Exercise 1: Control Structures

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Sql:**

SET SERVEROUTPUT ON;

BEGIN

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

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

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to CustomerID ' || rec.CustomerID);

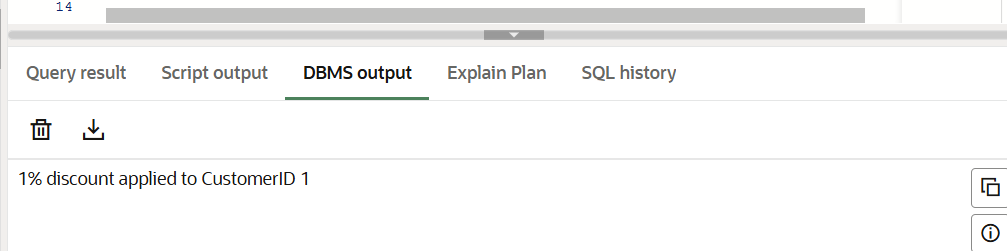
END IF;

END LOOP;

END;

/

**OUTPUT:**



**Scenario 2:** A customer can be promoted to VIP status based on their balance.

**Sql:**

SET SERVEROUTPUT ON;

BEGIN

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

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('CustomerID ' || rec.CustomerID || ' marked as VIP.');

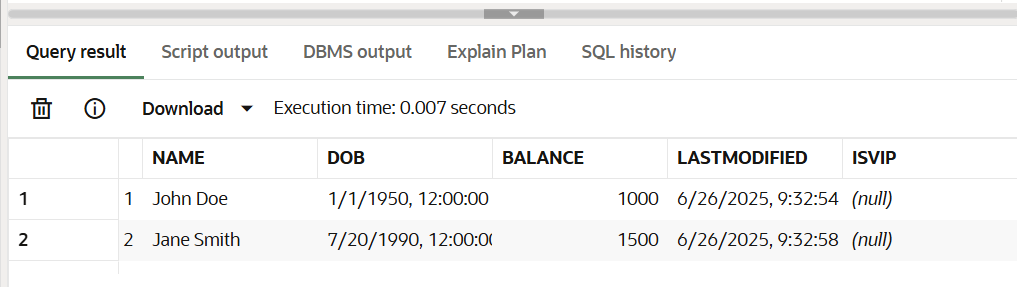
END IF;

END LOOP;

END;

/

**OUTPUT:**

****

**Scenario 3**: The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Sql:**

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (

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

FROM Loans l

JOIN Customers c ON c.CustomerID = l.CustomerID

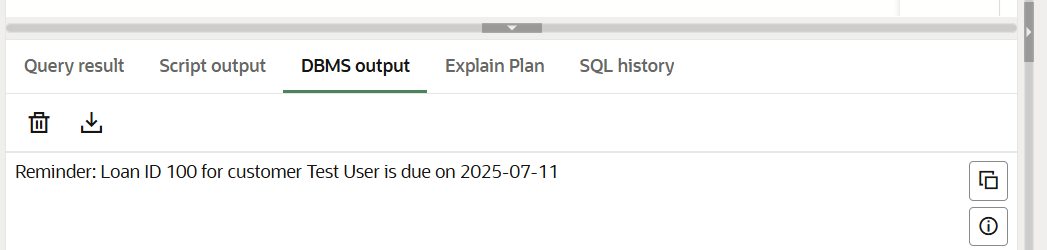
WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

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

END LOOP;

END;

**OUTPUT:** 

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

**SQL:**

BEGIN

-- Get balance of source account

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_acc\_id

FOR UPDATE

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in source account.');

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_acc\_id;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_acc\_id;

DBMS\_OUTPUT.PUT\_LINE('₹' || p\_amount || ' transferred from Account ' || p\_from\_acc\_id || ' to ' || p\_to\_acc\_id);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

-- Optional: Log to ErrorLog table

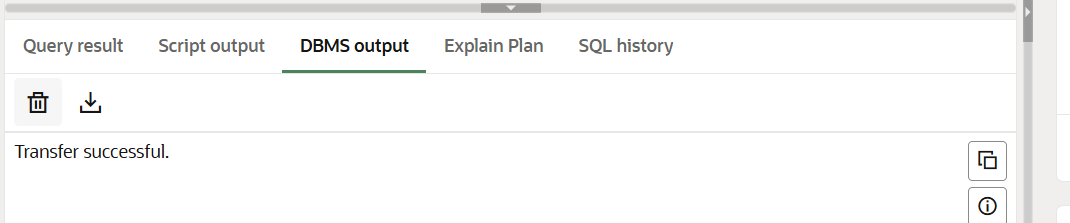
INSERT INTO ErrorLog(ErrorMessage)

