**✅ Q1: Bank Database ER + DDL + Views + Index + Sequence**

**1. ER Diagram (Entities & Relationships)**

Entities:

* **Customer** (CustomerID, Name, Address, Phone)
* **Account** (AcctNo, Type, Balance)
* **Branch** (BranchID, BranchName, Location)
* **Loan** (LoanID, Amount, CustomerID)
* **Transaction** (TransID, AcctNo, Amount, Date, Type)

**Relationships:**

* A **Customer** can have multiple **Accounts**
* An **Account** belongs to one **Branch**
* A **Customer** can take **Loans**
* **Transactions** are performed on **Accounts**

**2. Create Tables with Constraints (DDL)**

CREATE TABLE Customer (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

Address VARCHAR(200),

Phone VARCHAR(15)

);

CREATE TABLE Branch (

BranchID INT PRIMARY KEY,

BranchName VARCHAR(100),

Location VARCHAR(100)

);

CREATE TABLE Account (

AcctNo INT PRIMARY KEY,

Type VARCHAR(50),

Balance DECIMAL(10,2),

CustomerID INT,

BranchID INT,

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),

FOREIGN KEY (BranchID) REFERENCES Branch(BranchID)

);

CREATE TABLE Loan (

LoanID INT PRIMARY KEY,

Amount DECIMAL(10,2),

CustomerID INT,

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)

);

CREATE TABLE Transaction (

TransID INT PRIMARY KEY,

AcctNo INT,

Amount DECIMAL(10,2),

Date DATE,

Type VARCHAR(20),

FOREIGN KEY (AcctNo) REFERENCES Account(AcctNo)

);

**3. Create Views (Using JOINs)**

-- View 1: Customer and Account Info

CREATE VIEW Customer\_Account\_View AS

SELECT c.CustomerID, c.Name, a.AcctNo, a.Type, a.Balance

FROM Customer c

JOIN Account a ON c.CustomerID = a.CustomerID;

-- View 2: Customer and Loan Info

CREATE VIEW Customer\_Loan\_View AS

SELECT c.CustomerID, c.Name, l.LoanID, l.Amount

FROM Customer c

JOIN Loan l ON c.CustomerID = l.CustomerID;

**4. Create Index (CustomerId ordered by Customer name)**

CREATE INDEX idx\_CustomerName ON Customer(Name ASC);

**5. Create Sequence for AcctNo**

(For Oracle-style DBMS)

CREATE SEQUENCE acctno\_seq

START WITH 1001

INCREMENT BY 1

NOCACHE

NOCYCLE;

**✅ Q2: Company Database ER + DDL + Views + Index + Sequence**

-- Table creation

CREATE TABLE Department (

DeptID INT PRIMARY KEY,

DeptName VARCHAR(50)

);

CREATE TABLE Employee (

EmpID INT PRIMARY KEY,

EmpName VARCHAR(100),

Salary DECIMAL(10,2),

DeptID INT,

FOREIGN KEY (DeptID) REFERENCES Department(DeptID)

);

CREATE TABLE Project (

ProjID INT PRIMARY KEY,

ProjName VARCHAR(50),

DeptID INT,

FOREIGN KEY (DeptID) REFERENCES Department(DeptID)

);

CREATE TABLE Works\_On (

EmpID INT,

ProjID INT,

Hours INT,

PRIMARY KEY (EmpID, ProjID),

FOREIGN KEY (EmpID) REFERENCES Employee(EmpID),

FOREIGN KEY (ProjID) REFERENCES Project(ProjID)

);

-- Views

CREATE VIEW EmpDept\_View AS

SELECT e.EmpName, d.DeptName FROM Employee e

JOIN Department d ON e.DeptID = d.DeptID;

CREATE VIEW ProjectDept\_View AS

SELECT p.ProjName, d.DeptName FROM Project p

JOIN Department d ON p.DeptID = d.DeptID;

-- Index

CREATE INDEX idx\_Emp\_Dept ON Employee(DeptID ASC, EmpID ASC);

-- Sequence

CREATE SEQUENCE emp\_id\_seq START WITH 101 INCREMENT BY 1;

**✅ Q3: Trigger on Library Table for ISSUE/RETURN & Fine Calculation**

CREATE OR REPLACE TRIGGER trg\_library\_update

AFTER INSERT OR UPDATE ON Library

FOR EACH ROW

DECLARE

v\_days INT;

v\_fine INT;

