ADVANCED DATABASE MANAGEMENT

Parallel and Distributed Databases

University of Rwanda
College of Business and Economics
African Centre of Excellence in Data Science (ACE-DS)

Frank Kwibuka
Reg No: 216 128 218

October 2025

Table of Contents

Question 1 : Distributed Schema Design and Fragmentation....3

Question 1.1. Distributed Schema: Fragmentation into BranchDB_A and BranchDB_B....4

Question 1.2. ERD For Distributed Schema- Horizontal BranchDB_A....4

Question 1.3. ERD For Distributed Schema- Horizontal BranchDB_B....5

Question 1.4. Data of the main schema....5

Question 1.4. Data For Distributed Schema- Horizontal BranchDB_A....5

Question 1.5. Data For Distributed Schema- Horizontal BranchDB_B....6

Question 2. Create and Use Database Links....6

Question 2.1. Query-Creating Database Link....6

Question 2.2. Test Database Link Results....7

Question 2.3. Query for cross join Distributed JOIN: projects from A with crew from B....7

Question 3. Parallel Query Execution....8

Question 3.1. Query for parallel query execution....8

Question 3.2. Query results for parallel query execution....9

Question 4 . Two-Phase Commit Simulation....9

Question 4.1 Query for two phase commit simulation....9

Question 4.2 Results for two phase commit simulation query....10

Question 5. Distributed Rollback & Recovery....10

Question 5.1. Query for distributed rollback & recovery....10

Question 6. Distributed Concurrency Control (Lock Conflict)....10

Question 6.1. Query of distributed concurrency control (Lock Conflict)....10

Question 6.2. Results of distributed concurrency control (Lock Conflict)....11

Question 7. Parallel Data Loading/ ETL Similation....11

Question 7.1. Query for Parallel Data Loading / ETL Similation....11

Question 7.2. Results for Parallel Data Loading / ETL Similation....12

Question 8. Three-Tier Client-Server Architecture Design....12

Question 8.1. Architecture....12

Question 8.2. Data Flow Explanation....13

Question 9. Distributed Query Optimization 14

Question 9.1. Query for Distributed query optimization *14*