VALUES(SQLERRM);

COMMIT; -- commit the log entry

**EXEC SafeTransferFunds(1, 2, 200);**

**OUTPUT:**



**Scenario 2:** Manage errors when updating employee salaries.

**SQL:**

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee ID not found: ' || p\_emp\_id);

END IF;

DBMS\_OUTPUT.PUT\_LINE('Salary updated for Employee ID ' || p\_emp\_id);

EXCEPTION

WHEN OTHERS THEN

v\_error\_message := SUBSTR(SQLERRM, 1, 400);

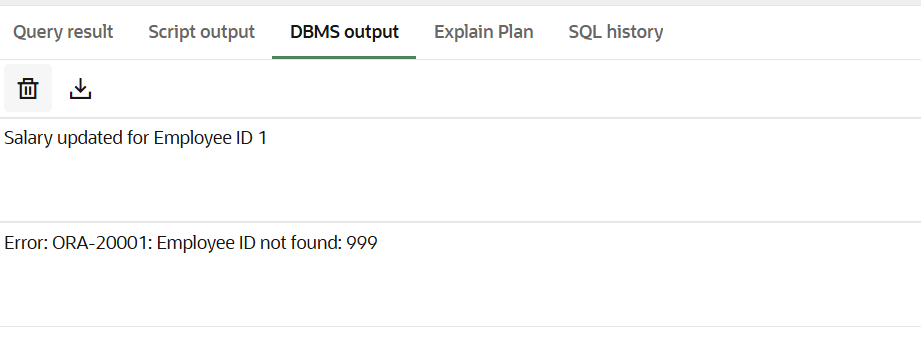
DBMS\_OUTPUT.PUT\_LINE('Error: ' || v\_error\_message);

INSERT INTO ErrorLog (ErrorMessage) VALUES (v\_error\_message);

COMMIT;

END;

/ **OUTPUT:**



**Scenario 3:** Ensure data integrity when adding a new customer.

**SQL:**

BEGIN

-- Try to insert the customer

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

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

v\_error\_message := 'Duplicate CustomerID: ' || p\_customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || v\_error\_message);

INSERT INTO ErrorLog (ErrorMessage) VALUES (v\_error\_message);

COMMIT;

WHEN OTHERS THEN

v\_error\_message := SUBSTR(SQLERRM, 1, 400);

DBMS\_OUTPUT.PUT\_LINE('Unexpected Error: ' || v\_error\_message);

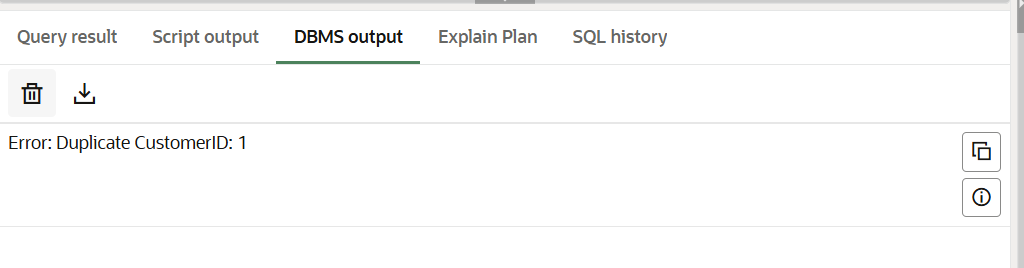
INSERT INTO ErrorLog (ErrorMessage) VALUES (v\_error\_message);

COMMIT;

END;

/

OUTPUT:



**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.

**SQL:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountType = 'Savings';

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

**OUTPUT:**  
A screenshot of a computer

AI-generated content may be incorrect.

**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.

**SQL:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

END;

/

BEGIN

UpdateEmployeeBonus('HR', 10); -- Gives 10% bonus to HR department

END;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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.

