INTRODUCTION

You have a library of books that contain your favorite stories: *Fahrenheit 451*, *Brave New World*, *The Lord of the Rings*, *Pride and Prejudice*. Well, a virtual library, at least. The books are stored inside of a database on your computer. You'd like to be able to add or remove books from your library and keep things organized. Heck, you'd like to even *see* the books in your library. But how do you get them out of the database? C# has a way for interacting with a database called DAO.

WHAT IS DAO?

DAO stands for Data Access Object. It is a software design pattern that specifies how an application written in C# can interact with a database.

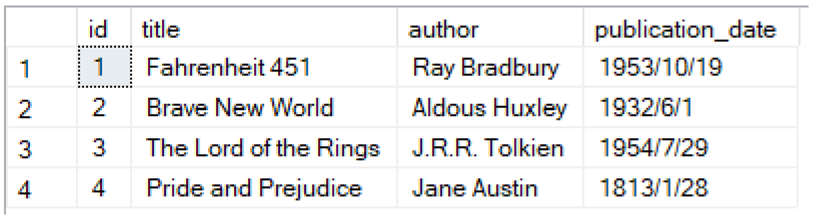
The DAO pattern consists of three components: the domain object, which is represented by a class in C#; the corresponding interface for that domain object; and the implementation of the interface.

Using DAO is important because it promotes encapsulation by storing all logic needed to communicate with a database inside of the classes. It also promotes loosely coupled coding because your application can use different objects in order to access different data types from a database. And it promotes single responsibility, meaning that each class has dominion over a single entity, usually a database table.

Well, that may sound great (or confusing) but how can the DAO pattern be achieved through code? We will be using SQL Server for the database and C# in Visual Studio to work with it. Let's check it out!

THE DATABASE

In our Library database, we have a table of books which contains values for title, author, and publication date.

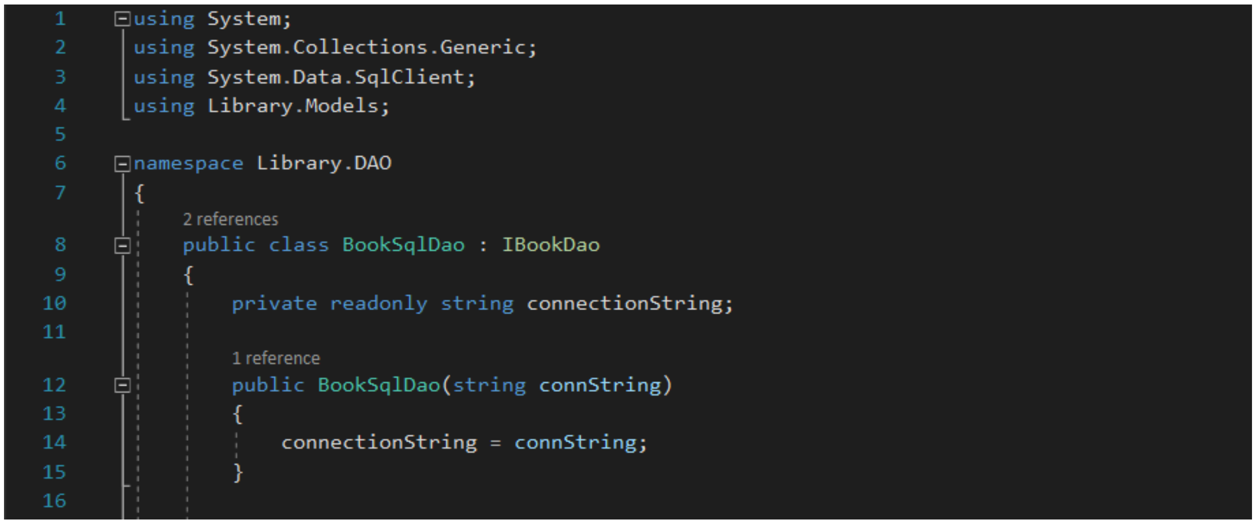


	id	title	author	publication_date
1	1	Fahrenheit 451	Ray Bradbury	1953/10/19
2	2	Brave New World	Aldous Huxley	1932/6/1
3	3	The Lord of the Rings	J.R.R. Tolkien	1954/7/29
4	4	Pride and Prejudice	Jane Austin	1813/1/28

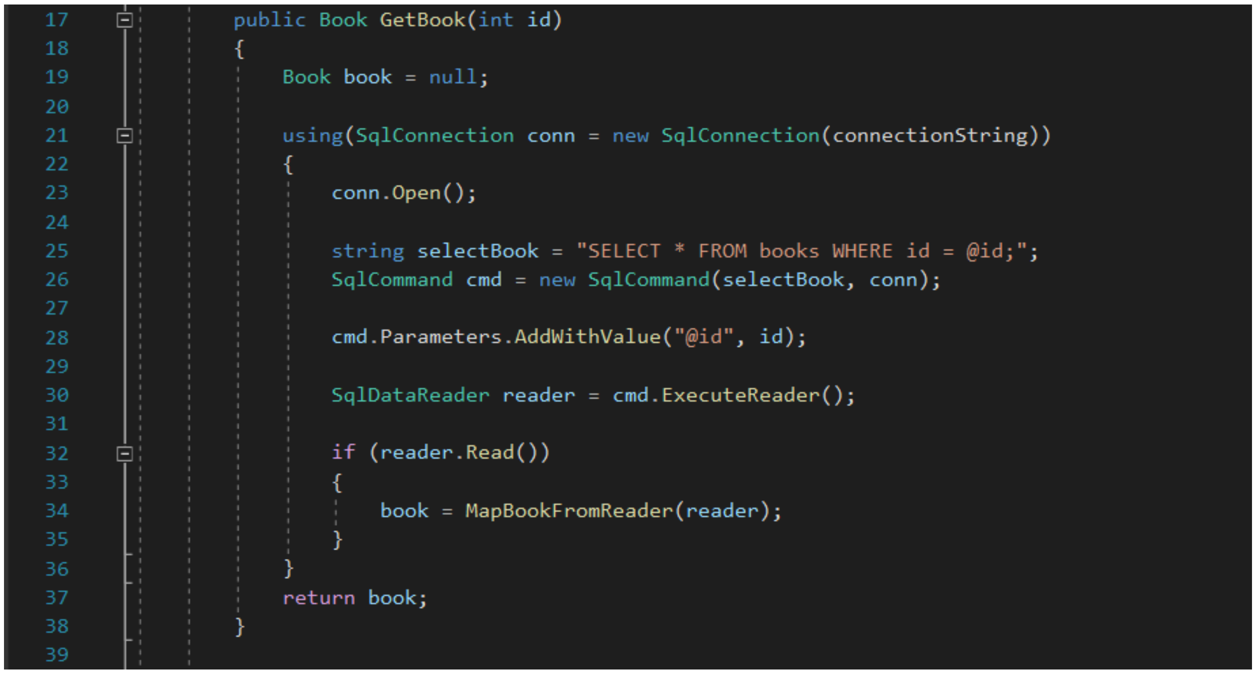
THE OBJECTIVE

Now that we can see our lovely library, let's write some code that can interact with the database. We will write methods to get a list of books in the library or a specific book, and also ways to modify a book, add a new book to the library, or remove a book from the library.

