Database Project Part 2

Advanced Queries, Views, and Stored Procedures

Section 1: Complex Queries with Joins

```sql
--5. Book Popularity Report

SELECT
 B.Title,
 B.ISBN,
 B.Genre,
 COUNT(L.Loan_ID) AS Times_Loaned, --عدد مرات الإعارة
 AVG(R.Rating) AS Average_Rating --متوسط التقييمات
 FROM Book B
 JOIN Loan L ON B.Book_ID = L.Book_ID
 LEFT JOIN Review R ON B.Book_ID = R.Book_ID
 GROUP BY B.Title, B.ISBN, B.Genre
HAVING COUNT(L.Loan_ID) >= 3 --تستخدم مع الدوال التجميعية HAVING
```

Purpose: Identify books that are **most frequently borrowed** and their average ratings.

Explanation:

- Shows books loaned **at least 3 times**.
- Counts total loans per book (`Times_Loaned`).
- Calculates average review ratings (`Average_Rating`).
- `HAVING` filters aggregate results.

```sql
--6. Member Reading History

SELECT
M.Full_Name,
B.Title,
L.Loan_Date,
L.Return_Date,
R.Rating,
R.Comments
FROM Member M
LEFT JOIN Loan L ON M.Member_ID = L.Member_ID --حتى الأعضاء الذين لم يستعيروا يظهروا
LEFT JOIN Book B ON L.Book_ID = B.Book_ID
LEFT JOIN Review R --الربط المزدوج في Review يمنع ظهور تقييمات خاطئة
ON R.Book_ID = B.Book_ID
AND R.Member_ID = M.Member_ID
ORDER BY M.Full_Name, L.Loan_Date  --الترتيب يخلي القراءة أسهل
```

Purpose: Provide a **complete borrowing history** for each member along with any reviews they wrote.

Explanation:

- Displays all borrowed books, including **currently borrowed and returned**.
- Includes reviews (`Rating` and `Comments`).
- `LEFT JOIN` ensures even members with no loans appear.
- Orders by member name and loan date for clarity.

```sql
--7. Revenue Analysis by Genre

SELECT
B.Genre,
COUNT(DISTINCT L.Loan_ID) AS Total_Loans, --DISTINCT يمنع تكرار القرض
SUM(P.Amount) AS Total_Fines, --إجمالي الغرامات
AVG(P.Amount) AS Average_Fine --متوسط الغرامة
FROM Book B
JOIN Loan L ON B.Book_ID = L.Book_ID
LEFT JOIN Payment P ON L.Loan_ID = P.Loan_ID
GROUP BY B.Genre
```

Purpose: Analyze **financial performance per book genre**, including loans and fines.

Explanation:

- Counts total loans per genre (`Total_Loans`).
- Calculates total fines collected (`Total_Fines`) and average fine (`Average_Fine`).
- Helps identify which genres generate the most revenue.
- `DISTINCT` avoids double-counting loans.

```sql
--12. Payment Pattern Analysis:

SELECT
Method AS Payment_Method,
COUNT(Payment_ID) AS Number_Of_Transactions,
SUM(Amount) AS Total_Collected, --SUM(Amount) and AVG(Amount) calculate total and average values.
AVG(Amount) AS Average_Payment,
(SUM(Amount) * 100.0 / (SELECT SUM(Amount) FROM Payment))
AS Percentage_Of_Total_Revenue
FROM Payment
GROUP BY Method --GROUP BY Method aggregates payments by payment method.
--The subquery computes total revenue across all payment methods
--The percentage shows how significant each payment method is
```

This query analyzes how payments are made in the library system.
It shows:

- Which payment methods are used (Credit Card, Cash, Pending, else.)
- How many payments were made using each method
- How much money each method collected
- The average payment amount
- How important each method is compared to total revenue (percentage)

```sql
--3. Overdue Loans with Member Details:

SELECT
M.Full_Name,
M.Phone_Number,
B.Title,
Lib.Name AS Library_Name,
DATEDIFF(DAY, L.Due_Date, GETDATE()) AS Days_Overdue, -- يحسب عدد أيام التأخير DATEDIFF
ISNULL(SUM(P.Amount), 0) AS Total_Fine_Paid -- يحول NULL 0 إلى ISNULL
FROM Loan L
JOIN Member M ON L.Member_ID = M.Member_ID
JOIN Book B ON L.Book_ID = B.Book_ID
JOIN Library Lib ON B.Library_ID = Lib.Library_ID
LEFT JOIN Payment P ON L.Loan_ID = P.Loan_ID --لأن بعض القروض لم تُدفع غرامتها LEFT JOIN Payment
WHERE L.Status = 'Overdue'
GROUP BY M.Full_Name, M.Phone_Number, B.Title, Lib.Name, L.Due_Date --ضروري بسبب  SUM
```

This query retrieves detailed information about overdue loans in the library system. It focuses on identifying members who have not returned borrowed books on time, calculating the number of overdue days, and determining any fines paid for those loans.

```sql
--14. vw_LibraryStatistics:

CREATE VIEW vw_LibraryStatistics AS
SELECT
    L.Library_ID,
    L.Name AS Library_Name,
    COUNT(DISTINCT B.Book_ID) AS Total_Books, --COUNT(DISTINCT ...) prevents duplicate counting due to joins
    SUM(CASE WHEN B.IsAvailable = 1 THEN 1 ELSE 0 END) AS Available_Books, --CASE WHEN used for conditional aggregation
    COUNT(DISTINCT Lo.Member_ID) AS Total_Members,
    SUM(CASE WHEN Lo.Status IN ('Issued', 'Overdue') THEN 1 ELSE 0 END) AS Active_Loans,
    COUNT(DISTINCT S.Staff_ID) AS Total_Staff,
    ISNULL(SUM(P.Amount), 0) AS Total_Fine_Revenue --ISNULL() ensures revenue shows 0 instead of NULL
FROM Library L
LEFT JOIN Book B ON L.Library_ID = B.Library_ID --LEFT JOIN ensures libraries appear even if some data is missing
LEFT JOIN Loan Lo ON B.Book_ID = Lo.Book_ID
LEFT JOIN Payment P ON Lo.Loan_ID = P.Loan_ID
LEFT JOIN Staff S ON L.Library_ID = S.Library_ID
GROUP BY L.Library_ID, L.Name
--Provides a single, comprehensive statistical view per library
```

