Shreya Bansal
23070521144
B2

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

```
CREATE OR REPLACE PROCEDURE transfer_funds(
    p_from_acc NUMBER,
    p_to_acc NUMBER,
    p_amount NUMBER
CREATE OR REPLACE PROCEDURE is used to either 1. Create a new
procedure or 2. Replace the data if the procedure by the same name
already exists

Here NUMBER specifies the data type as number as we are talking about
payments
) AS
    v_balance NUMBER; Here AS is used to begin the procedure by
    storing the balance too where v_balance is the declared variable
BEGIN Begins the procedure
    -- Check if sender has sufficient balance
    SELECT balance INTO v_balance FROM accounts WHERE account_no =
p_from_acc; Here SELECT selects data from database

    INTO specifies in which variable/database to look for WHERE
specifies which row's data to select here where account no.
equals to the payment from account

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

        IF THEN is the conditional keyword pair which does what's
    written if the condition specified is true

        Here it is to raise error if the payment amount exceeds
    the sender's balance

        END IF specifies that the IF statement is closed here

    -- Deduct amount from sender

    UPDATE accounts SET balance = balance - p_amount WHERE account_no
= p_from_acc; UPDATE keyword is used to edit/update/change the data
where data was already inserted; that's how it is different from
INSERT

    -- Log transaction
```

```
    INSERT INTO transactions (from_account, to_account, amount)
    VALUES (p_from_acc, p_to_acc, p_amount);
    INSERT is used to insert data in the rows mentioned and
    VALUES specify their value

    -- Commit transaction
    COMMIT; COMMIT saves all changes permanently

    DBMS_OUTPUT.PUT_LINE('Transfer successful.'); Gives (Prints) the
    OUTPUT successfully

EXCEPTION specifies the exception (errors and how to handle them)
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20002, 'Invalid account number.');
    WHEN OTHERS THEN Here when SELECT returns no data
        ROLLBACK; ROLLBACK reverts all changes when an error is found
RAISE_APPLICATION_ERROR(-20003, 'Transaction failed: ' || SQLERRM);
SQLERM returns the last error
END;
/
Specifies the end of our program/queries
```

Step 4: Execute Procedure

```
BEGIN
    transfer_funds(101, 102, 1000);
END;
/
```

Step 5: Verify Results

Check Account Balances

```
SELECT * FROM accounts;
```

Check Transactions Log

```
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.

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.

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

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.

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.

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.

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;
```

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.

