**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

CREATE TABLE CUSTOMERS (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

age NUMBER

);

CREATE TABLE LOANS (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

loan\_amount NUMBER,

interest\_rate NUMBER,

CONSTRAINT fk\_customer

FOREIGN KEY (customer\_id)

REFERENCES CUSTOMERS(customer\_id)

);

INSERT INTO CUSTOMERS (customer\_id, name, age) VALUES (1, ' John Doe', 65);

INSERT INTO CUSTOMERS (customer\_id, name, age) VALUES (2, 'Jane Smith', 58);

INSERT INTO CUSTOMERS (customer\_id, name, age) VALUES (3, 'Alice Johnson', 70);

INSERT INTO CUSTOMERS (customer\_id, name, age) VALUES (4, 'Robert Brown', 45);

INSERT INTO LOANS (loan\_id, customer\_id, loan\_amount, interest\_rate) VALUES (1, 1, 10000, 5.5);

INSERT INTO LOANS (loan\_id, customer\_id, loan\_amount, interest\_rate) VALUES (2, 2, 15000, 4.5);

INSERT INTO LOANS (loan\_id, customer\_id, loan\_amount, interest\_rate) VALUES (3, 3, 20000, 6.0);

INSERT INTO LOANS (loan\_id, customer\_id, loan\_amount, interest\_rate) VALUES (4, 4, 25000, 5.0);

DECLARE

CURSOR cust\_cursor IS

SELECT customer\_id, age

FROM CUSTOMERS

WHERE age > 60;

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

v\_age CUSTOMERS.age%TYPE;

BEGIN

FOR cust\_record IN cust\_cursor LOOP

v\_customer\_id := cust\_record.customer\_id;

v\_age := cust\_record.age;

UPDATE LOANS

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = v\_customer\_id

AND interest\_rate > 1;

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END;

/

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

DECLARE

CURSOR cust\_cursor IS

SELECT customer\_id

FROM CUSTOMERS

WHERE balance > 10000;

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

BEGIN

FOR cust\_record IN cust\_cursor LOOP

v\_customer\_id := cust\_record.customer\_id;

UPDATE CUSTOMERS

SET IsVIP = 'TRUE'

WHERE customer\_id = v\_customer\_id;

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END;

/

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* + **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

DECLARE

CURSOR loan\_cursor IS

SELECT customer\_id, loan\_id, due\_date

FROM LOANS

WHERE due\_date BETWEEN SYSDATE AND SYSDATE + 30;

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

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

v\_due\_date LOANS.due\_date%TYPE;

BEGIN

FOR loan\_record IN loan\_cursor LOOP

v\_customer\_id := loan\_record.customer\_id;

v\_loan\_id := loan\_record.loan\_id;

v\_due\_date := loan\_record.due\_date;

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

' has a loan (Loan ID: ' || v\_loan\_id ||

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

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

balance NUMBER

);

CREATE TABLE TransferLogs (

log\_id NUMBER PRIMARY KEY,

from\_account\_id NUMBER,

to\_account\_id NUMBER,

amount NUMBER,

transfer\_date DATE,

status VARCHAR2(20),

message VARCHAR2(255)

);

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

pragma EXCEPTION\_INIT(insufficient\_funds, -20001);

BEGIN

DECLARE

v\_from\_balance NUMBER;

BEGIN

SELECT balance INTO v\_from\_balance

FROM Accounts

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

IF v\_from\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

END;

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;

INSERT INTO TransferLogs (log\_id, from\_account\_id, to\_account\_id, amount, transfer\_date, status, message)

VALUES (TRANSFER\_LOG\_SEQ.NEXTVAL, p\_from\_account\_id, p\_to\_account\_id, p\_amount, SYSDATE, 'SUCCESS', 'Transfer completed successfully');

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

INSERT INTO TransferLogs (log\_id, from\_account\_id, to\_account\_id, amount, transfer\_date, status, message)

VALUES (TRANSFER\_LOG\_SEQ.NEXTVAL, p\_from\_account\_id, p\_to\_account\_id, p\_amount, SYSDATE, 'FAILED', 'Insufficient funds');

ROLLBACK;

WHEN OTHERS THEN

INSERT INTO TransferLogs (log\_id, from\_account\_id, to\_account\_id, amount, transfer\_date, status, message)

