-- Create Tables (with minor adjustments for exercises)

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE,

IsVIP VARCHAR2(3) DEFAULT 'NO' -- Added for Exercise 1 Scenario 2

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Additional table for Exercise 5 Scenario 2

CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

TransactionID NUMBER,

LogDate DATE,

Action VARCHAR2(100)

);

-- Create sequence for AuditLog IDs

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

-- Insert Sample Data

INSERT INTO Customers VALUES (1, 'John Doe', DATE '1985-05-15', 5000, SYSDATE, 'NO');

INSERT INTO Customers VALUES (2, 'Jane Smith', DATE '1955-08-22', 25000, SYSDATE, 'NO');

INSERT INTO Accounts VALUES (101, 1, 'Savings', 5000, SYSDATE);

INSERT INTO Accounts VALUES (102, 2, 'Checking', 25000, SYSDATE);

INSERT INTO Transactions VALUES (1001, 101, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (1002, 102, SYSDATE, 500, 'Withdrawal');

INSERT INTO Loans VALUES (9001, 1, 20000, 7.5, SYSDATE, ADD\_MONTHS(SYSDATE, 36));

INSERT INTO Loans VALUES (9002, 2, 50000, 6.0, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Employees VALUES (501, 'Alice Chen', 'Manager', 75000, 'Loans', DATE '2020-01-15');

INSERT INTO Employees VALUES (502, 'Bob Johnson', 'Teller', 42000, 'Branch', DATE '2022-03-01');

COMMIT;

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

-- Exercise 1: Control Structures

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

-- Scenario 1: Senior discount on loans

BEGIN

FOR customer IN (SELECT \* FROM Customers)

LOOP

-- Calculate age

IF MONTHS\_BETWEEN(SYSDATE, customer.DOB)/12 > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = customer.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount for customer: ' || customer.Name);

END IF;

END LOOP;

COMMIT;

END;

/

-- Scenario 2: VIP status

BEGIN

FOR customer IN (SELECT \* FROM Customers)

LOOP

IF customer.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'YES'

WHERE CustomerID = customer.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('VIP status granted to: ' || customer.Name);

END IF;

END LOOP;

COMMIT;

END;

/

-- Scenario 3: Loan reminders

BEGIN

FOR loan IN (

SELECT Loans.\*, Customers.Name

FROM Loans

JOIN Customers ON Loans.CustomerID = Customers.CustomerID

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30

)

LOOP

DBMS\_OUTPUT.PUT\_LINE('REMINDER: ' || loan.Name ||

', your loan #' || loan.LoanID ||

' is due on ' || TO\_CHAR(loan.EndDate, 'Mon DD, YYYY'));

END LOOP;

END;

/

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

-- Exercise 2: Error Handling

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

-- Scenario 1: Safe funds transfer

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

from\_acct NUMBER,

to\_acct NUMBER,

amount NUMBER

) IS

BEGIN

-- Check if accounts exist

IF from\_acct IS NULL OR to\_acct IS NULL THEN

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

END IF;

-- Check sufficient balance

DECLARE

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_acct;

IF from\_balance < amount THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient funds');

END IF;

END;

-- Perform transfer

UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = from\_acct;

UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = to\_acct;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful!');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Account not found');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END;

/

-- Scenario 2: Salary update with error check

CREATE OR REPLACE PROCEDURE UpdateSalary(

emp\_id NUMBER,

percent NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary \* (1 + percent/100)

WHERE EmployeeID = emp\_id;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ' || emp\_id || ' not found');

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated for employee ' || emp\_id);

END IF;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error updating salary: ' || SQLERRM);

END;

/

-- Scenario 3: Add new customer with duplicate check

CREATE OR REPLACE PROCEDURE AddNewCustomer(

c\_id NUMBER,

c\_name VARCHAR2,

c\_dob DATE,

c\_balance NUMBER

) IS

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (c\_id, c\_name, c\_dob, c\_balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID ' || c\_id || ' already exists');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error adding customer: ' || SQLERRM);

END;

/

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

-- Exercise 3: Stored Procedures

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

-- Scenario 1: Monthly interest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to savings accounts');

END;

/

-- Scenario 2: Employee bonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept VARCHAR2,

bonus\_percent NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary \* (1 + bonus\_percent/100)

WHERE Department = dept;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to ' || SQL%ROWCOUNT || ' employees in ' || dept);

END;

/

-- Scenario 3: Funds transfer

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_acct NUMBER,

to\_acct NUMBER,

amount NUMBER

) IS

source\_bal NUMBER;

BEGIN

SELECT Balance INTO source\_bal

FROM Accounts

WHERE AccountID = from\_acct;

IF source\_bal < amount THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient funds');

END IF;

UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = from\_acct;

UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = to\_acct;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer completed successfully');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account not found');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer error: ' || SQLERRM);

END;

/

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

-- Exercise 4: Functions

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

-- Scenario 1: Age calculation

CREATE OR REPLACE FUNCTION CalculateAge(birth\_date DATE)

RETURN NUMBER IS

BEGIN

RETURN TRUNC(MONTHS\_BETWEEN(SYSDATE, birth\_date)/12);

END;

