SQL Library Management Project

SQL(mySQL) Portfolio Project 1: Library Management

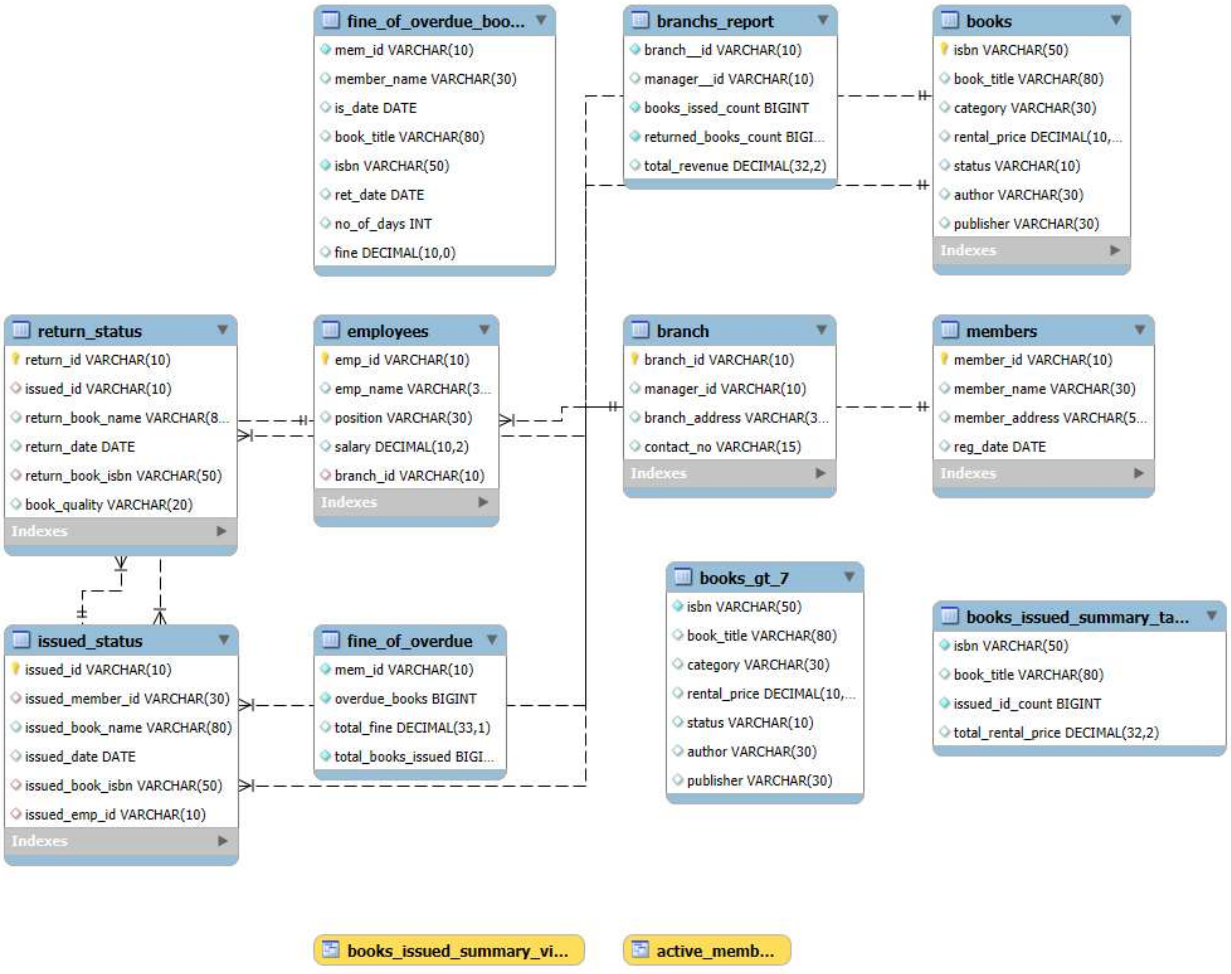
An end-to-end SQL project for Library Project Management System.

