**Exercise 1: Control Structures**

**Scenario 1:**

DECLARE

CURSOR customer\_cursor IS

SELECT customer\_id, age, loan\_interest\_rate

FROM customers

WHERE age > 60;

v\_customer\_id customers.customer\_id%TYPE;

v\_age customers.age%TYPE;

v\_loan\_interest\_rate customers.loan\_interest\_rate%TYPE;

BEGIN

OPEN customer\_cursor;

LOOP

FETCH customer\_cursor INTO v\_customer\_id, v\_age, v\_loan\_interest\_rate;

EXIT WHEN customer\_cursor%NOTFOUND;

UPDATE customers

SET loan\_interest\_rate = loan\_interest\_rate - (loan\_interest\_rate \* 0.01)

WHERE customer\_id = v\_customer\_id;

END LOOP;

CLOSE customer\_cursor;

COMMIT;

END;

**Scenario 2:**

DECLARE

CURSOR customer\_cursor IS

SELECT customer\_id, balance

FROM customers

WHERE balance > 10000;

v\_customer\_id customers.customer\_id%TYPE;

v\_balance customers.balance%TYPE;

BEGIN

OPEN customer\_cursor;

LOOP

FETCH customer\_cursor INTO v\_customer\_id, v\_balance;

EXIT WHEN customer\_cursor%NOTFOUND;

UPDATE customers

SET IsVIP = TRUE

WHERE customer\_id = v\_customer\_id;

END LOOP;

CLOSE customer\_cursor;

COMMIT;

END;

**Scenario 3:**

DECLARE

CURSOR loan\_cursor IS

SELECT customer\_id, loan\_due\_date

FROM loans

WHERE loan\_due\_date BETWEEN SYSDATE AND SYSDATE + 30;

v\_customer\_id loans.customer\_id%TYPE;

v\_loan\_due\_date loans.loan\_due\_date%TYPE;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_customer\_id, v\_loan\_due\_date;

EXIT WHEN loan\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ID ' || v\_customer\_id ||

', your loan is due on ' || TO\_CHAR(v\_loan\_due\_date, 'DD-MON-YYYY') || '.');

END LOOP;

CLOSE loan\_cursor;

END;

**Exercise 2 : Error Handling**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

e\_insufficient\_funds EXCEPTION;

v\_from\_balance NUMBER;

v\_to\_balance NUMBER;

BEGIN

-- Check the balance of the from\_account

SELECT balance INTO v\_from\_balance

FROM accounts

WHERE account\_id = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

RAISE e\_insufficient\_funds;

END IF;

-- Deduct the amount from the from\_account

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account\_id;

-- Add the amount to the to\_account

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN e\_insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account\_id || '.');

WHEN OTHERS THEN

ROLLBACK;

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

END SafeTransferFunds;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

e\_employee\_not\_found EXCEPTION;

v\_current\_salary NUMBER;

BEGIN

SELECT salary INTO v\_current\_salary

FROM employees

WHERE employee\_id = p\_employee\_id;

UPDATE employees

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

WHERE employee\_id = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE e\_employee\_not\_found;

WHEN e\_employee\_not\_found THEN

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

WHEN OTHERS THEN

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

END UpdateSalary;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_age IN NUMBER,

p\_balance IN NUMBER

) IS

e\_duplicate\_customer EXCEPTION;

BEGIN

-- Insert the new customer

INSERT INTO customers (customer\_id, name, age, balance)

VALUES (p\_customer\_id, p\_name, p\_age, p\_balance);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

RAISE e\_duplicate\_customer;

WHEN e\_duplicate\_customer THEN

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

WHEN OTHERS THEN

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

END AddNewCustomer;

/

**Exercise 3 : Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_type = 'SAVINGS';

COMMIT;

END ProcessMonthlyInterest;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department\_id IN NUMBER,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE employees

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

WHERE department\_id = p\_department\_id;

COMMIT;

END UpdateEmployeeBonus;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

v\_from\_balance NUMBER;

BEGIN

SELECT balance INTO v\_from\_balance

FROM accounts

WHERE account\_id = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

END IF;

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account\_id;

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END TransferFunds;

/

**Exercise 4: Functions**

**Scenario 1:**

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob DATE

) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END CalculateAge;

/

**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_duration\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_num\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_rate := p\_interest\_rate / 12 / 100;

v\_num\_payments := p\_duration\_years \* 12;

v\_monthly\_installment := p\_loan\_amount \* v\_monthly\_rate / (1 - POWER(1 + v\_monthly\_rate, -v\_num\_payments));

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END HasSufficientBalance;

/

**Exercise 5: Triggers**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

**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, description)

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

END LogTransaction;

