**Exercise 1: Control Structures**

**Scenario 1: Apply 1% interest rate discount for customers above 60**

BEGIN

FOR cust IN (SELECT c.CustomerID, l.LoanID, l.InterestRate, c.DOB

FROM Customers c JOIN Loans l ON c.CustomerID = l.CustomerID) LOOP

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

UPDATE Loans

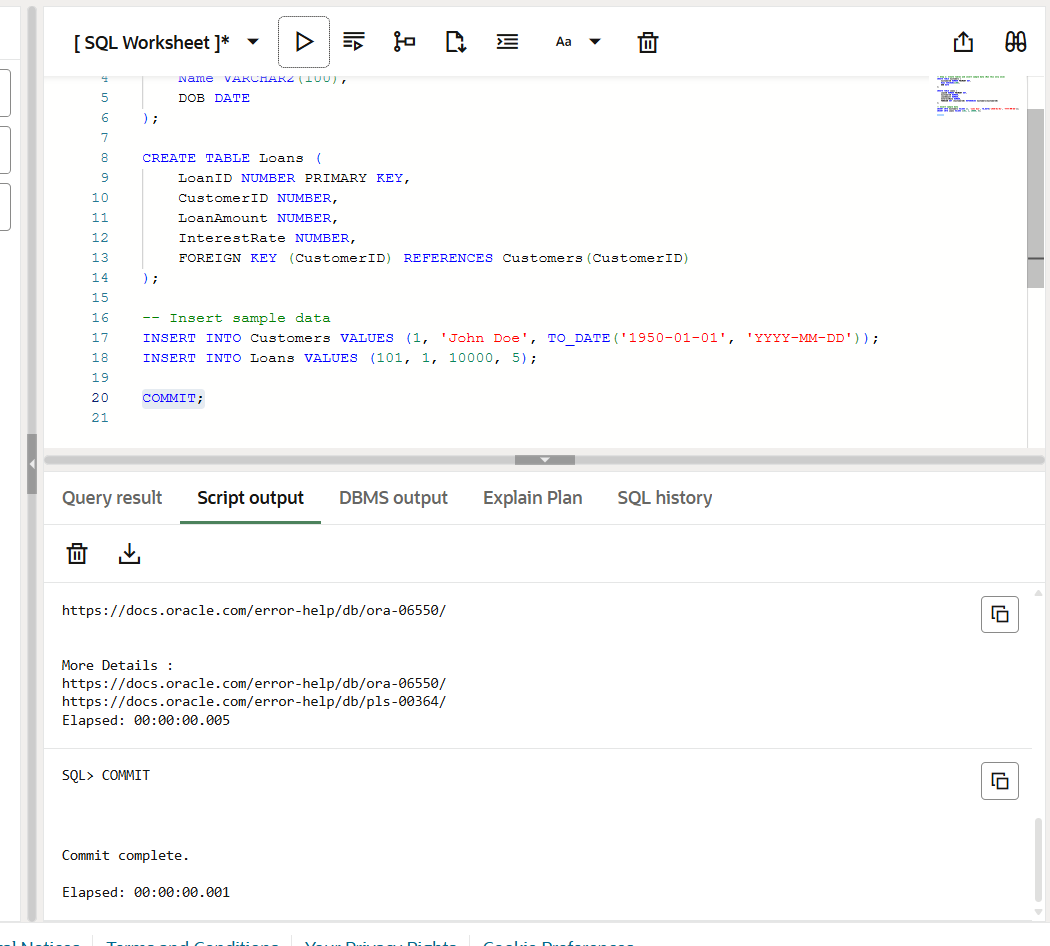
SET InterestRate = InterestRate - 1

WHERE LoanID = cust.LoanID;

END IF;

END LOOP;

END;



**Scenario 2: Set IsVIP = TRUE for customers with balance > 10,000**

**Add Column (if not already exists):**

BEGIN

FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

END;

**Scenario 3: Send reminders for loans due within 30 days**

BEGIN

FOR l IN (SELECT l.LoanID, c.Name, l.EndDate

FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || l.LoanID || ' for ' || l.Name || ' is due on ' || TO\_CHAR(l.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

**Exercise 2: Error Handling**

**Scenario 1: Safe Transfer of Funds**

BEGIN

SELECT Balance INTO v\_from\_balance FROM Accounts WHERE AccountID = p\_from FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error during transfer: ' || SQLERRM);

END;

**Scenario 2: Update Salary with Error Handling**

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_empid NUMBER, p\_percent NUMBER) AS

