实验5 数据库程序设计

一、实验目的

1、 设计并实现一个精简的图书管理系统，具有入库、查询、借书、还书、借书证管理等基本功能。

2、 通过本次设计来加深对数据库的了解和使用，同时提高自身的系统编程能力。

二、实验平台

开发工具：VSCode IDE 、 .Net开发工具；数据库平台：SQLServer

三、总体设计

1、系统架构描述

本系统主要包括管理员登录、用户登录、图书管理、图书查询、借还书管理、借书证管理六大功能模块。系统处理基本流程如下：

系统初始化

借还书操作

图书管理

借书证管理

图书查询

用户登录

管理员登陆

系统初始化时仅有用户登录和管理员登陆两个选项卡，未登陆时仅可以查看图书馆藏书情况，管理员登陆后，可以进行图书入库操作、借书证管理操作、图书查询操作；用户登录后可以进行借还书操作和图书查询操作。

各个模块功能说明如下：

|  |  |
| --- | --- |
| 模块名称 | 功能描述 |
| 管理员登陆 | 输入管理员密码进行登录（默认为 admin） |
| 图书入库 | 1. 单本图书入库，需依次输入类目、书名、奖项、出版时间、作者、价格、存货数量。（ID会自动分配） 2. 批量导入图书，从外部excel文件中批量导入图书，excel文件中按类目、书名、奖项、出版时间、作者、价格、存货数量的格式记录。 3. 更新书籍，填入书籍ID和你所需要更新的信息即可。 4. 删除书籍，填入书籍ID即可删除对应书籍。 |

|  |  |
| --- | --- |
| 图书查询 | 1. 按书的名称、作者进行查询 |
| 借还书 | 1. 借书时，输入对应书籍ID即可借取书籍，借取书籍后，会生成一条借书记录，包括Card ID、Book ID、   借书时间、和还书时间（如已归还），若还未归还则会显示 Not Returned。借书成功后对应书籍存货减一，存货为零则无法借取。   1. 还书时，输入对应书籍ID即可归还书籍，归还书籍后，会更新对应借书记录中的Not Returned为归还时间。归还后，对应书籍存货加一。 |
| 借书证管理 | 1. 添加新借书证 2. 更新借书证信息 3. 删除借书证 |
| 用户登录 | 输入借书证id和密码进行登录（密码默认为password） |

1. 数据库表格设计

图书信息表格：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 主键 | 说明 |
| book\_Id | Int (not null) | Y | 书籍ID |
| category | varchar(63) (not null) |  | 类目 |
| title | varchar(63) (not null) |  | 书名 |
| press | varchar(63) (not null) |  | 奖项 |
| Publish\_year | int (not null) |  | 出版时间 |
| author | varchar(63) (not null) |  | 作者 |
| price | decimal(7,2) (not null) |  | 价格 |
| stock | int (not null) |  | 存货 |

借书证表格：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 主键 | 说明 |
| card\_id | int (not null) | Y | 借书证ID |
| name | varchar(63) (not null) |  | 持证人姓名 |
| department | varchar(63) (not null) |  | 持证人所在学院 |
| type | char(1) (not null) |  | 证件类型（老师or 学生） |

借书记录表格：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 主键 | 说明 |
| card\_id | int (not null) | Y | 书籍ID |
| book\_id | int (not null) | Y | 借书证ID |
| borrow\_time | int (not null) |  | 借书时间 |
| return\_time | int (not null) |  | 还书时间 |

1. 开发技术：

本次图书管理系统的开发采用了 SQL Server 和 C#（.NET 、 AOD）技术，具体使用了 SQL Server 作为数据库管理系统，并利用 C# 的强大功能进行业务逻辑的处理。下面从这两方面来进行说明：

SQL Server：

SQL Server 是微软公司开发的关系数据库管理系统（RDBMS），广泛应用于各类数据存储与管理任务。SQL Server 在数据存储和查询方面具有高效的性能，支持复杂的查询语句和事务管理，能够满足图书管理系统对数据高效、可靠存储的需求。通过 SQL Server，系统能够方便地进行数据库设计与管理，包括创建数据库、表格、视图、存储过程等。在图书管理系统中，我们利用 SQL Server 进行数据库的创建和数据的管理，借助 SQL Server 提供的图形化管理工具，能够直观地创建和维护数据表，轻松地进行数据的增、删、改、查操作。同时，SQL Server 也提供了强大的 SQL 编辑器，使得在执行 SQL 语句之前，可以先进行语法检查，减少错误的发生，提升开发效率。

C#（.NET）：

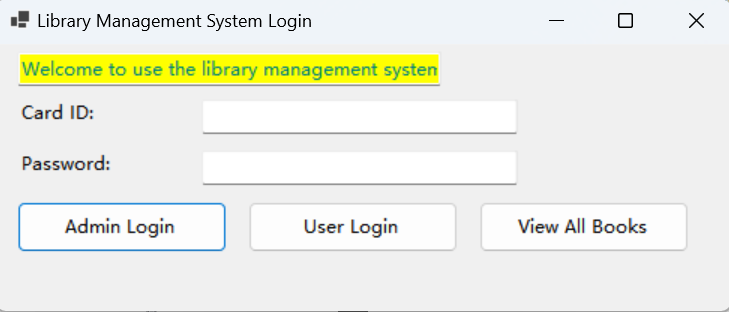
C# 是一种面向对象的编程语言，结合 .NET 框架提供了强大的功能和丰富的类库，适用于开发各种类型的应用程序。本次图书管理系统的业务逻辑部分主要使用 C#（.NET）进行开发，利用其强大的对象模型和丰富的 API，实现了系统的前端界面与后端数据库之间的高效交互。C# 的语言特性，如垃圾回收、类型安全、类库支持等，使得开发过程更加高效和稳定。系统的用户界面设计使用 Windows Forms，配合 C# 提供的控件和事件驱动模型，开发人员能够快速构建出图形化界面，并在用户与系统之间建立便捷的交互。

ADO.NET：

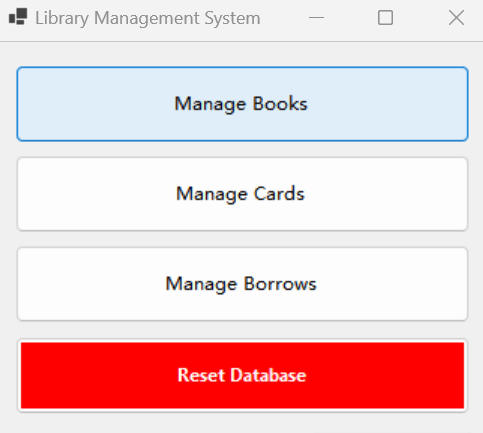
在数据库访问方面，C#（.NET）提供了 ADO.NET 技术，用于与 SQL Server 数据库进行数据交互。ADO.NET 是一种高度集成的数据库访问框架，能够实现数据的查询、更新、插入和删除等操作。其主要组成包括 SqlConnection、SqlCommand、SqlDataReader 和 SqlDataAdapter 等组件，可以方便地连接数据库，执行 SQL 语句，并将数据返回到应用程序中。在图书管理系统中，我们通过 SqlConnection 组件与 SQL Server 建立连接，使用 SqlCommand 执行 SQL 查询和更新操作，利用 SqlDataReader 获取查询结果，以便显示在界面上。通过 ADO.NET，开发人员能够以简单而高效的方式与数据库进行通信，保证系统的性能和可靠性。

1. 详细设计原理：  
   1、主界面设计

程序界面采用了多窗体模式，通过标签页或导航栏的方式进行模块化管理。下方为导航区域，用户可以点击不同的按钮来切换不同的功能模块，点击后对跳转到对应模块的窗体。主界面会显示登陆模块，你可以通过下方按钮来选择登陆身份：用户登录、管理员登录或者查看全部书籍。

