
SQL Project
Library Management System

DATABASE

```sql
--creating data base
CREATE DATABASE LMS;
```

BRANCH TABLE

```sql
DROP TABLE IF EXISTS branch;
CREATE TABLE branch(
    branch_id VARCHAR(10) PRIMARY KEY,
    manager_id VARCHAR(10),
    branch_address VARCHAR(100),
    contact_no VARCHAR(20)
);
```

EMPLOYEES TABLE

```sql
DROP TABLE IF EXISTS employees;
CREATE TABLE employees(
    emp_id VARCHAR(10) PRIMARY KEY,
    emp_name VARCHAR(50),
    position VARCHAR(50),
    salary NUMERIC(10,2),
    branch_id VARCHAR(10) --FK
);
```

MEMBERS TABLE

```sql
DROP TABLE IF EXISTS members;
CREATE TABLE members(
    member_id VARCHAR(10) PRIMARY KEY,
    member_name VARCHAR(50),
    member_address VARCHAR(100),
    reg_date DATE
);
```

BOOKS TABLE

```sql
DROP TABLE IF EXISTS books;
CREATE TABLE books(
    isbn VARCHAR(50) PRIMARY KEY,
    book_title VARCHAR(50),
    category VARCHAR(50),
    rental_price NUMERIC(10,2),
    status VARCHAR(10),
    author VARCHAR(50),
    publisher VARCHAR(50)
);
```

ISSUED STATUS TABLE

```sql
DROP TABLE IF EXISTS issued_status;
CREATE TABLE issued_status(
    hoi VARCHAR(10) PRIMARY KEY,
    issued_member_id VARCHAR(10), --FK
    issued_book_name VARCHAR(50),
    issued_date DATE,
    issued_book_isbn VARCHAR(50), --FK
    issued_emp_id VARCHAR(10) --FK
);
```

RETURN STATUS TABLE

```sql
DROP TABLE IF EXISTS return_status;
CREATE TABLE return_status(
    return_id VARCHAR(50) PRIMARY KEY,
    issued_id VARCHAR(50), --FK
    return_book_name VARCHAR(25),
    return_date DATE,
    return_book_isbn VARCHAR(50) --Foriegn Key
);
```

CREATING DATABASE & TABLES

Solving Questions - Project Tasks

Task 1. Creating a New Book Record

```sql
SELECT * FROM books;
INSERT INTO books (isbn,book_title,category,rental_price,status,author,publisher)
VALUES ('978-1-60129-456-2', 'To Kill a Mockingbird', 'Classic', 6.00, 'yes', 'Harper Lee', 'J.B. Lippincott & Co.');
```

Task 2: Update an Existing Member's Address

```sql
UPDATE members
SET member_address='g2 nellore'
WHERE member_id='C101';
```

Task 3: Delete a Record from the Issued Status Table

```sql
DELETE FROM issued_status
WHERE hoi='IS121';
```

Task 4: Retrieve All Books Issued by a Specific Employee

```sql
SELECT issued_emp_id,issued_book_name
FROM issued_status
WHERE issued_emp_id='E101';
```

Task 5: List Members Who Have Issued More Than One Book
Objective: Use GROUP BY to find members who have issued more than one book.

```sql
SELECT issued_emp_id,COUNT(hoi) AS total_books_issued
FROM issued_status
GROUP BY 1
HAVING COUNT(hoi)>1;
```

Task 6: Create Summary Tables: Used CTAS to generate new tables based on query results - each book and total book_issued_count

```sql
CREATE TABLE book_counts AS
    (
    SELECT book_title,COUNT(ist.hoi) AS total_book_issues FROM books AS b
    INNER JOIN issued_status AS ist
    ON b.book_title = ist.issued_book_name
    GROUP BY 1
    )

SELECT * FROM book_counts;
```

Task 7. Retrieve All Books in a Specific Category:

```sql
SELECT *
FROM books
WHERE category = 'Classic';
```

Task 8: Find Total Rental Income by Category:

```sql
SELECT b.category,b.rental_price AS book_rental_price_one_time, COUNT(hoi) AS no_times_issued,SUM(b.rental_price) AS total_revenue
FROM books AS b
INNER JOIN issued_status AS ist
ON b.isbn = ist.issued_book_isbn
GROUP BY 1,2
ORDER BY 1;
```

Task 9: List Members Who Registered in the Last 180 Days:

```sql
SELECT *
FROM members
WHERE reg_date >= CURRENT_DATE - INTERVAL '180 days';
```

Task 10: List Employees with Their Branch Manager's Name and their branch details:

```sql
SELECT * FROM employees;
SELECT * FROM branch;

SELECT em1.*,br.manager_id AS manager_id,em2.emp_name AS manager
FROM employees AS em1
INNER JOIN branch AS br
ON em1.branch_id = br.branch_id
INNER JOIN employees AS em2
ON em2.emp_id = br.manager_id;
```

Task 11. Create a Table of Books with Rental Price Above a Certain Threshold: rental price more than 5

```sql
CREATE TABLE books_costs_above_rs5 AS
    (
    SELECT * FROM books
    WHERE rental_price >= 5
    );

SELECT * FROM books_costs_above_rs5;
```

Task 12: Retrieve the List of Books Not Yet Returned will use joins, between return and issued status

```sql
SELECT i_s.issued_book_name AS Books_Not_Yet_Returned,i_s.issued_date AS Book_Issued_On
FROM return_status AS r_s
RIGHT JOIN issued_status AS i_s
ON r_s.issued_id = i_s.hoi
WHERE r_s.return_id IS NULL;
```

