Introduction

Project Overview

This project aims to develop a database system for a small library, enhancing scalability, item management, membership tracking, borrow tracking, and report generation. The system will deliver these features through an intuitive and user-friendly interface.

Scope

The system will enable efficient cataloging and tracking of loanable items, enforce borrowing policies, and generate detailed reports to support library management. Its core functionalities will include item management, membership management, borrowing and returning processes, and query and report generation, all while ensuring database integrity.

Glossary

- **UI (User Interface)**
    - A User Interface is a high level application that allows an end user to interact with a system.
- **3 Schema Architecture**
    - A database design approach that splits data views into external, conceptual and internal layers.
- **Documentation**
    - An official piece of information that provides internal and external parties information about a topic.

- **Internal Stakeholders**
    - People or groups inside a company that have a direct interest in the success of a project.
- **External Stakeholders**
    - People or groups outside of a company that have an indirect interest in the success of a project.
- **Functional Requirements**
    - Requirements for a system that specify what the system should be able to do. This includes its functions, behaviors, and operations.
- **Data Entities**
    - Data entities are an abstraction away from fully implemented tables. They specify attributes, data-types, and constraints from a high level view.
- **Entity Attributes**
    - A characteristic or trait of an entity type that describes the entity, for example, the Person entity type has the Date of Birth attribute
- **ER (entity-relationship) Diagram**
    - An entity–relationship model which describes interrelated things of interest in a specific domain of knowledge.
- **End User**
    - A user that interacts with the top level form of an application. (e.g. a library user interacting with the library database by using a system)

Platform

Chosen Platform: SQLite

Reasoning: We chose SQLite due to its easy to use and serverless database. It's built for smaller applications so it is perfect for what we are trying to make. SQLite does not require us to install anything complicated or any configuration to get it to work, which is perfect because many of us don't have much experience programming in SQL. It's very fast and has excellent storage so we can create this database without needing to wait around or fear loss of information.

