ITVEDANT EDUCATION

HYDERABAD Branch.

NAME: RAKESH KUDACHI.

COURSE: MASTER IN FULL STACK DEVELOPER WITH JAVA.

BATCH: 20063.

PROJECT NAME: LIBRARY MANAGEMENT SYSTEM USING SQL.

Title: Library Management System Project

Introduction:

This project demonstrates the implementation of a Library Management System using SQL. It includes creating and managing tables, performing CRUD operations, and executing advanced SQL queries. The goal is to showcase skills in database design, manipulation, and querying.

Objectives:

1. **Set up the Library Management System Database**: Create and populate the database with tables for branches, employees, members, books, issued status, and return status.
2. **CRUD Operations**: Perform Create, Read, Update, and Delete operations on the data.
3. **CTAS (Create Table As Select)**: Utilize CTAS to create new tables based on query results.
4. **Advanced SQL Queries**: Develop complex queries to analyze and retrieve specific data.

Project Structure:

1. Database Setup

- **Database Creation**: Created a database named library_db.
- **Table Creation**: Created tables for branches, employees, members, books, issued status, and return status. Each table includes relevant columns and relationships.

```
CREATE DATABASE library_db;


DROP TABLE IF EXISTS branch;
CREATE TABLE branch
(
```

```
    branch_id VARCHAR(10) PRIMARY KEY,
    manager_id VARCHAR(10),
    branch_address VARCHAR(30),
    contact_no VARCHAR(15)
);
```

-- Create table "Employee"

```
DROP TABLE IF EXISTS employees;
CREATE TABLE employees
(
    emp_id VARCHAR(10) PRIMARY KEY,
    emp_name VARCHAR(30),
    position VARCHAR(30),
    salary DECIMAL(10,2),
    branch_id VARCHAR(10),
    FOREIGN KEY (branch_id) REFERENCES  branch(branch_id)
);
```

-- Create table "Members"

```
DROP TABLE IF EXISTS members;
CREATE TABLE members
(
    member_id VARCHAR(10) PRIMARY KEY,
    member_name VARCHAR(30),
    member_address VARCHAR(30),
    reg_date DATE
);
```

```
-- Create table "Books"
DROP TABLE IF EXISTS books;
CREATE TABLE books
(
        isbn VARCHAR(50) PRIMARY KEY,
        book_title VARCHAR(80),
        category VARCHAR(30),
        rental_price DECIMAL(10,2),
        status VARCHAR(10),
        author VARCHAR(30),
        publisher VARCHAR(30)
);



-- Create table "IssueStatus"
DROP TABLE IF EXISTS issued_status;
CREATE TABLE issued_status
(
        issued_id VARCHAR(10) PRIMARY KEY,
        issued_member_id VARCHAR(30),
        issued_book_name VARCHAR(80),
        issued_date DATE,
        issued_book_isbn VARCHAR(50),
        issued_emp_id VARCHAR(10),
        FOREIGN KEY (issued_member_id) REFERENCES members(member_id),
        FOREIGN KEY (issued_emp_id) REFERENCES employees(emp_id),
        FOREIGN KEY (issued_book_isbn) REFERENCES books(isbn)
);
```

-- Create table "ReturnStatus"

```
DROP TABLE IF EXISTS return_status;
CREATE TABLE return_status
(
        return_id VARCHAR(10) PRIMARY KEY,
        issued_id VARCHAR(30),
        return_book_name VARCHAR(80),
        return_date DATE,
        return_book_isbn VARCHAR(50),
        FOREIGN KEY (return_book_isbn) REFERENCES books(isbn)
);
```

2. CRUD Operations

- **Create**: Inserted sample records into the books table.
- **Read**: Retrieved and displayed data from various tables.
- **Update**: Updated records in the employees table.
- **Delete**: Removed records from the members table as needed.

Task 1. Create a New Book Record -- "('978-1-60129-456-2', 'To Kill a Mockingbird', 'Classic', 6.00, 'yes', 'Harper Lee', 'J.B. Lippincott & Co.')"

```
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.');
SELECT * FROM books;
```

Task 2: Update an Existing Member's Address

```
UPDATE members
```

```
SET member_address = '125 Oak St'
WHERE member_id = 'C103';
```

Task 3: Delete a Record from the Issued Status Table -- Objective: Delete the record with issued_id = 'IS121' from the issued_status table.

```
DELETE FROM issued_status
WHERE   issued_id =   'IS121';
```

Task 4: Retrieve All Books Issued by a Specific Employee -- Objective: Select all books issued by the employee with emp_id = 'E101'.

```
SELECT * 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.

