**Exercise 1: Control Structures**

**Scenario 1**

BEGIN

FOR customer IN (SELECT customer\_id, age, loan\_interest\_rate FROM Customers) LOOP

IF customer.age > 60 THEN

UPDATE Loans

SET interest\_rate = loan\_interest\_rate \* 0.99

WHERE customer\_id = customer.customer\_id;

END IF;

END LOOP;

END;

**Scenario 2**

BEGIN

FOR customer IN (SELECT customer\_id, balance FROM Customers) LOOP

IF customer.balance > 10000 THEN

UPDATE Customers

SET IsVIP = TRUE

WHERE customer\_id = customer.customer\_id;

END IF;

END LOOP;

END;

**Scenario 3**

BEGIN

FOR loan IN (SELECT loan\_id, customer\_id, due\_date FROM Loans WHERE due\_date BETWEEN SYSDATE AND SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || loan.loan\_id || ' for customer ' || loan.customer\_id || ' is due on ' || TO\_CHAR(loan.due\_date, 'DD-MON-YYYY'));

END LOOP;

END;

**Exercise 2: Error Handling**

**Scenario 1**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from\_account IN NUMBER, p\_to\_account IN NUMBER, p\_amount IN NUMBER) IS

INSUFFICIENT\_FUNDS EXCEPTION;

BEGIN

BEGIN

UPDATE Accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

IF SQL%ROWCOUNT = 0 THEN

RAISE INSUFFICIENT\_FUNDS;

END IF;

UPDATE Accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

COMMIT;

EXCEPTION

WHEN INSUFFICIENT\_FUNDS THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

**Scenario 2**

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_employee\_id IN NUMBER, p\_percentage IN NUMBER) IS

employee\_not\_found EXCEPTION;

BEGIN

BEGIN

UPDATE Employees

SET salary = salary \* (1 + p\_percentage / 100)

WHERE employee\_id = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE employee\_not\_found;

END IF;

COMMIT;

EXCEPTION

WHEN 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;

END;

**Scenario 3**

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_dob IN DATE) IS

customer\_exists EXCEPTION;

BEGIN

BEGIN

INSERT INTO Customers (customer\_id, name, dob)

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

RAISE customer\_exists;

WHEN OTHERS THEN

ROLLBACK;

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

END;

EXCEPTION

WHEN customer\_exists THEN

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

END;

**Exercise 3: Stored Procedures**

**Scenario 1**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR account IN (SELECT account\_id, balance FROM SavingsAccounts) LOOP

UPDATE SavingsAccounts

SET balance = balance \* 1.01

WHERE account\_id = account.account\_id;

END LOOP;

END;

**Scenario 2**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_department\_id IN NUMBER, p\_bonus\_percentage IN NUMBER) IS

BEGIN

UPDATE Employees

SET salary = salary \* (1 + p\_bonus\_percentage / 100)

WHERE department\_id = p\_department\_id;

COMMIT;

END;

**Scenario 3**

CREATE OR REPLACE PROCEDURE TransferFunds(p\_from\_account IN NUMBER, p\_to\_account IN NUMBER, p\_amount IN NUMBER) IS

insufficient\_balance EXCEPTION;

BEGIN

BEGIN

UPDATE Accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account

AND balance >= p\_amount;

IF SQL%ROWCOUNT = 0 THEN

RAISE insufficient\_balance;

END IF;

UPDATE Accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

COMMIT;

EXCEPTION

WHEN insufficient\_balance THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

**Exercise 4: Functions**

**Scenario 1**

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

v\_age NUMBER;

BEGIN

SELECT TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12) INTO v\_age FROM DUAL;

RETURN v\_age;

END;

**Scenario 2**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_loan\_amount IN NUMBER, p\_interest\_rate IN NUMBER, p\_loan\_duration 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\_loan\_duration \* 12)));

RETURN v\_monthly\_installment;

END;

**Scenario 3**

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 account\_id = p\_account\_id;

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

**Exercise 5: Triggers**

**Scenario 1**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

**Scenario 2**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (transaction\_id, account\_id, amount, transaction\_date)

VALUES (:NEW.transaction\_id, :NEW.account\_id, :NEW.amount, :NEW.transaction\_date);

END;

**Scenario 3**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

IF :NEW.amount < 0 THEN

SELECT balance INTO v\_balance FROM Accounts WHERE account\_id = :NEW.account\_id;

IF v\_balance + :NEW.amount < 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance for withdrawal.');

END IF;

END IF;

IF :NEW.amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive.');

END IF;

END;