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:

```
SELECT * FROM transactions
```

```
WHERE (from_account = acc_no OR to_account = acc_no)
AND TO_CHAR(transaction_date, 'MM-YYYY') = '04-2025';
```

```
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> BEGIN
  2      EXECUTE IMMEDIATE 'DROP TABLE transactions';
  3  EXCEPTION
  4      WHEN OTHERS THEN NULL;
  5  END;
  6  /

PL/SQL procedure successfully completed.

SQL>
SQL> BEGIN
  2      EXECUTE IMMEDIATE 'DROP TABLE accounts';
  3  EXCEPTION
  4      WHEN OTHERS THEN NULL;
  5  END;
  6  /

PL/SQL procedure successfully completed.

SQL>
SQL> CREATE TABLE accounts (
  2      account_no NUMBER PRIMARY KEY,
  3      holder_name VARCHAR2(100),
  4      balance NUMBER(10,2) CHECK (balance >= 0)
  5  );

Table created.

SQL>
SQL> CREATE TABLE transactions (
  2      transaction_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  3      from_account NUMBER,
  4      to_account NUMBER,
  5      amount NUMBER(10,2),
  6      transaction_date TIMESTAMP DEFAULT SYSTIMESTAMP
  7  );
    transaction_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
                                                *
ERROR at line 2:
ORA-02000: missing ALWAYS keyword


SQL>
SQL> INSERT INTO accounts VALUES (101, 'Alice', 5000.00);

1 row created.

SQL> INSERT INTO accounts VALUES (102, 'Bob', 3000.00);
```

```
1 row created.

SQL> COMMIT;

Commit complete.

SQL>
SQL> CREATE OR REPLACE PROCEDURE transfer_funds(
  2      p_from_acc NUMBER,
  3      p_to_acc NUMBER,
  4      p_amount NUMBER
  5  ) AS
  6      v_balance NUMBER;
  7  BEGIN
  8      IF p_from_acc = p_to_acc THEN
  9          RAISE_APPLICATION_ERROR(-20004, 'Sender and receiver cannot be the same.');
 10      END IF;
 11      IF p_amount > 10000 THEN
 12          RAISE_APPLICATION_ERROR(-20005, 'Transfer amount exceeds the limit of ₹10,000.');
 13      END IF;
 14      SELECT balance INTO v_balance FROM accounts WHERE account_no = p_from_acc;
 15      IF v_balance < p_amount THEN
 16          RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance.');
 17      END IF;
 18      UPDATE accounts SET balance = balance - p_amount WHERE account_no = p_from_acc;
 19      UPDATE accounts SET balance = balance + p_amount WHERE account_no = p_to_acc;
 20      INSERT INTO transactions (from_account, to_account, amount) VALUES (p_from_acc, p_to_acc, p_amount);
 21      COMMIT;
 22      DBMS_OUTPUT.PUT_LINE('Transfer successful.');
 23  EXCEPTION
 24      WHEN NO_DATA_FOUND THEN
 25          RAISE_APPLICATION_ERROR(-20002, 'Invalid account number.');
 26      WHEN OTHERS THEN
 27          ROLLBACK;
 28          RAISE_APPLICATION_ERROR(-20003, 'Transaction failed: ' || SQLERRM);
 29  END;
 30  /

Warning: Procedure created with compilation errors.

SQL>
SQL> CREATE OR REPLACE FUNCTION get_balance(p_acc_no NUMBER) RETURN NUMBER AS
  2      v_balance NUMBER;
  3  BEGIN
  4      SELECT balance INTO v_balance FROM accounts WHERE account_no = p_acc_no;
  5      RETURN v_balance;
  6  END;
  7  /

Function created.
```

```
SQL>
SQL> CREATE OR REPLACE PROCEDURE generate_monthly_statement(
  2      p_acc_no NUMBER,
  3      p_month_year VARCHAR2
  4  ) AS
  5  BEGIN
  6      FOR r IN (
  7          SELECT * FROM transactions
  8          WHERE (from_account = p_acc_no OR to_account = p_acc_no)
  9          AND TO_CHAR(transaction_date, 'MM-YYYY') = p_month_year
 10      ) LOOP
 11          DBMS_OUTPUT.PUT_LINE('Transaction ID: ' || r.transaction_id || ', From: ' || r.from_account || ', To: ' || r.to_account || ', Amount: ' |
-MON-YYYY'));
 12      END LOOP;
 13  END;
 14  /

Warning: Procedure created with compilation errors.

SQL>
SQL> BEGIN
  2      transfer_funds(101, 102, 500);
  3  END;
  4  /
    transfer_funds(101, 102, 500);
    *
ERROR at line 2:
ORA-06550: line 2, column 5:
PLS-00905: object SYSTEM.TRANSFER_FUNDS is invalid
ORA-06550: line 2, column 5:
PL/SQL: Statement ignored


SQL>
SQL> SELECT * FROM accounts;

ACCOUNT_NO
----------
HOLDER_NAME
--------------------------------------------------------------------------------
   BALANCE
----------
       101
Alice
      5000

       102
Bob
      3000
```

```
ACCOUNT_NO
----------
HOLDER_NAME
--------------------------------------------------------------------------------
   BALANCE
----------
       101
Alice
      5000

       102
Bob
      3000

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


SQL> SELECT * FROM transactions;
SELECT * FROM transactions
              *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL>
SQL> BEGIN
  2      generate_monthly_statement(101, '04-2025');
  3  END;
  4  /
    generate_monthly_statement(101, '04-2025');
    *
ERROR at line 2:
ORA-06550: line 2, column 5:
PLS-00905: object SYSTEM.GENERATE_MONTHLY_STATEMENT is invalid
ORA-06550: line 2, column 5:
PL/SQL: Statement ignored


SQL>
SQL> SELECT get_balance(101) FROM dual;

GET_BALANCE(101)
----------------
            5000
```