MPUNGA EMMANUEL

Reg No: 224019555

African Centre of Excellence in Data Science (ACE-DS)

Masters of Data Science in Mining

UR-CBE Gikondo Campus

Module: Advanced Database and Technology

```
-- 1. Create all six tables with PK, FK, CHECK constraints
-- 1. AUTHOR TABLE
-------------------------------------------------------
CREATE TABLE Author (
   AuthorID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   FullName VARCHAR2(100) NOT NULL,
   Nationality VARCHAR2(50),
   BirthYear NUMBER(4) CHECK (BirthYear BETWEEN 1900 AND EXTRACT(YEAR FROM SYSDATE))
);


-------------------------------------------------------
-- 2. BOOK TABLE
-------------------------------------------------------
CREATE TABLE Book (
   BookID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   Title VARCHAR2(150) NOT NULL,
   AuthorID NUMBER NOT NULL,
   Genre VARCHAR2(50),
```

```
    PublicationYear NUMBER(4) CHECK (PublicationYear BETWEEN 1900 AND EXTRACT(YEAR
FROM SYSDATE)),
    Status VARCHAR2(20) DEFAULT 'Available' CHECK (Status IN ('Available','Borrowed')),
    CONSTRAINT fk_book_author FOREIGN KEY (AuthorID) REFERENCES Author(AuthorID)
);

-----------------------------------------------------
-- 3. MEMBER TABLE
-----------------------------------------------------
CREATE TABLE Member (
    MemberID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    FullName VARCHAR2(100) NOT NULL,
    Contact VARCHAR2(15),
    Address VARCHAR2(200),
    Email VARCHAR2(100)
);

-----------------------------------------------------
-- 4. STAFF TABLE
-----------------------------------------------------
CREATE TABLE Staff (
    StaffID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    FullName VARCHAR2(100) NOT NULL,
    Role VARCHAR2(50),
    Phone VARCHAR2(15),
    Shift VARCHAR2(20) CHECK (Shift IN ('Morning','Evening'))
);

-----------------------------------------------------
-- 5. BORROW TABLE
```

```
-----------------------------------------------------

CREATE TABLE Borrow (

   BorrowID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

   BookID NUMBER NOT NULL,

   MemberID NUMBER NOT NULL,

   StaffID NUMBER NOT NULL,

   BorrowDate DATE DEFAULT SYSDATE,

   DueDate DATE,

   ReturnDate DATE,

   CONSTRAINT fk_borrow_book FOREIGN KEY (BookID) REFERENCES Book(BookID),

   CONSTRAINT fk_borrow_member FOREIGN KEY (MemberID) REFERENCES
Member(MemberID),

   CONSTRAINT fk_borrow_staff FOREIGN KEY (StaffID) REFERENCES Staff(StaffID)

);

-----------------------------------------------------

-- 6. FINE TABLE (CASCADE DELETE from Borrow)

-----------------------------------------------------

CREATE TABLE Fine (

   FineID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

   BorrowID NUMBER NOT NULL,

   Amount NUMBER(8,2) CHECK (Amount >= 0),

   PaymentDate DATE,

   Status VARCHAR2(20) CHECK (Status IN ('Pending','Paid')),

   CONSTRAINT fk_fine_borrow FOREIGN KEY (BorrowID) REFERENCES Borrow(BorrowID) ON
DELETE CASCADE

);

-- Insert sample data


-- Insert Authors

INSERT INTO Author (FullName, Nationality, BirthYear)
```

```
VALUES ('John Smith', 'American', 1975);

INSERT INTO Author VALUES (DEFAULT, 'Jane Doe', 'British', 1980);

INSERT INTO Author VALUES (DEFAULT, 'Paul Nduwayezu', 'Rwandan', 1965);

INSERT INTO Author VALUES (DEFAULT, 'Maria Lopez', 'Spanish', 1972);

INSERT INTO Author VALUES (DEFAULT, 'Ken Tanaka', 'Japanese', 1985);


-- Insert Books

INSERT INTO Book (Title, AuthorID, Genre, PublicationYear, Status)

VALUES ('Database Systems', 1, 'Technology', 2019, 'Available');

INSERT INTO Book VALUES (DEFAULT, 'Machine Learning Basics', 2, 'AI', 2020, 'Available');

INSERT INTO Book VALUES (DEFAULT, 'Rwanda History', 3, 'History', 2018, 'Available');

INSERT INTO Book VALUES (DEFAULT, 'Advanced SQL', 1, 'Technology', 2021, 'Available');

INSERT INTO Book VALUES (DEFAULT, 'Data Mining Concepts', 2, 'AI', 2022, 'Available');

INSERT INTO Book VALUES (DEFAULT, 'Modern Java', 4, 'Programming', 2020, 'Available');

INSERT INTO Book VALUES (DEFAULT, 'Web Development', 5, 'Technology', 2019, 'Available');

INSERT INTO Book VALUES (DEFAULT, 'Python for Data Science', 1, 'Programming', 2022,
'Available');

INSERT INTO Book VALUES (DEFAULT, 'Library Management', 3, 'Education', 2021, 'Available');

INSERT INTO Book VALUES (DEFAULT, 'AI Ethics', 2, 'AI', 2023, 'Available');


-- Insert Members

INSERT INTO Member (FullName, Contact, Address, Email)

VALUES ('Alice Mukamana', '0788000001', 'Kigali', 'alice@gmail.com');

INSERT INTO Member VALUES (DEFAULT, 'John Doe', '0788000002', 'Huye', 'john@gmail.com');

INSERT INTO Member VALUES (DEFAULT, 'Sarah Uwase', '0788000003', 'Musanze',
'sarah@gmail.com');

INSERT INTO Member VALUES (DEFAULT, 'David Nkurunziza', '0788000004', 'Kicukiro',
'david@gmail.com');

INSERT INTO Member VALUES (DEFAULT, 'Grace Niyonsaba', '0788000005', 'Nyagatare',