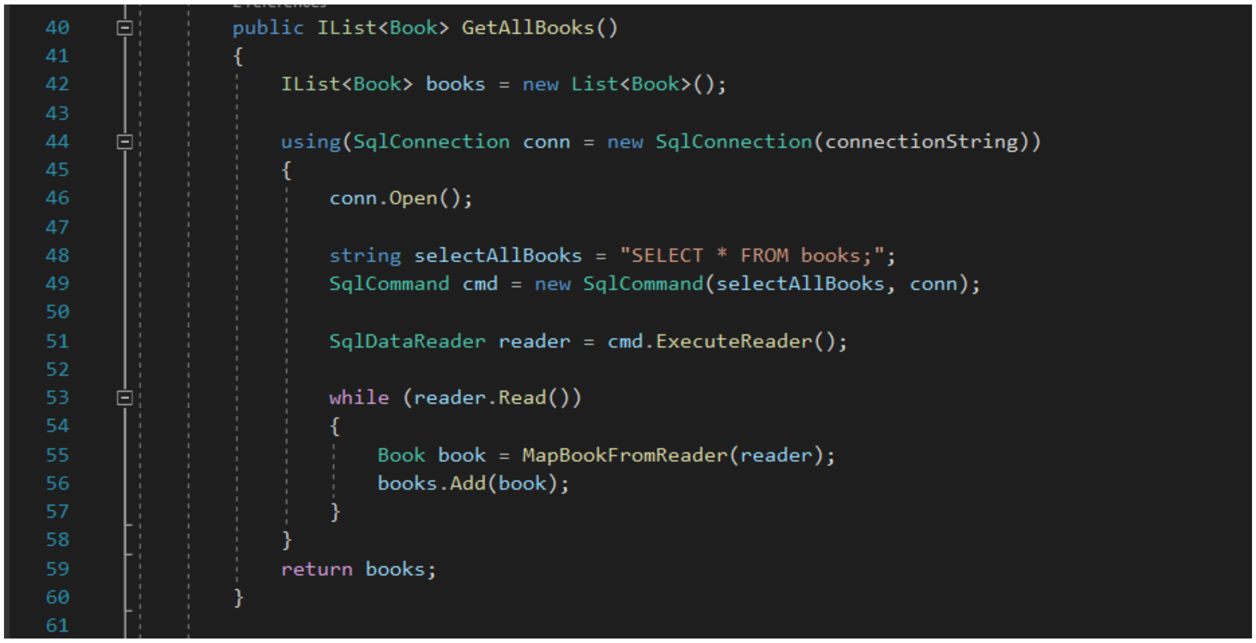
<u>OVERVIEW</u>
Before we begin, here is an overview of the BookSqlDao implementation we will be using to interact with our Library database.



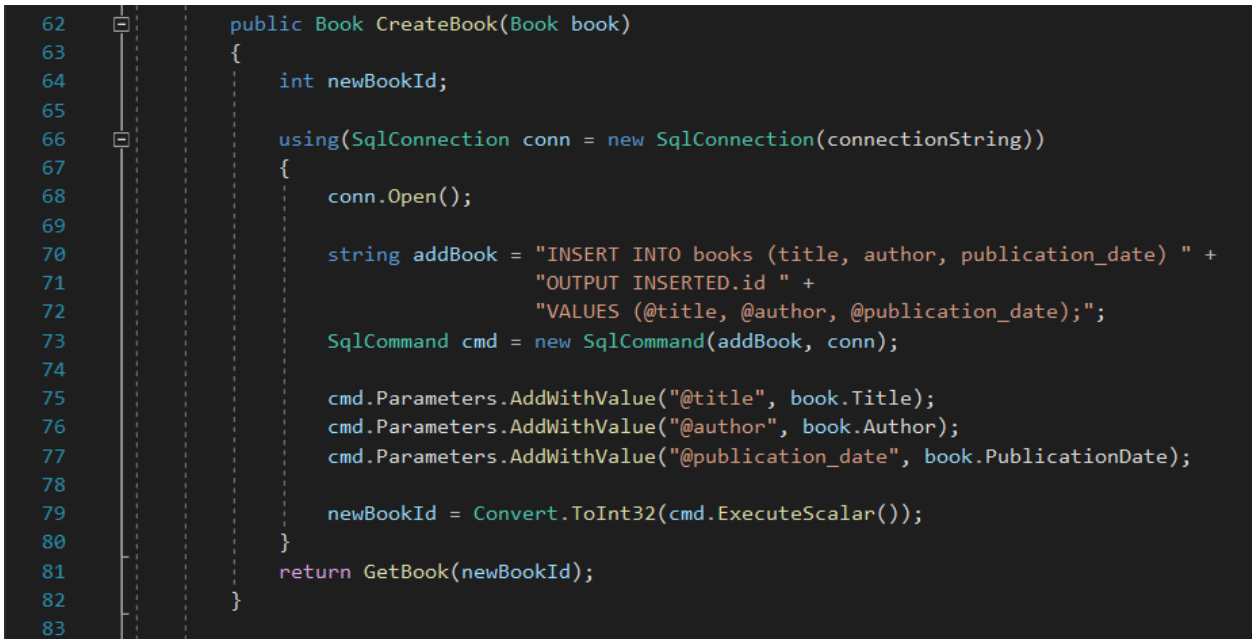
```csharp
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using Library.Models;

namespace Library.DAO
{
    2 references
    public class BookSqlDao : IBookDao
    {
        private readonly string connectionString;

        1 reference
        public BookSqlDao(string connString)
        {
            connectionString = connString;
        }