VALUES (TRANSFER\_LOG\_SEQ.NEXTVAL, p\_from\_account\_id, p\_to\_account\_id, p\_amount, SYSDATE, 'FAILED', SQLERRM);

ROLLBACK;

END SafeTransferFunds;

/

**Scenario 2:** Manage errors when updating employee salaries.

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

CREATE TABLE Employees (

employee\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

salary NUMBER

);

CREATE TABLE SalaryLogs (

log\_id NUMBER PRIMARY KEY,

employee\_id NUMBER,

change\_date DATE,

status VARCHAR2(20),

message VARCHAR2(255)

);

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

employee\_not\_found EXCEPTION;

pragma EXCEPTION\_INIT(employee\_not\_found, -20002);

BEGIN

UPDATE Employees

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

WHERE employee\_id = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE employee\_not\_found;

END IF;

INSERT INTO SalaryLogs (log\_id, employee\_id, change\_date, status, message)

VALUES (SALARY\_LOG\_SEQ.NEXTVAL, p\_employee\_id, SYSDATE, 'SUCCESS', 'Salary updated successfully');

COMMIT;

EXCEPTION

WHEN employee\_not\_found THEN

INSERT INTO SalaryLogs (log\_id, employee\_id, change\_date, status, message)

VALUES (SALARY\_LOG\_SEQ.NEXTVAL, p\_employee\_id, SYSDATE, 'FAILED', 'Employee ID not found');

ROLLBACK;

WHEN OTHERS THEN

INSERT INTO SalaryLogs (log\_id, employee\_id, change\_date, status, message)

VALUES (SALARY\_LOG\_SEQ.NEXTVAL, p\_employee\_id, SYSDATE, 'FAILED', SQLERRM);

ROLLBACK;

END UpdateSalary;

/

**Scenario 3:** Ensure data integrity when adding a new customer.

* + **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

CREATE TABLE Customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

age NUMBER,

balance NUMBER,

IsVIP VARCHAR2(5)

);

CREATE TABLE CustomerLogs (

log\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

log\_date DATE,

status VARCHAR2(20),

message VARCHAR2(255)

);

CREATE SEQUENCE TRANSFER\_LOG\_SEQ START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE SALARY\_LOG\_SEQ START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE CUSTOMER\_LOG\_SEQ START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_age IN NUMBER,

p\_balance IN NUMBER

) IS

duplicate\_customer EXCEPTION;

pragma EXCEPTION\_INIT(duplicate\_customer, -20003);

BEGIN

INSERT INTO Customers (customer\_id, name, age, balance, IsVIP)

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

INSERT INTO CustomerLogs (log\_id, customer\_id, log\_date, status, message)

VALUES (CUSTOMER\_LOG\_SEQ.NEXTVAL, p\_customer\_id, SYSDATE, 'SUCCESS', 'Customer added successfully');

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

INSERT INTO CustomerLogs (log\_id, customer\_id, log\_date, status, message)

VALUES (CUSTOMER\_LOG\_SEQ.NEXTVAL, p\_customer\_id, SYSDATE, 'FAILED', 'Duplicate customer ID');

ROLLBACK;

WHEN OTHERS THEN

INSERT INTO CustomerLogs (log\_id, customer\_id, log\_date, status, message)

VALUES (CUSTOMER\_LOG\_SEQ.NEXTVAL, p\_customer\_id, SYSDATE, 'FAILED', SQLERRM);

ROLLBACK;

END AddNewCustomer;

/

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

CREATE TABLE SavingsAccounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER

);

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE SavingsAccounts

SET balance = balance + (balance \* 0.01);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END ProcessMonthlyInterest;

/

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

CREATE TABLE Employees (

employee\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

department\_id NUMBER,

salary NUMBER

);

CREATE TABLE Departments (

department\_id NUMBER PRIMARY KEY,

name VARCHAR2(100)

);

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;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END UpdateEmployeeBonus;

/

**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER

);

CREATE TABLE TransferLogs (

log\_id NUMBER PRIMARY KEY,

from\_account\_id NUMBER,

to\_account\_id NUMBER,

amount NUMBER,

transfer\_date DATE,

status VARCHAR2(20),

message VARCHAR2(255)

);

