ASSIGNMENT 4

Q NO. 5

Introduction:

The purpose of this report is to explain the two PL/SQL blocks that have been created to implement the book issue and book return functionalities in a library management system. The PL/SQL blocks have been designed based on the schema created in Assignment 3, which includes three tables: BOOK, MEMBER, and TRANSACTION. The PL/SQL blocks have been created to ensure proper validation and checks are in place to prevent unauthorized book issue or return.

PL/SQL Block for Book Issue (i):

The PL/SQL block for book issue has been designed to issue a book to a valid member, check the availability of the book, and ensure that the member is eligible to borrow the book. The PL/SQL block takes input parameters for the member ID, book ID, and serial number.

The PL/SQL block first checks if the member is eligible to borrow the book based on their membership type and the number of books they have already borrowed. If the member is not eligible, an application error is raised indicating that the member has exceeded their borrowing limit.

Next, the PL/SQL block checks if the book is available to be borrowed. If the book is not available, an application error is raised indicating that the book is already issued to another member.

If the member is eligible and the book is available, the PL/SQL block updates the TRANSACTION table to add a new transaction record with the details of the book issue. The book status is also updated to 'issued' in the BOOK table, and the last serial number of the book is updated in the TRANSACTION table. A success message is printed to indicate that the book has been issued successfully.

Here is the code –

```
DECLARE
  v_member_type MEMBER.MEMBER_TYPE%TYPE;
  v_max_books_count MEMBER.MAX_BOOKS_COUNT%TYPE;
  v_current_books_count NUMBER;
  v_book_status BOOK.STATUS%TYPE;
BEGIN
  -- Check if member exists and get member details
  SELECT MEMBER_TYPE, MAX_BOOKS_COUNT INTO v_member_type, v_max_books_count
  FROM MEMBER WHERE MEMBER_ID = :member_id;

  -- Check if member is eligible to issue more books
```

```
  SELECT COUNT(*) INTO v_current_books_count
  FROM TRANSACTION WHERE MEMBER_ID = :member_id AND TRANSACTION_TYPE =
'issue';
  IF v_current_books_count >= v_max_books_count THEN
    RAISE_APPLICATION_ERROR(-20001, 'Member has already issued maximum
allowed books.');
  END IF;

  -- Check if book is available and get book details
  SELECT STATUS INTO v_book_status FROM BOOK
  WHERE BOOK_ID = :book_id AND SERIAL_NO = :serial_no;
  IF v_book_status != 'available' THEN
    RAISE_APPLICATION_ERROR(-20002, 'Book is not available for issue.');
  END IF;

  -- Update book status and create transaction record
  UPDATE BOOK SET STATUS = 'issued', LAST_SERIAL = :serial_no
  WHERE BOOK_ID = :book_id AND SERIAL_NO = :serial_no;
  INSERT INTO TRANSACTION (TRANSACTION_ID, TRANSACTION_DATE,
TRANSACTION_TYPE,
    TO_BE_RETURNED, BOOK_ID, SERIAL_NO, MEMBER_ID)
  VALUES ('TRN'||TO_CHAR(SYSDATE, 'YYYYMMDDHH24MISS'), SYSDATE, 'issue',
    SYSDATE + 7, :book_id, :serial_no, :member_id);
  COMMIT;

  DBMS_OUTPUT.PUT_LINE('Book issued successfully.');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20003, 'Member or book not found.');
  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20004, 'Error in issuing book.');
END;
```

PL/SQL Block for Book Return (ii):

The PL/SQL block for book return has been designed to ensure that the book copy being returned was issued to the member who is returning it. The

PL/SQL block takes input parameters for the member ID, book ID, and serial number.

It first checks if there is a transaction record for the book copy that was issued to this member. If no such record is found, an application error is raised indicating that the book copy was not issued to this member.

If a transaction record is found, it updates the transaction record to set the transaction type to 'return', the transaction date to the current date, and the "to_be_returned" date to NULL. The book status is also updated to 'available' in the BOOK table. A success message is printed to indicate that the book has been returned successfully.

Here is the code −

```
DECLARE
  v_member_id MEMBER.MEMBER_ID%TYPE := :member_id;
  v_book_id BOOK.BOOK_ID%TYPE := :book_id;
  v_serial_no BOOK.SERIAL_NO%TYPE := :serial_no;
BEGIN
  -- Check if the book copy was issued to this member
  IF NOT EXISTS (
    SELECT 1 FROM TRANSACTION
    WHERE MEMBER_ID = v_member_id AND BOOK_ID = v_book_id AND SERIAL_NO =
v_serial_no AND TRANSACTION_TYPE = 'issue'
  ) THEN
    RAISE_APPLICATION_ERROR(-20001, 'This book copy was not issued to this
member.');
  END IF;

  -- Update the transaction record for the book return
  UPDATE TRANSACTION SET
    TRANSACTION_TYPE = 'return',
    TRANSACTION_DATE = SYSDATE,
    TO_BE_RETURNED = NULL
  WHERE MEMBER_ID = v_member_id AND BOOK_ID = v_book_id AND SERIAL_NO =
v_serial_no AND TRANSACTION_TYPE = 'issue';

  -- Update the book status to 'available'
  UPDATE BOOK SET STATUS = 'available' WHERE BOOK_ID = v_book_id AND
SERIAL_NO = v_serial_no;

  -- Print success message
```

```
  DBMS_OUTPUT.PUT_LINE('Book returned successfully.');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);
END;
```

Conclusion:

The PL/SQL blocks for book issue and book return have been designed to ensure proper validation and checks are in place to prevent unauthorized book issue or return. These PL/SQL blocks can be integrated into a larger library management system to enable librarians and library users to borrow and return books in a secure and efficient manner.