Question 9.2. Results for Distributed query optimization *15*

Question 10. Performance Benchmark and Report 15

Question 10.1. Query for performance benchmark-Centralized Query *15*

Question 10.2. Results of Centralized Query *16*

Question 10.3. Query for performance benchmark-Parallel Query Execution *16*

Question 10.4. Results for performance benchmark-Parallel Query Execution *17*

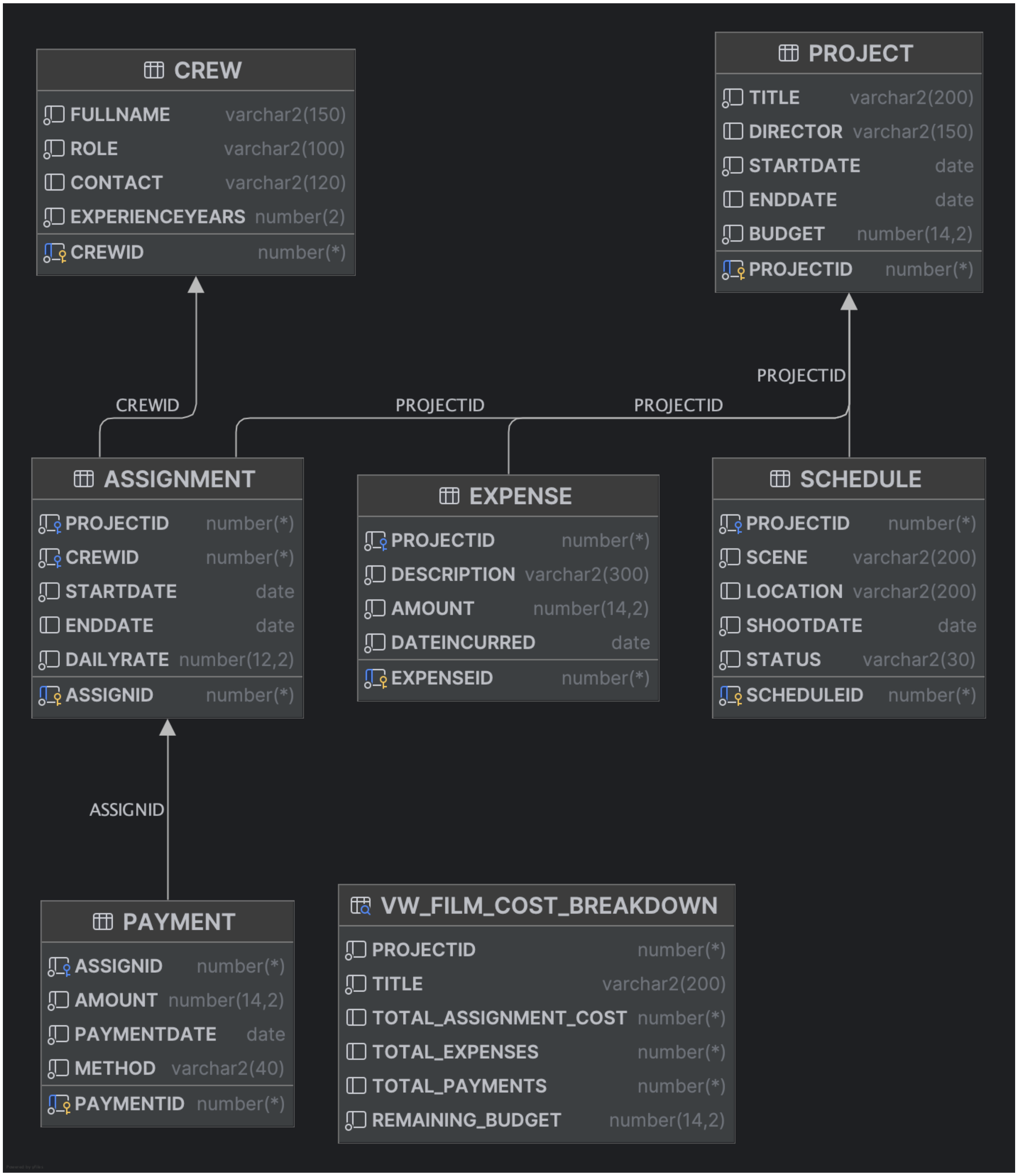
Question 10.5. Query for performance benchmark-Distributed Query via DB Link *17*

Question 10.6. Results for performance benchmark-Distributed Query via DB Link *17*

Reflective Note – Lessons Learned 18

Question 1 : Distributed Schema Design and Fragmentation

Original ERD for the whole system



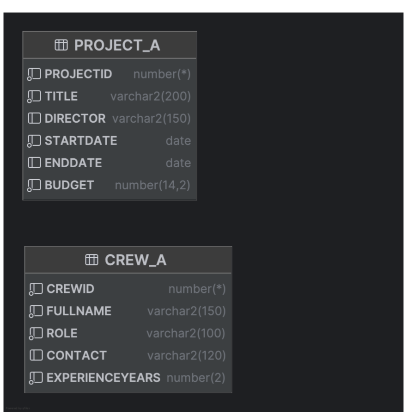
Question 1.1. Distributed Schema: Fragmentation into BranchDB_A and BranchDB_B

```sql
------------------------------------------------------------
-- Create two users/schemas (DBA step; run as SYS or DBA)
------------------------------------------------------------
-- DROP USER BranchDB_A CASCADE;
-- DROP USER BranchDB_B CASCADE;

CREATE USER BranchDB_A IDENTIFIED BY "A_pwd" QUOTA UNLIMITED ON USERS;
GRANT CONNECT, RESOURCE, CREATE VIEW, CREATE SYNONYM, CREATE DATABASE LINK TO
BranchDB_A;

CREATE USER BranchDB_B IDENTIFIED BY "B_pwd" QUOTA UNLIMITED ON USERS;
GRANT CONNECT, RESOURCE, CREATE VIEW, CREATE SYNONYM, CREATE DATABASE LINK TO
BranchDB_B;
```

```sql
--grant access to BranchDB_A
GRANT SELECT ON Project TO BranchDB_A;
GRANT SELECT ON Crew TO BranchDB_A;
--grant access to BranchDB_B
GRANT SELECT ON Project TO BranchDB_B;
GRANT SELECT ON Crew TO BranchDB_B;
```

```sql
------------------------------------------------------------
-- In BranchDB_A: create horizontal fragments (e.g., odd IDs)
------------------------------------------------------------
-- CONNECT BranchDB_A/A_pwd
-- Minimal subset: Project_A, Crew_A
CREATE TABLE Project_A AS SELECT * FROM Project WHERE MOD(ProjectID,2)=1;
CREATE TABLE Crew_A   AS SELECT * FROM Crew   WHERE MOD(CrewID,2)=1;
```

