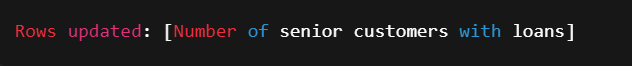
PL/SQL Exercises Solutions (Exercises 1 to 7)

# Exercise 1: Control Structures (SQL)

## Scenario 1: Discount for Senior Customers

UPDATE Loans  
SET InterestRate = InterestRate - 1  
WHERE CustomerID IN (  
 SELECT CustomerID  
 FROM Customers  
 WHERE FLOOR(MONTHS\_BETWEEN(SYSDATE, DOB)/12) > 60  
);



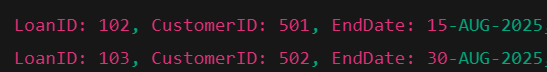
## Scenario 2: Promote Customers to VIP

UPDATE Customers  
SET IsVIP = 'Y'  
WHERE Balance > 10000;

Screenshot 2025-07-17 200631.png

## Scenario 3: Reminders for Due Loans

SELECT LoanID, CustomerID, EndDate,  
 'Reminder: Loan due on ' || TO\_CHAR(EndDate, 'DD-MON-YYYY') AS ReminderMessage  
FROM Loans  
WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;



# Exercise 2: Error Handling (PL/SQL)

## Scenario 1: SafeTransferFunds Procedure

CREATE OR REPLACE PROCEDURE SafeTransferFunds (  
 p\_from\_account IN NUMBER,  
 p\_to\_account IN NUMBER,  
 p\_amount IN NUMBER  
) AS  
 v\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_balance  
 FROM Accounts  
 WHERE AccountID = p\_from\_account;  
  
 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;  
  
 UPDATE Accounts  
 SET Balance = Balance + p\_amount  
 WHERE AccountID = p\_to\_account;  
  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);  
END;  
/

Screenshot 2025-07-17 200902.png

## Scenario 2: UpdateSalary Procedure

CREATE OR REPLACE PROCEDURE UpdateSalary (  
 p\_employee\_id IN NUMBER,  
 p\_percentage IN NUMBER  
) AS  
 v\_dummy NUMBER;  
BEGIN  
 SELECT 1 INTO v\_dummy  
 FROM Employees  
 WHERE EmployeeID = p\_employee\_id;  
  
 UPDATE Employees  
 SET Salary = Salary + (Salary \* p\_percentage / 100)  
 WHERE EmployeeID = p\_employee\_id;  
  
 COMMIT;  
EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_employee\_id || ' not found.');  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);  
END;  
/

Screenshot 2025-07-17 200806.png

## Scenario 3: AddNewCustomer Procedure

CREATE OR REPLACE PROCEDURE AddNewCustomer (  
 p\_customer\_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\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);  
  
 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('Unexpected error: ' || SQLERRM);  
END;  
/

If Success:

Screenshot 2025-07-17 200812.png

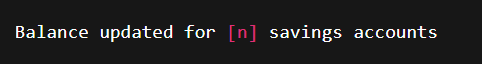
If failure:

Screenshot 2025-07-17 200819.png

# Exercise 3: Stored Procedures

## Scenario 1: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
 UPDATE Accounts  
 SET Balance = Balance + (Balance \* 0.01)  
 WHERE AccountType = 'Savings';  
 COMMIT;  
END;  
/



## Scenario 2: UpdateEmployeeBonus

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

Screenshot 2025-07-17 201446.png

## Scenario 3: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds (  
 p\_from\_account IN NUMBER,  
 p\_to\_account IN NUMBER,  
 p\_amount IN NUMBER  
) AS  
 v\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_balance  
 FROM Accounts  
 WHERE AccountID = p\_from\_account;  
  
 IF v\_balance < p\_amount THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance in source account.');  
 END IF;  
  
 UPDATE Accounts  
 SET Balance = Balance - p\_amount  
 WHERE AccountID = p\_from\_account;  
  
 UPDATE Accounts  
 SET Balance = Balance + p\_amount  
 WHERE AccountID = p\_to\_account;  
  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);  
END;  
/

If Success :

