**Customers Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CUSTOMERID | NAME | DOB | BALANCE | LASTMODIFIED |
| 3 | Ravi Kumar | 1959-01-10T00:00:00Z | 20000 | 2025-06-27T15:11:37Z |
| 4 | Anita Desai | 1962-09-25T00:00:00Z | 8000 | 2025-06-27T15:11:46Z |
| 5 | Michael Chen | 1975-03-15T00:00:00Z | 3000 | 2025-06-27T15:11:52Z |
| 1 | John Doe | 1985-05-15T00:00:00Z | 1000 | 2025-06-27T15:01:35Z |
| 2 | Jane Smith | 1990-07-20T00:00:00Z | 1500 | 2025-06-27T15:01:19Z |

**Accounts Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACCOUNTID | CUSTOMERID | ACCOUNTTYPE | BALANCE | LASTMODIFIED |
| 1 | 1 | Savings | 1000 | 2025-06-27T15:03:03Z |
| 2 | 2 | Checking | 1500 | 2025-06-27T15:03:18Z |
| 3 | 3 | Savings | 25000 | 2025-06-27T15:18:46Z |
| 5 | 5 | Savings | 4000 | 2025-06-27T15:19:10Z |
| 4 | 4 | Checking | 5000 | 2025-06-27T15:19:00Z |

**Transactions Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TRANSACTIONID | ACCOUNTID | TRANSACTIONDATE | AMOUNT | TRANSACTIONTYPE |
| 1 | 1 | 2025-06-27T15:03:33Z | 200 | Deposit |
| 2 | 2 | 2025-06-27T15:03:47Z | 300 | Withdrawal |
| 4 | 4 | 2025-06-27T15:22:32Z | 700 | Withdrawal |
| 3 | 3 | 2025-06-27T15:21:52Z | 1000 | Deposit |
| 5 | 5 | 2025-06-27T15:22:39Z | 200 | Deposit |

**Employees Table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPLOYEEID | NAME | POSITION | SALARY | DEPARTMENT | HIREDATE |
| 4 | Amit Gupta | Developer | 62000 | IT | 2018-07-10T00:00:00Z |
| 3 | Meera Sinha | Clerk | 35000 | Finance | 2020-02-01T00:00:00Z |
| 1 | Alice Johnson | Manager | 70000 | HR | 2015-06-15T00:00:00Z |
| 2 | Bob Brown | Developer | 60000 | IT | 2017-03-20T00:00:00Z |

**Loans Table:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| LOANID | CUSTOMERID | LOANAMOUNT | INTERESTRATE | STARTDATE | | ENDDATE |
| 2 | 2 | 10000 | 6.5 | 2025-06-27T15:25:33Z | 2029-06-27T15:25:33Z | |
| 1 | 1 | 5000 | 5 | 2025-06-27T15:04:02Z | 2030-06-27T15:04:02Z | |
| 3 | 3 | 20000 | 4.5 | 2025-06-27T15:25:45Z | 2028-06-27T15:25:45Z | |
| 4 | 4 | 7000 | 5.5 | 2025-06-27T15:25:53Z | 2027-06-27T15:25:53Z | |

**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.

**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.

**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.

**Solution:**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER

) IS

v\_balance NUMBER;

BEGIN

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

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;

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

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

**Result:**

Procedure SAFETRANSFERFUNDS compiled

Elapsed: 00:00:00.015

**Scenario 2:**

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

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

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

END;

**Result:**

Procedure SAFETRANSFERFUNDS compiled

Elapsed: 00:00:00.015

**Scenario 3:**

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('Customer ID already exists.');

WHEN OTHERS THEN

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

END;

**Result:**

Procedure ADDNEWCUSTOMER compiled

Elapsed: 00:00:00.016

**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.

**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.

**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.

**Solution**

**Scenario 1:**

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

BEGIN

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

END;

**Result:**

Function CALCULATEAGE compiled  
Elapsed: 00:00:00.017

**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_amount NUMBER, p\_rate NUMBER, p\_years NUMBER

) RETURN NUMBER IS

r\_monthly NUMBER := p\_rate / 12 / 100;

n NUMBER := p\_years \* 12;

BEGIN

RETURN p\_amount \* r\_monthly / (1 - POWER(1 + r\_monthly, -n));

END;

**Result:**

Function CALCULATEMONTHLYINSTALLMENT compiled  
Elapsed: 00:00:00.011

**Scenario 3:**

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

v\_bal NUMBER;

BEGIN

SELECT Balance INTO v\_bal FROM Accounts WHERE AccountID = p\_acct;

RETURN v\_bal >= p\_amt;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

**Result:**

Function HASSUFFICIENTBALANCE compiled  
Elapsed: 00:00:00.016

**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.

**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.

**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.

**Solution**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

**Result:**

Trigger UPDATECUSTOMERLASTMODIFIED compiled  
Elapsed: 00:00:00.019

**Scenario 2:**

CREATE TABLE AuditLog (

LogID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,

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(AccountID, Amount, TransactionType, LogDate)

