

SOLUTIONS TO EXAM QUESTIONS

By

Amoss Robert (224019944)

Database System Used:

Oracle 21c Enterprise Edition

Section A

A1_1

The screenshot shows a 'Worksheet' tab in the top left corner. The code in the editor is as follows:

```
Worksheet Query Builder
show user;

-- Creating a new database user by the name Kigali_Branch in container database
CREATE USER c##Node_A IDENTIFIED BY nodea2025;
GRANT UNLIMITED TABLESPACE TO c##Node_A;
GRANT RESOURCE, DBA, CONNECT TO c##Node_A;

-- Creating another database by the name Bugesera_Branch in container database
CREATE USER c##Node_B IDENTIFIED BY nodeb2025;
GRANT UNLIMITED TABLESPACE TO c##Node_B;
GRANT RESOURCE, DBA, CONNECT TO c##Node_B;
```

Figure 1: Node_A and Node_B creation

A1_2 Refer to A1 SQL code file

A1_3

The screenshot shows a 'Worksheet' tab in the top left corner. The code in the editor is as follows:

```
-- *****
-- Database Link
-- Creating a database link between Node_A and Node_B and test witht SELECT and distributed join
CREATE DATABASE LINK proj_link
CONNECT TO c##Node_B IDENTIFIED BY nodeb2025
USING 'localhost:1521/orcl';

-- *****
-- Creating Appointment_ALL View
```

Below the editor, the 'Script Output' tab is active, showing the error message:

```
Error starting at line : 120 in command -
CREATE DATABASE LINK proj_link
CONNECT TO c##Node_B IDENTIFIED BY nodeb2025
USING 'localhost:1521/orcl'
Error report -
ORA-02011: duplicate database link name

https://docs.oracle.com/error-help/db/ora-02011/02011. 00000 - "duplicate database link name"
Cause:   The database link name specified in a CREATE DATABASE
```

Figure 2: Database Link

A1_4

```
SELECT * FROM Appointment_B@proj_link;

-- A.3.
-- Testing a VIEW on UNION ALL
SELECT * FROM Appointment_A
UNION ALL
SELECT * FROM Appointment_B@proj_link;
```

Script Output | Query Result | All Rows Fetched: 9 in 0.009 seconds

	APPOINTMENTID	PATIENTID	DOCTORID	VISITDATE	DIAGNOSIS	STATUS
1	5	21	28	20-SEP-25	Gestational Diabetes Mellitus	pending
2	6	19	28	20-SEP-25	Pregnancy-Induced Hypertension	pending
3	8	24	27	29-SEP-25	Epilepsy - generalised tonic-clonic	pending
4	11	17	29	29-JAN-25	Stable angina due to coronary artery disease	pending
5	4	23	25	25-APR-25	Urinary Tract Infection	pending
6	7	22	28	10-AUG-25	Vaginal Candidiasis	pending
7	9	20	27	15-JUL-25	Ischemic stroke - left hemisphere	pending
8	10	16	29	15-MAR-25	Primary hypertension	pending
9	12	18	30	14-JAN-25	Acute appendicitis	pending

Figure 3: View of Appointments

A1_4

```
-- A.4.
Select count(*) from Appointment_A;
select count(*) from Appointment_B@proj_link;
```

Script Output | Query Result | All Rows Fetched: 1 in 0.009 seconds

COUNT(*)
1
4

Figure 4: Matching Count(*)

A2_1 refer to the A2 SQL code file

A2_2

-- A2.2 REMOTE SELECT
SELECT * FROM Doctor_B@proj_link;

Query Result | SQL | All Rows Fetched: 3 in 0.011 seconds

DOCTORID	FULLNAME	SPECIALTY	DEPTID	PHONE	EMAIL
1	25 Dr. Evelyn Banda	Gynecologist	1	797576277	eviebanda@gmail.com
2	28 Dr. Mercy Phiri	Gynecologist	1	798576209	mercyphiri@gmail.com
3	29 Dr. Issac Merger	Cardiologist	2	799576286	issacmerger99@gmail.com

Figure 5: Database link Already Created

A2_3

-- A2.3 DIstributed join from Appointment_A with Patient_B
SELECT a.AppointmentID,
 p.FullName AS Patient_Name,
 p.Gender,
 a.VisitDate,
 a.Diagnosis,
 d.FullName AS Doctor_Name
FROM Appointment_A a
JOIN Patient_B@proj_link p ON a.PatientID = p.PatientID
JOIN Doctor_B@proj_link d ON a.DoctorID = d.DoctorID
WHERE a.VisitDate BETWEEN DATE '2023-01-01' AND DATE '2025-10-30' -- Date range filter
 AND ROWNUM <= 10 -- Strict row limit
ORDER BY a.VisitDate;

Query Result | SQL | All Rows Fetched: 2 in 0.008 seconds

APPOINTMENTID	PATIENT_NAME	GENDER	VISITDATE	DIAGNOSIS	DOCTOR_NAME
1	11 Moses Peter	Male	29-JAN-25	Stable angina due to coronary artery disease	Dr. Issac Merger
2	6 Eveless Eneya	Female	20-SEP-25	Pregnancy-Induced Hypertension	Dr. Mercy Phiri

Figure 6: Distributed Join

A3_1

The screenshot shows the Oracle SQL Developer interface. On the left, a code editor displays a SQL query:

```
SELECT
    Status,
    COUNT(*) as Total_Appointments,
    AVG(MONTHS_BETWEEN(SYSDATE, VisitDate)) as Avg_Months_Since_Visit,
    MIN(VisitDate) as Earliest_Visit,
    MAX(VisitDate) as Latest_Visit
FROM Appointment_B@proj_link
GROUP BY Status
ORDER BY Total_Appointments DESC;
```

On the right, the 'Query Result' tab is open, showing the execution details and the resulting data:

STATUS	TOTAL_APPPOINTMENTS	AVG_MONTHS_SINCE_VISIT	EARLIEST_VISIT	LATEST_VISIT
1 pending	5 5.82190748207885304659498207885304659498	14-JAN-25	10-AUG-25	

Figure 7: Serial Aggregation on Appointments all

A3_2

The screenshot shows the Oracle SQL Developer interface. On the left, two code snippets are shown, separated by a double vertical line:

```
-- /*+ PARALLEL(appointment_b@proj_link, 8) */
SELECT
    Status,
    COUNT(*) as Total_Appointments,
    AVG(MONTHS_BETWEEN(SYSDATE, VisitDate)) as Avg_Months_Since_Visit,
    MIN(VisitDate) as Earliest_Visit,
    MAX(VisitDate) as Latest_Visit
FROM appointment_b@proj_link
GROUP BY Status
ORDER BY Total_Appointments DESC;
-- SET AUTOTRACE OFF;
```

On the right, the 'Query Result' tab is open, showing the execution details and the resulting data.