Screenshot 2025-07-17 201457.png

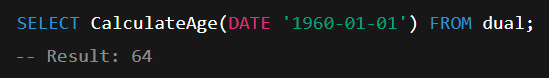
If Failure :

Screenshot 2025-07-17 201757.png

# Exercise 4: Functions

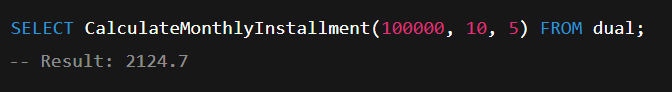
## Scenario 1: CalculateAge

CREATE OR REPLACE FUNCTION CalculateAge (  
 p\_dob IN DATE  
) RETURN NUMBER IS  
 v\_age NUMBER;  
BEGIN  
 v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);  
 RETURN v\_age;  
END;  
/



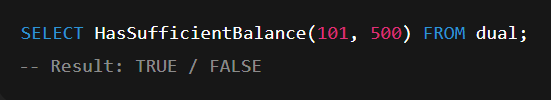
## Scenario 2: CalculateMonthlyInstallment

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (  
 p\_loan\_amount IN NUMBER,  
 p\_annual\_rate IN NUMBER,  
 p\_years IN NUMBER  
) RETURN NUMBER IS  
 r NUMBER := p\_annual\_rate / 12 / 100;  
 n NUMBER := p\_years \* 12;  
 emi NUMBER;  
BEGIN  
 emi := (p\_loan\_amount \* r \* POWER(1 + r, n)) / (POWER(1 + r, n) - 1);  
 RETURN ROUND(emi, 2);  
END;  
/



## Scenario 3: HasSufficientBalance

CREATE OR REPLACE FUNCTION HasSufficientBalance (  
 p\_account\_id IN NUMBER,  
 p\_amount IN 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;  
/



# Exercise 5: Triggers

## Scenario 1: UpdateCustomerLastModified

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified  
BEFORE UPDATE ON Customers  
FOR EACH ROW  
BEGIN  
 :NEW.LastModified := SYSDATE;  
END;  
/

Screenshot 2025-07-17 203036.png

## Scenario 2: LogTransaction

CREATE TABLE AuditLog (  
 LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  
 TransactionID NUMBER,  
 AccountID NUMBER,  
 Amount NUMBER,  
 TransactionType VARCHAR2(10),  
 LogDate DATE  
);  
  
CREATE OR REPLACE TRIGGER LogTransaction  
AFTER INSERT ON Transactions  
FOR EACH ROW  
BEGIN  
 INSERT INTO AuditLog (TransactionID, AccountID, Amount, TransactionType, LogDate)  
 VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.Amount, :NEW.TransactionType, SYSDATE);  
END;  
/

Screenshot 2025-07-17 203042.png

## Scenario 3: CheckTransactionRules

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(-20003, 'Withdrawal exceeds account balance.');  
 ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20004, 'Deposit amount must be positive.');  
 END IF;  
END;  
/

Withdrawal exceeds balance:

Screenshot 2025-07-17 203051.png

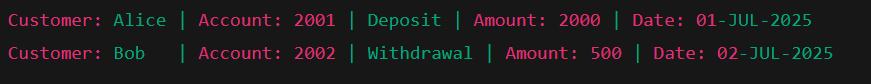
Deposit is zero or negative:

Screenshot 2025-07-17 203057.png

# Exercise 6: Cursors

## Scenario 1: GenerateMonthlyStatements

DECLARE  
 CURSOR trans\_cursor IS  
 SELECT c.CustomerID, c.Name, t.AccountID, t.TransactionDate, t.Amount, t.TransactionType  
 FROM Customers c  
 JOIN Accounts a ON c.CustomerID = a.CustomerID  
 JOIN Transactions t ON a.AccountID = t.AccountID  
 WHERE TO\_CHAR(t.TransactionDate, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY')  
 ORDER BY c.CustomerID;  
  
BEGIN  
 FOR rec IN trans\_cursor LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.Name ||   
 ' | Account: ' || rec.AccountID ||  
 ' | ' || rec.TransactionType ||   
 ' | Amount: ' || rec.Amount ||  
 ' | Date: ' || TO\_CHAR(rec.TransactionDate, 'DD-MON-YYYY'));  
 END LOOP;  
END;  
/