```



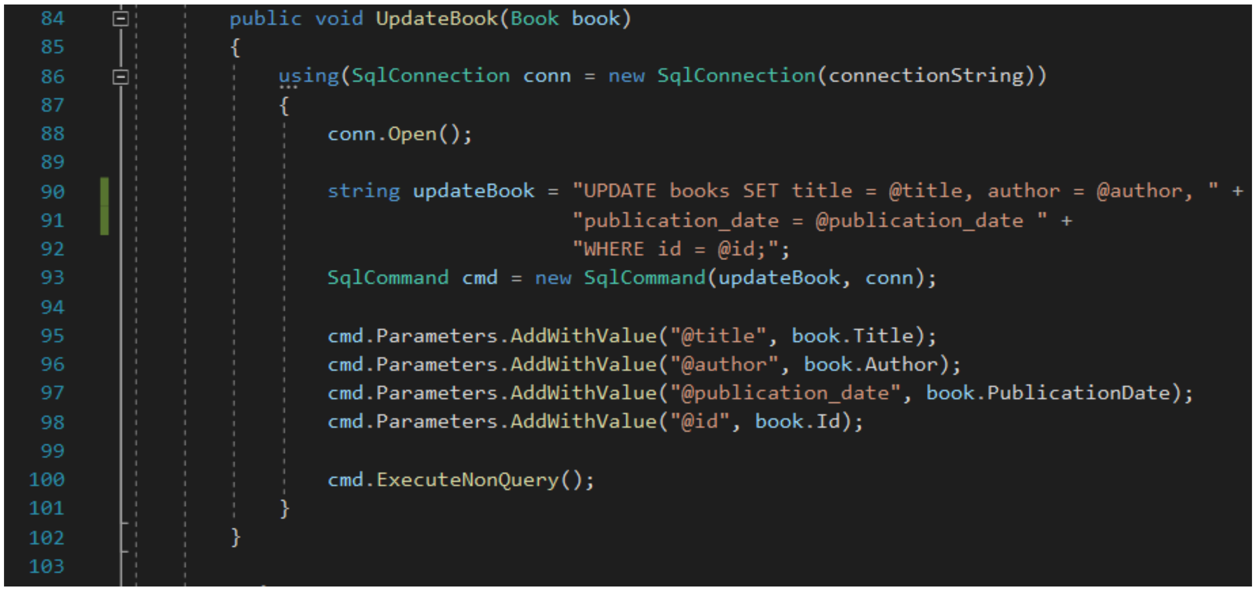
```csharp
        public Book GetBook(int id)
        {
            Book book = null;

            using(SqlConnection conn = new SqlConnection(connectionString))
            {
                conn.Open();

                string selectBook = "SELECT * FROM books WHERE id = @id;";
                SqlCommand cmd = new SqlCommand(selectBook, conn);

                cmd.Parameters.AddWithValue("@id", id);

                SqlDataReader reader = cmd.ExecuteReader();

                if (reader.Read())
                {
                    book = MapBookFromReader(reader);
                }
            }
            return book;
        }

```



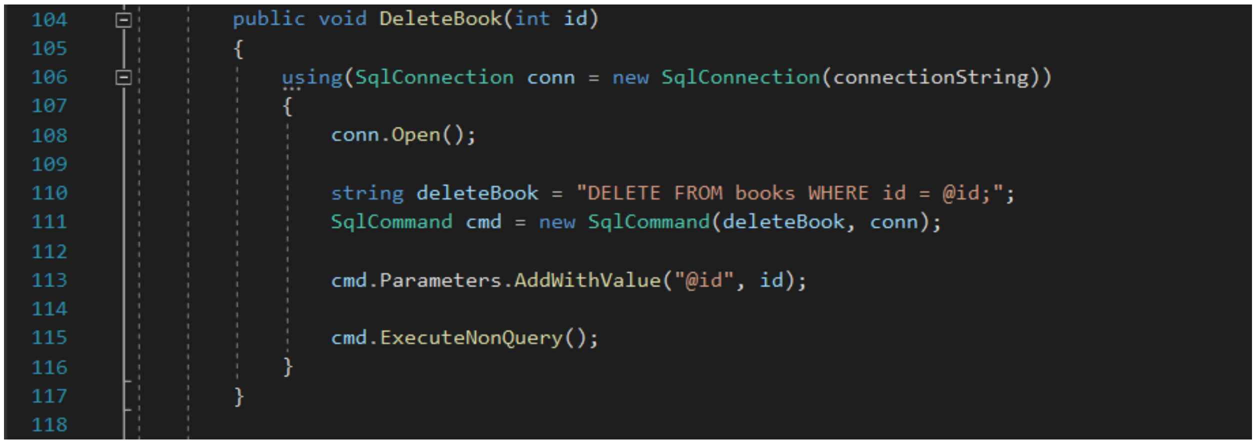
```
        public IList<Book> GetAllBooks()
        {
            IList<Book> books = new List<Book>();

            using(SqlConnection conn = new SqlConnection(connectionString))
            {
                conn.Open();

                string selectAllBooks = "SELECT * FROM books;";
                SqlCommand cmd = new SqlCommand(selectAllBooks, conn);

                SqlDataReader reader = cmd.ExecuteReader();

                while (reader.Read())
                {
                    Book book = MapBookFromReader(reader);
                    books.Add(book);
                }
            }
            return books;
        }

```



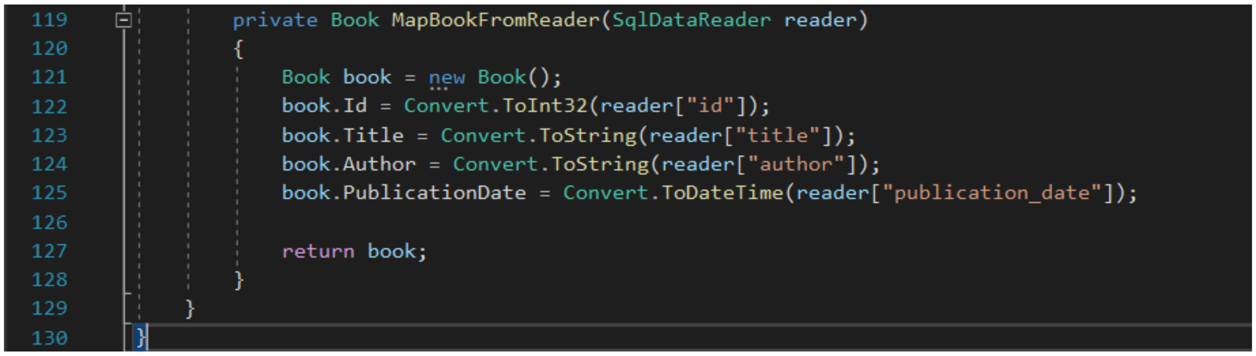
```
        public Book CreateBook(Book book)
        {
            int newBookId;

            using(SqlConnection conn = new SqlConnection(connectionString))
            {
                conn.Open();

                string addBook = "INSERT INTO books (title, author, publication_date) " +
                                 "OUTPUT INSERTED.id " +
                                 "VALUES (@title, @author, @publication_date);";
                SqlCommand cmd = new SqlCommand(addBook, conn);

                cmd.Parameters.AddWithValue("@title", book.Title);
                cmd.Parameters.AddWithValue("@author", book.Author);
                cmd.Parameters.AddWithValue("@publication_date", book.PublicationDate);

                newBookId = Convert.ToInt32(cmd.ExecuteScalar());
            }
            return GetBook(newBookId);
        }

```



```
        public void UpdateBook(Book book)
        {
            using(SqlConnection conn = new SqlConnection(connectionString))
            {
                conn.Open();

                string updateBook = "UPDATE books SET title = @title, author = @author, " +
                                    "publication_date = @publication_date " +
                                    "WHERE id = @id;";
                SqlCommand cmd = new SqlCommand(updateBook, conn);

                cmd.Parameters.AddWithValue("@title", book.Title);
                cmd.Parameters.AddWithValue("@author", book.Author);
                cmd.Parameters.AddWithValue("@publication_date", book.PublicationDate);
                cmd.Parameters.AddWithValue("@id", book.Id);

                cmd.ExecuteNonQuery();
            }
        }