Figure 8: Aggregations on Parallel Hint

A3_3

The screenshot shows the Oracle SQL Developer interface with the 'SQL' tab selected. The title bar indicates 'All Rows Fetched: 14 in 0.236 seconds'. The main area displays the DBMS_XPLAN output for a query. The output includes a header 'PLAN_TABLE_OUTPUT', a note about a remote statement, and a detailed execution plan with columns for Id, Operation, Name, Rows, Bytes, Cost (%CPU), Time, and Inst. The plan shows a SELECT STATEMENT REMOTE operation at step 0, followed by a SORT ORDER BY and HASH GROUP BY operation at step 1, and finally a TABLE ACCESS FULL operation at step 3.

PLAN_TABLE_OUTPUT							
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Inst
0	SELECT STATEMENT REMOTE		1	16	5 (40)	00:00:01	
1	SORT ORDER BY		1	16	5 (40)	00:00:01	
2	HASH GROUP BY		1	16	5 (40)	00:00:01	
3	TABLE ACCESS FULL	APPOINTMENT_B	5	80	3 (0)	00:00:01	ORCL

Figure 9: DBMS_XPLAN Output

A3_4

The screenshot shows the Oracle SQL Developer interface with the 'SQL' tab selected. The title bar indicates 'All Rows Fetched: 2 in 0.003 seconds'. The main area displays two parts of a script. The first part is an EXECUTE PLAN FOR statement for a query involving parallel operations. The second part is a SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY) command to view the execution plan details. Below the code, the results show a comparison table for execution types.

EXECUTION_TYPE	PLAN_NOTES	PROCESSING_METHOD	EXPECTED_COST	PERFORMANCE_NOTE
1 Serial	No parallel operations	Single process execution	Higher for large datasets	Slower for large data
2 Parallel (DOP=8)	PX COORDINATOR, PX SEND/RECEIVE operations	8 parallel processes distributed	Lower per-process cost	Faster for large data, overhead for small

Figure 10: Series vs Parallel comparison

A4_1

```
-- =====
-- PL/SQL BLOCK: Distributed Transaction with Two-Phase Commit
-- =====
-- Description: Inserts one local row on Node_A and one remote row via database link
-- Demonstrates atomic distributed transaction with COMMIT
-- =====

SET SERVEROUTPUT ON;

DECLARE
    v_local_appointment_id NUMBER;
    v_remote_prescription_id NUMBER;
BEGIN
    -- Generate unique IDs for the new records
    SELECT COALESCE(MAX(AppointmentID), 0) + 1 INTO v_local_appointment_id FROM Appointment_A;
    SELECT COALESCE(MAX(PrescriptionID), 0) + 1 INTO v_remote_prescription_id FROM Prescription_B@proj_link;

    -- Insert LOCAL row into Appointment_A on Node_A
    INSERT INTO Appointment_A (
        AppointmentID, PatientID, DoctorID, VisitDate, Diagnosis, Status
    ) VALUES (
        v_local_appointment_id,
        1, -- Existing PatientID
        1, -- Existing DoctorID
        SYSDATE,
        'Routine checkup distributed transaction test',
        'pending'
    );

    DBMS_OUTPUT.PUT_LINE('Local appointment inserted: ' || v_local_appointment_id);

    -- Insert REMOTE row into Prescription@proj_link on Node_B
    INSERT INTO Prescription_B@proj_link (
        PrescriptionID, AppointmentID, Notes, DateIssued
    ) VALUES (
        v_remote_prescription_id,
        v_local_appointment_id, -- Links to the local appointment
        'Prescription for distributed transaction test',
        SYSDATE
    );
}

-- COMMIT both inserts atomically (Two-Phase Commit)
COMMIT;

DBMS_OUTPUT.PUT_LINE('Distributed transaction committed successfully!');

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Transaction rolled back due to error');
    END;
    /

```

Script Output X | Task completed in 0.146 seconds

```
Local appointment inserted: 12
Remote prescription inserted: 11
Distributed transaction committed successfully!

PL/SQL procedure successfully completed.
```

Figure 11: Distributed Transactions with 2 Phase Commit

A4_2

The screenshot shows a PL/SQL block in the SQL editor:

```
-- A2 -----
-- PL/SQL BLOCK: Induce Distributed Transaction Failure by disabling the proj_link through dropping it
-- 
DROP DATABASE LINK proj_link;
```

The output window shows the error:

```
Error starting at line : 64 in command -
DROP DATABASE LINK proj_link
Error report -
ORA-02018: database link of same name has an open connection
```

Figure 12: Attempt to simulate database link failure to insert data

This result affected A4_4 and 4

A5_1

The screenshot shows a query in the SQL editor:

```
-- Logged in c##Bugesera_branch
SELECT sid,      serial# FROM v$session WHERE username = USER AND status = 'ACTIVE'; -- Session ID
--Found SID is 857 and SERIAL# is 25804

select * from prescription_B@proj_link; -- checking existing prescription records  from Node_B prescription Table
-- updating doctor information in the Kigali_Branch database
UPDATE prescription_B@proj_linkk SET NOTES = 'New infection detected' WHERE prescriptionid = 3;
```

The output window shows the results of the query:

