**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.

**QUERIES:**

DECLARE

CURSOR c\_customers IS

SELECT CustomerID, DOB

FROM Customers;

v\_customer\_id Customers.CustomerID%TYPE;

v\_dob Customers.DOB%TYPE;

v\_age NUMBER;

BEGIN

FOR customer\_rec IN c\_customers LOOP

-- Calculate age

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, customer\_rec.DOB) / 12);

IF v\_age > 60 THEN

-- Apply 1% discount to loan interest rates

UPDATE Loans

SET InterestRate = InterestRate \* 0.99

WHERE CustomerID = customer\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount to loan for CustomerID: ' || customer\_rec.CustomerID || ' (Age: ' || v\_age || ')');

END IF;

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

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.

**QUERIES:**

DECLARE

CURSOR c\_customers IS

SELECT CustomerID, Balance

FROM Customers;

v\_customer\_id Customers.CustomerID%TYPE;

v\_balance Customers.Balance%TYPE;

BEGIN

FOR customer\_rec IN c\_customers LOOP

IF customer\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = customer\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Promoted CustomerID: ' || customer\_rec.CustomerID || ' to VIP status.');

ELSE

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = customer\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || customer\_rec.CustomerID || ' is not VIP.');

END IF;

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

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.

**QUERIES:**

DECLARE

CURSOR c\_loans\_due IS

SELECT c.Name, l.LoanID, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND (SYSDATE + 30);

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Loan Reminders (Due in next 30 days) ---');

FOR loan\_rec IN c\_loans\_due LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder for ' || loan\_rec.Name || ': Your loan (ID: ' || loan\_rec.LoanID || ') is due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || '.');

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('--- End of Reminders ---');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No loans due in the next 30 days.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

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.

QUERIES:

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_source\_account\_id IN NUMBER,

p\_target\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_source\_balance NUMBER;

v\_target\_balance NUMBER;

BEGIN

-- Check if accounts exist

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

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

IF v\_source\_balance < p\_amount THEN

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

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_source\_account\_id;

-- Add to target

UPDATE Accounts

SET Balance = Balance + p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_target\_account\_id;

-- Log the transaction

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

VALUES (SEQ\_TRANSACTION\_ID.NEXTVAL, p\_source\_account\_id, SYSDATE, p\_amount, 'TransferOut');

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

VALUES (SEQ\_TRANSACTION\_ID.NEXTVAL, p\_target\_account\_id, SYSDATE, p\_amount, 'TransferIn');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully from AccountID ' || p\_source\_account\_id || ' to AccountID ' || p\_target\_account\_id || '. Amount: ' || p\_amount);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

INSERT INTO AuditLog (Action, Message) VALUES ('Fund Transfer Error', 'One or both account IDs do not exist: Source ' || p\_source\_account\_id || ', Target ' || p\_target\_account\_id);

COMMIT; -- Commit the audit log entry

DBMS\_OUTPUT.PUT\_LINE('Error: One or both accounts not found.');

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

INSERT INTO AuditLog (Action, Message) VALUES ('Fund Transfer Error', 'Duplicate transaction ID encountered during transfer.');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Error: Duplicate transaction ID. Please retry.');

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO AuditLog (Action, Message) VALUES ('Fund Transfer Error', 'An unexpected error occurred during transfer: ' || SQLERRM);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('An error occurred during fund transfer: ' || SQLERRM);

END;

/

-- Need a sequence for TransactionID if not already present

CREATE SEQUENCE SEQ\_TRANSACTION\_ID START WITH 100 INCREMENT BY 1 NOCACHE;

-- Example usage:

-- EXEC SafeTransferFunds(1, 2, 500); -- Successful transfer

-- EXEC SafeTransferFunds(1, 2, 2000); -- Insufficient funds

-- EXEC SafeTransferFunds(99, 1, 100); -- Non-existent account

**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.

QUERIES:

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage\_increase IN NUMBER

)

IS

v\_current\_salary Employees.Salary%TYPE;

BEGIN

SELECT Salary INTO v\_current\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

UPDATE Employees

SET Salary = v\_current\_salary \* (1 + p\_percentage\_increase / 100)

WHERE EmployeeID = p\_employee\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for EmployeeID: ' || p\_employee\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

INSERT INTO AuditLog (Action, Message) VALUES ('Salary Update Error', 'EmployeeID ' || p\_employee\_id || ' does not exist.');

COMMIT;

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

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO AuditLog (Action, Message) VALUES ('Salary Update Error', 'An unexpected error occurred while updating salary for EmployeeID ' || p\_employee\_id || ': ' || SQLERRM);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END;

