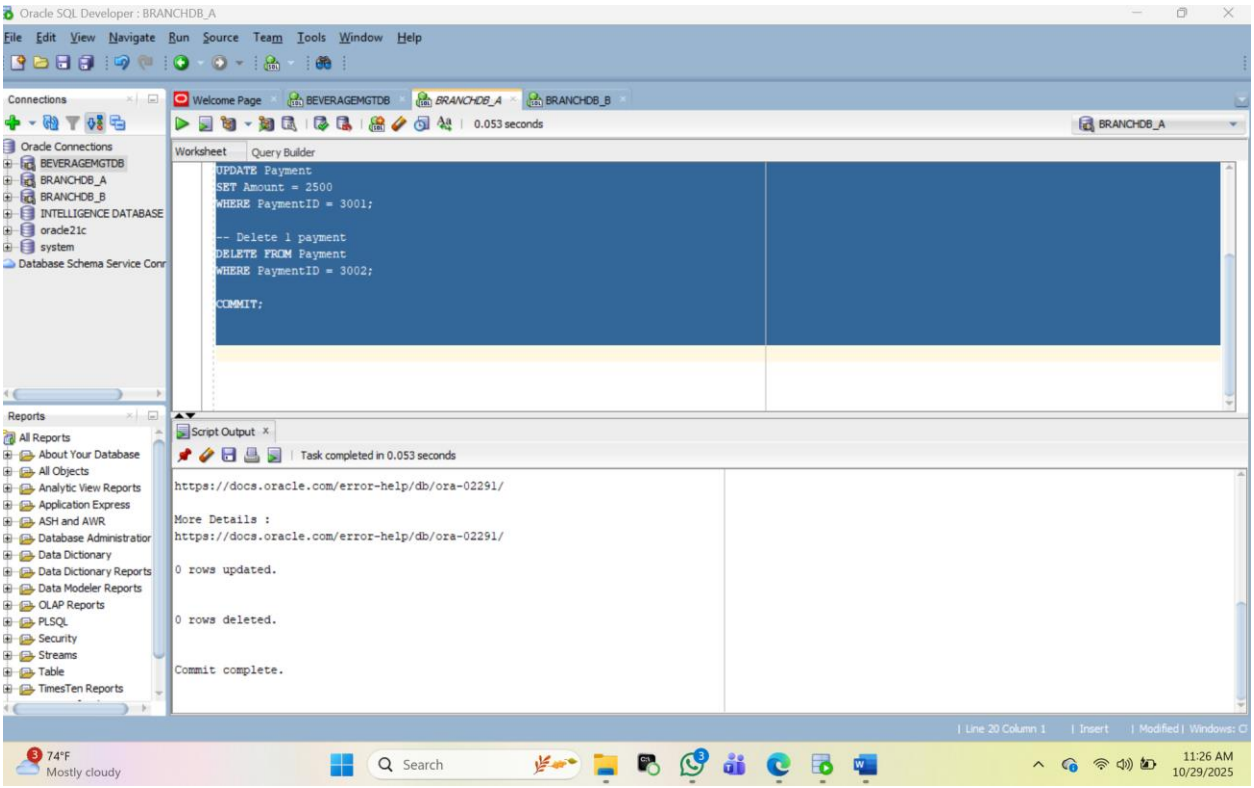
WHERE PaymentID = 3001;

-- Delete 1 payment

DELETE FROM Payment

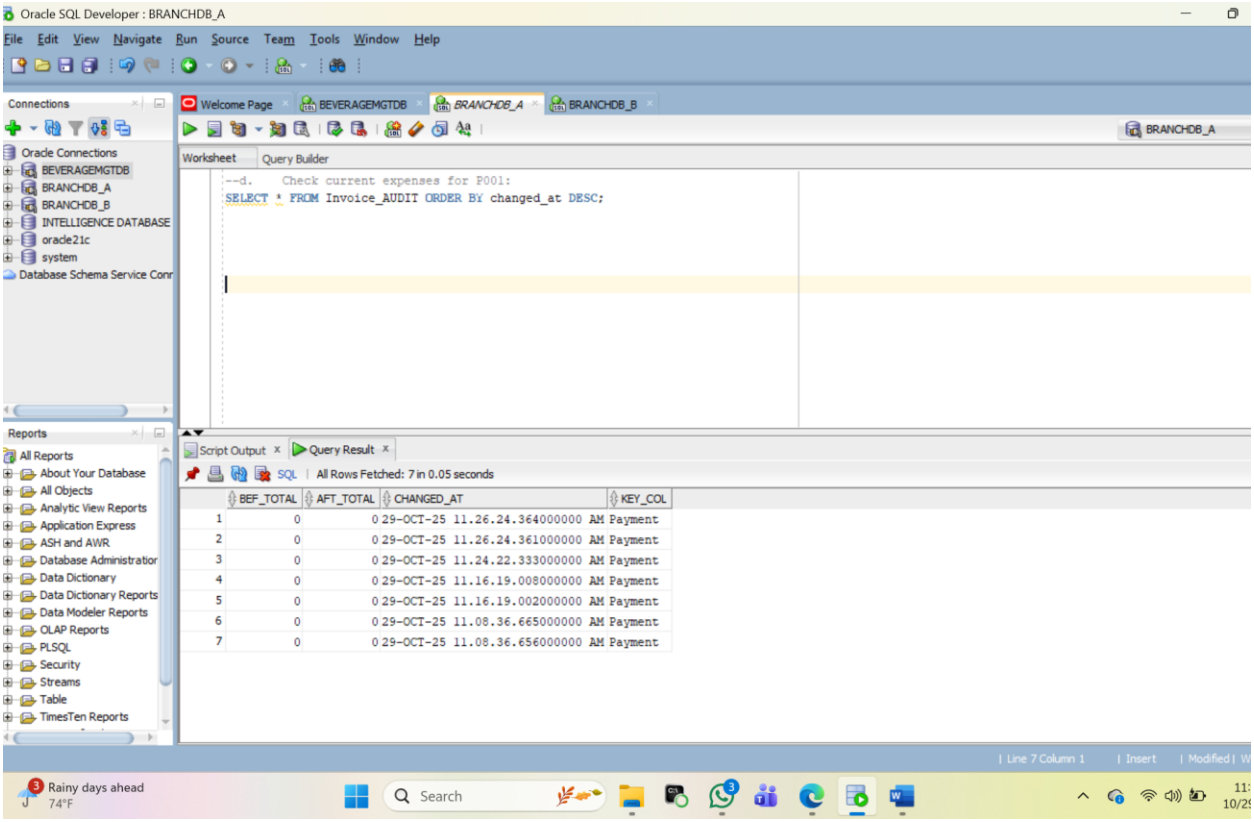
WHERE PaymentID = 3002;