This project covers the following important concepts of SQL(MySQL):

- Designing Databases
- Constraints primary key, foreign key
- Data Cleaning: Techniques to clean and prepare the dataset for analysis.
- CRUD Operations,
- Store Procedures
- CTAS in SQL
- Advanced SQL Queries and multiple joins on multiple tables

Creating database and tables

Here is ER Diagram of Library management Database



```sql
drop database if exists library_management;
create database if not exists library_management;
use library_management;

drop table if exists branch;
create table branch (
    branch_id varchar(10),
    manager_id  varchar(10),
    branch_address varchar(30),
    contact_no varchar(15),
    primary key (branch_id)
    );

drop table if exists employees;
create table employees (
    emp_id varchar(10),
    emp_name varchar(30),
    position varchar(30),
    salary decimal(10,2),
    branch_id varchar(10),
    primary key (emp_id),
    foreign key (branch_id) references branch(branch_id)
    );

drop table if exists members;
create table members (
    member_id varchar(10),
    member_name varchar(30),
    member_address varchar(50),
    reg_date date,
    primary key (member_id)
    );
alter table members modify reg_date varchar(15);
alter table members modify reg_date date;

UPDATE members
SET reg_date = STR_TO_DATE(reg_date, '%m/%d/%Y');

drop table if exists books;
create table books (
    isbn varchar(50),
    book_title varchar(80),
    category varchar(30),
    rental_price decimal(10,2),
    status varchar(10),
```

```
    author varchar(30),
    publisher varchar(30),
    primary key(isbn)
    );
alter table books modify category varchar(30);
-- alter table books alter column category type varchar(30); -- command in
PostGreSQL

drop table if exists issued_status;
create table issued_status (
    issued_id varchar(10),
    issued_member_id varchar(30),
    issued_book_name varchar(80),
    issued_date date,
    issued_book_isbn varchar(50),
    issued_emp_id varchar(10),
    primary key (issued_id),
    foreign key (issued_member_id) references members(member_id),
    foreign key (issued_book_isbn) references books(isbn),
    foreign key (issued_emp_id) references employees(emp_id)
    );
-- alter table  issued_status add constraint fk_issued_member_id foreign key
(issued_member_id) references members(member_id);

alter table issued_status modify issued_date varchar(15);
alter table issued_status modify issued_date date;
UPDATE issued_status
SET issued_date = STR_TO_DATE(issued_date, '%m/%d/%Y');

drop table if exists return_status;
create table return_status (
    return_id varchar(10),
    issued_id varchar(10),
    return_book_name varchar(80),
    return_date date,
    return_book_isbn varchar(50),
    primary key (return_id),
    foreign key (return_book_isbn) references books(isbn)
    );
alter table return_status add constraint fk_issued_id foreign key (issued_id)
references issued_status(issued_id);

alter table return_status modify return_date varchar(15);
alter table return_status modify return_date date;
UPDATE return_status
```

```
SET issued_date = STR_TO_DATE(issued_date, '%m/%d/%Y');
```

Inserting data

We have used this code to insert data from csv files. I have attaches the complete dataset in this repository

```
-- inserting sample data into tables
use library_management;
LOAD DATA local INFILE "members.csv"
INTO TABLE members
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(@member_id, @member_name, @member_address, @reg_date)
SET reg_date = DATE_FORMAT(STR_TO_DATE(@reg_date, '%m/%d/%Y'), '%Y-%m-%d');
SET GLOBAL local_infile = 1;
```

Some basic Queries

```
SET sql_safe_updates=0;
SET sql_safe_updates=1;
use library_management;
set foreign_key_checks=0;
set foreign_key_checks=1;
-- truncate table members;
select * from members;
SET GLOBAL local_infile = 1;
SHOW VARIABLES LIKE 'secure_file_priv';
use library_management;
```

Task 1: CRUD Operations

```
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.');
update books set publisher="hamdard" where isbn="978-1-60129-456-2";
delete from books where isbn='978-1-60129-456-2';
SELECT * FROM issued_status
WHERE issued_emp_id = 'E101';
```

Task 2

Select all employees who have issued books more than 1 time.

```
select issued_member_id, count(issued_id) from issued_status group by 1 having
count(issued_id)>1; -- 1 mean issued_member_id column
select issued_emp_id, count(issued_id) from issued_status group by issued_emp_id
having count(issued_id)>1;
```

```
use library_management;
```

Task 3

CTAS- Create a new summary table of how many times a book has been issued?

```
create table books_issued_summary_table as select b.isbn, b.book_title,
count(iss.issued_id) as issued_id_count, sum(b.rental_price) as
total_rental_price
    from books as b join issued_status as iss on b.isbn=iss.issued_book_isbn
    group by b.isbn, b.book_title having issued_id_count>0;
-- creating view
create view books_issued_summary_view as select b.isbn, b.book_title,
count(iss.issued_id) as issued_id_count, sum(b.rental_price) as
total_rental_price
    from books as b join issued_status as iss on b.isbn=iss.issued_book_isbn
    group by b.isbn, b.book_title having issued_id_count>1;

select * from books_issued_summary_view;
select * from books_issued_summary_table;
```