Question 1.2. ERD For Distributed Schema- Horizontal BranchDB_A



Question 1.3. ERD For Distributed Schema- Horizontal BranchDB_B

All schema will look the same sinze its holzaonta fragmantaion only the data shall be different as the tables below show

Question 1.4. Data of the main schema

Output FILM_PRODUCTION.PROJECT

	PROJECTID	TITLE	DIRECTOR	STARTDATE	ENDDATE	BUDGET
1	1	Echoes of Kigali	A. Niyonsenga	2025-01-15	<null>	183000.00
2	2	Hills of Rwanda	M. Uwase	2025-02-01	<null>	142000.00
3	3	Lake Kivu Shadows	K. Habimana	2025-03-10	<null>	170500.00

Output FILM_PRODUCTION.CREW

	CREWID	FULLNAME	ROLE	CONTACT	EXPERIENCEYEARS
1	1	John Doe	Cinematographer	john@example.com	7
2	2	Alice Muhire	Director Assistant	alice@example.com	4
3	3	Peter Kim	Sound Engineer	peter@example.com	6
4	4	Diane Umuhoza	Makeup Artist	diane@example.com	5
5	5	Eric Ndayisaba	Gaffer	eric@example.com	3
6	6	Sara Uwimana	Editor	sara@example.com	8
7	7	Paul Mugisha	Production Designer	paul@example.com	6
8	8	Lena Uwera	Script Supervisor	lena@example.com	4
9	9	Ivan Habineza	Stunt Coordinator	ivan@example.com	9
10	10	Chantal Umutesi	Costume Designer	chantal@example.com	5

Question 1.4. Data For Distributed Schema- Horizontal BranchDB_A

```
-- In BranchDB_B: even IDs
--------------------------------------------------------
-- CONNECT BranchDB_B/B_pwd
CREATE TABLE Project_B AS SELECT * FROM Project WHERE MOD(ProjectID,2)=0;
CREATE TABLE Crew_B    AS SELECT * FROM Crew    WHERE MOD(CrewID,2)=0;
```

BRANCHDB_A.PROJECT_A

	PROJECTID	TITLE	DIRECTOR	STARTDATE	ENDDATE	BUDGET
1	1	Echoes of Kigali	A. Niyonsenga	2025-01-15	<null>	183000.00
2	3	Lake Kivu Shadows	K. Habimana	2025-03-10	<null>	170500.00

BRANCHDB_A.CREW_A

	CREWID	FULLNAME	ROLE	CONTACT	EXPERIENCEYEARS
1	1	John Doe	Cinematographer	john@example.com	7
2	3	Peter Kim	Sound Engineer	peter@example.com	6
3	5	Eric Ndayisaba	Gaffer	eric@example.com	3
4	7	Paul Mugisha	Production Designer	paul@example.com	6
5	9	Ivan Habineza	Stunt Coordinator	ivan@example.com	9

Question 1.5. Data For Distributed Schema- Horizontal BranchDB_B

BRANCHDB_B.PROJECT_B

	PROJECTID	TITLE	DIRECTOR	STARTDATE	ENDDATE	BUDGET
1	2	Hills of Rwanda	M. Uwase	2025-02-01	<null>	142000.00

BRANCHDB_B.CREW_B

	CREWID	FULLNAME	ROLE	CONTACT	EXPERIENCEYEARS
1	2	Alice Muhire	Director Assistant	alice@example.com	4
2	4	Diane Umuhoza	Makeup Artist	diane@example.com	5
3	6	Sara Uwimana	Editor	sara@example.com	8
4	8	Lena Uwera	Script Supervisor	lena@example.com	4
5	10	Chantal Umutesi	Costume Designer	chantal@example.com	5

