Sushmit Partakke
23070521156
Lab 8
PL/SQL Procedure for Fund Transfer

Step 1: Create Database Tables

1.1 Create `accounts` Table

```
CREATE TABLE accounts (
account_no NUMBER PRIMARY KEY,
holder_name VARCHAR2(100),
    balance NUMBER(10,2) CHECK (balance >= 0) );
```

1.2 Create `transactions` Table

```
CREATE TABLE transactions (
    transaction_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY
KEY,
    from_account NUMBER,
to_account NUMBER,     amount
NUMBER(10,2),
    transaction_date TIMESTAMP DEFAULT SYSTIMESTAMP );
```

Step 2: Insert Sample Data

```
INSERT INTO accounts VALUES (101, 'Alice', 5000.00);
INSERT INTO accounts VALUES (102, 'Bob', 3000.00); COMMIT;
```

Step 3: Write PL/SQL Procedure

```
-- Procedure: transfer_funds
-- Transfers a specified amount from one account to another
CREATE OR REPLACE PROCEDURE transfer_funds(
    p_from_acc NUMBER,  -- Sender's account number
    p_to_acc NUMBER,    -- Receiver's account number
    p_amount NUMBER     -- Transfer amount
) AS
    v_balance NUMBER;  -- Variable to store sender's balance
BEGIN
    -- Get sender's current balance
    SELECT balance INTO v_balance FROM accounts WHERE account_no =
p_from_acc;

    -- Check if balance is sufficient
    IF v_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance.');
    END IF;

    -- Deduct amount from sender
    UPDATE accounts SET balance = balance - p_amount WHERE account_no
= p_from_acc;

    -- Add amount to receiver
    UPDATE accounts SET balance = balance + p_amount WHERE account_no
= p_to_acc;

    -- Insert transaction record
    INSERT INTO transactions (from_account, to_account, amount)
    VALUES (p_from_acc, p_to_acc, p_amount);

    -- Commit transaction
    COMMIT;

    -- Print success message
    DBMS_OUTPUT.PUT_LINE('Transfer successful.');

EXCEPTION
    -- Handle case where account does not exist
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20002, 'Invalid account number.');
```

```
    -- Handle other errors
    WHEN OTHERS THEN
        ROLLBACK;  -- Undo changes if an error occurs
        RAISE_APPLICATION_ERROR(-20003, 'Transaction failed: ' ||
SQLERRM);
END;
/
```

Step 4: Execute Procedure

```
-- Transfer ₹1000 from account 101 to 102
BEGIN
    transfer_funds(101, 102, 1000);
END; /
```

```
Transfer successful.

ACCOUNT_NO
----------
HOLDER_NAME
--------------------------------------------------------------------------------
   BALANCE
----------
       101
Alice
      4000

       102
Bob
      4000

ACCOUNT_NO
----------
HOLDER_NAME
--------------------------------------------------------------------------------
   BALANCE
----------
```

```
TRANSACTION_ID FROM_ACCOUNT TO_ACCOUNT     AMOUNT
-------------- ------------ ---------- ----------
TRANSACTION_DATE
---------------------------------------------------------------------------
             1          101        102       1000
03-APR-25 09.09.19.274466 AM


TRANSACTION_ID FROM_ACCOUNT TO_ACCOUNT     AMOUNT
-------------- ------------ ---------- ----------
TRANSACTION_DATE
---------------------------------------------------------------------------
             1          101        102       1000
03-APR-25 09.09.19.274466 AM
```

Step 5: Verify Results

Check Account Balances `SELECT`

```
-- Display all accounts and
balances
* FROM accounts;
```

Check Transactions Log

```
-- Display all transaction records
SELECT * FROM transactions;
```

Task: Fund Transfer Validation and Execution

Task 1: Check Account Balance Before Transfer - Write a PL/SQL block that takes an account number as input and displays the account balance.

```
-- Fetch and display balance for a specific account

DECLARE

    v_balance NUMBER;

BEGIN

    SELECT balance INTO v_balance FROM accounts WHERE account_no =
101;

    DBMS_OUTPUT.PUT_LINE('Balance: ' || v_balance);

END;

/
```

```
Balance: 4000
```

Hint: Use `SELECT balance INTO` inside a PL/SQL block and `DBMS_OUTPUT.PUT_LINE` to display the balance.