PRESCRIPTIONID	APPOINTMENTID	NOTES	DATEISSUED
1	3	4 Positive urine culture, hydrate well	05-MAY-25
2	4	5 Controlled diet, monitor glucose	28-AUG-25
3	7	8 Start antiepileptic therapy	01-OCT-25
4	9	10 Monitor BP daily, lifestyle changes advised, antihypertensive therapy initiated.	20-MAR-25
5	10	11 Advise reduced cholesterol diet, initiate anti-anginal therapy, schedule follow-up for stress test	05-FEB-25
6	11	12 Prescription for distributed transaction test	28-OCT-25

Figure 13: Checking all available prescriptions

A5_1 Running update from Node_A in session1

The screenshot shows the Oracle SQL Developer interface. The 'Worksheet' tab is active, displaying the following SQL code:

```
-- Logged in c##Bugesera_branch
SELECT sid,      serial# FROM v$session WHERE username = 'USER' AND status = 'ACTIVE'; -- Session ID

--Found SID is 857 and SERIAL# is 25804

select * from prescription_B@proj_link; -- checking existing prescription records  from Node_B prescription Table

-- updating doctor information in the Kigali_Branch database
UPDATE prescription_B@proj_link SET NOTES = 'New infection detected' WHERE prescriptionid = 3;
```

Below the code, the 'Query Result' tab is selected, showing a table with six rows of prescription data. A red arrow points to the 'NOTES' column for the first row, which contains the value '4 New infection detected'.

PREScriptionID	APPOINTMENTID	NOTES	DATEISSUED
1	3	4 New infection detected	05-MAY-25
2	4	5 Controlled diet, monitor glucose	28-AUG-25
3	7	8 Start antiepileptic therapy	01-OCT-25
4	9	10 Monitor BP daily, lifestyle changes advised, antihypertensive therapy initiated.	20-MAR-25
5	10	11 Advise reduced cholesterol diet, initiate anti-anginal therapy, schedule follow-up for stress test	05-FEB-25
6	11	12 Prescription for distributed transaction test	28-OCT-25

Figure 14: Session 1 attempt to update records

A5_2 Opening session two from Node_B

The screenshot shows the Oracle SQL Developer interface. The 'Worksheet' tab is active, displaying the following SQL code:

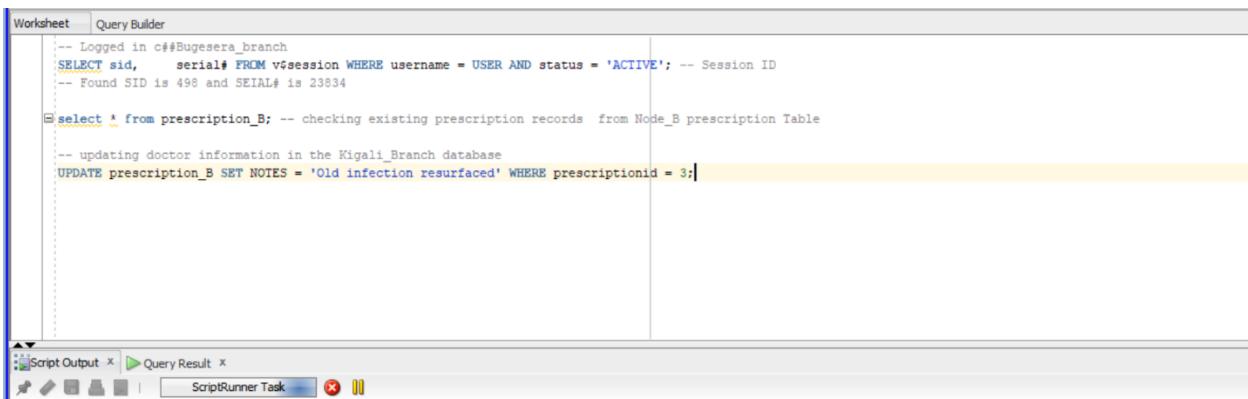
```
-- Found SID is 498 and SERIAL# is 23834
select * from prescription_B; -- checking existing prescription records  from Node_B prescription Table
```

Below the code, the 'Query Result' tab is selected, showing a table with the same six rows of prescription data as in Figure 14. The 'NOTES' column values are identical to those in Figure 14.

PREScriptionID	APPOINTMENTID	NOTES	DATEISSUED
1	3	4 Positive urine culture, hydrate well	05-MAY-25
2	4	5 Controlled diet, monitor glucose	28-AUG-25
3	7	8 Start antiepileptic therapy	01-OCT-25
4	9	10 Monitor BP daily, lifestyle changes advised, antihypertensive therapy initiated.	20-MAR-25
5	10	11 Advise reduced cholesterol diet, initiate anti-anginal therapy, schedule follow-up for stress test	05-FEB-25
6	11	12 Prescription for distributed transaction test	28-OCT-25

Figure 15: Before updating records at Node_B in session2

A5_3 Running update from session 2 – result shows task keep running



The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there are tabs for 'Worksheet' and 'Query Builder'. The main area contains the following SQL code:

```
-- Logged in C##Bugesera_branch
SELECT sid,      serial# FROM v$session WHERE username = 'USER' AND status = 'ACTIVE'; -- Session ID
-- Found SID is 498 and SERIAL# is 23834

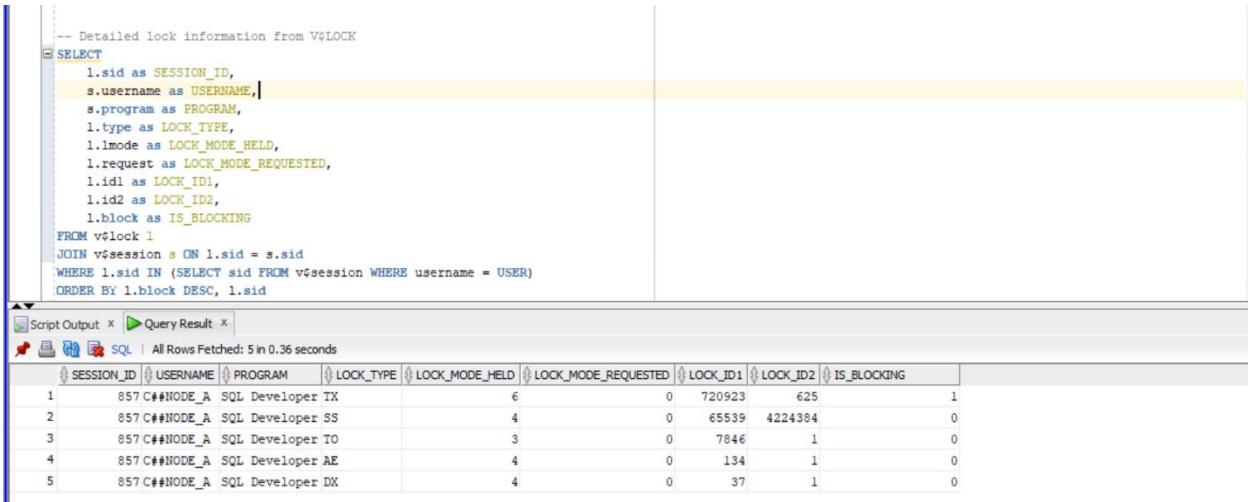
-- select * from prescription_B; -- checking existing prescription records   from Node_B prescription Table

-- updating doctor information in the Kigali_Branch database
UPDATE prescription_B SET NOTES = 'Old infection resurfaced' WHERE prescriptionid = 3;
```

