实验二：小型图书管理系统

1 系统功能需求

本个人图书管理系统旨在提供一个便捷的平台，用于管理个人藏书、读者信息以及图书的借阅和归还过程。系统应具备以下主要功能：

1. **图书管理 (Books Management)**
    - **添加图书**: 管理员能够向系统中添加新的图书信息，包括书名、作者、ISBN（国际标准书号，具有唯一性）、出版社、出版年份、分类、总库存量。添加时，可借阅库存默认为总库存。
    - **编辑图书**: 管理员能够修改已存在的图书信息，包括上述所有字段。修改库存时，需确保可借阅库存不大于总库存，且两者均不能为负数。
    - **删除图书**: 管理员能够从系统中删除图书。如果图书当前有未归还的借阅记录，则不允许删除，并提示用户。
    - **查询与浏览图书**:
        - 用户可以浏览所有图书的列表，列表应包含关键信息（如书名、作者、ISBN、分类、总库存、可借阅库存）。
        - 支持分页显示图书列表，以优化大量数据时的浏览体验。
        - 支持按书名、作者或 ISBN 进行模糊搜索或精确搜索。
    - **查看图书详情**: 能够查看单本图书的完整详细信息。
2. **读者管理 (Readers Management)**
    - **添加读者**: 管理员能够添加新的读者信息，包括姓名、读者编号、联系方式。
    - **编辑读者**: 管理员能够修改已存在的读者信息。
    - **删除读者**: 管理员能够删除读者。如果读者当前有未归还的图书，则不允许删除，并提示用户。
    - **查询与浏览读者**: 用户可以浏览所有读者的列表，列表应包含关键信息（如姓名、读者编号、联系方式）。
3. **借阅管理 (Loans Management)**

- **借书**:
  - 管理员能够为指定读者借阅指定的图书。
  - 借书时需选择图书和读者，并指定应归还日期。
  - 系统自动记录借阅日期为当前日期。
  - 借书成功后，对应图书的“可借阅库存”会自动减 1。
  - 如果图书的可借阅库存为 0，则不允许借阅。
  - 借书操作应在一个数据库事务中完成，确保数据一致性。
- **还书**:
  - 管理员能够记录指定借阅记录的归还操作。
  - 还书时，系统自动记录归还日期为当前日期。
  - 还书成功后，对应图书的“可借阅库存”会自动加 1。
  - 还书操作应在一个数据库事务中完成。
- **查询当前借阅**: 用户可以查看所有当前未归还的借阅记录列表，包括借阅的图书信息、读者信息、借阅日期和应归还日期。
- **查询逾期借阅**: 用户可以查看所有已到期但尚未归还的借阅记录列表，以便进行催还。

4. **综合查询 (Comprehensive Search/Reports)**

- **查询读者借阅历史**: 用户可以根据读者查询其所有的借阅历史记录（包括已归还和未归还的）。
- **查询图书借阅历史**: 用户可以根据图书查询其所有的被借阅历史记录。

5. **系统层面**

- **数据库初始化**: 提供命令行工具，用于初始化数据库表结构及预定义的视图和触发器。
- **用户界面**: 提供简洁易用的 Web 用户界面，方便用户进行各项操作。
- **错误提示与反馈**: 对用户的操作提供明确的成功或失败提示，对错误操作（如输入不合法、违反约束等）给出清晰的错误信息。

2 数据库设计

2.1 E-R 图设计及其说明

实体 (Entities)

根据系统需求，我们识别出以下主要实体：

- **读者 (Reader)**: 存储读者的基本信息。
- **图书 (Book)**: 存储图书的详细信息。
- **借阅 (Loan)**: 记录图书的借阅活动，连接读者和图书。

关系 (Relationships)

