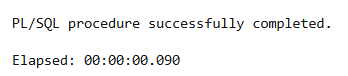
**Exercise 1: Control Structures**

**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.

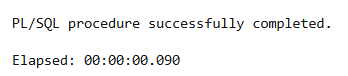
1. BEGIN
2. FOR c IN (SELECT \* FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID) LOOP
3. IF TRUNC(MONTHS\_BETWEEN(SYSDATE, c.DOB)/12) > 60 THEN
4. UPDATE Loans SET InterestRate = InterestRate - 1 WHERE LoanID = c.LoanID;
5. END IF;
6. END LOOP;
7. END;
8. /



**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.

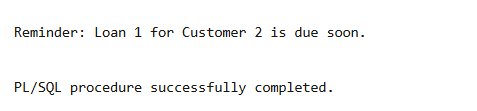
1. ALTER TABLE Customers ADD IsVIP VARCHAR2(5);
2. BEGIN
3. FOR c IN (SELECT \* FROM Customers) LOOP
4. IF c.Balance > 10000 THEN
5. UPDATE Customers SET IsVIP = 'TRUE', LastModified = SYSDATE WHERE CustomerID = c.CustomerID;
6. END IF;
7. END LOOP;
8. END;
9. /



**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.

1. BEGIN
2. FOR l IN (SELECT \* FROM Loans WHERE EndDate <= SYSDATE + 30) LOOP
3. DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || l.LoanID || ' for Customer ' || l.CustomerID || ' is due soon.');
4. END LOOP;
5. END;
6. /



**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.

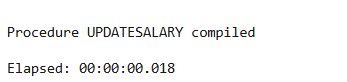
1. CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amt NUMBER) AS
2. BEGIN
3. UPDATE Accounts SET Balance = Balance - p\_amt WHERE AccountID = p\_from;
4. UPDATE Accounts SET Balance = Balance + p\_amt WHERE AccountID = p\_to;
5. COMMIT;
6. EXCEPTION
7. WHEN OTHERS THEN
8. ROLLBACK;
9. DBMS\_OUTPUT.PUT\_LINE('Error in fund transfer: ' || SQLERRM);
10. END;
11. /



**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.

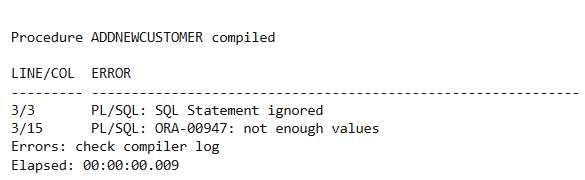
1. CREATE OR REPLACE PROCEDURE UpdateSalary(p\_empid NUMBER, p\_percent NUMBER) AS
2. BEGIN
3. UPDATE Employees SET Salary = Salary + (Salary \* p\_percent/100) WHERE EmployeeID = p\_empid;
4. IF SQL%ROWCOUNT = 0 THEN
5. RAISE\_APPLICATION\_ERROR(-20001, 'Employee not found');
6. END IF;
7. EXCEPTION
8. WHEN OTHERS THEN
9. DBMS\_OUTPUT.PUT\_LINE('Error updating salary: ' || SQLERRM);
10. END;
11. /



**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.

1. CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_bal NUMBER) AS
2. BEGIN
3. INSERT INTO Customers VALUES(p\_id, p\_name, p\_dob, p\_bal, SYSDATE);
4. EXCEPTION
5. WHEN DUP\_VAL\_ON\_INDEX THEN
6. DBMS\_OUTPUT.PUT\_LINE('Customer ID already exists.');
7. END;
8. /

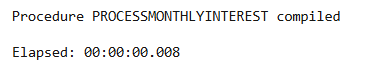


**Exercise 3: Stored Procedures**

**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.

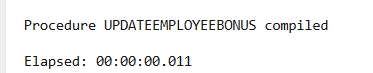
1. CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS
2. BEGIN
3. UPDATE Accounts SET Balance = Balance + (Balance \* 0.01) WHERE AccountType = 'Savings';
4. END;
5. /