Below the code, there are two tabs: 'Script Output' and 'Query Result'. The 'Query Result' tab is selected, showing the output of the update statement.

Figure 16: Session 2 attempt to update same records as done in session 1 -task keep running

A5_3 Checking waiting sessions



The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there are tabs for 'Worksheet' and 'Query Builder'. The main area contains the following SQL code:

```
-- Detailed lock information from V$LOCK
SELECT
    l.sid AS SESSION_ID,
    s.username AS USERNAME,
    s.program AS PROGRAM,
    l.type AS LOCK_TYPE,
    l.lmode AS LOCK_MODE_HELD,
    l.request AS LOCK_MODE_REQUESTED,
    l.id1 AS LOCK_ID1,
    l.id2 AS LOCK_ID2,
    l.block AS IS_BLOCKING
FROM v$lock l
JOIN v$session s ON l.sid = s.sid
WHERE l.sid IN (SELECT sid FROM v$session WHERE username = 'USER')
ORDER BY l.block DESC, l.sid
```

Below the code, there are two tabs: 'Script Output' and 'Query Result'. The 'Query Result' tab is selected, showing the output of the query. The results are displayed in a table:

SESSION_ID	USERNAME	PROGRAM	LOCK_TYPE	LOCK_MODE_HELD	LOCK_MODE_REQUESTED	LOCK_ID1	LOCK_ID2	IS_BLOCKING
1	857 C##NODE_A	SQL Developer	TX	6	0	720923	625	1
2	857 C##NODE_A	SQL Developer	SS	4	0	65539	4224384	0
3	857 C##NODE_A	SQL Developer	TO	3	0	7846	1	0
4	857 C##NODE_A	SQL Developer	AE	4	0	134	1	0
5	857 C##NODE_A	SQL Developer	DX	4	0	37	1	0

Figure 17: blocked and waiting sessions queried from Node_A session1

A5_4 releasing the lock made at session 1 by committing the update in session 1

The screenshot shows the Oracle SQL Developer interface. The top pane contains a script with four lines of SQL:

```
-- A5_2 running update to the same table from both node_A and node_B  
UPDATE prescription_B@proj_link SET NOTES = 'New infection detected' WHERE prescriptionid = 3;  
  
-- A5_4 Releasing the lock by committing update  
COMMIT;  
--  
-- A5_3 QUERY LOCK VIEWS TO SHOW BLOCKING SESSIONS
```

The bottom pane shows the results of the query:

Script Output X | Query Result X
Task completed in 0.143 seconds
https://docs.oracle.com/error-help/db/ora-02019/
1 row updated.

Commit complete.

Figure 18: Results of releasing session 1 lock

A5_4 session 2 update becomes successful after releasing a lock created in session 1

The screenshot shows the Oracle SQL Developer interface. The top pane contains a script with several lines of SQL:

```
-- Logged in c##Bugesera_branch  
SELECT sid, serial# FROM v$session WHERE username = 'USER' AND status = 'ACTIVE'; -- Session ID  
-- Found SID is 498 and SERIAL# is 23834  
  
select * from prescription_B; -- checking existing prescription records from Node_B prescription Table  
  
-- updating doctor information in the Kigali_Branch database  
UPDATE prescription_B SET NOTES = 'Old infection resurfaced' WHERE prescriptionid = 3;
```

The bottom pane shows the results of the query:

Script Output X | Query Result X
SQL | All Rows Fetched: 6 in 0.011 seconds

PREScriptionID	APPOINTMENTID	NOTES	DATEISSUED
1	3	4 Old infection resurfaced	05-MAY-25
2	4	5 Controlled diet, monitor glucose	28-AUG-25
3	7	8 Start antiepileptic therapy	01-OCT-25
4	9	10 Monitor BP daily, lifestyle changes advised, antihypertensive therapy initiated.	20-MAR-25
5	10	11 Advise reduced cholesterol diet, initiate anti-anginal therapy, schedule follow-up for stress test	05-FEB-25
	..		

A red arrow points to the 'NOTES' column of the first row, highlighting the updated value 'Old infection resurfaced'.

