MarwadiUniversity

FacultyofComputer Applications

Subject Details

Academic Year:	2025-2026	**Subject Code:**	05MC0105
Course:	M.C.A.	**Subject Name:**	RDBMS
Semester:	1	**Division:**	A

Course Work-Extended Case Study Title:

"Design and Implementation of a Bank Management Database Using SQL&PL/SQL"

Group Details:

Sr. No.	Roll No	Student Name	Signature of Student
1	92500584046	Raj Vadaviya	
2	92500584053	Parth Ghelani	
3	92500584032	Harshit Vora	

Submission Details:

Subject Faculty	
Date	
Signature	

MarwadiUniversity

FacultyofComputer Applications

Requirement analysis

1.Customer Management

The system must allow storing, updating, and removing customer details, including name, gender, date of birth, contact number, email, and address.

2.Account Management

The system should support creating and managing various bank account types. It must maintain account balance, opening date, and link each account to a valid customer.

3.Transaction Processing

The system must record and process all types of banking transactions such as deposits, withdrawals, and fund transfers while updating account balances accordingly.

4.Employee & Branch Management

The system should maintain employee details, handle staff assignments, and manage branch-level data such as branch location, IFSC code, and branch manager.

5.Loan Management

The system must allow applying for, approving, and maintaining loan records. It should store loan amount, duration, interest rate, and ensure each loan is linked to both customer and responsible employee.

6.Automated Operations via Triggers

The system must automatically enforce business rules and system validations through triggers, such as balance checks, auto-updating logs, and preventing invalid operations.

MarwadiUniversity

FacultyofComputer Applications

Database Dictionary:

1.CUSTOMER

Field name	Data type	Constraint	Description
CustomerID	NUMBER(5)	Primary key	This field store unique ID of each Customer.
FirstName	VARCHAR2(20)	Not null	This field hold first name of each customer. This field must not be empty
LastName	VARCHAR2(20)	NOT NULL	This field hold Last name of each customer.
Gender	CHAR(1)	Check(Gender)	This field hold genderdeatail.
DOB	DATE		Date of birth of customer.
Phone	Varchar2(15)	UNIQUE	This field hold mobile number.
Email	VARCHAR2(50)	NOT NULL	This field holds Email Id of customer
Address	VARCHAR2(100)		This field holds address of customer

2.Accounts

Field name	Data type	Constraint	Description
Account_NO	NUMBER(10)	PRIMARY KEY	Unique numbers for each account holder
Customer_ID	NUMBER(5)	FOREIGN KEY PREFERENCE Customer(Customer Id)	Id that linked with account holder(customer)
Account_Type	VARCHAR(15)	CHECK (Account_Type IN ('Savings','Current','Fixed Deposit'))	Type of customer's account
Balance	VARCHAR2(12,2)	Not Null	Accpunt holder's balance

MarwadiUniversity

FacultyofComputer Applications

3.Transaction

Field name	Data type	Constraint	Description
Tans_ID	NUMBER(10)	PRIMARY KEY	Unique ID for each transaction
Account_No	NUMBER(10)	FOREIGN KEY REFERENCES Account(Account_No)	Numbervthat linked with account
Transaction_Type	VARCHAR(10)	CHECK (Trans_Type IN ('Deposit','Withdraw','Transfer'))	
Balance	VARCHAR2(12,2)	CHECK (Amount > 0)	Which type type of transaction
Transaction_Date	DATE	DEFAULTSYSDATE	Date of transsaction

4.Employee

Field name	Data type	Constraint	Description
Emp_Id	NUMBER(5)	PRIMARY KEY	Unique ID for employee
Emp_Name	VARCAHR2(30)	NOT NULL	This field is holds employee name
Salary	NUMBER2(50)	CHECK (Salary >0)	Employee salary
Email	VARCHAR2(50)		Employee email id
Designation	VARCHAR2(20)	CHECK (Designation IN ('Manager','Clerk','Cashier','Officer'))	Each Employe are in which job role

MarwadiUniversity

FacultyofComputer Applications

5.Branch

Field name	Data type	Constraint	Description
Branch_ID	NUMBER(5)	PRIMARY KEY	Unique ID for each bank branch
Branch_Name	VARCHAR2(50)	NOT NULL	Name of branch
Location	VARCHAR2(100)	NOT NULL	Address of the branch
IFSC_Code	VARCHAR2(15)	UNIQUE	Branch IFDC code
Manager_ID	NUMBER	FOREIGN KEY REFERENCES (Emp_ID)	Manager assigned

6.Loan

Field name	Data type	Constraint	Description
Loan_ID	NUMBER(6)	PRIMARY KEY	Unique Loan ID
Customer_ID	NUMBER(5)	FOREIGN KEY REFERENCES Customer(Customer_ID)	Customer (Borrower)
Emp_ID	NUMBER(5)	FOREIGN KEY REFERENCES Employee(Emp_ID)	Employee handling loan
Lan_Type	VARCHAR2(30)	CHECK (Loan_Type IN ('Home','Personal','Education','Vehicle'))	Type of Loan
Amount	NUMBER(12,2)	CHECK (Amount > 0)	Loan Amount
Interest_Rate	NUMBER(5,2)		Interest rate percentage
Duration_Months	NUMBER(3)		Loan period in months