COMMIT;



3.Check current invoice_audit changed desc

SELECT * FROM Invoice_AUDIT ORDER BY changed_at DESC;



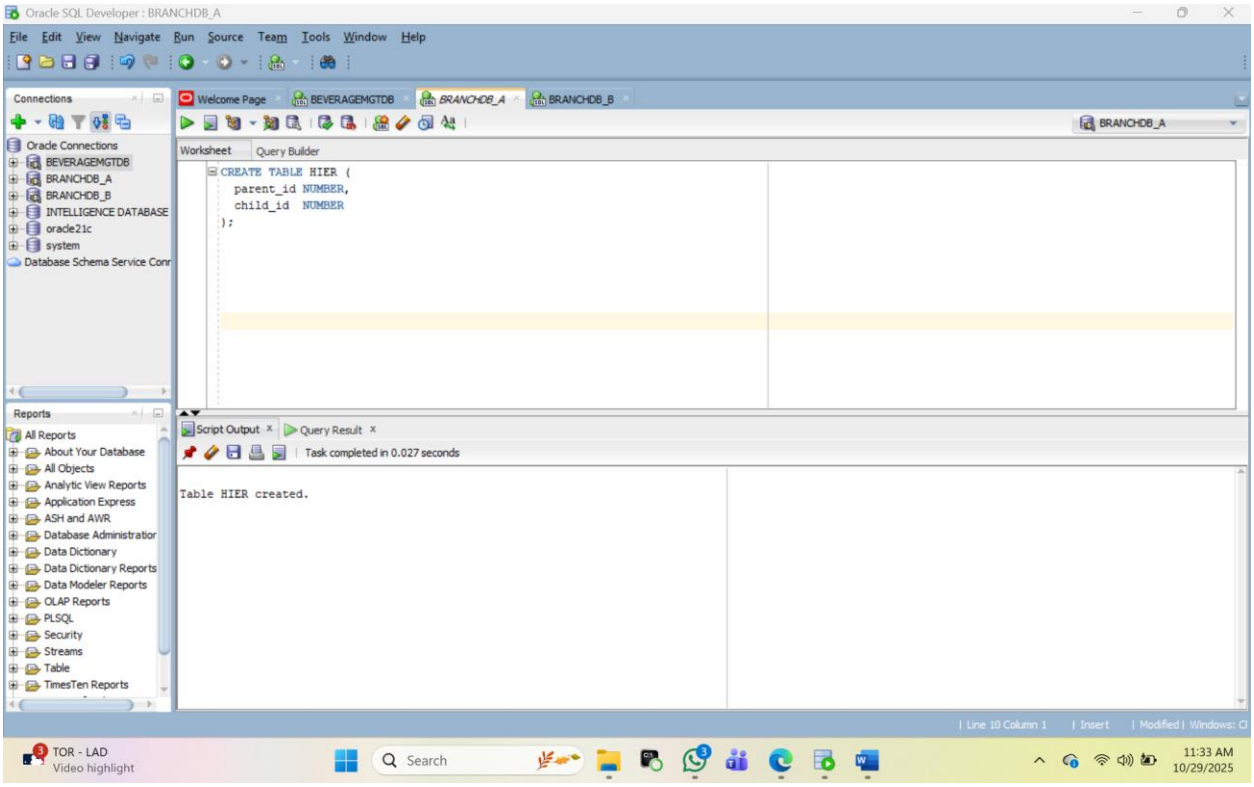
B8: Recursive Hierarchy Roll-Up

1. CREATE TABLE HIER (

parent_id NUMBER,

child_id NUMBER

);



