Lab 8
PL/SQL Procedure for Fund Transfer
SHIVANSH LOHANI
23070521141

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,    -- Sender's account number
        p_to_acc NUMBER,      -- Receiver's account number
        p_amount NUMBER        -- Amount to be transferred
) AS
        v_balance NUMBER;     Variable to store sender's account balance
BEGIN
        -- Check if sender has sufficient balance
        SELECT balance INTO v_balance FROM accounts WHERE account_no = p_from_acc;
        If sender's balance is less than the transfer amount, raise an error

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

        -- Deduct the transfer amount from the sender's accou
        UPDATE accounts SET balance = balance - p_amount WHERE account_no = p_from_acc;

        -- Add the transfer amount to the receiver's account
        UPDATE accounts SET balance = balance + p_amount WHERE account_no = p_to_acc;

        -- Log the transaction details in the transactions table
        INSERT INTO transactions (from_account, to_account, amount) VALUES
        (p_from_acc, p_to_acc, p_amount);

        -- Commit the transaction to permanently save changes
        COMMIT;

        DBMS_OUTPUT.PUT_LINE('Transfer successful.');

EXCEPTION        Handle any other errors that occur during the transaction
        WHEN NO_DATA_FOUND THEN
                                        RAISE_APPLICATION_ERROR(-20002, 'Invalid account number.');
        WHEN OTHERS THEN    Handle any other errors that occur during the transaction
                ROLLBACK;     Undo any changes if an error occurs
                RAISE_APPLICATION_ERROR(-20003, 'Transaction failed: ' || SQLERRM);
END;
/
```

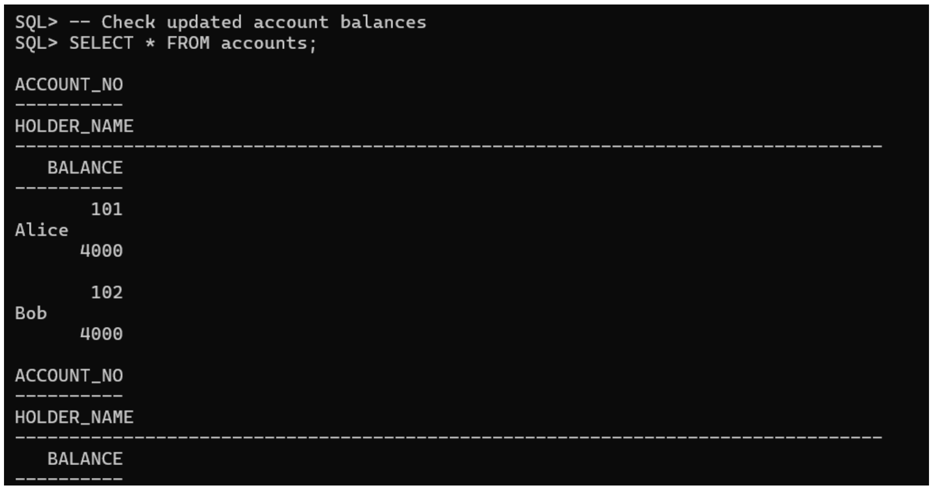
Step 4: Execute Procedure

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

Step 5: Verify Results

Check Account Balances

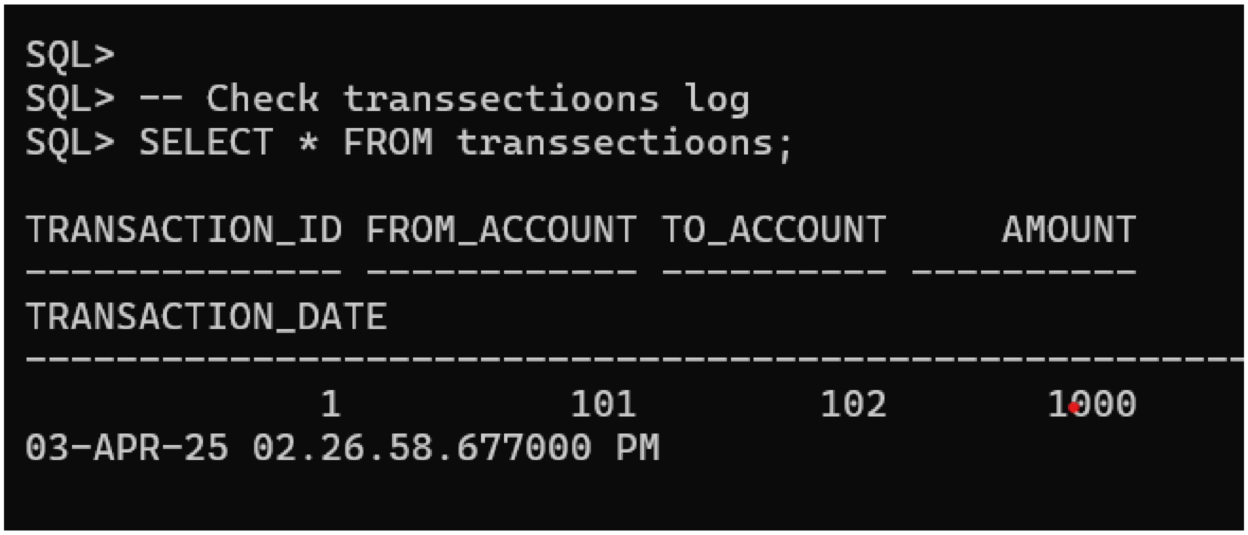
```
SELECT * FROM accounts;
```