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

pragma EXCEPTION\_INIT(insufficient\_funds, -20001);

BEGIN

DECLARE

v\_from\_balance NUMBER;

BEGIN

SELECT balance INTO v\_from\_balance

FROM Accounts

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

IF v\_from\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

END;

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;

INSERT INTO TransferLogs (log\_id, from\_account\_id, to\_account\_id, amount, transfer\_date, status, message)

VALUES (TRANSFER\_LOG\_SEQ.NEXTVAL, p\_from\_account\_id, p\_to\_account\_id, p\_amount, SYSDATE, 'SUCCESS', 'Transfer completed successfully');

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

INSERT INTO TransferLogs (log\_id, from\_account\_id, to\_account\_id, amount, transfer\_date, status, message)

VALUES (TRANSFER\_LOG\_SEQ.NEXTVAL, p\_from\_account\_id, p\_to\_account\_id, p\_amount, SYSDATE, 'FAILED', 'Insufficient funds');

ROLLBACK;

WHEN OTHERS THEN

INSERT INTO TransferLogs (log\_id, from\_account\_id, to\_account\_id, amount, transfer\_date, status, message)

VALUES (TRANSFER\_LOG\_SEQ.NEXTVAL, p\_from\_account\_id, p\_to\_account\_id, p\_amount, SYSDATE, 'FAILED', SQLERRM);

ROLLBACK;

END TransferFunds;

/

**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

* + **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

CREATE TABLE Customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

date\_of\_birth DATE

);

CREATE OR REPLACE FUNCTION CalculateAge(p\_date\_of\_birth DATE) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

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

RETURN v\_age;

END CalculateAge;

/

**Scenario 2:** The bank needs to compute the monthly installment for a loan.

* + **Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

CREATE TABLE Loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

loan\_amount NUMBER,

interest\_rate NUMBER,

duration\_years NUMBER

);

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_duration\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_installment NUMBER;

v\_monthly\_interest\_rate NUMBER;

v\_number\_of\_payments NUMBER;

BEGIN

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

v\_number\_of\_payments := p\_duration\_years \* 12;

IF v\_monthly\_interest\_rate > 0 THEN

v\_monthly\_installment := p\_loan\_amount \* (v\_monthly\_interest\_rate \* POWER(1 + v\_monthly\_interest\_rate, v\_number\_of\_payments)) /

(POWER(1 + v\_monthly\_interest\_rate, v\_number\_of\_payments) - 1);

ELSE

v\_monthly\_installment := p\_loan\_amount / v\_number\_of\_payments;

END IF;

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

* + **Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER

);

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:** Automatically update the last modified date when a customer's record is updated.

* + **Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

**Create table:**

CREATE TABLE Customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

date\_of\_birth DATE,

last\_modified DATE

);

**Trigger:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.last\_modified := SYSDATE;

END UpdateCustomerLastModified;

/

**Scenario 2:** Maintain an audit log for all transactions.

* + **Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

**Create Table:**

CREATE TABLE Transactions (

transaction\_id NUMBER PRIMARY KEY,

account\_id NUMBER,

transaction\_date DATE,

amount NUMBER

);

CREATE TABLE AuditLog (

log\_id NUMBER PRIMARY KEY,

transaction\_id NUMBER,

log\_date DATE,

action VARCHAR2(50)

);

**Trigger:**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (log\_id, transaction\_id, log\_date, action)

VALUES (AUDIT\_LOG\_SEQ.NEXTVAL, :NEW.transaction\_id, SYSDATE, 'INSERT');

END LogTransaction;

/

**Scenario 3:** Enforce business rules on deposits and withdrawals.

* + **Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

**Create Table:**

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

balance NUMBER

);

CREATE TABLE Transactions (

transaction\_id NUMBER PRIMARY KEY,

account\_id NUMBER,

transaction\_date DATE,

amount NUMBER,

transaction\_type VARCHAR2(10)

);

**Trigger:**

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 account\_id = :NEW.account\_id;

IF :NEW.transaction\_type = 'WITHDRAWAL' THEN

IF v\_balance < :NEW.amount THEN

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

END IF;

ELSIF :NEW.transaction\_type = 'DEPOSIT' THEN

IF :NEW.amount <= 0 THEN

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

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'Invalid transaction type.');

END IF;

END CheckTransactionRules;

