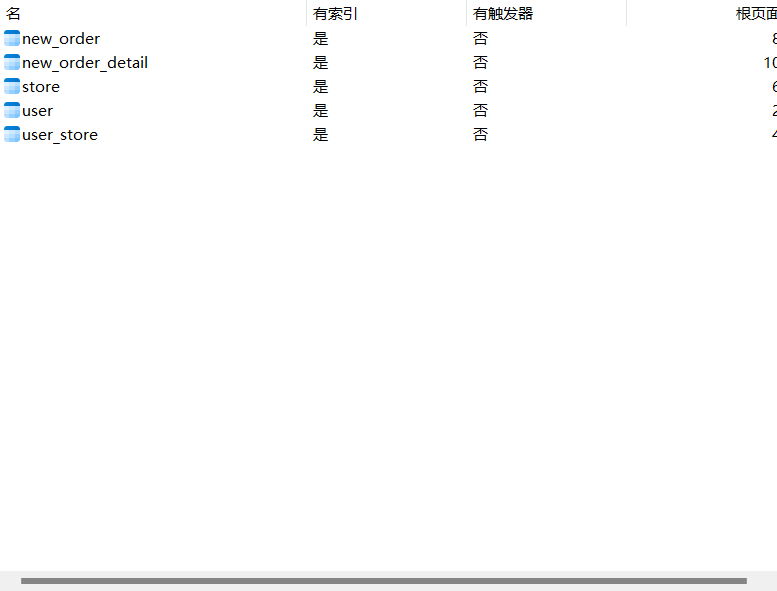
**设计思路和数据库转化**

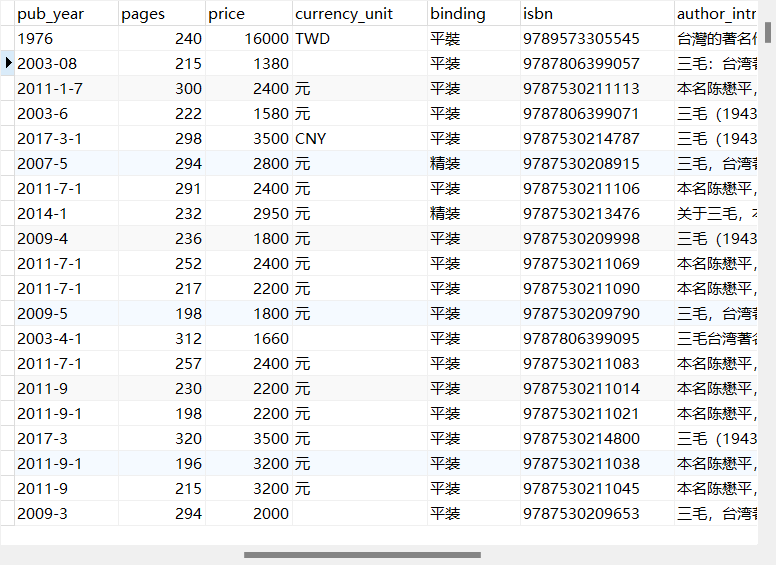
实验开始前我们一共有两个SQLite数据库，分别是be.db和book\_lx.db

Be.db中有这些表格，其中user和user\_store是有数据的，记录用户id和其对应的店铺id



Book\_lx.db中则记录了一些书的基本信息







我们的的数据库设计包括book、store、user、new\_order\_detail、new\_order和user\_store一共五张数据表。

**1. 书籍表 (book):**

存储书籍的详细信息。

字段有：书籍ID（id），书名（title），作者（author），出版社（publisher），原书名（original\_title），翻译（translator），出版年（pub\_year），页数（pages），价格（price），货币单位（currency\_unit），装订方式（binding），国际标准书号（isbn），作者介绍（author\_intro），书籍介绍（book\_intro），内容摘要（content），标签（tags）和图片（picture，以二进制大对象存储）。