```
SQL> -- Check updated account balances
SQL> SELECT * FROM accounts;

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

       102
Bob
      4000

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

Check Transactions Log

```
SELECT * FROM transactions;
```

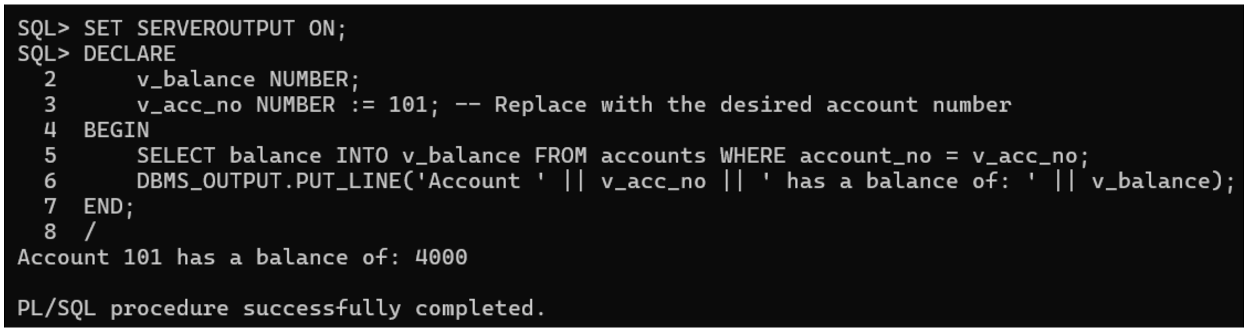


```
SQL>
SQL> -- Check transsectioons log
SQL> SELECT * FROM transsectioons;

TRANSACTION_ID FROM_ACCOUNT TO_ACCOUNT     AMOUNT
-------------- ------------ ---------- ----------
TRANSACTION_DATE
---------------------------------------------------------------------------
             1          101        102       1000
03-APR-25 02.26.58.677000 PM
```

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.

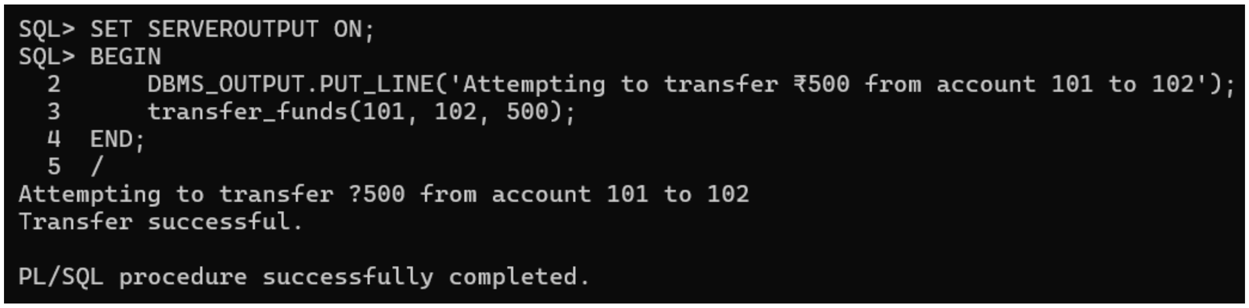
Hint: Use SELECT balance INTO inside a PL/SQL block and DBMS_OUTPUT.PUT_LINE to display the balance.



```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
         v_balance NUMBER;
         v_acc_no NUMBER := 101; -- Replace with the desired account number
     BEGIN
         SELECT balance INTO v_balance FROM accounts WHERE account_no = v_acc_no;
         DBMS_OUTPUT.PUT_LINE('Account ' || v_acc_no || ' has a balance of: ' || v_balance);
     END;
     /
Account 101 has a balance of: 4000

PL/SQL procedure successfully completed.
```

