Exercise 1: Control Structures

DECLARE

v\_age NUMBER;

CURSOR c\_customers IS

SELECT CustomerID, EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOB) AS age

FROM Customers;

BEGIN

FOR customer IN c\_customers LOOP

v\_age := customer.age;

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = customer.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

BEGIN

FOR customer IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = customer.CustomerID;

END LOOP;

COMMIT;

END;

/

DECLARE

CURSOR c\_loans IS

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR loan IN c\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan.LoanID || ' for Customer ' || loan.Name || ' is due soon.');

END LOOP;

END;

/

Exercise 2: Error Handling

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

e\_insufficient\_funds EXCEPTION;

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

RAISE e\_insufficient\_funds;

END IF;

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

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

COMMIT;

EXCEPTION

WHEN e\_insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in source account.');

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

e\_employee\_not\_found EXCEPTION;

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_percentage / 100)

WHERE EmployeeID = p\_employee\_id;

IF SQL%NOTFOUND THEN

RAISE e\_employee\_not\_found;

END IF;

COMMIT;

EXCEPTION

WHEN e\_employee\_not\_found THEN

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

WHEN OTHERS THEN

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

END;

/

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) IS

e\_customer\_exists EXCEPTION;

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

RAISE e\_customer\_exists;

WHEN e\_customer\_exists THEN

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

WHEN OTHERS THEN

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

END;

/

Exercise 3: Stored Procedures

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

END;

/

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percentage / 100)

WHERE Department = p\_department;

COMMIT;

END;

/

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

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

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

COMMIT;

END;

/

Exercise 4: Functions

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

SELECT EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM p\_dob)

INTO v\_age

FROM DUAL;

RETURN v\_age;

END;

/

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount IN NUMBER,

p\_interest\_rate IN NUMBER,

p\_years IN NUMBER

) RETURN NUMBER IS

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_installment := p\_loan\_amount \* p\_interest\_rate / 1200 /

(1 - POWER(1 + p\_interest\_rate / 1200, -p\_years \* 12));

RETURN v\_monthly\_installment;

END;

/

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN NUMBER,

p\_amount IN NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

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

RETURN v\_balance >= p\_amount;

END;

/

Exercise 5: Triggers

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType);

END;

/

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

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

IF v\_balance < :NEW.Amount THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Deposit amount must be positive.');

END IF;

END IF;

END;

/

Exercise 6: Cursors

DECLARE

CURSOR c\_transactions IS

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

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE t.TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

BEGIN

FOR txn IN c\_transactions LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ' || txn.Name || ' had a ' || txn.TransactionType ||

' of ' || txn.Amount || ' on ' || txn.TransactionDate);

END LOOP;

END;

/

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance FROM Accounts;

v\_annual\_fee CONSTANT NUMBER := 50;

BEGIN

FOR account IN c\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee

WHERE AccountID = account.AccountID;

END LOOP;

COMMIT;

END;

/

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate FROM Loans;

v\_new\_rate CONSTANT NUMBER := 7; -- New interest rate

BEGIN

FOR loan IN c\_loans LOOP

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = loan.LoanID;

END LOOP;

COMMIT;

END;

/

Exercise 7: Packages

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_dob IN DATE, p\_balance IN NUMBER);

FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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

BEGIN

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

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

END AddCustomer;

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_dob IN DATE, p\_balance IN NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_name, DOB = p\_dob, Balance = p\_balance, LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

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

RETURN v\_balance;

END GetCustomerBalance;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2, p\_hire\_date IN DATE);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2);

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2, p\_hire\_date IN DATE) IS

BEGIN

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, p\_hire\_date);

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2) IS

BEGIN

UPDATE Employees

SET Name = p\_name, Position = p\_position, Salary = p\_salary, Department = p\_department

WHERE EmployeeID = p\_employee\_id;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER) RETURN NUMBER IS

v\_annual\_salary NUMBER;

BEGIN

SELECT Salary \* 12 INTO v\_annual\_salary FROM Employees WHERE EmployeeID = p\_employee\_id;

RETURN v\_annual\_salary;

END CalculateAnnualSalary;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_account\_type IN VARCHAR2, p\_balance IN NUMBER);

PROCEDURE CloseAccount(p\_account\_id IN NUMBER);

FUNCTION GetTotalBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_account\_type IN VARCHAR2, p\_balance IN NUMBER) IS

BEGIN

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (p\_account\_id, p\_customer\_id, p\_account\_type, p\_balance, SYSDATE);

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id IN NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_account\_id;

END CloseAccount;

FUNCTION GetTotalBalance(p\_customer\_id IN NUMBER) RETURN NUMBER IS

v\_total\_balance NUMBER;

BEGIN

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

RETURN v\_total\_balance;

END GetTotalBalance;

END AccountOperations;

/