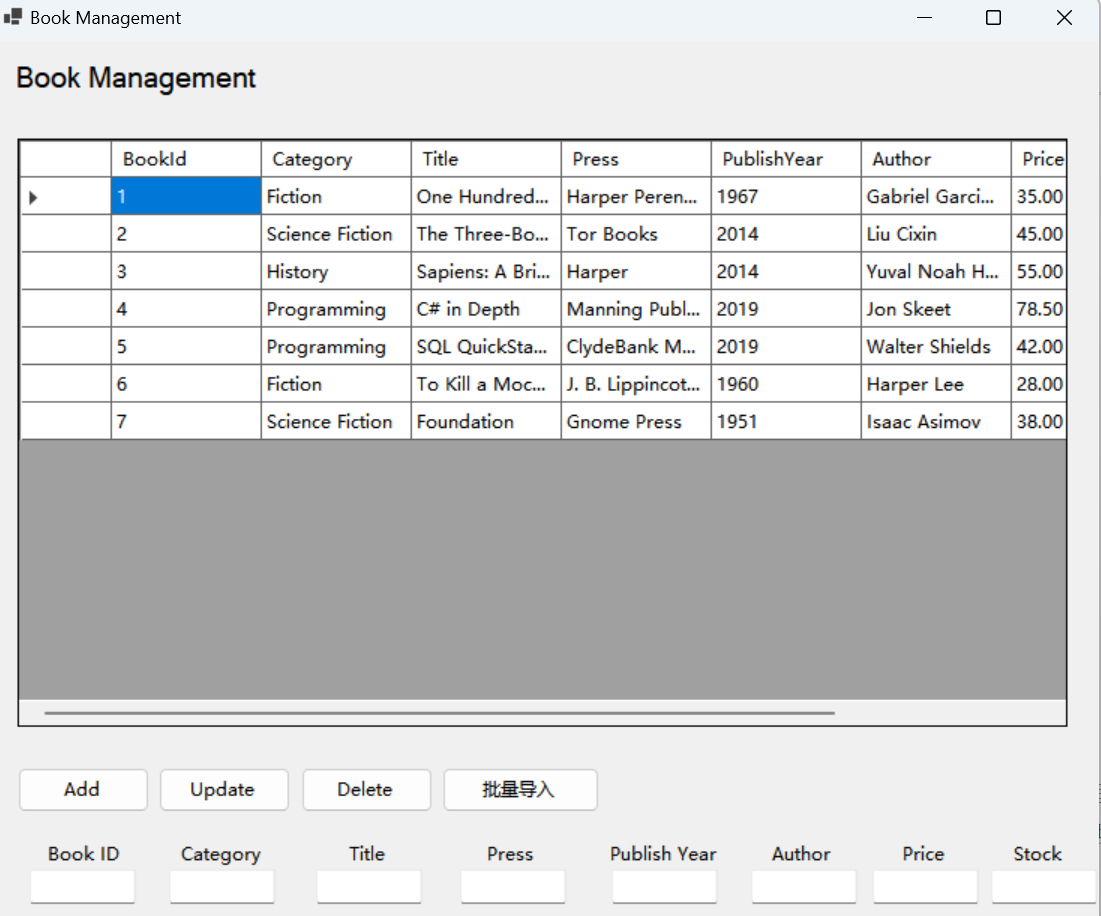


管理员登陆后，会显示新的窗体，包含Manage Books , Manage Cards , Manage Borrows , Reset Database四个功能：



1. 图书管理模块：

图书管理模块实现了四个功能：图书入库（单本和批量）、图书更新、图书删除。



2.1**单本入库**：

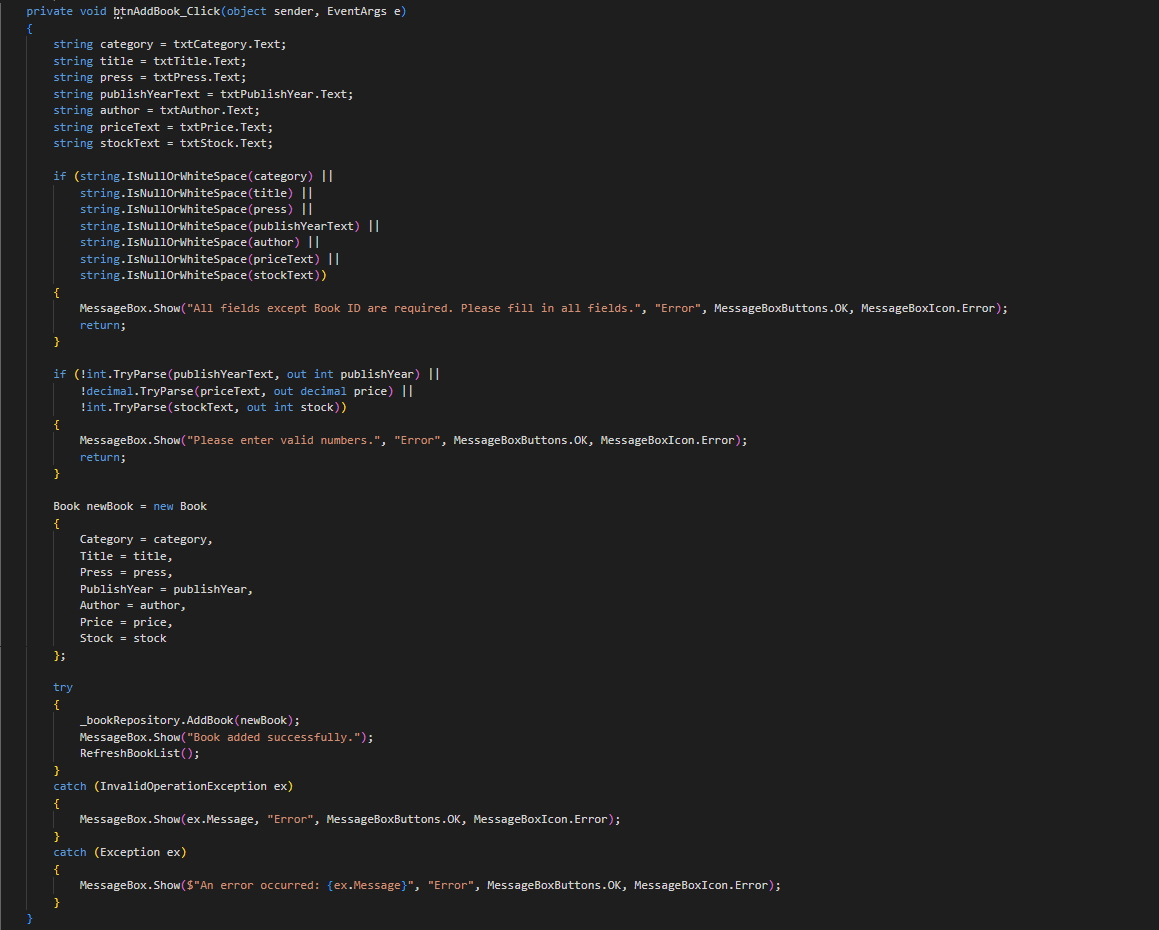
依次输入类目、书名、奖项、出版时间、作者、价格、存货数量（ID无需输入、系统自动按序分配）后，单击Add即可实现入库。

**主要逻辑代码**：



首先，定义了一个 INSERT SQL 语句，使用参数化查询将书籍的各项信息（如类别、标题、作者等）填充到 SQL 语句的占位符中。通过 GetConnection() 获取数据库连接，并使用 OdbcCommand 执行 SQL 命令。在执行插入操作时，如果插入的数据违反唯一约束（如重复的书籍记录），则抛出自定义异常，提示用户无法创建重复的书籍。其他数据库错误则会被重新抛出，确保能够进一步处理。