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

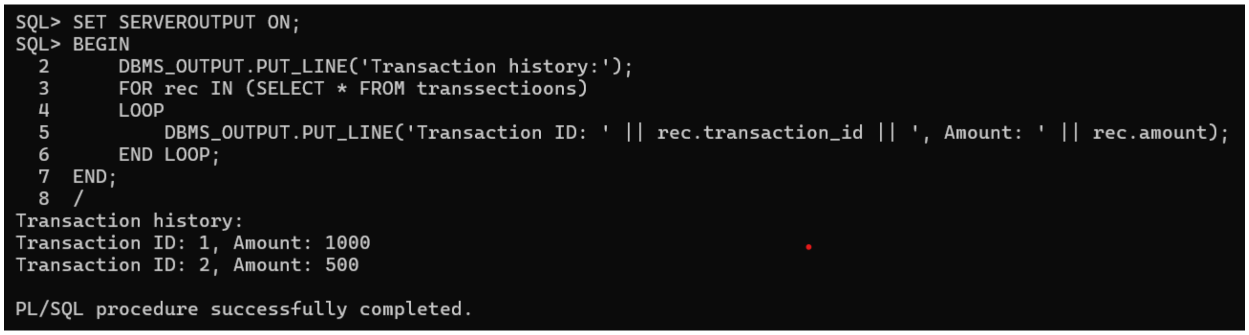
Hint: Use the BEGIN...END; block to execute the procedure.



```
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
         DBMS_OUTPUT.PUT_LINE('Attempting to transfer ₹500 from account 101 to 102');
         transfer_funds(101, 102, 500);
     END;
     /
Attempting to transfer ?500 from account 101 to 102
Transfer successful.

PL/SQL procedure successfully completed.
```

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.

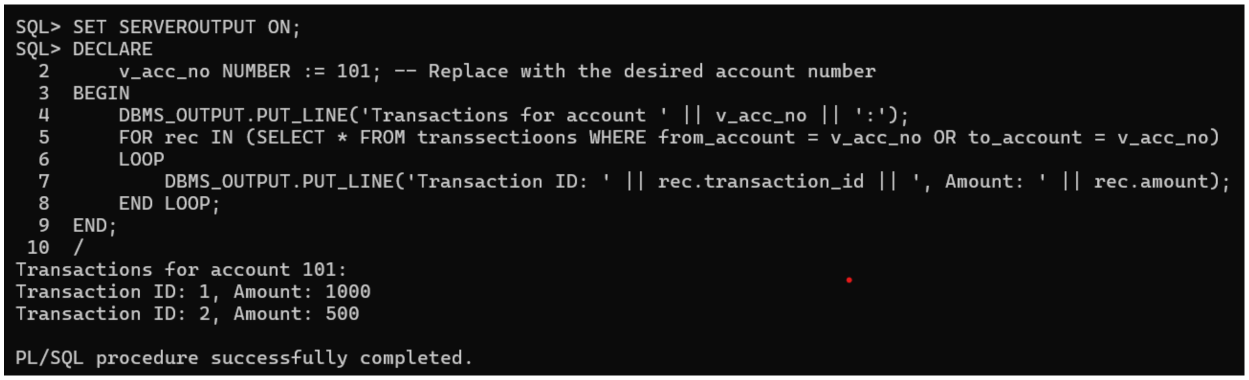


```
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
         DBMS_OUTPUT.PUT_LINE('Transaction history:');
         FOR rec IN (SELECT * FROM transsectioons)
         LOOP
             DBMS_OUTPUT.PUT_LINE('Transaction ID: ' || rec.transaction_id || ', Amount: ' || rec.amount);
         END LOOP;
     END;
     /
Transaction history:
Transaction ID: 1, Amount: 1000
Transaction ID: 2, Amount: 500

PL/SQL procedure successfully completed.
```

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.

