Lab5

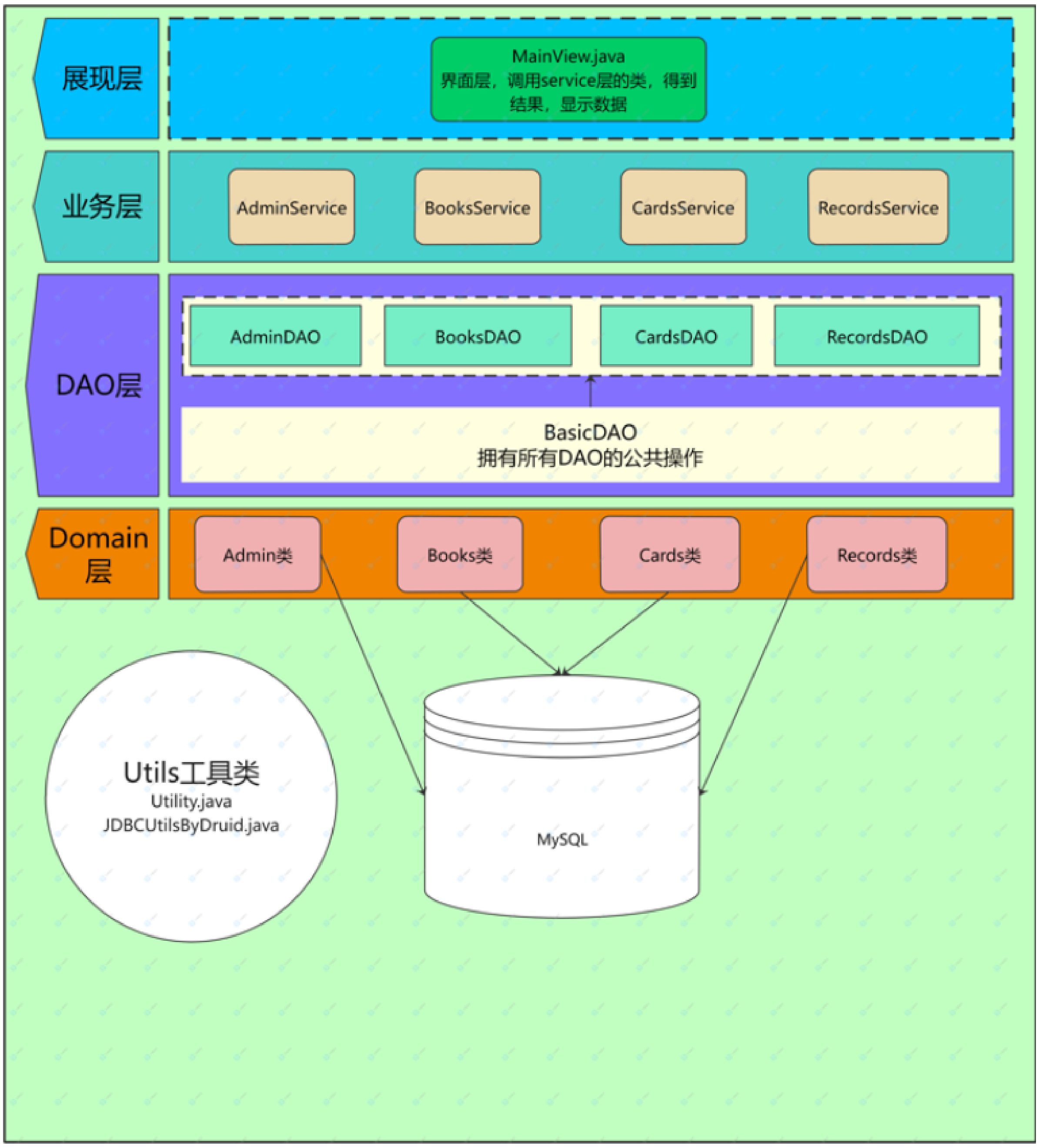
图书管理系统

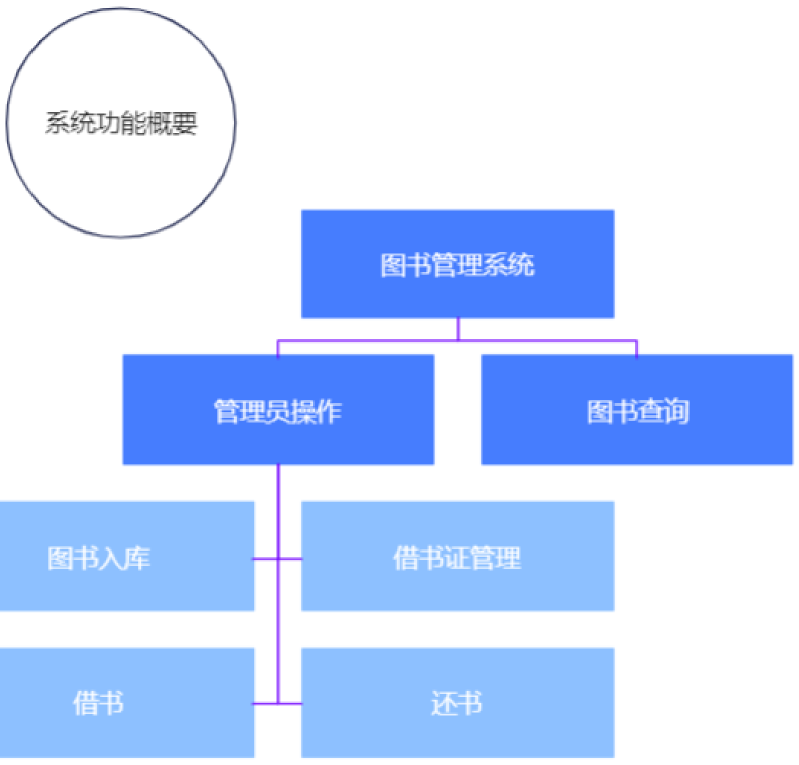
Date:2022-04-19

一、总体设计

1. 系统架构描述

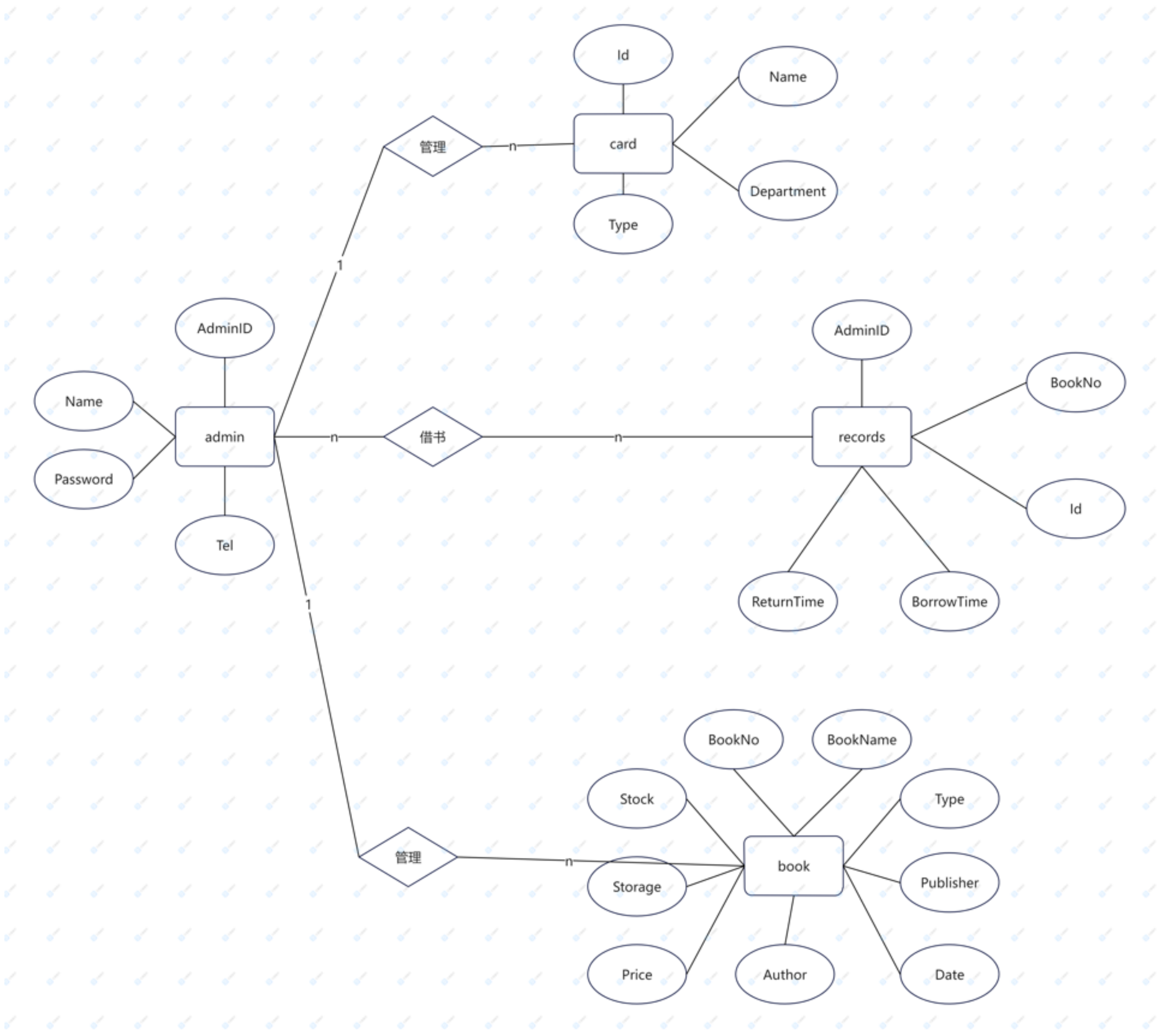
本系统采用java语言，故能有很好的封装性和层次性，对应表的操作放在对应的业务类中，极度方便后续的审查和修改。额外使用了已封装的Apache工具类和Druid连接池。Apache能有效防止注入，且能在连接关闭后保留获取的信息。Druid连接池的性能则能确保系统的速度，从而支持大量用户的快速访问。