2.-- 6 rows: 1 root → 2 regionals → 3 locals

INSERT INTO HIER VALUES (1, 2); -- National → Regional A

INSERT INTO HIER VALUES (1, 3); -- National → Regional B

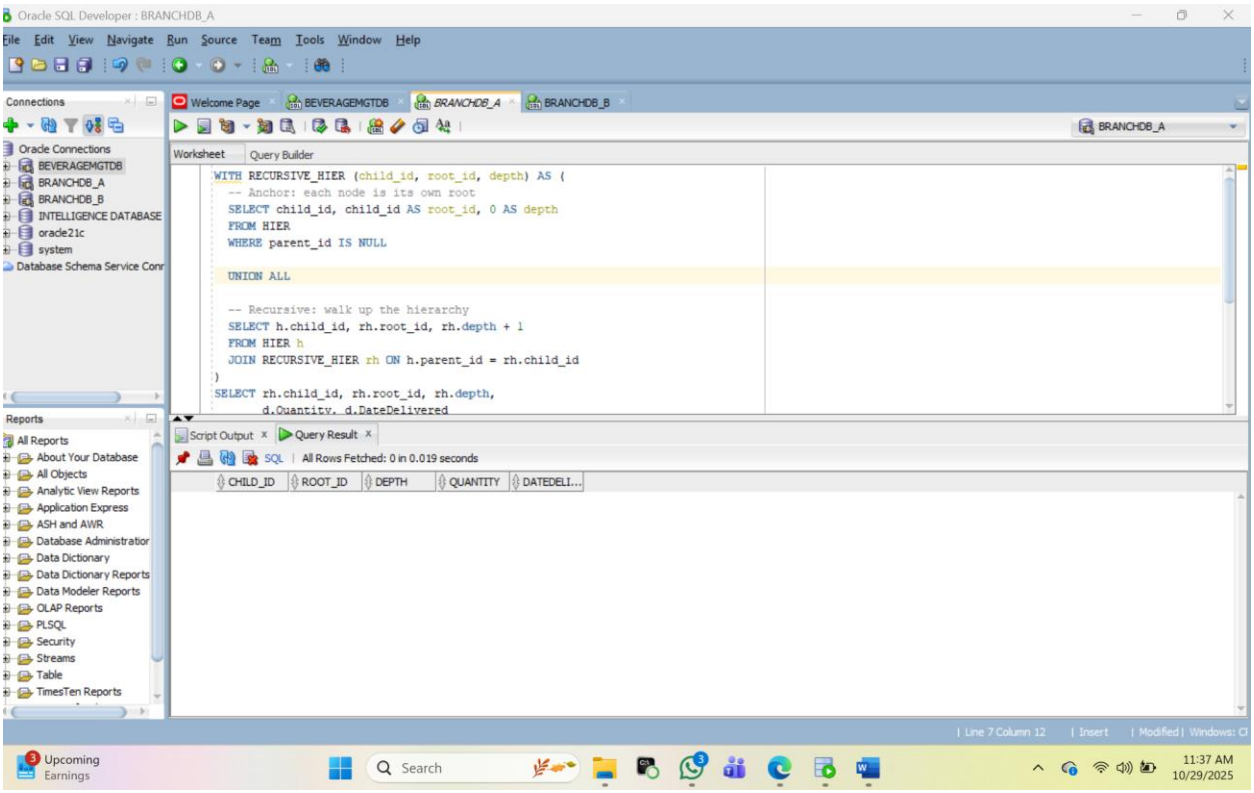
INSERT INTO HIER VALUES (2, 4); -- Regional A → Local A1

INSERT INTO HIER VALUES (2, 5); -- Regional A → Local A2

INSERT INTO HIER VALUES (3, 6); -- Regional B → Local B1

INSERT INTO HIER VALUES (3, 7); -- Regional B → Local B2

COMMIT;



