Simple Database Design

Functional Requirements

- Manage books: add, update, delete, search, and list.
- Manage borrowers: register, update, delete, and list.
- Borrowing process: borrow books, return books, track due dates, and list overdue books.

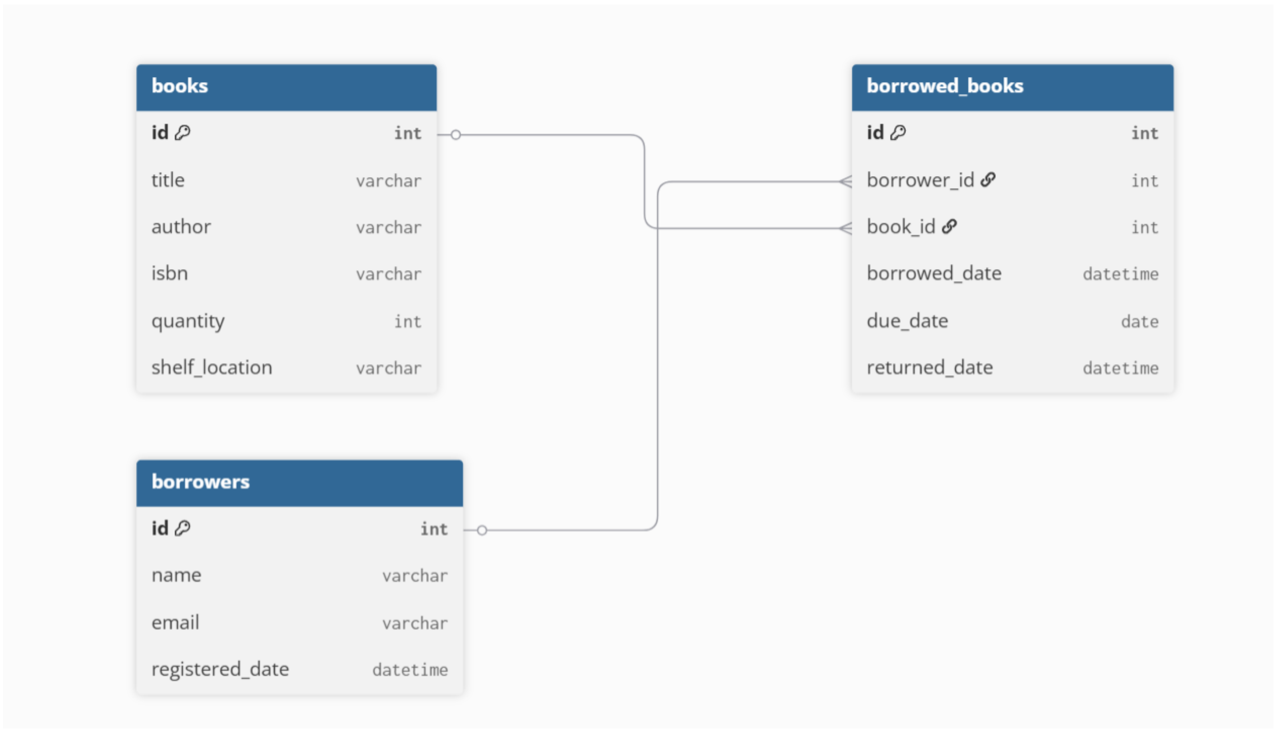
Conceptual Design
The conceptual design represents the core entities and their relationships.

Entities:

- **Book** – details of available books.
- **Borrower** – registered library users.
- **BorrowedBook** – records each borrowing transaction.

Relationships:

- A borrower can borrow many books.
- A book can be borrowed by many borrowers (over time).
- BorrowedBook is the linking table (many-to-many relationship).



Note on Normalization
The database schema is designed according to normalization principles (up to 3rd Normal Form). This ensures:

- No duplicate data between tables.
- Clear separation of entities (`books`, `borrowers`, `borrowed_books`).
- Better data integrity and easier maintenance.

Logical Design
The database has three tables:

books

- `id` (PK)
- `title`
- `author`
- `isbn` (unique)
- `quantity`
- `shelf_location`

borrowers

- `id` (PK)
- `name`
- `email` (unique)
- `registered_date`

borrowed_books

- `id` (PK)
- `borrower_id` (FK → borrowers.id)
- `book_id` (FK → books.id)
- `borrowed_date`
- `due_date`
- `returned_date`

Physical Design
Indexes are added to speed up search and lookups:

- `books`: indexes on `title`, `author`, and `isbn`.

Implementation Notes

- The database schema was implemented using **Sequelize** (Node.js ORM).
- Sequelize models were created for each table (`Book`, `Borrower`, `BorrowedBook`) with appropriate data types, primary keys, foreign keys, and indexes.
- Sequelize's associations were used to define relationships between entities.
- `sequelize.sync()` was used to create and synchronize tables in MySQL automatically.