**前端窗口代码**：

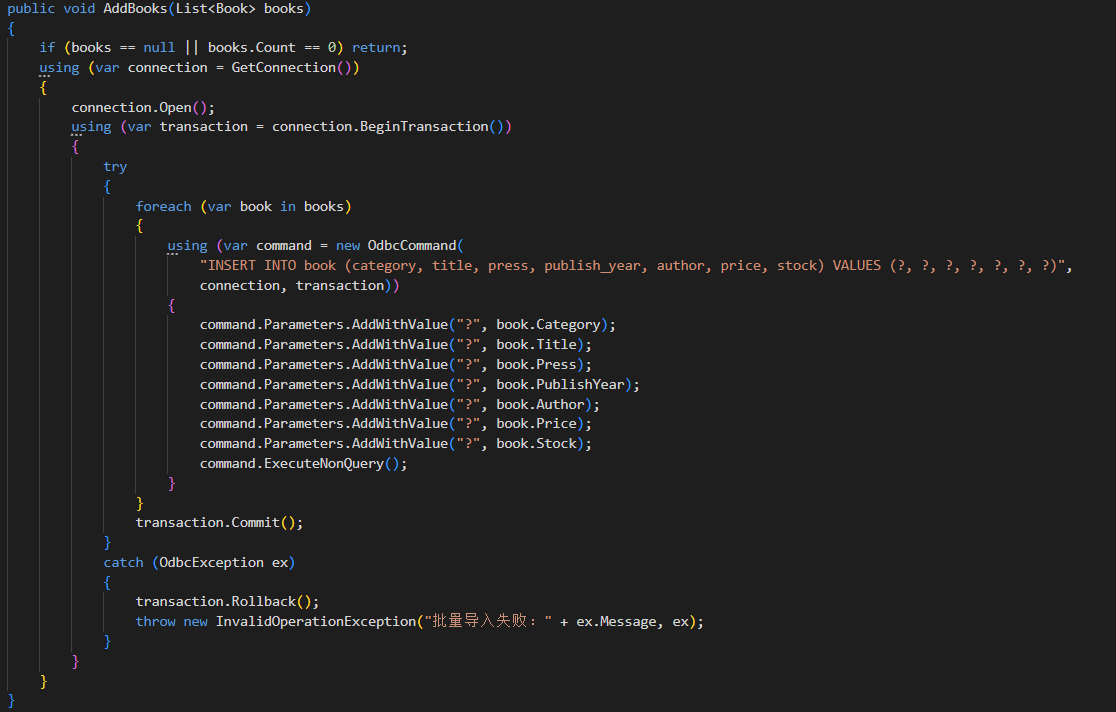


首先，它从前端界面的文本框中获取用户输入的数据，并进行验证，确保所有必填字段均已填写，且出版年份、价格和库存等数值字段符合正确格式（使用 int.TryParse 和 decimal.TryParse 进行格式验证）。如果验证失败，系统会弹出错误提示，要求用户重新输入。在数据验证通过后，创建一个新的 Book 对象，并将用户输入的数据填充到该对象中。接着，调用 AddBook 方法将 Book 对象传递给数据访问层，负责将书籍信息存储到数据库中。如果操作成功，用户会收到提示；若发生数据库相关错误（如唯一约束违规），则捕获特定的异常并显示相应的错误信息。

2.2**批量入库**：

从外部excel文件中批量导入图书，excel文件中按类目、书名、奖项、出版时间、作者、价格、存货数量的格式记录

**主要逻辑代码**：



首先，检查传入的书籍列表是否为空或无书籍，若是则直接返回，避免无效操作。接着，使用 GetConnection() 获取数据库连接，并在连接上开启事务。通过事务确保批量插入操作的原子性，要么全部成功，要么全部失败。对于每本书籍，创建一OdbcCommand 对象，使用参数化查询的方式将书籍信息插入到数据库的 book

表中。在循环过程中，逐一将书籍的各项属性绑定到 SQL 命令的占位符上，并执行 ExecuteNonQuery() 方法进行插入。如果所有书籍都成功插入，事务会通过 Commit() 提交，确保数据的持久化。如果在插入过程中发生异常，事务会通过 Rollback() 回滚，确保数据库不受部分数据插入的影响。捕获到的异常会抛出 InvalidOperationException，并附带详细的错误信息。

**前端窗口代码**：



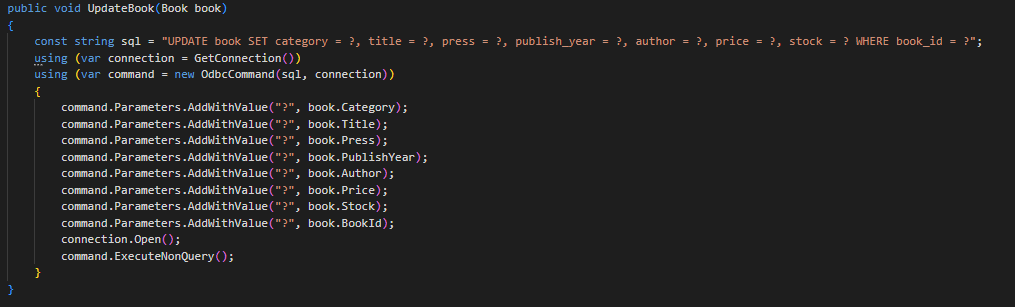


首先，方法通过 OpenFileDialog 弹出文件选择框，让用户选择一个 CSV 文件。文件过滤器限制为 CSV 格式，确保用户选择正确类型的文件。当用户选择文件后，方法尝试读取该文件的内容，使用 System.IO.File.ReadAllLines 读取所有行数据。每一行代表一条书籍记录，假设 CSV 文件的字段按顺序分别为：类别、标题、出版社、出版年份、作者、价格和库存。程序将每行数据拆分成数组，并通过 int.TryParse 和 decimal.TryParse 对出版年份、价格和库存进行格式验证，确保其为合法数字。对于有效的书籍数据，程序创建 Book 对象并将其添加到 books 列表中。如果列表中有有效的书籍数据，接着调用 \_bookRepository.AddBooks(books) 将其批量插入数据库。导入成功后，系统弹出提示框显示成功导入的书籍数量，并刷新书籍列表；如果没有有效数据或发生错误，系统会给出相应的提示。

2.3**图书更新**

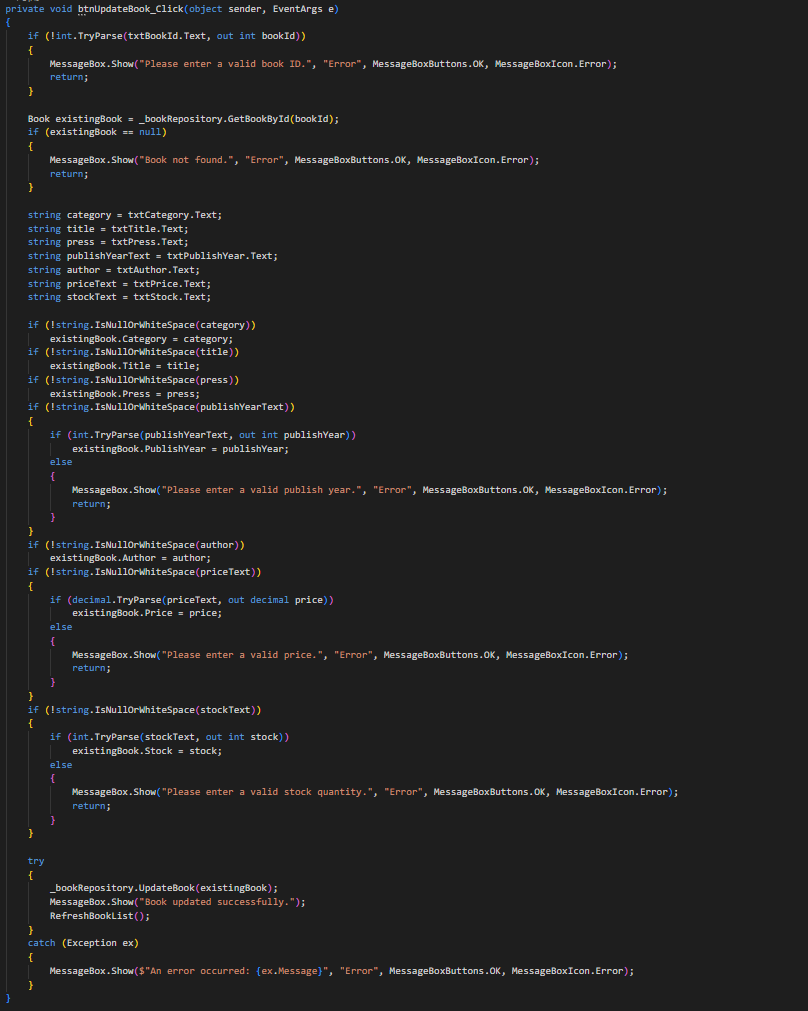
图书更新，填入书籍ID和所需要更新的信息后单击update即可。

**主要逻辑代码**：



首先，定义了一个 UPDATE SQL 语句，通过书籍的 book\_id 唯一标识书籍记录，并更新相应的书籍属性（如类别、标题、出版社、出版年份、作者、价格和库存）。在执行更新操作前，通过 GetConnection() 获取数据库连接，并使用 OdbcCommand 对象执行 SQL 命令。通过参数化查询，将 book 对象的各个属性（如 Category、Title、Press 等）绑定到 SQL 语句中的占位符，连接打开后，执行 ExecuteNonQuery() 执行 SQL 语句，更新数据库中对应书籍的信息。

**前端代码：**



首先，通过 txtBookId 获取用户输入的书籍 ID，并验证其是否为有效的整数。如果无效，显示错误提示并退出方法。接着，通过 GetBookById 方法从数据库获取该 ID 对应的书籍记录。如果未找到书籍，系统会提示用户并停止操作。随后，程序从文本框获取用户输入的新书籍信息，并逐一验证每个字段。如果字段

不为空且格式正确（如出版年份、价格和库存需要是有效的数字），则更新对应书籍对象的属性。若用户输入格式不正确，程序会显示相应的错误信息并终止更新。最后，调用 \_bookRepository.UpdateBook(existingBook) 将更新后的书籍信息保存到数据库中。如果更新成功，显示成功提示并刷新书籍列表；如果发生异常，则捕获并显示错误信息。