**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.

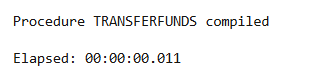
1. CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_dept VARCHAR2, p\_bonus NUMBER) AS
2. BEGIN
3. UPDATE Employees SET Salary = Salary + (Salary \* p\_bonus/100) WHERE Department = p\_dept;
4. END;
5. /



**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.

1. CREATE OR REPLACE PROCEDURE TransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amt NUMBER) AS
2. v\_balance NUMBER;
3. BEGIN
4. SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from;
5. IF v\_balance >= p\_amt THEN
6. UPDATE Accounts SET Balance = Balance - p\_amt WHERE AccountID = p\_from;
7. UPDATE Accounts SET Balance = Balance + p\_amt WHERE AccountID = p\_to;
8. ELSE
9. RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient funds');
10. END IF;
11. END;
12. /

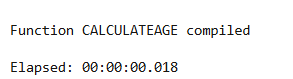


**Exercise 4: Functions**

**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.

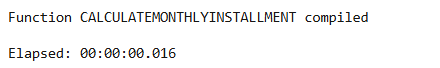
1. CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS
2. BEGIN
3. RETURN TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob)/12);
4. END;
5. /



**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.

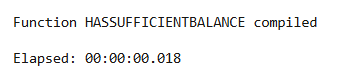
1. CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_amount NUMBER, p\_rate NUMBER, p\_years NUMBER)
2. RETURN NUMBER IS
3. v\_monthly\_rate NUMBER := p\_rate / 12 / 100;
4. v\_months NUMBER := p\_years \* 12;
5. BEGIN
6. RETURN (p\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -v\_months));
7. END;
8. /



**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.

1. CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_accid NUMBER, p\_amt NUMBER)
2. RETURN BOOLEAN IS
3. v\_balance NUMBER;
4. BEGIN
5. SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_accid;
6. RETURN v\_balance >= p\_amt;
7. END;
8. /

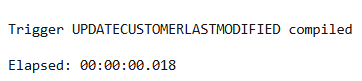


**Exercise 5: Triggers**

**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.

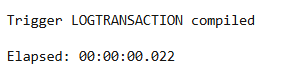
1. CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
2. BEFORE UPDATE ON Customers
3. FOR EACH ROW
4. BEGIN
5. :NEW.LastModified := SYSDATE;
6. END;
7. /



**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.

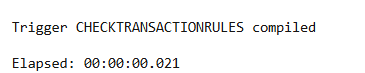
1. CREATE TABLE AuditLog (
2. LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
3. AccountID NUMBER,
4. TransactionDate DATE,
5. Amount NUMBER,
6. TransactionType VARCHAR2(10)
7. );
8. CREATE OR REPLACE TRIGGER LogTransaction
9. AFTER INSERT ON Transactions
10. FOR EACH ROW
11. BEGIN
12. INSERT INTO AuditLog(AccountID, TransactionDate, Amount, TransactionType)
13. VALUES(:NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType);
14. END;
15. /



**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.

1. CREATE OR REPLACE TRIGGER CheckTransactionRules
2. BEFORE INSERT ON Transactions
3. FOR EACH ROW
4. DECLARE
5. v\_balance NUMBER;
6. BEGIN
7. SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;
8. IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN
9. RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient funds');
10. ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN
11. RAISE\_APPLICATION\_ERROR(-20004, 'Deposit must be positive');
12. END IF;
13. END;
14. /

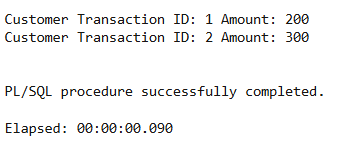


**Exercise 6: Cursors**

**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.

1. DECLARE
2. CURSOR c\_txn IS
3. SELECT \* FROM Transactions WHERE EXTRACT(MONTH FROM TransactionDate) = EXTRACT(MONTH FROM SYSDATE);
4. BEGIN
5. FOR rec IN c\_txn LOOP
6. DBMS\_OUTPUT.PUT\_LINE('Customer Transaction ID: ' || rec.TransactionID || ' Amount: ' || rec.Amount);
7. END LOOP;
8. END;
9. /