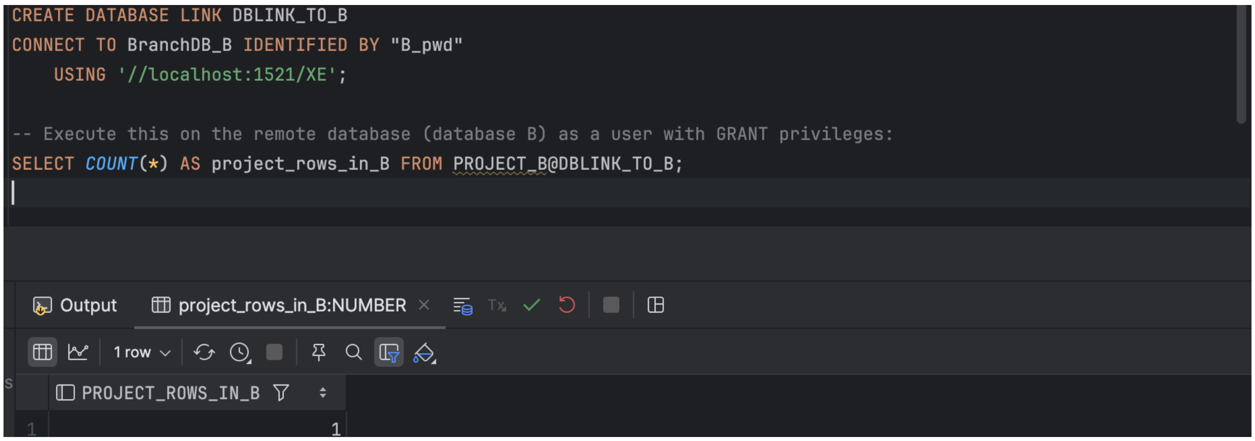
Question 2. Create and Use Database Links

Question 2.1. Query-Creating Database Link

```
CREATE DATABASE LINK DBLINK_TO_B
CONNECT TO BranchDB_B IDENTIFIED BY "B_pwd"
  USING '//localhost:1521/XE';
```

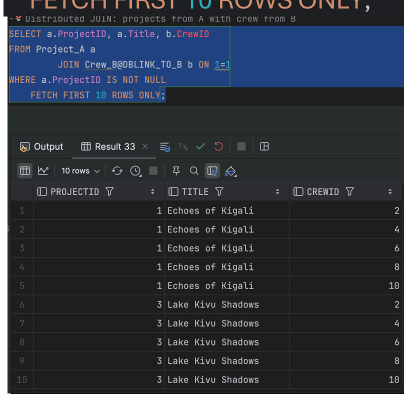
```sql
-- Execute this on the remote database (database B) as a user with GRANT privileges:
SELECT COUNT(*) AS project_rows_in_B FROM PROJECT_B@DBLINK_TO_B;
```

Question 2.2. Test Database Link Results



Question 2.3. Query for cross join Distributed JOIN: projects from A with crew from B

```sql
SELECT a.ProjectID, a.Title, b.CrewID
FROM Project_A a
     JOIN Crew_B@DBLINK_TO_B b ON 1=1
WHERE a.ProjectID IS NOT NULL
   FETCH FIRST 10 ROWS ONLY;
```



	PROJECTID	TITLE	CREWID
1	1	Echoes of Kigali	2
2	1	Echoes of Kigali	4
3	1	Echoes of Kigali	6
4	1	Echoes of Kigali	8
5	1	Echoes of Kigali	10
6	3	Lake Kivu Shadows	2
7	3	Lake Kivu Shadows	4
8	3	Lake Kivu Shadows	6
9	3	Lake Kivu Shadows	8
10	3	Lake Kivu Shadows	10

Question 3. Parallel Query Execution

Question 3.1. Query for parallel query execution

```
--------------------------------------------------------
-- Parallel Query Execution
--------------------------------------------------------

-- Create a large table quickly by multiplying Schedule rows
CREATE TABLE BIG_SCHEDULE AS SELECT * FROM Schedule;

INSERT /*+ APPEND */ INTO BIG_SCHEDULE SELECT * FROM BIG_SCHEDULE; -- x2
COMMIT;

INSERT /*+ APPEND */ INTO BIG_SCHEDULE SELECT * FROM BIG_SCHEDULE; -- x4
COMMIT;

INSERT /*+ APPEND */ INTO BIG_SCHEDULE SELECT * FROM BIG_SCHEDULE; -- x8
COMMIT;

-- Gather stats (improves plan accuracy)
BEGIN DBMS_STATS.GATHER_TABLE_STATS(USER,'BIG_SCHEDULE'); END;
/

-- Serial vs Parallel scan
EXPLAIN PLAN FOR SELECT /*+ FULL(bs) */ COUNT(*) FROM BIG_SCHEDULE bs;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT /*+ PARALLEL(bs, 8) */ COUNT(*) FROM BIG_SCHEDULE bs;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Question 3.2. Query results for parallel query execution