3.WITH RECURSIVE_HIER (child_id, root_id, depth) AS (

-- Anchor: each node is its own root

SELECT child_id, child_id AS root_id, 0 AS depth

FROM HIER

WHERE parent_id IS NULL

UNION ALL

-- Recursive: walk up the hierarchy

SELECT h.child_id, rh.root_id, rh.depth + 1

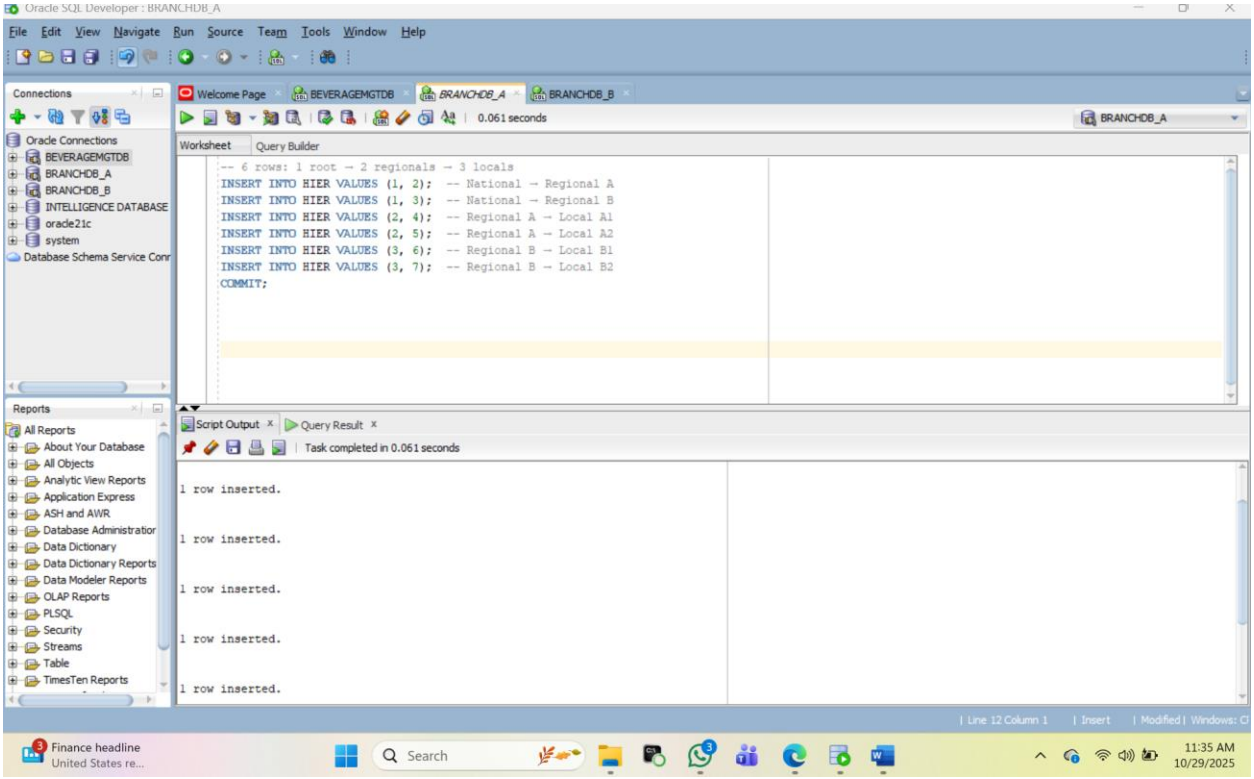
FROM HIER h

JOIN RECURSIVE_HIER rh ON h.parent_id = rh.child_id

)

SELECT rh.child_id, rh.root_id, rh.depth,

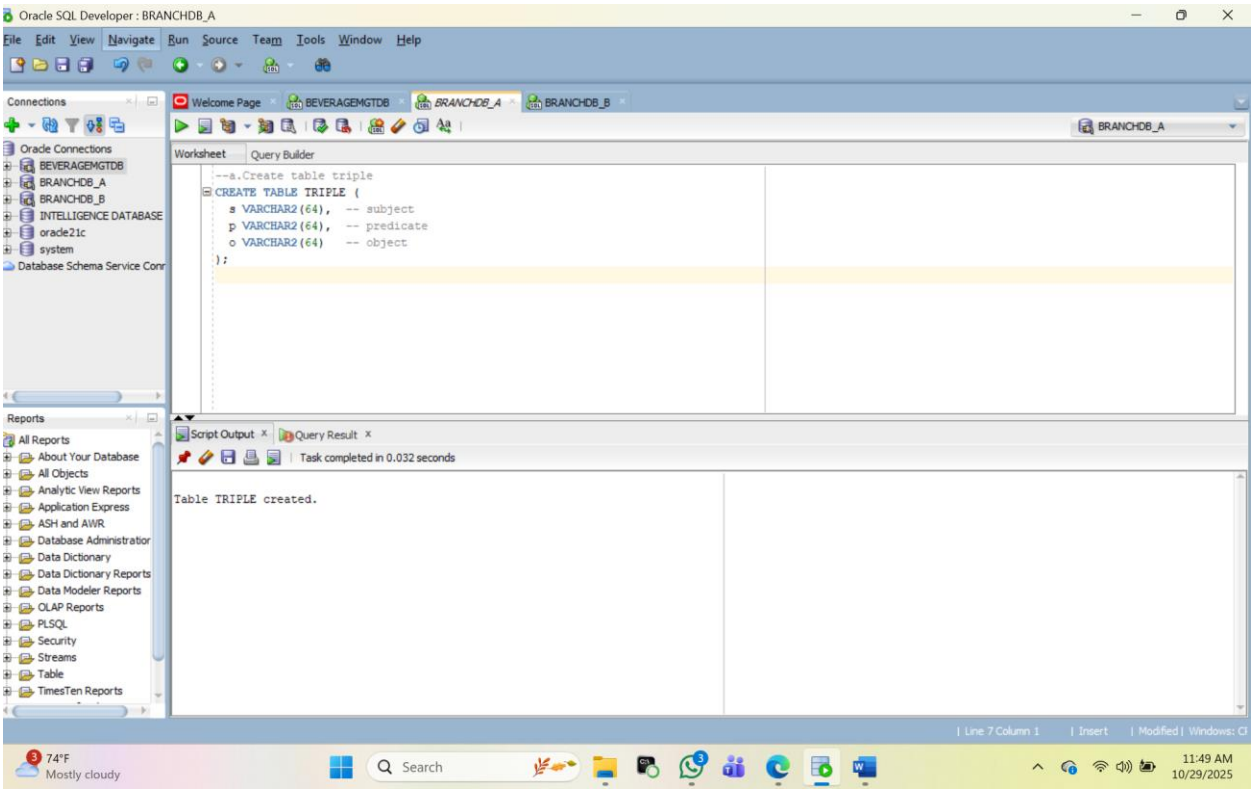
```
    d.Quantity, d.DateDelivered
FROM RECURSIVE_HIER rh
JOIN Delivery d ON d.DistributorID = rh.child_id
ORDER BY rh.root_id, rh.child_id;
```



B9 :Mini-Knowledge Base with Transitive Inference

1.Create table triple

```
CREATE TABLE TRIPLE (
  s VARCHAR2(64),  -- subject
  p VARCHAR2(64),  -- predicate
  o VARCHAR2(64)   -- object
);
```



2.- Type hierarchy: isA relationships

INSERT INTO TRIPLE VALUES ('Distributor_A1', 'isA', 'Distributor');