Advanced Queries

Task 13: Identify Members with Overdue Books

Write a query to identify members who have overdue books (assume a 30-day return period). Display the member's_id, member's name, book title, issue date, and days overdue

```sql
SELECT m.member_id,m.member_name,ist.issued_book_name,ist.issued_date,(CURRENT_DATE - ist.issued_date) AS Over_due
FROM members AS m
INNER JOIN issued_status AS ist
    ON m.member_id=ist.issued_member_id
LEFT JOIN return_status AS r
    ON ist.hoi=r.issued_id
WHERE CURRENT_DATE - ist.issued_date > 30 AND r.return_date IS NULL
ORDER BY 1;
```

Task 14: Update Book Status on Return

Write a query to update the status of books in the books table to "Yes" .when they are returned (based on entries in the return_status table)

```
CREATE OR REPLACE PROCEDURE add_return_records(u_return_id VARCHAR(50),u_hoi VARCHAR(10),u_book_quality VARCHAR(15))
LANGUAGE plpgsql
AS $$

DECLARE
    v_isbn VARCHAR(50);
    v_book_name VARCHAR(50);

BEGIN

    INSERT INTO return_status(return_id,issued_id,return_date,book_quality)
    VALUES (u_return_id,u_hoi,CURRENT_DATE,u_book_quality);

    SELECT issued_book_isbn,issued_book_name
        INTO v_isbn,v_book_name
    FROM issued_status
    WHERE hoi = u_hoi;

    UPDATE books
    SET status = 'yes'
    WHERE v_isbn = isbn;

    RAISE NOTICE 'Thanks for returning the book %',v_book_name;
END;
$$
```

```
--calling function
CALL add_return_records('RS120','IS134','Bad');
```

Task 15: Branch Performance Report

Create a query that generates a performance report for each branch, showing the number of books issued, the number of books returned, and the total revenue generated from book rentals

```sql
CREATE TABLE Branch_performance AS
(
    SELECT br.branch_id,br.manager_id,COUNT(ist.hoi) AS  no_books_issued,COUNT(rs.return_id) AS no_books_return,SUM(bk.rental_price)
    FROM issued_status AS ist
    JOIN employees AS e
        ON e.emp_id = ist.issued_emp_id
    JOIN branch AS br
        ON e.branch_id=br.branch_id
    LEFT JOIN return_status AS rs
        ON rs.issued_id = ist.hoi
    JOIN books AS bk
        ON ist.issued_book_isbn=bk.isbn
    GROUP BY 1,2
);

SELECT * FROM branch_performance;
```

Task 16: CTAS: Create a Table of Active Members

Use the CREATE TABLE AS (CTAS) statement to create a new table active_members containing members who have issued at least one book in the last 2 months.

```sql
CREATE TABLE active_members AS
(
    SELECT * FROM members
    WHERE member_id IN (
                        SELECT DISTINCT(issued_member_id)
                        FROM issued_status
                        WHERE issued_date >= CURRENT_DATE - INTERVAL '2 month'
                       )
)

SELECT * FROM active_members;
```

Task 17: Find Employees with the Most Book Issues Processed

Write a query to find the top 3 employees who have processed the most book issues. Display the employee name, number of books processed, and their branch

```sql
SELECT e.emp_name,COUNT(ist.hoi) AS no_books_issued,b.branch_id
FROM employees AS e
JOIN issued_status AS ist
    ON e.emp_id=ist.issued_emp_id
JOIN branch AS b
    ON e.branch_id=b.branch_id
GROUP BY 1,3
ORDER BY 2 DESC
LIMIT 3;
```

Task 18: Identify Members Issuing High-Risk Books

Write a query to identify members who have issued books atleast once with the status "damaged" in the books table. Display the member name, book title, and the number of times they've issued damaged books

```sql
SELECT m.member_name,ist.issued_book_name,COUNT(ist.hoi) AS no_times_issued
FROM members AS m
JOIN issued_status AS ist
    ON  m.member_id=ist.issued_member_id
JOIN return_status AS rs
    ON ist.hoi = rs.issued_id
WHERE book_quality = 'Damaged'
GROUP BY 1,2;
```

Task 19: Stored Procedure Objective: Create a stored procedure to manage the status of books in a library system.

**Description: Write a stored procedure that updates the status of a book in the library based on its issuance.
The procedure should function as follows: The stored procedure should take the book_id as an input parameter.
The procedure should first check if the book is available (status = 'yes'). If the book is available, it should be issued,
and the status in the books table should be updated to 'no'. If the book is not available (status = 'no'),
the procedure should return an error message indicating that the book is currently not available.**

```
CREATE OR REPLACE PROCEDURE status_of_book(u_issue_id VARCHAR(10),u_issued_member_id VARCHAR(10),u_issued_book_isbn VARCHAR(50),u_issued_emp_id VARCHAR(10))
LANGUAGE plpgsql
AS $$

    DECLARE
    --all variables
    v_status VARCHAR(10);

BEGIN
    --all the code
    --checking book IS if avaliable

    SELECT status
    INTO v_status
    FROM books
    WHERE isbn = u_issued_book_isbn;

    IF v_status='yes' THEN

        INSERT INTO issued_status(hoi, issued_member_id, issued_date, issued_book_isbn, issued_emp_id)
        VALUES (u_issue_id, u_issued_member_id, CURRENT_DATE, u_issued_book_isbn, u_issued_emp_id);

        RAISE NOTICE 'Book records added successfully for book: %', u_issued_book_isbn;

        UPDATE books
        SET status = 'no'
        WHERE isbn = u_issued_book_isbn;

    ELSE
        RAISE NOTICE 'Sorry Book is not avaliable for now: %', u_issued_book_isbn;
    END IF;
END;
$$
```

```
--CALLING FUNCTION
CALL status_of_book('IS206', 'C106', '978-0-330-25864-8', 'E104');
```