/

-- Example usage:

-- EXEC UpdateSalary(1, 10); -- Successful update

-- EXEC UpdateSalary(99, 5); -- Non-existent employee

**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.

QUERIES:

CREATE OR REPLACE PROCEDURE AddNewCustomer (

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);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('New customer added successfully with CustomerID: ' || p\_customer\_id);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

INSERT INTO AuditLog (Action, Message) VALUES ('Add Customer Error', 'Customer with ID ' || p\_customer\_id || ' already exists. Insertion prevented.');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' already exists. Cannot add duplicate.');

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO AuditLog (Action, Message) VALUES ('Add Customer Error', 'An unexpected error occurred while adding customer ' || p\_customer\_id || ': ' || SQLERRM);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END;

/

-- Example usage:

-- EXEC AddNewCustomer(5, 'Eva Green', TO\_DATE('1995-01-01', 'YYYY-MM-DD'), 2500); -- Successful

-- EXEC AddNewCustomer(1, 'Duplicate User', TO\_DATE('1980-01-01', 'YYYY-MM-DD'), 100); -- Duplicate ID

**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.

QUERIES:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01, -- 1% interest

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An error occurred during monthly interest processing: ' || SQLERRM);

END;

/

**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.

QUERIES:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

)

IS

v\_rows\_updated NUMBER;

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

v\_rows\_updated := SQL%ROWCOUNT;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to ' || v\_rows\_updated || ' employees in ' || p\_department || ' department.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An error occurred while updating employee bonus: ' || SQLERRM);

END;

/

-- Example usage:

-- EXEC UpdateEmployeeBonus('HR', 5);

-- EXEC UpdateEmployeeBonus('IT', 7);

**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.

QUERIES:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account\_id IN NUMBER,

p\_target\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_source\_balance Accounts.Balance%TYPE;

account\_not\_found EXCEPTION;

PRAGMA EXCEPTION\_INIT(account\_not\_found, -20002);

BEGIN

IF p\_amount <= 0 THEN

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

END IF;

SELECT Balance INTO v\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account\_id;

IF v\_source\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_source\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_target\_account\_id;

-- Log the transaction (similar to SafeTransferFunds, ensure SEQ\_TRANSACTION\_ID exists)

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

VALUES (SEQ\_TRANSACTION\_ID.NEXTVAL, p\_source\_account\_id, SYSDATE, p\_amount, 'TransferOut');

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

VALUES (SEQ\_TRANSACTION\_ID.NEXTVAL, p\_target\_account\_id, SYSDATE, p\_amount, 'TransferIn');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Successfully transferred ' || p\_amount || ' from account ' || p\_source\_account\_id || ' to ' || p\_target\_account\_id || '.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'One or both account IDs do not exist.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An error occurred during fund transfer: ' || SQLERRM);

END;

/

-- Example usage:

-- EXEC TransferFunds(1, 2, 100); -- Successful transfer

-- EXEC TransferFunds(1, 2, 2000); -- Insufficient balance

-- EXEC TransferFunds(1, 99, 50); -- Invalid target account

**Schema to be Created**

*CREATE TABLE Customers (*

*CustomerID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*DOB DATE,*

*Balance NUMBER,*

*LastModified DATE*

*);*

*CREATE TABLE Accounts (*

*AccountID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*AccountType VARCHAR2(20),*

*Balance NUMBER,*

*LastModified DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*

*CREATE TABLE Transactions (*

*TransactionID NUMBER PRIMARY KEY,*

*AccountID NUMBER,*

*TransactionDate DATE,*

*Amount NUMBER,*

*TransactionType VARCHAR2(10),*

*FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)*

*);*

*CREATE TABLE Loans (*

*LoanID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*LoanAmount NUMBER,*

*InterestRate NUMBER,*

*StartDate DATE,*

*EndDate DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*

*CREATE TABLE Employees (*

*EmployeeID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*Position VARCHAR2(50),*

*Salary NUMBER,*

*Department VARCHAR2(50),*

*HireDate DATE*

*);*

**Example Scripts for Sample Data Insertion**

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

*VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);*

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

*VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);*

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

*VALUES (1, 1, 'Savings', 1000, SYSDATE);*

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

*VALUES (2, 2, 'Checking', 1500, SYSDATE);*

*INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)*

*VALUES (1, 1, SYSDATE, 200, 'Deposit');*

*INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)*

*VALUES (2, 2, SYSDATE, 300, 'Withdrawal');*

*INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)*

*VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));*

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

*VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));*

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

*VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));*