- 一个 **读者**可以有多条 **借阅**记录 (一对多关系: Reader 1 — * Loan)。
- 一本 **图书**可以有多条 **借阅**记录 (一对多关系: Book 1 — * Loan)。
  - 因此，**借阅 (Loan)** 实体是一个关联实体，它解决了 **读者 (Reader)** 和 **图书 (Book)** 之间的多对多关系（一个读者可以借阅多本书，一本书可以被多个读者借阅）。

属性 (Attributes)

- **读者 (Reader)**: `reader_id` (主键), `name`, `reader_number` (唯一), `contact`
- **图书 (Book)**: `book_id` (主键), `title`, `author`, `isbn` (唯一), `publisher`, `publication_year`, `category`, `total_stock`, `available_stock`
- **借阅 (Loan)**: `loan_id` (主键), `book_id` (外键), `reader_id` (外键), `loan_date`, `due_date`, `return_date`

E-R 图 (Conceptual)

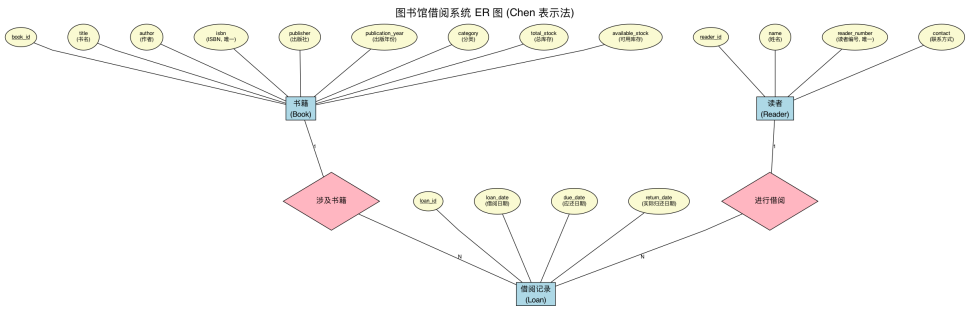

图 1: 个人图书管理系统 E-R 图

2.2 数据库逻辑结构设计

数据库包含三个主要表：readers, books, 和 loans，以及两个视图 view_activeloans 和 view_overdueloans。这些结构定义在 personal_library/schema.sql 文件中。

2.2.1 表结构

readers 表

- **说明**: 存储读者信息。
- **逻辑结构**:
  - reader_id: SERIAL, 主键 (PK)。自动递增的唯一标识符。
  - name: VARCHAR(100), NOT NULL。读者姓名。
  - reader_number: VARCHAR(50), UNIQUE, NOT NULL。读者编号，具有唯一性约束。
  - contact: VARCHAR(100)。读者联系方式，可为空。

books 表

- **说明**: 存储图书信息。
- **逻辑结构**:
  - book_id: SERIAL, 主键 (PK)。自动递增的唯一标识符。
  - title: VARCHAR(255), NOT NULL。图书标题。
  - author: VARCHAR(100), NOT NULL。图书作者。
  - isbn: VARCHAR(20), UNIQUE, NOT NULL。国际标准书号，具有唯一性约束。
  - publisher: VARCHAR(100)。出版社，可为空。
  - publication_year: INTEGER。出版年份，可为空。
  - category: VARCHAR(50)。图书分类，可为空。
  - total_stock: INTEGER, NOT NULL, DEFAULT 0。总库存量。
    - 用户定义完整性: CONSTRAINT chk_total_stock CHECK (total_stock >= 0) 确保总库存不为负。
  - available_stock: INTEGER, NOT NULL, DEFAULT 0。可借阅库存量。
    - 用户定义完整性: CONSTRAINT chk_available_stock CHECK (available_stock >= 0 AND available_stock <= total_stock) 确保可借阅库存不为负且不超过总库存。