/

**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

* + **Question:** Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

**Create table:**

CREATE TABLE Transactions (

transaction\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

transaction\_date DATE,

amount NUMBER

);

CREATE TABLE Customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100)

);

DECLARE

CURSOR c\_monthly\_statements IS

SELECT customer\_id, SUM(amount) AS total\_amount

FROM Transactions

WHERE EXTRACT(MONTH FROM transaction\_date) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM transaction\_date) = EXTRACT(YEAR FROM SYSDATE)

GROUP BY customer\_id;

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

v\_total\_amount NUMBER;

BEGIN

OPEN c\_monthly\_statements;

LOOP

FETCH c\_monthly\_statements INTO v\_customer\_id, v\_total\_amount;

EXIT WHEN c\_monthly\_statements%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ', Total Transactions: ' || v\_total\_amount);

END LOOP;

CLOSE c\_monthly\_statements;

END;

/

**Scenario 2:** Apply annual fee to all accounts.

* + **Question:** Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

**Create Table:**

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

balance NUMBER

);

DECLARE

CURSOR c\_accounts IS

SELECT account\_id, balance

FROM Accounts;

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

v\_balance NUMBER;

annual\_fee NUMBER := 100;

BEGIN

OPEN c\_accounts;

LOOP

FETCH c\_accounts INTO v\_account\_id, v\_balance;

EXIT WHEN c\_accounts%NOTFOUND;

UPDATE Accounts

SET balance = v\_balance - annual\_fee

WHERE account\_id = v\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_account\_id || ', New Balance: ' || (v\_balance - annual\_fee));

END LOOP;

CLOSE c\_accounts;

COMMIT;

END;

/

**Scenario 3:** Update the interest rate for all loans based on a new policy.

* + **Question:** Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy.

**Create Table:**

CREATE TABLE Loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

loan\_amount NUMBER,

interest\_rate NUMBER

);

DECLARE

CURSOR c\_loans IS

SELECT loan\_id, interest\_rate

FROM Loans;

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

v\_current\_rate NUMBER;

v\_new\_rate NUMBER;

BEGIN

OPEN c\_loans;

LOOP

FETCH c\_loans INTO v\_loan\_id, v\_current\_rate;

EXIT WHEN c\_loans%NOTFOUND;

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

UPDATE Loans

SET interest\_rate = v\_new\_rate

WHERE loan\_id = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || v\_loan\_id || ', New Interest Rate: ' || v\_new\_rate);

END LOOP;

CLOSE c\_loans;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

* + **Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

**Create Table:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_date\_of\_birth DATE);

PROCEDURE UpdateCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_date\_of\_birth DATE);

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

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_date\_of\_birth DATE) IS

BEGIN

INSERT INTO Customers (customer\_id, name, date\_of\_birth, last\_modified)

VALUES (p\_customer\_id, p\_name, p\_date\_of\_birth, SYSDATE);

COMMIT;

END AddCustomer;

PROCEDURE UpdateCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2, p\_date\_of\_birth DATE) IS

BEGIN

UPDATE Customers

SET name = p\_name, date\_of\_birth = p\_date\_of\_birth, last\_modified = SYSDATE

WHERE customer\_id = p\_customer\_id;

COMMIT;

END UpdateCustomer;

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

v\_balance NUMBER;

BEGIN

SELECT SUM(balance) INTO v\_balance

FROM Accounts

WHERE customer\_id = p\_customer\_id

GROUP BY customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2:** Create a package to manage employee data.

* + **Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

**Create Table:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

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

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

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\_salary NUMBER, p\_department VARCHAR2) IS

BEGIN

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

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

COMMIT;

END HireEmployee;

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

BEGIN

UPDATE Employees

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

WHERE employee\_id = p\_employee\_id;

COMMIT;

END UpdateEmployee;

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 0;

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3:** Group all account-related operations into a package.

* + **Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

**Create Table:**

CREATE OR REPLACE PACKAGE AccountOperations AS

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

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) IS

BEGIN

INSERT INTO Accounts (account\_id, customer\_id, balance)

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

COMMIT;

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id NUMBER) IS

BEGIN

DELETE FROM Accounts

WHERE account\_id = p\_account\_id;

COMMIT;

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;

END GetTotalBalance;

END AccountOperations;

/