**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.
  + **Code:**

BEGIN

FOR rec IN (SELECT l.LoanID, l.InterestRate, c.DOB

FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID) LOOP

IF MONTHS\_BETWEEN(SYSDATE, rec.DOB)/12 > 60 THEN

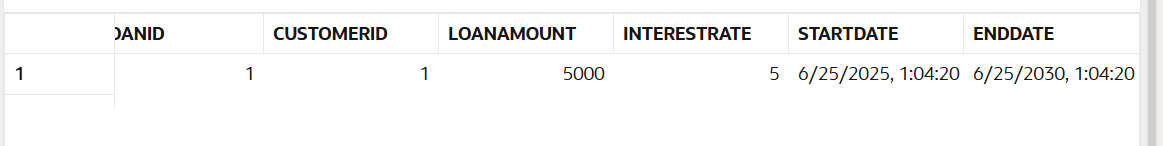
UPDATE Loans SET InterestRate = rec.InterestRate - 1

WHERE LoanID = rec.LoanID;

END IF;

END LOOP;

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.
  + **Code:**

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

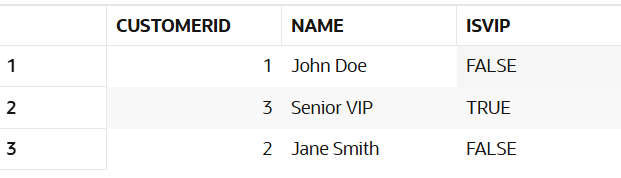
IF rec.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

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.
  + **Code:**

BEGIN

FOR rec IN (SELECT l.LoanID, c.Name FROM Loans l JOIN Customers c

ON l.CustomerID = c.CustomerID

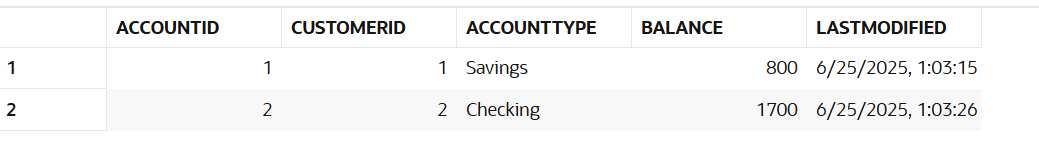
WHERE l.EndDate <= SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID ||

' for ' || rec.Name || ' is due soon.');

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.
  + **Code:**

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

BEGIN

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

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'From account not found.');

END IF;

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

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'To account not found.');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

**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.
  + **Code:**

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_empid NUMBER, p\_percent NUMBER) IS

BEGIN

UPDATE Employees SET Salary = Salary + (Salary \* p\_percent / 100)

WHERE EmployeeID = p\_empid;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Employee not found.');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

**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.
  + **Code:**

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

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

ROLLBACK;

END;

**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.
  + **Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

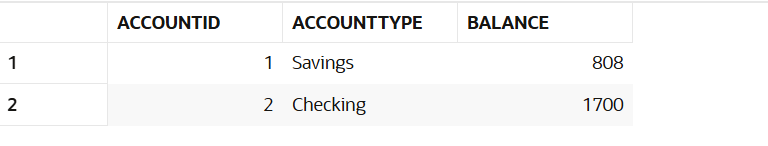
BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

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.
  + **Code:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_dept VARCHAR2, p\_bonus NUMBER) IS

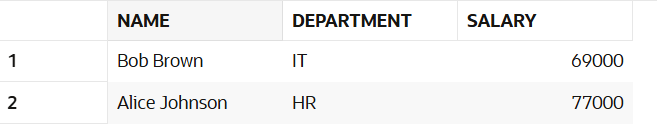
BEGIN

UPDATE Employees

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

WHERE Department = p\_dept;

END;



**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.
  + **Code:**

CREATE OR REPLACE PROCEDURE TransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amt NUMBER) IS

v\_balance NUMBER;

BEGIN

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

IF v\_balance >= p\_amt THEN

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

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

COMMIT;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient funds.');

END IF;

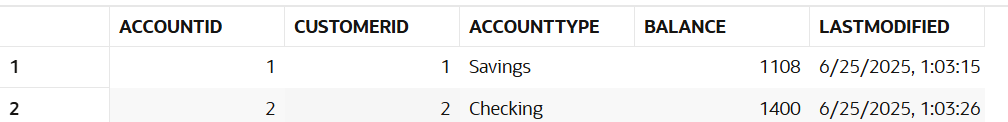
EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