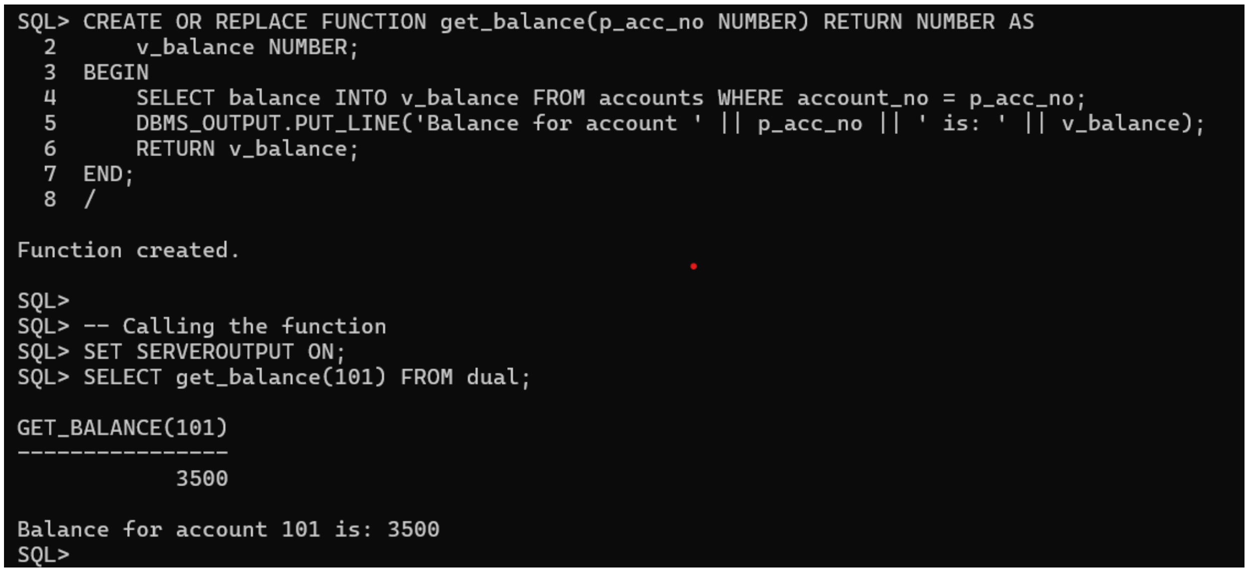

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
         v_acc_no NUMBER := 101; -- Replace with the desired account number
     BEGIN
         DBMS_OUTPUT.PUT_LINE('Transactions for account ' || v_acc_no || ':');
         FOR rec IN (SELECT * FROM transsectioons WHERE from_account = v_acc_no OR to_account = v_acc_no)
         LOOP
             DBMS_OUTPUT.PUT_LINE('Transaction ID: ' || rec.transaction_id || ', Amount: ' || rec.amount);
         END LOOP;
     END;
     /
Transactions for account 101:
Transaction ID: 1, Amount: 1000
Transaction ID: 2, Amount: 500

PL/SQL procedure successfully completed.
```

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;
```


```
SQL> 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;
         DBMS_OUTPUT.PUT_LINE('Balance for account ' || p_acc_no || ' is: ' || v_balance);
         RETURN v_balance;
     END;
     /

Function created.

SQL>
SQL> -- Calling the function
SQL> SET SERVEROUTPUT ON;
SQL> SELECT get_balance(101) FROM dual;

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

Balance for account 101 is: 3500
SQL>
```