loans 表

- **说明**: 存储借阅记录，关联 books 表和 readers 表。
- **逻辑结构**:
  - loan_id: SERIAL, 主键 (PK)。自动递增的唯一标识符。
  - book_id: INTEGER, NOT NULL。外键 (FK)，参照 books(book_id)。
    - 参照完整性: ON DELETE RESTRICT - 如果某本图书在 books 表中被删除，但其 book_id 仍存在于 loans 表的未归还记录中，则禁止删除该图书。
  - reader_id: INTEGER, NOT NULL。外键 (FK)，参照 readers(reader_id)。
    - 参照完整性: ON DELETE RESTRICT - 如果某个读者在 readers 表中被删除，但其 reader_id 仍存在于 loans 表的未归还记录中，则禁止删除该读者。
  - loan_date: DATE, NOT NULL, DEFAULT CURRENT_DATE。借阅日期，默认为当前日期。
  - due_date: DATE, NOT NULL。应归还日期。
  - return_date: DATE, DEFAULT NULL。实际归还日期，默认为 NULL (表示未归还)。

2.2.2 视图结构

view_activeloans 视图

- **说明**: 显示所有当前未归还的借阅记录的详细信息。
- **逻辑结构**: 该视图通过连接 loans, books, 和 readers 表，筛选出 loans.return_date 为 NULL 的记录。
  - loan_id: 借阅 ID。
  - reader_name: 读者姓名。
  - reader_number: 读者编号。
  - book_title: 图书标题。
  - isbn: 图书 ISBN。
  - loan_date: 借阅日期。
  - due_date: 应归还日期。

view_overdueloans 视图

- **说明**: 显示所有已逾期但尚未归还的借阅记录。
- **逻辑结构**: 该视图基于 loans, books, 和 readers 表，筛选出 loans.return_date 为 NULL 且 loans.due_date 早于当前日期的记录。
  - loan_id: 借阅 ID。

  - reader_name: 读者姓名。
  - reader_number: 读者编号。
  - reader_contact: 读者联系方式。
  - book_title: 图书标题。
  - isbn: 图书 ISBN。
  - loan_date: 借阅日期。
  - due_date: 应归还日期。

2.3 数据库物理设计

为了优化查询性能，在数据库表的一些关键列上创建了索引。这些索引定义在 personal_library/schema.s 文件中。

2.3.1 索引定义

books 表索引

- CREATE INDEX idx_books_title ON books(title);
  - **说明**: 在 books 表的 title 列上创建索引，以加速按书名搜索和排序。
- CREATE INDEX idx_books_author ON books(author);
  - **说明**: 在 books 表的 author 列上创建索引，以加速按作者搜索和排序。
- CREATE INDEX idx_books_category ON books(category);
  - **说明**: 在 books 表的 category 列上创建索引，以加速按分类筛选和排序。
- 主键 book_id 和唯一键 isbn 会自动创建索引。

readers 表索引

- CREATE INDEX idx_readers_name ON readers(name);
  - **说明**: 在 readers 表的 name 列上创建索引，以加速按读者姓名搜索和排序。
- 主键 reader_id 和唯一键 reader_number 会自动创建索引。

loans 表索引

- CREATE INDEX idx_loans_book_id ON loans(book_id);
  - **说明**: 在 loans 表的 book_id (外键) 列上创建索引，以优化涉及与 books 表连接的查询，以及按特定图书查询借阅记录的性能。
- CREATE INDEX idx_loans_reader_id ON loans(reader_id);

  - **说明**: 在 `loans` 表的 `reader_id` (外键) 列上创建索引，以优化涉及与 `readers` 表连接的查询，以及按特定读者查询借阅记录的性能。

- `CREATE INDEX idx_loans_due_date ON loans(due_date);`
  - **说明**: 在 `loans` 表的 `due_date` 列上创建索引，以加速查询逾期借阅或按应归还日期排序的记录。

- `CREATE INDEX idx_loans_return_date ON loans(return_date);`
  - **说明**: 在 `loans` 表的 `return_date` 列上创建索引，以加速筛选已归还或未归还记录的查询 (特别是 `WHERE return_date IS NULL`)。

