**北京交通大学计算机信息与技术学院**

**数据库系统原理课程设计**

**图书借阅管理系统**

学号：12281116

姓名：卢秋明

班级：计科1202

目录

[1 需求分析 4](#_Toc407807242)

[1.1 引言 4](#_Toc407807243)

[1.1.1 编写目的 4](#_Toc407807244)

[1.1.2 背景 4](#_Toc407807245)

[1.1.3 定义 5](#_Toc407807246)

[1.1.4 参考资料 5](#_Toc407807247)

[1.2 任务概述 5](#_Toc407807248)

[1.2.1 目标 5](#_Toc407807249)

[1.2.2 用户的特点 5](#_Toc407807250)

[1.2.3 假定和约束 6](#_Toc407807251)

[1.3 需求规定 6](#_Toc407807252)

[1.3.1 对功能需求的规定 6](#_Toc407807253)

[1.3.2 对性能的规定 9](#_Toc407807254)

[1.3.3 输入输出要求 9](#_Toc407807255)

[1.3.4 数据管理能力要求 9](#_Toc407807256)

[1.3.5 故障处理要求 10](#_Toc407807257)

[1.4 运行环境规定 10](#_Toc407807258)

[1.4.1 设备 10](#_Toc407807259)

[1.4.2 支持软件 10](#_Toc407807260)

[1.4.3 接口 10](#_Toc407807261)

[1.4.4 控制 10](#_Toc407807262)

[2 概要设计 11](#_Toc407807263)

[2.1 ER图 11](#_Toc407807264)

[2.2 系统功能设计 12](#_Toc407807265)

[2.2.1 总体功能 12](#_Toc407807266)

[2.2.2 各模块具体功能 13](#_Toc407807267)

[3 详细设计 14](#_Toc407807268)

[3.1 数据库设计 14](#_Toc407807269)

[3.1.1 创建数据库 14](#_Toc407807270)

[3.1.2 数据字典 14](#_Toc407807271)

[3.1.3 E-R图向关系模型的转换 16](#_Toc407807272)

[3.1.4 详细数据表设计 17](#_Toc407807273)

[3.1.5 约束 20](#_Toc407807274)

[3.1.6 对数据库安全性的研究。 21](#_Toc407807275)

[3.1.7 对数据库效率优化的探究 21](#_Toc407807276)

[3.2 应用层设计 24](#_Toc407807277)

[3.2.1 Qt Model/View 模式 24](#_Toc407807278)

[3.2.2 为了更好的扩展性，还用了很多设计模式的思想 25](#_Toc407807279)

[4 系统实现 27](#_Toc407807280)

[4.1 系统部署 27](#_Toc407807281)

[4.1.1 数据库部署 27](#_Toc407807282)

[4.2 基本功能 27](#_Toc407807283)

[4.2.1 用户登录 27](#_Toc407807284)

[4.2.2 用户注销 29](#_Toc407807285)

[4.2.3 图书信息查询 30](#_Toc407807286)

[4.2.4 借出还书功能 30](#_Toc407807287)

[4.2.5 添加新书，编辑旧书 32](#_Toc407807288)

[4.2.6 人员信息注册 34](#_Toc407807289)

[4.2.7 人员信息的编辑 35](#_Toc407807290)

[4.2.8 会员级别字典表 35](#_Toc407807291)

[4.2.9 图书类别字典表 36](#_Toc407807292)

[5 附录：数据库设计的优点说明 36](#_Toc407807293)

[5.1.1 建立存储过程 36](#_Toc407807294)

[5.1.2 SQL拼接 37](#_Toc407807295)

# 需求分析

## 引言

### 编写目的

此软件需求说明书编写是为了方便编码人员及测试和项目负责人能够对整个软件的功能有整体的把握以及严格的按照需求分析进行实现，可以使得整个软件的开发过程能够更加有效的实现。通过软件需求说明书其中对系统的各个功能都作了详细的描述，通过对需求的分析，了解客户的要求，可以保证软件的开发符合客户的需求，以及能够准确及时地完成整个项目的开发。本次课程设计虽然只是一个人负责完成，但为了保证完成的质量，可开发的每个环节都不可或缺。做好需求分析可以为数据库设计、应用设计以及最终编码奠定基础，提高开发效率，确保应用质量。