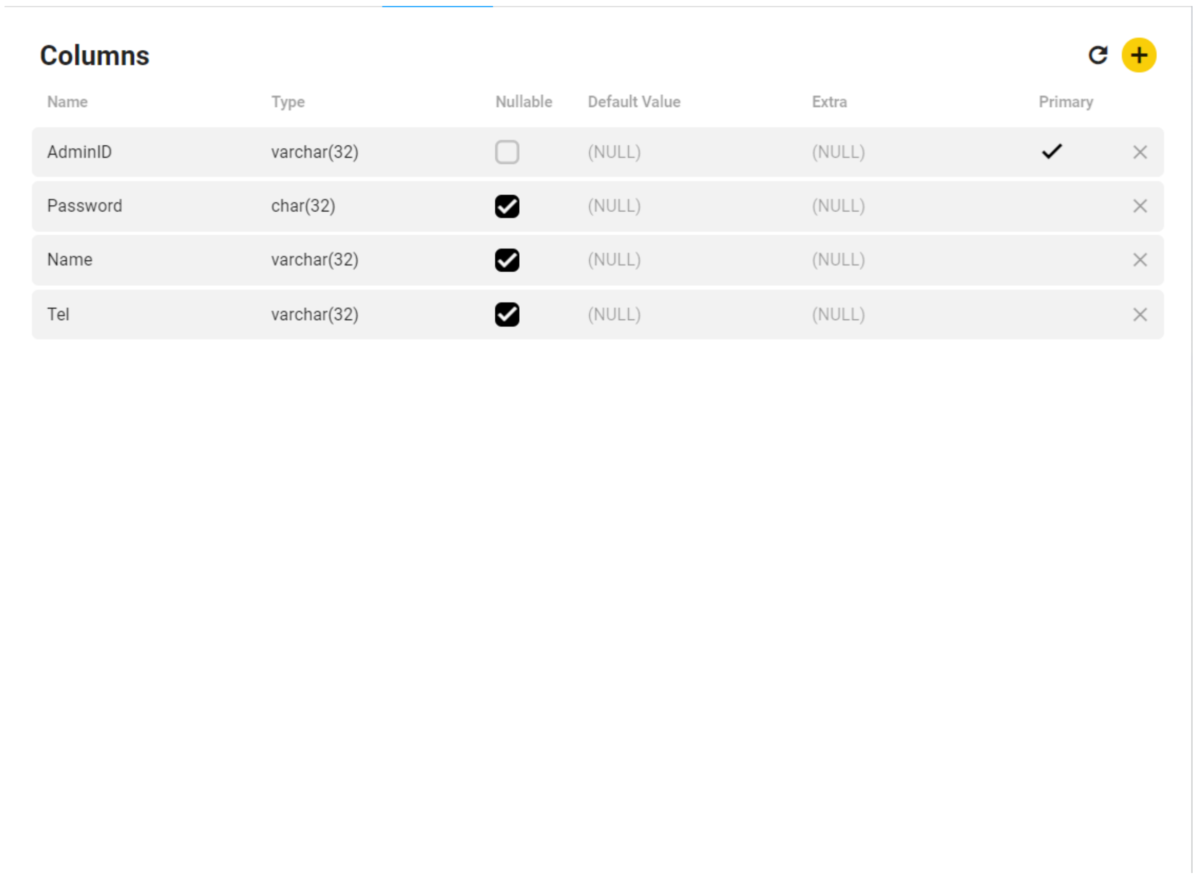


2. 数据库表结构设计

ER模型：



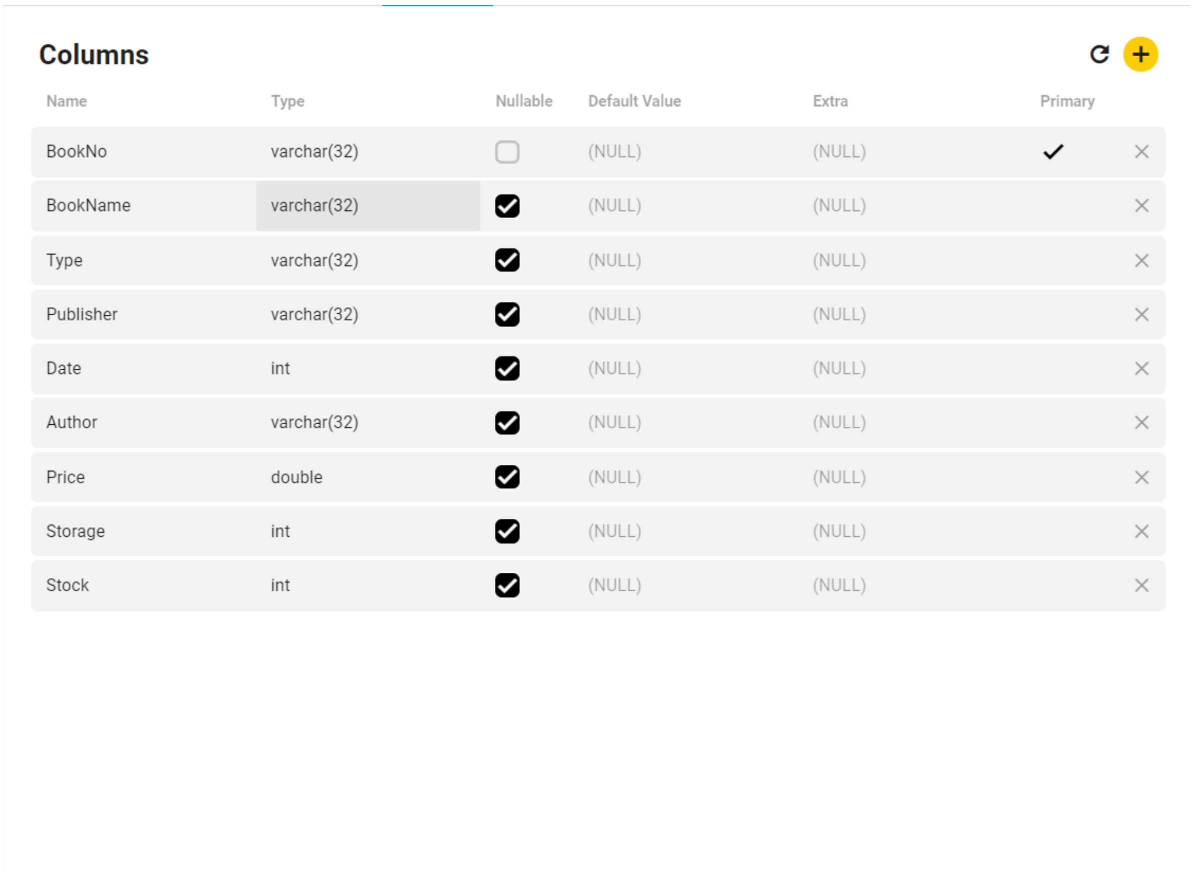
表一：admin



Columns

Name	Type	Nullable	Default Value	Extra	Primary
AdminID	varchar(32)	☐	(NULL)	(NULL)	✓
Password	char(32)	☑	(NULL)	(NULL)	
Name	varchar(32)	☑	(NULL)	(NULL)	
Tel	varchar(32)	☑	(NULL)	(NULL)	

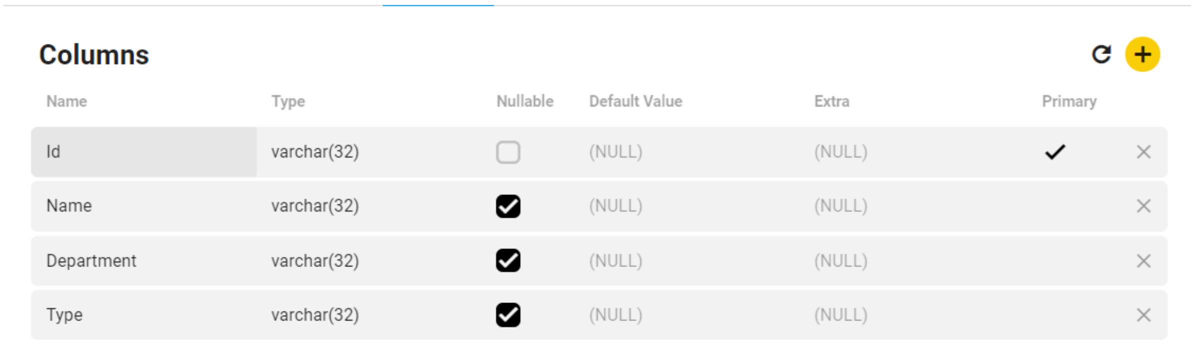
表二：book



Columns

Name	Type	Nullable	Default Value	Extra	Primary
BookNo	varchar(32)	☐	(NULL)	(NULL)	✓
BookName	varchar(32)	☑	(NULL)	(NULL)	
Type	varchar(32)	☑	(NULL)	(NULL)	
Publisher	varchar(32)	☑	(NULL)	(NULL)	
Date	int	☑	(NULL)	(NULL)	
Author	varchar(32)	☑	(NULL)	(NULL)	
Price	double	☑	(NULL)	(NULL)	
Storage	int	☑	(NULL)	(NULL)	
Stock	int	☑	(NULL)	(NULL)	

