**Advanced Database COMP214 – Section 009**

**Group Project**

**Due Date: Midnight of December -04, 2023(Monday) [28 Marks]**

**Purpose:** The purpose of this assignment is to help you:

• Apply your comprehensive database knowledge to solve practical problem

• Become familiar with Cursors, Exceptions, Procedures, Functions, Packages, Triggers, etc.

**Instructions**: Be sure to read the following general instructions carefully:

This is a group-based project. Only one group member submits the solution **through the dropbox.** Your submission should include script file(s) to create table(s) and populate sample data to the database, ERD of your database, PL/SQL code and the screenshot of code execution result, the submission must be named according to the following rule: **Group#lastname\_firstname.doc**. e.g., **Group1Patrick\_Smith.doc**

Create database back-end functionality to facilitate a library to provide its service. It provides the following functionalities to user and staffs.

|  |  |
| --- | --- |
| **User** | **Library Staff** |
| Search book  Borrow book  Return book  Reserve book | Add book  Delete book  Generate report  Notify user |

**Search book:** User performs search operation based on book title or book id. If book is available display the message <Count> copies of <Title of the book> is available <location> (Implement using function)

Example: **5** copies of **PLSQL Programming** is available in **A12R10**

**Borrow book:** When a book is borrowed by the user, the count is decreased by one from the **Books** table and this is entered into **Borrowed\_books** table with relevant user details. (Use trigger for **Borrowed\_books** entry)

**Return book:** When a book is returned by the user, the count is increased by one in the **Books** table.( Use trigger to notify the user if the book is reserved by the customer)

**Reserve book:** When a book is reserved by the user, the reserve flag is set to TRUE and this book can not be borrowed by other users if the count is 1. Use trigger to enter into **Reserved\_books** table with relevant user details.

**Add book:** Library staff add book(s) into **Books** table with all necessary fields. While adding the existing book it just increases the count. You should perform Add book operation for both new and existing books. Use function which returns the updated count of the books.

**Delete book:** Library staff delete book(s) from Books table with book\_id and count fields. (Use procedure with book\_id and count as arguments)

**Generate Report:** Library staff generates report based on following categories:

Books borrowed

Books reserved

**Notify user:** Library staff notifies the user when the reserved book is available. If user borrows the reserved book it has to be removed from Reserved\_books table.

Draw an ER Diagram for the database. **(2 Marks)**

Create required tables with suitable fields to perform the data base operations. **(2 Marks)**

Populate at least TEN rows in to Books table. **(2 Marks)**

Create two packages for User and Staff functionalities. **(4 Marks)**

Call the package elements to perform the following operations. **(8 Marks)**

Search book

Borrow book

Return book

Reserve book

Add book

Delete book

Generate report

Notify user

Use triggers to notify user on reserved book availability. **(2 Marks)**

Use cursors to generate report operation. **(2 Marks)**

Viva **(6 Marks)**

***Note: Display suitable messages after each operation. You should populate your tables with meaningful data.***

**Screenshots**

**Entity Relationship Diagram**

**A diagram of a computer program

Description automatically generated with medium confidence**

**USERS table**

**A screenshot of a computer

Description automatically generated**

**LIBRARY\_BOOKS table (already borrowed and reserved some books)**

**A screenshot of a computer

Description automatically generated**

**BORROWED\_BOOKS table (borrowed some books)**

**A screenshot of a computer

Description automatically generated**

**RESERVED\_BOOKS table (reserved some books)**

**A screenshot of a computer

Description automatically generated**

**USERPACKAGE FUNCTIONS AND PROCEDURES**

**Procedures will be performed in a way that will exhibit program functionalities the most.**

**Searching for a book**

**A screenshot of a computer

Description automatically generated**

**Reserving a book (reserve the book 2 times by different users)**

**A screenshot of a computer error

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Trying to borrow a reserved book that only has one copy left.**

**A screenshot of a computer message

Description automatically generated**

**Borrowing a book (the borrower is the same person who reserved the book first)**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated** **A screenshot of a computer

