WEEK -2 JAVA FSE PL/SQL EXERCISES---------

EXERCISE -1

PROGRAM--1

```
 Table: Customers
 Columns: CustomerID, Age, LoanInterestRate
BEGIN
  FOR cust_rec IN (SELECT CustomerID, Age, LoanInterestRate FROM Customers) LOOP
    IF cust_rec.Age > 60 THEN
      UPDATE Customers
      SET LoanInterestRate = LoanInterestRate - 1
      WHERE CustomerID = cust_rec.CustomerID;
    END IF;
  END LOOP;
  COMMIT;
END;
/
```

PROGRAM -2

- Table: Customers
- Columns: CustomerID, Balance, IsVIP

sql

CopyEdit

```
BEGIN
  FOR cust_rec IN (SELECT CustomerID, Balance FROM Customers) LOOP
    IF cust_rec.Balance > 10000 THEN
      UPDATE Customers
      SET IsVIP = 'TRUE'
      WHERE CustomerID = cust_rec.CustomerID;
    END IF;
  END LOOP;
  COMMIT;
```

```
END;
/
```

PROGRAM-3

- Table: Loans
- Columns: LoanID, CustomerID, DueDate
- Table: Customers
- Columns: CustomerID, CustomerName

sql

CopyEdit

```
DECLARE
 v_today DATE := SYSDATE;
BEGIN
 FOR loan_rec IN (
  SELECT l.LoanID, c.CustomerName, l.DueDate
  FROM Loans l
  JOIN Customers c ON l.CustomerID = c.CustomerID
  WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30
 ) LOOP
  DBMS_OUTPUT.PUT_LINE('Reminder: Loan ' || loan_rec.LoanID ||
             ' for customer ' || loan_rec.CustomerName ||
             ' is due on ' || TO_CHAR(loan_rec.DueDate, 'DD-MON-YYYY'));
 END LOOP;
END;
/
```

OUTPUT-----

```
Reminder: Loan 101 for customer John Doe is due on 15-JUL-2025
Reminder: Loan 205 for customer Anita Reddy is due on 28-JUN-2025
Reminder: Loan 309 for customer Ravi Kumar is due on 10-JUL-2025
```

EXERCISE -2

PROGRAM-1

- Table: Accounts (AccountID, Balance)
- Table: Error_Log (ErrorTime TIMESTAMP, ErrorMessage VARCHAR2(4000))

sql

CopyEdit

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds(
  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(-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;
    INSERT INTO Error_Log(ErrorTime, ErrorMessage)
    VALUES (SYSTIMESTAMP, 'Transfer failed: ' || SQLERRM);
END;
/
```

PROGRAM-2

- Table: Employees (EmployeeID, Salary)
- Table: Error_Log

sql

CopyEdit

```
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 ID not found.');
  END IF;

  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    INSERT INTO Error_Log(ErrorTime, ErrorMessage)
    VALUES (SYSTIMESTAMP, 'Salary update failed: ' || SQLERRM);
END;
```

/

PROGRAM-3

- Table: Customers (CustomerID, CustomerName, Age, Balance)
- Table: Error_Log

sql

CopyEdit

```
CREATE OR REPLACE PROCEDURE AddNewCustomer(
  p_customer_id IN NUMBER,
  p_name IN VARCHAR2,
  p_age IN NUMBER,
  p_balance IN NUMBER
)
IS
BEGIN
  INSERT INTO Customers(CustomerID, CustomerName, Age, Balance)
  VALUES (p_customer_id, p_name, p_age, p_balance);

  COMMIT;
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    ROLLBACK;
    INSERT INTO Error_Log(ErrorTime, ErrorMessage)
    VALUES (SYSTIMESTAMP, 'Customer insert failed: Duplicate ID.');
  WHEN OTHERS THEN
    ROLLBACK;
    INSERT INTO Error_Log(ErrorTime, ErrorMessage)
    VALUES (SYSTIMESTAMP, 'Customer insert failed: ' || SQLERRM);
END;
/
```

OUTPUT----