BEGIN

UPDATE Employees

SET Salary = Salary + Salary \* p\_percent / 100

WHERE EmployeeID = p\_empid;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee not found');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

**Scenario 3: Add New Customer with Duplicate Check**

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) AS

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer with ID ' || p\_id || ' already exists.');

WHEN OTHERS THEN

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

END;

**Exercise 3: Stored Procedures**

**Scenario 1: Process Monthly Interest (1%)**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + Balance \* 0.01

WHERE AccountType = 'Savings';

COMMIT;

END;

**Scenario 2: Update Bonus Based on Department**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_dept VARCHAR2, p\_bonus\_percent NUMBER) AS

BEGIN

UPDATE Employees

SET Salary = Salary + Salary \* p\_bonus\_percent / 100

WHERE Department = p\_dept;

COMMIT;

END;

**Scenario 3: Transfer Funds Between Customer Accounts**

CREATE OR REPLACE PROCEDURE TransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) AS

v\_balance Accounts.Balance%TYPE;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient balance in source account');

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

**Exercise 4: Functions**

**Scenario 1: Calculate Customer Age**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS

BEGIN

RETURN TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob)/12);

END;

**Scenario 2: Calculate Monthly Loan Installment**

sql

CopyEdit

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_amount NUMBER, p\_rate NUMBER, p\_years NUMBER) RETURN NUMBER IS

v\_monthly\_rate NUMBER := p\_rate / (12 \* 100);

v\_months NUMBER := p\_years \* 12;

BEGIN

RETURN (p\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -v\_months));

END;

**Scenario 3: Check Sufficient Balance**

sql

CopyEdit

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_accid NUMBER, p\_amount NUMBER) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_accid;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

**Exercise 5: Triggers**

**Scenario 1: Update LastModified**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

**Scenario 2: Log Transactions**

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY,

AccountID NUMBER,

Amount NUMBER,

TransactionType VARCHAR2(10),

LogDate DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (AccountID, Amount, TransactionType, LogDate)

VALUES (:NEW.AccountID, :NEW.Amount, :NEW.TransactionType, SYSDATE);

END;

**Scenario 3: Enforce Deposit/Withdrawal Rules**

sql

CopyEdit

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Withdrawal exceeds account balance');

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

RAISE\_APPLICATION\_ERROR(-20005, 'Deposit amount must be positive');

END IF;

END;

**Exercise 6: Cursors**

**Scenario 1: Generate Monthly Statements**

sql

CopyEdit

DECLARE

CURSOR cur\_statements IS

SELECT a.CustomerID, t.AccountID, t.TransactionDate, t.Amount, t.TransactionType

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

WHERE TO\_CHAR(t.TransactionDate, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY');

BEGIN

FOR rec IN cur\_statements LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CustomerID || ', Account ' || rec.AccountID ||

': ' || rec.TransactionType || ' of ' || rec.Amount || ' on ' || rec.TransactionDate);

END LOOP;

END;

**Scenario 2: Apply Annual Fee**

sql

CopyEdit

DECLARE

CURSOR cur\_accounts IS SELECT AccountID, Balance FROM Accounts;

BEGIN

FOR acc IN cur\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - 100 -- annual fee

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

**Scenario 3: Update Loan Interest Rates**

sql

CopyEdit

DECLARE

CURSOR cur\_loans IS SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR l IN cur\_loans LOOP

UPDATE Loans

SET InterestRate = InterestRate + 0.5 -- e.g., policy increase

WHERE LoanID = l.LoanID;

END LOOP;

COMMIT;

END;

**Exercise 7: Packages**

**Scenario 1: CustomerManagement Package**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2);

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

INSERT INTO Customers VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);

END;

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2) IS

BEGIN

UPDATE Customers SET Name = p\_name WHERE CustomerID = p\_id;

END;

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_id;

RETURN v\_balance;

END;

END;

**Scenario 2: EmployeeManagement Package**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2, p\_hire DATE);

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_name VARCHAR2);

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2, p\_hire DATE) IS

BEGIN

INSERT INTO Employees VALUES (p\_id, p\_name, p\_pos, p\_sal, p\_dept, p\_hire);

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_name VARCHAR2) IS

BEGIN

UPDATE Employees SET Name = p\_name WHERE EmployeeID = p\_id;

END;

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_id;

RETURN v\_salary \* 12;

END;

END;

**Scenario 3: AccountOperations Package**

sql

CopyEdit

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_accid NUMBER);

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (p\_accid, p\_custid, p\_type, p\_balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_accid NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_accid;

END;

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER IS

v\_total NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_custid;

RETURN v\_total;

END;

END;