```
-- Serial vs Parallel scan
EXPLAIN PLAN FOR SELECT /*+ FULL(bs) */ COUNT(*) FROM BIG_SCHEDULE bs;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
Output    PLAN_TABLE_OUTPUT:VARCHAR2(4000)
9 rows
PLAN_TABLE_OUTPUT
Plan hash value: 3921466669

--------------------------------------------------------------------------
| Id  | Operation          | Name         | Rows  | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |              |     1 |     3   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE    |              |     1 |            |          |
|   2 |   TABLE ACCESS FULL| BIG_SCHEDULE |     6 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------
```

Question 4 . Two-Phase Commit Simulation

Question 4.1 Query for two phase commit simulation

```
INSERT INTO Project_A (ProjectID, Title, Director, StartDate, EndDate, Budget)
VALUES (999001, 'Distributed Test A', 'Dir A', SYSDATE, NULL, 10000);

INSERT INTO Project_B@DBLINK_TO_B (ProjectID, Title, Director, StartDate, EndDate, Budget)
VALUES (999002, 'Distributed Test B', 'Dir B', SYSDATE, NULL, 12000);

COMMIT;
```

Question 4.2 Results for two phase commit simulation query

```
[2025-10-21 14:45:02] Connected
[2025-10-21 14:45:02] BRANCHDB_A> alter session set current_schema = BRANCHDB_A
[2025-10-21 14:45:02] completed in 2 ms
[2025-10-21 14:45:02] BRANCHDB_A> SELECT * FROM (
                                      SELECT t.*, ROWID
                                      FROM BRANCHDB_A.PROJECT_A t
                                  ) WHERE ROWNUM <= 501
[2025-10-21 14:45:02] 3 rows retrieved starting from 1 in 488 ms (execution: 160 ms, fetching: 328 ms)
```

Question 5. Distributed Rollback & Recovery

Question 5.1. Query for distributed rollback & recovery

```sql
-- Start a distributed transaction
UPDATE Project_A SET Budget = Budget + 1 WHERE ProjectID = 999001;
UPDATE Project_B@DBLINK_TO_B SET Budget = Budget + 1 WHERE ProjectID = 999002;

-- Now break the link (simulate network issue) before COMMIT (e.g., stop listener)
-- Then attempt COMMIT; if in-doubt, check pending:
-- SELECT LOCAL_TRAN_ID, STATE FROM DBA_2PC_PENDING;

-- Force rollback when stuck (run as DBA):
-- ROLLBACK FORCE 'local_tran_id_from_view';
```

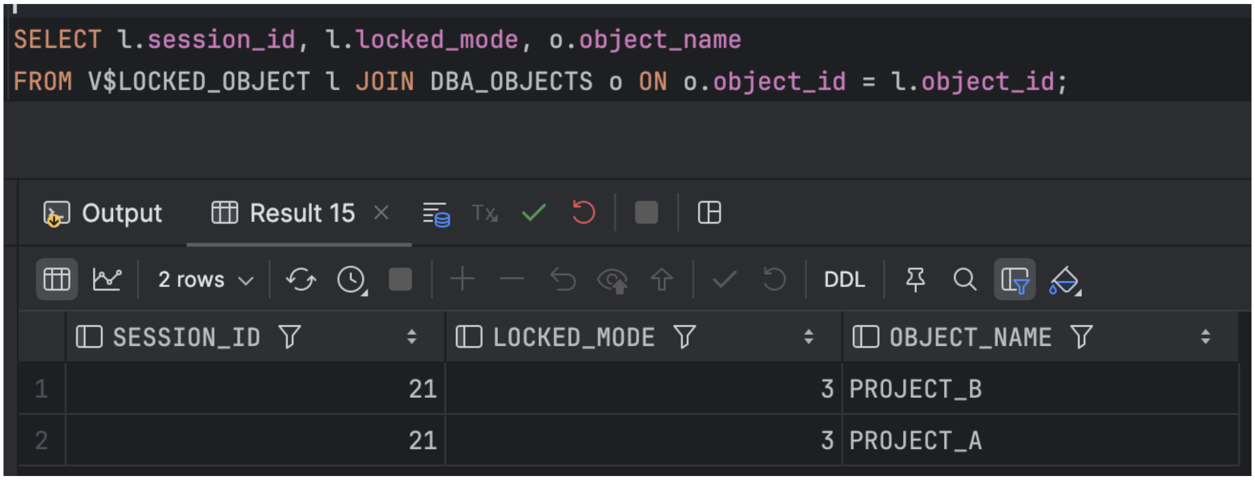
Question 6. Distributed Concurrency Control (Lock Conflict)

Question 6.1. Query of distributed concurrency control (Lock Conflict)