```
SELECT
   issued_emp_id,
   COUNT(*)
FROM issued_status
GROUP BY 1
HAVING COUNT(*) > 1
```

3. CTAS (Create Table As Select):

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

```
CREATE TABLE book_issued_cnt AS
SELECT b.isbn, b.book_title, COUNT(ist.issued_id) AS issue_count
```

```
FROM issued_status as ist
JOIN books as b
ON ist.issued_book_isbn = b.isbn
GROUP BY b.isbn, b.book_title;
```

4. Data Analysis & Findings:

The following SQL queries were used to address specific questions:

Task 7. Retrieve All Books in a Specific Category

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

Task 8: Find Total Rental Income by Category

```
SELECT
   b.category,
   SUM(b.rental_price),
   COUNT(*)
FROM
issued_status as ist
JOIN
books as b
ON b.isbn = ist.issued_book_isbn
GROUP BY 1
```

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

```
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

```
SELECT
```

```
    e1.emp_id,
    e1.emp_name,
    e1.position,
    e1.salary,
    b.*,
    e2.emp_name as manager
FROM employees as e1
JOIN
branch as b
ON e1.branch_id = b.branch_id
JOIN
employees as e2
ON e2.emp_id = b.manager_id
```

Task 11. Create a Table of Books with Rental Price Above a Certain Threshold

```
CREATE TABLE expensive_books AS
SELECT * FROM books
WHERE rental_price > 7.00;
```

Task 12: **Retrieve the List of Books Not Yet Returned**

```
SELECT * FROM issued_status as ist
LEFT JOIN
return_status as rs
ON rs.issued_id = ist.issued_id
WHERE rs.return_id IS NULL;
```

Advanced SQL Operations:

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.

```
SELECT
   ist.issued_member_id,
   m.member_name,
   bk.book_title,
   ist.issued_date,
   -- rs.return_date,
   CURRENT_DATE - ist.issued_date as over_dues_days
FROM issued_status as ist
JOIN
members as m
   ON m.member_id = ist.issued_member_id
JOIN
books as bk
ON bk.isbn = ist.issued_book_isbn
LEFT JOIN
return_status as rs
ON rs.issued_id = ist.issued_id
WHERE
   rs.return_date IS NULL
   AND
   (CURRENT_DATE - ist.issued_date) > 30
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).

```
Delimiter $$
```

```
CREATE PROCEDURE add_return_records(p_return_id VARCHAR(10), p_issued_id
VARCHAR(10), p_book_quality VARCHAR(10))
AS
DECLARE
   v_isbn VARCHAR(50);
   v_book_name VARCHAR(80);
BEGIN
   -- all your logic and code
   -- inserting into returns based on users input
   INSERT INTO return_status(return_id, issued_id, return_date, book_quality)
   VALUES(p_return_id, p_issued_id, CURRENT_DATE, p_book_quality);

   SELECT
      issued_book_isbn,
      issued_book_name
      INTO
      v_isbn,
      v_book_name
   FROM issued_status
   WHERE issued_id = p_issued_id;

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

   RAISE NOTICE 'Thank you for returning the book: %', v_book_name;
END ;
Delimiter $$

-- Testing FUNCTION add_return_records
```

issued_id = IS135

ISBN = WHERE isbn = '978-0-307-58837-1'

```
SELECT * FROM books
WHERE isbn = '978-0-307-58837-1';

SELECT * FROM issued_status
WHERE issued_book_isbn = '978-0-307-58837-1';

SELECT * FROM return_status
WHERE issued_id = 'IS135';

-- calling function
CALL add_return_records('RS138', 'IS135', 'Good');

-- calling function
CALL add_return_records('RS148', 'IS140', 'Good');
```

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.

```
CREATE TABLE branch_reports
AS
SELECT
    b.branch_id,
    b.manager_id,
    COUNT(ist.issued_id) as number_book_issued,
    COUNT(rs.return_id) as number_of_book_return,
    SUM(bk.rental_price) as total_revenue
```

```
FROM issued_status as ist
JOIN
employees as e
ON e.emp_id = ist.issued_emp_id
JOIN
branch as b
ON e.branch_id = b.branch_id
LEFT JOIN
return_status as rs
ON rs.issued_id = ist.issued_id
JOIN
books as bk
ON ist.issued_book_isbn = bk.isbn
GROUP BY 1, 2;

SELECT * FROM branch_reports;
```

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.

```
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.

```
SELECT
   e.emp_name,
   b.*,
   COUNT(ist.issued_id) as no_book_issued
FROM issued_status as ist
JOIN
employees as e
ON e.emp_id = ist.issued_emp_id
JOIN
branch as b
ON e.branch_id = b.branch_id
GROUP BY 1, 2
```

Task 18: 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
            m.member_id,
            m.member_name,
            ist.issued_book_name,
            COUNT(*)
FROM return_status AS rs
JOIN issued_status AS ist ON rs.issued_id = ist.issued_id
JOIN members AS m ON m.member_id = ist.issued_member_id
```

```
WHERE rs.book_quality = 'Damaged'
GROUP BY 1, 3
HAVING COUNT(*) > 2;
```

Reports:

-- **Database Schema**: Detailed table structures and relationships. -- **Data Analysis**: Insights into book categories, employee salaries, member registration trends, and issued books. -- **Summary Reports**: Aggregated data on high-demand books and employee performance.

Conclusion:

This project demonstrates the application of SQL skills in creating and managing a library management system. It includes database setup, data manipulation, and advanced querying, providing a solid foundation -- for data management and analysis.