Figure 19: session2 updated successfully after reasing session1

Section B

B6_1 Checking existing null constraints in medication table

The screenshot shows a SQL Worksheet interface. In the top-left pane, there is a code editor containing the following SQL query:

```
-- Check existing constraints on Medication table
SELECT constraint_name, constraint_type, search_condition, status
FROM user_constraints
WHERE table_name = 'MEDICATION_A'
ORDER BY constraint_type, constraint_name;
```

In the bottom-right pane, the results of the query are displayed in a grid format:

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	STATUS
1 SYS_C008578	C	"MEDID" IS NOT NULL	ENABLED
2 SYS_C008579	C	"PRESCRIPTIONID" IS NOT NULL	ENABLED
3 SYS_C008580	C	"DRUGNAME" IS NOT NULL	ENABLED
4 SYS_C008581	C	"USAGE" IS NOT NULL	ENABLED
5 SYS_C008582	C	"DURATIONN" IS NOT NULL	ENABLED
6 SYS_C008583	C	"QUANTITY" IS NOT NULL	ENABLED
7 PK_MEDICATION_A	P	(null)	ENABLED

Figure 20: Checking existing null constraints in medication table

Checking existing null constraints in prescription table

The screenshot shows a SQL Worksheet interface. In the top-left pane, there is a code editor containing the following SQL query:

```
-- Check existing constraints on Prescription table
SELECT constraint_name, constraint_type, search_condition, status
FROM user_constraints
WHERE table_name = 'PRESCRIPTION_A'
ORDER BY constraint_type, constraint_name;
```

In the bottom-right pane, the results of the query are displayed in a grid format:

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	STATUS
1 SYS_C008575	C	"PRESCRIPTIONID" IS NOT NULL	ENABLED
2 SYS_C008576	C	"APPOINTMENTID" IS NOT NULL	ENABLED
3 SYS_C008577	C	"DATEISSUED" IS NOT NULL	ENABLED
4 PK_PRESCRIPTION_A	P	(null)	ENABLED

Figure 21: Checking existing null constraints in prescription table

B6_2 Prepare 2 failing and 2 passing INSERTs per table to validate rules but wrap failing ones in a block and ROLLBACK so committed rows stay within ≤10 total.

2_Passing Insertions on Prescription_A Table

```
-- Test INSERTs for Prescription table using existing constraints  
-- PASSING INSERT 1: Valid prescription that should work with current constraints  
INSERT INTO Prescription_A (PrescriptionID, AppointmentID, Notes, DateIssued)  
VALUES (1001, 1, 'Regular medication for blood pressure', SYSDATE);  
select * from prescription_a;  
commit;
```

Query Result | All Rows Fetched: 6 in 0.007 seconds

PRESCRIPTIONID	APPOINTMENTID	NOTES	DATEISSUED
1	5	6 Reduce salt intake, monitor BP	28-AUG-25
2	6	7 Antifungal therapy	20-AUG-25
3	8	9 Antiplatelet and physiotherapy	30-JUL-25
4	11	12 Patient scheduled for appendectomy. Pre-op antibiotics given, monitor vitals.	05-FEB-25
5	1001	1 Regular medication for blood pressure	28-OCT-25



Figure 22: First passing insert in prescription table

```
-- PASSING INSERT 2: Valid prescription within existing rules  
INSERT INTO Prescription_A (PrescriptionID, AppointmentID, Notes, DateIssued)  
VALUES (1002, 2, 'Follow-up prescription', SYSDATE - 1);  
select * from prescription_a;  
commit;
```

Query Result | All Rows Fetched: 6 in 0.007 seconds

PRESCRIPTIONID	APPOINTMENTID	NOTES	DATEISSUED
1	5	6 Reduce salt intake, monitor BP	28-AUG-25
2	6	7 Antifungal therapy	20-AUG-25
3	8	9 Antiplatelet and physiotherapy	30-JUL-25
4	11	12 Patient scheduled for appendectomy. Pre-op antibiotics given, monitor vitals.	05-FEB-25
5	1001	1 Regular medication for blood pressure	28-OCT-25
6	1002	2 Follow-up prescription	27-OCT-25



Figure 23: Second passing insert in prescription table

2_failing insertions on prescription table

The screenshot shows a SQL script in the top pane and its output in the bottom pane.

```
-- FAILING INSERT 1: Test what happens with NULL in required field
BEGIN
    INSERT INTO Prescription_A (PrescriptionID, AppointmentID, Notes, DateIssued)
    VALUES (1003, NULL, 'Test prescription with NULL AppointmentID', SYSDATE);
    commit;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('PRESCRIPTION FAIL 1: ' || SQLERRM);
        ROLLBACK;
END;
/
```

Script Output:

Task completed in 0.156 seconds

Error starting at line : 42 in command -
INSERT INTO Prescription_A (PrescriptionID, AppointmentID, Notes, DateIssued)
VALUES (1003, NULL, 'Test prescription with NULL AppointmentID', SYSDATE)
Error at Command Line : 43 Column : 19
Error report -
SQL Error: ORA-01400: cannot insert NULL into ("C##NODE_A"."PRESCRIPTION_A"."APPOINTMENTID")
[https://docs.oracle.com/error-help/db/ora-01400/01400. 00000 - "cannot insert NULL into \(%s\)"](https://docs.oracle.com/error-help/db/ora-01400/01400. 00000 -)
Cause: An attempt was made to insert NULL into previously listed objects.
Action: These objects cannot accept NULL values. Reservable columns cannot accept NULL values.

Figure 24: First failing insert operation on prescription table _ AppointmentID not existing

```

-- FAILING INSERT 2: Test invalid data based on existing constraints - unique primary key
BEGIN
    -- This will fail if there are CHECK constraints or foreign key violations
    INSERT INTO Prescription_A (PrescriptionID, AppointmentID, Notes, DateIssued)
    VALUES (1004, 9999, 'Test with non-existent AppointmentID', SYSDATE);
    commit;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('PRESCRIPTION FAIL 2: ' || SQLERRM);
        ROLLBACK;
END;
/

```

Script Output | Task completed in 0.067 seconds

```

Error starting at line : 56 in command -
INSERT INTO Prescription_A (PrescriptionID, AppointmentID, Notes, DateIssued)
    VALUES (1004, 9999, 'Test with non-existent AppointmentID', SYSDATE)
Error report -
ORA-00001: unique constraint (C#NODE_A.PK_PRESCRIPTION_A) violated
https://docs.oracle.com/error-help/db/ora-00001/

More Details :
https://docs.oracle.com/error-help/db/ora-00001/

```

Figure 25: Second failing insert on prescription due to unique constraint of PK

2_passing insertions on medication_A table

```

Worksheet | Query Builder
/
-- Test INSERTs for Medication table using existing constraints
-- PASSING INSERT 1: Valid medication
INSERT INTO Medication_A (MedID, PrescriptionID, DrugName, Dosage, DurationN, Quantity)
VALUES (5001, 1001, 'Paracetamol', '500mg', '7 days', '30 tablets');
select * from medication_a;