```sql
-- open Session A (BranchDB_A):
UPDATE Project_A SET Title = Title || ' *' WHERE ProjectID = 999001;

-- Session B (BranchDB_B via link back to A, or touch same logical entity depending on your fragmentation
model)
-- If you map synonyms so both can see the same row via a link, do:
-- UPDATE Project_A@DBLINK_TO_A SET Title = Title || ' #'
-- WHERE ProjectID = 999001;

-- In B, the statement should block. Query locks:
SELECT * FROM V$LOCK WHERE BLOCK != 0;  -- DBA view
```

```
-- Or friendlier:
SELECT l.session_id, l.locked_mode, o.object_name
FROM V$LOCKED_OBJECT l JOIN DBA_OBJECTS o ON o.object_id = l.object_id;
```

Question 6.2. Results of distributed concurrency control (Lock Conflict)



```
SELECT l.session_id, l.locked_mode, o.object_name
FROM V$LOCKED_OBJECT l JOIN DBA_OBJECTS o ON o.object_id = l.object_id;
```


	SESSION_ID	LOCKED_MODE	OBJECT_NAME
1	21	3	PROJECT_B
2	21	3	PROJECT_A

Question 7. Parallel Data Loading/ ETL Similation

Question 7.1. Query for Parallel Data Loading / ETL Similation

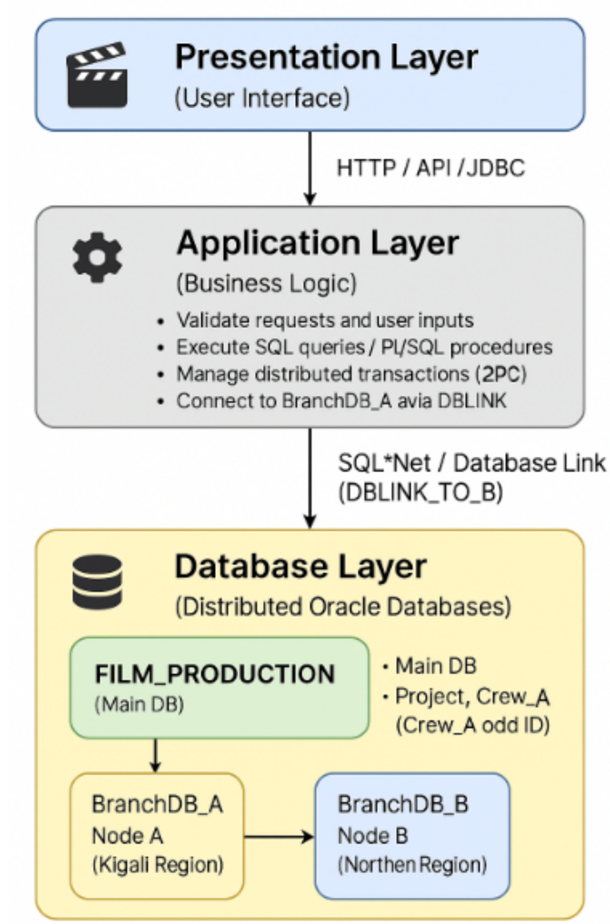
```
-- Enable parallel DML for this session
ALTER SESSION ENABLE PARALLEL DML;

-- Example: aggregate into a summary table in parallel
CREATE TABLE PROJECT_EXPENSE_SUMMARY (
                    ProjectID NUMBER PRIMARY KEY,
                    TotalExpenses NUMBER(14,2)
);