- 主键 `loan_id` 会自动创建索引。

3 详细设计与实现

3.1 图书管理 (Books Management)

3.1.1 添加新图书

实现过程:

1. 用户通过 Web 表单提交图书信息（书名、作者、ISBN、出版社、出版年份、分类、总库存）。
2. 应用后端（例如 Flask 路由，如 `app.py` 中的相关函数）接收表单数据。
3. 进行基本的数据校验（如必填项、总库存非负）。
4. 初始时，将“可借阅库存”设置为等于“总库存”。
5. 构造 `INSERT` SQL 语句，将图书信息插入到 `books` 表中。
6. 执行插入操作。如果 ISBN 已存在（违反 `UNIQUE` 约束）或库存设置不当（违反 `CHECK` 约束 `chk_available_stock` 或 `chk_total_stock`），数据库会返回错误，应用捕获该错误并向用户显示相应的提示信息。
7. 成功插入后，重定向到图书列表页面并显示成功消息。

数据库事务: 添加图书的操作本身构成一个单一的数据库事务。在数据库交互模块（例如 `db.py` 中的 `query_db` 函数）中，如果操作是写入型（如 `INSERT`），则在执行成功后提交事务 (`db.commit()`)。若发生错误，则回滚事务 (`db.rollback()`)。

视图与触发器: 此功能不直接使用视图。不直接涉及触发器，但 `books` 表上的 `CHECK` 约束（如 `chk_total_stock` 和 `chk_available_stock`）由数据库在 `INSERT` 时自动强制执行。

3.1.2 编辑图书信息

实现过程:

1. 用户选择编辑某本图书，系统加载该图书的现有信息到编辑表单。
2. 用户修改信息并提交。
3. 应用后端接收更新后的数据。
4. 进行数据校验（必填项、库存逻辑：可借阅库存 $\leq$ 总库存，两者均非负）。
5. 构造 `UPDATE` SQL 语句，更新 `books` 表中对应 `book_id` 的记录。
6. 执行更新操作。数据库会检查 ISBN 唯一性及库存相关的 `CHECK` 约束。
7. 成功更新后，重定向到图书列表页面。

数据库事务： 编辑图书操作构成一个单一事务，通过数据库交互模块的提交和回滚机制保证。

视图与触发器： 不直接使用视图或触发器。数据库的 UNIQUE 和 CHECK 约束在此处发挥作用。

3.1.3 删除图书

实现过程：

1. 用户请求删除某本图书。
2. 应用后端首先查询 loans 表，检查该图书是否有未归还的借阅记录 (return_date IS NULL)。
3. 如果存在未归还记录，则禁止删除，并提示用户。这是通过应用层逻辑实现的，也可以通过数据库的 ON DELETE RESTRICT 外键约束（已在 loans.book_id 上定义）来强制执行，但应用层检查可以提供更友好的用户提示。
4. 如果没有未归还记录，则执行 DELETE SQL 语句从 books 表中删除该图书。
5. 成功删除后，重定向到图书列表页面。

数据库事务： 删除图书操作构成一个单一事务。

视图与触发器： 不直接使用视图。loans 表中 book_id 外键的 ON DELETE RESTRICT 约束确保了如果存在关联的借阅记录，则无法直接删除图书，从而维护了参照完整性。

3.1.4 查询与浏览图书

实现过程：

1. 用户访问图书列表页面。
2. 应用后端构造 SELECT SQL 语句从 books 表查询数据。
3. 支持基于书名、作者或 ISBN 的搜索。如果提供了搜索词，则在 WHERE 子句中添加 ILIKE (忽略大小写模糊匹配) 或 = (精确匹配) 条件。
4. 实现分页功能，使用 LIMIT 和 OFFSET 子句获取当前页的数据。同时执行 COUNT(*) 查询获取总记录数以计算总页数。
5. 将查询结果传递给模板进行展示。