MarwadiUniversity

FacultyofComputer Applications

7.Transaction_Audit

Field name	Data type	Constraint	Description
Audit_ID	NUMBER(10)	PRIMARY KEY	Unique audit record ID
Trans_ID	NUMBER(10)	FOREIGN KEY REFERENCES Transaction(Trans_ID)	Related transaction ID
Account_No	NUMBER(10)	FOREIGN KEY REFERENCES Account(Account_No)	Account number involved in the transaction
Action	VARCHAR2(20)	CHECK (Action IN ('DEPOSIT','WITHDRAW','TRANSFER','UPDATE'))	Type of action performed
Old_Balance	NUMBER(14,2)		Account balance before the transaction
New_Balance	NUMBER(14,2)		Account balance after the transaction

MarwadiUniversity

FacultyofComputer Applications

Entity Relationship Diagram (ERD)

E–R Diagram for the Bank Management System Schema that includes:

- Composite attribute
- Multivalued attribute
- Derived attribute
- Mapping cardinality limits

Key Enhancements for ER Model

To satisfy your requirements:

- **Composite Attribute**→*Guest Name* (split into *FirstName* + *LastName*).
- **Multivalued Attribute**→*Guest Preferences* (since a guest can have multiple preferences).
- **Derived Attribute**→*StayDuration* (derived from CheckOutDate – CheckInDate in Reservations).

Entities and Attributes

1. **Customer**
    - Customer_ID (PK)
    - Name (Composite: FirstName, LastName)
    - Gender
    - Contact_No(Multicvalued)
    - Email
    - Address
2. **Acoount**
    - Account_no
    - Customer_ID
    - Account_Type(Saving/Current/Fixed Deposite)
    - Balance
    - Opening_Date
    - Account_Age()

3.Reservation

- Transactio_ID
- Account_NO
- Transaction_Type(Deposite/Withdraw/Transfer)
- Ammount
- Trans_Date

MarwadiUniversity

FacultyofComputer Applications

4.Employee

- Employee_ID
- Employee_Name
- Salary
- Email
- Designation

5.Branch

- Branch_ID
- Branch_Name
- Location
- Contact_No

6.Loan

- Loan_ID
- Customer_ID
- Loan_Type
- Amount
- Insert_Rate
- Start_Date
- End_Date

Relationships with Cardinalities

- **Customer – Account**:
  - One account can open many accounts, but each transaction is handled *one account.*
  - 1 : M
  - Cardinality:1 : M

- **Employee – Transaction**:
  - One employee can be handle *many transaction*, but each transaction is handled *one employee.*
  - 1 : M

- **Employee – Transaction**
  - One employee can handle many transactions**.**
  - Each transaction is handled by one employee**.**
  - Cardinality: 1 : M

- **Employee – Branch**
  - One branch can have many employees**.**
  - Each employee works in one branch**.**
  - Cardinality: 1 : M