INSERT INTO TRIPLE VALUES ('Distributor_A2', 'isA', 'Distributor');

INSERT INTO TRIPLE VALUES ('Distributor', 'isA', 'Partner');

INSERT INTO TRIPLE VALUES ('Partner', 'isA', 'Entity');

-- Role relationships

INSERT INTO TRIPLE VALUES ('Distributor_A1', 'delivers', 'Batch_101');

INSERT INTO TRIPLE VALUES ('Distributor_A2', 'delivers', 'Batch_102');

INSERT INTO TRIPLE VALUES ('Batch_101', 'contains', 'Product_X');

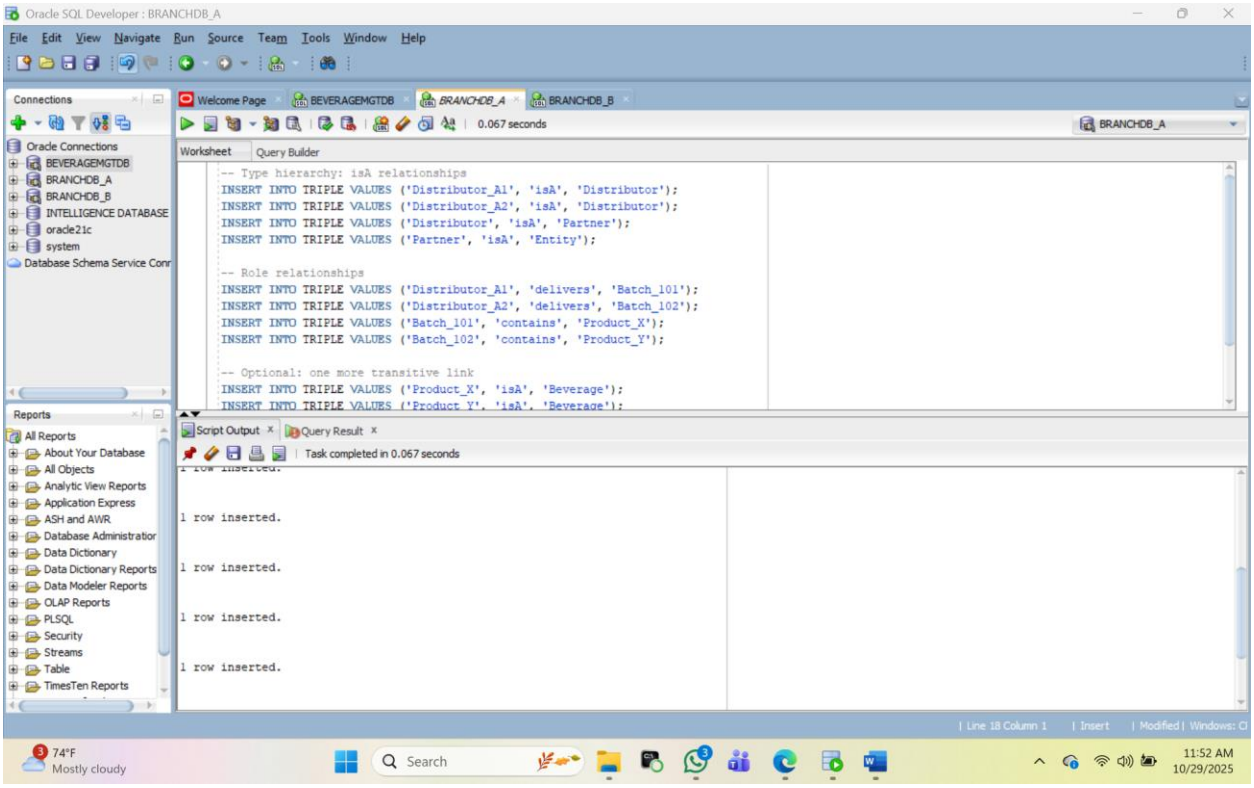
INSERT INTO TRIPLE VALUES ('Batch_102', 'contains', 'Product_Y');

-- Optional: one more transitive link

INSERT INTO TRIPLE VALUES ('Product_X', 'isA', 'Beverage');

INSERT INTO TRIPLE VALUES ('Product_Y', 'isA', 'Beverage');

COMMIT;



3.WITH isA_chain (s, root, depth) AS (

-- Anchor: direct isA relationships

SELECT s, o AS root, 1 AS depth

FROM TRIPLE

WHERE p = 'isA'