****

**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.
  + **Code:**

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

BEGIN

RETURN TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

END;



**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.
  + **Code:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_amount NUMBER, p\_rate NUMBER, p\_years NUMBER) RETURN NUMBER IS

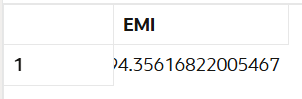
v\_months NUMBER := p\_years \* 12;

v\_monthly\_rate NUMBER := p\_rate / 1200;

BEGIN

RETURN (p\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -v\_months));

END;



**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.
  + **Code:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_acctid NUMBER, p\_amt NUMBER) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

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

RETURN v\_balance >= p\_amt;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;



**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.
  + **Code:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

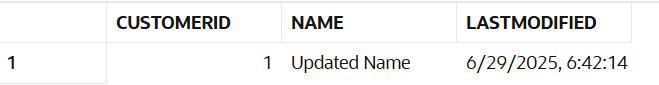
BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;



**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.
  + **Code:**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (LogID, AccountID, LogDate, Details)

VALUES (AuditLog\_seq.NEXTVAL, :NEW.AccountID, SYSDATE, 'Transaction recorded.');

END;



**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.
  + **Code:**

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 AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

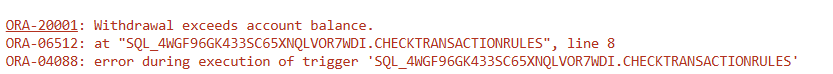
RAISE\_APPLICATION\_ERROR(-20004, 'Insufficient funds.');

ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Deposit amount must be positive.');

END IF;

END;

****

**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.
  + **Code:**

DECLARE

CURSOR txn\_cursor IS

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

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE TO\_CHAR(t.TransactionDate, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY');

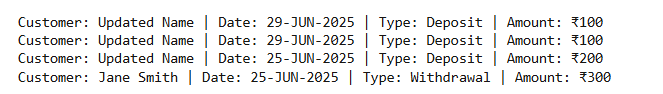
BEGIN

FOR rec IN txn\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.Name || ' | Amount: ' || rec.Amount || ' | Date: ' || rec.TransactionDate);

END LOOP;

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.
  + **Code:**

DECLARE

CURSOR acct\_cursor IS SELECT AccountID FROM Accounts;

BEGIN

FOR rec IN acct\_cursor LOOP

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = rec.AccountID;

END LOOP;

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.
  + **Code:**

DECLARE

CURSOR loan\_cursor IS SELECT LoanID, InterestRate FROM Loans;

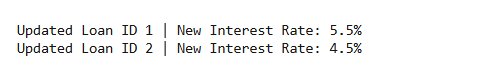
BEGIN

FOR rec IN loan\_cursor LOOP

UPDATE Loans SET InterestRate = rec.InterestRate + 0.5 WHERE LoanID = rec.LoanID;

END LOOP;

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.
  + **Code:**

CREATE OR REPLACE PACKAGE CustomerManagement IS

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

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2);

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER;

END;

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

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

BEGIN

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

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

COMMIT;

END;

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2) IS

BEGIN

UPDATE Customers

SET Name = p\_name, LastModified = SYSDATE

WHERE CustomerID = p\_id;

COMMIT;

END;

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

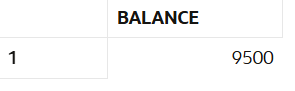
BEGIN

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

RETURN v\_balance;

END;

END;

/

**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.
  + **Code:**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_hiredate DATE) IS

BEGIN

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

VALUES (p\_id, p\_name, p\_position, p\_salary, p\_dept, p\_hiredate);

COMMIT;

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER) IS

BEGIN

UPDATE Employees

SET Salary = p\_salary

WHERE EmployeeID = p\_id;

COMMIT;

END;

FUNCTION GetAnnualSalary(p\_empid NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

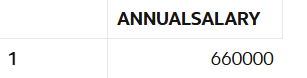
BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_empid;

RETURN v\_salary \* 12;

END;

END;

/

**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.
  + **Code:**

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_acctid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

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

VALUES (p\_acctid, p\_custid, p\_type, p\_balance, SYSDATE);

COMMIT;

END;

PROCEDURE CloseAccount(p\_acctid NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_acctid;

COMMIT;

END;

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER IS

v\_total NUMBER;

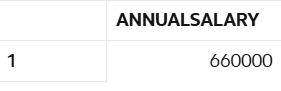
BEGIN

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

RETURN NVL(v\_total, 0);

END;

END;

/

**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'));*