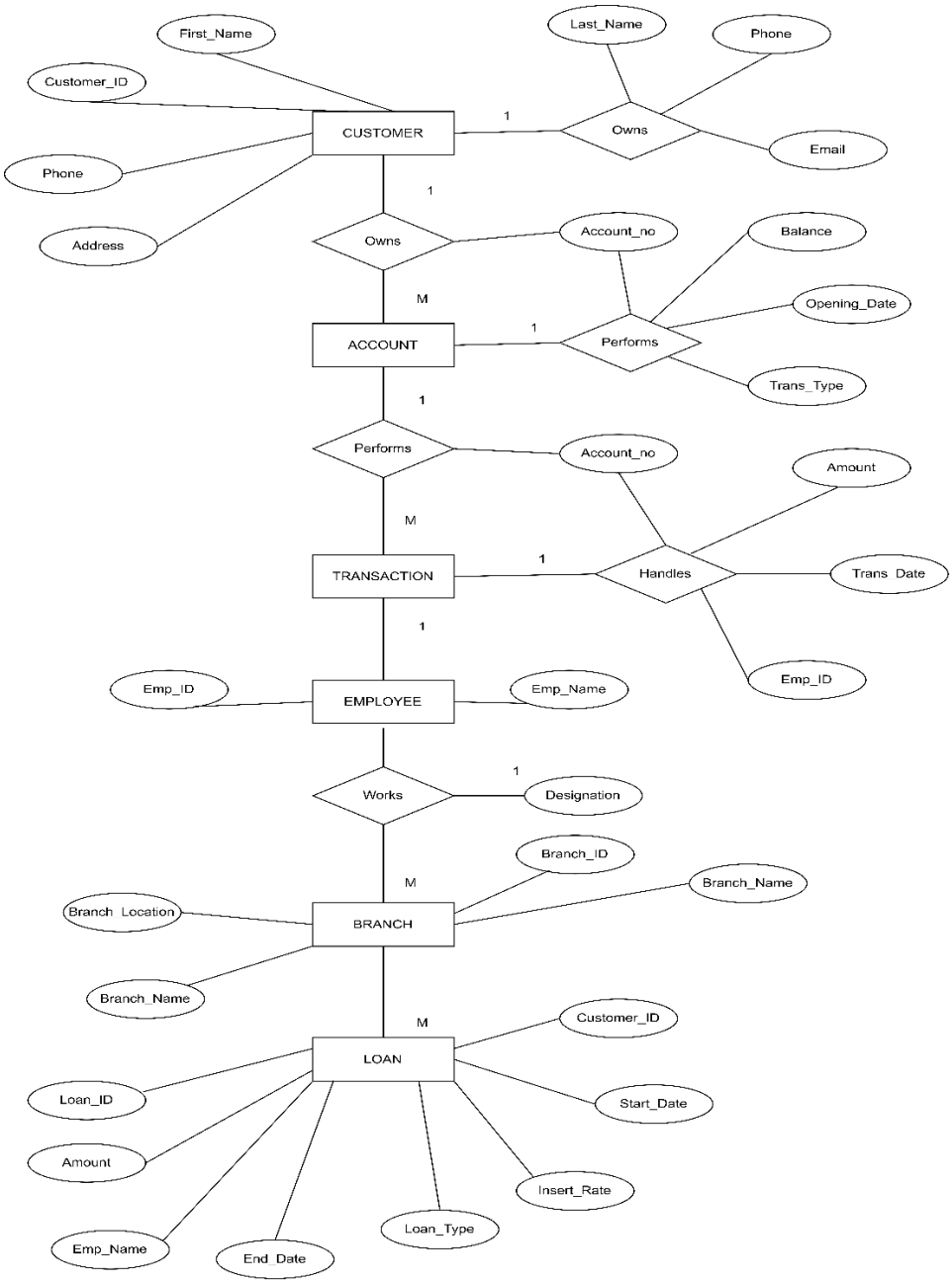
- **Customer – Loan**
  - One customer can have many loans**.**
  - Each loan is linked to one customer**.**
  - Cardinality**:** 1 : M

- **Account – Loan** (optional, if you want to link loan repayment through account)
  - One account can be used for many loan repayments**.**
  - Each loan repayment is done through one account**.**
  - Cardinality: 1 : M

MarwadiUniversity

FacultyofComputer Applications

ER DIAGRAM



MarwadiUniversity

FacultyofComputer Applications

Table Creation Queries

Create tables : Customer

```
CREATE TABLE Customer (
  CustomerIDNUMBER(5)PRIMARY KEY,
  FirstNameVARCHAR2(20)NOT NULL,
  LastNameVARCHAR2(20)NOT NULL,
  Gender CHAR(1)CHECK (Gender IN ('M','F','O')),
  DOBDATE,
  PhoneVARCHAR2(15)UNIQUE,
  EmailVARCHAR2(50)NOT NULL,
  Address VARCHAR2(200)
);
```

Create table : Branch

```
CREATE TABLE Branch (
  Branch_IDNUMBER(5)PRIMARY KEY,
  Branch_NameVARCHAR2(50)NOT NULL,
  LocationVARCHAR2(100)NOT NULL,
  IFSC_CodeVARCHAR2(15) UNIQUE,
  Manager_IDNUMBER(5)NULL);
```

Create tables : Employee

```
CREATE TABLE Employee (
  Emp_IDNUMBER(5)PRIMARY KEY,
  Emp_NameVARCHAR2(50) NOT NULL,
  SalaryNUMBER(12,2)CHECK (Salary > 0),
  EmailVARCHAR2(50),
  DesignationVARCHAR2(20) CHECK (Designation IN
('Manager','Clerk','Cashier','Officer')),
  Branch_IDNUMBER(5)NOT NULL,
  CONSTRAINT fk_emp_branch FOREIGN KEY (Branch_ID) REFERENCES
Branch(Branch_ID)
);
```

MarwadiUniversity

FacultyofComputer Applications

Create tables : Account

```
CREATE TABLE Account (
  Account_NoNUMBER(10)PRIMARY KEY,
  Customer_IDNUMBER(5)NOT NULL,
  Account_Type  VARCHAR2(15) CHECK (Account_Type IN
('Savings','Current','Fixed Deposit')),
  Balance NUMBER(14,2)DEFAULT 0 NOT NULL,
  Opening_Date  DATEDEFAULT SYSDATE,
  Branch_IDNUMBER(5)NOT NULL,
  CONSTRAINT fk_acc_cust FOREIGN KEY (Customer_ID) REFERENCES
Customer(CustomerID),
  CONSTRAINT fk_acc_branch FOREIGN KEY (Branch_ID) REFERENCES
Branch(Branch_ID)
);
```