Description automatically generated**

**Returning a book**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**The book became available but there was another person (USER\_ID 3 – Miha) who reserved it, so they get notified.**

**Others still can’t borrow the book.**

**A screenshot of a computer screen

Description automatically generated**

**But Miha Can**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**STAFFPACKAGE Procedures**

**Table before STAFFPACKAGE procedures**

**A screenshot of a computer

Description automatically generated**

**Adding books**

**A screenshot of a computer program

Description automatically generated**

**Deleting books**

**A screenshot of a computer

Description automatically generated**

**Generating Report**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**Notify User has been showed previously in “Returning a book”. Snippet is from USERPACKAGE.RETURNBOOK()**

**A close up of text

Description automatically generated**

**Code for table creation and data population**

|  |
| --- |
| **CREATE SEQUENCE BOOK\_ID\_SEQ**  **START WITH 1000**  **INCREMENT BY 1**  **MAXVALUE 9999;**    **CREATE SEQUENCE B\_BOOKID\_SEQ**  **START WITH 100**  **INCREMENT BY 1**  **MAXVALUE 999;**    **CREATE SEQUENCE USERS\_SEQ**  **START WITH 1**  **INCREMENT BY 1**  **MAXVALUE 999;**    **CREATE SEQUENCE R\_BOOKID\_SEQ**  **START WITH 100**  **INCREMENT BY 1**  **MAXVALUE 999;**    **CREATE TABLE USERS (**  **USER\_ID NUMBER(5) DEFAULT USERS\_SEQ.NEXTVAL PRIMARY KEY,**  **USER\_NAME VARCHAR2(50)**  **);**    **CREATE TABLE LIBRARY\_BOOKS (**  **BOOK\_ID NUMBER(4) DEFAULT BOOK\_ID\_SEQ.NEXTVAL PRIMARY KEY,**  **BOOK\_NAME VARCHAR2(30),**  **BOOK\_LOC VARCHAR2(6),**  **BOOK\_COUNT NUMBER(2),**  **RESERVED VARCHAR2(5) DEFAULT 'FALSE'**  **);**  **CREATE TABLE BORROWED\_BOOKS (**  **B\_BOOK\_ID NUMBER(3) DEFAULT B\_BOOKID\_SEQ.NEXTVAL PRIMARY KEY,**  **BOOK\_ID NUMBER(6),**  **BOOK\_NAME VARCHAR2(30),**  **BORROWER\_ID NUMBER(5),**    **CONSTRAINT LIB\_BOOKS\_FK\_BBOOKS**  **FOREIGN KEY (BOOK\_ID) REFERENCES LIBRARY\_BOOKS(BOOK\_ID),**    **CONSTRAINT USER\_FK\_BBOOKS**  **FOREIGN KEY (BORROWER\_ID) REFERENCES USERS(USER\_ID)**  **);**  **CREATE TABLE RESERVED\_BOOKS (**  **R\_BOOK\_ID NUMBER(3) DEFAULT R\_BOOKID\_SEQ.NEXTVAL PRIMARY KEY,**  **BOOK\_ID NUMBER(6),**  **BOOK\_NAME VARCHAR2(30),**  **RESERVER\_ID NUMBER(5),**    **CONSTRAINT LIB\_BOOKS\_FK\_RBOOKS**  **FOREIGN KEY (BOOK\_ID) REFERENCES LIBRARY\_BOOKS(BOOK\_ID),**    **CONSTRAINT USER\_FK\_RBOOKS**  **FOREIGN KEY (BORROWER\_ID) REFERENCES USERS(USER\_ID)**  **);**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES ('PLSQL Programming', 'A12R10', 25);**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES ('WEB-Development', 'B13R20', 30);**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES ('Canadian History', 'C14R30', 32);**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES ('Java Programming', 'D15R40',1);**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES ('Cook Book', 'E16R50', 75);**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES ('Biology-101', 'F17R60', 28);**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES ('Physics-2', 'G18R70', 27);**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES ('Astronomy-5vol', 'H19R80', 30);**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES ('Game-Programming', 'I20R90', 33);**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES ('Quantum Physics', 'J21R20', 29);**  **INSERT INTO USERS (USER\_NAME)VALUES ('Antonio');**  **INSERT INTO USERS (USER\_NAME)VALUES ('Carlo');**  **INSERT INTO USERS (USER\_NAME)VALUES ('Miha');**  **INSERT INTO USERS (USER\_NAME)VALUES ('Cathlyn');**  **INSERT INTO USERS (USER\_NAME)VALUES ('Samuel');** |