VALUES (:NEW.AccountID, :NEW.Amount, :NEW.TransactionType, SYSDATE);

END;

**Result:**

Trigger LOGTRANSACTION compiled  
LINE/COL ERROR  
--------- -------------------------------------------------------------  
2/3 PL/SQL: SQL Statement ignored  
2/15 PL/SQL: ORA-00942: table or view does not exist  
Errors: check compiler log  
Elapsed: 00:00:00.018

**Scenario 3:**

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;

**Result:**  
Trigger CHECKTRANSACTIONRULES compiled  
Elapsed: 00:00:00.022

**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.

**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.

**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.

**Solution**

**Scenario 1:**

DECLARE

CURSOR trans\_cursor IS

SELECT \* FROM Transactions WHERE TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

BEGIN

FOR t IN trans\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ' || t.AccountID || ' transaction: ' || t.Amount || ' on ' || t.TransactionDate);

END LOOP;

END;

**Result:**

Account 1 transaction: 200 on 27-JUN-25

Account 2 transaction: 300 on 27-JUN-25

Account 4 transaction: 700 on 27-JUN-25

Account 3 transaction: 1000 on 27-JUN-25

Account 5 transaction: 200 on 27-JUN-25

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.015

**Scenario 2:**

DECLARE

CURSOR acct\_cursor IS SELECT AccountID, Balance FROM Accounts;

BEGIN

FOR a IN acct\_cursor LOOP

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccountID = a.AccountID;

END LOOP;

COMMIT;

END;

**Result:**

PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.012

**Scenario 3:**

DECLARE

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

BEGIN

FOR l IN loan\_cursor LOOP

UPDATE Loans

SET InterestRate = l.InterestRate + 0.5

WHERE LoanID = l.LoanID;

END LOOP;

COMMIT;

END;

**Result:**

PL/SQL procedure successfully completed. Elapsed: 00:00:00.01

**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.

**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.

**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.

**Solution**

**Scenario 1:**

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

END;

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

BEGIN

UPDATE Customers SET Name = p\_name WHERE CustomerID = p\_id;

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;

**Result:**

Package Body CUSTOMERMANAGEMENT compiled

LINE/COL ERROR

--------- -------------------------------------------------------------

0/0 PL/SQL: Compilation unit analysis terminated

1/14 PLS-00201: identifier 'CUSTOMERMANAGEMENT' must be declared

1/14 PLS-00304: cannot compile body of 'CUSTOMERMANAGEMENT' without its specification

Errors: check compiler log

Elapsed: 00:00:00.011

**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2, p\_date DATE);

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_pos VARCHAR2);

FUNCTION AnnualSalary(p\_id NUMBER) RETURN NUMBER;

END;

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2, p\_date DATE) IS

BEGIN

INSERT INTO Employees VALUES (p\_id, p\_name, p\_pos, p\_sal, p\_dept, p\_date);

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_pos VARCHAR2) IS

BEGIN

UPDATE Employees SET Position = p\_pos WHERE EmployeeID = p\_id;

END;

FUNCTION AnnualSalary(p\_id NUMBER) RETURN NUMBER IS

v\_sal NUMBER;

BEGIN

SELECT Salary INTO v\_sal FROM Employees WHERE EmployeeID = p\_id;

RETURN v\_sal \* 12;

END;

END;

**Result:**

Package Body EMPLOYEEMANAGEMENT compiled

LINE/COL ERROR

--------- -------------------------------------------------------------

0/0 PL/SQL: Compilation unit analysis terminated

1/14 PLS-00201: identifier 'EMPLOYEEMANAGEMENT' must be declared

1/14 PLS-00304: cannot compile body of 'EMPLOYEEMANAGEMENT' without its specification

Errors: check compiler log

Elapsed: 00:00:00.008

**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_bal NUMBER);

PROCEDURE CloseAccount(p\_id NUMBER);

FUNCTION TotalBalance(p\_cust\_id NUMBER) RETURN NUMBER;

END;

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_bal NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (p\_id, p\_cust\_id, p\_type, p\_bal, SYSDATE);

END;

PROCEDURE CloseAccount(p\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_id;

END;

FUNCTION TotalBalance(p\_cust\_id NUMBER) RETURN NUMBER IS

v\_total NUMBER;

BEGIN

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

RETURN v\_total;

END;

END;

**Result:**

Package Body ACCOUNTOPERATIONS compiled

LINE/COL ERROR

--------- -------------------------------------------------------------

0/0 PL/SQL: Compilation unit analysis terminated

1/14 PLS-00201: identifier 'ACCOUNTOPERATIONS' must be declared

1/14 PLS-00304: cannot compile body of 'ACCOUNTOPERATIONS' without its specification

Errors: check compiler log

Elapsed: 00:00:00.009