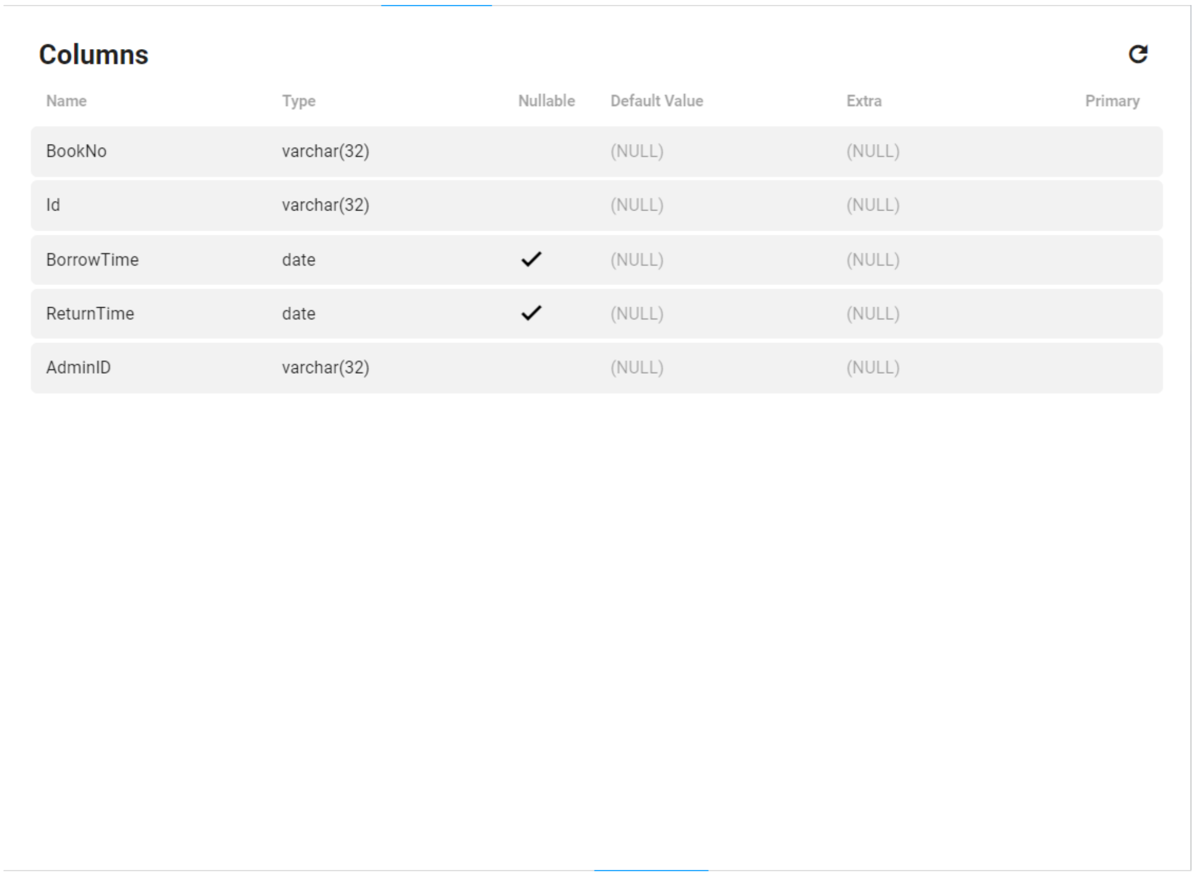
表三：card



Columns

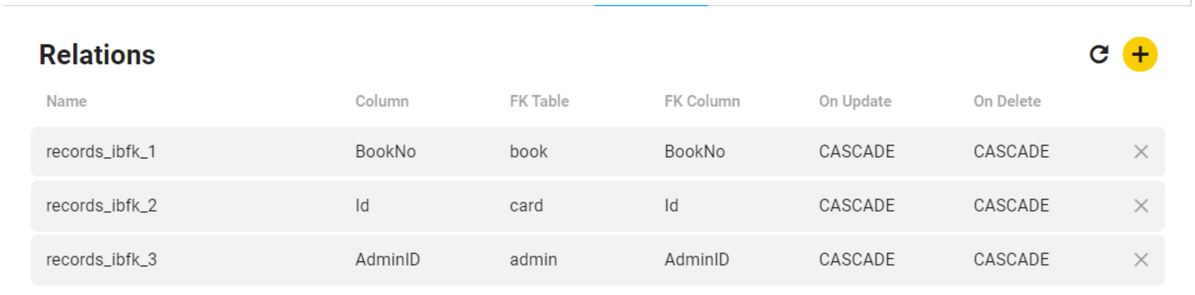
Name	Type	Nullable	Default Value	Extra	Primary
Id	varchar(32)	☐	(NULL)	(NULL)	✓
Name	varchar(32)	☑	(NULL)	(NULL)	
Department	varchar(32)	☑	(NULL)	(NULL)	
Type	varchar(32)	☑	(NULL)	(NULL)	

表四：records，与前三张表的主键级联，保证了数据的一致性



Columns

Name	Type	Nullable	Default Value	Extra	Primary
BookNo	varchar(32)		(NULL)	(NULL)	
Id	varchar(32)		(NULL)	(NULL)	
BorrowTime	date	✓	(NULL)	(NULL)	
ReturnTime	date	✓	(NULL)	(NULL)	
AdminID	varchar(32)		(NULL)	(NULL)	



Relations

Name	Column	FK Table	FK Column	On Update	On Delete
records_ibfk_1	BookNo	book	BookNo	CASCADE	CASCADE
records_ibfk_2	Id	card	Id	CASCADE	CASCADE
records_ibfk_3	AdminID	admin	AdminID	CASCADE	CASCADE