Task 6: Create a Function to Check Account Balance

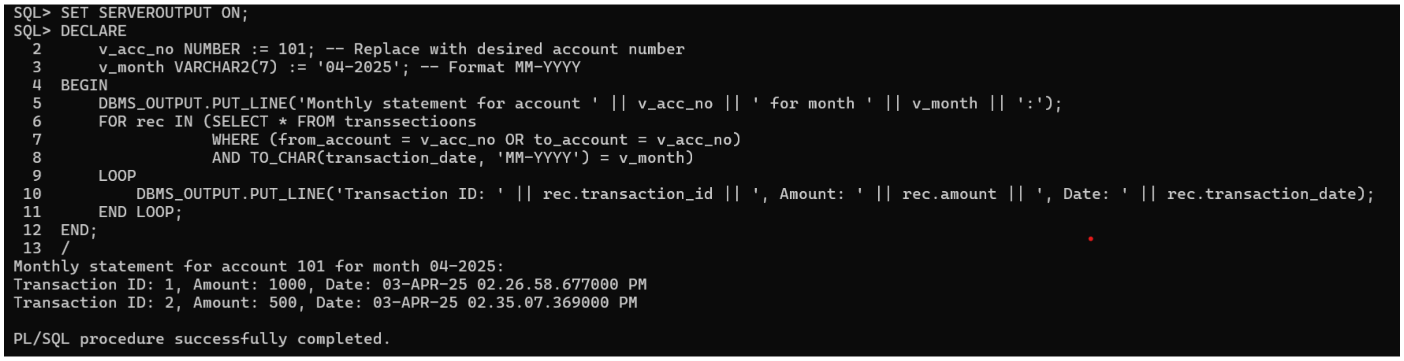
Write a PL/SQL function named get_balance that takes an account number as input and returns the current balance.

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;
```



```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2      v_acc_no NUMBER := 101; -- Replace with desired account number
  3      v_month VARCHAR2(7) := '04-2025'; -- Format MM-YYYY
  4  BEGIN
  5      DBMS_OUTPUT.PUT_LINE('Monthly statement for account ' || v_acc_no || ' for month ' || v_month || ':');
  6      FOR rec IN (SELECT * FROM transsectioons
  7                  WHERE (from_account = v_acc_no OR to_account = v_acc_no)
  8                  AND TO_CHAR(transaction_date, 'MM-YYYY') = v_month)
  9      LOOP
 10          DBMS_OUTPUT.PUT_LINE('Transaction ID: ' || rec.transaction_id || ', Amount: ' || rec.amount || ', Date: ' || rec.transaction_date);
 11      END LOOP;
 12  END;
 13  /
Monthly statement for account 101 for month 04-2025:
Transaction ID: 1, Amount: 1000, Date: 03-APR-25 02.26.58.677000 PM
Transaction ID: 2, Amount: 500, Date: 03-APR-25 02.35.07.369000 PM