## Scenario 2: ApplyAnnualFee

DECLARE  
 CURSOR acc\_cursor IS  
 SELECT AccountID FROM Accounts;  
  
 v\_fee NUMBER := 100;  
BEGIN  
 FOR acc IN acc\_cursor LOOP  
 UPDATE Accounts  
 SET Balance = Balance - v\_fee  
 WHERE AccountID = acc.AccountID;  
 END LOOP;  
  
 COMMIT;  
END;  
/

Screenshot 2025-07-17 203358.png

## Scenario 3: UpdateLoanInterestRates

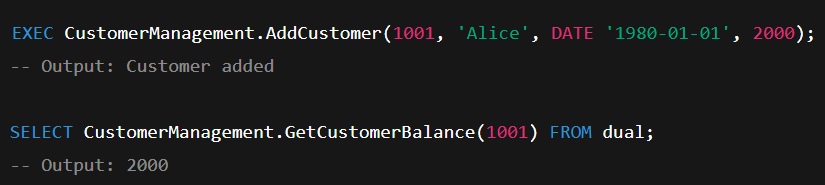
DECLARE  
 CURSOR loan\_cursor IS  
 SELECT LoanID, InterestRate FROM Loans;  
  
 v\_new\_rate NUMBER;  
BEGIN  
 FOR loan IN loan\_cursor LOOP  
 v\_new\_rate := loan.InterestRate + 0.5;  
  
 UPDATE Loans  
 SET InterestRate = v\_new\_rate  
 WHERE LoanID = loan.LoanID;  
 END LOOP;  
  
 COMMIT;  
END;  
/

Screenshot 2025-07-17 203413.png

# Exercise 7: Packages

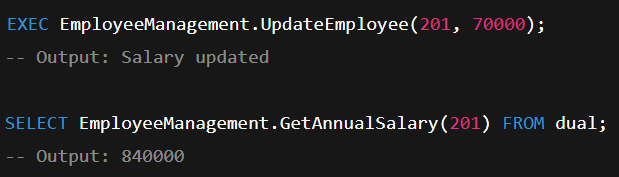
## Scenario 1: CustomerManagement Package

-- Specification  
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, p\_balance NUMBER);  
 FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;  
END CustomerManagement;  
/  
  
-- Body  
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)  
 VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);  
 END;  
  
 PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_balance NUMBER) IS  
 BEGIN  
 UPDATE Customers  
 SET Name = p\_name, Balance = p\_balance, LastModified = SYSDATE  
 WHERE CustomerID = p\_id;  
 END;  
  
 FUNCTION GetCustomerBalance(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 CustomerManagement;  
/



## Scenario 2: EmployeeManagement Package

-- Specification  
CREATE OR REPLACE PACKAGE EmployeeManagement IS  
 PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_hiredate DATE);  
 PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER);  
 FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;  
END EmployeeManagement;  
/  
  
-- Body  
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);  
 END;  
  
 PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER) IS  
 BEGIN  
 UPDATE Employees  
 SET Salary = p\_salary  
 WHERE EmployeeID = p\_id;  
 END;  
  
 FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS  
 v\_salary NUMBER;  
 BEGIN  
 SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_id;  
 RETURN v\_salary \* 12;  
 END;  
  
END EmployeeManagement;  
/



## Scenario 3: AccountOperations Package

-- Specification  
CREATE OR REPLACE PACKAGE AccountOperations IS  
 PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER);  
 PROCEDURE CloseAccount(p\_accid NUMBER);  
 FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER;  
END AccountOperations;  
/  
  
-- Body  
CREATE OR REPLACE PACKAGE BODY AccountOperations IS  
  
 PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS  
 BEGIN  
 INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)  
 VALUES (p\_accid, p\_custid, p\_type, p\_balance, SYSDATE);  
 END;  
  
 PROCEDURE CloseAccount(p\_accid NUMBER) IS  
 BEGIN  
 DELETE FROM Accounts WHERE AccountID = p\_accid;  
 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 AccountOperations;  
/