2.4**图书删除**

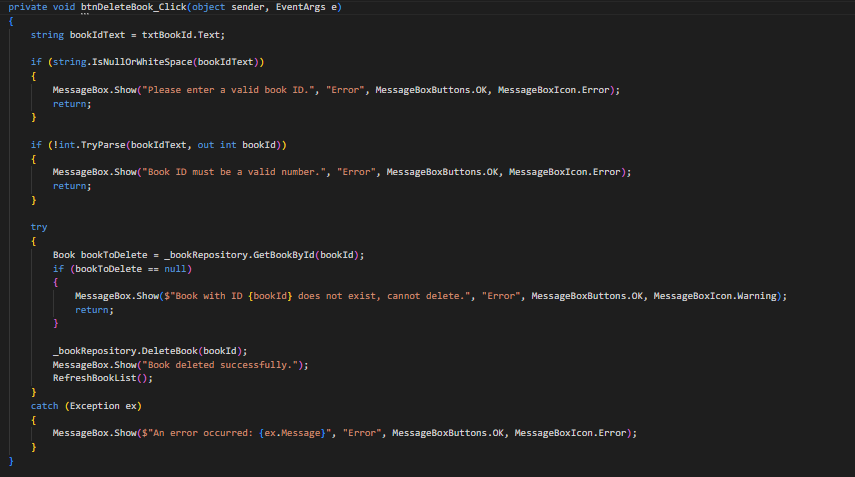
删除书籍，填入书籍ID后单击delete即可删除对应书籍。

**主要逻辑代码**：



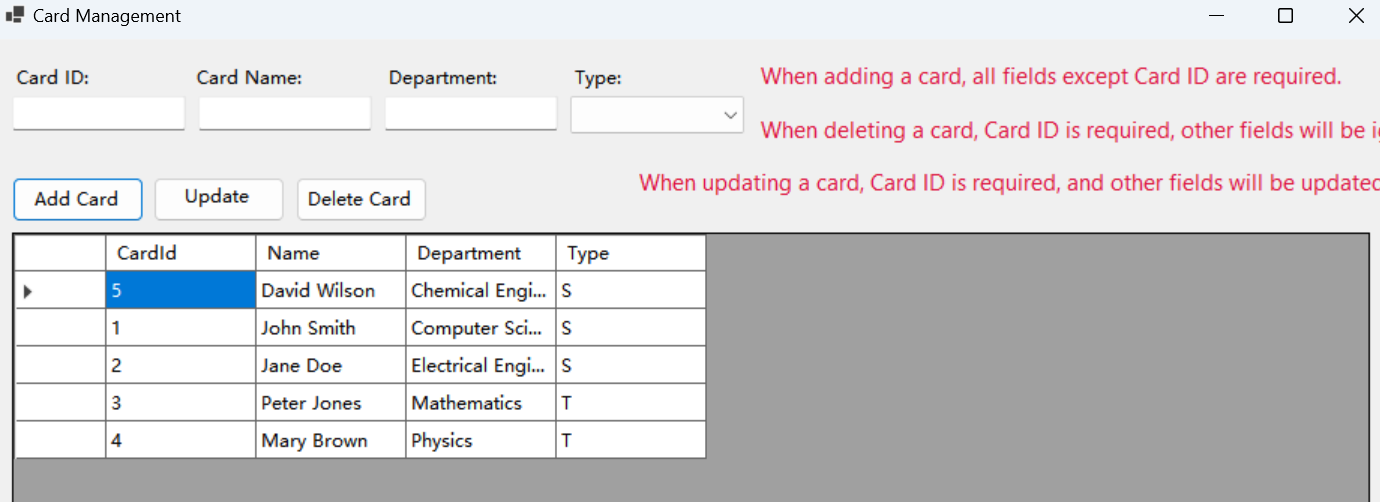
首先，定义了一个 DELETE SQL 语句，根据给定的 book\_id 删除数据库中对应的书籍记录。通过 GetConnection() 获取数据库连接，并使用 OdbcCommand 创建 SQL 命令。接着，使用参数化查询将 bookId 绑定到 SQL 语句中的占位符；连接打开后，调用 ExecuteNonQuery() 执行 SQL 删除命令，删除指定 book\_id 的书籍记录。

**前端代码：**



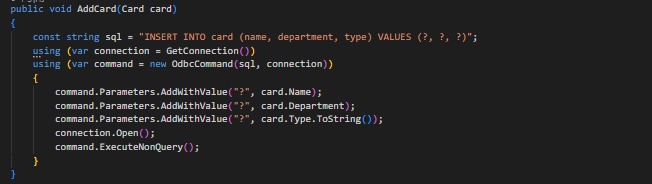
首先，通过 txtBookId 获取用户输入的书籍 ID，并检查输入是否为空或格式是否正确（即是否为有效的整数）。如果输入无效，显示错误信息并停止执行。接着，程序调用 \_bookRepository.GetBookById(bookId) 方法检查数据库中是否存在该书籍。如果未找到对应书籍，显示警告信息并结束操作。如果书籍存在，调用 \_bookRepository.DeleteBook(bookId) 方法删除书籍。删除成功后，显示成功提示框，并通过 RefreshBookList() 刷新书籍列表。如果过程中发生任何异常，程序会捕获并显示详细的错误信息。

1. 借书证管理模块：

借书证管理模块实现了3个功能：添加借书证、更新借书证信息、删除借书证。  


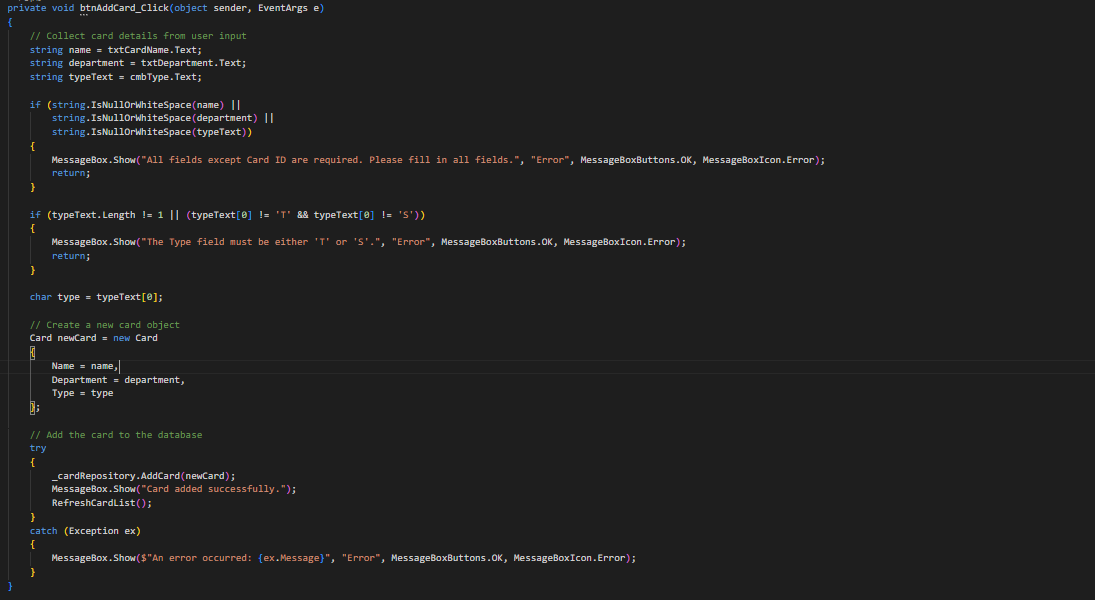
3.1**添加借书证**

填写借书证的姓名、部门和类型（'T' 或 'S'）信息来添加新的借书证。

**主要逻辑代码**：  


首先，检查用户输入的各个字段，确保姓名、部门和类型不能为空，并且类型必须是 'T' 或 'S'。若输入无效，系统会弹出错误提示，要求用户重新输入。验证通过后，程序会创建一个新的 Card 对象，将用户输入的数据填充到该对象中。接着，调用 CardRepository 的 AddCard 方法将借书证信息存入数据库。如果操作成功，系统会提示用户借书证已成功添加，并刷新显示借书证列表。

**前端代码：**

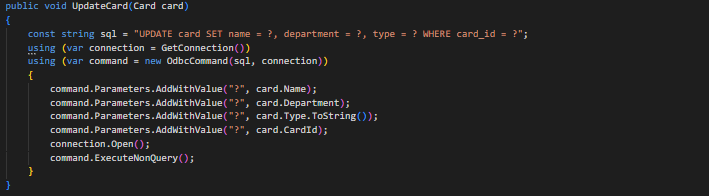


首先，它从用户输入的文本框（txtCardName、txtDepartment 和 cmbType）获取卡片的详细信息。如果任何一个必填字段为空或未填写，程序会弹出错误提示，并终止执行。接着，对卡片类型字段（Type）进行验证，确保用户输入的是字符 'T' 或 'S'。如果用户输入无效，程序会显示错误提示并停止操作。如果验证通过，将卡片类型转换为字符类型 char。之后，创建一个新的 Card 对象，将用户输入的数据赋值给相应属性。紧接着，调用 AddCard 方法将新的卡片信息添加到数据库中。在操作过程中，出现任何错误时会捕获并提示给用户。若卡片添加成功，程序会显示成功信息并刷新卡片列表。