```



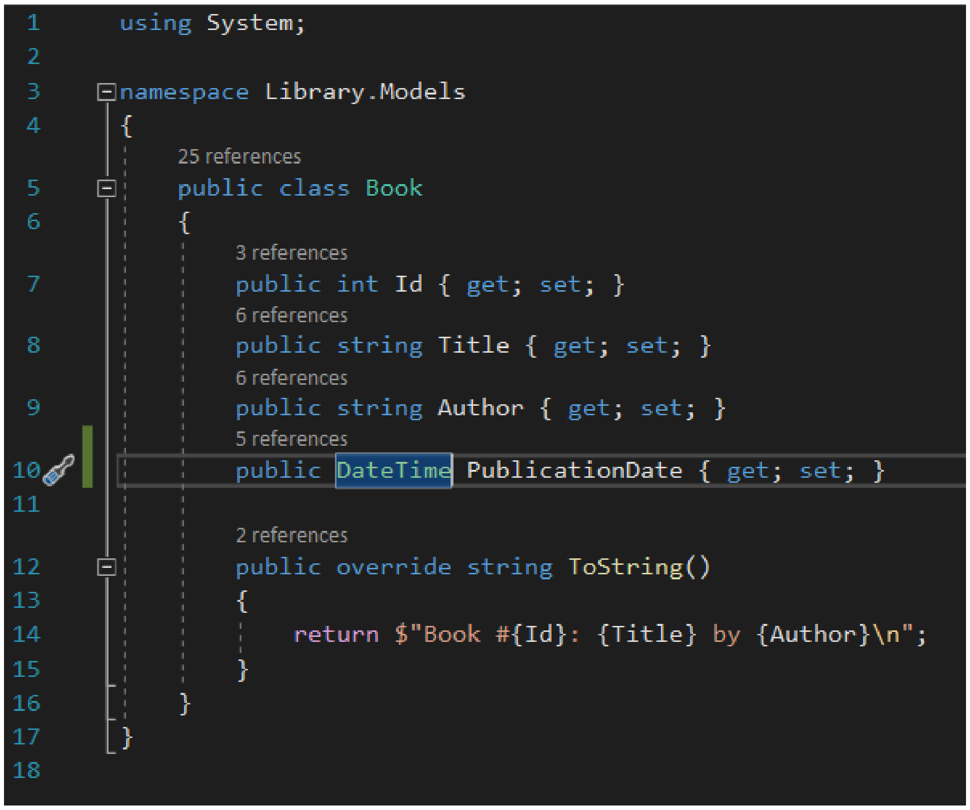
```
        public void DeleteBook(int id)
        {
            using(SqlConnection conn = new SqlConnection(connectionString))
            {
                conn.Open();

                string deleteBook = "DELETE FROM books WHERE id = @id;";
                SqlCommand cmd = new SqlCommand(deleteBook, conn);

                cmd.Parameters.AddWithValue("@id", id);

                cmd.ExecuteNonQuery();
            }
        }

```



```
        private Book MapBookFromReader(SqlDataReader reader)
        {
            Book book = new Book();
            book.Id = Convert.ToInt32(reader["id"]);
            book.Title = Convert.ToString(reader["title"]);
            book.Author = Convert.ToString(reader["author"]);
            book.PublicationDate = Convert.ToDateTime(reader["publication_date"]);

            return book;
        }
    }
}
```

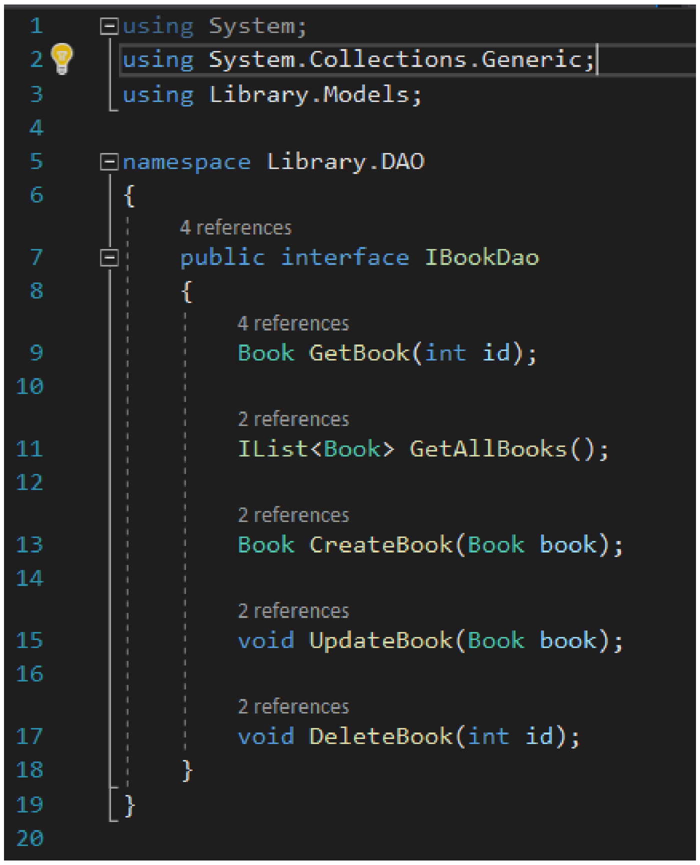
<u>COMPONENTS OF THE DAO</u>

As stated previously, the three components of the DAO pattern are the domain object, the interface, and the implementation of the interface. In this example, our domain object will be a book class. It could look something like this:



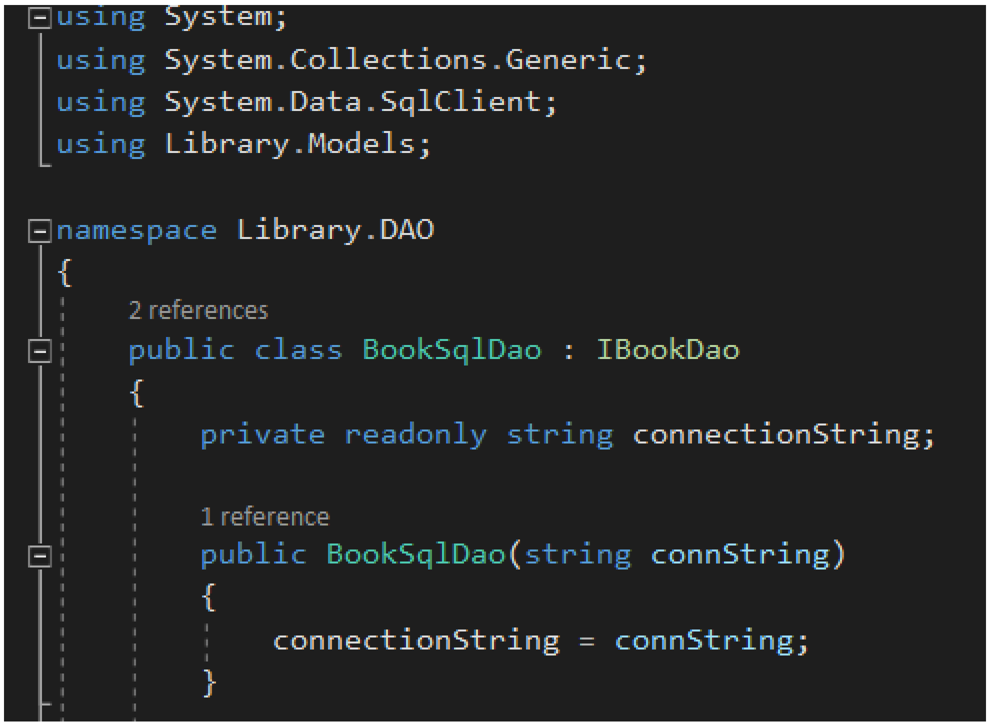
```
using System;

namespace Library.Models
{
    25 references
    public class Book
    {
        3 references
        public int Id { get; set; }
        6 references
        public string Title { get; set; }
        6 references
        public string Author { get; set; }
        5 references
        public DateTime PublicationDate { get; set; }

        2 references
        public override string ToString()
        {
            return $"Book #{Id}: {Title} by {Author}\n";
        }
    }
}

```

The interface would list the methods used to communicate with the Library database, and our implementation would define how those methods worked. An example of the interface might be:



```
using System;
using System.Collections.Generic;
using Library.Models;

namespace Library.DAO
{
    4 references
    public interface IBookDao
    {
        4 references
        Book GetBook(int id);

        2 references
        IList<Book> GetAllBooks();

        2 references
        Book CreateBook(Book book);

        2 references
        void UpdateBook(Book book);

        2 references
        void DeleteBook(int id);
    }
}
```

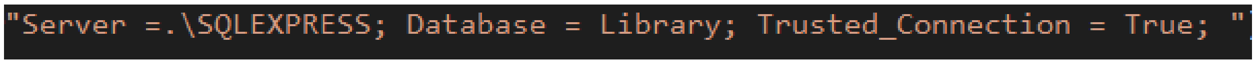
Implementing our interface could look like the following:



```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using Library.Models;

namespace Library.DAO
{
    2 references
    public class BookSqlDao : IBookDao
    {
        private readonly string connectionString;

        1 reference
        public BookSqlDao(string connString)
        {
            connectionString = connString;
        }
```

CONNECTING TO A DATABASE

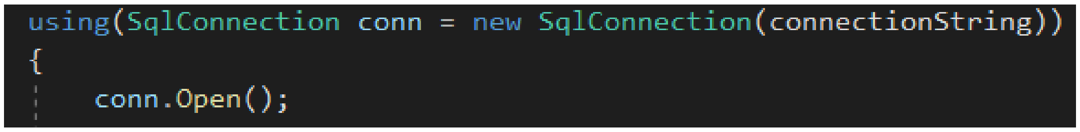
In order to work with a database, we need to connect to it. This is done through the connection string. The connection string is the hostname or IP address of the server, the name of the database to connect to, the username, and the user password. Because we don't want to put any sensitive information in our application (such as a password), we can use the string "Trusted_Connection=true" in lieu of the username and password. Our connection string would look something like this:



```
"Server =.\SQLEXPRESS; Database = Library; Trusted_Connection = True; "
```

The classes used to connect to a database and manipulate data can be found in the namespace System.Data.SqlClient.

In order to connect to the database, we use the SqlConnection class and pass in the connection string as a parameter when instantiating a SqlConnection object. The connection must then be explicitly opened using the Open() method of the SqlConnection object. SqlConnection implements the IDisposable interface, meaning that we can use the object in a using block which will automatically terminate our connection to the database when we no longer require a connection from our method.

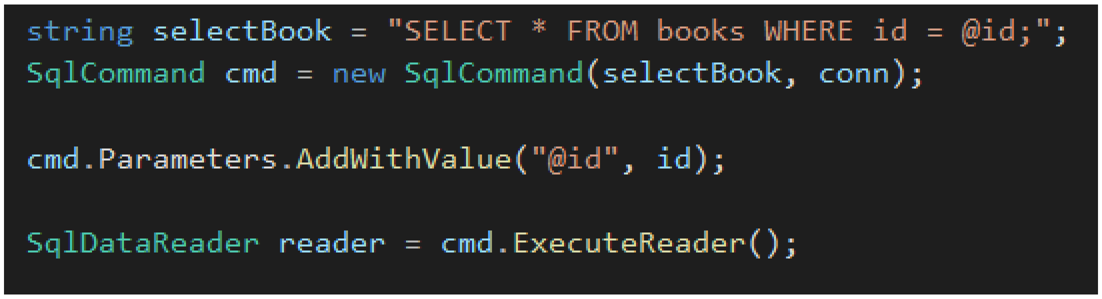


```
using(SqlConnection conn = new SqlConnection(connectionString))
{
    conn.Open();
```

INTERACTING WITH THE DATABASE

Now that we have established a connection to the database, we can interact with the data. This can be done via the SqlCommand class. We instantiate a new object and pass in two arguments to SqlCommand: a SQL query statement as a string and the SqlConnection object we had just created. If the statement has any parameters, the parameters must be added using the AddWithValue() method of the SqlCommand object.

Once the query statement and connection object have been stored in the command object, the statement is executed using the ExecuteReader() method of the command object. The return values (the data we pull from the database) must be stored in a variable of data type SqlDataReader. This data reader variable will be able to read the results pulled from the database.

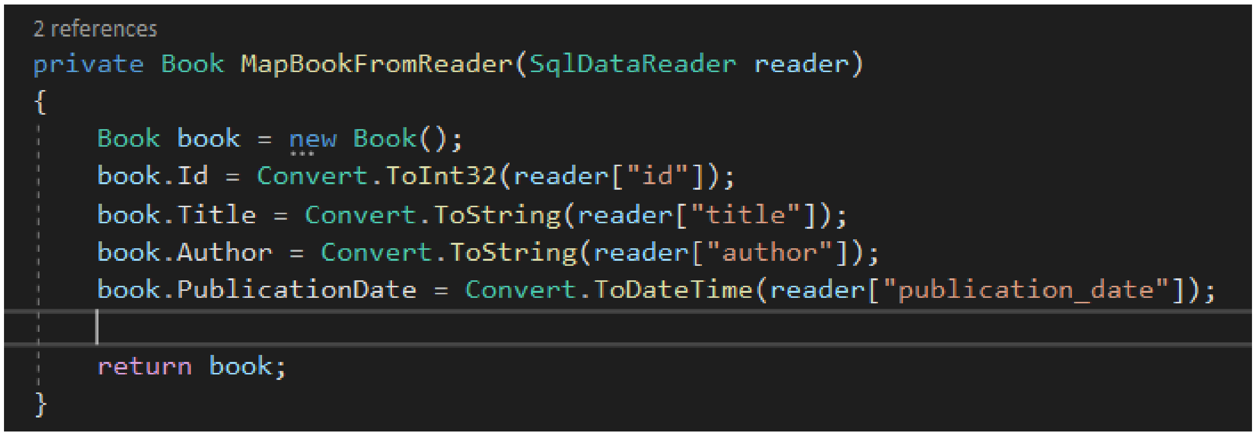


```
string selectBook = "SELECT * FROM books WHERE id = @id;";
SqlCommand cmd = new SqlCommand(selectBook, conn);

cmd.Parameters.AddWithValue("@id", id);

SqlDataReader reader = cmd.ExecuteReader();
```