/

-- Scenario 2: Loan installment

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

amount NUMBER,

rate NUMBER,

years NUMBER

) RETURN NUMBER IS

monthly\_rate NUMBER := rate/100/12;

payments NUMBER := years \* 12;

BEGIN

RETURN amount \* monthly\_rate / (1 - POWER(1 + monthly\_rate, -payments));

END;

/

-- Scenario 3: Balance check

CREATE OR REPLACE FUNCTION HasSufficientBalance(

acc\_id NUMBER,

amount NUMBER

) RETURN VARCHAR2 IS

acc\_bal NUMBER;

BEGIN

SELECT Balance INTO acc\_bal

FROM Accounts

WHERE AccountID = acc\_id;

IF acc\_bal >= amount THEN

RETURN 'TRUE';

ELSE

RETURN 'FALSE';

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'ACCOUNT\_NOT\_FOUND';

END;

/

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

-- Exercise 5: Triggers

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

-- Scenario 1: Auto-update last modified

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

-- Scenario 2: Transaction audit log

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (LogID, TransactionID, LogDate, Action)

VALUES (audit\_seq.NEXTVAL, :NEW.TransactionID, SYSDATE,

'New ' || :NEW.TransactionType || ' of $' || :NEW.Amount);

END;

/

-- Scenario 3: Transaction validation

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

acc\_bal NUMBER;

BEGIN

-- Validate deposit amount

IF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20010, 'Deposit amount must be positive');

END IF;

-- Validate withdrawal

IF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO acc\_bal

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF acc\_bal < :NEW.Amount THEN

RAISE\_APPLICATION\_ERROR(-20011, 'Insufficient funds for withdrawal');

END IF;

END IF;

END;

/

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

-- Exercise 6: Cursors

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

-- Scenario 1: Monthly statements

DECLARE

CURSOR stmt\_cursor IS

SELECT c.CustomerID, c.Name, t.TransactionDate, t.Amount, t.TransactionType

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE)

AND EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE);

BEGIN

FOR stmt\_rec IN stmt\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Statement for ' || stmt\_rec.Name || ': ' ||

stmt\_rec.TransactionType || ' of $' || stmt\_rec.Amount ||

' on ' || TO\_CHAR(stmt\_rec.TransactionDate, 'MM/DD'));

END LOOP;

END;

/

-- Scenario 2: Annual fees

DECLARE

CURSOR fee\_cursor IS

SELECT AccountID, Balance

FROM Accounts

FOR UPDATE;

annual\_fee NUMBER := 25;

BEGIN

FOR acc\_rec IN fee\_cursor LOOP

UPDATE Accounts

SET Balance = Balance - annual\_fee

WHERE CURRENT OF fee\_cursor;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Annual fees applied');

END;

/

-- Scenario 3: Update loan rates

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate

FROM Loans

FOR UPDATE;

new\_rate NUMBER;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

-- Simple policy: decrease rates for loans > $30,000

IF loan\_rec.InterestRate > 5.0 THEN

new\_rate := loan\_rec.InterestRate - 0.25;

UPDATE Loans

SET InterestRate = new\_rate

WHERE CURRENT OF loan\_cursor;

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Loan rates updated');

END;

/

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

-- Exercise 7: Packages

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

-- Scenario 1: Customer management

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(

c\_id NUMBER,

c\_name VARCHAR2,

c\_dob DATE,

c\_balance NUMBER

);

PROCEDURE UpdateCustomer(

c\_id NUMBER,

new\_name VARCHAR2,

new\_dob DATE

);

FUNCTION GetBalance(c\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(

c\_id NUMBER,

c\_name VARCHAR2,

c\_dob DATE,

c\_balance NUMBER

) IS

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (c\_id, c\_name, c\_dob, c\_balance, SYSDATE);

COMMIT;

END;

PROCEDURE UpdateCustomer(

c\_id NUMBER,

new\_name VARCHAR2,

new\_dob DATE

) IS

BEGIN

UPDATE Customers

SET Name = new\_name,

DOB = new\_dob

WHERE CustomerID = c\_id;

COMMIT;

END;

FUNCTION GetBalance(c\_id NUMBER) RETURN NUMBER IS

cust\_balance NUMBER;

BEGIN

SELECT Balance INTO cust\_balance

FROM Customers

WHERE CustomerID = c\_id;

RETURN cust\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN -1; -- Error code

END;

END CustomerManagement;

/

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

-- Testing Section

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

-- Testing VIP status update

SELECT \* FROM Customers WHERE IsVIP = 'YES';

-- Testing age calculation

SELECT Name, DOB, CalculateAge(DOB) AS Age FROM Customers;

-- Testing monthly installment

SELECT CalculateMonthlyInstallment(20000, 7.5, 3) AS Monthly\_Payment FROM DUAL;

-- Testing balance check

SELECT AccountID, Balance,

HasSufficientBalance(101, 1000) AS Has\_Enough

FROM Accounts;

-- Testing SafeTransferFunds

CALL SafeTransferFunds(102, 101, 1000);

-- Testing AddNewCustomer

CALL AddNewCustomer(3, 'Mike Wilson', DATE '1990-11-15', 8000);

-- View results

SELECT \* FROM Customers;

SELECT \* FROM Accounts;

SELECT \* FROM AuditLog;