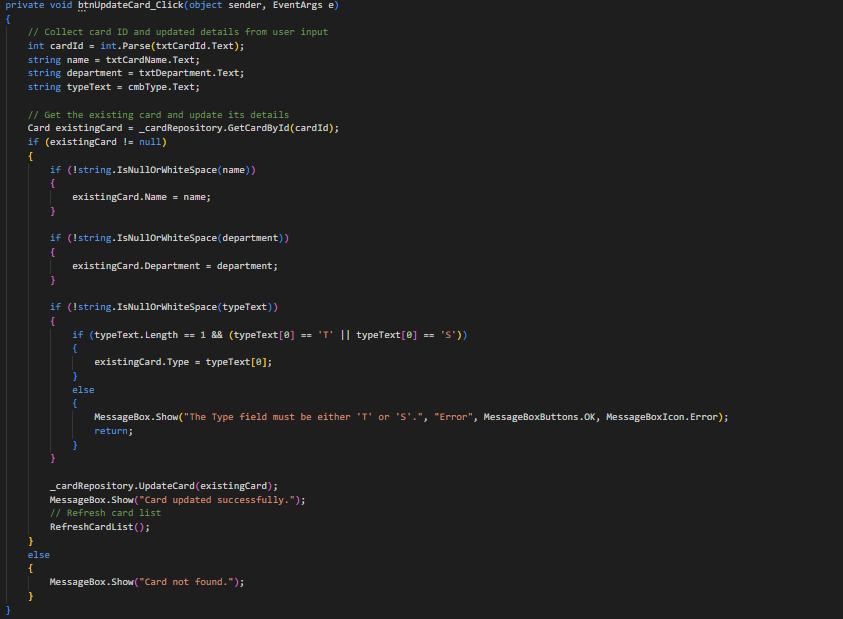
3.2**更新借书证**

填写借书证的id和需要更新的信息后单击updata即可更新信息。

**主要逻辑代码**：



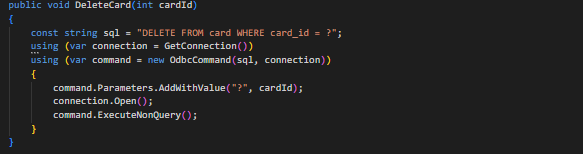
首先，定义一个 SQL UPDATE 语句，该语句通过 card\_id 唯一标识需要更新的卡片记录，并更新卡片的 name、department 和 type 属性。使用 GetConnection() 方法获取数据库连接，并通过 OdbcCommand 创建 SQL 命令对象。通过参数化查询，将 card 对象的各个属性（如 Name、Department 和 Type）绑定到 SQL 语句中的占位符，调用 connection.Open() 打开数据库连接，并执行 ExecuteNonQuery() 执行 SQL 语句，将更新的卡片信息写入数据库。

**前端代码：**

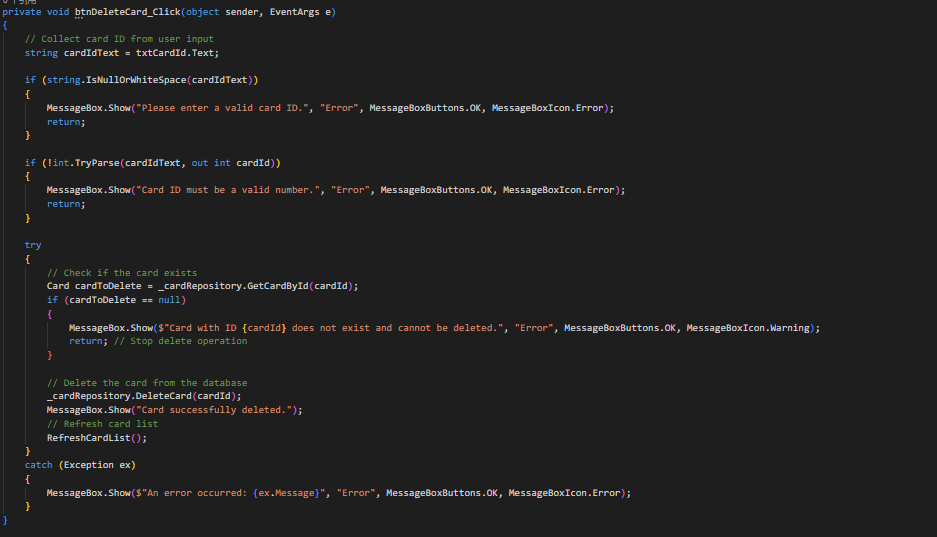
首先，从用户输入的文本框收集卡片 ID 和需要更新的卡片信息。通过卡片 ID 调用 \_cardRepository.GetCardById(cardId) 来从数据库中获取已存在的卡片记录。如果卡片存在，逐个检查用户输入的字段是否为空，并根据输入更新卡片对象的相应属性。如果卡片名称 (Name) 不为空，则更新卡片的名称。如果卡片部门 (Department) 不为空，则更新卡片的部门。对卡片类型 (Type) 进行验证，确保用户输入的是 'T' 或 'S'，如果无效，则弹出错误提示，终止更新。如果所有输入均有效，程序通过 \_cardRepository.UpdateCard(existingCard) 更新卡片信息并保存至数据库。更新成功后，显示成功提示并刷新卡片列表。

3.3**删除借书证：**填写借书证的id后单击delete即可删除借书证。

**主要逻辑代码**：



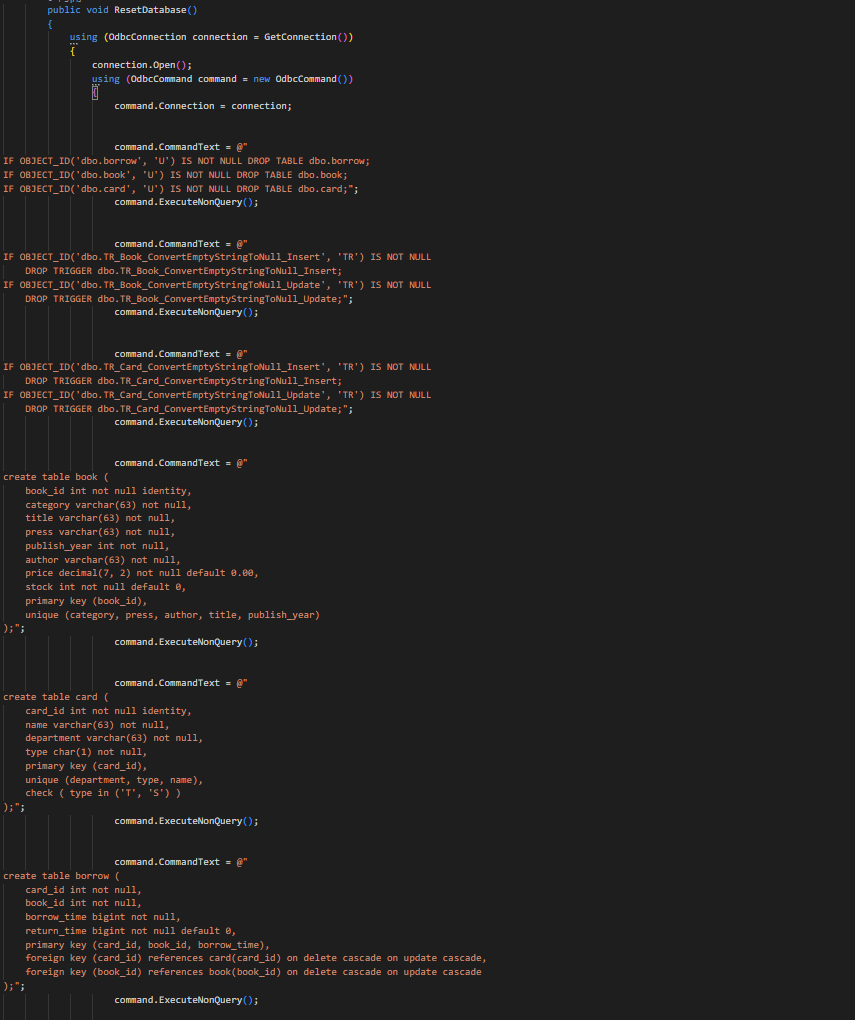
首先，定义了一个 SQL DELETE 语句，利用 card\_id 唯一标识来删除对应的卡片记录。通过 GetConnection() 获取数据库连接，并使用 OdbcCommand 创建 SQL 命令对象。然后，使用 command.Parameters.AddWithValue("?", cardId) 将传入的 cardId 参数与 SQL 语句中的占位符进行绑定。接下来，调用 connection.Open() 打开数据库连接，并通过 command.ExecuteNonQuery() 执行该 SQL 语句，删除指定卡片信息。

