Tables and Relationships

User

Description: Represents both library members and librarians.

Fields:

- `id` (Integer, Primary Key): Unique identifier for the user
- `name` (String): Name of the user
- `email` (String): Unique email address for login
- `password` (String): Hashed password for authentication
- `role` (Enum): Either 'Member' or 'Librarian'
- `barcode` (String): Unique barcode for user identification
- `created_at` (DateTime): When the user account was created
- `notifications` (Text): JSON string storing user notifications

Relationships:

- One-to-many relationship with `Transaction`
- One-to-many relationship with `Reservation`

Methods:

- `add_notification(subject, body, book_title=None)`: Adds a notification to the user's notifications JSON field
- `generate_barcode()`: Generates a unique barcode for the user

Book

Description: Represents a book title in the library catalog.

Fields:

- `id` (Integer, Primary Key): Unique identifier for the book
- `barcode` (String): Unique barcode for book identification
- `title` (String): Title of the book
- `author` (String): Author of the book
- `subject_category` (String): Subject or genre category
- `publication_date` (Date): Date when the book was published

Relationships:

- One-to-many relationship with `BookCopy` (with cascade delete)
- One-to-many relationship with `Reservation`

Methods:

- `generate_barcode()`: Generates a unique barcode for the book

BookCopy

Description: Represents individual physical copies of books in the library.

Fields:

- `id` (Integer, Primary Key): Unique identifier for the book copy
- `book_id` (Integer, Foreign Key): Reference to the book

- `rack_location` (String): Physical location in the library
- `is_available` (Boolean): Whether the copy is available for borrowing

Relationships:

- Many-to-one relationship with `Book`
- One-to-many relationship with `Transaction`

Transaction

Description: Records book checkouts, returns, and associated fines.

Fields:

- `id` (Integer, Primary Key): Unique identifier for the transaction
- `user_id` (Integer, Foreign Key): Reference to the borrowing user
- `book_copy_id` (Integer, Foreign Key): Reference to the borrowed book copy
- `checkout_date` (DateTime): When the book was checked out
- `due_date` (DateTime): When the book is due to be returned
- `return_date` (DateTime, Nullable): When the book was returned (null if not yet returned)
- `fine_amount` (Float): Amount of fine for late returns

Relationships:

- Many-to-one relationship with `User`
- Many-to-one relationship with `BookCopy`

Reservation

Description: Tracks book reservations made by users.

Fields:

- `id` (Integer, Primary Key): Unique identifier for the reservation
- `user_id` (Integer, Foreign Key): Reference to the user making the reservation
- `book_id` (Integer, Foreign Key): Reference to the reserved book
- `reservation_date` (DateTime): When the reservation was made
- `status` (Enum): 'Pending', 'Fulfilled', or 'Cancelled'

Relationships:

- Many-to-one relationship with `User`
- Many-to-one relationship with `Book`

Key Features

1. **User Management**:
    - Support for two user roles: Member and Librarian
    - Unique barcodes for user identification
    - Embedded notification system within the User model
2. **Book Management**:
    - Distinction between book titles (Book) and physical copies (BookCopy)
    - Tracking of book availability status

    - Detailed book metadata (author, category, publication date)

3. **Circulation Management**:
    - Complete transaction history with checkout and return dates
    - Fine calculation for overdue books stored with each transaction
    - Reservation system for books

SQLAlchemy Implementation

The database is implemented using SQLAlchemy ORM with proper relationship definitions including:

- Foreign keys to maintain referential integrity
- Backref attributes for convenient bidirectional access
- Cascade delete rules where appropriate (e.g., deleting a book deletes all its copies)

Notes

- Notifications are stored directly in the User model as a JSON field rather than in a separate table
- Each book and user has a unique barcode generated using UUID
- Fine amounts are stored directly in the Transaction model rather than in a separate table