Week 2 – PL/SQL_Mandatory_HandsOn
Skill: PL/SQL Programming
Candidate: Projita Kar
Superset Id: 6407705
Type: Mandatory Hands-On

Exercise 1: Control Structures

- Creating Tables and Inserting Data

```
-- Create Customer table
CREATE TABLE customers (
  customer_id NUMBER PRIMARY KEY,
  age NUMBER,
  balance NUMBER,
  loan_interest_rate NUMBER,
  IsVIP VARCHAR2(5)
);

-- Create Loans table
CREATE TABLE loans (
  loan_id NUMBER PRIMARY KEY,
  customer_id NUMBER,
  due_date DATE
);

-- Insert sample data
INSERT INTO customers VALUES (1, 65, 15000, 8.5, 'FALSE');
INSERT INTO customers VALUES (2, 45, 8000, 9.0, 'FALSE');

INSERT INTO loans VALUES (101, 1, SYSDATE + 20);
INSERT INTO loans VALUES (102, 2, SYSDATE + 45);

COMMIT;

-- View initial data
SELECT * FROM customers;
SELECT * FROM loans;
```

	CUSTOMER_ID	AGE	BALANCE	LOAN_INTEREST_R/	ISVIP
1	1	65	15000	8.5	FALSE
2	2	45	8000	9	FALSE

	LOAN_ID	CUSTOMER_ID	DUE_DATE
1	101	1	7/17/2025, 11:15:50 AM
2	102	2	8/11/2025, 11:15:50 AM

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.

```
BEGIN
  FOR customer_rec IN (
    SELECT customer_id, age FROM customers
  ) LOOP
    IF customer_rec.age > 60 THEN
      UPDATE customers
      SET loan_interest_rate = loan_interest_rate - 1
      WHERE customer_id = customer_rec.customer_id;
    END IF;
  END LOOP;
  COMMIT;
END;

-- View result
SELECT * FROM customers;
```

	CUSTOMER_ID	AGE	BALANCE	LOAN_INTEREST_R/	ISVIP
1	1	65	15000	7.5	FALSE
2	2	45	8000	9	FALSE

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.

```
BEGIN
  FOR cust IN (
    SELECT customer_id, balance FROM customers
  ) LOOP
```

```
    IF cust.balance > 10000 THEN
      UPDATE customers
      SET IsVIP = 'TRUE'
      WHERE customer_id = cust.customer_id;
    END IF;
  END LOOP;
  COMMIT;
END;

-- View result
SELECT * FROM customers;
```

	CUSTOMER_ID	AGE	BALANCE	LOAN_INTEREST_R/	ISVIP
1	1	65	15000	7.5	TRUE
2	2	45	8000	9	FALSE

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.

```
BEGIN
  FOR loan_rec IN (
    SELECT customer_id, loan_id, due_date
    FROM loans
    WHERE due_date BETWEEN SYSDATE AND SYSDATE + 30
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan ID ' || loan_rec.loan_id ||
                         ' for Customer ID ' || loan_rec.customer_id ||
                         ' is due on ' || TO_CHAR(loan_rec.due_date, 'DD-MON-YYYY'));
  END LOOP;
END;
```

```
Reminder: Loan ID 101 for Customer ID 1 is due on 17-JUL-2025
```

Exercise 3: Stored Procedures

- Step 1: Creating Tables and Inserting Data

```
-- Savings Accounts Table
CREATE TABLE savings_accounts (
  account_id NUMBER PRIMARY KEY,
```

```
  customer_id NUMBER,
  balance NUMBER
);

-- Employees Table
CREATE TABLE employees (
  emp_id NUMBER PRIMARY KEY,
  emp_name VARCHAR2(100),
  department_id NUMBER,
  salary NUMBER
);

-- Generic Accounts Table (for fund transfer)
CREATE TABLE accounts (
  account_id NUMBER PRIMARY KEY,
  customer_id NUMBER,
  balance NUMBER
);

-- Insert data for savings_accounts (for interest processing)
INSERT INTO savings_accounts VALUES (201, 1, 10000);
INSERT INTO savings_accounts VALUES (202, 2, 15000);
INSERT INTO savings_accounts VALUES (203, 3, 20000);

-- Insert data for employees (for bonus calculation)
INSERT INTO employees VALUES (1, 'Amit Roy', 101, 50000);
INSERT INTO employees VALUES (2, 'Sneha Das', 101, 60000);
INSERT INTO employees VALUES (3, 'Ravi Mehra', 102, 55000);

-- Insert data for accounts (for fund transfer)
INSERT INTO accounts VALUES (1001, 1, 8000);
INSERT INTO accounts VALUES (1002, 2, 4000);
INSERT INTO accounts VALUES (1003, 3, 2000);

COMMIT;

-- View initial data
SELECT * FROM savings_accounts;
SELECT * FROM employees;
SELECT * FROM accounts;
```

	ACCOUNT_ID	CUSTOMER_ID	BALANCE
1	201	1	10000
2	202	2	15000
3	203	3	20000

	EMP_ID	EMP_NAME	DEPARTMENT_ID	SALARY
1	1	Amit Roy	101	50000
2	2	Sneha Das	101	60000
3	3	Ravi Mehra	102	55000

	ACCOUNT_ID	CUSTOMER_ID	BALANCE
1	1001	1	8000
2	1002	2	4000
3	1003	3	2000

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.

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  FOR acc IN (SELECT account_id, balance FROM savings_accounts) LOOP
    UPDATE savings_accounts
    SET balance = balance + (acc.balance * 0.01)
    WHERE account_id = acc.account_id;
  END LOOP;
  COMMIT;
END;

-- Run procedure
```

```
EXEC ProcessMonthlyInterest;

-- View result
SELECT * FROM SAVINGS_ACCOUNTS;
```

	ACCOUNT_ID	CUSTOMER_ID	BALANCE
1	201	1	10100
2	202	2	15150
3	203	3	20200

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.

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(
  dept_id IN NUMBER,
  bonus_percent IN NUMBER
) IS
BEGIN
  UPDATE employees
  SET salary = salary + (salary * (bonus_percent / 100))
  WHERE department_id = dept_id;

  COMMIT;
END;

-- Run procedure
EXEC UpdateEmployeeBonus(101, 10);

-- View result
SELECT * FROM EMPLOYEES;
```

	EMP_ID	EMP_NAME	DEPARTMENT_ID	SALARY
1	1	Amit Roy	101	55000
2	2	Sneha Das	101	66000
3	3	Ravi Mehra	102	55000

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.

```
CREATE OR REPLACE PROCEDURE TransferFunds(
  from_account IN NUMBER,
  to_account IN NUMBER,
  amount IN NUMBER
) IS
  from_balance NUMBER;
BEGIN
  SELECT balance INTO from_balance FROM accounts
  WHERE account_id = from_account FOR UPDATE;

  IF from_balance < amount THEN
    RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance in source account.');
  END IF;

  UPDATE accounts
  SET balance = balance - amount
  WHERE account_id = from_account;

  UPDATE accounts
  SET balance = balance + amount
  WHERE account_id = to_account;

  COMMIT;
END;

-- Run procedure
EXEC TransferFunds(1001, 1002, 500);

-- View result
SELECT * FROM ACCOUNTS;
```

	ACCOUNT_ID	CUSTOMER_ID	BALANCE
1	1001	1	7500
2	1002	2	4500
3	1003	3	2000