```
ErrorTime: 28-JUN-2025 13:47:10
```

ErrorMessage: Salary update failed: ORA-20002: Employee ID not found.

EXERCISE -3

PROGRAM-1

- Table: Accounts (AccountID, AccountType, Balance)
- Only accounts with AccountType = 'SAVINGS' are eligible.
- Interest Rate: 1%

sql

CopyEdit

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest
IS
BEGIN
  FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'SAVINGS') LOOP
    UPDATE Accounts
    SET Balance = Balance + (acc.Balance * 0.01)
    WHERE AccountID = acc.AccountID;
  END LOOP;
  COMMIT;
END;
/
```

PROGRAM-2

- Table: Employees (EmployeeID, DepartmentID, Salary)
- Bonus applied as a percentage to all employees in a specified department

sql

CopyEdit

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(
  p_department_id IN NUMBER,
  p_bonus_percent IN NUMBER
)
IS
BEGIN
  UPDATE Employees
  SET Salary = Salary + (Salary * p_bonus_percent / 100)
  WHERE DepartmentID = p_department_id;

  COMMIT;
END;
/
```

PROGRAM-3

- Table: Accounts (AccountID, Balance)
- Must validate that the from_account has sufficient balance.

sql

CopyEdit

```
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(-20001, '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('Error during transfer: ' || SQLERRM);
END;
/
```

OUTPUT----

AccountID	**AccountType**	**Balance**
101	SAVINGS	10,000
102	SAVINGS	5,000
103	CURRENT	12,000

AccountID	AccountType	Balance
101	SAVINGS	10,100
102	SAVINGS	5,050
103	CURRENT	12,000

EmployeeID	DepartmentID	Salary
201	5	44,000
202	5	55,000
203	6	60,000

AccountID	**Balance**
101	8,000

AccountID	AccountType	Balance
202	10,000	

EXERCISE -4

PROGRAM-1

```
CREATE OR REPLACE FUNCTION CalculateAge(p_dob DATE)
RETURN NUMBER
IS
  v_age NUMBER;
BEGIN
  v_age := TRUNC(MONTHS_BETWEEN(SYSDATE, p_dob) / 12);
  RETURN v_age;
END;
/

SELECT CalculateAge(DATE '2000-01-15') AS Age FROM DUAL;
```

O/P-

```
Age
---
25
```

PROGRAM-2

```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(
  p_loan_amount IN NUMBER,
  p_annual_rate IN NUMBER,
  p_years IN NUMBER
)
RETURN NUMBER
IS
  v_monthly_rate NUMBER := p_annual_rate / (12 * 100);
  v_months      NUMBER := p_years * 12;
  v_emi         NUMBER;
BEGIN
  IF v_monthly_rate = 0 THEN
    v_emi := p_loan_amount / v_months;
  ELSE
    v_emi := (p_loan_amount * v_monthly_rate * POWER(1 + v_monthly_rate, v_months)) /
             (POWER(1 + v_monthly_rate, v_months) - 1);
  END IF;

  RETURN ROUND(v_emi, 2);
END;
/

SELECT CalculateMonthlyInstallment(100000, 12, 5) AS EMI FROM DUAL;
```

O/P—

```
EMI
-----
2224.44
```

PROGRAM-3

```
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;
  WHEN OTHERS THEN
    RETURN FALSE;
END;
/

DECLARE
  result BOOLEAN;
BEGIN
  result := HasSufficientBalance(101, 5000);
  IF result THEN
    DBMS_OUTPUT.PUT_LINE('Sufficient Balance');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Insufficient Balance');
  END IF;
END;
/
```

EXERCISE -5

PROGRAM-1

- Table: Customers (CustomerID, Name, Balance, LastModified DATE)

sql

CopyEdit

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON Customers
```

```
FOR EACH ROW
BEGIN
  :NEW.LastModified := SYSDATE;
END;
/
```

