**Exercise 1: Control Structures**

**Code:**

**Applying 1% Interest Discount for Customers Above 60:**

BEGIN

FOR rec IN (SELECT c.CustomerID, c.Name, l.LoanID, l.InterestRate, c.DOB

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID)

LOOP

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

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Applied discount to ' || rec.Name || '''s loan.');

END IF;

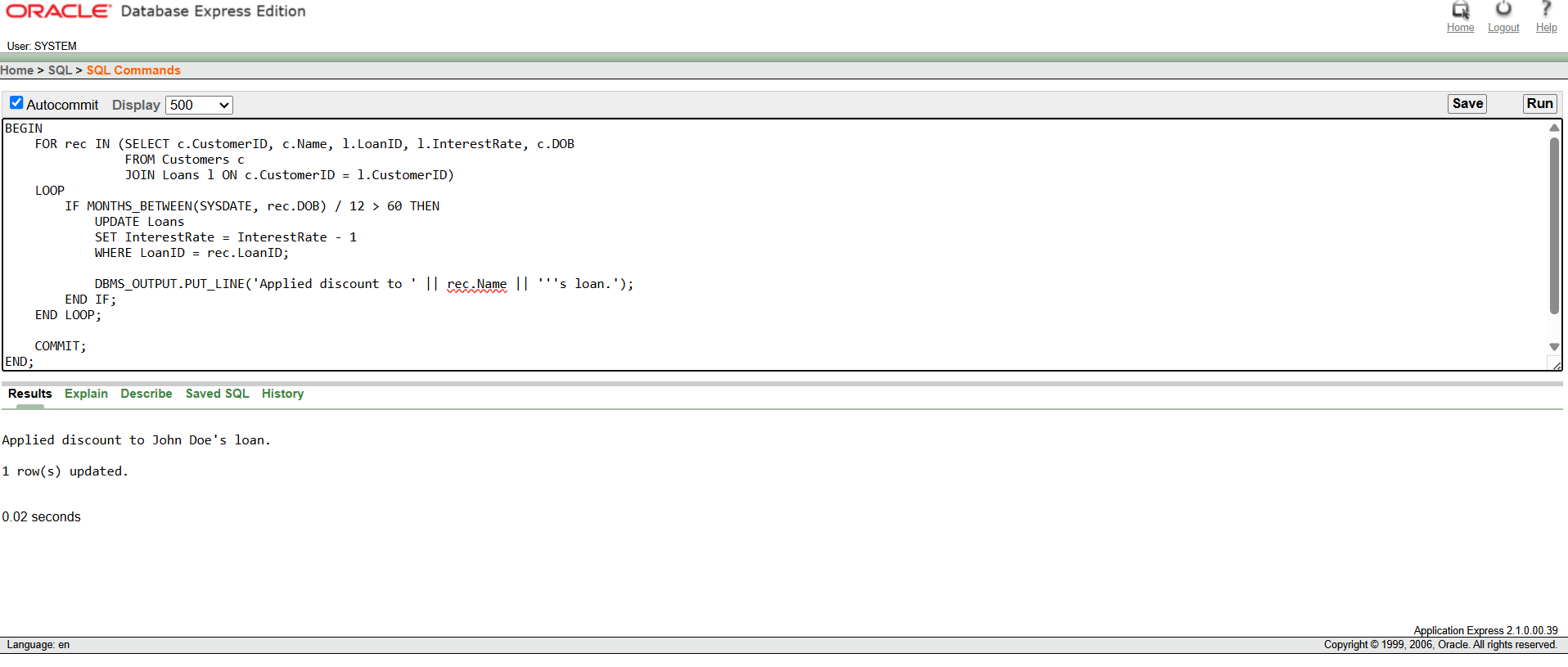
END LOOP;

COMMIT;

END;

/

**Output:**

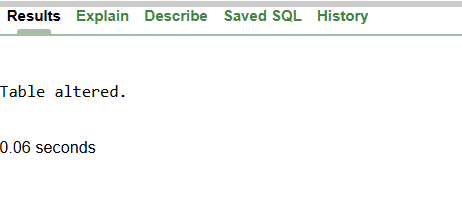
****

Promoting Customers with Balance > $10,000 to VIP:

**Code:**

**ALTER TABLE Customers ADD IsVIP VARCHAR2(5);**

**Output:**

****

**Code:**

BEGIN

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

LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Promoted ' || rec.Name || ' to VIP.');

END IF;

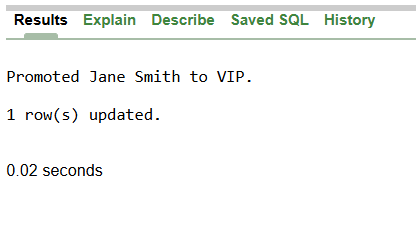
END LOOP;

COMMIT;

END;

/

**Output:**

****

**Reminding Customers with Loans Due in 30 Days**

**Code:**

BEGIN

FOR rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

)

LOOP

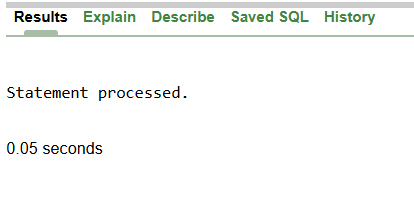
DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || rec.Name || ', your loan (ID: ' || rec.LoanID || ') is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Output:**

****

**Exercise-2: Error Handling**

**Safe Fund Transfer Between Accounts**

**Code:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_from\_balance NUMBER;

BEGIN

-- Check balance of source account

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

IF v\_from\_balance < p\_amount THEN

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

END IF;

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

-- Add to destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

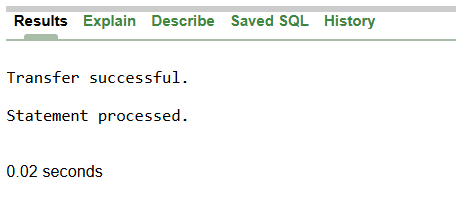
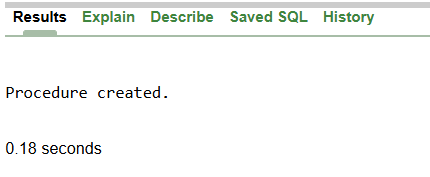
BEGIN

SafeTransferFunds(1, 2, 500);

END;

/

**Output:**



**Update Employee Salary with Error Logging**

**Code:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_emp\_id IN NUMBER,

p\_percent IN 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;

DBMS\_OUTPUT.PUT\_LINE('Salary updated.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

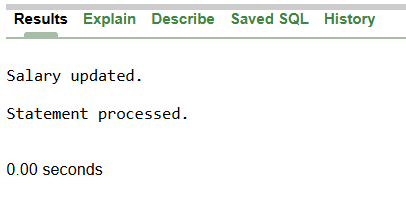
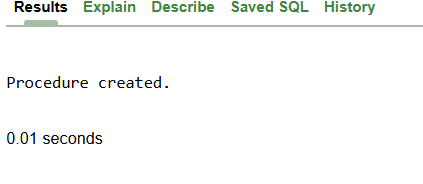
BEGIN

UpdateSalary(2, 10);

END;

/

**Output:**

****

**Adding New Customer with Duplicate ID Handling:**

**Code:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

)

IS

BEGIN

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID already exists.');

WHEN OTHERS THEN

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

END;

/

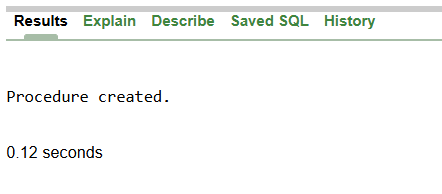
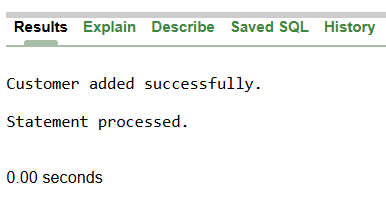
BEGIN

AddNewCustomer(3, 'Charlie Chaplin', TO\_DATE('1970-01-01', 'YYYY-MM-DD'), 3000);

END;