<ins>MAPPING DATA FROM THE DATABASE</ins>

Now that we have the data from the database, we need to be able to use it. The SqlDataReader class has a Read() method which allows us to read through each row returned from the database. We can also create our own method that maps this database data to an object and pass in the data we got from the database as an argument.

For example, if we wanted to create a book from the book data in our Library database, we could use something like the following:

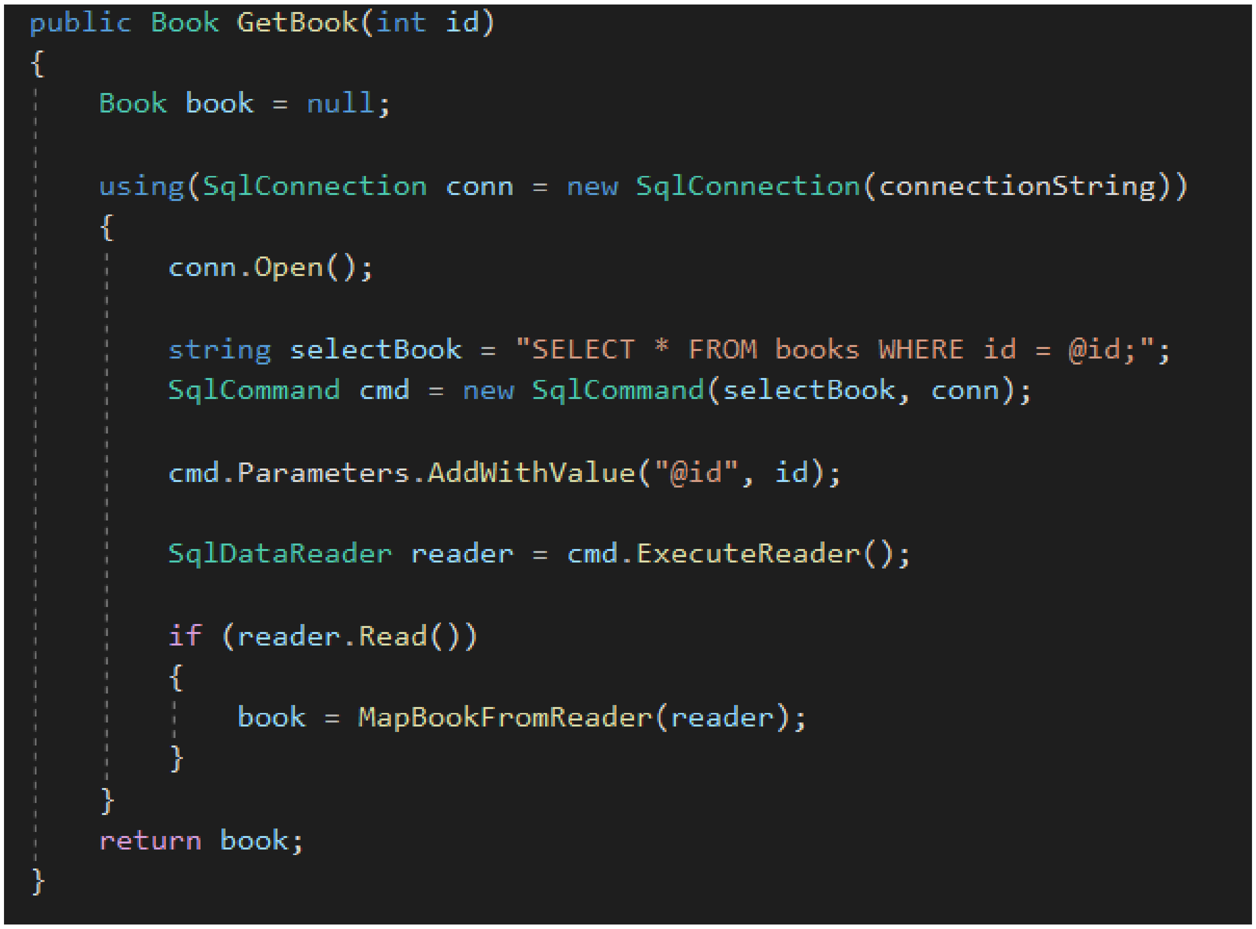


```
2 references
private Book MapBookFromReader(SqlDataReader reader)
{
    Book book = new Book();
    book.Id = Convert.ToInt32(reader["id"]);
    book.Title = Convert.ToString(reader["title"]);
    book.Author = Convert.ToString(reader["author"]);
    book.PublicationDate = Convert.ToDateTime(reader["publication_date"]);

    return book;
}
```

Because the data we pull from the database are not .NET data types, we must convert them using different Convert.To__() methods.