Task 2: Execute Fund Transfer Procedure - Call the `transfer_funds` procedure to transfer **₹500 from account 101 to account 102**.

Hint: Use the `BEGIN...END;` block to execute the procedure.

```
-- Transfer ₹500 from account 101 to 102

BEGIN

    transfer_funds(101, 102, 500);

END;

/
```

```
Transfer successful.
```

Task 3: Validate Transaction Log - After executing the transfer, write an SQL query to display all transactions recorded in the `transactions` table.

```
-- Display all transactions recorded in the transactions table

SELECT * FROM transactions ORDER BY transaction_date DESC;
```

```
TRANSACTION_ID FROM_ACCOUNT TO_ACCOUNT     AMOUNT
-------------- ------------ ---------- ----------
TRANSACTION_DATE
---------------------------------------------------------------------------
             2          101        102        500
03-APR-25 09.11.47.892030 AM

             1          101        102       1000
03-APR-25 09.11.47.839267 AM


TRANSACTION_ID FROM_ACCOUNT TO_ACCOUNT     AMOUNT
-------------- ------------ ---------- ----------
TRANSACTION_DATE
---------------------------------------------------------------------------
             2          101        102        500
03-APR-25 09.11.47.892030 AM

             1          101        102       1000
03-APR-25 09.11.47.839267 AM
```

Hint: Use `SELECT * FROM transactions;` to verify the transaction details.

Task 4: Check Transaction History for a Specific Account

Write a PL/SQL block that takes an account number as input and displays all transactions (both sent and received) related to that account.

```
-- PL/SQL Block to Display Transaction History for a Given Account
```

```
DECLARE
    acc_no NUMBER := 101;  -- Change this to the desired account
number
BEGIN
    -- Loop through all transactions related to the given account
    FOR rec IN (
        SELECT * FROM transactions
        WHERE from_account = acc_no OR to_account = acc_no
        ORDER BY transaction_date DESC
    )
    LOOP
        -- Display transaction details
        DBMS_OUTPUT.PUT_LINE('Transaction ID: ' || rec.transaction_id
||
                             ', From: ' || rec.from_account ||
                             ', To: ' || rec.to_account ||
                             ', Amount: ' || rec.amount ||
                             ', Date: ' || rec.transaction_date);
    END LOOP;
END;
/
```

```
Transaction ID: 2, From: 101, To: 102, Amount: 500, Date: 03-APR-25
09.12.51.413350 AM
Transaction ID: 1, From: 101, To: 102, Amount: 1000, Date: 03-APR-25
09.12.51.372460 AM
```

Hint: Use `SELECT * FROM transactions WHERE from_account = acc_no OR to_account = acc_no;` inside a PL/SQL block.

Task 5: Prevent Self-Transfer

Modify the `transfer_funds` procedure to prevent an account from transferring money to itself. If the sender and receiver accounts are the same, raise an error message.

```
-- Function: get_balance

-- Returns the current balance of a given account

CREATE OR REPLACE FUNCTION get_balance(p_acc_no NUMBER) RETURN NUMBER

AS

    v_balance NUMBER;

BEGIN

    SELECT balance INTO v_balance FROM accounts WHERE account_no =
p_acc_no;

    RETURN v_balance;

END;

/

-- Retrieve balance for account 101

SELECT get_balance(101) FROM dual;
```

```
GET_BALANCE(101)
----------------
            3500

GET_BALANCE(101)
----------------
            3500
```

Hint: Add a condition inside the procedure:

```
IF p_from_acc = p_to_acc THEN
   RAISE_APPLICATION_ERROR(-20004, 'Sender and receiver cannot be the
same.');   END IF;
```

Task 6: Create a Function to Check Account Balance

Write a PL/SQL function named `get_balance` that takes an account number as input and returns the current balance.

```
-- Function: get_balance

-- Returns the current balance of a given account

CREATE OR REPLACE FUNCTION get_balance(p_acc_no NUMBER) RETURN NUMBER

AS

    v_balance NUMBER;  -- Variable to store account balance

BEGIN

    -- Fetch balance for the given account

    SELECT balance INTO v_balance FROM accounts WHERE account_no =
p_acc_no;



    -- Return the retrieved balance

    RETURN v_balance;

EXCEPTION

    -- Handle case where account does not exist

    WHEN NO_DATA_FOUND THEN

        RETURN NULL;  -- Return NULL if the account is not found

END;
```