/

**Output:**

**Exercise-3: Stored Procedures**

**Processing Monthly Interest for Savings Accounts**

**Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

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

UPDATE Accounts

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

WHERE AccountID = rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID ' || rec.AccountID);

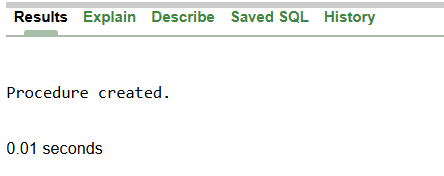
END LOOP;

COMMIT;

END;

/

**Output:**

****

**Bonus Scheme Based on Department:**

**Code:**

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;

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' employee(s) updated with bonus.');

COMMIT;

END;

/

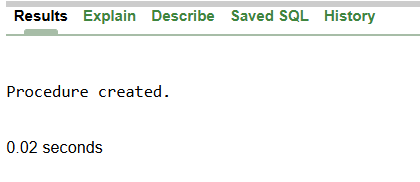
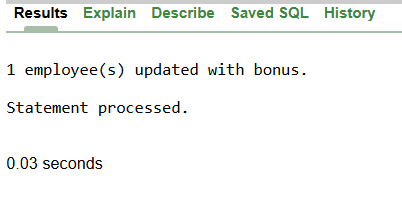
BEGIN

UpdateEmployeeBonus('IT', 15);

END;

/

**Output:**

**** ****

**Transfer Funds Between Accounts:**

**Code:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

IS

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(-20003, 'Insufficient balance.');

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;

DBMS\_OUTPUT.PUT\_LINE('Transfer complete.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

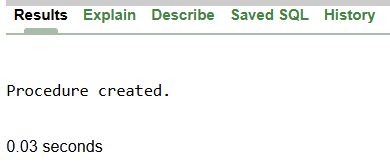
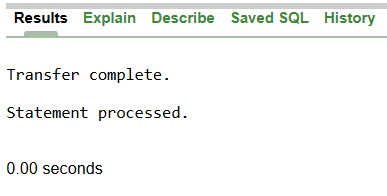
BEGIN

TransferFunds(1, 2, 300);

END;

/

**Output:**

**** ****

**Exercise-4: Functions**

**Calculate Age from DOB**

**Code:**

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;

/

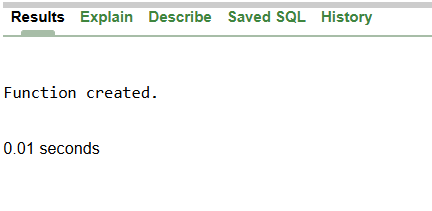
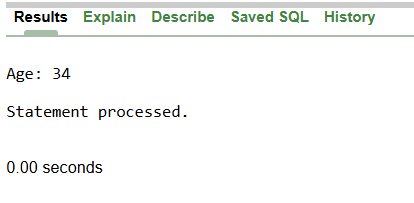
BEGIN

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

END;

/

**Output:**

**** ****

**Calculate Monthly Installment:**

Formula used:  
EMI = P × r × (1 + r)^n / ((1 + r)^n - 1)  
Where:

P = loan amount

r = monthly interest rate

n = total number of months

**Code:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount IN NUMBER,

p\_annual\_rate IN NUMBER,

p\_duration\_years IN NUMBER

) RETURN NUMBER

IS

v\_monthly\_rate NUMBER := p\_annual\_rate / 12 / 100;

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

v\_emi NUMBER;

BEGIN

v\_emi := (p\_loan\_amount \* v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_months)) /

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

RETURN ROUND(v\_emi, 2);

END;

/

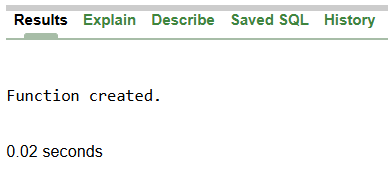
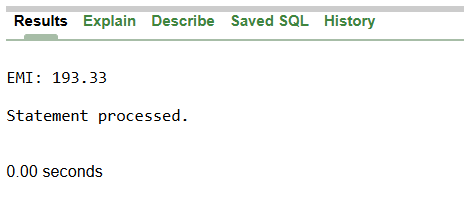
BEGIN