<ins>GETTING A BOOK FROM THE LIBRARY</ins>

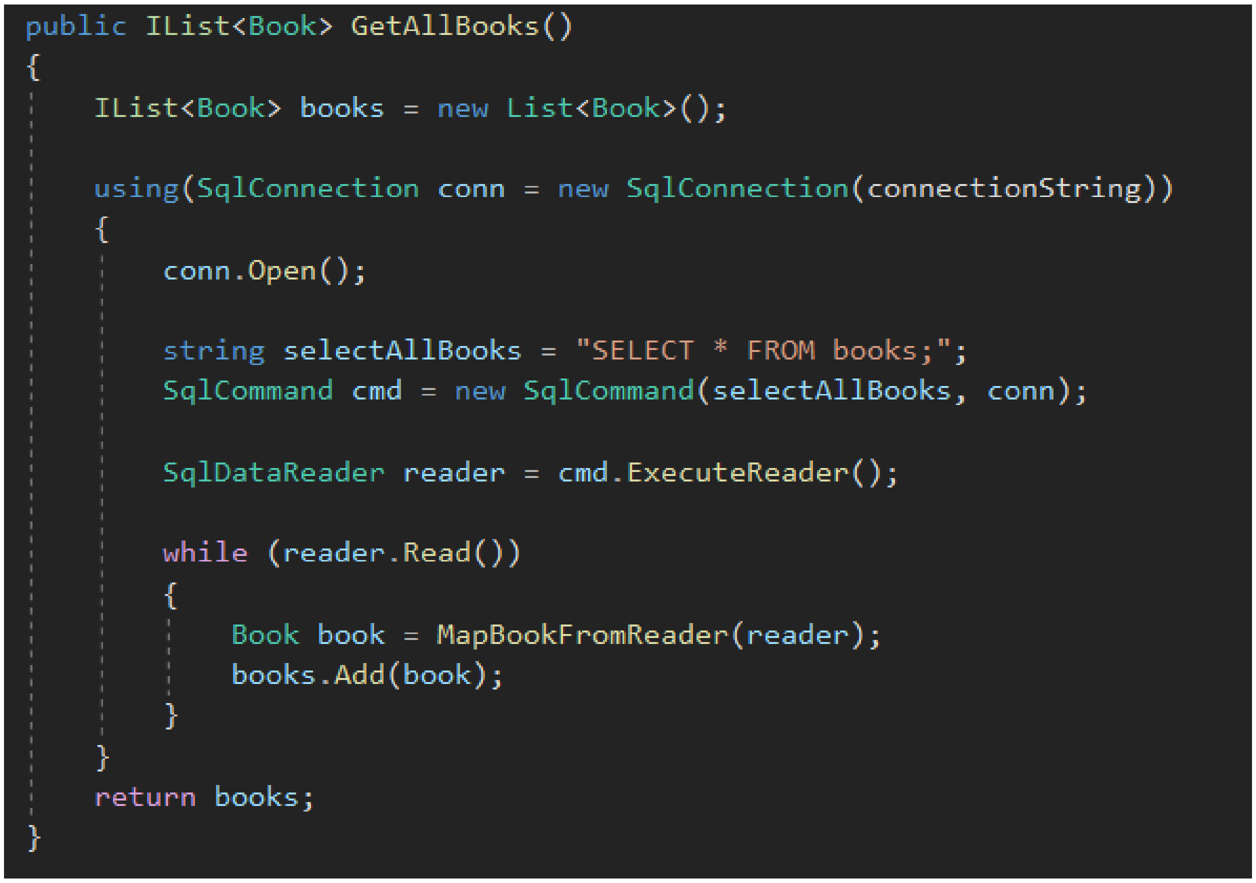
If we wanted to retrieve a book from our library, we could do this:



```
public Book GetBook(int id)
{
    Book book = null;

    using(SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();

        string selectBook = "SELECT * FROM books WHERE id = @id;";
        SqlCommand cmd = new SqlCommand(selectBook, conn);

        cmd.Parameters.AddWithValue("@id", id);

        SqlDataReader reader = cmd.ExecuteReader();

        if (reader.Read())
        {
            book = MapBookFromReader(reader);
        }
    }
    return book;
}
```

- The method to retrieve a specific book from the library takes an id of type integer as an argument.
- We create a new book object. This will be used to store our data pulled from the database later in the method.
- Open a connection to the database.
- Write a SQL query statement to retrieve a specific book from the Library database.
- Create a SqlCommand object which will communicate with the database and send the query statement to the database.
- Store the data pulled from the database in a SqlDataReader variable.
- Using the Read() method of the SqlDataReader variable called reader, use the MapBookFromReader() method to map the database data to our book object that we instantiated earlier in our method.
- Return the book containing the data we retrieved from the database.

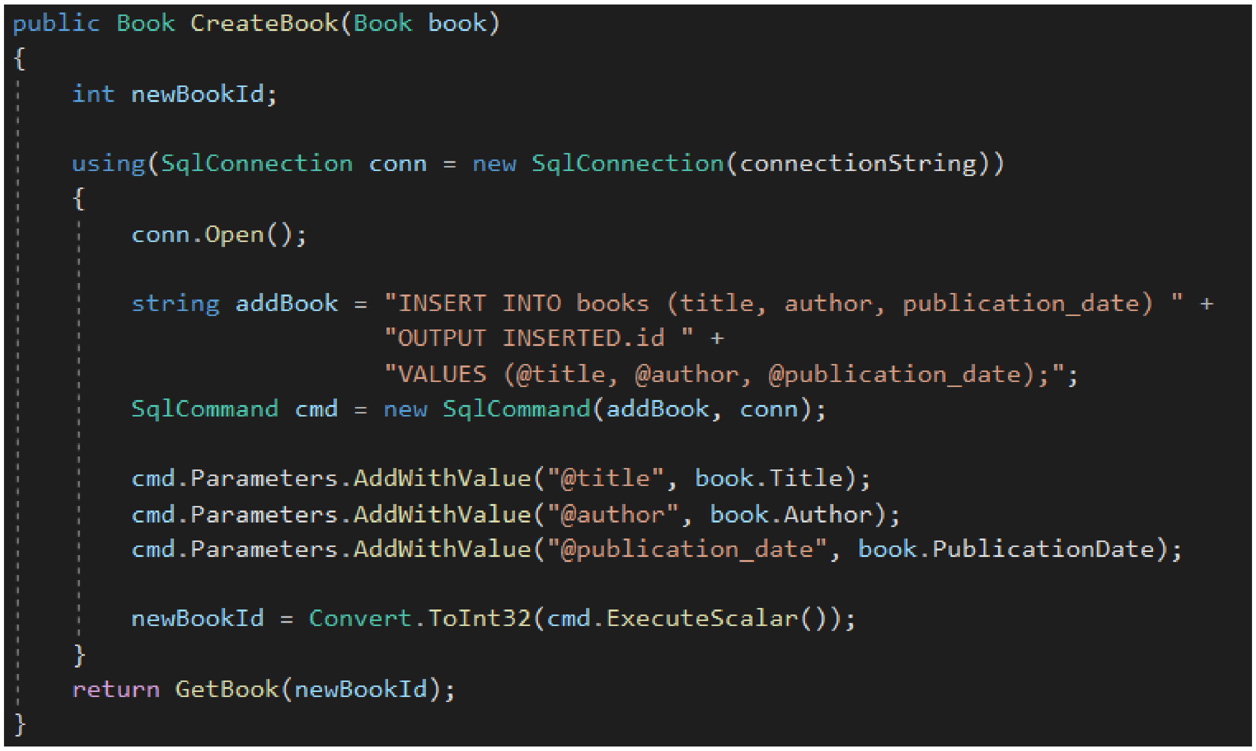
<u>GETTING ALL BOOKS FROM THE LIBRARY</u>
We can also use a method that would give us a list of all books in our library.



```
public IList<Book> GetAllBooks()
{
    IList<Book> books = new List<Book>();

    using(SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();

        string selectAllBooks = "SELECT * FROM books;";
        SqlCommand cmd = new SqlCommand(selectAllBooks, conn);

        SqlDataReader reader = cmd.ExecuteReader();

        while (reader.Read())
        {
            Book book = MapBookFromReader(reader);
            books.Add(book);
        }
    }
    return books;
}
```

<u>ADDING BOOKS, MODIFYING BOOKS, AND REMOVING BOOKS</u>
Different methods can be used to achieve different database operations like adding a book to the library (with the SQL insert operation), modifying a library book (with the SQL update operation), or removing a book (with the SQL delete operation).