数据库事务： 查询操作通常是只读的，不涉及数据修改，因此不显式开启或提交事务。

视图与触发器： 不直接使用视图或触发器。查询性能依赖于 books 表上定义的索引（如 idx_books_title, idx_books_author）。

3.2 读者管理 (Readers Management)

读者管理的添加、编辑、删除和查询功能与图书管理类似，主要区别在于操作的表是 `readers`，涉及的唯一性约束是 `reader_number`。删除读者时同样会检查其是否有未归还的借阅记录，依赖 `loans.reader_id` 外键的 `ON DELETE RESTRICT` 约束。

3.3 借阅管理 (Loans Management)

3.3.1 借书

实现过程:

1. 用户通过借书表单选择要借阅的图书、借阅的读者，并输入应归还日期。
2. 应用后端接收数据。
3. **事务开始**: 获取数据库连接和游标。
4. **库存检查**: 查询 `books` 表获取选定图书的当前 `available_stock`。为了防止并发问题，这里使用了 `SELECT ... FOR UPDATE` 来锁定该图书记录行，直到事务结束。
5. 如果 `available_stock` 大于 0，则允许借阅。
6. **插入借阅记录**: 执行 `INSERT` 语句，将 `book_id`, `reader_id`, `due_date`, `loan_date` (默认为 `CURRENT_DATE`) 插入到 `loans` 表。
7. **库存更新 (通过触发器)**: 当新的借阅记录成功插入 `loans` 表后，定义在 `loans` 表上的 `AFTER INSERT` 触发器 `trg_decrement_stock_on_borrow` 会自动执行。该触发器调用函数 `fn_decrement_stock_on_borrow`，此函数会更新 `books` 表中对应图书的 `available_stock`，将其减 1。数据库的 `CHECK` 约束 `chk_available_stock` 会确保 `available_stock` 不会变为负数。
8. **事务提交**: 如果所有操作成功，则提交事务 (`conn.commit()`)。
9. 如果库存不足或发生其他数据库错误，则回滚事务 (`conn.rollback()`) 并向用户显示错误信息。
10. **事务结束**: 关闭游标。

数据库事务: 借书操作被明确定义为一个数据库事务。它包括检查库存（带行锁）和插入借阅记录。这两个操作必须要么都成功，要么都失败回滚，以保证数据的一致性。

视图与触发器: 不直接使用视图。**触发器 `trg_decrement_stock_on_borrow`**: 在 `loans` 表 `INSERT` 后自动触发，负责减少 `available_stock`。这是将库存管理逻辑下沉到数据库的关键实现。

3.3.2 还书

实现过程:

1. 用户在当前借阅列表中选择“归还”一本特定的书。
2. 应用后端接收要归还的 `loan_id`。
3. **事务开始**: 获取数据库连接和游标。
4. **检查借阅记录**: 查询 `loans` 表, 确保该 `loan_id` 对应的记录存在且 `return_date` 为 `NULL`。同样可以使用 `SELECT ... FOR UPDATE` 锁定该借阅记录行。
5. 如果记录有效，则执行 `UPDATE` 语句，将该借阅记录的 `return_date` 设置为当前日期 (`CURRENT_DATE`)。
6. **库存更新 (通过触发器)**: 当 `loans` 表中某条记录的 `return_date` 从 `NULL` 更新为非 `NULL` 值后,定义在 `loans` 表上的 `AFTER UPDATE OF return_date` 触发器 `trg_update_stock_on_loan_change` 会自动执行。该触发器调用函数 `fn_update_stock_on_loan_change`,此函数会更新 `books` 表中对应图书的 `available_stock`,将其加 1。数据库的 `CHECK` 约束 `chk_available_stock` 会确保 `available_stock` 不会超过 `total_stock`。
7. **事务提交**: 如果所有操作成功，则提交事务。
8. 如果发生错误，则回滚事务并提示用户。
9. **事务结束**: 关闭游标。