DBMS\_OUTPUT.PUT\_LINE('EMI: ' || CalculateMonthlyInstallment(10000, 6, 5));

END;

/

**Output:**

**** ****

**Check Sufficient Balance:**

**Code:**

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;

/

DECLARE

result BOOLEAN;

BEGIN

result := HasSufficientBalance(1, 500);

IF result THEN

DBMS\_OUTPUT.PUT\_LINE('Balance is sufficient.');

ELSE

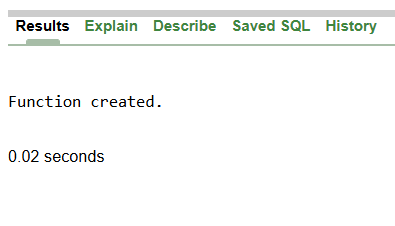
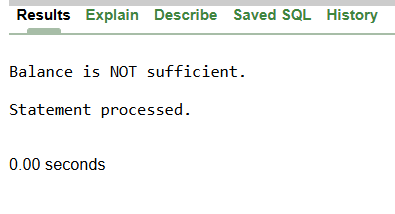
DBMS\_OUTPUT.PUT\_LINE('Balance is NOT sufficient.');

END IF;

END;

/

**Output:**

**** ****

**Exercise-5: Triggers**

**Auto-update LastModified on Customer Record Update**

**Code:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

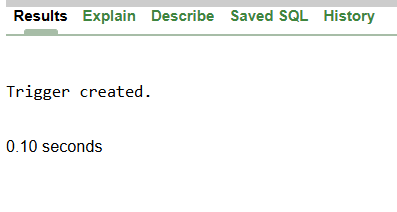
BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Output:**

****

**Audit Log for Transactions:**

**Code:**

CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

AccountID NUMBER,

Amount NUMBER,

TransactionType VARCHAR2(10),

LoggedAt DATE

);

CREATE SEQUENCE AuditLogSeq

START WITH 1

INCREMENT BY 1;

CREATE OR REPLACE TRIGGER LogTransaction

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

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

VALUES (

AuditLogSeq.NEXTVAL,

:NEW.AccountID,

:NEW.Amount,

:NEW.TransactionType,

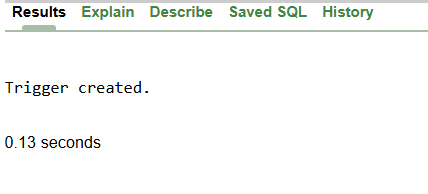
SYSDATE

);

END;

/

**Output:**

****

**Enforce Transaction Rules:**

**Code:**

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, 'Withdrawal exceeds available balance.');

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

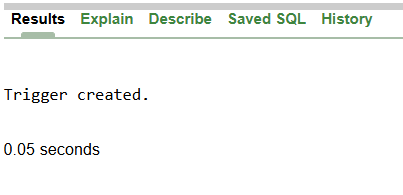
RAISE\_APPLICATION\_ERROR(-20005, 'Deposit amount must be positive.');

END IF;

END;

/

**Output:**

****

**Exercise-6: Cursors**

**Generate Monthly Statements**

**Code:**

DECLARE

CURSOR txn\_cursor IS

SELECT c.Name, a.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');

v\_name Customers.Name%TYPE;

v\_acc\_id Accounts.AccountID%TYPE;

v\_date DATE;

v\_amount NUMBER;

v\_type VARCHAR2(10);

BEGIN

OPEN txn\_cursor;

LOOP

FETCH txn\_cursor INTO v\_name, v\_acc\_id, v\_date, v\_amount, v\_type;

EXIT WHEN txn\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_name ||

' | Account ID: ' || v\_acc\_id ||

' | Date: ' || TO\_CHAR(v\_date, 'DD-MON-YYYY') ||

' | ' || v\_type || ': ' || v\_amount);