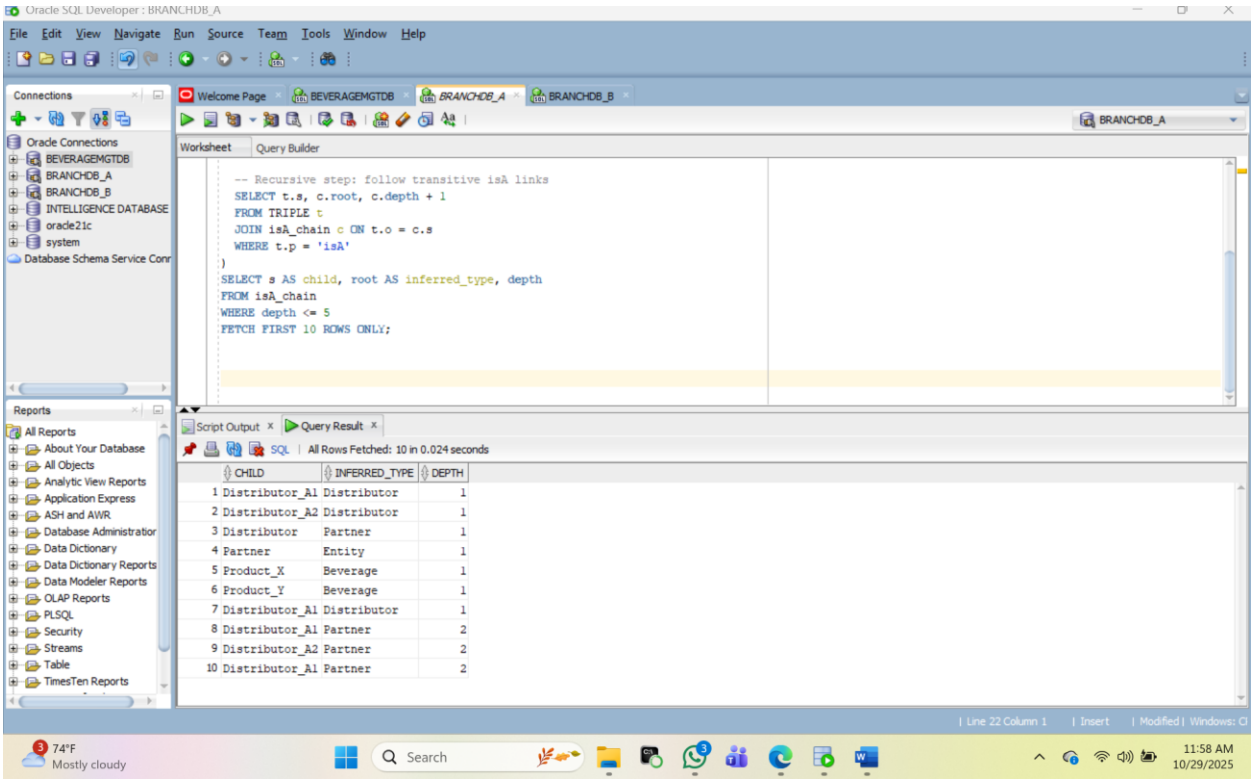
UNION ALL

-- Recursive step: follow transitive isA links

SELECT t.s, c.root, c.depth + 1

FROM TRIPLE t

```
  JOIN isA_chain c ON t.o = c.s
  WHERE t.p = 'isA'
)
SELECT s AS child, root AS inferred_type, depth
FROM isA_chain
WHERE depth <= 5
FETCH FIRST 10 ROWS ONLY;
```

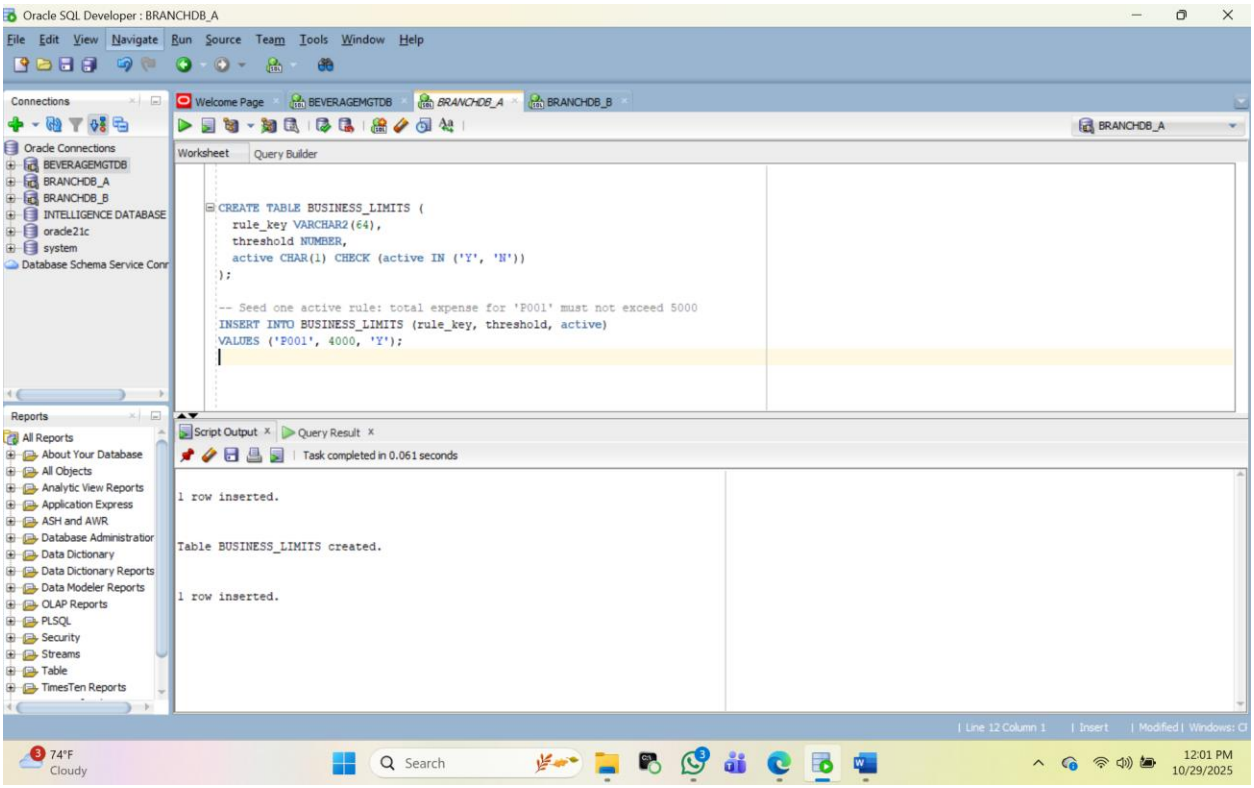


4.No need to delete a row as long as total committed rows across the project (including TRIPLE) remain ≤10