'grace@gmail.com');
```

```
INSERT INTO Member VALUES (DEFAULT, 'Paul Habimana', '0788000006', 'Gasabo',
'paul@gmail.com');

INSERT INTO Member VALUES (DEFAULT, 'Mary Iradukunda', '0788000007', 'Nyarugenge',
'mary@gmail.com');

INSERT INTO Member VALUES (DEFAULT, 'James Mugisha', '0788000008', 'Rubavu',
'james@gmail.com');

INSERT INTO Member VALUES (DEFAULT, 'Clara Uwera', '0788000009', 'Rusizi',
'clara@gmail.com');

INSERT INTO Member VALUES (DEFAULT, 'Eric Bizimana', '0788000010', 'Rwamagana',
'eric@gmail.com');


-- Insert Staff

INSERT INTO Staff (FullName, Role, Phone, Shift)

VALUES ('Jacques Hakizimana', 'Librarian', '0787000001', 'Morning');

INSERT INTO Staff VALUES (DEFAULT, 'Maria Umutoni', 'Assistant', '0787000002', 'Evening');


-- Insert Borrow Records

INSERT INTO Borrow (BookID, MemberID, StaffID, BorrowDate, DueDate, ReturnDate)

VALUES (1, 1, 1, SYSDATE-10, SYSDATE-3, NULL);

INSERT INTO Borrow VALUES (DEFAULT, 2, 2, 2, SYSDATE-20, SYSDATE-10, SYSDATE-8);

INSERT INTO Borrow VALUES (DEFAULT, 3, 3, 1, SYSDATE-5, SYSDATE+2, NULL);


-- Update borrowed book status

UPDATE Book SET Status='Borrowed' WHERE BookID IN (1,3);


-- 4. Retrieve all books currently borrowed and their due dates


-- List books not yet returned

SELECT b.Title, m.FullName AS Borrower, br.DueDate

FROM Borrow br

JOIN Book b ON br.BookID = b.BookID
```

```
JOIN Member m ON br.MemberID = m.MemberID
WHERE br.ReturnDate IS NULL;

-- Update a fine record when payment is received

-- Assume FineID=1 was paid
UPDATE Fine
SET PaymentDate = SYSDATE,
   Status = 'Paid'
WHERE FineID = 1;

-- List members with overdue returns beyond 7 days

SELECT m.FullName, b.Title, br.DueDate, (SYSDATE - br.DueDate) AS DaysOverdue
FROM Borrow br
JOIN Member m ON br.MemberID = m.MemberID
JOIN Book b ON br.BookID = b.BookID
WHERE br.ReturnDate IS NULL
  AND SYSDATE > br.DueDate + 7;

  -- 7. Create a view summarizing fines collected by month.
CREATE OR REPLACE VIEW Monthly_Fine_Summary AS
SELECT TO_CHAR(PaymentDate, 'YYYY-MM') AS Month_Year,
      SUM(Amount) AS Total_Collected
FROM Fine
WHERE Status = 'Paid'
GROUP BY TO_CHAR(PaymentDate, 'YYYY-MM');
```

```
-- 8. Implement a trigger to mark a book as ‘Available’ upon return.

CREATE OR REPLACE TRIGGER trg_update_book_status
AFTER UPDATE OF ReturnDate ON Borrow
FOR EACH ROW
WHEN (NEW.ReturnDate IS NOT NULL)
BEGIN
   UPDATE Book
   SET Status = 'Available'
   WHERE BookID = :NEW.BookID;
END;
/

-- Horizontal Fragmentation (Distributed Simulation)
-- Horizontal fragmentation using RANGE rule
-- Borrow_A will store Borrow records for MemberID <= 5
-- Borrow_B will store Borrow records for MemberID > 5

CREATE TABLE Borrow_A AS
SELECT * FROM Borrow WHERE MemberID <= 5;

CREATE TABLE Borrow_B AS
SELECT * FROM Borrow WHERE MemberID > 5;

-- Verify fragments
SELECT COUNT(*) AS Rows_in_A FROM Borrow_A;
SELECT COUNT(*) AS Rows_in_B FROM Borrow_B;

-- 10. Replication Example (Read Replica Simulation)
```

```
-- Replica of Book table for read operations (e.g., remote reporting)
CREATE TABLE Book_REPLICA AS
SELECT * FROM Book;

-- Example of periodic sync (manual simulation)
CREATE OR REPLACE PROCEDURE sync_book_replica AS
BEGIN
   DELETE FROM Book_REPLICA;
   INSERT INTO Book_REPLICA SELECT * FROM Book;
END;
/
-- 11. Distributed Query Example

-- Assume you have a remote node named SITE_B with a DB link
-- Example: CREATE DATABASE LINK siteB CONNECT TO user IDENTIFIED BY password USING 'SITEB';

-- Distributed join between local Borrow table and remote Member table
SELECT b.BorrowID, bk.Title, m.FullName
FROM Borrow b
JOIN Book bk ON b.BookID = bk.BookID
JOIN Member@mylink m ON b.MemberID = m.MemberID;

-- 12. Parallel Query Processing

-- Enable parallel execution for performance testing
ALTER SESSION ENABLE PARALLEL DML;

-- Execute a parallel query to compute fine totals
```

```
SELECT /*+ PARALLEL(f, 4) */
      TO_CHAR(PaymentDate, 'YYYY-MM') AS Month,
      SUM(Amount) AS Total_Collected
FROM Fine f
GROUP BY TO_CHAR(PaymentDate, 'YYYY-MM');


--13 Audit Table and Trigger for Distributed Updates
CREATE TABLE Borrow_AUDIT (
   bef_total NUMBER,
   aft_total NUMBER,
   changed_at TIMESTAMP DEFAULT SYSTIMESTAMP,
   source_node VARCHAR2(20)
);
-- Trigger that logs total borrow counts before and after each transaction
CREATE OR REPLACE TRIGGER trg_borrow_audit
AFTER INSERT OR UPDATE OR DELETE ON Borrow
DECLARE
   v_before NUMBER;
   v_after NUMBER;
BEGIN
   SELECT COUNT(*) INTO v_before FROM Borrow_AUDIT;
   SELECT COUNT(*) INTO v_after FROM Borrow;


   INSERT INTO Borrow_AUDIT (bef_total, aft_total, source_node)
   VALUES (v_before, v_after, 'NODE_A');
END;
/


-- 14. Reconstructing Distributed Data (Global View)
```

```
CREATE OR REPLACE VIEW Borrow_GLOBAL AS
SELECT * FROM Borrow_A
UNION ALL
SELECT * FROM Borrow_B;
```