PROGRAM-2

- Table: Transactions (TransactionID, AccountID, Amount, Type, TransactionDate)
- Audit Table: AuditLog (LogID, TransactionID, AccountID, ActionType, LogDate)

sql

CopyEdit

```
CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
  INSERT INTO AuditLog (LogID, TransactionID, AccountID, ActionType, LogDate)
  VALUES (AuditLog_seq.NEXTVAL, :NEW.TransactionID, :NEW.AccountID, 'INSERT', SYSDATE);
END;
/
```

PROGRAM-3

- Table: Accounts(AccountID, Balance)

sql

CopyEdit

```
CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
  v_balance NUMBER;
BEGIN
  -- Get current account balance
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = :NEW.AccountID;
```

```
 -- Withdrawal rule
 IF :NEW.Type = 'WITHDRAWAL' AND :NEW.Amount > v_balance THEN
   RAISE_APPLICATION_ERROR(-20001, 'Withdrawal amount exceeds available balance.');
 END IF;

 -- Deposit rule
 IF :NEW.Type = 'DEPOSIT' AND :NEW.Amount <= 0 THEN
   RAISE_APPLICATION_ERROR(-20002, 'Deposit amount must be positive.');
 END IF;
END;
/
```

OUTPUT---

CustomerID	**Name**	**LastModified**
101	Anil K.	28-JUN-2025 13:10

EXERCISE -6

PROGRAM-1

- Table: Transactions (TransactionID, CustomerID, Amount, Type, TransactionDate)
- Table: Customers (CustomerID, CustomerName)

sql

CopyEdit

```
DECLARE
 CURSOR txn_cursor IS
   SELECT c.CustomerID, c.CustomerName, t.TransactionID, t.Amount, t.Type, t.TransactionDate
   FROM Customers c
   JOIN Transactions t ON c.CustomerID = t.CustomerID
   WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');
 v_cust_id Customers.CustomerID%TYPE;
 v_name Customers.CustomerName%TYPE;
 v_txn_id Transactions.TransactionID%TYPE;
 v_amount Transactions.Amount%TYPE;
 v_type Transactions.Type%TYPE;
 v_date Transactions.TransactionDate%TYPE;
BEGIN
 OPEN txn_cursor;
 LOOP
  FETCH txn_cursor INTO v_cust_id, v_name, v_txn_id, v_amount, v_type, v_date;
  EXIT WHEN txn_cursor%NOTFOUND;

  DBMS_OUTPUT.PUT_LINE('Customer: ' || v_name || ' | Transaction ID: ' || v_txn_id ||
             ' | Type: ' || v_type || ' | Amount: ' || v_amount ||
             ' | Date: ' || TO_CHAR(v_date, 'DD-MON-YYYY'));
 END LOOP;
 CLOSE txn_cursor;
END;
/
```

PROGRAM-2

- Table: Accounts (AccountID, Balance)
- Fee: ₹250 per account

sql

CopyEdit

```
DECLARE
 CURSOR account_cursor IS
  SELECT AccountID, Balance FROM Accounts;

 v_acc_id Accounts.AccountID%TYPE;
 v_balance Accounts.Balance%TYPE;
 v_fee CONSTANT NUMBER := 250;
BEGIN
 OPEN account_cursor;
 LOOP
  FETCH account_cursor INTO v_acc_id, v_balance;
  EXIT WHEN account_cursor%NOTFOUND;

  UPDATE Accounts
  SET Balance = Balance - v_fee
  WHERE AccountID = v_acc_id;

  DBMS_OUTPUT.PUT_LINE('Annual fee applied to Account ' || v_acc_id ||
             '. New Balance will be updated.');
 END LOOP;
 CLOSE account_cursor;

 COMMIT;
END;
/
```

