Task (1)
"Advanced Library Management System"

Description:

You are tasked with enhancing a library management system by implementing features that automate operations, handle errors, and generate useful reports. You will be working with PL/SQL procedures, functions, triggers, cursors, and transaction handling.

Tables & Relationships

Books: (id, title, author, availability, type_id) Tracks book details and availability (available or borrowed).
Students: (id, name, membership_status) Contains information about library members (active or suspended).
BorrowingRecords: (id, book_id, student_id, borrow_date, return_date, status) Logs all borrowing and return transactions for students.
Penalties: (id, student_id, amount, reason) Records late fees and other penalties for students.
AuditTrail: (id, table_name, operation, old_data, new_data, timestamp) Logs updates or deletions to keep track of changes made to any table.
NotificationLogs: (id, student_id, book_id, overdue_days, notification_date) Logs overdue notifications sent to students.
BookTypes: (id, type_name, fee_rate) type_name, e.g, [regular book, reference book].

Features to be covered

1. **User Management and Privileges**
   - Create a Manager User and grant them the role to create two users. Let User 1 create the Books and BookTypes tables. Let User 2 insert 5 rows of book details and their respective types into the tables.
   - Write a PL/SQL Procedure to automatically log the creation of new database users into the UserCreationLog table, capturing the username, the user who created them, and the timestamp.
2. **Overdue Notifications:**
   - Write a PL/SQL procedure that identifies students with overdue books and logs notification details into the NotificationLogs table.
   - Check for overdue books (e.g., if the return date is more than 7 days past).
   - Each entry should include the student ID, book ID, overdue days, and the current date as the notification date.
3. **Dynamic Late Fee Calculation:**
   - Write a PL/SQL function that dynamically calculates late fees for books based on the number of overdue days and the type of book.
   - The function should take the borrow_record_id as input, calculate overdue days, and then determine the fee based on the book type (found in the Books table).
   - The fee calculation should vary, e.g., a flat fee of $1 per day for regular books and $2 per day for reference books.
   - The function should save the data into the penalties table, then return the calculated fee amount.
4. **Borrowing Validation Trigger:**
   - Create a PL/SQL trigger that prevents a student from borrowing a book if they have overdue books or have reached the borrowing limit.
   - Before a new borrowing record is inserted, the trigger should check if the student has overdue books (using BorrowingRecords) or if they have already borrowed the maximum number of books (e.g., 3)
   - The trigger should raise an exception and prevent the borrowing operation from proceeding.
5. **Audit Trail for Borrowing Records:**
   - Create a BEFORE UPDATE and BEFORE DELETE trigger that logs any modifications or deletions made to borrowing records into the AuditTrail table. The trigger should capture the table name, operation type (INSERT, UPDATE, DELETE), and both the old and new data for each operation. Ensure that the AuditTrail table stores the timestamp of the operation.
6. **Borrowing History Report:**
   - Write a PL/SQL cursor that retrieves a student's borrowing history, including overdue books and penalties. The cursor should take a student_id as input.

- It should display a list of books borrowed by the student, including the book title, borrow date, return date, status (overdue or on time), and any associated penalties.
- The cursor should also include the penalty from the Penalties table if it exists.

7. **Safe Return Process with Transactions:**
   - Write a PL/SQL block that handles returning multiple books at once, using transactions to ensure data integrity. The block should allow for the return of multiple books in a single transaction. It should first check if the student has any overdue penalties.
   - If any errors occur during the process, the block should use ROLLBACK to undo all changes.
   - If no errors occur, the books should be marked as returned in the BorrowingRecords table, and any penalties should be updated in the Penalties table.
8. **Books Availability Report:**
   - Write a PL/SQL block to generate a detailed report on the availability of books in the library.
   - The block should retrieve all books and their current availability status (whether they are available, borrowed, or overdue)
   - It should also display the student who has the book (if borrowed) and the overdue days (if applicable).
   - The report should be displayed in a readable format, potentially using DBMS_OUTPUT.
9. **Automated Suspension:**
   - Create a PL/SQL procedure to automatically suspend students who have penalties above a certain threshold (e.g., $50).
   - The procedure should check all students in the Penalties table for unpaid penalties. If a student's total unpaid penalties exceed the threshold, the procedure should set their membership_status to "suspended" in the students' table.
10. **Advanced Data Integrity and Reporting:**
    - Write a PL/SQL Function that calculates and returns the total number of books currently borrowed by all students (i.e., books in BorrowingRecords with a status of 'Borrowed' or return_date is NULL and status is not 'Returned').
    - Create a Trigger on the BorrowingRecords table that automatically updates the availability status of the corresponding book in the Books table to 'Available' whenever a borrowing record's status is updated to 'Returned'.

Bonus

11. **Blocker-Waiting Situation**
    - Demonstrate generating a blocker-waiting situation using two transactions by User 1 and User 2. User 1 locks the BorrowingRecords table while updating a borrowing record's status, and User 2 tries to simultaneously insert a penalty for the same borrowing record in the Penalties table.

12. **Identifying Blocker and Waiting Sessions**

- Identify the sessions in the blocker-waiting situation created in Question 10 using SID and SERIAL# for both the blocker and waiting sessions. Display the session details and explain how the waiting session can be resolved.

Instructions

- All team members should be aware of every task, as each member may be asked randomly about any task. Additionally, you must have a clear understanding of all the concepts studied in Oracle.
- All scripts should be placed in a single Oracle file with clear comments and documentation explaining the logic behind each point.
- **Prohibition of AI Tools: The use of AI tools such as ChatGPT or similar platforms is strictly prohibited. Submissions will be evaluated with AI detection tools. Teams found using AI tools will receive a zero for this task. Ensure the work reflects your own understanding and effort.**