1. Introduction

1.1. Project Overview

The MetaMinds Library Management System (LMS) advances from a logical blueprint into a fully operational physical database. This implementation phase solidifies the system's ability to manage users, transactions, reservations, library inventory, fees, notifications, and administrative roles efficiently. The physical schema is rigorously tested to ensure integrity, consistency, and compliance with the requirements previously outlined in the Vision, Requirements, and Logical Modeling phases.

By deploying the LMS database on Supabase PostgreSQL and leveraging robust data import techniques, we ensured scalability, performance, and reliability — preparing the database for real-world operational scenarios.

1.2. Scope

1.2.1. Includes:

- Complete creation of database tables with appropriate primary keys, foreign keys, and integrity constraints.
- Use of Supabase's "Import from CSV" functionality to bulk populate data generated via Python scripts and public datasets.
- Validation of all entities, relationships, and business rules (e.g., membership borrowing limits, late fees, notification generation).
- Execution of complex queries and reports to test the database's real-world readiness.

1.2.2. Does not Include:

- Integration with external digital book platforms.
- Physical barcode scanning or RFID tracking.
- Payment gateway integrations (e.g., Stripe, PayPal).
- Machine learning-driven recommendations.

1.3. Glossary

1.3.1. **Physical Schema:** The actual set of SQL statements used to define the database structure.

1.3.2. **DDL (Data Definition Language):** SQL commands such as CREATE TABLE used to build database schema.

1.3.3. **Bulk Insert:** Loading large amounts of data into database tables using CSV files or external tools.

1.3.4. **Supabase:** A backend-as-a-service platform providing hosted PostgreSQL databases, APIs, authentication, and more.

1.3.5. **Constraints:** Rules applied to table columns, such as CHECK constraints, to maintain data integrity.

2. Platform Selection

For the physical implementation, we selected **Supabase PostgreSQL**.

Reasons:

- Seamless hosting and management of a scalable PostgreSQL instance.
- Built-in CSV import functionality, enabling efficient large-scale data population.
- Full SQL compliance with strong support for modern database features.
- Ease of access and real-time database administration via Supabase UI.
- Familiarity with PostgreSQL from previous coursework and project phases.

Limitations considered: Slight learning curve on Supabase's specific UI features compared to pure SQL command line, but outweighed by its automation and scalability benefits.

3. Physical Database Creation

3.1. DDL Statements

3.1.1. AdminRoles

```
CREATE TABLE AdminRoles (
    Role_ID INT PRIMARY KEY,
      Role_Name TEXT NOT NULL CHECK (Role_Name IN ('Librarian', 'Manager', 'Clerk')),
```

```
    Permissions TEXT
);
```

3.1.2. Admins Table

```
CREATE TABLE Admins (
    Admin_ID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    Role_ID INT,
    FOREIGN KEY (Role_ID) REFERENCES AdminRoles(Role_ID)
);
```

3.1.3. Users Table

```
CREATE TABLE Users (
    User_ID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    Phone VARCHAR(20),
       Membership_type TEXT NOT NULL CHECK (Membership_type IN
('Free', 'Premium')),
    Borrowing_limit INT NOT NULL CHECK (Borrowing_limit ⩾ 0),
        Account_Status TEXT NOT NULL CHECK (Account_Status IN
('Active', 'Suspended'))
);
```

3.1.4. LibraryItems Table

```
CREATE TABLE LibraryItems (
    Item_ID INT PRIMARY KEY,
    Title VARCHAR(200) NOT NULL,
    Creator VARCHAR(100),
        Item_type TEXT NOT NULL CHECK (Item_type IN ('Book',
'Magazine', 'Digital Media')),
    ISBN VARCHAR(20),
    Genre VARCHAR(50),
    Publication_Year INT CHECK (Publication_Year ⩾ 0),
        Availability TEXT CHECK (Availability IN ('Available',
'Borrowed', 'Reserved'))
);
```