Create table : Trnsaction

```
CREATE TABLE Transactions (
  Trans_IDNUMBER(10)PRIMARY KEY,
  Account_No NUMBER(10)NOT NULL,
  Trans_Type VARCHAR2(10)CHECK (Trans_Type IN
('Deposit','Withdraw','Transfer')),
  Amount NUMBER(14,2)CHECK (Amount > 0),
  Trans_DateDATEDEFAULT SYSDATE,
  Handled_By_Emp  NUMBER(5),
  RemarksVARCHAR2(200),
  CONSTRAINT fk_tran_acc FOREIGN KEY (Account_No) REFERENCES
Account(Account_No),
  CONSTRAINT fk_tran_emp FOREIGN KEY (Handled_By_Emp) REFERENCES
Employee(Emp_ID)
);
```

MarwadiUniversity

FacultyofComputer Applications

Create table : Loan

```
CREATE TABLE Loan (
        Loan_ID NUMBER(6)PRIMARY KEY,
        Customer_IDNUMBER(5)NOT NULL,
        Emp_IDNUMBER(5)NOT NULL,
        Loan_Type VARCHAR2(30)CHECK (Loan_Type IN
        ('Home','Personal','Education','Vehicle')),
        AmountNUMBER(14,2) CHECK (Amount > 0),
        Interest_RateNUMBER(5,2),
        Duration_Months NUMBER(3),
        Start_DateDATE,
        End_DateDATE,
        CONSTRAINT fk_loan_cust FOREIGN KEY (Customer_ID) REFERENCES
        Customer(CustomerID),
        CONSTRAINT fk_loan_emp FOREIGN KEY (Emp_ID) REFERENCES
        Employee(Emp_ID)
    );
```

Create table : Transaction_audit

```
Transaction_Audit (audit table referenced in your SELECTs)
CREATE TABLE Transaction_Audit (
  Audit_IDNUMBER(10) PRIMARY KEY,
  Trans_ID NUMBER(10),
  Account_NNUMBER(10),
  ActionVARCHAR2(20),
  Old_Balance NUMBER(14,2),
  New_BalanceNUMBER(14,2),
  Action_Date   DATE DEFAULT SYSDATE
);
```

MarwadiUniversity

FacultyofComputer Applications

Sequences(consistent names)

```
CREATE SEQUENCE seq_customer START WITH 1 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE seq_emp START WITH 1 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE seq_account START WITH 100001 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE seq_trans START WITH 1 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE seq_loan START WITH 1001 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE seq_audit START WITH 1 INCREMENT BY 1 NOCACHE;
```

Indexes

```
CREATE INDEX idx_customer_email ON Customer(Email);
CREATE INDEX idx_account_customer ON Account(Customer_ID);
CREATE INDEX idx_trans_acc_date ON Transactions(Account_No, Trans_Date);
```

MarwadiUniversity

FacultyofComputer Applications

Sample Insert Data

Branches

```
INSERT INTO Branch VALUES (1, 'Central Branch', 'Ahmedabad', 'IFSC0001', NULL);
INSERT INTO Branch VALUES (2, 'Satellite Branch', 'Gandhinagar', 'IFSC0002', NULL);
```

Customers

```
INSERT INTO Customer VALUES (1, 'Rahul', 'Sharma', 'M', TO_DATE('15-MAY-1990','DD-MON-YYYY'),
'9876543210', 'rahul.sharma@example.com', 'Ahmedabad');
INSERT INTO Customer VALUES (2, 'Amit', 'Patel', 'M', TO_DATE('10-JUN-1998','DD-MON-YYYY'),
'9998887776', 'amit.patel@example.com', 'Surat');
INSERT INTO Customer VALUES (3, 'Sneha', 'Desai', 'F', TO_DATE('20-DEC-1995','DD-MON-YYYY'),
'9123456780', 'sneha.desai@example.com', 'Vadodara');
```

Employees

```
INSERT INTO Employee VALUES (1, 'Ravi Sharma', 60000, 'ravi.sharma@bank.com', 'Manager', 1);
INSERT INTO Employee VALUES (2, 'Kiran Joshi', 30000, 'kiran.joshi@bank.com', 'Clerk', 1);
INSERT INTO Employee VALUES (3, 'Pooja Mehta', 28000, 'pooja.mehta@bank.com', 'Cashier', 2);
```

Update branch manager pointer:

```
UPDATE Branch SET Manager_ID = 1 WHERE Branch_ID = 1;
```

Accounts (explicitly use 100001, 100002, 100003)

```
INSERT INTO Account VALUES (100001, 1, 'Savings', 50000.00, TO_DATE('01-JAN-2020','DD-MON-
YYYY'), 1);
INSERT INTO Account VALUES (100002, 2, 'Savings', 15000.00, TO_DATE('10-JUN-2023','DD-MON-
YYYY'), 1);
INSERT INTO Account VALUES (100003, 3, 'Current', 250000.00, TO_DATE('05-AUG-2022','DD-MON-
YYYY'), 2);
```

Transactions

```
INSERT INTO Transactions VALUES (1, 100001, 'Deposit', 10000, TO_DATE('15-SEP-2025','DD-MON-
YYYY'), 3, 'Salary deposit');
INSERT INTO Transactions VALUES (2, 100002, 'Withdraw', 2000, TO_DATE('20-SEP-2025','DD-MON-
YYYY'), 2, 'ATM withdraw');
INSERT INTO Transactions VALUES (3, 100003, 'Deposit', 50000, TO_DATE('25-SEP-2025','DD-MON-
YYYY'), 3, 'Business deposit');
```

