**Exercise 1: Control Structures**

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

);

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'));

BEGIN

FOR cust IN (SELECT CustomerID, DOB FROM Customers) LOOP

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

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

ALTER TABLE Customers ADD IsVIP CHAR(1);

BEGIN

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

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

SET SERVEROUTPUT ON;

BEGIN

FOR loan IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

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

END LOOP;

END;

/

**Output:**

|  |  |
| --- | --- |
|  |  |

**Exercise 2: Error Handling**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

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

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

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

VALUES (Transactions\_SEQ.NEXTVAL, p\_from\_account\_id, SYSDATE, 0, 'Error');

END;

/

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_emp\_id IN NUMBER,

p\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee not found');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

VALUES (Transactions\_SEQ.NEXTVAL, NULL, SYSDATE, 0, 'Error');

END;

/

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) AS

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

ROLLBACK;

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

VALUES (Transactions\_SEQ.NEXTVAL, NULL, SYSDATE, 0, 'Duplicate');

WHEN OTHERS THEN

ROLLBACK;

END;

/

**Output:**

|  |  |
| --- | --- |
|  |  |

**Exercise 3: Stored Procedures**

**CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS**

**BEGIN**

**FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP**

**UPDATE Accounts**

**SET Balance = Balance + (acc.Balance \* 0.01)**

**WHERE AccountID = acc.AccountID;**

**END LOOP;**

**COMMIT;**

**END;**

**/**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus ( p\_dept IN VARCHAR2, p\_bonus\_pct IN NUMBER ) IS BEGIN UPDATE Employees SET Salary = Salary + (Salary \* p\_bonus\_pct / 100) WHERE Department = p\_dept; COMMIT; END; /

CREATE OR REPLACE PROCEDURE TransferFunds ( p\_from\_acc IN NUMBER, p\_to\_acc IN NUMBER, p\_amount IN NUMBER ) IS v\_balance NUMBER; BEGIN SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_acc FOR UPDATE;

IF v\_balance < p\_amount THEN RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance'); END IF;

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

COMMIT; END; /

**Output:**

|  |  |
| --- | --- |
|  |  |

**Exercise 4: Functions**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE)

RETURN NUMBER IS

BEGIN

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

END;

/

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_years NUMBER

)

RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_months NUMBER;

v\_installment NUMBER;

BEGIN

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

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

v\_installment := (p\_loan\_amount \* v\_monthly\_rate) /

(1 - POWER(1 + v\_monthly\_rate, -v\_months));

RETURN v\_installment;

END;

/

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 AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

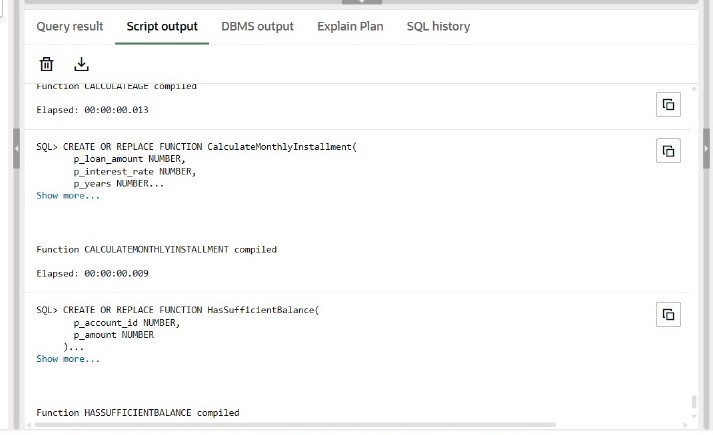
WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

**Output:**



**Exercise 5: Triggers**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

LogDate DATE,

Action VARCHAR2(50)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog(TransactionID, LogDate, Action)

VALUES (:NEW.TransactionID, SYSDATE, 'Transaction Inserted');

END;

/

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal exceeds account balance');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

END IF;

END;

/

**Output:**

|  |  |
| --- | --- |
|  |  |

**Exercise 6: Cursors**

SET SERVEROUTPUT ON;

DECLARE

CURSOR cur\_transactions IS

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

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

v\_account\_id Accounts.AccountID%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_date Transactions.TransactionDate%TYPE;

v\_type Transactions.TransactionType%TYPE;

v\_name Customers.Name%TYPE;

BEGIN

OPEN cur\_transactions;

LOOP

FETCH cur\_transactions INTO v\_account\_id, v\_amount, v\_date, v\_type, v\_name;

EXIT WHEN cur\_transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_name || ', Account: ' || v\_account\_id ||

', ' || v\_type || ' of $' || v\_amount || ' on ' || TO\_CHAR(v\_date, 'YYYY-MM-DD'));

END LOOP;

CLOSE cur\_transactions;

END;

/

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_acc\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_fee CONSTANT NUMBER := 100;

BEGIN

OPEN acc\_cursor;

LOOP

FETCH acc\_cursor INTO v\_acc\_id, v\_balance;