3.1.5. Transactions Table

```
CREATE TABLE Transactions (
    Transaction_ID INT PRIMARY KEY,
    User_ID INT,
    Item_ID INT,
    Borrow_Date TIMESTAMP NOT NULL,
    Due_Date TIMESTAMP NOT NULL,
    Return_Date TIMESTAMP,
    Late_Fee DECIMAL(5,2) DEFAULT 0,
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID),
    FOREIGN KEY (Item_ID) REFERENCES LibraryItems(Item_ID)
);
```

3.1.6. Reservations Table

```
CREATE TABLE Reservations (
    Reservation_ID INT PRIMARY KEY,
    User_ID INT,
    Item_ID INT,
    Reservation_Date TIMESTAMP NOT NULL,
    Expiration_Date TIMESTAMP NOT NULL,
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID),
    FOREIGN KEY (Item_ID) REFERENCES LibraryItems(Item_ID)
);
```

3.1.7. Notifications Table

```
CREATE TABLE Notifications (
    Notification_ID INT PRIMARY KEY,
    User_ID INT,
    Message TEXT,
    Sent_Date TIMESTAMP NOT NULL,
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID)
);
```

3.1.8. Reviews Table

```
CREATE TABLE Reviews (
    Review_ID INT PRIMARY KEY,
    User_ID INT,
```

```
    Item_ID INT,
    Rating INT CHECK (Rating ⩾ 1 AND Rating ⩽ 5),
    Review_Text TEXT,
    Review_Date TIMESTAMP NOT NULL,
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID),
    FOREIGN KEY (Item_ID) REFERENCES LibraryItems(Item_ID)
);
```

3.1.9. Fees Table

```
CREATE TABLE Fees (
    Fee_ID INT PRIMARY KEY,
    User_ID INT,
    Transaction_ID INT,
    Amount DECIMAL(5,2) NOT NULL CHECK (Amount ⩾ 0),
      Paid_Status TEXT NOT NULL CHECK (Paid_Status IN ('Paid',
'Unpaid')),
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID),
                FOREIGN KEY (Transaction_ID) REFERENCES
Transactions(Transaction_ID)
);
```

4. Data Population

To populate the database:

- Python scripts and online datasets were used to create CSV files for each table.
- Supabase's "Import from CSV" tool allowed seamless and efficient bulk insertion into each table.
- Data fields were validated before upload to maintain relational consistency (e.g., ensuring valid foreign key references).

4.1. Populated Table Size:

Table	Records
Notifications	49

Users	100
LibraryItems	156
Reviews	17
AdminRoles	3
Reservations	202
Fees	27
Transactions	200
Admins	2

5. Table Content Prints

```
SELECT * FROM Users;
SELECT * FROM LibraryItems;
SELECT * FROM Transactions;
SELECT * FROM Reservations;
SELECT * FROM Fees;
SELECT * FROM Notifications;
SELECT * FROM AdminRoles;
SELECT * FROM Admins;
SELECT * FROM Reviews;
```

The above commands were used to print and verify each table's data post-population. Each query result can be found by Project Artifacts > TableResults where each file name corresponds to the table name.

6. Functionality Testing

We tested the database using the following queries:

6.1. Find available books of a specific genre:

```
SELECT Title FROM LibraryItems
WHERE Genre = 'Fiction'
AND Availability = 'Available';
```

6.2. Find users with unpaid fees:

```
SELECT u.Name, SUM(f.Amount) AS Total_Unpaid
FROM Users u
JOIN Fees f ON u.User_ID = f.User_ID
WHERE f.Paid_Status = 'Unpaid'
GROUP BY u.Name;
```

6.3. Report: Most Reserved Items:

```
SELECT l.Title, COUNT(r.Reservation_ID) AS ReservationCount
FROM LibraryItems l
JOIN Reservations r ON l.Item_ID = r.Item_ID
GROUP BY l.Title
ORDER BY ReservationCount DESC;
```

6.4. Late Returns Report:

```
SELECT u.Name, t.Borrow_Date, t.Due_Date
FROM Users u
JOIN Transactions t ON u.User_ID = t.User_ID
WHERE t.Return_Date IS NULL AND t.Due_Date < NOW();
```

The testing results:

- Query execution time < 300 ms for all tested queries.
- Correct outputs for all entity relationships and edge cases.
- Validation for triggers like late fee calculation and borrowing limits handled externally at the application level as per project scope.