Loans

```
INSERT INTO Loan VALUES (1001, 1, 1, 'Home', 5000000, 7.50, 240, TO_DATE('01-JAN-2022','DD-MON-
YYYY'), TO_DATE('01-JAN-2042','DD-MON-YYYY'));
INSERT INTO Loan VALUES (1002, 2, 2, 'Education', 500000, 6.00, 60, TO_DATE('01-JUN-2024','DD-
MON-YYYY'), TO_DATE('01-JUN-2029','DD-MON-YYYY'));
```

MarwadiUniversity

FacultyofComputer Applications

SQL QUERIES

1. Show all customers
SQL>SELECT * FROM Customer;

2. Show customers whose ID is greater than 1
SQL>SELECT FirstName, LastName, Email FROM Customer WHERE CustomerID > 1;

3. Find customers whose email ends with "@gmail.com"
SQL>SELECT * FROM Customer WHERE Email LIKE '%@gmail.com';

4. Show accounts having balance between 10,000 and 1,00,000
SQL>SELECT * FROM Account WHERE Balance BETWEEN 10000 AND 100000;

5. Show details of customers having CustomerID 1 or 2
SQL>SELECT * FROM Customer WHERE CustomerID IN (1, 2);

6. Show all accounts in order of highest balance first
SQL>SELECT Account_No, Balance FROM Account ORDER BY Balance DESC;

7. Show all different types of accounts
SQL>SELECT DISTINCT Account_Type FROM Account;

8. Count total number of customers
SQL>SELECT COUNT(*) AS Total_Customers FROM Customer;

9. Count how many accounts exist per account type
SQL>SELECT Account_Type, COUNT(*) AS Total FROM Account GROUP BY Account_Type;

10. Show customer names and their account numbers
SQL>SELECT c.FirstName, c.LastName, a.Account_No, a.BalanceFROM Customer c, Account a
WHERE c.CustomerID = a.Customer_ID;

11. Show all customers and their transactions (if any)
SQL>SELECT c.FirstName, a.Account_No, t.Trans_ID, t.Amount FROM Customer c
LEFT JOIN Account a ON c.CustomerID = a.Customer_ID
LEFT JOIN Transactions t ON a.Account_No = t.Account_No;

MarwadiUniversity

FacultyofComputer Applications

12. Show customers who have account balance above 20,000

```
SQL>SELECT FirstName, LastName FROM Customer WHERE CustomerID IN (
        SELECT Customer_ID FROM Account WHERE Balance > 20000);
```

13. Show accounts whose balance is above average

```
SQL>SELECT Account_No, Balance FROM Account WHERE Balance > (SELECT AVG(Balance) FROM
    Account);
```

14. List names of all customers and employees (no duplicates)

```
SQL>SELECT FirstName AS Name FROM Customer UNION SELECT Emp_Name FROM Employee;
```

15. Show customers who have at least one account

```
SQL>SELECT CustomerID, FirstName FROM Customer WHERE EXISTS (
        SELECT 1 FROM Account WHERE Customer.CustomerID = Account.Customer_ID);
```

16. Show account numbers and balances ranked by balance

```
SQL>SELECT Account_No, Balance,
    RANK() OVER (ORDER BY Balance DESC) AS Rank_No
    FROM Account;
```

17. Show how many months each account has been active

```
SQL>SELECT Account_No,Opening_Date,
   ROUND(MONTHS_BETWEEN(SYSDATE, Opening_Date), 1) AS Months_Active
   FROM Account;
```

18. Convert all customer first names to uppercase

```
SQL>SELECT UPPER(FirstName) AS Upper_Name FROM Customer;
```

19. Try inserting invalid account type(will fail check constraint)

```
SQL>INSERT INTO Account(Account_No, Customer_ID, Account_Type, Balance,
    Branch_ID)VALUES (999999, 1, 'Invalid', 5000, 1);
```

20. Create a view to see customers and their accounts

```
SQL>CREATE OR REPLACE VIEW vw_customer_accounts AS
        SELECT c.CustomerID, c.FirstName, c.LastName, a.Account_No, a.Balance
        FROM Customer c
        JOIN Account a ON c.CustomerID = a.Customer_ID;
```

21. View all records from the created view

```
SQL>SELECT * FROM vw_customer_accounts;
```

MarwadiUniversity

FacultyofComputer Applications

22. Insert a new customer using sequence

```
SQL>INSERT INTO Customer(CustomerID, FirstName, LastName, Gender, DOB, Phone, Email,
      Address)
       VALUES (seq_customer.NEXTVAL,
        'Amit', 'Patel', 'M',TO_DATE('05-MAY-1998','DD-MON-YYYY'),
       '9876543210', 'amit.patel@gmail.com',
       'Ahmedabad'
        );
```