### 背景

在信息化时代，各种数据业务管理系统已经随处可见，这些管理软件为数据处理的效率性和准确性提供了保障。在现在互联网安全日益重要的情况下，C/S架构模式的业务系统越来越受到欢迎。比起B/S架构，C/S架构最主要的优点在于能够给服务器减轻压力，具有更高的安全性和稳定性。其数据库可以在客户端也可以在服务器端。

C/S一般面向相对固定的用户群，对信息安全的控制能力很强，但是C/S架构相对来说系统维护和升级相对较难。在目前，大多数图书借阅系统都是采用了B/S架构，但是由于学校的图书管理系统，面对的群体都相对较稳定，而且读者的信息安全也比较重要，B/S架构有时没有那么优的特点，所以，为了保障更好的服务效率和安全，本次课程设计采用了C/S架构来进行开发。

Qt是一个非常优秀的跨平台C++图形用户界面应用程序的开发框架，可以轻松地做到一次编写，到处运行的跨平台效果。所以，采用Qt开发，不仅可以让读者能够在windows下轻松方便地使用软件，而且可以在Linux，Mac等其他环境使用，高效的管理，使得图书借阅系统的使用更加方便，不仅使得服务器的压力大大减轻，而且也为使用人员放心，舒畅的使用。

### 定义

**C/S**：Client/Server，客户机/服务器模式。它是软件系统体系结构，通过它可以充分利用两端硬件环境的优势，将任务合理分配到Client端和Server端来实现，降低了系统的通讯开销。

**Qt：**由奇趣公司在1991年开发的一个跨平台C++图形用户界面应用程序开发框架，它提供给应用程序开发者建立艺术级的图形界面所需的功能。除此之外，还有一个非常优秀的品质就是具有优秀的跨平台特性。

### 参考资料

1. 数据库系统概论. 王珊，萨师煊. ，高等教育出版社 2010年12月
2. C++ GUI Qt 4 编程：第2版/（加）布兰切特（Blanceette,J.），（英）萨默菲尔德（Summerfield,M.）著；闫峰欣等译 编辑：电子工业出版社，2008.8

## 任务概述

### 目标

本实验要求实现一个图书借阅管理系统。该系统允许两类人员使用，管理员可以对系统进行初始化和修改数据库中的数据，如图书入库、办理借书证、图书借阅、超期处理和挂失处理等；读者可以查询图书信息和本人的借阅信息。

总的来看，项目属于数据库系统的高级应用开发。任务有需求方（甲方）数据库系统概论课程设计提出，由我方（乙方）即北京交通大学计算机系学生负责开发实现，通过相应的开发环境从而根据给定的具体的需求而完成项目。

### 用户的特点

图书借阅系统主要是涉及两类用户，读者和管理员。读者如果还没有读者编号，需要进行注册，申请等手续才可登陆。对于读者来说，使用的频率相对来说是没有很高，只是需要的时候才登陆上去进行借书和还书等功能。

对于管理员来说，使用的频率比较高，需要经常对图书进行编辑，新书入库等等，查阅图书的信息，所以需要完成图书系统的初始化和更好更快的服务，增强系统的安全性和稳定性，同时使系统使用起来更加流畅，更加方便。

### 假定和约束

数据库借阅系统为数据库系统概论的课程设计，现实中的系统复杂性高，需要团队合作开发。而课程设计从需求分析、概要设计、详细设计以及最终编码、测试运行全由一个人完成。因此，在不影响系统整体目标的情况下，对部分细节做出一些假定和约束，尽可能地使课程设计在可接受范围内取得最佳效果。以下为系统的一些假定和约束：