/ **Hint:**

```
CREATE OR REPLACE FUNCTION get_balance(p_acc_no NUMBER) RETURN NUMBER
AS
    v_balance NUMBER;
BEGIN
    SELECT balance INTO v_balance FROM accounts WHERE account_no =
p_acc_no;
    RETURN v_balance;
END; /
```

Call it using:

```
SELECT get_balance(101) FROM dual;
```

```
 GET_BALANCE(101)
 ----------------
             3500
```

Task 7: Implement a Transfer Limit

Modify the `transfer_funds` procedure to set a maximum transfer limit of ₹10,000 per transaction. If a user tries to transfer more than this amount, raise an error.

```
-- Procedure: transfer_funds

-- Transfers a specified amount from one account to another with a
transfer limit

CREATE OR REPLACE PROCEDURE transfer_funds(

    p_from_acc NUMBER,  -- Sender's account number

    p_to_acc NUMBER,    -- Receiver's account number

    p_amount NUMBER     -- Transfer amount

) AS

    v_balance NUMBER;  -- Variable to store sender's balance

BEGIN
```

```
    -- Prevent self-transfer
    IF p_from_acc = p_to_acc THEN
        RAISE_APPLICATION_ERROR(-20004, 'Sender and receiver cannot be
the same.');
    END IF;

    -- Enforce maximum transfer limit
    IF p_amount > 10000 THEN
        RAISE_APPLICATION_ERROR(-20005, 'Transfer amount exceeds the
limit of ₹10,000.');
    END IF;

    -- Get sender's current balance
    SELECT balance INTO v_balance FROM accounts WHERE account_no =
p_from_acc;

    -- Check if balance is sufficient
    IF v_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance.');
    END IF;

    -- Deduct amount from sender
    UPDATE accounts SET balance = balance - p_amount WHERE account_no
= p_from_acc;

    -- Add amount to receiver
```

```
    UPDATE accounts SET balance = balance + p_amount WHERE account_no
= p_to_acc;

    -- Insert transaction record
    INSERT INTO transactions (from_account, to_account, amount)
    VALUES (p_from_acc, p_to_acc, p_amount);

    -- Commit transaction
    COMMIT;

    -- Print success message
    DBMS_OUTPUT.PUT_LINE('Transfer successful.');

EXCEPTION
    -- Handle case where account does not exist
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20002, 'Invalid account number.');

    -- Handle other errors
    WHEN OTHERS THEN
        ROLLBACK;  -- Undo changes if an error occurs
        RAISE_APPLICATION_ERROR(-20003, 'Transaction failed: ' ||
SQLERRM);
END;
/
```

Hint: Add a condition:

```
IF p_amount > 10000 THEN
   RAISE_APPLICATION_ERROR(-20005, 'Transfer amount exceeds the limit
of ₹10,000.');
END IF;
```

Task 8: Generate a Monthly Statement

Write a PL/SQL procedure that takes an account number and a month-year (e.g., `04-2025`) as input and displays all transactions for that month.

Hint: Use `TO_CHAR(transaction_date, 'MM-YYYY')` in the `WHERE` clause:

```
-- Procedure: generate_monthly_statement
-- Displays all transactions for a given account and month-year
CREATE OR REPLACE PROCEDURE generate_monthly_statement(
    p_acc_no NUMBER,      -- Account number
    p_month_year VARCHAR2 -- Month and year in 'MM-YYYY' format
) AS
BEGIN
    -- Loop through transactions for the given account and month
    FOR rec IN (
        SELECT * FROM transactions
        WHERE (from_account = p_acc_no OR to_account = p_acc_no)
        AND TO_CHAR(transaction_date, 'MM-YYYY') = p_month_year
    ) LOOP
        -- Print transaction details
```

```
        DBMS_OUTPUT.PUT_LINE('Transaction ID: ' || rec.transaction_id
||
                             ', Date: ' || rec.transaction_date ||
                             ', From: ' || rec.from_account ||
                             ', To: ' || rec.to_account ||
                             ', Amount: ' || rec.amount);
    END LOOP;
END;
/
```

```
 TRANSACTION_ID FROM_ACCOUNT TO_ACCOUNT     AMOUNT
 -------------- ------------ ---------- ----------
 TRANSACTION_DATE
 ---------------------------------------------------------
              1          101        102       1000
 03-APR-25 09.18.42.418965 AM
```