**PL/SQL Code**

|  |
| --- |
| **-- Run first before doing anything in each session**  **SET SERVEROUTPUT ON;**  **SELECT R\_BOOKID\_SEQ.NEXTVAL FROM DUAL;**  **SELECT B\_BOOKID\_SEQ.NEXTVAL FROM DUAL;**  **-- USER PACKAGE STARTS HERE**  **CREATE OR REPLACE PACKAGE USERPACKAGE AS**  **FUNCTION SEARCHBOOK(BOOK\_ID NUMBER) RETURN VARCHAR2;**  **PROCEDURE BORROWBOOK (BOOK\_ID NUMBER, USER\_ID NUMBER);**  **PROCEDURE RETURNBOOK (BOOK\_ID NUMBER, USER\_ID NUMBER);**  **PROCEDURE RESERVEBOOK (BOOK\_ID NUMBER, USER\_ID NUMBER);**  **END USERPACKAGE;**  **CREATE OR REPLACE PACKAGE BODY USERPACKAGE AS**    **FUNCTION SEARCHBOOK(BOOK\_ID NUMBER) RETURN VARCHAR2 IS**  **AVAILABLE NUMBER(2);**  **BOOK\_TITLE VARCHAR2(30);**  **BOOK\_LOCATION VARCHAR2(6);**  **BEGIN**  **SELECT BOOK\_COUNT INTO AVAILABLE FROM LIBRARY\_BOOKS LB WHERE LB.BOOK\_ID = SEARCHBOOK.BOOK\_ID;**  **SELECT BOOK\_NAME INTO BOOK\_TITLE FROM LIBRARY\_BOOKS LB WHERE LB.BOOK\_ID = SEARCHBOOK.BOOK\_ID;**  **SELECT BOOK\_LOC INTO BOOK\_LOCATION FROM LIBRARY\_BOOKS LB WHERE LB.BOOK\_ID = SEARCHBOOK.BOOK\_ID;**    **-- If there's no book that has the id that the user gave**  **IF AVAILABLE = 0 THEN**  **RETURN 'Book ID not found.';**  **ELSE**  **RETURN AVAILABLE || ' copies of ' || BOOK\_TITLE || ' is available at ' || BOOK\_LOCATION;**  **END IF;**  **END SEARCHBOOK;**    **----------------------------------------------------------------------------------------------------------------**    **PROCEDURE BORROWBOOK (BOOK\_ID NUMBER, USER\_ID NUMBER) IS**  **B\_BOOK\_SEQ NUMBER; -- Checks the current number of the sequence B\_BOOK\_ID**  **BOOK\_NAME VARCHAR2(30); -- Book name**  **IS\_RESERVED VARCHAR2(5); -- False if the book is not reserved**  **COPY\_COUNT NUMBER; -- The count of available book copies**  **USER\_BORROWED VARCHAR2(5) := 'FALSE'; -- False if the user already borrowed the book they're trying to borrow**  **BORROWED VARCHAR2(5) := 'FALSE'; -- False if the book is successfully borrowed**  **RESERVED\_COUNT NUMBER; -- Number of copies of the book that is reserved in the reserved\_book table**  **PRIORITY\_ID NUMBER; -- ID of the first person to reserve (the one with the lowest r\_book\_id)**  **BEGIN**  **SELECT B\_BOOKID\_SEQ.CURRVAL INTO B\_BOOK\_SEQ FROM DUAL;**  **SELECT BOOK\_NAME, RESERVED, BOOK\_COUNT INTO BOOK\_NAME, IS\_RESERVED, COPY\_COUNT FROM LIBRARY\_BOOKS WHERE LIBRARY\_BOOKS.BOOK\_ID = BORROWBOOK.BOOK\_ID;**    **-- Loop checks if the BOOK\_ID given is already