Practical Tasks and Marking Guide

1. **Distributed Schema Design and Fragmentation.**

```
-- Borrow_A: MemberID <= 5
CREATE TABLE Borrow_A AS
SELECT * FROM Borrow WHERE MemberID <= 5;

-- Borrow_B: MemberID > 5
CREATE TABLE Borrow_B AS
SELECT * FROM Borrow WHERE MemberID > 5;
```

2 – **Create and Use Database Links (2 Marks)**

```
-- Create database link (replace with actual credentials)
CREATE DATABASE LINK BranchDB_B
CONNECT TO username IDENTIFIED BY password
USING 'BranchDB_B';
-- Remote SELECT
SELECT * FROM Member@BranchDB_B;

-- Distributed join
SELECT b.BorrowID, m.FullName, bk.Title
FROM Borrow_A b
JOIN Book bk ON b.BookID = bk.BookID
JOIN Member@BranchDB_B m ON b.MemberID = m.MemberID;
```

Task 3 – Parallel Query Execution (2 Marks)

```
-- Enable parallel execution
ALTER SESSION ENABLE PARALLEL DML;


-- Serial query
SELECT COUNT(*) FROM Fine;
-- Parallel query with hint
SELECT /*+ PARALLEL(Fine, 8) */ COUNT(*) FROM Fine;
```