二、详细设计

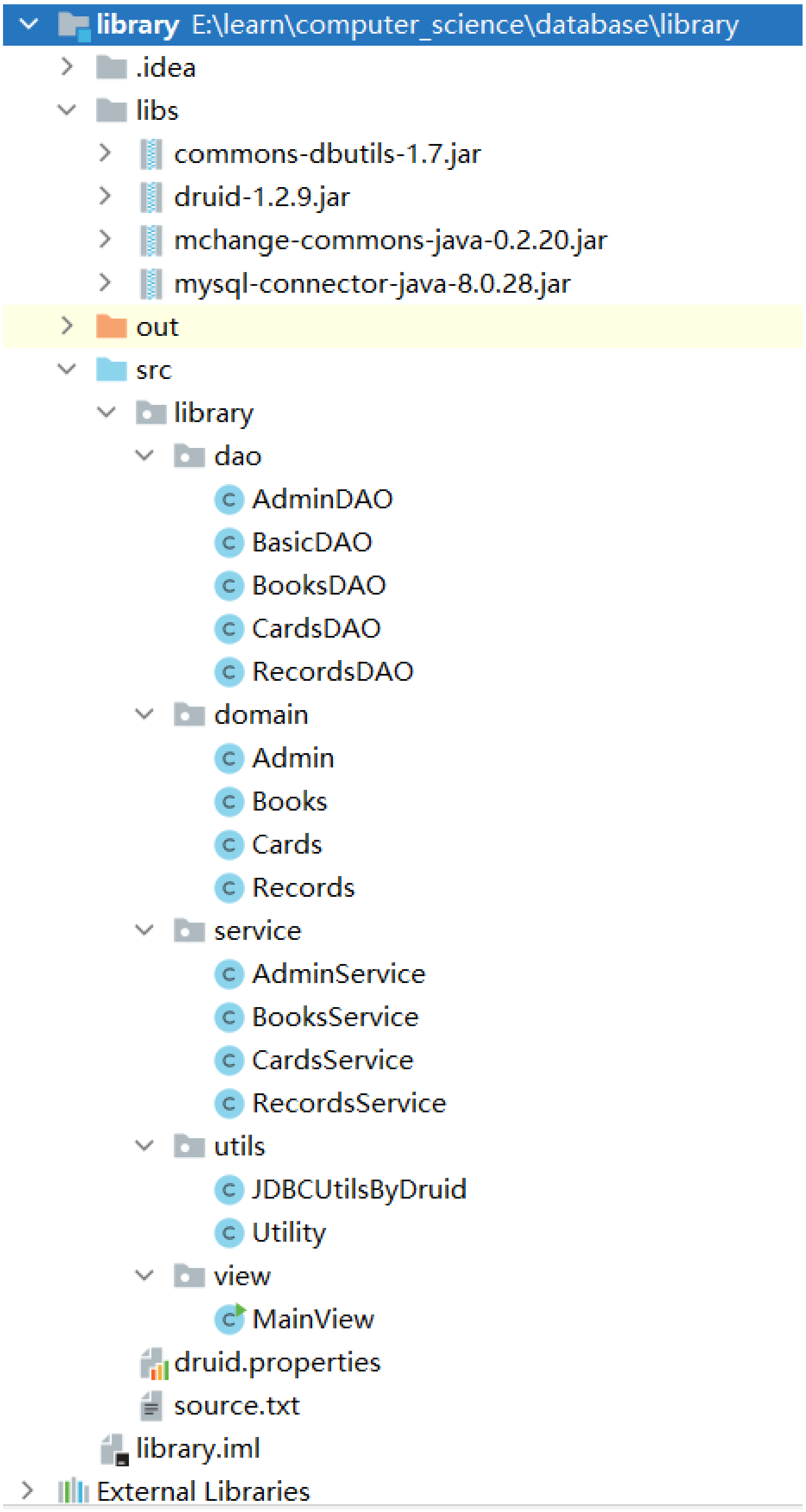
1. 采用的技术

数据库：MySQL
语言：java
连接API：MySQL JDBC 8.0.28
连接池：Druid
外部工具类：Apache

2. 详细描述

文件树：

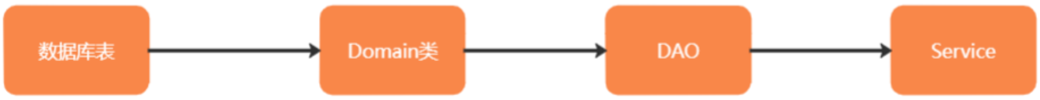
正如前文展示的系统架构一样，在代码方面利用包将相应的层级进行了分类，提高可读性和编写的效率



```
library E:\learn\computer_science\database\library
    .idea
    libs
        commons-dbutils-1.7.jar
        druid-1.2.9.jar
        mchange-commons-java-0.2.20.jar
        mysql-connector-java-8.0.28.jar
    out
    src
        library
            dao
                AdminDAO
                BasicDAO
                BooksDAO
                CardsDAO
                RecordsDAO
            domain
                Admin
                Books
                Cards
                Records
            service
                AdminService
                BooksService
                CardsService
                RecordsService
            utils
                JDBCUtilsByDruid
                Utility
            view
                MainView
        druid.properties
        source.txt
    library.iml
External Libraries
```

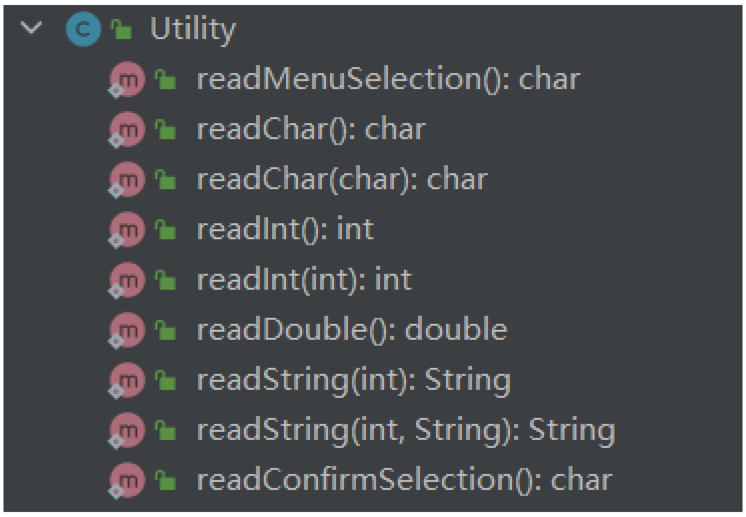
各模块设计：

由于各Service业务逻辑相同，都是从数据库的表创建Domain类，到基于BasicDAO构建DAO，再到调用DAO进行业务设计，故只需举一例详细介绍，在这之前，先介绍基本的工具类和BasicDAO


$$\begin{tikzcd}\text{数据库表} \arrow[r] & \text{Domain类} \arrow[r] & \text{DAO} \arrow[r] & \text{Service}\end{tikzcd}$$