borrowed by the user**  **FOR REC IN (SELECT BOOK\_ID, BORROWER\_ID FROM BORROWED\_BOOKS) LOOP**  **IF REC.BOOK\_ID = BOOK\_ID AND REC.BORROWER\_ID = USER\_ID THEN**  **USER\_BORROWED := 'TRUE';**  **END IF;**  **END LOOP;**    **-- If it's not reserved or there are more than 1 copies then continue**  **IF IS\_RESERVED = 'FALSE' OR COPY\_COUNT > 1 THEN**  **-- If the copy count is not 0 and the user hasn't borrowed the book he's trying to borrow yet, then let the user borrow the book**  **IF COPY\_COUNT <> 0 AND USER\_BORROWED = 'FALSE' THEN**  **UPDATE LIBRARY\_BOOKS SET BOOK\_COUNT = BOOK\_COUNT - 1 WHERE LIBRARY\_BOOKS.BOOK\_ID = BORROWBOOK.BOOK\_ID;**  **UPDATE BORROWED\_BOOKS SET BORROWER\_ID = USER\_ID WHERE B\_BOOK\_ID = (B\_BOOK\_SEQ + 1);**  **DBMS\_OUTPUT.PUT\_LINE('You have successfully borrowed the book ' || BOOK\_NAME || '. Your Borrowing Book ID is ' || (B\_BOOK\_SEQ + 1));**  **BORROWED := 'TRUE';**  **END IF;**  **ELSE -- IF IT IS RESERVED AND THERE'S ONLY 1 COPY LEFT, CHECK IF THE BORROWER IS THE SAME PERSON WHO RESERVED THE BOOK**  **-- If the copy count is not 0 and the user hasn't borrowed the book he's trying to borrow yet, then let the user borrow the book**  **IF COPY\_COUNT <> 0 AND USER\_BORROWED = 'FALSE' THEN**  **-- Select the first reserver of the book to be prioritized and notified if the book has been returned**  **SELECT RESERVER\_ID INTO PRIORITY\_ID**  **FROM (**  **SELECT RESERVER\_ID, R\_BOOK\_ID**  **FROM RESERVED\_BOOKS**  **WHERE BOOK\_ID = BORROWBOOK.BOOK\_ID**  **ORDER BY R\_BOOK\_ID**  **FETCH FIRST 1 ROW ONLY);**    **FOR REC IN (SELECT R\_BOOK\_ID, BOOK\_ID, RESERVER\_ID FROM RESERVED\_BOOKS) LOOP**  **-- If the user trying to borrow is the one that has the priority, then allow them to borrow the book**  **IF REC.BOOK\_ID = BOOK\_ID AND REC.RESERVER\_ID = USER\_ID AND REC.RESERVER\_ID = PRIORITY\_ID THEN -- IF THE BORROWER IS ALSO THE RESERVER**  **-- DO AN IF HERE TO CHECK IF THE LOWEST R\_BOOK\_ID IS THE ID OF THE PERSON BORROWING**  **UPDATE LIBRARY\_BOOKS SET BOOK\_COUNT = BOOK\_COUNT - 1 WHERE LIBRARY\_BOOKS.BOOK\_ID = BORROWBOOK.BOOK\_ID;**  **UPDATE BORROWED\_BOOKS SET BORROWER\_ID = USER\_ID WHERE B\_BOOK\_ID = (B\_BOOK\_SEQ + 1);**  **DELETE FROM RESERVED\_BOOKS WHERE R\_BOOK\_ID = REC.R\_BOOK\_ID;**  **DBMS\_OUTPUT.PUT\_LINE('You have successfully borrowed your reserved book ' || BOOK\_NAME || '. Your Borrowing Book ID is ' || (B\_BOOK\_SEQ + 1));**  **BORROWED := 'TRUE';**  **END