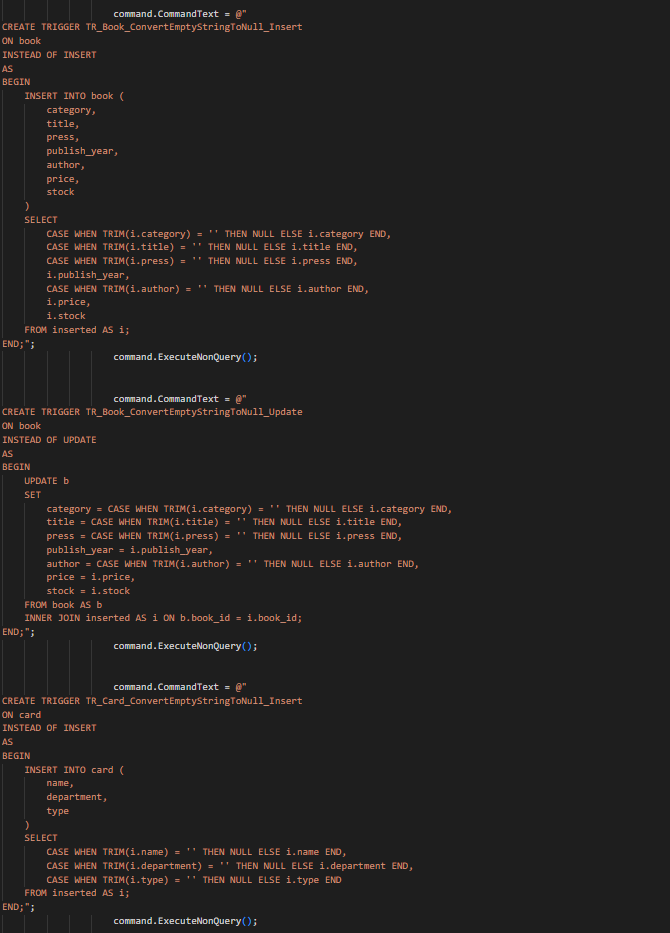
**前端代码：**

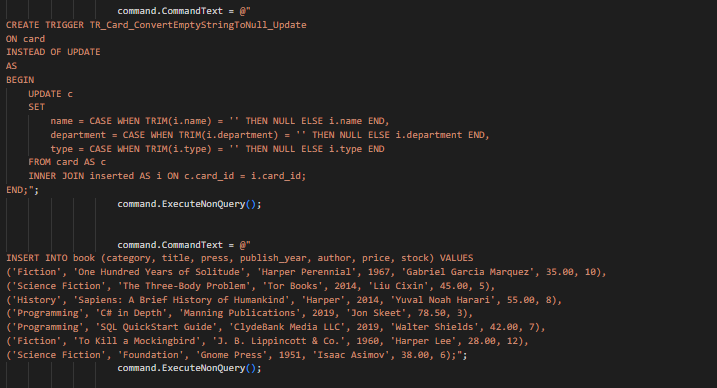
验证用户输入的卡片 ID 是否有效（非空且为整数），并通过 \_cardRepository.GetCardById(cardId) 查询数据库中是否存在对应卡片。若卡片存在，则调用 \_cardRepository.DeleteCard(cardId) 删除该卡片，并通过 RefreshCardList() 更新界面；若卡片不存在或输入无效，系统将给出相应的错误提示。

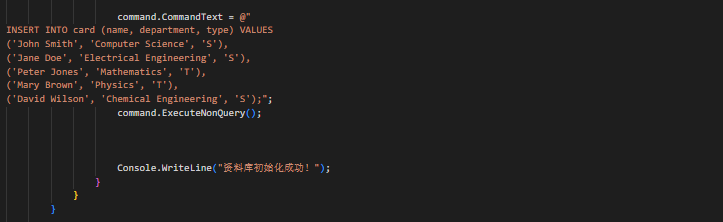
1. 重置数据库

主要代码：





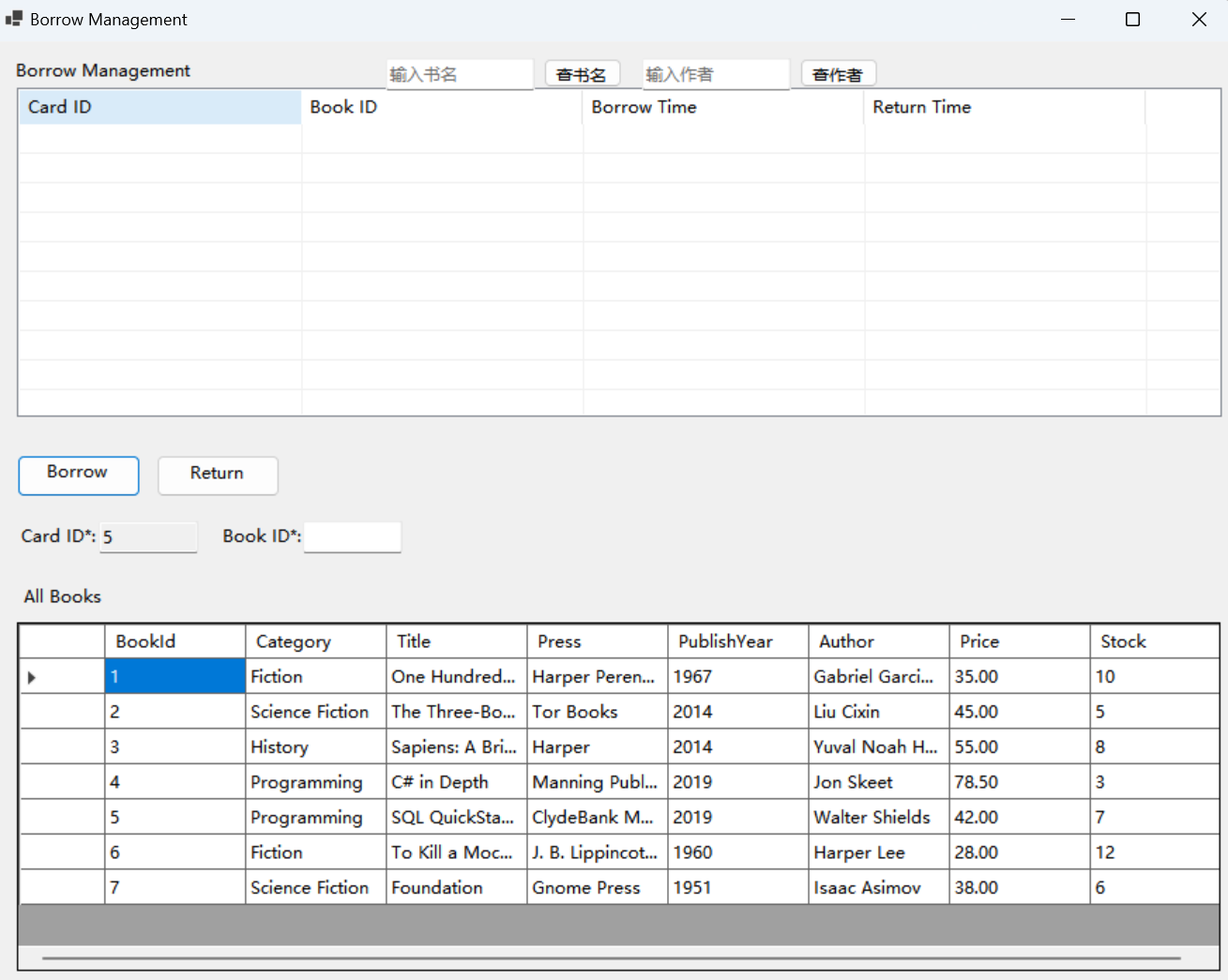




通过 ODBC 连接执行 SQL 脚本，首先删除现有的 borrow、book、card 表及与之关联的触发器。随后，重新创建这些表并定义相应的约束，如主键、外键、唯一性约束及类型检查（例如卡片类型限制为 'T' 或 'S'）。接着，定义四个触发器，用于确保在 book 和 card 表插入或更新时，空字符串自动转换为 NULL。最后，向 book 和 card 表插入初始数据，完成数据库结构的重建与数据初始化。