EXIT WHEN acc\_cursor%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_fee

WHERE AccountID = v\_acc\_id;

END LOOP;

CLOSE acc\_cursor;

COMMIT;

END;

/

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate FROM Loans;

v\_loan\_id Loans.LoanID%TYPE;

v\_old\_rate Loans.InterestRate%TYPE;

v\_new\_rate NUMBER;

BEGIN

OPEN loan\_cursor;

LOOP

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

EXIT WHEN loan\_cursor%NOTFOUND;

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

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = v\_loan\_id;

END LOOP;

CLOSE loan\_cursor;

COMMIT;

END;

/

**Output:**

|  |  |
| --- | --- |
|  |  |

**Exercise 7: Packages**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

PROCEDURE UpdateCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER);

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

BEGIN

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Customer added: ' || p\_Name);

END;

PROCEDURE UpdateCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_Name, Balance = p\_Balance, LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer updated: ' || p\_Name);

END;

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Customers WHERE CustomerID = p\_CustomerID;

RETURN v\_Balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

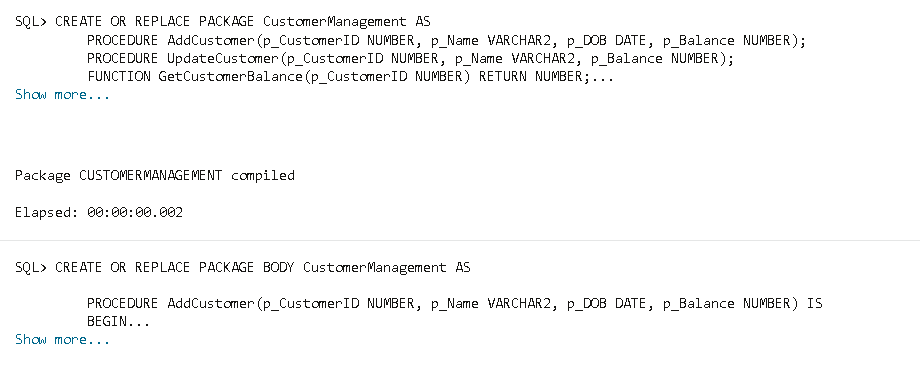
RETURN NULL;

END;

END CustomerManagement;

/

Output:



CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2);

PROCEDURE UpdateEmployee(p\_EmployeeID NUMBER, p\_Salary NUMBER);

FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2) IS

BEGIN

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

VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Employee hired: ' || p\_Name);

END;

PROCEDURE UpdateEmployee(p\_EmployeeID NUMBER, p\_Salary NUMBER) IS

BEGIN

UPDATE Employees

SET Salary = p\_Salary

WHERE EmployeeID = p\_EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Employee salary updated.');

END;

FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

SELECT Salary INTO v\_Salary FROM Employees WHERE EmployeeID = p\_EmployeeID;

RETURN v\_Salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

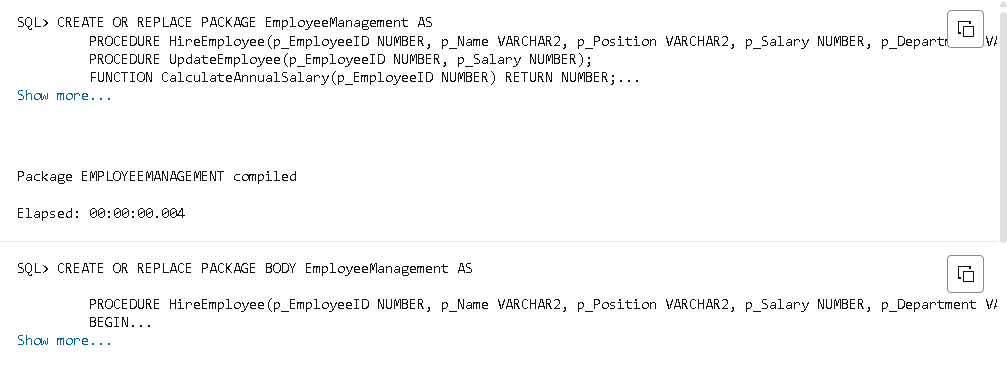
RETURN NULL;

END;

END EmployeeManagement;

/

Output:



CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_AccountType VARCHAR2, p\_Balance NUMBER);

PROCEDURE CloseAccount(p\_AccountID NUMBER);

FUNCTION GetTotalCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_AccountType VARCHAR2, p\_Balance NUMBER) IS

BEGIN

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

VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_Balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Account opened: ' || p\_AccountID);

END;

PROCEDURE CloseAccount(p\_AccountID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_AccountID;

DBMS\_OUTPUT.PUT\_LINE('Account closed: ' || p\_AccountID);

END;

FUNCTION GetTotalCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_TotalBalance FROM Accounts WHERE CustomerID = p\_CustomerID;

RETURN NVL(v\_TotalBalance, 0);

END;

END AccountOperations;

/

**Output**

|  |  |
| --- | --- |
|  |  |