数据库事务: 还书操作同样被定义为一个数据库事务，包括更新借阅记录和（通过触发器）更新图书库存。

视图与触发器: 不直接使用视图。**触发器 `trg_update_stock_on_loan_change`**: 在 `loans.return_date` 更新后自动触发，负责增加 `available_stock`。

3.3.3 查询当前借阅与逾期借阅

实现过程:

- **当前借阅**: 应用直接查询预定义的视图 `view_activeloans`。该视图封装了连接 `loans`, `books`, `readers` 表并筛选 `return_date IS NULL` 的逻辑。
- **逾期借阅**: 应用直接查询预定义的视图 `view_overdueloans`。该视图在 `view_activeloans` 的基础上进一步筛选 `due_date < CURRENT_DATE` 的记录。
- 查询结果传递给模板进行展示。

数据库事务: 只读查询，不涉及事务提交或回滚。

视图与触发器： 视图 `view_activeloans` 和 `view_overdueloans`：这两个视图是此功能的关键。它们将复杂的查询逻辑封装在数据库层面，使得应用层的代码非常简洁。不涉及触发器。

3.4 综合查询

3.4.1 查询读者借阅历史 / 图书借阅历史

实现过程：

- 用户选择查询某个读者或某本图书的借阅历史。
- 应用后端接收读者 ID 或图书 ID。
- 构造 `SELECT` SQL 语句，连接 `loans` 表与 `books` 表（对于读者历史）或 `readers` 表（对于图书历史）。
- 使用 `WHERE` 子句根据传入的 ID 进行筛选。
- 查询结果（包括已归还和未归还的记录）按借阅日期降序排列后展示。

数据库事务： 只读查询。

视图与触发器： 虽然这里没有直接使用之前定义的视图（因为查询目标不同），但其实现方式（JOIN 操作）与视图定义中的类似。可以考虑为这些常用历史查询也创建视图以进一步简化应用代码。不涉及触发器。查询性能依赖于 `loans` 表上为外键 `book_id` 和 `reader_id` 创建的索引。

3.5 数据库初始化

实现过程：

- 通过命令行工具（例如 Flask CLI 命令 `flask init-db`）执行。
- 该命令调用特定的初始化函数（例如 `db.py` 中的 `init_db_tables()` 函数）。
- 初始化函数读取 `schema.sql` 文件的内容。
- `schema.sql` 文件包含 `DROP TABLE`（如果存在）、`CREATE TABLE`（用于创建表和定义约束）、`CREATE INDEX`（用于创建索引）、`CREATE OR REPLACE VIEW`（用于创建或替换视图）以及 `CREATE OR REPLACE FUNCTION` 和 `CREATE TRIGGER`（用于定义触发器）等 SQL 语句。
- 这些 SQL 语句在一个事务中被执行，以确保数据库结构的完整创建。