Physical Schema

```
-- Book table: stores info about the books in the library
CREATE TABLE Book (
    book_id INTEGER PRIMARY KEY, -- Unique identifier for each book
    title VARCHAR(500) NOT NULL,
    author VARCHAR(500) NOT NULL,
    isbn VARCHAR(13) UNIQUE NOT NULL, -- ISBN number is unique
    publication_year INTEGER NOT NULL,
    genre VARCHAR(500) NOT NULL,
    availability_status TEXT CHECK (availability_status IN ('Available', 'CheckedOut',
'Reserved')) DEFAULT 'Available', -- Availability status of book with default being 'Available'
    popularity INTEGER -- optional
);

-- Media table: stores info about the various media types (ebook, audiobook, video)
CREATE TABLE Media (
    media_id INTEGER PRIMARY KEY, -- Unique identifier for each media item
    title VARCHAR(500) NOT NULL,
    creator VARCHAR(500) NOT NULL,
    media_type TEXT CHECK (media_type IN ('Ebook', 'Audiobook', 'Video')) NOT NULL, --
Media type must be either 'Ebook', 'Audiobook', or 'Video'
```

```
    isbn VARCHAR(13) UNIQUE NOT NULL, -- ISBN number is unique
    publication_year INTEGER NOT NULL,
    genre VARCHAR(500) NOT NULL,
    availability_status TEXT CHECK (availability_status IN ('Available', 'CheckedOut',
'Reserved')) DEFAULT 'Available', -- Availability status of media item with default being
'Available'
    popularity INTEGER --optional
);

-- Magazine table: stores info about the magazines in the library
CREATE TABLE Magazine (
    magazine_id INTEGER PRIMARY KEY, -- Unique identifier for each magazine
    title VARCHAR(500) NOT NULL,
    issue_number VARCHAR(20) NOT NULL,
    publication_date DATE NOT NULL,
    availability_status TEXT CHECK (availability_status IN ('Available', 'CheckedOut',
'Reserved')) DEFAULT 'Available', -- Availability status of magazine with default being 'Available'
    popularity INTEGER --optional
);

-- Membership_Type table: defines the different types of memberships in the library
CREATE TABLE Membership_Type (
    type TEXT PRIMARY KEY CHECK (type IN ('Regular', 'Student', 'Senior')), -- Membership
type must be either 'Regular', 'Student', or 'Senior'
    borrowing_limit INTEGER CHECK (borrowing_limit > 0) NOT NULL, -- Max number of items
a member can borrow
    daily_late_fee DECIMAL(5,2) CHECK (daily_late_fee >= 0) NOT NULL, -- Daily fee for late
returns
    extra_fees DECIMAL(5,2) CHECK (extra_fees >= 0) NOT NULL -- Extra fees
);

-- Client table: stores info about clients (members)
CREATE TABLE Client (
    client_id INTEGER PRIMARY KEY, -- Unique identifier for each client
    client_name VARCHAR(500) NOT NULL,
    contact_info VARCHAR(500) NOT NULL,
    publication_date DATE NOT NULL,
    membership_type TEXT DEFAULT 'Regular', -- Type of membership (defaults to Regular)
    account_status TEXT CHECK (account_status IN ('Active', 'Inactive')) DEFAULT 'Active', --
Status of the client's account (defaults to Active)
    FOREIGN KEY (membership_type) REFERENCES Membership_Type(type) -- Reference to
the Membership_Type table
);
```

```
-- Loan table: stores info about items borrowed by clients
CREATE TABLE Loan (
    loan_id INTEGER PRIMARY KEY, --Unique identifier for each loan
    client_id INTEGER,
    item_type TEXT CHECK (item_type IN ('Book', 'E-book', 'Audiobook', 'Video', 'Magazine'))
NOT NULL, -- Type of item borrowed
    media_id INTEGER, -- Media item borrowed (if applicable)
    book_id INTEGER, -- book borrowed (if applicable)
    magazine_id INTEGER, -- Magazine borrowed (if applicable)
    borrow_date DATE NOT NULL,
    due_date DATE NOT NULL,
    return_date DATE,
    fees_accrued DECIMAL(5,2) CHECK (fees_accrued >= 0), -- Late fees that have accrued
    FOREIGN KEY (client_id) REFERENCES Client(client_id), -- Reference to the Client table
    FOREIGN KEY (media_id) REFERENCES Media(media_id), -- Reference to the Media table
    FOREIGN KEY (book_id) REFERENCES Book(book_id), -- Reference to the Book table
    FOREIGN KEY (magazine_id) REFERENCES Magazine(magazine_id) -- Reference to the
Magazine table
);

-- Reservation table: stores info about items reserved by clients
CREATE TABLE Reservation (
    reservation_id INTEGER PRIMARY KEY, -- Unique identifier for each reservation
    client_id INTEGER,
    item_type TEXT CHECK (item_type IN ('Book', 'E-book', 'Audiobook', 'Video', 'Magazine'))
NOT NULL, -- Type of item reserved
    media_id INTEGER, -- Media item reserved (if applicable)
    book_id INTEGER, -- Book reserved (if applicable)
    magazine_id INTEGER, -- Magazine reserved (if applicable)
    reservation_date DATE NOT NULL,
    status TEXT CHECK (status IN ('Ready for pickup', 'In line', 'Processing')) DEFAULT
'Processing', -- Status of the reservation (default is Processing)
    place_in_line INTEGER,
    FOREIGN KEY (client_id) REFERENCES Client(client_id), -- Reference to the Client table
    FOREIGN KEY (media_id) REFERENCES Media(media_id), -- Reference to the Media table
    FOREIGN KEY (book_id) REFERENCES Book(book_id), -- Reference to the Book table
    FOREIGN KEY (magazine_id) REFERENCES Magazine(magazine_id) -- Reference to the
Magazine table
);

-- Notification table: stores notifications sent to clients
CREATE TABLE Notification (
    notification_id INTEGER PRIMARY KEY, -- Unique identifier for each notification
    client_id INTEGER,
```

```
    message TEXT NOT NULL, -- Content of the notification
    FOREIGN KEY (client_id) REFERENCES Client(client_id) -- Reference to the Client table
);

-- Gets table (client receives notifications) - many-to-many relationship between clients and notifications, tracks which clients recieve which notifications
CREATE TABLE Gets (
    client_id INTEGER,
    notification_id INTEGER,
    PRIMARY KEY (client_id, notification_id),
    FOREIGN KEY (client_id) REFERENCES Client(client_id),
    FOREIGN KEY (notification_id) REFERENCES Notification(notification_id)
);

-- Has table (client has membership type) - many-to-many relationship between clients and membership types
CREATE TABLE Has (
    type TEXT,
    client_id INTEGER,
    PRIMARY KEY (type, client_id), -- Composite primary key
    FOREIGN KEY (type) REFERENCES Membership_Type(type), -- Reference to the Membership_Type table
    FOREIGN KEY (client_id) REFERENCES Client(client_id) -- Reference to the Client table
);

-- Takes table (client takes loans) - many-to-many relationship between clients and loans
CREATE TABLE Takes (
    client_id INTEGER,
    loan_id INTEGER,
    PRIMARY KEY (client_id, loan_id), -- Composite primary key
    FOREIGN KEY (client_id) REFERENCES Client(client_id), -- Reference to the Client table
    FOREIGN KEY (loan_id) REFERENCES Loan(loan_id) -- Reference to the Loan table
);

-- Loaned table (loans linked to items) - many-to-many relationship between loans and items, tracks which items are associated with which loans
CREATE TABLE Loaned (
    loan_id INTEGER,
    media_id INTEGER,
    magazine_id INTEGER,
    book_id INTEGER,
    PRIMARY KEY (loan_id), -- Primary key is loan_id
    FOREIGN KEY (loan_id) REFERENCES Loan(loan_id), -- Reference to the Loan table
    FOREIGN KEY (media_id) REFERENCES Media(media_id), -- Reference to the Media table
```

```
    FOREIGN KEY (magazine_id) REFERENCES Magazine(magazine_id), -- Reference to the
Magazine table
    FOREIGN KEY (book_id) REFERENCES Book(book_id) -- Reference to the Book table
);
```