Utility

Utility共创建了以下的几个方法，用以实现高效的java输入，不再受Scanner的困扰



```
Utility
    readMenuSelection(): char
    readChar(): char
    readChar(char): char
    readInt(): int
    readInt(int): int
    readDouble(): double
    readString(int): String
    readString(int, String): String
    readConfirmSelection(): char
```

JDBCUtilsByDruid

载入配置文件druid.properties，利用Druid连接池获取连接，同时实现了连接的关闭方法

```java
public class JDBCUtilsByDruid {
    private static DataSource ds;
    static {
        Properties properties = new Properties();
        try {
            properties.load(new FileInputStream( name: "src\\druid.properties"));
            ds = DruidDataSourceFactory.createDataSource(properties);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static Connection getConnection() throws SQLException {
        return ds.getConnection();
    }
    public static void close(ResultSet resultSet, Statement statement, Connection connection) {
        try {
            if (resultSet != null)
                resultSet.close();
            if (statement != null)
                statement.close();
            if (connection != null)
                connection.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

BasicDAO

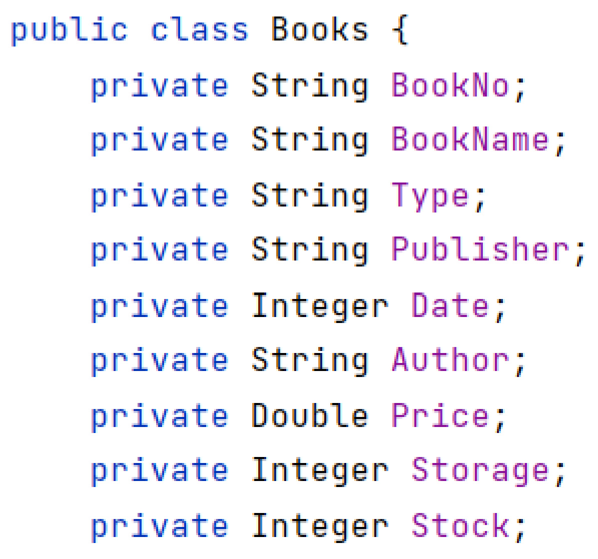
利用Apache的queryrunner，实现基本的update和select查询语句的执行，可以被任何DAO调用。基于此，不再需要设置与connection生命周期一样的PreparedStatement来接收查询值，增强了封装性和易用性，同时也能一样防止数据库注入的问题。

```java
public class BasicDAO<T> {
    QueryRunner qr = new QueryRunner();
    public int update(String sql, Object... parameters) {
        Connection connection = null;
        int update = 0;
        try {
            connection = JDBCUtilsByDruid.getConnection();
            update = qr.update(connection, sql, parameters);
            return update;
        } catch (SQLException e) {
            return update;
        } finally {
            JDBCUtilsByDruid.close( resultSet: null,  statement: null, connection);
        }
    }

    public List<T> queryMulti(String sql, Class<T> clazz, Object... parameters) {
        Connection connection = null;
        try {
            connection = JDBCUtilsByDruid.getConnection();
            return qr.query(connection, sql, new BeanListHandler<T>(clazz), parameters);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        } finally {
            JDBCUtilsByDruid.close( resultSet: null,  statement: null, connection);
        }
    }

    public T querySingle(String sql, Class<T> clazz, Object... parameters) {
        Connection connection = null;
        try {
            connection = JDBCUtilsByDruid.getConnection();
            return qr.query(connection, sql, new BeanHandler<T>(clazz), parameters);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        } finally {
            JDBCUtilsByDruid.close( resultSet: null,  statement: null, connection);
        }
    }

    public Object queryScalar(String sql, Object... parameters) {
        Connection connection = null;
        try {
            connection = JDBCUtilsByDruid.getConnection();
            return qr.query(connection, sql, new ScalarHandler(), parameters);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        } finally {
            JDBCUtilsByDruid.close( resultSet: null,  statement: null, connection);
        }
    }
}
```

Domain

相对应数据库中的表，创建变量与列名相同的Domain。由于在jar包的底层运行中会自动调用默认构造器，并使用getter，因此需要目标变量名与查询语句中一致。

如图创建Domain类，并相应地一键生成Generator，Getter和Setter

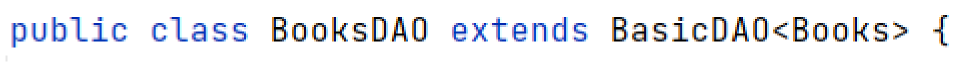