B10 :Business Limit Alert (Function + Trigger) (row-budget safe)

1 Create BUSINESS_LIMITS and seed exactly one active rule.

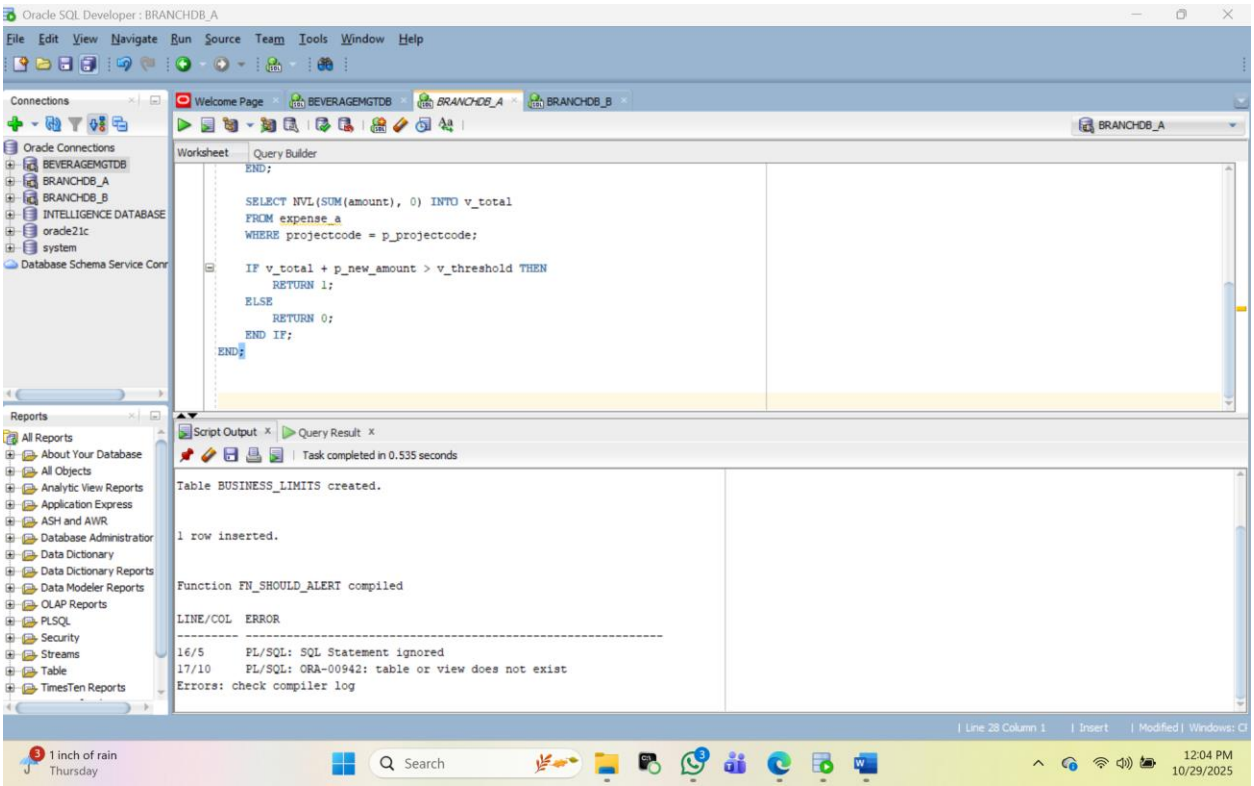
```
CREATE TABLE BUSINESS_LIMITS (
  rule_key VARCHAR2(64),
  threshold NUMBER,
  active CHAR(1) CHECK (active IN ('Y', 'N'))
);

-- Seed one active rule: total expense for 'P001' must not exceed 5000
INSERT INTO BUSINESS_LIMITS (rule_key, threshold, active)
VALUES ('P001', 4000, 'Y');
```



2. Function that reads BUSINESS_LIMITS and inspects current data in Expense or Project to decide a violation:

```
CREATE OR REPLACE FUNCTION fn_should_alert(p_projectcode VARCHAR2,
p_new_amount NUMBER)
RETURN NUMBER
IS
   v_threshold NUMBER := 0;
   v_total NUMBER := 0;
BEGIN
   BEGIN
      SELECT threshold INTO v_threshold
```