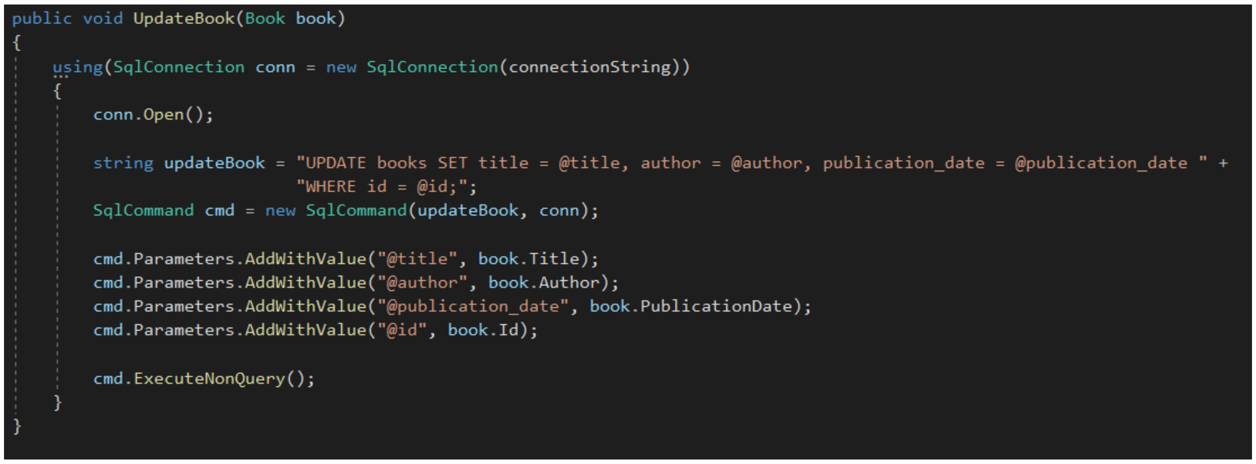
Adding a book:



```csharp
public Book CreateBook(Book book)
{
    int newBookId;

    using(SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();

        string addBook = "INSERT INTO books (title, author, publication_date) " +
                         "OUTPUT INSERTED.id " +
                         "VALUES (@title, @author, @publication_date);";
        SqlCommand cmd = new SqlCommand(addBook, conn);

        cmd.Parameters.AddWithValue("@title", book.Title);
        cmd.Parameters.AddWithValue("@author", book.Author);
        cmd.Parameters.AddWithValue("@publication_date", book.PublicationDate);

        newBookId = Convert.ToInt32(cmd.ExecuteScalar());
    }
    return GetBook(newBookId);
}
```

Note the ExecuteScalar() method on our SqlCommand object called cmd. This method returns a single value from the database, and we can use the book ID to retrieve the book from the Library database which the user just added.

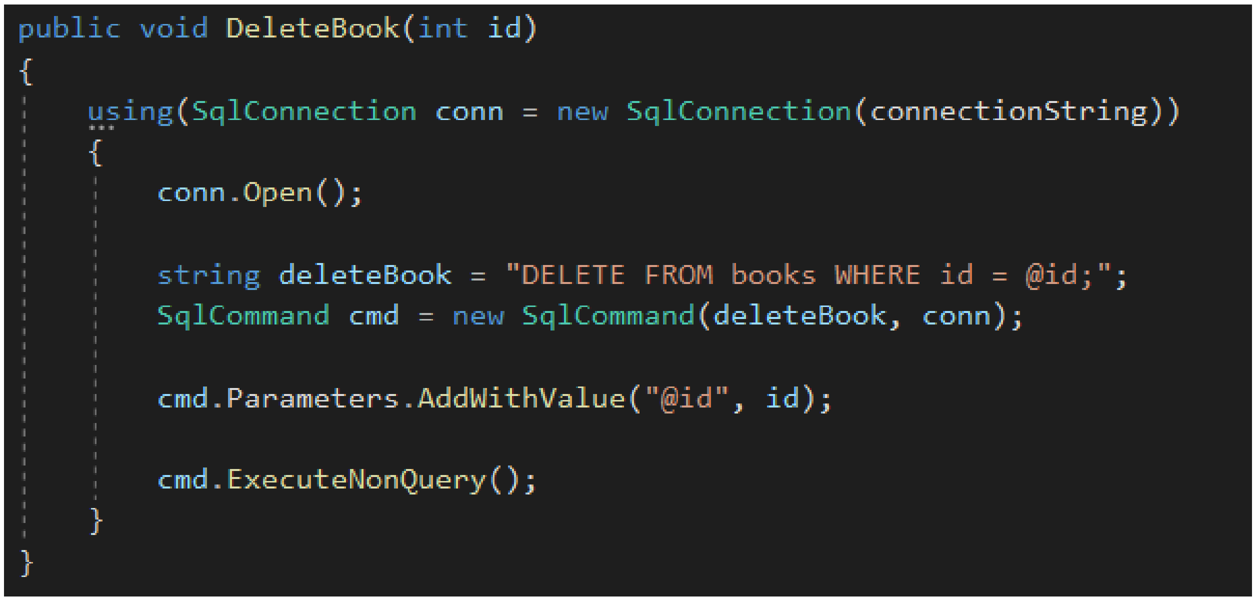
Modifying a book:



```csharp
public void UpdateBook(Book book)
{
    using(SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();

        string updateBook = "UPDATE books SET title = @title, author = @author, publication_date = @publication_date " +
                            "WHERE id = @id;";
        SqlCommand cmd = new SqlCommand(updateBook, conn);

        cmd.Parameters.AddWithValue("@title", book.Title);
        cmd.Parameters.AddWithValue("@author", book.Author);
        cmd.Parameters.AddWithValue("@publication_date", book.PublicationDate);
        cmd.Parameters.AddWithValue("@id", book.Id);

        cmd.ExecuteNonQuery();
    }
}
```

The method ExecuteNonQuery() returns the number of rows affected in the database. This can be used to check and see if the database was updated as expected or not.

Removing a book:



```
public void DeleteBook(int id)
{
    using(SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();

        string deleteBook = "DELETE FROM books WHERE id = @id;";
        SqlCommand cmd = new SqlCommand(deleteBook, conn);

        cmd.Parameters.AddWithValue("@id", id);

        cmd.ExecuteNonQuery();
    }
}
```

This will delete the book data from the Library database.

CONCLUSION

The DAO pattern is a useful way to interact with a database using C# and the .NET framework. It might take some time to get used to all the different parts of the pattern and how they interact, but once you get comfortable you'll be able to write your own applications that interact with the data in your databases.