-- Reserved table (reservation linked to items) - many-to-many relationship between reservations and items, tracks which items are associated with which reservations

```
CREATE TABLE Reserved (
    reservation_id INTEGER,
    media_id INTEGER,
    magazine_id INTEGER,
    book_id INTEGER,
    PRIMARY KEY (reservation_id), -- Primary key is reservation_id
    FOREIGN KEY (reservation_id) REFERENCES Reservation(reservation_id), -- Reference to
the Reservation table
    FOREIGN KEY (media_id) REFERENCES Media(media_id), -- Reference to the Media table
    FOREIGN KEY (magazine_id) REFERENCES Magazine(magazine_id), -- Reference to the
Magazine table
    FOREIGN KEY (book_id) REFERENCES Book(book_id) -- Reference to the Book table
);
```

-- Reserves table (client reserves items) - many-to-many relationship between clients and reservations

```
CREATE TABLE Reserves (
    client_id INTEGER,
    reservation_id INTEGER,
    PRIMARY KEY (client_id, reservation_id), -- Composite primary key
    FOREIGN KEY (client_id) REFERENCES Client(client_id), -- Reference to the Client table
    FOREIGN KEY (reservation_id) REFERENCES Reservation(reservation_id) -- Reference to
the Reservation table
);
```

Table Contents

select * from Book;

1|The Great Gatsby|F. Scott Fitzgerald|9780743273565|1925|Fiction|Available|8
2|1984|George Orwell|9780451524935|1949|Dystopian|CheckedOut|10
3|To Kill a Mockingbird|Harper Lee|9780061120084|1960|Classic|Available|9
4|The Catcher in the Rye|J.D. Salinger|9780316769488|1951|Classic|Reserved|7
5|Sapiens|Yuval Noah Harari|9780062316097|2011|History|Available|8

select * from Client;
1|Alice Johnson|alice@example.com|2023-05-01|Regular|Active
2|Bob Smith|bob@example.com|2023-06-15|Student|Active
3|Carol White|carol@example.com|2023-07-20|Senior|Inactive
4|David Lee|david@example.com|2024-01-10|Regular|Active
5|Eve Torres|eve@example.com|2024-02-11|Student|Active

select * from Gets;
1|1
2|2
4|3
5|4

select * from Has;
Regular|1
Student|2
Senior|3
Regular|4
Student|5

select * from Loan;
1|1|Book||1||2024-04-01|2024-04-15||0
2|2|E-book|2|||2024-04-10|2024-04-20||0
3|3|Magazine|||1|2024-03-01|2024-03-10|2024-03-11|0.3
4|4|Audiobook|3|||2024-04-15|2024-04-25||0
5|5|Book||5||2024-04-18|2024-05-01||0

select * from Loaned;
1|||1
2|2||
3||1|
4|3||
5|||5

select * from Magazine;
1|National Geographic|2023-09|2023-09-01|Available|6
2|Time|2023-10|2023-10-01|CheckedOut|7
3|Forbes|2023-11|2023-11-01|Available|5
4|Wired|2024-01|2024-01-01|Reserved|4

select * from Membership_Type;
Regular|5|0.5|0
Student|7|0.25|5

Senior|6|0.3|2.5

select * from Notification;
1|1|Your audiobook is ready for pickup.
2|2|The book you reserved is now available.
3|4|The video Cosmos is ready for pickup.
4|5|Your magazine reservation is being processed.