BEGIN

v\_days := SYSDATE - :NEW.doi;

IF :NEW.status = 'RETURN' THEN

INSERT INTO Library\_Audit VALUES(:NEW.bid, :NEW.bname, :NEW.noc + 1, SYSDATE);

ELSIF :NEW.status = 'ISSUE' THEN

INSERT INTO Library\_Audit VALUES(:NEW.bid, :NEW.bname, :NEW.noc - 1, SYSDATE);

END IF;

IF v\_days > 20 THEN

UPDATE Library SET status = 'FINE' WHERE bid = :NEW.bid;

v\_fine := v\_days \* 10;

INSERT INTO Library\_Audit(bid, bname, noc, timestampofquery) VALUES(:NEW.bid, :NEW.bname, v\_fine, SYSDATE);

END IF;

END;

**✅ Q4: Trigger for Auditing Updates/Deletes on Library Table**

CREATE OR REPLACE TRIGGER trg\_library\_audit

AFTER DELETE OR UPDATE ON Library

FOR EACH ROW

BEGIN

INSERT INTO Library\_Audit(bid, bname, noc, timestampofquery)

VALUES(:OLD.bid, :OLD.bname, :OLD.noc, SYSDATE);

END;

**✅ Q5: JAVA CRUD Program for Student Table**

Connection con = DriverManager.getConnection("jdbc:mysql://localhost/db","root","");

Statement stmt = con.createStatement();

stmt.executeUpdate("INSERT INTO Students VALUES (1, 'Ram', 'Pune', 85)");

ResultSet rs = stmt.executeQuery("SELECT \* FROM Students");

while(rs.next()) {

System.out.println(rs.getInt(1) + " " + rs.getString(2));

}

**✅ Q6: PL/SQL Procedure + Function for Population Density Classification**

CREATE OR REPLACE FUNCTION get\_density(pincode INT) RETURN VARCHAR IS

area FLOAT;

pop INT;

density FLOAT;

BEGIN

SELECT area, population INTO area, pop FROM CitiesIndia WHERE pincode = pincode;

IF pop < 10 OR pop > 25718 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Invalid population range');

END IF;

density := pop / area;

IF density > 3000 THEN

RETURN 'High Density';

ELSIF density BETWEEN 1000 AND 2999 THEN

RETURN 'Moderate';

ELSE

RETURN 'Low Density';

END IF;

END;

**✅ Q7: Database Navigation using JDBC (Java Example)**

// Edit student details by ID

PreparedStatement ps = con.prepareStatement("UPDATE Students SET stud\_name = ? WHERE stud\_id = ?");

ps.setString(1, "Updated Name");

ps.setInt(2, 101);

ps.executeUpdate();

**✅ Q8: PL/SQL Procedure + Function for Class Assignment + Exception Handling**

CREATE OR REPLACE FUNCTION get\_class(total INT) RETURN VARCHAR IS

BEGIN

IF total BETWEEN 990 AND 1499 THEN

RETURN 'Distinction';

ELSIF total BETWEEN 900 AND 989 THEN

RETURN 'First Class';

ELSIF total BETWEEN 825 AND 899 THEN

RETURN 'Higher Second';

ELSIF total BETWEEN 750 AND 824 THEN

RETURN 'Pass';

ELSE

RETURN 'Fail';

END IF;

END;

-- Exception handling block

BEGIN

FOR s IN (SELECT \* FROM Student) LOOP

IF s.Marks1 < 0 OR s.Marks1 > 100 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Invalid Marks');

END IF;

total := s.Marks1 + s.Marks2 + s.Marks3 + s.Marks4 + s.Marks5;

INSERT INTO Result VALUES(s.rollno, total, get\_class(total));

END LOOP;

END;

**✅ Q9: MapReduce in MongoDB (sites)**

db.sites.mapReduce(

function() { emit(this.url, 1); },

function(key, values) { return Array.sum(values); },

{ out: "site\_access\_count" }

)

**✅ Q10: Library ER + DDL + Views + Index + Sequence**

Same as Q1 with modified names.

**✅ Q11: PL/SQL Control + Exception Handling for Borrowing System**

Covered already in Q8-style block with status and fine logic. Combine triggers from Q3 + Q4.