```

Query Result | Script Output | Query Result 1 | Query Result 2 | Query Result 3 | SQL | All Rows Fetched: 10 in 0.01 seconds

MEDID	PREScriptionID	DRUGNAME	DOSAGE	DURATIONN	QUANTITY
1	2	3 Amoxicillin	500 mg	7 days	14
2	5	3 Paracetamol	500 mg	As needed	10
3	6	7 Sodium Valproate	500 mg	long term	60
4	8	8 Aspirin	75 mg	Long term	90
5	11	9 Hydrochlorothiazide 25 mg 1 tab daily morning		Long term	90
6	13	10 Atorvastatin 20 mg	1 tab daily at night	Long term	30
7	18	5 Labetalol	100 mg	Ongoing take with food	15
8	20	6 Clotrimazole	500 mg	7 days	10
9	21	11 Ceftriaxone 1 g IV	Every 12 hours	2 days administer in hospital	2
10	5001	1001 Paracetamol	500mg	7 days	30 tablets

Figure 26: First working query on medication table

```

INSERT INTO Medication_A (MedID, PrescriptionID, DrugName, Dosage, DurationN, Quantity)
VALUES (5002, 1002, 'Amoxicillin', '250mg', '10 days', '20 capsules');
commit;
select * from medication_a;

-- FAILING INSERT 1: Tear NULL in required field

```

Script Output | Query Result | SQL | All Rows Fetched: 13 in 0.006 seconds

MEDID	PRESCRIPTIONID	DRUGNAME	DOSAGE	DURATIONN	QUANTITY
1	2	3 Amoxicillin	500 mg	7 days	14
2	5	3 Paracetamol	500 mg	As needed	10
3	6	7 Sodium Valproate	500 mg	long term	60
4	8	8 Aspirin	75 mg	Long term	90
5	11	9 Hydrochlorothiazide 25 mg	1 tab daily morning	Long term	90
6	13	10 Atorvastatin 20 mg	1 tab daily at night	Long term	30
7	18	5 Labetalol	100 mg	Ongoing take with food	15
8	20	6 Clotrimazole	500 mg	7 days	10
9	21	11 Ceftriaxone 1 g IV	Every 12 hours	2 days administer in hospital	2
10	5001	1001 Paracetamol	500mg	7 days	30 tablets
11	5002	1002 Amoxicillin	250mg	10 days	20 capsules
12	6001	1001 Vitamin C	750mg	30 days	45 tablets
13	7001	1001 Aspirin	100mg	7 days	21 tablets

Figure 27: Second working query on medication_A table

2_failing insertions on medication_A table

```

BEGIN
    INSERT INTO Medication_A (MedID, PrescriptionID, DrugName, Dosage, DurationN, Quantity)
    VALUES (5003, 1001, NULL, '500mg', '7 days', '30 tablets');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('MEDICATION FAIL 1: ' || SQLERRM);
        ROLLBACK;
END;
/

```

Query Result | Script Output | Task completed in 0.086 seconds

MEDICATION FAIL 1: ORA-01400: cannot insert NULL into ("C##NODE_A"."MEDICATION_A"."DRUGNAME")

PL/SQL procedure successfully completed.

Figure 28: First failing query on medication on Null Constraint

```

BEGIN
    INSERT INTO Medication_A (MedID, PrescriptionID, DrugName, Dosage, DurationN, Quantity)
    VALUES (5004, 9999, 'Ibuprofen', '400mg', '5 days', '15 tablets');

    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('MEDICATION FAIL 2: ' || SQLERRM);
            ROLLBACK;
END;
/

```

Query Result x Script Output x Query Result 1 x Query Result 2 x Query Result 3 x

Task completed in 0.087 seconds

MEDICATION FAIL 2: ORA-00001: unique constraint (C#NODE_A.PK_MEDICATION_A) violated

PL/SQL procedure successfully completed.

Figure 29: Second failing query on medication on unique PK constraint

B6_3 Committed rows

```

-- Verify we stay within $10 committed rows budget
SELECT
    'Prescription' AS Table_Name,
    COUNT(*) AS Committed_Rows
FROM Prescription_A
WHERE PrescriptionID IN (1001, 1002)
UNION ALL
SELECT
    'Medication',
    COUNT(*)
FROM Medication_A
WHERE MedID IN (5001, 5002)
UNION ALL
SELECT
    'TOTAL_COMMITED',
    (SELECT COUNT(*) FROM Prescription_A WHERE PrescriptionID IN (1001, 1002)) +
    (SELECT COUNT(*) FROM Medication_A WHERE MedID IN (5001, 5002))
FROM dual;

```

Query Result x Script Output x Query Result 1 x Query Result 2 x

SQL | All Rows Fetched: 3 in 0.061 seconds

TABLE_NAME	COMMITTED_ROWS
1 Prescription	2
2 Medication	2
3 TOTAL_COMMITED	4

Figure 30: Committed rows

B7_1 Refer to the B7 SQL code file

B7_2 Correct recomputed totals for prescription done by a statement level trigger

```

-- 1. Show current medication counts per prescription
SELECT PrescriptionID, COUNT(*) as Medication_Count
FROM Medication_A
GROUP BY PrescriptionID
ORDER BY PrescriptionID;

-- 2. Show audit entries (should have 2-3 rows)
SELECT * FROM Prescription_AUDIT
ORDER BY changed_at;

-- 3. Verify total committed rows stay within budget
SELECT
    (SELECT COUNT(*) FROM Medication_A WHERE MedID IN (6001)) as Remaining_Medications,
    (SELECT COUNT(*) FROM Prescription_AUDIT) as Audit_Rows,
    (SELECT COUNT(*) FROM Medication_A WHERE MedID IN (6001)) +
    (SELECT COUNT(*) FROM Prescription_AUDIT) as Total_New_Rows
FROM dual;