23. Find customer details by email

```
SQL>SELECT * FROM Customer WHERE Email = 'amit.patel@gmail.com';
```

24. Transfer ?1000 from account 100001 to 100002 manually

```
SQL>UPDATE Account SET Balance = Balance - 1000 WHERE Account_No = 100001;
          UPDATE Account SET Balance = Balance + 1000 WHERE Account_No = 100002;
         COMMIT;
```

25. Use package function to calculate EMI

```
SQL>SELECT bank_pkg.calc_emi(100000, 10, 12) AS Monthly_EMI FROM dual;
```

26.Deposit money using package procedure

```
SQL>BEGIN
              bank_pkg.deposit(100001, 2000, 1);
      END;
        /
```

27. Show employees working in the same branch

```
SQL>SELECT e1.Emp_Name AS Employee1, e2.Emp_Name AS Employee2
       FROM Employee e1, Employee e2
       WHERE e1.Branch_ID = e2.Branch_ID AND e1.Emp_ID <> e2.Emp_ID;
```

28. Show total transaction amount for each account

```
SQL>SELECT a.Account_No, SUM(t.Amount) AS Total_Amount
      FROM Account a
      JOIN Transactions t ON a.Account_No = t.Account_No
      GROUP BY a.Account_No;
```

29. Show customers who have taken loans

```
SQL>SELECT DISTINCT c.CustomerID, c.FirstName, c.LastName
       FROM Customer c
       JOIN Loan l ON c.CustomerID = l.Customer_ID;
```

30. Show all records from audit table (trigger)

```
SQL>SELECT * FROM Transaction_Audit ORDER BY Action_Date DESC;
```

MarwadiUniversity

FacultyofComputer Applications

PL/SQL PROGRAMS

1.Deposit procedure

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE deposit_amount(
   p_account_no IN NUMBER,
   p_amount     IN NUMBER)
AS
BEGIN
   UPDATE Account
   SET Balance = Balance + p_amount
   WHERE Account_No = p_account_no;

   DBMS_OUTPUT.PUT_LINE('Amount ' || p_amount || ' deposited to Account ' || p_account_no);
END;
/
```

2. Withdraw procedure

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE withdraw_amount(
   p_account_no IN NUMBER,
   p_amount     IN NUMBER
) AS
   v_balance NUMBER;
BEGIN
   SELECT Balance INTO v_balance FROM Account WHERE Account_No = p_account_no;
   IF v_balance >= p_amount THEN
      UPDATE Account
      SET Balance = Balance - p_amount
      WHERE Account_No = p_account_no;
      DBMS_OUTPUT.PUT_LINE('Withdrawn ' || p_amount || ' from Account ' || p_account_no);
   ELSE
      DBMS_OUTPUT.PUT_LINE('Insufficient balance!');
   END IF;
END;
/
```

3.Get_balance function

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE FUNCTION get_balance(p_account_no IN NUMBER)
RETURN NUMBER IS
   v_balance NUMBER;
BEGIN
   SELECT Balance INTO v_balance FROM Account WHERE Account_No = p_account_no;
   RETURN v_balance;
END;
/
```

4.calc_interest function (simple)

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE FUNCTION calc_interest(p_amount NUMBER)
RETURN NUMBER IS
BEGIN
   RETURN p_amount * 0.05; -- 5% interest
END;
/
```

5. Account number BEFORE INSERT trigger (uses seq_account)

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER trg_account_no
BEFORE INSERT ON Account
FOR EACH ROW
   WHEN (NEW.Account_No IS NULL)
BEGIN
   SELECT seq_account.NEXTVAL INTO :NEW.Account_No FROM dual;
END;
/
```

MarwadiUniversity

FacultyofComputer Applications

6.AFTER UPDATE trigger to log transactions and audit

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER trg_transaction_log
AFTER UPDATE OF Balance ON Account
FOR EACH ROW
DECLARE
   v_type VARCHAR2(10);
   v_amount NUMBER;
   v_trans_id NUMBER;
BEGIN
   v_amount := NVL(:NEW.Balance,0) - NVL(:OLD.Balance,0);
   IF v_amount = 0 THEN
      RETURN; -- nothing changed
   ELSIF v_amount > 0 THEN
      v_type := 'Deposit';
   ELSE
      v_type := 'Withdraw';
      v_amount := ABS(v_amount);
   END IF;

   insert into Transactions
   v_trans_id := seq_trans.NEXTVAL;
   INSERT INTO Transactions(Trans_ID, Account_No, Trans_Type, Amount, Trans_Date)
   VALUES (v_trans_id, :NEW.Account_No, v_type, v_amount, SYSDATE);

   insert into audit table
   INSERT INTO Transaction_Audit(Audit_ID, Trans_ID, Account_No, Action, Old_Balance,
New_Balance, Action_Date)
   VALUES (seq_audit.NEXTVAL, v_trans_id, :NEW.Account_No, v_type, :OLD.Balance,
:NEW.Balance, SYSDATE);
END;
/
```

MarwadiUniversity

FacultyofComputer Applications

7 + 8) bank_pkg spec + body (added calc_emi & account_age)