Task 4

Calculate total rental price by each category

```
select b.category as cat, sum(b.rental_price) as tot_price, count(*) from
issued_status as iss join books as b on iss.issued_book_isbn=b.isbn group by cat;
```

Task 5

Identify Members who have registered in last 180 days

```
select * from members;
SELECT * FROM members WHERE reg_date >= current_date() - INTERVAL 180 DAY;
```

Task 6

List employees with their branch manager's names and their branch details

```
select * from branch;
select * from manager;
select em.*, br.manager_id, em2.emp_name as manager_name from employees as em
join branch as br on em.branch_id=br.branch_id join employees as em2 on
br.manager_id=em2.emp_id;
```

Task 7

Books which have been issued but not returned yet

```
-- where rental price is greater than some threshhold
create table books_gt_7 as select * from books where rental_price > 7;
select * from books_gt_7;
-- books which have been issued but not returned yet..
```

```
select iss.issued_id, iss.issued_book_name, rs.return_id from issued_status as
iss left join return_status as rs on iss.issued_id=rs.issued_id where
rs.issued_id is null;
```

Inserting Some More data to database

```
INSERT INTO issued_status(issued_id, issued_member_id, issued_book_name,
issued_date, issued_book_isbn, issued_emp_id) VALUES
('IS151', 'C118', 'The Catcher in the Rye', CURRENT_DATE() - INTERVAL 24
day,  '978-0-553-29698-2', 'E108'),
('IS152', 'C119', 'The Catcher in the Rye', CURRENT_DATE() - INTERVAL 13
day,  '978-0-553-29698-2', 'E109'),
('IS153', 'C106', 'Pride and Prejudice', CURRENT_DATE() - INTERVAL 7 day,  '978-
0-14-143951-8', 'E107'),
('IS154', 'C105', 'The Road', CURRENT_DATE() - INTERVAL 32 day,  '978-0-375-
50167-0', 'E101');
-- adding book_quality column
UPDATE return_status
SET book_quality = 'Damaged'
WHERE issued_id
    IN ('IS110', 'IS115', 'IS116');

select * from return_status;
```

Task 8

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.

```
select
    mem.member_id as mem_id,
    mem.member_name,
    iss.issued_date as is_date,
    bk.book_title, bk.isbn,
    rs.return_date as ret_date,
    DATEDIFF(COALESCE(rs.return_date, CURDATE()), iss.issued_date) AS no_of_days
from
    members as mem
    join issued_status as iss on mem.member_id=iss.issued_member_id
    join books as bk on iss.issued_book_isbn=bk.isbn
    left join return_status as rs on iss.issued_id=rs.issued_id
where
    rs.return_date is null and DATEDIFF(COALESCE(rs.return_date, CURDATE()),
iss.issued_date) > 50;
```

Task 9

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).

```
select * from return_status;

DELIMITER //
CREATE PROCEDURE add_returned_books(
    p_return_id VARCHAR(10),
    p_issued_id VARCHAR(10),
    p_book_quality VARCHAR(20)
)
BEGIN
    DECLARE b_isbn VARCHAR(50);
    DECLARE b_title VARCHAR(50);

    -- Validate input parameters
    IF NOT EXISTS (SELECT 1 FROM issued_status WHERE issued_id = p_issued_id)
THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid issued ID';
    END IF;

    -- Update return status and get book information
    INSERT INTO return_status (return_id, issued_id, return_date, book_quality)
    VALUES (p_return_id, p_issued_id, CURDATE(), p_book_quality);

    SELECT issued_book_isbn, issued_book_name
    INTO b_isbn, b_title
    FROM issued_status
    WHERE issued_id = p_issued_id;

    -- Update book status
    UPDATE books
    SET status = 'yes'
    WHERE isbn = b_isbn;
    SIGNAL SQLSTATE '01000' SET MESSAGE_TEXT = 'Thanks for returning the
book: '

END//
DELIMITER ;
-- drop procedure add_returned_books;
CALL add_returned_books('RS140', 'IS139', 'Good');
```

Task 10

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.

```
create table branchs_report as select br.branch_id as branch__id, br.manager_id
as manager__id, count(iss.issued_id) as books_issed_count, count(rs.return_id) as
returned_books_count, sum(bk.rental_price) as total_revenue from issued_status as
iss
join employees as em on iss.issued_emp_id=em.emp_id join branch as br on
em.branch_id=br.branch_id
left join return_status as rs on iss.issued_id=rs.issued_id join books as bk on
bk.isbn=issued_book_isbn group by 1, 2;

select * from branchs_report;
```