/

**Scenario 3:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

IF :NEW.amount < 0 THEN

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = :NEW.account\_id

FOR UPDATE;

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

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

END IF;

ELSIF :NEW.amount = 0 THEN

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

END IF;

END CheckTransactionRules;

/

**Exercise 6: Cursors**

**Scenario 1:**

DECLARE

CURSOR transaction\_cursor IS

SELECT customer\_id, transaction\_date, amount, description

FROM transactions

WHERE transaction\_date BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

v\_customer\_id transactions.customer\_id%TYPE;

v\_transaction\_date transactions.transaction\_date%TYPE;

v\_amount transactions.amount%TYPE;

v\_description transactions.description%TYPE;

BEGIN

OPEN transaction\_cursor;

LOOP

FETCH transaction\_cursor INTO v\_customer\_id, v\_transaction\_date, v\_amount, v\_description;

EXIT WHEN transaction\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id);

DBMS\_OUTPUT.PUT\_LINE('Date: ' || TO\_CHAR(v\_transaction\_date, 'DD-MON-YYYY'));

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || v\_amount);

DBMS\_OUTPUT.PUT\_LINE('Description: ' || v\_description);

DBMS\_OUTPUT.PUT\_LINE('-------------------------');

END LOOP;

CLOSE transaction\_cursor;

END;

/

**Scenario 2:**

DECLARE

CURSOR account\_cursor IS

SELECT account\_id, balance

FROM accounts;

v\_account\_id accounts.account\_id%TYPE;

v\_balance accounts.balance%TYPE;

v\_annual\_fee CONSTANT NUMBER := 50;

BEGIN

OPEN account\_cursor;

LOOP

FETCH account\_cursor INTO v\_account\_id, v\_balance;

EXIT WHEN account\_cursor%NOTFOUND;

UPDATE accounts

SET balance = balance - v\_annual\_fee

WHERE account\_id = v\_account\_id;

END LOOP;

CLOSE account\_cursor;

COMMIT;

END;

/

**Scenario 3:**

DECLARE

CURSOR loan\_cursor IS

SELECT loan\_id, interest\_rate

FROM loans;

v\_loan\_id loans.loan\_id%TYPE;

v\_interest\_rate loans.interest\_rate%TYPE;

v\_new\_interest\_rate NUMBER;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_loan\_id, v\_interest\_rate;

EXIT WHEN loan\_cursor%NOTFOUND;

v\_new\_interest\_rate := v\_interest\_rate + 0.5;

UPDATE loans

SET interest\_rate = v\_new\_interest\_rate

WHERE loan\_id = v\_loan\_id;

END LOOP;

CLOSE loan\_cursor;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_age NUMBER, p\_balance NUMBER);

PROCEDURE UpdateCustomerDetails(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_age NUMBER);

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

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_age NUMBER, p\_balance NUMBER) IS

BEGIN

INSERT INTO customers (customer\_id, name, age, balance)

VALUES (p\_customer\_id, p\_name, p\_age, p\_balance);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_age NUMBER) IS

BEGIN

UPDATE customers

SET name = p\_name, age = p\_age

WHERE customer\_id = p\_customer\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID ' || p\_customer\_id || ' does not exist.');

WHEN OTHERS THEN

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

END UpdateCustomerDetails;

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

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance

FROM customers

WHERE customer\_id = p\_customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id NUMBER, p\_name VARCHAR2, p\_department\_id NUMBER, p\_salary NUMBER);

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

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

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id NUMBER, p\_name VARCHAR2, p\_department\_id NUMBER, p\_salary NUMBER) IS

BEGIN

INSERT INTO employees (employee\_id, name, department\_id, salary)

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END HireEmployee;

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

BEGIN

UPDATE employees

SET name = p\_name, department\_id = p\_department\_id, salary = p\_salary

WHERE employee\_id = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

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

END UpdateEmployeeDetails;

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

v\_salary NUMBER;

BEGIN

SELECT salary INTO v\_salary

FROM employees

WHERE employee\_id = p\_employee\_id;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations AS

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

PROCEDURE CloseAccount(p\_account\_id NUMBER);

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

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

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

BEGIN

INSERT INTO accounts (account\_id, customer\_id, balance, account\_type)

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id NUMBER) IS

BEGIN

DELETE FROM accounts

WHERE account\_id = p\_account\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account ID ' || p\_account\_id || ' does not exist.');

WHEN OTHERS THEN

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

END CloseAccount;

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

v\_total\_balance NUMBER;

BEGIN

SELECT SUM(balance) INTO v\_total\_balance

FROM accounts

WHERE customer\_id = p\_customer\_id;

RETURN v\_total\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

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

RETURN 0;

END GetTotalBalance;

END AccountOperations;

/