PL/SQL procedure successfully completed.
```

Task 7: Implement a Transfer Limit

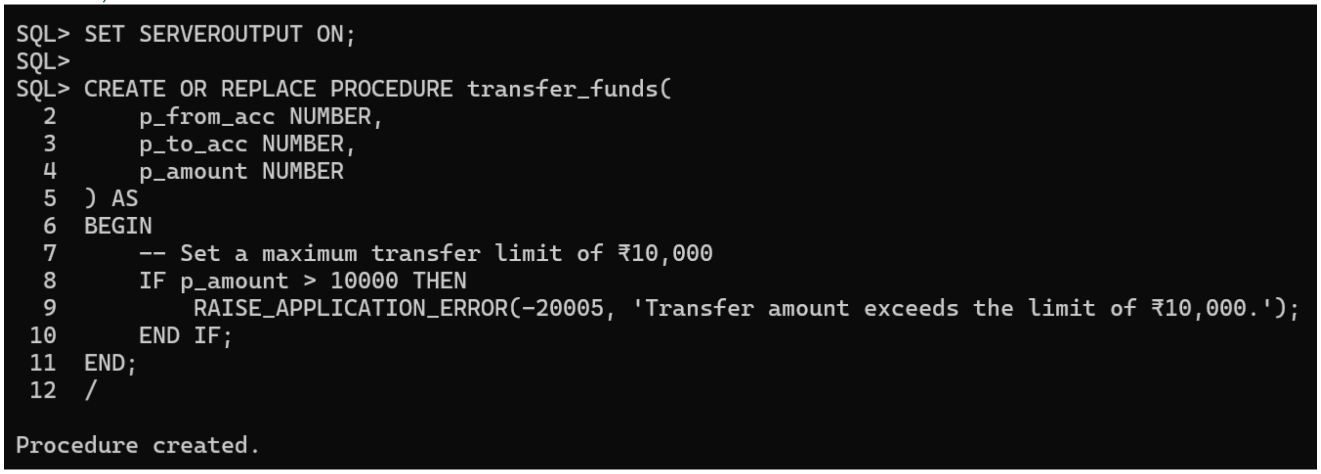
Modify the transfer_funds procedure to set a maximum transfer limit of ₹10,000 per transaction. If a user tries to transfer more than this amount, raise an error.

Hint: Add a condition:

IF p_amount > 10000 THEN

    RAISE_APPLICATION_ERROR(-20005, 'Transfer amount exceeds the limit of ₹10,000.');

END IF;



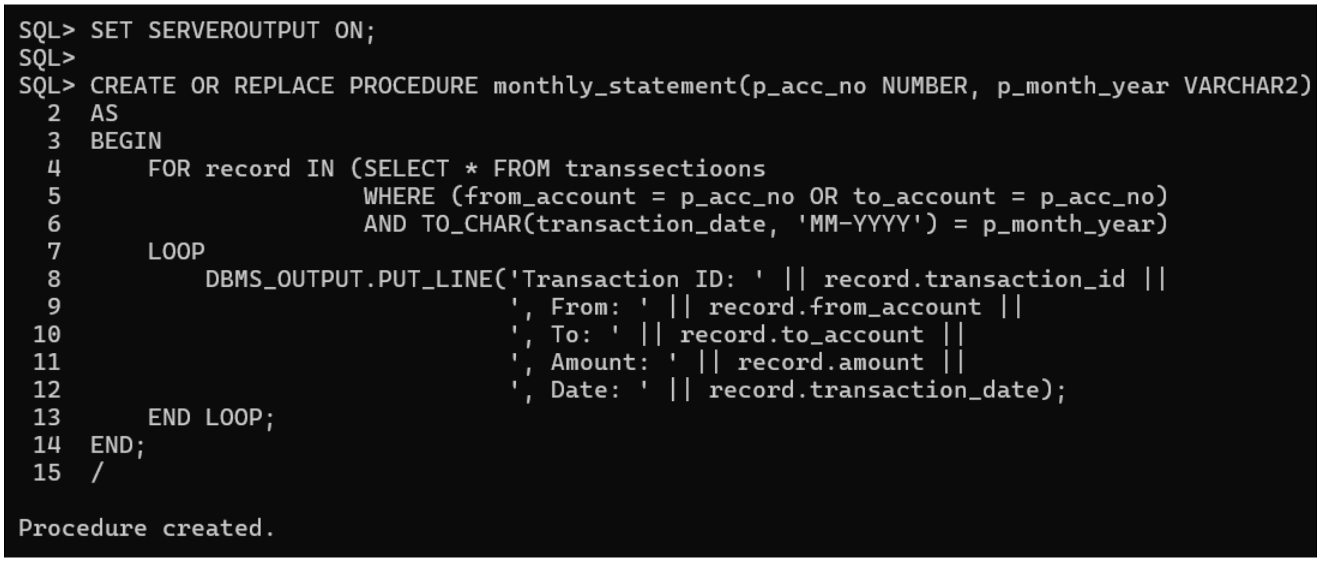
```
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> CREATE OR REPLACE PROCEDURE transfer_funds(
        p_from_acc NUMBER,
        p_to_acc NUMBER,
        p_amount NUMBER
    ) AS
    BEGIN
        -- Set a maximum transfer limit of ₹10,000
        IF p_amount > 10000 THEN
            RAISE_APPLICATION_ERROR(-20005, 'Transfer amount exceeds the limit of ₹10,000.');
        END IF;
    END;
    /

Procedure created.
```

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> CREATE OR REPLACE PROCEDURE monthly_statement(p_acc_no NUMBER, p_month_year VARCHAR2)
AS
BEGIN
    FOR record IN (SELECT * FROM transsectioons
                   WHERE (from_account = p_acc_no OR to_account = p_acc_no)
                   AND TO_CHAR(transaction_date, 'MM-YYYY') = p_month_year)
    LOOP
        DBMS_OUTPUT.PUT_LINE('Transaction ID: ' || record.transaction_id ||
                             ', From: ' || record.from_account ||
                             ', To: ' || record.to_account ||
                             ', Amount: ' || record.amount ||
                             ', Date: ' || record.transaction_date);
    END LOOP;
END;
/

Procedure created.
```