Q NO. 7

This PL/SQL block is designed to find the information of the pending deliveries for the orders placed between two user-input dates. The block starts by accepting the two dates as input from the user. It then initializes a counter variable for pending deliveries to 0.

A loop is then used to iterate through each order placed between the start and end dates, and for each order, a nested loop is used to iterate through each item in the order. The total quantity of each item that has been ordered is then calculated, along with the total quantity that has been delivered. If the total delivered quantity is less than the ordered quantity, then the item is considered pending, and its details are printed to the console.

The details printed for each pending item include the item number, the ordered quantity, the delivered quantity, the pending quantity, the delivery number, the delivery date, and the pending quantity for the delivery. Finally, the total number of pending deliveries is printed to the console.

Overall, this PL/SQL block allows for efficient tracking of pending deliveries for orders placed between two user-input dates, which can help to streamline delivery logistics and ensure that orders are fulfilled in a timely manner.

Here is the code –

```
DECLARE
    v_start_date DATE := '&enter_start_date'; -- User inputs start date
    v_end_date DATE := '&enter_end_date'; -- User inputs end date
    v_pending_delv_cnt NUMBER := 0; -- Counter for pending deliveries
BEGIN
    FOR order_rec IN (SELECT DISTINCT ORDER_NO FROM ORDERMAST WHERE ORDER_DT
BETWEEN v_start_date AND v_end_date)
```

```
    LOOP
        DBMS_OUTPUT.PUT_LINE('Order No: ' || order_rec.ORDER_NO);
        FOR item_rec IN (SELECT ITEM_NO, SUM(QTY) AS order_qty FROM
ORDERDETAILS WHERE ORDER_NO = order_rec.ORDER_NO GROUP BY ITEM_NO)
        LOOP
            -- Calculate total quantity of items delivered for this order and
item combination
            SELECT NVL(SUM(QTY), 0) INTO v_delivered_qty FROM
DELIVERY_DETAILS WHERE ORDER_NO = order_rec.ORDER_NO AND ITEM_NO =
item_rec.ITEM_NO;


            -- If total delivered quantity is less than ordered quantity,
this item is pending
            IF v_delivered_qty < item_rec.order_qty THEN
                -- Retrieve information about the pending delivery
                SELECT * INTO v_delv_mast_rec FROM DELIVERYMAST WHERE
ORDER_NO = order_rec.ORDER_NO AND ITEM_NO = item_rec.ITEM_NO;
                v_pending_delv_cnt := v_pending_delv_cnt + 1;


                -- Print the details of the pending delivery
                DBMS_OUTPUT.PUT_LINE('  Item No: ' || item_rec.ITEM_NO || ',
Ordered Qty: ' || item_rec.order_qty || ', Delivered Qty: ' ||
v_delivered_qty || ', Pending Qty: ' || (item_rec.order_qty -
v_delivered_qty));
                DBMS_OUTPUT.PUT_LINE('    Delivery No: ' ||
v_delv_mast_rec.DELV_NO || ', Delivery Date: ' || v_delv_mast_rec.DELV_DT ||
', Pending Qty: ' || (item_rec.order_qty - v_delivered_qty));
            END IF;
        END LOOP;
    END LOOP;


    -- Print the total number of pending deliveries
    IF v_pending_delv_cnt = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No pending deliveries found.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Total pending deliveries: ' ||
v_pending_delv_cnt);
    END IF;
END;
```

ASSIGNMENT 5

Introduction

Triggers are database objects in Oracle that can be used to automatically execute code in response to specific events, such as insertions, deletions, or updates in a table. Triggers can be useful for enforcing business rules, auditing database changes, and implementing complex data validation or transformation logic. In this report, we will discuss first question of the assignment about creating triggers in Oracle databases.

Question 1

The first question is about creating a trigger for the **RESULT** and **BACKPAPER** tables. Specifically, we are asked to create a trigger that performs the following actions:

- Whenever a row is inserted or updated in the **RESULT** table, if the marks is 50 or more, delete the corresponding row, if any, from the **BACKPAPER** table.
- Otherwise, insert a row in the **BACKPAPER** table if not already present there.

To implement this trigger, we can use the **CREATE TRIGGER** statement in Oracle. The trigger should be defined as an **AFTER INSERT OR UPDATE** trigger that fires **FOR EACH ROW**. Inside the trigger body, we can use an **IF** statement to check the value of the **MARKS** column in the inserted or updated row. If the value is 50 or more, we can delete any matching row from the **BACKPAPER** table using a **DELETE** statement. Otherwise, we can check if a matching row already exists in the **BACKPAPER** table using a **SELECT** statement with a **WHERE NOT EXISTS** clause. If no matching row is found, we can insert a new row into the **BACKPAPER** table using an **INSERT INTO** statement.

The final trigger code in Oracle should look like this:

```
CREATE TRIGGER result_trigger
AFTER INSERT OR UPDATE ON RESULT
FOR EACH ROW
ENABLE
BEGIN
  IF :NEW.MARKS >= 50 THEN
    DELETE FROM BACKPAPER WHERE ROLL = :NEW.ROLL AND SCODE = :NEW.SCODE;
  ELSE
    INSERT INTO BACKPAPER (ROLL, SCODE)
      SELECT :NEW.ROLL, :NEW.SCODE
      WHERE NOT EXISTS (
        SELECT 1 FROM BACKPAPER
        WHERE ROLL = :NEW.ROLL AND SCODE = :NEW.SCODE
      );
  END IF;
END;
```