数据库事务： 整个 `schema.sql` 文件的执行被视为一个事务。如果任何语句失败，则会回滚，防止数据库处于部分初始化的状态。

视图与触发器： 此过程直接负责在数据库中创建和定义所有的表、视图和触发器。

A 代码附录

A.1 personal_library/schema.sql

```sql
DROP TABLE IF EXISTS loans CASCADE;
DROP TABLE IF EXISTS books CASCADE;
DROP TABLE IF EXISTS readers CASCADE;

CREATE TABLE readers (
    reader_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    reader_number VARCHAR(50) UNIQUE NOT NULL,
    contact VARCHAR(100)
);

CREATE TABLE books (
    book_id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(100) NOT NULL,
    isbn VARCHAR(20) UNIQUE NOT NULL,
    publisher VARCHAR(100),
    publication_year INTEGER,
    category VARCHAR(50),
    total_stock INTEGER NOT NULL DEFAULT 0,
    available_stock INTEGER NOT NULL DEFAULT 0,
    CONSTRAINT chk_total_stock CHECK (total_stock >= 0),
    CONSTRAINT chk_available_stock CHECK (available_stock >= 0 AND
        available_stock <= total_stock)
);

CREATE TABLE loans (
    loan_id SERIAL PRIMARY KEY,
    book_id INTEGER NOT NULL REFERENCES books(book_id) ON DELETE RESTRICT,
    reader_id INTEGER NOT NULL REFERENCES readers(reader_id) ON DELETE
        RESTRICT,
    loan_date DATE NOT NULL DEFAULT CURRENT_DATE,
    due_date DATE NOT NULL,
    return_date DATE DEFAULT NULL
);


CREATE INDEX idx_books_title ON books(title);
CREATE INDEX idx_books_author ON books(author);
CREATE INDEX idx_books_category ON books(category);

CREATE INDEX idx_readers_name ON readers(name);

CREATE INDEX idx_loans_book_id ON loans(book_id);
CREATE INDEX idx_loans_reader_id ON loans(reader_id);
```

```sql
CREATE INDEX idx_loans_due_date ON loans(due_date);
CREATE INDEX idx_loans_return_date ON loans(return_date);

CREATE OR REPLACE VIEW view_activeloans AS
SELECT
    L.loan_id,
    R.name AS reader_name,
    R.reader_number,
    B.title AS book_title,
    B.isbn,
    L.loan_date,
    L.due_date
FROM loans L
JOIN books B ON L.book_id = B.book_id
JOIN readers R ON L.reader_id = R.reader_id
WHERE L.return_date IS NULL;

CREATE OR REPLACE VIEW view_overdueloans AS
SELECT
    L.loan_id,
    R.name AS reader_name,
    R.reader_number,
    R.contact AS reader_contact,
    B.title AS book_title,
    B.isbn,
    L.loan_date,
    L.due_date
FROM loans L
JOIN books B ON L.book_id = B.book_id
JOIN readers R ON L.reader_id = R.reader_id
WHERE L.return_date IS NULL AND L.due_date < CURRENT_DATE;

CREATE OR REPLACE FUNCTION fn_decrement_stock_on_borrow()
RETURNS TRIGGER AS $$
BEGIN

    IF NEW.return_date IS NULL THEN
        UPDATE books
        SET available_stock = available_stock - 1
        WHERE book_id = NEW.book_id;

    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_decrement_stock_on_borrow
AFTER INSERT ON loans
FOR EACH ROW
```

```sql
    EXECUTE FUNCTION fn_decrement_stock_on_borrow();

    CREATE OR REPLACE FUNCTION fn_update_stock_on_loan_change()
    RETURNS TRIGGER AS $$
    BEGIN
        IF OLD.return_date IS NULL AND NEW.return_date IS NOT NULL THEN
            UPDATE books
            SET available_stock = available_stock + 1
            WHERE book_id = NEW.book_id;
        ELSIF OLD.return_date IS NOT NULL AND NEW.return_date IS NULL THEN
            UPDATE books
            SET available_stock = available_stock - 1
            WHERE book_id = NEW.book_id;
        END IF;
        RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;

    CREATE TRIGGER trg_update_stock_on_loan_change
    AFTER UPDATE OF return_date ON loans
    FOR EACH ROW
    EXECUTE FUNCTION fn_update_stock_on_loan_change();

    CREATE OR REPLACE FUNCTION fn_increment_stock_on_loan_delete()
    RETURNS TRIGGER AS $$
    BEGIN
        IF OLD.return_date IS NULL THEN
            UPDATE books
            SET available_stock = available_stock + 1
            WHERE book_id = OLD.book_id;
        END IF;
        RETURN OLD;
    END;
    $$ LANGUAGE plpgsql;

    CREATE TRIGGER trg_increment_stock_on_loan_delete
    AFTER DELETE ON loans
    FOR EACH ROW
    EXECUTE FUNCTION fn_increment_stock_on_loan_delete();
```

Listing A.1: personal_library/schema.sql SQL 脚本