**✅ Q12: MongoDB with Python**

from pymongo import MongoClient

client = MongoClient()

db = client.bank

db.customers.insert\_one({'name': 'Ram', 'balance': 5000})

**✅ Q13: SQL Objects on Borrower**

CREATE VIEW Borrower\_View AS SELECT \* FROM Borrower;

CREATE INDEX idx\_borrower ON Borrower(Name);

CREATE SEQUENCE borrower\_seq START WITH 1 INCREMENT BY 1;

CREATE SYNONYM Brr FOR Borrower;

**✅ Q14: 10 SQL Queries with Joins, Views, Subqueries**

Examples:

* SELECT \* FROM Student WHERE Marks1 > 90;
* SELECT \* FROM Student s JOIN Result r ON s.rollno = r.RollNo;
* SELECT rollno FROM Student WHERE Marks1 = (SELECT MAX(Marks1) FROM Student);

**✅ Q15: COMPANY Queries**

-- 1

SELECT p.Pnumber, d.Dno, e.Lname, e.Address, e.Bdate

FROM PROJECT p JOIN DEPARTMENT d ON p.Dno = d.Dno

JOIN EMPLOYEE e ON d.Mgr\_ssn = e.Ssn

WHERE p.Plocation = 'Stafford';

-- 2

(Use UNION of workers and managers on 'Smith')

**✅ Q16 to Q20: Account, Loan, Borrower Queries**

Same schema repeated:

SELECT AVG(balance) FROM Account GROUP BY branch\_name;

SELECT COUNT(\*) FROM Depositor GROUP BY branch\_name;

SELECT \* FROM Loan WHERE branch\_name='Akurdi' AND amount > 12000;

Use UNION, INTERSECT, MINUS for set-based queries.

**✅ Q21 to Q24: Schema + DML + Condition Queries**

Mostly involve:

* JOINs across Customers, Deposit, and Borrow
* WHERE clauses with city match (e.g., Customers.city = 'Pune')
* MAX(), MIN(), and GROUP BY clauses

**✅ Q25: MongoDB CRUD on Orders**

db.Orders.insertOne({cust\_id: 1, amount: 2000, status: 'Pending'});

db.Orders.updateOne({cust\_id: 1}, {$set: {status: 'Paid'}});

db.Orders.deleteOne({cust\_id: 1});

**✅ Q26: MongoDB Aggregation + Indexing**

db.Employee.createIndex({emp\_name: 1});

db.Employee.aggregate([{$group: {\_id: "$emp\_dept", total: {$sum: "$salary"}}}]);

**✅ Q27 to Q28: Same as earlier schemas with aggregate queries and DML**

Use GROUP BY, MAX(), SUM() with filters on location/date.

**✅ Q29: MongoDB Games Collection**

db.games.insertMany([...]);

db.games.find();

db.games.find().sort({score: -1}).limit(3);

db.games.find({achievements: {$all: ['Game Master', 'Speed Demon']}});

**✅ Q30: PL/SQL for Tax Calculation**

CREATE OR REPLACE PROCEDURE calc\_tax IS

BEGIN

FOR r IN (SELECT \* FROM Employee\_salary) LOOP

tax := r.gross\_salary \* 0.1;

DBMS\_OUTPUT.PUT\_LINE('Name: ' || r.emp\_no || ' Tax: ' || tax);

END LOOP;

END;

**✅ Q31: PL/SQL Fine Logic - Already covered (Same as Q11)**

**✅ Q32: Procedure for Count by Percent Range**

-- Use CASE WHEN for percentage and COUNT(\*) GROUP BY

**✅ Q33: Merge Data using Cursor**

CURSOR c1 IS SELECT \* FROM N\_RollCall;

FOR rec IN c1 LOOP

IF NOT EXISTS (SELECT \* FROM O\_RollCall WHERE id = rec.id) THEN

INSERT INTO O\_RollCall VALUES(rec.id, rec.name);

END IF;

END LOOP;

**✅ Q34: proc\_Grade Stored Procedure**

Already handled in Q8 with get\_class() function logic.

**✅ Q35/Q36: MongoDB Aggregation + Index**

db.Citydetails.aggregate([

{$group: {\_id: "$name", total\_pop: {$sum: "$population.total"}}}

]);

db.Citydetails.createIndex({name: 1});