```

Script Output | Query Result | Query Result 1 | Query Result 2 |

SQL | All Rows Fetched: 1 in 0.007 seconds

	REMAINING_MEDICATIONS	AUDIT_ROWS	TOTAL_NEW_ROWS
1	1	4	5

Figure 31: Showing recomputed totals for prescription

B7_3 Audited prescriptions with 3 entries

```

--- B7_3
SELECT * FROM Prescription_AUDIT WHERE ROWNUM <=3

```

Script Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3 |

SQL | All Rows Fetched: 3 in 0.051 seconds

	BEF_TOTAL	AFT_TOTAL	CHANGED_AT	KEY_COL
1	2	2	28-OCT-25 10.47.22.080000000 PM	Prescription_1001
2	3	3	28-OCT-25 10.47.22.091000000 PM	Prescription_1001
3	3	3	28-OCT-25 10.47.22.100000000 PM	Prescription_1001

Figure 32: Showing audited prescriptions with 3 entries

B8_1, 2 DDL + INSERTs for HIER (6–10 rows)

DEPARTMENT	DOCTOR_COUNT	TOTAL_APPOINTMENTS
1 Cardiology	2	0
2 Dr_Brown	1	0
3 Dr_Green	1	0
4 Dr_Jones	2	0
5 Dr_Smith	2	0
6 Neurology	2	0

Figure 33: Showing inserted 6 rows of DDL + INSERTs for HIER table

B8_3 Recursive WITH SQL and sample output (4 rows) shown according to what are available.

DOCTOR_OR_DEPT	ROOT_DEPARTMENT	HIERARCHY_LEVEL	APPOINTMENT_COUNT
1 Dr_Jones	Cardiology	2	0
2 Dr_Green	Cardiology	3	0
3 Dr_Smith	Neurology	2	0
4 Dr_Brown	Neurology	3	0

Figure 34: Showing 4 rows of recursive sample output

B8_4 Control aggregations validating rollup correctness

DEPARTMENT	DOCTOR_COUNT	TOTAL_APPOINTMENTS
1 Cardiology	2	0
2 Dr_Brown	1	0
3 Dr_Green	1	0
4 Dr_Jones	2	0
5 Dr_Smith	2	0
6 Neurology	2	0

Figure 35: Control aggregations

B9_1, 2 Created triple table and 8 rows content

select * from triple; -- show triple inserted contents		
S	P	O
1 Influenza	isA	ViralInfection
2 CommonCold	isA	ViralInfection
3 ViralInfection	isA	InfectiousDisease
4 BacterialPneumonia	isA	InfectiousDisease
5 Paracetamol	treats	Fever
6 Antiviral	treats	ViralInfection
7 Antibiotic	treats	BacterialInfection
8 BacterialPneumonia	isA	BacterialInfection

Figure 36: Triple domain facts

B9_3. Recursive inference query implementing transitive isA*

CONDITION	INFERRED_TYPES
1 BacterialPneumonia	BacterialInfection, BacterialInfection, InfectiousDisease, InfectiousDisease
2 CommonCold	InfectiousDisease, ViralInfection
3 Influenza	InfectiousDisease, ViralInfection
4 ViralInfection	InfectiousDisease

Figure 37: Result of recursive inference query

B9_4 Grouping counts prove inferred labels are consistent.

INFERRED_CATEGORY	CONDITION_COUNT	CONDITIONS
1 InfectiousDisease	4	BacterialPneumonia, CommonCold, Influenza, ViralInfection
2 ViralInfection	2	CommonCold, Influenza
3 BacterialInfection	1	BacterialPneumonia

Figure 38: Grouped inferred labels

B10_1 Business limits and rules

The screenshot shows an Oracle SQL Developer interface. In the top-left pane, there is a code editor with the following SQL script:

```
INSERT INTO BUSINESS_LIMITS VALUES (
    'MAX_MEDS_PER_PRESCRIPTION', -- Rule: Maximum medications per prescription
    3,                           -- Threshold: Only 3 medications allowed per prescription
    'Y'                          -- Active: Rule is enabled
);
COMMIT;
select * from business_limits;
```

In the bottom-right pane, there is a table named "BUSINESS_LIMITS" with three columns: RULE_KEY, THRESHOLD, and ACTIVE. The data is:

RULE_KEY	THRESHOLD	ACTIVE
MAX_MEDS_PER_PRESCRIPTION	3	Y

Figure 39: Business limit and one active rule inserted

B10_2 and B10_3 refer to the B10 SQL code file for implementation code

B10_4 Demonstrating passing DML cases

The screenshot shows an Oracle SQL Developer interface. In the top-left pane, there is a code editor with the following SQL script:

```
-- PASSING INSERT 1: First medication for prescription (within limit)
INSERT INTO Medication_A (MedID, PrescriptionID, DrugName, Dosage, DurationN, Quantity)
VALUES (7001, 1001, 'Aspirin', '100mg', '7 days', '21 tablets');
commit;
select * from Medication_A;
```

In the bottom-right pane, there is a table named "Medication_A" with six columns: MEDID, PRESCRIPTIONID, DRUGNAME, DOSAGE, DURATIONN, and QUANTITY. The data is:

MEDID	PRESCRIPTIONID	DRUGNAME	DOSAGE	DURATIONN	QUANTITY
4	8	Aspirin	75 mg	Long term	90
5	11	Hydrochlorothiazide	25 mg	1 tab daily morning	90
6	13	Atorvastatin	20 mg	1 tab daily at night	30
7	18	Labetalol	100 mg	Ongoing take with food	15
8	20	Clotrimazole	500 mg	7 days	10
9	21	Ceftriaxone	1 g IV	Every 12 hours	2 days administer in hospital
10	5001	Paracetamol	500mg	7 days	30 tablets
11	5002	Amoxicillin	250mg	10 days	20 capsules
12	6001	Vitamin C	750mg	30 days	45 tablets
13	7001	Aspirin	100mg	7 days	21 tablets

Figure 40: First passing DML case

B10_4 Demonstrating passing DML case

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there is a code editor window containing the following SQL script:

```
-- PASSING INSERT 2: Second medication for same prescription (within limit)
INSERT INTO Medication_A (MedID, PrescriptionID, DrugName, Dosage, DurationN, Quantity)
VALUES (7002, 1001, 'Vitamin D', '1000IU', '30 days', '30 tablets');
commit;
select * from Medication_A;
```

In the bottom-right pane, there is a "Query Result" window showing a table of medication data. The table has columns: MEDID, PRESCRIPTIONID, DRUGNAME, DOSAGE, DURATIONN, and QUANTITY. The data includes 14 rows, with the last row (MedID 14, PrescriptionID 7002, DrugName 'Vitamin D') highlighted by a red arrow pointing to it.