```
public class Books {
    private String BookNo;
    private String BookName;
    private String Type;
    private String Publisher;
    private Integer Date;
    private String Author;
    private Double Price;
    private Integer Storage;
    private Integer Stock;
```

BooksDAO

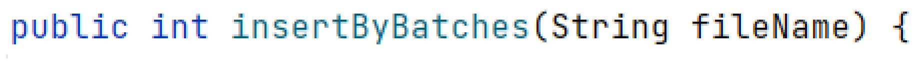
接下来就以BooksDAO来展示DAO对BasicDAO的继承：

直接用extends，后在尖括号中用相应的类覆盖T，即可实现继承



```
public class BooksDAO extends BasicDAO<Books> {
```

由于对于Books类，我们还需要实现根据文件名来批量导入，所以还需额外创建一个insert方法，具体代码不做赘述，值得一提的是在此使用batch来实现批处理，在接收完文件中所有的数据后统一执行数据的插入，只需一次对数据库的连接，还减少了编译的次数，大大提升了性能。据测试，在万级数据时，即可提升超百倍的速度，将时间从几十秒压缩至数毫秒之内。

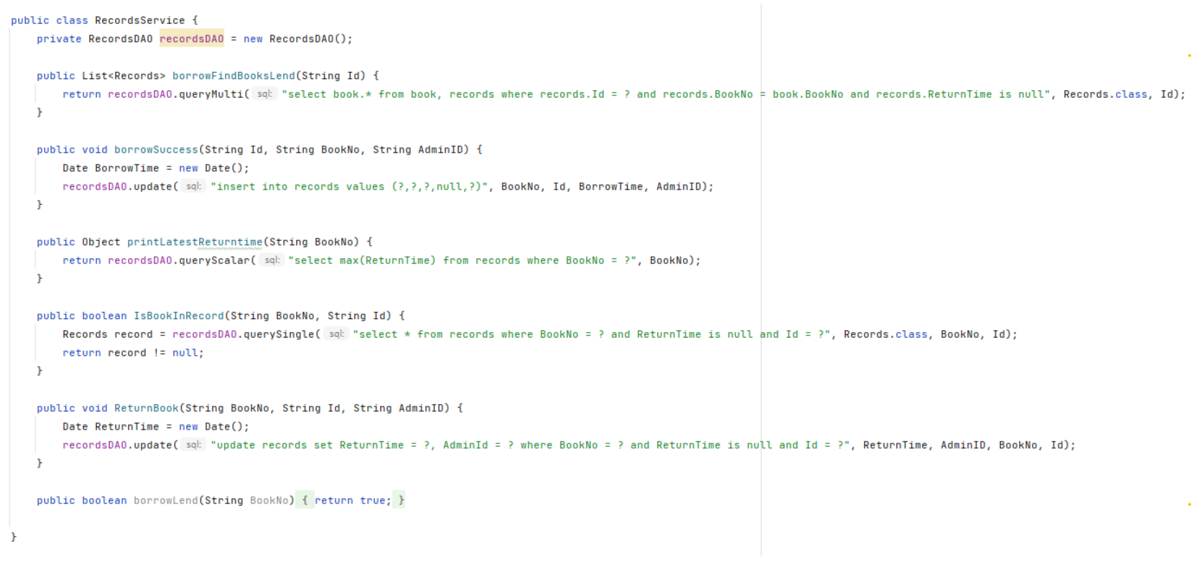


```
public int insertByBatches(String fileName) {
```

Service

最后我们需要使用DAO来完成各种功能，这些功能根据操作的对象存于各个不同的Service中，下以RecordsService为例

可以看到，Service新建DAO对象，并执行DAO中的方法来实现各种具体的查询语句。如寻找已借书籍的记录，就调用queryMulti，返回一张Records类的表，从而让程序进一步执行后续功能。



```java
public class RecordsService {
    private RecordsDAO recordsDAO = new RecordsDAO();

    public List<Records> borrowFindBooksLend(String Id) {
        return recordsDAO.queryMulti( sql: "select book.* from book, records where records.Id = ? and records.BookNo = book.BookNo and records.ReturnTime is null", Records.class, Id);
    }

    public void borrowSuccess(String Id, String BookNo, String AdminID) {
        Date BorrowTime = new Date();
        recordsDAO.update( sql: "insert into records values (?,?,?,null,?)", BookNo, Id, BorrowTime, AdminID);
    }

    public Object printLatestReturntime(String BookNo) {
        return recordsDAO.queryScalar( sql: "select max(ReturnTime) from records where BookNo = ?", BookNo);
    }

    public boolean IsBookInRecord(String BookNo, String Id) {
        Records record = recordsDAO.querySingle( sql: "select * from records where BookNo = ? and ReturnTime is null and Id = ?", Records.class, BookNo, Id);
        return record != null;
    }

    public void ReturnBook(String BookNo, String Id, String AdminID) {
        Date ReturnTime = new Date();
        recordsDAO.update( sql: "update records set ReturnTime = ?, AdminId = ? where BookNo = ? and ReturnTime is null and Id = ?", ReturnTime, AdminID, BookNo, Id);
    }

    public boolean borrowLend(String BookNo) { return true; }

}
```

MainView

最终设计完上述所有模块后，我们在MainView中生成命令行界面（由于java的图形化基本不太常用，我就没有学习制作相关的gui），调用各Service中的方法，完成图书管理系统的全部功能。