IF;**  **END LOOP;**    **SELECT COUNT(\*) INTO RESERVED\_COUNT FROM RESERVED\_BOOKS WHERE RESERVED\_BOOKS.BOOK\_NAME = BORROWBOOK.BOOK\_NAME;**  **-- If the book is not reserved by anyone else**  **IF RESERVED\_COUNT = 0 THEN**  **UPDATE LIBRARY\_BOOKS SET RESERVED = 'FALSE' WHERE LIBRARY\_BOOKS.BOOK\_ID = BORROWBOOK.BOOK\_ID;**  **END IF;**  **END IF;**  **END IF;**    **-- Change output depending on the state of USER\_BORROWED**  **IF USER\_BORROWED = 'TRUE' THEN**  **DBMS\_OUTPUT.PUT\_LINE('You have already borrowed this book.');**  **ELSIF BORROWED = 'FALSE' THEN**  **DBMS\_OUTPUT.PUT\_LINE('This book is unavailable for borrowing. You can reserve it and you will be notified when the book is available again.');**  **DBMS\_OUTPUT.PUT\_LINE('If you have already reserved this book, then you are in a queue for the book.');**  **END IF;**    **EXCEPTION**  **WHEN OTHERS THEN**  **DBMS\_OUTPUT.PUT\_LINE('Please check if either BOOK\_ID or USER\_ID are valid IDs. ' || SQLERRM);**  **END BORROWBOOK;**    **----------------------------------------------------------------------------------------------------------------**    **PROCEDURE RETURNBOOK (BOOK\_ID NUMBER, USER\_ID NUMBER) IS**  **RECORD\_COUNT NUMBER;**  **TITLE\_COUNT NUMBER;**  **BEGIN**  **SELECT COUNT(\*) INTO RECORD\_COUNT FROM BORROWED\_BOOKS WHERE BORROWED\_BOOKS.BOOK\_ID = BOOK\_ID AND BORROWED\_BOOKS.BORROWER\_ID = USER\_ID;**  **SELECT COUNT(\*) INTO TITLE\_COUNT FROM BORROWED\_BOOKS WHERE BORROWED\_BOOKS.BOOK\_ID = BOOK\_ID AND BORROWED\_BOOKS.BORROWER\_ID = USER\_ID;**    **-- Checks if the user borrowed the book that they're trying to return**  **IF RECORD\_COUNT <> 0 THEN**  **UPDATE LIBRARY\_BOOKS SET BOOK\_COUNT = BOOK\_COUNT + 1 WHERE LIBRARY\_BOOKS.BOOK\_ID = BOOK\_ID;**  **DELETE FROM BORROWED\_BOOKS WHERE BORROWED\_BOOKS.BOOK\_ID = BOOK\_ID AND BORROWED\_BOOKS.BORROWER\_ID = USER\_ID;**  **DBMS\_OUTPUT.PUT\_LINE('Thank you for returning the book you have borrowed.');**  **STAFFPACKAGE.NOTIFYUSER(BOOK\_ID);**  **ELSE**  **DBMS\_OUTPUT.PUT\_LINE('No record of a user borrowing the book you have entered.');**  **END IF;**  **EXCEPTION**  **WHEN OTHERS THEN**  **DBMS\_OUTPUT.PUT\_LINE('Please check if either BOOK\_ID or USER\_ID are valid IDs. ' || SQLERRM);**  **END RETURNBOOK;**    **----------------------------------------------------------------------------------------------------------------**    **PROCEDURE RESERVEBOOK (BOOK\_ID NUMBER, USER\_ID NUMBER) IS**  **R\_BOOK\_SEQ NUMBER;**  **BOOK\_NAME VARCHAR2(30);**  **BEGIN**  **SELECT R\_BOOKID\_SEQ.CURRVAL INTO R\_BOOK\_SEQ FROM DUAL;**  **SELECT BOOK\_NAME INTO BOOK\_NAME FROM LIBRARY\_BOOKS WHERE LIBRARY\_BOOKS.BOOK\_ID = RESERVEBOOK.BOOK\_ID;**    **-- Triggers the RESERVE\_BOOK\_TRIGGER**  **UPDATE LIBRARY\_BOOKS SET RESERVED = 'TRUE' WHERE LIBRARY\_BOOKS.BOOK\_ID = RESERVEBOOK.BOOK\_ID;**  **-- Sets the RESERVER\_ID to USER\_ID**  **UPDATE RESERVED\_BOOKS SET RESERVER\_ID = USER\_ID WHERE R\_BOOK\_ID = (R\_BOOK\_SEQ + 1);**    **DBMS\_OUTPUT.PUT\_LINE('You have successfully reserved the book ' || BOOK\_NAME || '. Your Reserving Book ID is ' || (R\_BOOK\_SEQ + 1));**  **EXCEPTION**  **WHEN OTHERS THEN**  **DBMS\_OUTPUT.PUT\_LINE('Please check if either BOOK\_ID or USER\_ID are valid IDs. ' || SQLERRM);**  **END RESERVEBOOK;**    **END USERPACKAGE;**  **-- TRIGGER FOR BORROWING BOOKS**  **CREATE OR REPLACE TRIGGER BORROW\_BOOK\_TRIGGER BEFORE UPDATE OF BOOK\_COUNT ON LIBRARY\_BOOKS**  **FOR EACH ROW WHEN (NEW.BOOK\_COUNT < OLD.BOOK\_COUNT)**  **DECLARE**  **BOOK\_ID NUMBER(6);**  **BOOK\_NAME VARCHAR2(30);**  **BEGIN**  **-- Retrieve user and book details from the updated row**  **BOOK\_ID := :NEW.BOOK\_ID;**  **BOOK\_NAME := :NEW.BOOK\_NAME;**    **INSERT INTO BORROWED\_BOOKS (BOOK\_ID, BOOK\_NAME) VALUES (BOOK\_ID, BOOK\_NAME);**  **END;**  **/**  **-- TRIGGER FOR RESERVING BOOKS**  **CREATE OR REPLACE TRIGGER RESERVE\_BOOK\_TRIGGER BEFORE UPDATE OF RESERVED ON LIBRARY\_BOOKS**  **FOR EACH ROW WHEN (NEW.RESERVED = 'TRUE')**  **DECLARE**  **BOOK\_ID NUMBER(6);**  **BOOK\_NAME VARCHAR2(30);**  **BEGIN**  **-- Retrieve user and book details from the updated row**  **BOOK\_ID := :NEW.BOOK\_ID;**  **BOOK\_NAME := :NEW.BOOK\_NAME;**    **INSERT INTO RESERVED\_BOOKS (BOOK\_ID, BOOK\_NAME) VALUES (BOOK\_ID, BOOK\_NAME);**  **END;**  **/**  **-- SEARCH BOOK**  **DECLARE**  **IS\_AVAILABLE VARCHAR2(60);**  **BEGIN**  **IS\_AVAILABLE := USERPACKAGE.SEARCHBOOK(1055);**  **DBMS\_OUTPUT.PUT\_LINE(IS\_AVAILABLE);**  **END;**  **-- BORROW BOOK**  **BEGIN**  **USERPACKAGE.BORROWBOOK(1055, 2);**  **END;**  **-- RETURN BOOK**  **BEGIN**  **USERPACKAGE.RETURNBOOK(1055, 5);**  **END;**  **-- RESERVE BOOK**  **BEGIN**  **USERPACKAGE.RESERVEBOOK(1055, 3);**  **END;**  **-- USER PACKAGE ENDS HERE -------------------------------------------------------------------------------------------**  **-- STAFF PACKAGE STARTS HERE ----------------------------------------------------------------------------------------**  **CREATE OR REPLACE PACKAGE STAFFPACKAGE AS**  **PROCEDURE ADDBOOK(TITLE VARCHAR2, LOCATION VARCHAR2, COUNT NUMBER);**  **PROCEDURE DELETEBOOK(BOOK\_ID NUMBER, DEL\_COUNT NUMBER);**  **PROCEDURE GENERATEREPORT(REPORT\_TYPE VARCHAR2);**  **PROCEDURE NOTIFYUSER(BOOK\_ID NUMBER);**  **END STAFFPACKAGE;**  **CREATE OR REPLACE PACKAGE BODY STAFFPACKAGE AS**  **PROCEDURE ADDBOOK(TITLE VARCHAR2, LOCATION VARCHAR2, COUNT NUMBER) IS**  **BOOK\_ID LIBRARY\_BOOKS.BOOK\_ID%TYPE;**  **BOOK\_COUNT LIBRARY\_BOOKS.BOOK\_COUNT%TYPE;**  **BEGIN**  **-- Check if the book already exists**  **SELECT BOOK\_ID, BOOK\_COUNT INTO BOOK\_ID, BOOK\_COUNT FROM LIBRARY\_BOOKS WHERE BOOK\_NAME = TITLE AND BOOK\_LOC = LOCATION;**  **-- If book exists then update the count**  **UPDATE LIBRARY\_BOOKS SET BOOK\_COUNT = BOOK\_COUNT + COUNT WHERE LIBRARY\_BOOKS.BOOK\_ID = ADDBOOK.BOOK\_ID;**  **DBMS\_OUTPUT.PUT\_LINE('Book count successfully updated');**  **-- If book doesn't exist then insert a new record**  **EXCEPTION**  **WHEN NO\_DATA\_FOUND THEN**  **INSERT INTO LIBRARY\_BOOKS (BOOK\_NAME, BOOK\_LOC, BOOK\_COUNT) VALUES (TITLE, LOCATION, COUNT);**  **DBMS\_OUTPUT.PUT\_LINE('New book successfully added.');**  **END ADDBOOK;**    **PROCEDURE DELETEBOOK(BOOK\_ID NUMBER, DEL\_COUNT NUMBER) IS**  **BK\_COUNT LIBRARY\_BOOKS.BOOK\_COUNT%TYPE;**  **BEGIN**  **-- Retrieve the existing count for the book**  **SELECT BOOK\_COUNT INTO BK\_COUNT FROM LIBRARY\_BOOKS WHERE LIBRARY\_BOOKS.BOOK\_ID = DELETEBOOK.BOOK\_ID;**  **-- Check if the count to delete is greater than the existing count**  **IF DEL\_COUNT > BK\_COUNT THEN**  **DBMS\_OUTPUT.PUT\_LINE('Error: Cannot delete more copies than available.');**  **ELSE**  **-- Update the count**  **UPDATE LIBRARY\_BOOKS SET BOOK\_COUNT = BK\_COUNT - DEL\_COUNT WHERE BOOK\_ID = DELETEBOOK.BOOK\_ID;**  **DBMS\_OUTPUT.PUT\_LINE('Successfully updated the copies available.');**  **END IF;**  **END DELETEBOOK;**    **PROCEDURE GENERATEREPORT(REPORT\_TYPE VARCHAR2) IS**  **-- Declare cursors for each report type**  **CURSOR BORROWED\_CURSOR IS**  **SELECT BOOK\_ID, BOOK\_NAME, BORROWER\_ID FROM BORROWED\_BOOKS;**    **CURSOR RESERVED\_CURSOR IS**  **SELECT BOOK\_ID, BOOK\_NAME, RESERVER\_ID FROM RESERVED\_BOOKS;**    **-- Declare variables to store fetched data**  **BORROWED\_ID BORROWED\_BOOKS.BOOK\_ID%TYPE;**  **BORROWED\_NAME BORROWED\_BOOKS.BOOK\_NAME%TYPE;**  **BORROWER\_ID BORROWED\_BOOKS.BORROWER\_ID%TYPE;**    **RESERVED\_ID RESERVED\_BOOKS.BOOK\_ID%TYPE;**  **RESERVED\_NAME RESERVED\_BOOKS.BOOK\_NAME%TYPE;**  **RESERVER\_ID RESERVED\_BOOKS.RESERVER\_ID%TYPE;**  **BEGIN**  **-- Generate report based on the specified type**  **IF REPORT\_TYPE = 'BORROWED' THEN**  **OPEN BORROWED\_CURSOR;**    **-- Fetch and print records**  **LOOP**  **FETCH BORROWED\_CURSOR INTO BORROWED\_ID, BORROWED\_NAME, BORROWER\_ID;**  **EXIT WHEN BORROWED\_CURSOR%NOTFOUND;**  **DBMS\_OUTPUT.PUT\_LINE('Book ID: ' || BORROWED\_ID || ', Book Name: ' || BORROWED\_NAME || ', User ID: ' || BORROWER\_ID);**  **END LOOP;**  **CLOSE BORROWED\_CURSOR;**    **ELSIF REPORT\_TYPE = 'RESERVED' THEN**  **OPEN RESERVED\_CURSOR;**    **-- Fetch and print records**  **LOOP**  **FETCH RESERVED\_CURSOR INTO RESERVED\_ID, RESERVED\_NAME, RESERVER\_ID;**  **EXIT WHEN RESERVED\_CURSOR%NOTFOUND;**  **DBMS\_OUTPUT.PUT\_LINE('Book ID: ' || RESERVED\_ID || ', Book Name: ' || RESERVED\_NAME || ', User ID: ' || RESERVER\_ID);**  **END LOOP;**    **CLOSE RESERVED\_CURSOR;**    **ELSE**  **DBMS\_OUTPUT.PUT\_LINE('Error: Invalid report type.');**  **END IF;**  **END GENERATEREPORT;**    **PROCEDURE NOTIFYUSER(BOOK\_ID NUMBER) IS**  **RESERVED\_COUNT NUMBER;**  **NEXT\_PRIORITY\_ID NUMBER;**  **BOOK\_NAME LIBRARY\_BOOKS.BOOK\_NAME%TYPE;**  **PRIORITY\_NAME USERS.USER\_NAME%TYPE;**  **BEGIN**  **SELECT COUNT(\*) INTO RESERVED\_COUNT FROM RESERVED\_BOOKS WHERE BOOK\_ID = NOTIFYUSER.BOOK\_ID;**    **IF RESERVED\_COUNT <> 0 THEN**  **-- This is going to select the user who has the priority of borrowing the book**  **SELECT RESERVER\_ID INTO NEXT\_PRIORITY\_ID**  **FROM (**  **SELECT RESERVER\_ID, R\_BOOK\_ID**  **FROM RESERVED\_BOOKS**  **WHERE BOOK\_ID = NOTIFYUSER.BOOK\_ID**  **ORDER BY R\_BOOK\_ID**  **FETCH FIRST 1 ROW ONLY);**    **SELECT USER\_NAME INTO PRIORITY\_NAME FROM USERS WHERE USERS.USER\_ID = NEXT\_PRIORITY\_ID;**  **SELECT BOOK\_NAME INTO BOOK\_NAME FROM LIBRARY\_BOOKS WHERE BOOK\_ID = NOTIFYUSER.BOOK\_ID;**    **DBMS\_OUTPUT.PUT\_LINE('Mr./Ms. ' || PRIORITY\_NAME || ', your reserved book ' || BOOK\_NAME || ' is available for borrowing.');**  **END IF;**  **END NOTIFYUSER;**    **END STAFFPACKAGE;**  **/**  **-- ADD BOOK**  **BEGIN**  **STAFFPACKAGE.ADDBOOK('PLSQL Programming', 'A12R10', 3);**  **END;**  **-- DELETE BOOK**  **BEGIN**  **STAFFPACKAGE.DELETEBOOK(1056, 26);**  **END;**  **-- GENERATE REPORT**  **BEGIN**  **STAFFPACKAGE.GENERATEREPORT('BORROWED');**  **END;** |