Task 11

Active members who have issued at least one book in previous two months

```
create view active_members as select members.* from members where member_id in
(select distinct issued_member_id from issued_status
where issued_date >= current_date()-  interval 2 month);
-- drop view active_members;
select * from active_members;
```

Task 12

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.

```
select em.emp_name as empolyee_name,  br.*, count(iss.issued_id) as no_of_books
from employees as em
join issued_status as iss on em.emp_id=iss.issued_emp_id
join branch as br on br.branch_id=em.branch_id group by 1,2 order by no_of_books
desc limit 3;
```

Task 13

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.
```
```

```
DELIMITER //
create procedure check_and_issue_book(
    -- p_boook_id varchar(30),
    p_issued_id varchar(10),
    p_issued_member_id varchar(10),
    -- p_issued_book_name varchar(80),
    p_issued_book_isbn varchar(30),
    p_issued_emp_id varchar(10)
)
begin
    declare book_av varchar(10);
    select status into book_av from books where isbn=p_issued_book_isbn;
    if book_av='yes' then
        insert into issued_status (issued_id, issued_member_id, issued_date,
issued_book_isbn, issued_emp_id)
        values(p_issued_id, p_issued_member_id, current_date(),
p_issued_book_isbn, p_issued_emp_id);
        update books set status='no' where isbn=p_issued_book_isbn;
        SIGNAL SQLSTATE '01000' SET MESSAGE_TEXT = 'BOOK issued! Thanks';
    else
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'BOOk is not Available!';
    end if;
END//
DELIMITER ;
drop procedure check_and_issue_book;

CALL check_and_issue_book('IS155', 'C108', '978-0-553-29698-2', 'E104');
call check_and_issue_book('IS156', 'C108', '978-0-375-41398-8', 'E104');

select * from books where isbn='978-0-553-29698-2';
```

Task 14

Identify Members Issuing High-Risk Books

Write a query to identify members who have issued books more than twice with the status "damaged" in the books table.

Display the member name, book title, and the number of times they've issued damaged books.

```
select iss.issued_member_id, mem.member_name, COUNT(CASE WHEN rs.book_quality =
'Damaged' THEN 1 END) as damaged_times
from issued_status as iss
join return_status as rs on iss.issued_id=rs.issued_id
join members as mem on mem.member_id=iss.issued_member_id
group by 1,2 having damaged_times>=2 order by damaged_times desc;
```

Task 15

Create Table As Select (CTAS) Objective: Create a CTAS (Create Table As Select) query to identify overdue books and calculate fines.
Description: Write a CTAS query to create a new table that lists each member and the books they have issued but
not returned within 30 days. The table should include: The number of overdue books.
The total fines, with each day's fine calculated at $0.50.
The number of books issued by each member. The resulting table should show:
Member ID Number of overdue books Total fines

```
create table fine_of_overdue_books as select
    mem.member_id as mem_id,
    mem.member_name,
    iss.issued_date as is_date,
    bk.book_title, bk.isbn,
    rs.return_date as ret_date,
    DATEDIFF(COALESCE(rs.return_date, CURDATE()), iss.issued_date) AS no_of_days
from
    members as mem
    join issued_status as iss on mem.member_id=iss.issued_member_id
    join books as bk on iss.issued_book_isbn=bk.isbn
    left join return_status as rs on iss.issued_id=rs.issued_id
where
    DATEDIFF(COALESCE(rs.return_date, CURDATE()), iss.issued_date) > 50;

alter table fine_of_overdue_books add fine decimal(10) default 0;
set sql_safe_updates=0;
-- set sql_safe_updates=1;
update fine_of_overdue_books set fine = (no_of_days-60)*0.5 where no_of_days>60;
select * from fine_of_overdue_books;

-- Create the fine_of_overdue_books table with aggregated information per member
CREATE TABLE fine_of_overdue AS
SELECT
    mem.member_id AS mem_id,
    COUNT(CASE WHEN DATEDIFF(COALESCE(rs.return_date, CURDATE()),
iss.issued_date) > 30 THEN 1 END) AS overdue_books,
    SUM(CASE WHEN DATEDIFF(COALESCE(rs.return_date, CURDATE()), iss.issued_date)
> 30
             THEN (DATEDIFF(COALESCE(rs.return_date, CURDATE()), iss.issued_date)
- 30) * 0.5
             ELSE 0 END) AS total_fine,
    COUNT(iss.issued_id) AS total_books_issued
FROM
    members AS mem
```

```
    JOIN issued_status AS iss ON mem.member_id = iss.issued_member_id
    LEFT JOIN return_status AS rs ON iss.issued_id = rs.issued_id
GROUP BY
    mem.member_id;

-- Query to check the results
SELECT * FROM fine_of_overdue;
```