select * from Reservation;
1|1|Audiobook|1|||2024-04-25|Processing|1
2|2|Book||2||2024-04-26|In line|2
3|4|Video|4|||2024-04-20|Ready for pickup|1
4|5|Magazine|||4|2024-04-22|Processing|1

select * from Reserved;
1|1||
2|||2
3|4||
4||4|

select * from Reserves;
1|1
2|2
4|3
5|4

select * from Takes;
1|1
2|2
3|3
4|4
5|5

Insert Queries

Insert a new client:

INSERT INTO Client (client_id, client_name, contact_info, membership_type, account_status)
VALUES (1, ‘John Doe’, ‘johndoe@gmail.com’, ‘Regular’, ‘Active’);’

Insert a new book:

```
INSERT INTO Book (book_id, title, author, isbn, publication_year, genre, avaliability_status,
popularity)
VALUES (111, 'The Great Gatsby', 'F. Scott Fitzgerald', '1234567890123', 1925, 'Classic',
'Available', 10);
```

Insert a new loan:

```
INSERT INTO Loan (loan_id, client_id, item_type, media_id, book_id, magazine_id,
borrow_date, due_date, return_date, fees_accrued);
VALUES (101, 999, 'Book', NULL, 101, NULL, CURRENT_DATE,
DATE_ADD(CURRENT_DATE, INTERVAL 14 DAY), NULL, 0.00);
```

Insert a reservation:

```
INSERT INTO Reservation (reservation_id, client_id, item_type, media_id, book_id,
magazine_id, reservation_date, status, place_in_line);
VALUES (111, 999, 'Book', NULL, 101, NULL, CURRENT_DATE, 'In line', 6);
```

Update Queries

Updating a Client:

```
UPDATE Client
SET         contact_info = 'johndoe1@gmail.com'
            membership_type = 'Senior'
WHERE   client_id = 1;
```

Updating a Book:

```
UPDATE Book
SET         availability_status = 'Reserved'
WHERE   book_id = 111;
```

Search Queries

Search for books by title:

```
SELECT * FROM Book
```

WHERE title LIKE ‘%Gatsby%’;w

Search for magazine by issue number:

```
SELECT * FROM Magazine
WHERE issue_number = 1;
```

Search for media by type and genre:

```
SELECT * FROM Media
WHERE media_type = ‘Video’ AND genre = ‘Sci-Fi’;
```

Search for a client’s active loans:

```
SELECT *
FROM Loan
JOIN Client ON Loan.client_id = Client.client_id
WHERE Client.client_id = 1 AND Loan.return_date IS NULL;
```

Notification Queries

Find loans due today:

```
SELECT client_id, loan_id
FROM Loan
WHERE due_date = CURRENT_DATE AND return_date IS NULL;
```

```
INSERT INTO Notification (notification_id, client_id, message)
VALUES (2001, 1, ‘Your loan is due today. Please return it by the end of the day to avoid late fees.’);
```

Find reservations where item became available:

```
SELECT Reservation.client_id, Reservation.reservation_id
FROM Reservation
JOIN Book ON Reservation.book_id = Book.book_id
WHERE Reservation.status = ‘In line’ AND Book.availiability_status = ‘Available’
ORDER BY place_in_line ASC;
```

```
INSERT INTO Notification (notification_id, client_id, message)
VALUES (2001, 1, ‘Your reservation is ready to pick up.’);
```

Report Queries

Report for 10 most popular books:

```
SELECT book_id, title, author, popularity
FROM Book
ORDER BY popularity DESC
LIMIT 10;
```

Report for clients with outstanding fees:

```
SELECT Client.name, SUM(Loan.fees_accrued) AS total_fees
FROM Loan
JOIN Client ON Loan.client_id = Client.client_id
WHERE Loan.return_date IS NOT NULL
GROUP BY Client.client_id
HAVING total_fees > 0
ORDER BY total_fees DESC;
```

Report for average borrowed items per membership type:

```
SELECT membership_type, COUNT(Loan.loan_id) / COUNT(DISTINCT Client.client_id) AS
avg_loans_per_client
FROM Client
JOIN Loan on Client.client_id = Loan.client_id
GROUP BY membership_type;
```