**SQL:**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_source\_account IN NUMBER,

p\_target\_account IN NUMBER,

p\_amount IN NUMBER

) IS

source\_balance NUMBER;

BEGIN

-- Check source balance

SELECT Balance INTO source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account;

IF source\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_source\_account;

-- Add to target

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_target\_account;

-- Log transactions (optional)

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

VALUES (Transactions\_SEQ.NEXTVAL, p\_source\_account, SYSDATE, p\_amount, 'Transfer Out');

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

VALUES (Transactions\_SEQ.NEXTVAL, p\_target\_account, SYSDATE, p\_amount, 'Transfer In');

END;

/

BEGIN

TransferFunds(1, 2, 200); -- Transfer 200 from Account 1 to 2

END;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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.

**SQL:**

CREATE OR REPLACE FUNCTION CalculateAge(dob DATE)

RETURN NUMBER IS

age NUMBER;

BEGIN

age := FLOOR(MONTHS\_BETWEEN(SYSDATE, dob) / 12);

RETURN age;

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Age: ' || CalculateAge(TO\_DATE('1990-07-20', 'YYYY-MM-DD')));

END;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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.

**SQL:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

loan\_amount NUMBER,

annual\_interest\_rate NUMBER,

loan\_duration\_years NUMBER

)

RETURN NUMBER IS

monthly\_rate NUMBER;

total\_months NUMBER;

emi NUMBER;

BEGIN

monthly\_rate := annual\_interest\_rate / 12 / 100;

total\_months := loan\_duration\_years \* 12;

IF monthly\_rate = 0 THEN

emi := loan\_amount / total\_months;

ELSE

emi := loan\_amount \* monthly\_rate \* POWER(1 + monthly\_rate, total\_months) /

(POWER(1 + monthly\_rate, total\_months) - 1);

END IF;

RETURN ROUND(emi, 2);

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Monthly EMI: ' || CalculateMonthlyInstallment(50000, 7.5, 5));

END;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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.

**SQL:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_account\_id NUMBER,

p\_amount NUMBER

)

RETURN NUMBER IS

available\_balance NUMBER;

BEGIN

SELECT Balance INTO available\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

IF available\_balance >= p\_amount THEN

RETURN 1;

ELSE

RETURN 0;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

/

DECLARE

result NUMBER;

BEGIN

result := HasSufficientBalance(1, 500);

IF result = 1 THEN

DBMS\_OUTPUT.PUT\_LINE('Sufficient balance.');

ELSE

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

END IF;

END;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 5: Triggers**

**Scenario 1:** Automatically update the last modified date when a customer's record is updated.

**o 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.

**SQL:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

UPDATE Customers SET Name = 'John A. Doe' WHERE CustomerID = 1;

SELECT Name, LastModified FROM Customers WHERE CustomerID = 1;

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 2:** Maintain an audit log for all transactions.

o **Question:** Write a trigger LogTransaction that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

**SQL:**

CREATE TABLE AuditLog (

AuditID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

Amount NUMBER,

TransactionType VARCHAR2(10),

LoggedAt DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, Amount, TransactionType, LoggedAt)

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

END;

/

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

VALUES (3, 1, SYSDATE, 100, 'Deposit');

SELECT \* FROM AuditLog;

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 3:** Enforce business rules on deposits and withdrawals.

**o 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.

**SQL:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

account\_balance NUMBER;

BEGIN

-- Get current balance

SELECT Balance INTO account\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

-- Rule 1: Deposit must be positive

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

RAISE\_APPLICATION\_ERROR(-20001, 'Deposit amount must be positive.');

END IF;

-- Rule 2: Withdrawal must not exceed balance

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

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal.');

END IF;

END;

/

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

VALUES (4, 1, SYSDATE, 50, 'Withdrawal');

-- This will throw an error

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

VALUES (5, 1, SYSDATE, -200, 'Deposit');

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

**SQL:**

SET SERVEROUTPUT ON;

DECLARE

    CURSOR txn\_cursor IS

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

        FROM Transactions t

        JOIN Accounts a ON a.AccountID = t.AccountID

        JOIN Customers c ON c.CustomerID = a.CustomerID

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

    v\_name Customers.Name%TYPE;

    v\_acct\_id Accounts.AccountID%TYPE;

    v\_date DATE;

    v\_amt NUMBER;

    v\_type VARCHAR2(10);

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('--- Monthly Statements ---');

    OPEN txn\_cursor;

    LOOP

        FETCH txn\_cursor INTO v\_name, v\_acct\_id, v\_date, v\_amt, v\_type;