The `vw_LibraryStatistics` view provides a comprehensive statistical overview of each library. It consolidates operational, administrative, and financial data into a single, reusable view to support reporting and management of decision-making.

```sql
--9. Member Engagement Metrics:

SELECT
M.Full_Name,
COUNT(L.Loan_ID) AS Total_Borrowed, --COUNT(L.Loan_ID) counts how many books each member has borrowed
SUM(CASE WHEN L.Status IN ('Issued', 'Overdue') THEN 1 ELSE 0 END) AS Currently_On_Loan, --SUM(CASE WHEN ...) counts only currently active loans.
ISNULL(SUM(P.Amount), 0) AS Total_Fines_Paid, --ISNULL() replaces NULL fine values with zero.
AVG(R.Rating) AS Average_Rating --AVG(R.Rating) calculates the average rating given by the member.
FROM Member M
JOIN Loan L ON M.Member_ID = L.Member_ID --JOIN Loan ensures only members with borrowing activity are included.
LEFT JOIN Payment P ON L.Loan_ID = P.Loan_ID
LEFT JOIN Review R ON M.Member_ID = R.Member_ID
GROUP BY M.Full_Name
```

Purpose of the Query:

- To evaluate member engagement and activity within the library system.
- To measure borrowing behavior, financial contribution, and review participation.

```sql
--8. Monthly Loan Statistics:

SELECT
DATENAME(MONTH, Loan_Date) AS Month_Name,
COUNT(Loan_ID) AS Total_Loans,
SUM(CASE WHEN Status = 'Returned' THEN 1 ELSE 0 END) AS Total_Returned, -- is used to count loans by status
SUM(CASE WHEN Status IN ('Issued', 'Overdue') THEN 1 ELSE 0 END) AS Active_Loans
FROM Loan
WHERE YEAR(Loan_Date) = YEAR(GETDATE()) --filters loans for the current year only.
GROUP BY DATENAME(MONTH, Loan_Date), MONTH(Loan_Date) -- converts the loan date into a readable month name // DATENAME
ORDER BY MONTH(Loan_Date)
```

purpose of the Query

- To analyze monthly borrowing trends within the library system for the current year.
- To compare issued, returned, and active loans monthly.

How Do Stored Procedures Handle Edge Cases?

- The stored procedures check if a book is available before issuing it.
- Members with overdue loans are not allowed to borrow new books.
- Input values such as `MemberID`, `BookID`, and `LoanID` are validated before processing.
- A book cannot be returned twice for the same loan.
- Overdue fines are calculated only when the return date is after the due date.
- No fine is added if the book is returned on time.
- `ISNULL ()` is used to avoid NULL values in calculations.
- Transactions ensure that all related updates succeed or fail together.
- Duplicate records (such as issuing the same book twice) are prevented.
- Errors stop the process safely without damaging the database.

Important Points:

SQL Commands and Their Usage

SQL Command	Purpose (What We Use It For)
`SELECT`	Retrieves data from one or more tables.
`JOIN`	Combines related data from multiple tables.
`LEFT JOIN`	Displays all records from the main table even if related data is missing.
`WHERE`	Filters records based on specific conditions.
`GROUP BY`	Groups rows apply aggregate functions.
`HAVING`	Filters aggregated results after grouping.
`COUNT ()`	Counts the number of rows or records.
`SUM()`	Calculates the total number of numeric values.
`AVG()`	Calculates the average value.
`DISTINCT`	Removes duplicate values from results.
`CASE WHEN`	Applies conditional logic within queries.
`ISNULL()`	Replaces NULL values with a default value (e.g., 0).
`ORDER BY`	Sorts of query results in ascending or descending order.
`DATENAME()`	Extracts readable date parts (e.g., month name).
`DATEDIFF()`	Calculates the difference between two dates.
`YEAR()` / `MONTH()`	Extracts year or month from date values.
`CREATE VIEW`	Creates virtual tables for simplified data access.
`CREATE PROCEDURE`	Creates stored procedures to automate database operations.
`BEGIN TRANSACTION`	Starts a transaction to ensure data consistency.
`COMMIT`	Saves all changes made in a transaction.
`ROLLBACK`	Cancels change if an error occurs.
`IF` / `EXISTS`	Perform validation and conditional checks.

Stored Procedures Testing:

. sp_IssueBook

- **Success:**
  The book is available and the member has no overdue loans, so the book is issued successfully.
- **Error:**
  The book is already borrowed, or the member has overdue loans, so the procedure shows an error and stops.

2. sp_ReturnBook

- **Success:**
  The book is returned on time, the loan status is updated, and no fine is added.
- **Error:**
  A wrong loan ID is entered, so the procedure shows an error and no changes are made.

3. sp_GetMemberReport

- **Success:**
  A valid member ID is entered, and the system shows the member's details, loans, fines, and reviews.
- **Error:**
  An invalid member ID is entered, and no data is returned.

4. sp_MonthlyLibraryReport

- **Success:**
  A valid library, month, and year are entered, and the monthly report is generated.
- **Error:**
  An invalid library ID is entered, so the report returns zero results.