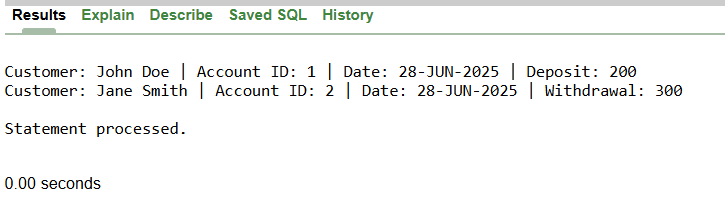
END LOOP;

CLOSE txn\_cursor;

END;

/

**Output:**

****

**Apply Annual Fee:**

**Code:**

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_id Accounts.AccountID%TYPE;

v\_bal Accounts.Balance%TYPE;

v\_fee NUMBER := 100; -- fixed annual fee

BEGIN

OPEN acc\_cursor;

LOOP

FETCH acc\_cursor INTO v\_id, v\_bal;

EXIT WHEN acc\_cursor%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_fee

WHERE AccountID = v\_id;

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

END LOOP;

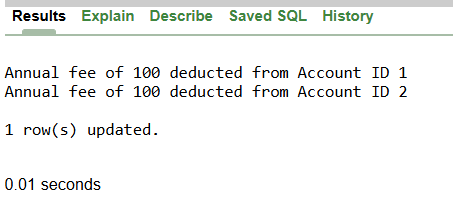
CLOSE acc\_cursor;

COMMIT;

END;

/

**Output:**

****

**Update Loan Interest Rates(Policy Change):**

**Code:**

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;

UPDATE Loans

SET InterestRate = v\_rate + 0.5

WHERE LoanID = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated interest rate for Loan ID ' || v\_loan\_id);

END LOOP;

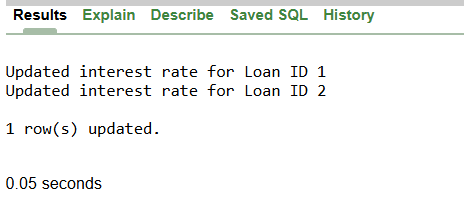
CLOSE loan\_cursor;

COMMIT;

END;

/

**Output:**

****

**Exercise-7: Packages**

**Package for Customer Management**

**Code:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

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;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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;

/

BEGIN

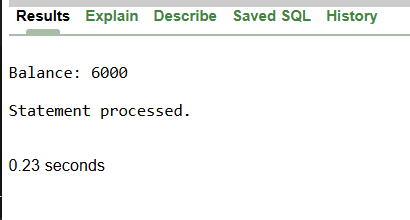
CustomerManagement.AddCustomer(4, 'Arnold Watt', TO\_DATE('1999-02-18', 'YYYY-MM-DD'), 6000);

DBMS\_OUTPUT.PUT\_LINE('Balance: ' || CustomerManagement.GetCustomerBalance(4));

END;

/

**Output:**

****

**Employee Management Package**

**Code:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_hire\_date DATE);

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

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

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

BEGIN

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

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

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;

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;

/

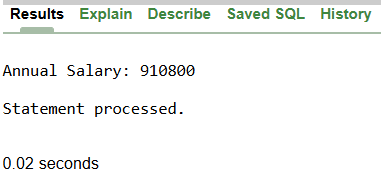
BEGIN

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.GetAnnualSalary(2));

END;

/

**Output:**

****

**Account Operations Package**

**Code:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_id NUMBER, p\_customer\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_id NUMBER);

FUNCTION GetTotalCustomerBalance(p\_customer\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

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

BEGIN

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

VALUES (p\_id, p\_customer\_id, p\_type, p\_balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_id;

END;

FUNCTION GetTotalCustomerBalance(p\_customer\_id NUMBER) RETURN NUMBER IS

v\_total NUMBER;

BEGIN

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

RETURN v\_total;

END;

END AccountOperations;

/

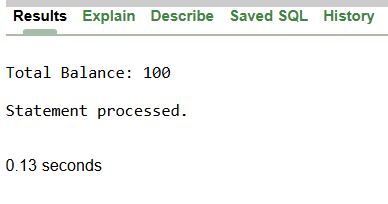
BEGIN

DBMS\_OUTPUT.PUT\_LINE('Total Balance: ' || AccountOperations.GetTotalCustomerBalance(1));

END;

/

**Output:**

****