**到此关于管理员有关的操作模块介绍完毕（管理员和用户共有的模块只介绍一次）。**

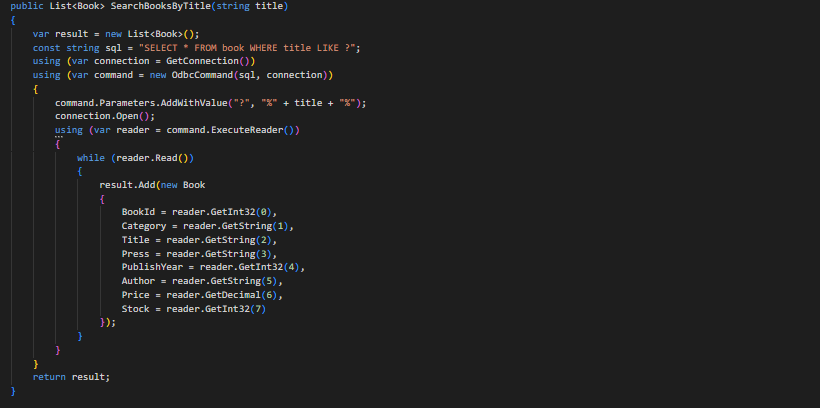
而用户登陆后，会显示新的窗体，包含Borrow Management 和图书查询功能：



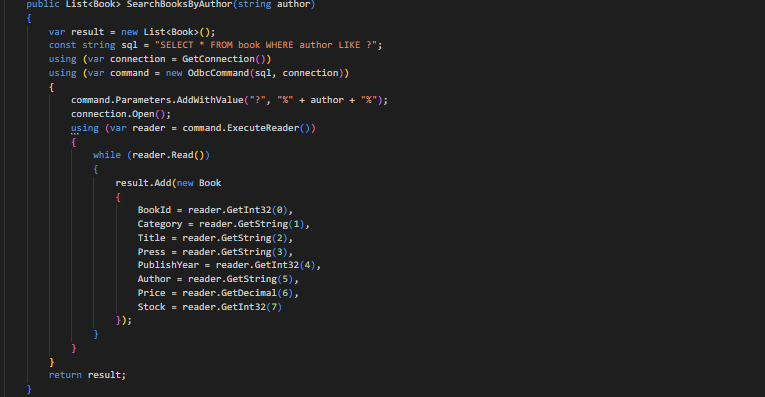
1. 查询模块

按书的名称、作者进行查询。

**主要逻辑代码**：

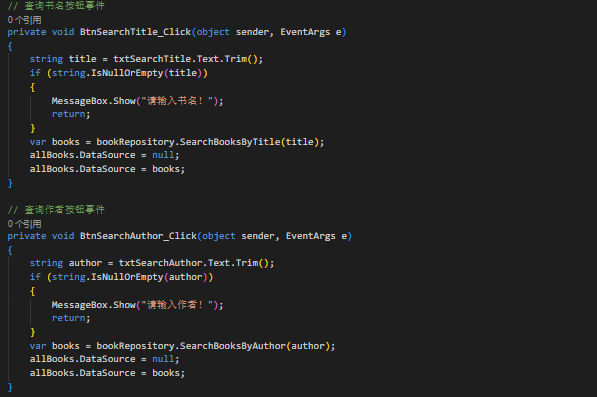


代码实现了根据书籍标题模糊查询书籍的功能。它通过 SQL 查询语句使用 LIKE 操作符，，匹配标题包含指定字符串的书籍。首先，接受一个 title 参数，将其与 % 符号拼接，生成用于模糊匹配的查询条件。然后，使用 OdbcConnection 和 OdbcCommand 执行查询，查询结果通过 OdbcDataReader 逐行读取，每一行的数据都被封装到一个 Book 对象中，并加入到 List<Book> 结果集合中。最终，返回所有匹配标题的书籍列表。



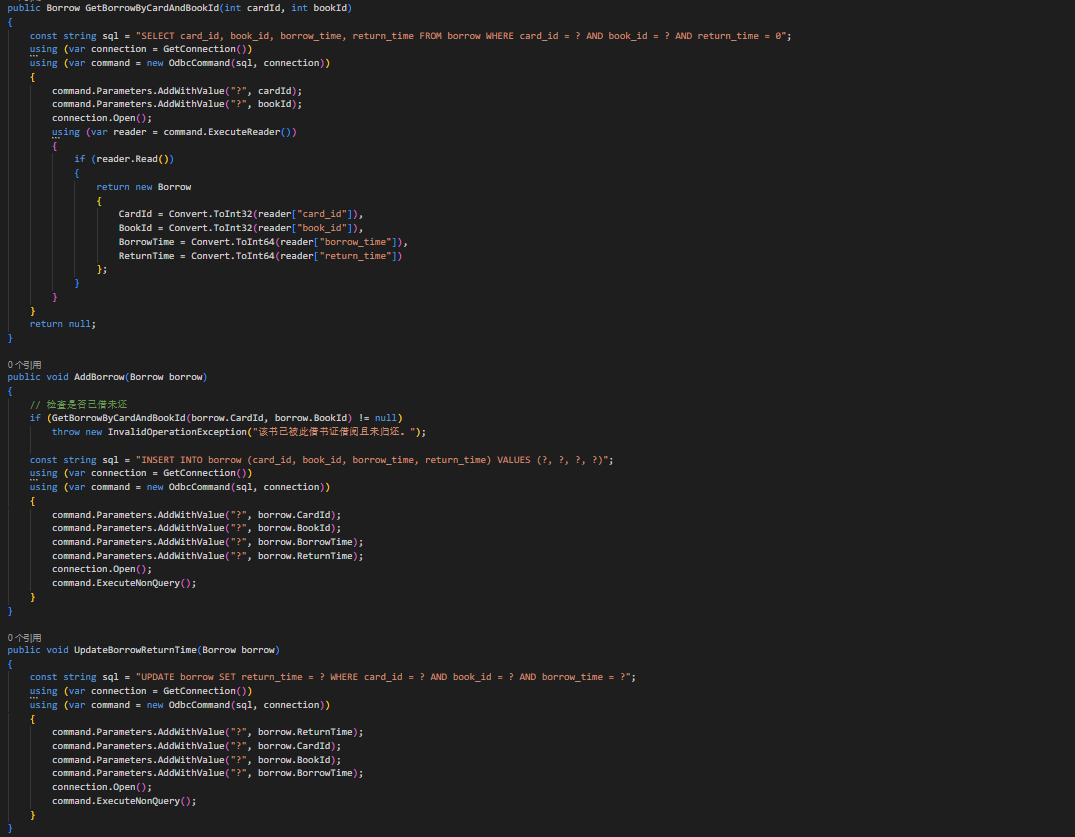
代码实现了根据作者名模糊查询书籍的功能。它通过 SQL 查询语句使用 LIKE 操作符来匹配作者名称中包含指定字符串的书籍。首先，接受一个 author 参数，将其与 % 符号拼接，以便进行模糊匹配。然后，使用 OdbcConnection 和 OdbcCommand 执行查询，查询结果通过 OdbcDataReader 逐行读取。每一行的数据被封装为一个 Book 对象，并添加到 List<Book> 结果集合中。最后，返回所有符合条件的书籍列表。