**2. 商店表 (store):**

记录商店和存储的书籍以及书籍的价格和库存水平。

字段有：商店ID（store\_id），书籍ID（book\_id）用于与书籍表关联，价格（price）和库存水平（stock\_level）。

**3. 用户表 (user):**

存储用户基本信息，包括买家和卖家。

字段有：用户ID（user\_id），密码（password），账户余额（balance），认证令牌（token）和用户终端信息（terminal）。

new\_order 和 new\_order\_detail 表存储订单的信息。

user\_store 表关联用户和他们的商店。

**4. 订单详情表 (new\_order\_detail):**

提供订单的详细信息，包括购买的书籍和数量。字段有：订单ID（order\_id）用于与订单表关联，书籍ID（book\_id）用于与书籍表关联，价格（price），数量（count），状态（state），订单时间（order\_time），发货时间（delivery\_time），收货时间（receipt\_time），付款时间（payment\_time）。

**5. 订单表 (new\_order):**

记录用户订单信息。

字段有：订单ID（order\_id），用户ID（user\_id）用于与用户表关联，商店ID（store\_id）用于与商店表关联，以及付款时间（payment\_time）。

**6. 用户商店关联表 (user\_store):**

跟踪哪个用户拥有哪个商店。

字段有：用户ID（user\_id）和商店ID（store\_id）。

**关系描述**

store 和 book：一对多关系，表示一个商店可以销售多本书。通过 store 实体中的 book\_id 字段来实现关联。

User 和 New\_Order：一对多关系，表示一个用户可以有多个订单。通过 new\_order实体中的 order\_id 字段来实现关联。

New\_Order 和 New\_Order\_Detail：一一对应的关系，通过 order\_id 字段来实现关联。

User 和 Store：通过 user\_store 实体建立一对多的关系，表示一个用户（卖家）可以开多个商店。

Store 和 New\_Order：一对多关系，表示一个商店可以有多个订单。通过store 实体中的 store\_id 字段来实现关联。

User 和 User\_Store：一一对应关系，通过 两个实体中的 user\_id 字段来实现关联。

**设计原因**

**书籍详细信息：**独立存储在 book 表中，以避免冗余，并便于管理。

**商店和库存管理：**store 表连接了书籍和商店，便于跟踪每本书在不同商店的库存和价格，为库存管理提供了基础。

**用户信息安全：**user 表中的密码字段应使用加密存储，token 字段用于认证和会话管理，提高安全性。

**订单处理：**new\_order 和 new\_order\_detail 表之间的关系允许系统记录订单的总体信息和具体的每项订单细节，如单个书籍的购买数量和价格等。这样的设计有助于简化订单处理流程和历史订单的查询。

**用户商店关系：**user\_store 表说明一个用户可以拥有多个商店，这为平台上的卖家提供了灵活性，使他们可以经营多个商店。

**数据正规化：**整个设计通过关联不同的表，避免了数据冗余，使得数据库更加正规化。例如，书籍信息、用户信息、订单信息都分别存储在独立的表中，表与表之间可以有关联。

**扩展性和灵活性**：这种设计也考虑到了未来可能的扩展性，比如添加新的字段或表以支持新功能，例如亮点功能推荐系统。

**性能考量：**分开存储信息可以提高查询效率，特别是对于复杂的查询操作，如联合多个表来检索订单详情或计算商店的总销售额等。

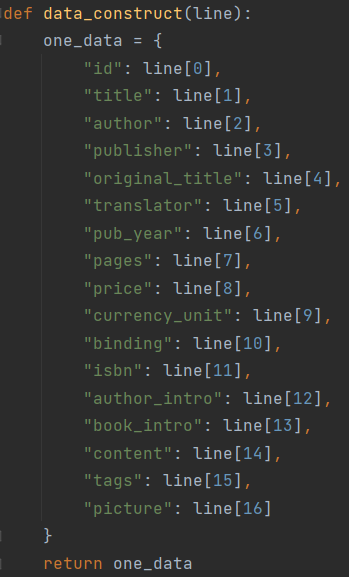
总的来说，我们提供了一个结构化且灵活的数据库设计，它能够支持一个在线书店后端所需的所有基本功能，包括用户管理、库存管理、订单处理以及书籍的搜寻和购买。同时，它也考虑到了安全性、性能和未来的可扩展性。

**数据库转化：**

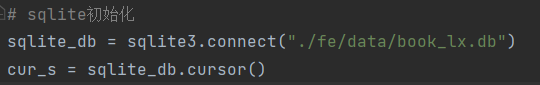
根据demo给出的be.db和book\_lx.db 两个SQLite 数据库，我们需要创建本地的MongoDB 数据库，将两个SQLite 数据库中的内容以合适的形式存入本地数据库，后续所有数据读写都在本地的 MongoDB 数据库中进行。所以简单来说，我们要进行从 SQLite 数据库到 MongoDB 数据库的数据迁移操作。