PROGRAM-3

```
DECLARE
 CURSOR loan_cursor IS
  SELECT LoanID, LoanType, InterestRate FROM Loans;

 v_loan_id Loans.LoanID%TYPE;
 v_type Loans.LoanType%TYPE;
 v_rate Loans.InterestRate%TYPE;
BEGIN
```

```
 OPEN loan_cursor;
 LOOP
  FETCH loan_cursor INTO v_loan_id, v_type, v_rate;
  EXIT WHEN loan_cursor%NOTFOUND;

  IF v_type = 'HOME' THEN
   UPDATE Loans SET InterestRate = 6.5 WHERE LoanID = v_loan_id;
  ELSIF v_type = 'CAR' THEN
   UPDATE Loans SET InterestRate = 8.0 WHERE LoanID = v_loan_id;
  ELSIF v_type = 'PERSONAL' THEN
   UPDATE Loans SET InterestRate = 10.5 WHERE LoanID = v_loan_id;
  END IF;

  DBMS_OUTPUT.PUT_LINE('Updated interest for Loan ID ' || v_loan_id ||
             ' of type ' || v_type);
 END LOOP;
 CLOSE loan_cursor;

 COMMIT;
END;
/
```

OUTPUT---

```
Customer: Yeshwanth| Transaction ID: 105 | Type: DEPOSIT | Amount: 5000 | Date: 10-JUN-2025
```

EXERCISE-7

PROGRAM-1

```
CREATE OR REPLACE PACKAGE CustomerManagement AS
 PROCEDURE AddCustomer(p_id NUMBER, p_name VARCHAR2, p_age NUMBER, p_balance NUMBER);
 PROCEDURE UpdateCustomer(p_id NUMBER, p_name VARCHAR2, p_age 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_age NUMBER, p_balance
NUMBER) IS
  BEGIN
    INSERT INTO Customers(CustomerID, CustomerName, Age, Balance)
    VALUES (p_id, p_name, p_age, p_balance);
    COMMIT;
  END;

  PROCEDURE UpdateCustomer(p_id NUMBER, p_name VARCHAR2, p_age NUMBER) IS
  BEGIN
    UPDATE Customers
    SET CustomerName = p_name, Age = p_age
    WHERE CustomerID = p_id;
    COMMIT;
  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;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN NULL;
  END;
```

```
END CustomerManagement;
/
PROGRAM-2

CREATE OR REPLACE PACKAGE EmployeeManagement AS
  PROCEDURE HireEmployee(p_id NUMBER, p_name VARCHAR2, p_dept NUMBER, p_salary
NUMBER);
  PROCEDURE UpdateEmployee(p_id NUMBER, p_name VARCHAR2, p_dept NUMBER);
  FUNCTION CalculateAnnualSalary(p_id NUMBER) RETURN NUMBER;
END EmployeeManagement;
/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

  PROCEDURE HireEmployee(p_id NUMBER, p_name VARCHAR2, p_dept NUMBER, p_salary
NUMBER) IS
  BEGIN
    INSERT INTO Employees(EmployeeID, EmployeeName, DepartmentID, Salary)
    VALUES (p_id, p_name, p_dept, p_salary);
    COMMIT;
  END;

  PROCEDURE UpdateEmployee(p_id NUMBER, p_name VARCHAR2, p_dept NUMBER) IS
  BEGIN
    UPDATE Employees
    SET EmployeeName = p_name, DepartmentID = p_dept
    WHERE EmployeeID = p_id;
    COMMIT;
  END;

  FUNCTION CalculateAnnualSalary(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 NULL;
 END;

END EmployeeManagement;
/
```

PROGRAM-3

```
CREATE OR REPLACE PACKAGE AccountOperations AS
 PROCEDURE OpenAccount(p_acc_id NUMBER, p_cust_id NUMBER, p_type VARCHAR2,
p_balance NUMBER);
 PROCEDURE CloseAccount(p_acc_id NUMBER);
 FUNCTION GetTotalBalance(p_cust_id NUMBER) RETURN NUMBER;
END AccountOperations;
/

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)
  VALUES (p_acc_id, p_cust_id, p_type, p_balance);
  COMMIT;
 END;
 PROCEDURE CloseAccount(p_acc_id NUMBER) IS
 BEGIN
  DELETE FROM Accounts WHERE AccountID = p_acc_id;
  COMMIT;
 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—

```
Total Balance: 10000
Annual Salary: 540000
```