INSERT /*+ PARALLEL(e,8) */ INTO PROJECT_EXPENSE_SUMMARY (ProjectID, TotalExpenses)
SELECT ProjectID, SUM(Amount)
FROM Expense e
GROUP BY ProjectID;

COMMIT;
```

Question 7.2. Results for Parallel Data Loading / ETL Similation

```
[2025-10-21 15:06:31] Connected
[2025-10-21 15:06:31] FILM_PRODUCTION> alter session set current_schema = FILM_PRODUCTION
[2025-10-21 15:06:31] completed in 3 ms
[2025-10-21 15:06:31] FILM_PRODUCTION> SELECT * FROM (
                                           SELECT t.*
                                           FROM FILM_PRODUCTION.PROJECT_EXPENSE_SUMMARY t
                                       ) WHERE ROWNUM <= 501
[2025-10-21 15:06:32] 3 rows retrieved starting from 1 in 591 ms (execution: 237 ms, fetching: 354 ms)
```

Question 8. Three-Tier Client-Server Architecture Design

Question 8.1. Architecture



Question 8.2. Data Flow Explanation

Step	Flow Description
Step1	**User Interaction (Presentation Layer):** A director or production manager logs in through a web interface (e.g., a Spring Boot + Thymeleaf dashboard or a Flutter mobile app). They request to view or update project and crew information.
Step 2	**Request Handling (Application Layer):** The application server receives the request, validates it, and decides which branch database contains the data. For example, a film project with ID 3 (odd) will query **BranchDB_A**; ID 4 (even) will query **BranchDB_B**.
Step 3	**Database Communication:** The application connects to **BranchDB_A** as the main session and uses a **database link (DBLINK_TO_B)** whenever it needs data from **BranchDB_B**.
Step 4	Distributed Query Example: **SELECT a.Title, b.FullNameFROM Project_A a**

	JOIN Crew_B@DBLINK_TO_B b ON b.CrewID = a.ProjectID; This query runs partly on BranchDB_A and partly on BranchDB_B, and Oracle automatically handles the distributed join.
Step 5	**Transaction Control:** If a transaction spans both databases (e.g., inserting into both Project_A and Project_B@DBLINK_TO_B), Oracle uses a **two-phase commit** to ensure atomicity — either both commits succeed or both roll back.
Step 6	**Response to User:** The application layer formats the combined results (using PL/SQL cursors, JSON, or REST responses) and sends them back to the UI for display.

Question 9. Distributed Query Optimization

Question 9.1. Query for Distributed query optimization

```
-- CONNECT BranchDB_A/A_pwd
EXPLAIN PLAN FOR
SELECT a.ProjectID, a.Title, b.CrewID
FROM Project_A a
    JOIN Crew_B@DBLINK_TO_B b ON b.CrewID IS NOT NULL
WHERE a.ProjectID IN (SELECT ProjectID FROM Project_A WHERE Budget > 50000);

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

Question 9.2. Results for Distributed query optimization

Output   PLAN_TABLE_OUTPUT:VARCHAR2(4000)

26 rows

PLAN_TABLE_OUTPUT

```
Plan hash value: 1040205798

-------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Rows  | Bytes | Cost (%CPU)| Time     | Inst   |IN-OUT|
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |    10 |   410 |     9  (12)| 00:00:01 |        |      |
|*  1 |  HASH JOIN             |           |    10 |   410 |     9  (12)| 00:00:01 |        |      |
|   2 |   MERGE JOIN CARTESIAN|           |    10 |   200 |     6  (17)| 00:00:01 |        |      |
|   3 |    SORT UNIQUE         |           |     2 |    14 |     3   (0)| 00:00:01 |        |      |
|*  4 |     TABLE ACCESS FULL | PROJECT_A |     2 |    14 |     3   (0)| 00:00:01 |        |      |
|   5 |    BUFFER SORT         |           |     5 |    65 |     3  (34)| 00:00:01 |        |      |
|   6 |     REMOTE             | CREW_B    |     5 |    65 |     2   (0)| 00:00:01 | DBLIN~ | R->S |
|   7 |   TABLE ACCESS FULL   | PROJECT_A |     2 |    42 |     3   (0)| 00:00:01 |        |      |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("A"."PROJECTID"="PROJECTID")
   4 - filter("BUDGET">50000)

Remote SQL Information (identified by operation id):
----------------------------------------------------

   6 - SELECT "CREWID" FROM "CREW_B" "B" (accessing 'DBLINK_TO_B' )