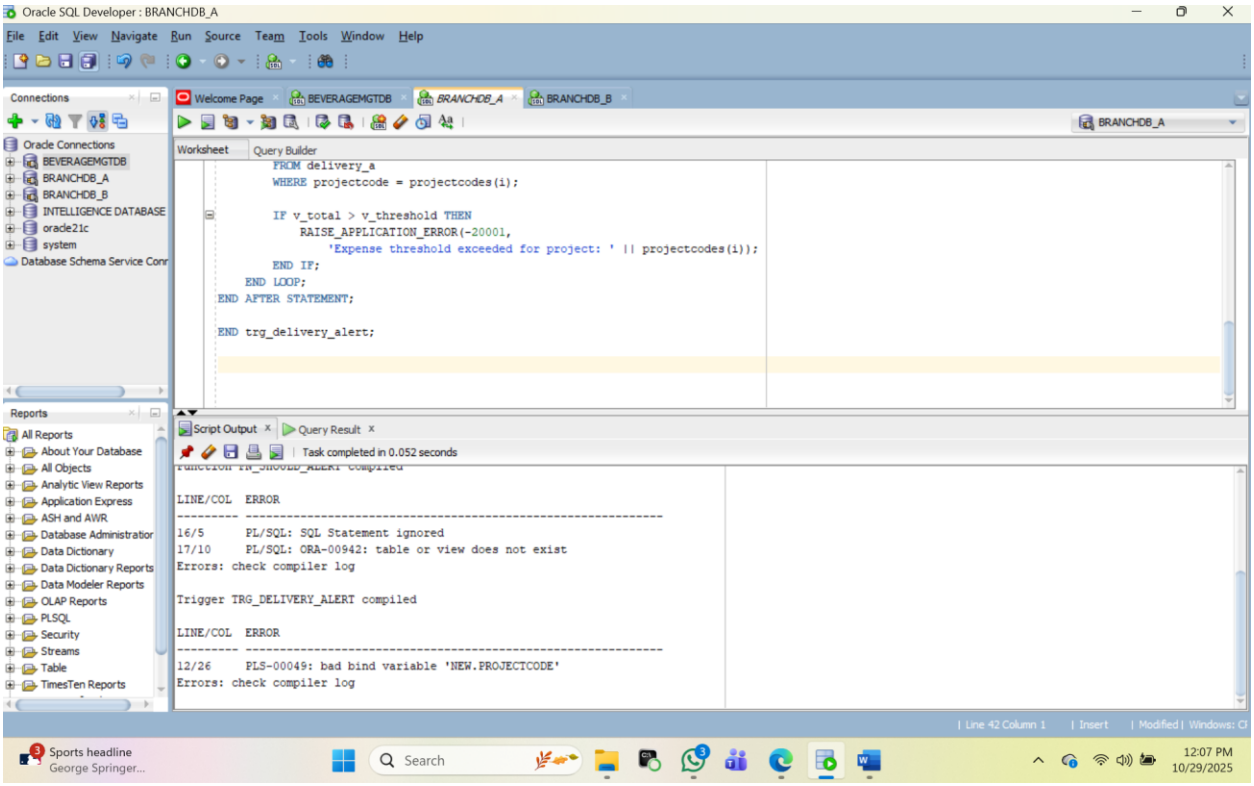
```
    FROM BUSINESS_LIMITS
    WHERE rule_key = 'MAX_PROJECT_EXPENSE' AND active = 'Y';
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN 0; -- No rule found, allow insert/update
  END;

  SELECT NVL(SUM(amount), 0) INTO v_total
  FROM expense_a
  WHERE projectcode = p_projectcode;

  IF v_total + p_new_amount > v_threshold THEN
    RETURN 1;
  ELSE
    RETURN 0;
  END IF;
END;
```



3. CREATE OR REPLACE TRIGGER trg_delivery_alert

```
FOR INSERT OR UPDATE ON delivery_a
COMPOUND TRIGGER

-- Declare a collection to hold project codes being modified
TYPE projectcode_tab IS TABLE OF VARCHAR2(64) INDEX BY PLS_INTEGER;
projectcodes projectcode_tab;
idx INTEGER := 0;

-- BEFORE EACH ROW: collect project codes
BEFORE EACH ROW IS
BEGIN
   idx := idx + 1;
   projectcodes(idx) := :NEW.projectcode;
END BEFORE EACH ROW;

-- AFTER STATEMENT: validate thresholds
AFTER STATEMENT IS
   v_total NUMBER;
   v_threshold NUMBER;
BEGIN
   -- Get threshold
   SELECT threshold INTO v_threshold
   FROM BUSINESS_LIMITS
   WHERE rule_key = 'MAX_PROJECT_EXPENSE' AND active = 'Y';

   -- Check each projectcode once
```

```
    FOR i IN 1 .. projectcodes.COUNT LOOP
        SELECT NVL(SUM(amount), 0) INTO v_total
        FROM delivery_a
        WHERE projectcode = projectcodes(i);

        IF v_total > v_threshold THEN
            RAISE_APPLICATION_ERROR(-20001,
                'Expense threshold exceeded for project: ' || projectcodes(i));
        END IF;
    END LOOP;
END AFTER STATEMENT;

END trg_delivery_alert;
```



4. 2 Passing DML:

```
-- PASSING CASE 1
INSERT INTO Payment (PaymentID, InvoiceID, Amount, PaymentDate, Method)
VALUES (4001, 1003, 3000, SYSDATE, 'CASH');

-- PASSING CASE 2
INSERT INTO Payment (PaymentID, InvoiceID, Amount, PaymentDate, Method)
VALUES (4002, 1003, 4500, SYSDATE, 'CARD');
```

2 Failing DML:

```
BEGIN
   INSERT INTO expense_a VALUES (309, 'Threshold Breach Test', 300, 'P001');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('FAILED: ' || SQLERRM);
      ROLLBACK;
END;



BEGIN
   INSERT INTO expense_a VALUES (310, 'Executive Travel Package', 3500, 'P003');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('FAILED: ' || SQLERRM);
```

ROLLBACK;

END

Counting Rows:

SELECT COUNT(*) FROM delivery_a;

Resulting committed data:

SELECT * FROM deliver_a;