Task 4 – Two-Phase Commit Simulation (2 Marks)

```
SET AUTOCOMMIT OFF;
BEGIN
  -- Insert on Node A
  INSERT INTO Borrow_A (BookID, MemberID, StaffID, BorrowDate, DueDate)
  VALUES (1, 1, 1, SYSDATE, SYSDATE+7);
  -- Insert on Node B via DB link
  INSERT INTO Borrow_B@BranchDB_B (BookID, MemberID, StaffID, BorrowDate, DueDate)
  VALUES (2, 6, 2, SYSDATE, SYSDATE+7);
  COMMIT; -- 2PC ensures both commits succeed or rollback
END;
/
```

Task 5 – Distributed Rollback and Recovery (2 Marks)

```
SELECT * FROM DBA_2PC_PENDING;
```

Task 1 – Distributed Schema Design and Fragmentation (2 Marks)

Description: Split your database into two logical nodes using horizontal fragmentation.

SQL Template:

```
-- Borrow_A: MemberID <= 5
CREATE TABLE Borrow_A AS
SELECT * FROM Borrow WHERE MemberID <= 5;
```

-- Borrow_B: MemberID > 5

CREATE TABLE Borrow_B AS

SELECT * FROM Borrow WHERE MemberID > 5;

Deliverables: ER Diagram, SQL scripts.

Task 2 – Create and Use Database Links (2 Marks)

Description: Create DB link between nodes and demonstrate distributed queries.

SQL Template:

CREATE DATABASE LINK BranchDB_B

CONNECT TO username IDENTIFIED BY password

USING 'BranchDB_B';

-- Remote SELECT

SELECT * FROM Member@BranchDB_B;

-- Distributed join

SELECT b.BorrowID, m.FullName, bk.Title

FROM Borrow_A b

JOIN Book bk ON b.BookID = bk.BookID

JOIN Member@BranchDB_B m ON b.MemberID = m.MemberID;

Task 3 – Parallel Query Execution (2 Marks)

Description: Compare serial vs parallel query performance.

SQL Template:

ALTER SESSION ENABLE PARALLEL DML;

-- Serial query

SELECT COUNT(*) FROM Fine;

-- Parallel query

SELECT /*+ PARALLEL(Fine, 8) */ COUNT(*) FROM Fine;

Task 4 – Two-Phase Commit Simulation (2 Marks)

Description: Perform inserts on two nodes and commit once.

SQL Template:

SET AUTOCOMMIT OFF;

BEGIN

INSERT INTO Borrow_A (BookID, MemberID, StaffID, BorrowDate, DueDate)

VALUES (1, 1, 1, SYSDATE, SYSDATE+7);

INSERT INTO Borrow_B@BranchDB_B (BookID, MemberID, StaffID, BorrowDate, DueDate)

VALUES (2, 6, 2, SYSDATE, SYSDATE+7);

COMMIT;

END;

/

Task 5 – Distributed Rollback and Recovery (2 Marks)

Description: Simulate network failure and resolve using ROLLBACK FORCE.

SQL Template:

-- Check pending transactions

SELECT * FROM DBA_2PC_PENDING;

-- Resolve transaction

ROLLBACK FORCE 'transaction_id';

Task 6 – Distributed Concurrency Control (2 Marks)

Description: Demonstrate lock conflict with two sessions.

SQL Template:

-- Session 1

UPDATE Member SET Contact='0788009999' WHERE MemberID=1;

-- leave open

-- Session 2

UPDATE Member@BranchDB_B SET Contact='0788008888' WHERE MemberID=1;

-- Check locks

SELECT * FROM DBA_LOCKS;

Task 7 – Parallel Data Loading / ETL Simulation (2 Marks)

Description: Perform parallel data aggregation using PARALLEL DML.

SQL Template:

ALTER SESSION ENABLE PARALLEL DML;

SELECT /*+ PARALLEL(Fine,8) */ MemberID, SUM(Amount) AS TotalFines

FROM Fine

GROUP BY MemberID;

Task 8 – Three-Tier Client–Server Architecture Design (2 Marks)

Description: Draw and explain a 3-tier architecture.

Deliverables: Diagram showing Presentation, Application, Database layers and DB link interaction.

Task 9 – Distributed Query Optimization (2 Marks)

Description: Use EXPLAIN PLAN and DBMS_XPLAN.DISPLAY to analyze distributed join.

SQL Template:

EXPLAIN PLAN FOR

SELECT b.BorrowID, bk.Title, m.FullName

FROM Borrow_A b

JOIN Book bk ON b.BookID = bk.BookID

JOIN Member@BranchDB_B m ON b.MemberID = m.MemberID;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

Task 10 – Performance Benchmark and Report (2 Marks)

Description: Run complex query centralized, parallel, and distributed. Compare time and I/O.

SQL Template:

SET AUTOTRACE ON;

SELECT ... -- complex query

SET AUTOTRACE OFF;