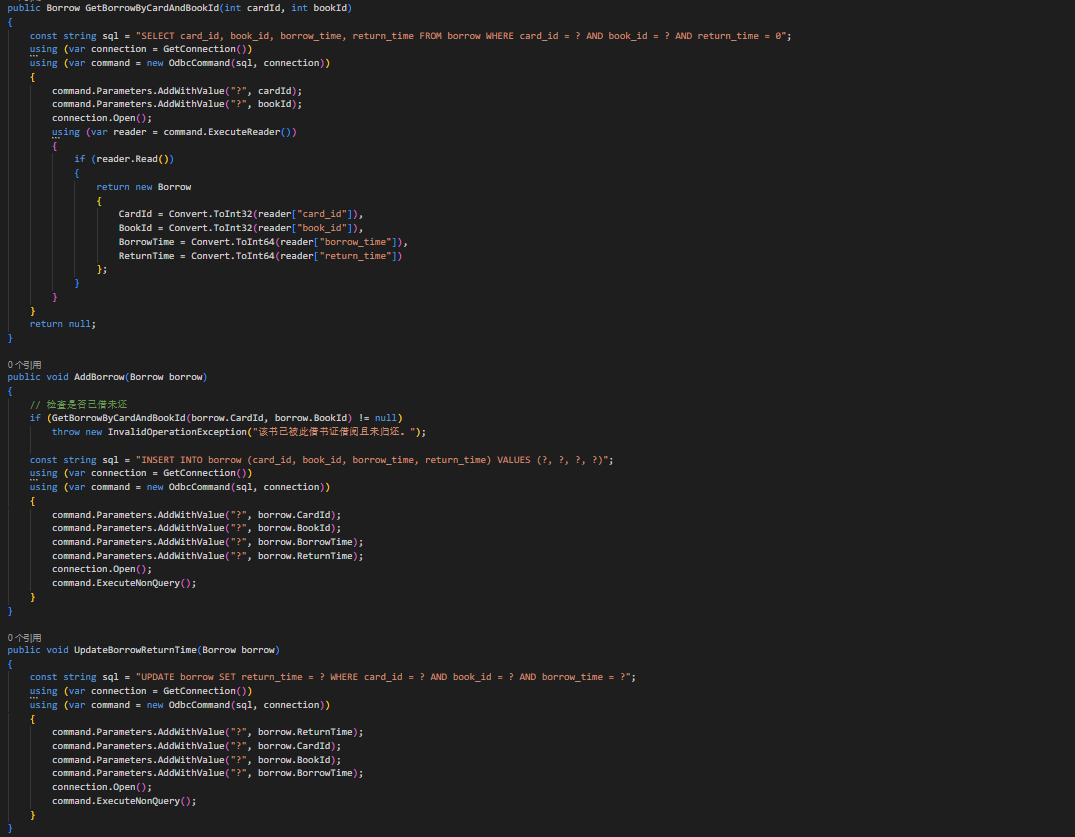
**前端代码：**



代码实现了书籍查询功能的两个按钮事件处理。BtnSearchTitle\_Click 方法根据用户输入的书名进行搜索。首先，获取用户输入的书名并去除多余的空格，如果输入为空，则弹出提示框要求用户输入书名；若输入有效，则调用 bookRepository.SearchBooksByTitle 方法获取匹配的书籍，并将结果绑定到 allBooks 控件上。BtnSearchAuthor\_Click 方法与之类似，但用于根据作者名查询书籍。它获取用户输入的作者名，若为空则提示用户输入作者，输入有效时调用 bookRepository.SearchBooksByAuthor 进行搜索，并更新书籍列表。两者均通过清空 allBooks 的 DataSource 来重新绑定查询结果。

1. **借还书模块**



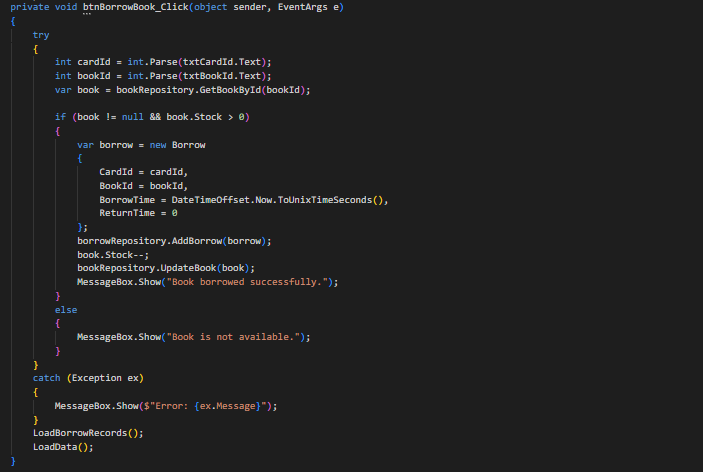


这段代码包含了三个与借书记录（Borrow）相关的方法，分别用于获取、添加和更新借书记录。

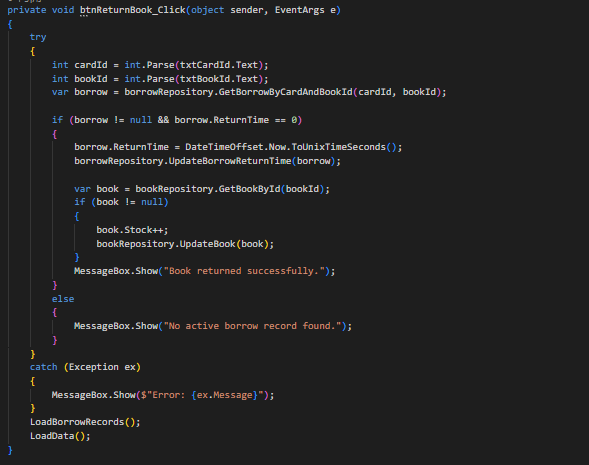
GetBorrowByCardAndBookId：根据借书证 ID (cardId) 和书籍 ID (bookId) 查询数据库中的借书记录，且设定该书籍尚未归还（return\_time = 0）。若找到符合条件的记录，返回相应的 Borrow 对象，否则返回 null。

AddBorrow：新增一条借书记录。首先，它会检查该书籍是否已经被当前借书证借阅且未归还（调用 GetBorrowByCardAndBookId 方法）。如果已借未还，则抛出 InvalidOperationException 异常。否则，它将执行插入操作，将借书信息添加到数据库中的 borrow 表。

UpdateBorrowReturnTime：更新借书记录的归还时间。根据借书证 ID、书籍 ID 和借书时间（borrow\_time）定位记录，更新其 return\_time 字段为新的归还时间。

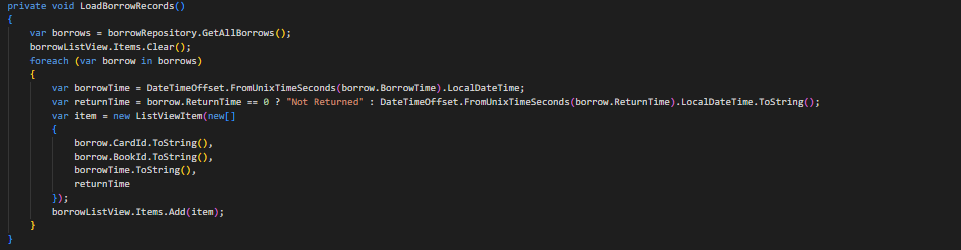
**前端代码：**

代码实现了借书操作的按钮点击事件处理。首先，它从文本框中获取借书证 ID 和书籍 ID，接着通过 bookRepository.GetBookById 获取书籍信息。若书籍存在且库存大于零，则创建一个新的 Borrow 对象，记录借书证 ID、书籍 ID、借书时间（当前时间）以及归还时间（默认为 0，表示未归还）。然后，通过 borrowRepository.AddBorrow 方法将借书记录添加到数据库，并减少该书籍的库存量，更新书籍信息。若借书成功，弹出提示框通知用户。如果书籍不存在或库存为零，则提示书籍不可借。在操作完成后，LoadBorrowRecords() 和 LoadData() 方法被调用，分别用于加载借书记录和更新数据。



代码实现了书籍归还操作的按钮点击事件处理。首先，从文本框中获取借书证 ID 和书籍 ID，接着通过 borrowRepository.GetBorrowByCardAndBookId 方法查找相应的借书记录。若找到了借书记录且该书籍尚未归还（即 ReturnTime == 0），则将当前时间设置为归还时间（通过 DateTimeOffset.Now.ToUnixTimeSeconds() 获取当前 Unix 时间戳），并调用 borrowRepository.UpdateBorrowReturnTime 更新数据库中的归还时间。

接下来，获取该书籍的信息，增加书籍库存，并通过 bookRepository.UpdateBook 更新库存。成功归还后，弹出提示框通知用户。如果没有找到有效的借书记录（即没有未归还的书籍记录），则弹出提示框通知用户。操作结束后，调用 LoadBorrowRecords() 和 LoadData() 方法来重新加载借书记录和更新界面数据。



代码实现了加载所有借书记录并显示在界面上的功能。首先，通过 borrowRepository.GetAllBorrows() 获取所有借书记录。然后清空 borrowListView 控件的现有项目，准备显示新的记录。接着，循环遍历所有借书记录，对于每条记录，它将借书时间 (borrow.BorrowTime) 转换为本地时间，并判断是否已经归还。如果未归还（ReturnTime == 0），则显示为 "Not Returned"；否则，将归还时间 (borrow.ReturnTime) 转换为本地时间并格式化为字符串。

1. 实验心得

本次实验我实现了一个基础的图书管理系统，加深了对数据库与应用软件开发流程和技术栈的理解。整个项目不仅涉及到了基础sql语句的增删改查操作，还涉及到了数据库完整性约束、异常处理以及用户界面的交互设计。通过实现如借书、还书、书籍查询等功能，我学会了如何更好地与数据库进行交互，使用参数化查询确保操作的安全性等。整体上，这个项目锻炼了我的项目开发能力，教会我实际软件设计时要注重用户体验和系统稳定性，同时也加深了我对数据库软件设计的理解。