**book\_lx.db数据库迁移**

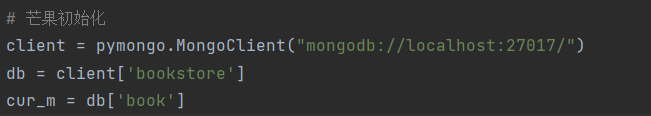
定义构造函数：data\_construct 函数接受一个包含书籍信息的行（line），然后将这些信息构造成一个字典（one\_data），字典的键是书籍的各个属性，值是来自输入行的相应数据。



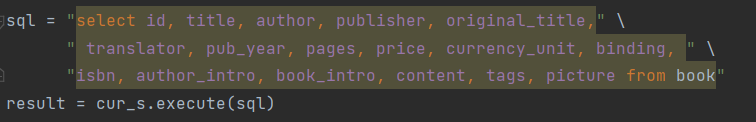
连接 SQLite 数据库：使用 sqlite3.connect 连接到一个名为 book\_lx.db 的 SQLite 数据库文件，并创建一个游标 cur\_s 以便执行 SQL 查询。



连接 MongoDB 数据库：使用 pymongo.MongoClient 连接到运行在本地的 MongoDB 实例，选择 bookstore 数据库，以及该数据库中的 book 集合（类似于 SQL 中的表）。



执行 SQL 查询：构造一个 SQL 查询，选择 book 表中所有的列，并执行这个查询。



数据迁移：对于 SQL 查询的结果，迭代每一行结果，使用 data\_construct 函数将每行数据转换为字典，并将这些字典存储在 book\_data 列表中。最后，使用 MongoDB 的 insert\_many 方法将这个列表中的所有字典插入到 MongoDB 的 book 集合中。



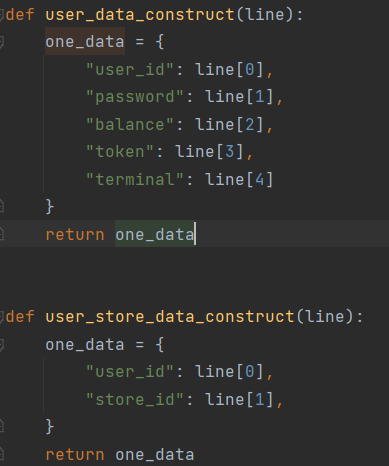
**be.db数据库迁移**

由于be.db数据库中只有user和user\_store表是有数据的，所以我们只需要对它们进行数据迁移的操作。

**定义数据构造函数：**

user\_data\_construct：这个函数负责将 user 表中的一行数据转换成字典格式。

user\_store\_data\_construct：这个函数负责将 user\_store 表中的一行数据转换成字典格式。



这两个函数的目的是为了将 SQLite 数据库中的表行数据转换为 MongoDB 可以接受的文档（字典）格式。

**连接数据库：**

通过 sqlite3.connect 连接到 SQLite 数据库文件（./fe/data/be.db）。

使用 pymongo.MongoClient 连接到本地运行的 MongoDB 实例，并选择 bookstore 数据库中的 user 和 user\_store 集合。

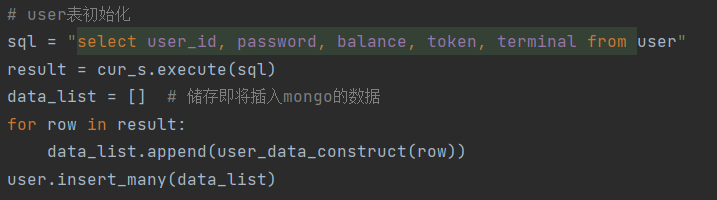


**迁移 user 表：**

使用 SQL 语句从 SQLite 的 user 表中查询所需的字段。

通过迭代查询结果，使用 user\_data\_construct 函数构造字典列表。

使用 MongoDB 的 insert\_many 方法批量插入字典列表到 user 集合中。



**迁移 user\_store 表：**

使用 SQL 语句从 SQLite 的 user\_store 表中查询所需的字段。

与 user 表类似，迭代查询结果，并使用 user\_store\_data\_construct 函数构造字典列表。

使用 insert\_many 方法批量插入字典列表到 user\_store 集合中。