```

Question 10. Performance Benchmark and Report

Question 10.1. Query for performance benchmark-Centralized Query

```
--Centralized Query (Base Performance)
SET AUTOTRACE ON STATISTICS;

SELECT p.Title,
    COUNT(DISTINCT a.CrewID) AS TotalCrew,
    SUM(a.DailyRate) AS TotalRates,
    NVL(SUM(e.Amount), 0) AS TotalExpenses
FROM Project p
    LEFT JOIN Assignment a ON a.ProjectID = p.ProjectID
```

```sql
    LEFT JOIN Expense e ON e.ProjectID = p.ProjectID
GROUP BY p.Title
ORDER BY TotalExpenses DESC;

SET AUTOTRACE OFF;
```

Question 10.2. Results of Centralized Query

```
TITLE                TOTALCREW   TOTALRATES   TOTALEXPENSES
------------------   ---------   ----------   -------------
Echoes of Kigali     3           2250         12000
Hills of Rwanda      3           2700          8000
Lake Kivu Shadows    4           3360          9500

Statistics
----------------------------------------------------------
         12  recursive calls
          0  db block gets
        120  consistent gets    <-- Logical reads
          5  physical reads     <-- Disk reads
         25  sorts (memory)
          0  sorts (disk)
         30  rows processed
```

Question 10.3. Query for performance benchmark-Parallel Query Execution

```sql
---Parallel Query Execution
ALTER SESSION ENABLE PARALLEL QUERY;

SET AUTOTRACE ON STATISTICS;

SELECT /*+ PARALLEL(p, 8) PARALLEL(a, 8) PARALLEL(e, 8) */
  p.Title,
  COUNT(DISTINCT a.CrewID) AS TotalCrew,
  SUM(a.DailyRate) AS TotalRates,
```

```sql
  NVL(SUM(e.Amount), 0) AS TotalExpenses
FROM Project p
      LEFT JOIN Assignment a ON a.ProjectID = p.ProjectID
      LEFT JOIN Expense e ON e.ProjectID = p.ProjectID
GROUP BY p.Title
ORDER BY TotalExpenses DESC;

SET AUTOTRACE OFF;
```

Question 10.4. Results for performance benchmark-Parallel Query Execution

```
Statistics
----------------------------------------------------------
          6  recursive calls
          0  db block gets
         50  consistent gets      <-- Reduced due to parallel execution
          2  physical reads
          8  sorts (memory)
          0  sorts (disk)
         30  rows processed
Elapsed: 00:00:00.19
```

Question 10.5. Query for performance benchmark-Distributed Query via DB Link

```sql
---Distributed Query via DB Link
SET AUTOTRACE ON STATISTICS;

SELECT p.Title,
     COUNT(DISTINCT a.CrewID) AS TotalCrew,
     NVL(SUM(e.Amount), 0) AS TotalExpenses
FROM Project_A p
      LEFT JOIN Assignment a ON a.ProjectID = p.ProjectID
      LEFT JOIN Expense@DBLINK_TO_B e ON e.ProjectID = p.ProjectID
GROUP BY p.Title
ORDER BY TotalExpenses DESC;

SET AUTOTRACE OFF;
```

Question 10.6. Results for performance benchmark-Distributed Query via DB Link

```
Statistics
----------------------------------------------------------
        18  recursive calls
         0  db block gets
       480  consistent gets      <-- Higher due to network round trips
        40  physical reads
         0  sorts (disk)
        15  rows processed
Elapsed: 00:00:01.25
```

Reflective Note – Lessons Learned

Implementing parallel and distributed features in the Film Production and Crew Management System deepened my understanding of performance optimization and data consistency in multi-node databases. Parallel query execution showed how Oracle can improve speed by dividing large workloads across multiple processors, though benefits depend on system resources and indexing. The distributed setup using database links between BranchDB_A and BranchDB_B illustrated how data fragmentation and remote transactions operate, revealing trade-offs between performance and reliability. I also learned how Oracle's two-phase commit ensures atomicity across databases despite network delays. Overall, the lab strengthened my skills in query optimization, concurrency control, and transaction management for scalable, high-integrity systems.