**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.

1. DECLARE
2. CURSOR c\_acct IS SELECT AccountID FROM Accounts;
3. BEGIN
4. FOR rec IN c\_acct LOOP
5. UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = rec.AccountID;
6. END LOOP;
7. END;
8. /



**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.

1. DECLARE
2. CURSOR c\_loan IS SELECT LoanID FROM Loans;
3. BEGIN
4. FOR rec IN c\_loan LOOP
5. UPDATE Loans SET InterestRate = InterestRate + 0.5 WHERE LoanID = rec.LoanID;
6. END LOOP;
7. END;
8. /

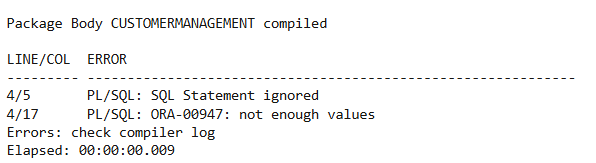


**Exercise 7: Packages**

**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.

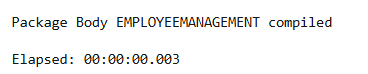
1. CREATE OR REPLACE PACKAGE CustomerManagement AS
2. PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_bal NUMBER);
3. PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2);
4. FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;
5. END;
6. /
7. CREATE OR REPLACE PACKAGE BODY CustomerManagement AS
8. PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_bal NUMBER) IS
9. BEGIN
10. INSERT INTO Customers VALUES(p\_id, p\_name, p\_dob, p\_bal, SYSDATE);
11. END;
12. PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2) IS
13. BEGIN
14. UPDATE Customers SET Name = p\_name WHERE CustomerID = p\_id;
15. END;
16. FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS
17. v\_bal NUMBER;
18. BEGIN
19. SELECT Balance INTO v\_bal FROM Customers WHERE CustomerID = p\_id;
20. RETURN v\_bal;
21. END;
22. END;
23. /



**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.

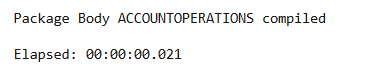
1. CREATE OR REPLACE PACKAGE EmployeeManagement AS
2. PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2);
3. PROCEDURE UpdateEmployee(p\_id NUMBER, p\_sal NUMBER);
4. FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;
5. END;
6. /
7. CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS
8. PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2) IS
9. BEGIN
10. INSERT INTO Employees VALUES(p\_id, p\_name, p\_pos, p\_sal, p\_dept, SYSDATE);
11. END;
12. PROCEDURE UpdateEmployee(p\_id NUMBER, p\_sal NUMBER) IS
13. BEGIN
14. UPDATE Employees SET Salary = p\_sal WHERE EmployeeID = p\_id;
15. END;
16. FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS
17. v\_sal NUMBER;
18. BEGIN
19. SELECT Salary INTO v\_sal FROM Employees WHERE EmployeeID = p\_id;
20. RETURN v\_sal \* 12;
21. END;
22. END;
23. /



**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.

1. CREATE OR REPLACE PACKAGE AccountOperations AS
2. PROCEDURE OpenAccount(p\_id NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_bal NUMBER);
3. PROCEDURE CloseAccount(p\_id NUMBER);
4. FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER;
5. END;
6. /
7. CREATE OR REPLACE PACKAGE BODY AccountOperations AS
8. PROCEDURE OpenAccount(p\_id NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_bal NUMBER) IS
9. BEGIN
10. INSERT INTO Accounts VALUES(p\_id, p\_custid, p\_type, p\_bal, SYSDATE);
11. END;
12. PROCEDURE CloseAccount(p\_id NUMBER) IS
13. BEGIN
14. DELETE FROM Accounts WHERE AccountID = p\_id;
15. END;
16. FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER IS
17. v\_total NUMBER;
18. BEGIN
19. SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_custid;
20. RETURN v\_total;
21. END;
22. END;
23. /



**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*

*);*

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

*INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1950-07-20', 'YYYY-MM-DD'), 15000, SYSDATE);*

*INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);*

*INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);*

*INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');*

*INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');*

*INSERT INTO Loans VALUES (1, 2, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 1));*

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

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

**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'));*