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PACKAGE bank_pkg AS
   PROCEDURE deposit(p_acc_no NUMBER, p_amount NUMBER);
   PROCEDURE withdraw(p_acc_no NUMBER, p_amount NUMBER);
   FUNCTION calc_emi(p_principal NUMBER, p_rate_per_year NUMBER, p_months NUMBER)
RETURN NUMBER;
   FUNCTION account_age(p_acc_no NUMBER) RETURN NUMBER; -- months active
END bank_pkg;
/
```

8.body package

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PACKAGE BODY bank_pkg AS

   PROCEDURE deposit(p_acc_no NUMBER, p_amount NUMBER) IS
   BEGIN
      UPDATE Account SET Balance = Balance + p_amount WHERE Account_No = p_acc_no;
      DBMS_OUTPUT.PUT_LINE('Deposited ' || p_amount || ' to ' || p_acc_no);
   END deposit;

   PROCEDURE withdraw(p_acc_no NUMBER, p_amount NUMBER) IS
      v_bal NUMBER;
   BEGIN
      SELECT Balance INTO v_bal FROM Account WHERE Account_No = p_acc_no;
      IF v_bal >= p_amount THEN
         UPDATE Account SET Balance = Balance - p_amount WHERE Account_No = p_acc_no;
         DBMS_OUTPUT.PUT_LINE('Withdrawn ' || p_amount || ' from ' || p_acc_no);
      ELSE
         DBMS_OUTPUT.PUT_LINE('Insufficient balance!');
      END IF;
   END withdraw;

   FUNCTION calc_emi(p_principal NUMBER, p_rate_per_year NUMBER, p_months NUMBER)
RETURN NUMBER IS
      v_monthly_rate NUMBER;
```

```
    v_emi NUMBER;
  BEGIN
    IF p_months <= 0 THEN
      RETURN NULL;
    END IF;
    v_monthly_rate := (p_rate_per_year / 100) / 12;
    IF v_monthly_rate = 0 THEN
      v_emi := p_principal / p_months;
    ELSE
      v_emi := (p_principal * v_monthly_rate) / (1 - POWER(1 + v_monthly_rate, -p_months));
    END IF;
    RETURN ROUND(v_emi,2);
  END calc_emi;

  FUNCTION account_age(p_acc_no NUMBER) RETURN NUMBER IS
    v_open DATE;
  BEGIN
    SELECT Opening_Date INTO v_open FROM Account WHERE Account_No = p_acc_no;
    RETURN TRUNC(MONTHS_BETWEEN(SYSDATE, v_open));
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN NULL;
  END account_age;

END bank_pkg;
/
```

9. Example cursor program (unchanged but uses correct account numbers)

```
SET SERVEROUTPUT ON;
DECLARE
   CURSOR c_high IS SELECT Account_No, Balance FROM Account WHERE Balance > 50000;
   v_acc Account.Account_No%TYPE;
   v_bal Account.Balance%TYPE;
BEGIN
   OPEN c_high;
   LOOP
      FETCH c_high INTO v_acc, v_bal;
      EXIT WHEN c_high%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Account: ' || v_acc || ' | Balance: ' || v_bal);
   END LOOP;
   CLOSE c_high;
END;
/
```

10.Exception handling sample (use an account that exists or not)

```
SET SERVEROUTPUT ON;
DECLARE
   v_balance NUMBER;
BEGIN
   SELECT Balance INTO v_balance FROM Account WHERE Account_No = 9999; -- no data
expected
   DBMS_OUTPUT.PUT_LINE('Balance: ' || v_balance);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Account not found!');
END;
/
```

MarwadiUniversity

FacultyofComputer Applications

11.Calculate and display 5% interest for each account

```
SET SERVEROUTPUT ON;
DECLARE
   v_acc_no   Account.Account_No%TYPE;
   v_balance  Account.Balance%TYPE;
   v_interest NUMBER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('Account_No | Balance | Interest(5%)');
   DBMS_OUTPUT.PUT_LINE('---------------------------------');

   FOR rec IN (SELECT Account_No, Balance FROM Account) LOOP
      v_acc_no := rec.Account_No;
      v_balance := rec.Balance;
      v_interest := v_balance * 0.05;

      DBMS_OUTPUT.PUT_LINE(v_acc_no || ' | ' || v_balance || ' | ' || v_interest);
   END LOOP;
END;
/
```

12.Increase balance by 1000 if below 10,000; otherwise by 500

```
SET SERVEROUTPUT ON;
DECLARE
   v_acc_no   Account.Account_No%TYPE;
   v_balance  Account.Balance%TYPE;
BEGIN
   FOR rec IN (SELECT Account_No, Balance FROM Account) LOOP
      v_acc_no := rec.Account_No;
      v_balance := rec.Balance;

      IF v_balance < 10000 THEN
         UPDATE Account SET Balance = Balance + 1000 WHERE Account_No = v_acc_no;
         DBMS_OUTPUT.PUT_LINE('Account ' || v_acc_no || ' updated by +1000');
      ELSE
         UPDATE Account SET Balance = Balance + 500 WHERE Account_No = v_acc_no;
         DBMS_OUTPUT.PUT_LINE('Account ' || v_acc_no || ' updated by +500');
      END IF;
   END LOOP;

   COMMIT;
   DBMS_OUTPUT.PUT_LINE('Balances updated successfully!');
END;
/
```