MEDID	PRESCRIPTIONID	DRUGNAME	DOSAGE	DURATIONN	QUANTITY
6	13	10 Atorvastatin 20 mg	1 tab daily at night	Long term	30
7	18	5 Labetalol	100 mg	Ongoing take with food	15
8	20	6 Clotrimazole	500 mg	7 days	10
9	21	11 Ceftriaxone 1 g IV	Every 12 hours	2 days administer in hospital	2
10	5001	1001 Paracetamol	500mg	7 days	30 tablets
11	5002	1002 Amoxicillin	250mg	10 days	20 capsules
12	6001	1001 Vitamin C	750mg	30 days	45 tablets
13	7001	1001 Aspirin	100mg	7 days	21 tablets
14	7002	1001 Vitamin D	1000IU	30 days	30 tablets

Figure 41: Second passing DML case

B10_4 Demonstrating failing DML case

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there is a code editor window containing the following SQL script:

```
INSERT INTO Medication_A (MedID, PrescriptionID, DrugName, Dosage, DurationN, Quantity)
VALUES (7002, 1001, 'Vitamin D', '1000IU', '30 days', '30 tablets');
commit;
select * from Medication_A;
```

In the bottom-right pane, there is a "Query Result" window showing an error message. The message indicates that an error occurred at line 103, specifically a business rule violation: "ORA-20001: Business rule violation: Maximum 3 medications allowed per prescription. Prescription ID: 1001". It also mentions trigger errors: "ORA-06512: at "C#NODE_A.TRG_MEDICATION_LIMIT", line 9" and "ORA-04088: error during execution of trigger 'C#NODE_A.TRG_MEDICATION_LIMIT'".

```
Error starting at line : 103 in command -
INSERT INTO Medication_A (MedID, PrescriptionID, DrugName, Dosage, DurationN, Quantity)
VALUES (7002, 1001, 'Vitamin D', '1000IU', '30 days', '30 tablets')
Error report -
ORA-20001: Business rule violation: Maximum 3 medications allowed per prescription. Prescription ID: 1001
ORA-06512: at "C#NODE_A.TRG_MEDICATION_LIMIT", line 9
ORA-04088: error during execution of trigger 'C#NODE_A.TRG_MEDICATION_LIMIT'

https://docs.oracle.com/error-help/db/ora-20001

More Details :
https://docs.oracle.com/error-help/db/ora-20001/
https://docs.oracle.com/error-help/db/ora-06512/
https://docs.oracle.com/error-help/db/ora-04088/
```

Figure 42: First failing DML case - maximum rules reached

B10_4 Demomstrating failing DML case

The screenshot shows the 'Script Output' tab of Oracle SQL Developer. It displays the following log:

```
PL/SQL procedure successfully completed.

FAILED INSERT 2: ORA-20001: Business rule violation: Maximum 3 medications allowed per prescription. Prescription ID: 1001
ORA-06512: at "C##NODE_A.TRG_MEDICATION_LIMIT", line 9
ORA-04088: error during execution of trigger 'C##NODE_A.TRG_MEDICATION_LIMIT'

PL/SQL procedure successfully completed.

Commit complete.
```

Figure 43: Second failing DML case - maximum rule violation

B10_4 Total committed rows

The screenshot shows the 'Script Output' and 'Query Result' tabs of Oracle SQL Developer. The script in the 'Script Output' tab is:

```
-- PASSING INSERT 1: First medication for prescription (within limit)
INSERT INTO Medication_A (MedID, PrescriptionID, DrugName, Dosage, DurationN, Quantity)
VALUES (7001, 1001, 'Aspirin', '100mg', '7 days', '21 tablets');
commit;
select * from Medication_A;

-- PASSING INSERT 2: Second medication for same prescription (within limit)
INSERT INTO Medication_A (MedID, PrescriptionID, DrugName, Dosage, DurationN, Quantity)
VALUES (7002, 1001, 'Vitamin D', '1000IU', '30 days', '30 tablets');
commit;
select * from Medication_A;
```

The 'Query Result' tab displays the committed rows:

MEDID	PRESCRIPTIONID	DRUGNAME	DOSAGE	DURATIONN	QUANTITY
6	13	10 Atorvastatin 20 mg	1 tab daily at night	Long term	30
7	18	5 Labetalol	100 mg	Ongoing take with food	15
8	20	6 Clotrimazole	500 mg	7 days	10
9	21	11 Ceftriaxone 1 g IV	Every 12 hours	2 days administer in hospital	2
10	5001	1001 Paracetamol	500mg	7 days	30 tablets
11	5002	1002 Amoxicillin	250mg	10 days	20 capsules
12	6001	1001 Vitamin C	750mg	30 days	45 tablets
13	7001	1001 Aspirin	100mg	7 days	21 tablets
14	7002	1001 Vitamin D	1000IU	30 days	30 tablets

Figure 44: Committed rows

B10 verifying laws enforcement of business limits and rules

The screenshot shows a SQL query editor interface with two tabs: "Script Output" and "Query Result". The "Query Result" tab is active, displaying the output of the following SQL code:

```
-- 1. Show current medication counts per prescription
SELECT
    PrescriptionID,
    COUNT(*) as Medication_Count
FROM Medication_A
GROUP BY PrescriptionID
ORDER BY PrescriptionID;

-- 2. Show committed medications (should only have 2 rows from passing inserts)
SELECT
```

The output table has two columns: "PRESCRIPTIONID" and "MEDICATION_COUNT". The data is as follows:

PRESCRIPTIONID	MEDICATION_COUNT
1	3
2	5
3	6
4	7
5	8
6	9
7	10
8	11
9	1001
10	1002

THE END