        EXIT WHEN txn\_cursor%NOTFOUND;

        DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_name || ', Account: ' || v\_acct\_id || ', Date: ' || TO\_CHAR(v\_date, 'YYYY-MM-DD') || ', Amount: ' || v\_amt || ', Type: ' || v\_type);

    END LOOP;

    CLOSE txn\_cursor;

END;

/

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Scenario 2:** Apply annual fee to all accounts

**SQL:**

SET SERVEROUTPUT ON;

DECLARE

    CURSOR acc\_cursor IS

        SELECT AccountID, Balance FROM Accounts;

    v\_acc\_id Accounts.AccountID%TYPE;

    v\_balance NUMBER;

    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,

            LastModified = SYSDATE

        WHERE AccountID = v\_acc\_id;

        DBMS\_OUTPUT.PUT\_LINE('Annual fee of ₹' || v\_fee || ' deducted from Account ' || v\_acc\_id);

    END LOOP;

    CLOSE acc\_cursor;

END;

/

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Scenario 3:** Update the interest rate for all loans based on a new policy.

**SQL:**

SET SERVEROUTPUT ON;

DECLARE

    CURSOR loan\_cursor IS

        SELECT LoanID, InterestRate FROM Loans;

    v\_loan\_id Loans.LoanID%TYPE;

    v\_rate Loans.InterestRate%TYPE;

BEGIN

    OPEN loan\_cursor;

    LOOP

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

        EXIT WHEN loan\_cursor%NOTFOUND;

        IF v\_rate < 6 THEN

            UPDATE Loans SET InterestRate = InterestRate + 0.5 WHERE LoanID = v\_loan\_id;

            DBMS\_OUTPUT.PUT\_LINE('Loan ' || v\_loan\_id || ': Interest increased to ' || (v\_rate + 0.5));

        ELSE

            UPDATE Loans SET InterestRate = InterestRate - 0.25 WHERE LoanID = v\_loan\_id;

            DBMS\_OUTPUT.PUT\_LINE('Loan ' || v\_loan\_id || ': Interest reduced to ' || (v\_rate - 0.25));

        END IF;

    END LOOP;

    CLOSE loan\_cursor;

END;

/

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

**SQL:**

SET SERVEROUTPUT ON;

DECLARE

    CURSOR txn\_cursor IS

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

        FROM Transactions t

        JOIN Accounts a ON a.AccountID = t.AccountID

        JOIN Customers c ON c.CustomerID = a.CustomerID

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

    v\_name Customers.Name%TYPE;

    v\_acct\_id Accounts.AccountID%TYPE;

    v\_date DATE;

    v\_amt NUMBER;

    v\_type VARCHAR2(10);

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('--- Monthly Statements ---');

    OPEN txn\_cursor;

    LOOP

        FETCH txn\_cursor INTO v\_name, v\_acct\_id, v\_date, v\_amt, v\_type;

        EXIT WHEN txn\_cursor%NOTFOUND;

        DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_name || ', Account: ' || v\_acct\_id || ', Date: ' || TO\_CHAR(v\_date, 'YYYY-MM-DD') || ', Amount: ' || v\_amt || ', Type: ' || v\_type);

    END LOOP;

    CLOSE txn\_cursor;

END;

/

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Scenario 2:** Create a package to manage employee data

**SQL:**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

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

    BEGIN

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

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

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

    END;

    PROCEDURE UpdateEmployee(p\_id NUMBER, p\_position VARCHAR2, p\_salary NUMBER) IS

    BEGIN

        UPDATE Employees

        SET Position = p\_position, Salary = p\_salary

        WHERE EmployeeID = p\_id;

        DBMS\_OUTPUT.PUT\_LINE('Employee updated: ' || 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;

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

            RETURN -1;

    END;

END EmployeeManagement;

/

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Scenario 3:** Group all account-related operations into a package.

**SQL:**

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

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

    BEGIN

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

        VALUES (p\_acc\_id, p\_cust\_id, p\_type, p\_balance, SYSDATE);

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

    END;

    PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS

    BEGIN

        DELETE FROM Accounts WHERE AccountID = p\_acc\_id;

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

    END;

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

        v\_total NUMBER;

    BEGIN

        SELECT NVL(SUM(Balance), 0) INTO v\_total FROM Accounts WHERE CustomerID = p\_cust\_id;

        RETURN v\_total;

    END;

END AccountOperations;

/

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**