MarwadiUniversity

FacultyofComputer Applications

Dynamic version: Ask user to enter Account Number

```
SET SERVEROUTPUT ON;
DECLARE
   v_acc_no   Account.Account_No%TYPE := &Enter_Account_No;
   v_acc_type Account.Account_Type%TYPE;
BEGIN
   SELECT Account_Type INTO v_acc_type
   FROM Account
   WHERE Account_No = v_acc_no;

   IF v_acc_type = 'Saving' THEN
      DBMS_OUTPUT.PUT_LINE('Account ' || v_acc_no || ' is a Saving Account.');
   ELSIF v_acc_type = 'Current' THEN
      DBMS_OUTPUT.PUT_LINE('Account ' || v_acc_no || ' is a Current Account.');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Account ' || v_acc_no || ' has an Unknown Type.');
   END IF;

EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE(' Account number ' || v_acc_no || ' not found.');
END;
/
```

Challenges

1. Maintaining data integrity across multiple interrelated tables such as *Customer*, *Account*, *Transactions*, and *Loan*, while enforcing primary key, foreign key, and check constraints.
2. Automating banking operations (deposits, withdrawals, transfers, and audit logging) using PL/SQL procedures, functions, triggers, and packages.
3. Ensuring transaction consistency to prevent overdrafts, double transactions, or partial updates during deposits and withdrawals.
4. Optimizing performance through proper indexing, efficient joins, and query optimization for large customer and transaction datasets.
5. Scaling the system to support new banking services, digital payments, multi-branch operations, and advanced reporting features.

Learning During the Project

1. Gained practical knowledge of relational database design using normalization, primary keys, foreign keys, and integrity constraints.
2. Learned to implement core banking business logic through PL/SQL *procedures*, *functions*, and *triggers*.
3. Understood how to use PL/SQL packages for modular, organized, and reusable code development.
4. Practiced transaction management techniques using COMMIT, ROLLBACK, and SAVEPOINT to ensure data consistency.
5. Improved skills in writing optimized SQL queries involving joins, subqueries, aggregates, and indexes.
6. Learned to handle derived attributes (e.g., total balance per customer) through PL/SQL functions and views.
7. Understood the importance of automation and audit trails, such as updating balances and recording logs via triggers for every transaction.

Conclusion

This extended Bank Management System case study provided valuable hands-on experience in the complete lifecycle of database design and implementation. It demonstrated how real-world banking operations can be modelled using relational tables with appropriate constraints, relationships, and automation through PL/SQL.

By developing procedures, functions, triggers, and packages, key banking operations like deposits, withdrawals, fund transfers, and audit logging were effectively automated. The project emphasized crucial aspects such as data integrity, transaction consistency, system scalability, and performance optimization.

Overall, this coursework enhanced practical skills in SQL and PL/SQL programming, deepened the understanding of transactional database systems, and fostered analytical thinking for building reliable, secure, and scalable financial applications.

MarwadiUniversity

FacultyofComputer Applications

References

Online References

- Oracle, "PL/SQL Packages and Types Reference," *Oracle Database Documentation 21c*, 2023. [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/21/lnpls/plsql-packages.html. [Accessed: Sept. 16, 2025].
- TutorialsPoint, "PL/SQL Tutorial," *TutorialsPoint*, 2023. [Online]. Available: https://www.tutorialspoint.com/plsql/index.htm. [Accessed: Jan. 10, 2025].
- GeeksforGeeks, "PL/SQL Tutorial," *GeeksforGeeks*, 2023. [Online]. Available: https://www.geeksforgeeks.org/pl-sql-tutorial/. [Accessed: Feb. 06, 2025].
- W3Schools, "SQL Tutorial," *W3Schools.com*, 2023. [Online]. Available: https://www.w3schools.com/sql/. [Accessed: Aug. 26, 2025].
- Stack Overflow, "Oracle PL/SQL: Calling Procedures from Package," *StackOverflow.com*, 2023. [Online]. Available: https://stackoverflow.com/. [Accessed: Oct. 20, 2025].

Offline References

- S. Feuerstein, *Oracle PL/SQL Programming*, 6th ed., Sebastopol, CA, USA: O'Reilly Media,2014.
- I. Bayross, *SQL, PL/SQL: The Programming Language of Oracle*, New Delhi, India: BPB Publications, 2010.
- R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed., Boston, MA, USA: Pearson Education, 2016.
- S. K. Singh, *Database Systems: Concepts, Design and Applications*, New Delhi, India: Pearson Education, 2011.
- B. Pribyl and S. Feuerstein, *Learning Oracle PL/SQL*, 2nd ed., Sebastopol, CA, USA: O'Reilly Media, 2002.