1.读者有ID唯一标识；

2.人员的性别只能是“男”，“女”，“保密”几种状态之一；

3.每本书都有ID唯一标识，且每次只能是一个人借出。

## 需求规定

### 对功能需求的规定

#### 功能需求概述

|  |  |
| --- | --- |
| **功能描述** | **简要说明** |
| 登陆 | 读者、管理员、超级管理员身份认证，进入系统 |
| 注销 | 安全退出系统 |
| 图书查询 | 图书的基本信息和状态信息的查询 |
| 人员查询 | 人员的基本信息和人员的证件状态信息 |
| 借阅查询 | 查询图书的借阅情况 |
| 编辑图书信息 | 对图书信息进行编辑 |
| 图书入库 | 新书编辑入库 |
| 办理图书证 | 添加新的读者信息 |
| 挂失处理 | 证件挂失和图书挂失 |
| 逾期未还处理 | 列出未还图书清单和读者的信息 |
| 借书还书交互 | 读者对图书进行借阅 |
| 人员信息修改 | 对人员的信息进行修改 |
| 初始化系统 | 清空当前信息，重置系统 |

#### 功能需求描述

该系统有普通用户、管理员用户和超级管理员用户三种，每种用户可以使用的功能不同。

##### 读者用户

1. **查询图书的各种信息**

按照书名、图书类别、作者、出版社等进行模糊查询。

1. **查询本人的借阅信息**

列出本人所有的借阅历史信息（六个月内，一年内）和当前借阅信息（指未 还的书籍）；

1. **查看本人的基本信息**

列出本人的一些基本信息，如手机号、证件名称等。

1. **修改本人的基本信息**

对本人的一些基本信息可以进行修改，如登录密码，手机号等。

1. **进行借书和还书等交互**

对图书馆里还未借出的书可以进行借阅，对于自己已经借阅的书可以进行还 书操作。

##### **管理员**用户

1. **查询图书的各种信息**

查询馆藏图书和借阅图书，分开不同的类别以列表来进行显示，点击可以查 询该图书的详细信息。

1. **查询读者的各种信息**

对图书馆的人员分三种类别以列表的形式来进行显示，点击可以查看该人员 的详细信息。

1. **编辑图书信息**

可以对图书馆的书籍进行编辑。

1. **图书入库**

新书编辑入库。

1. **办理借书证**

添加新读者，完成新读者的借书证问题。

1. **图书借阅和归还**

对图书的借阅和归还进行处理。

1. **逾期未还处理**

列出未还的图书清单和读者清单。

1. **挂失处理**

包括借书证的挂失和图书遗失处理。

1. **查看并添加图书类别字典表**

添加新的图书类型，或是修改员图书类型。

1. **查看并添加人员类别等级字典表**

添加新的人员类型，或是修改人员类型的一些基本信息。

##### 超级管理员用户

1. **在普通管理员的基础上，增加添加和删除管理员的权限。**

### 对性能的规定

#### 精度

此应用系统设计到许多敏感信息，如读者的证件号，读者的会员费用等等，这些数据需要比较严格精度控制，不仅需要数据库中的数据类型设计合理，而且在数据录入时要减小出错率。

#### 时间特性要求

对于一个读者量较少的图书馆来说，时间在系统中不是主要矛盾。但如果终端比较多时，就要考虑多种情况，如系统的响应时间，并发控制，多表同时操作等等。为了系统的健壮性，还需要从不同的角度入手，减小时间因素对系统运行的影响。为此，还需要从数据库，应用程序等等进行优化。

#### 灵活性

在C/S模式下，系统相对比较稳定，而且响应时间也比较快。因为采用的是Qt来开发，所以做到跨平台。而且，相对于Java等其他语言来说，C++的运行效率明显的提高，能做到更好的服务。

### 输入输出要求

本系统对于输入输出无特殊要求，所有的信息都在界面上进行录入，处理结果也以界面的形式展示给使用者。

### 数据管理能力要求

系统需要将对数据库的所有操作封装到应用层，方便使用者在图形化的界面下直观地对数据进行操作。

### 故障处理要求

任何系统都有可能遭受风险而出现故障，而数据对一个应用来说最重要了，因此需要对数据库定期备份，在出现问题时可快速恢复。

## 运行环境规定

### 设备

#### 数据库服务器

系统使用的数据库为MySQL数据库，所以数据库服务器可以为Windows，Linux，Unix等。

#### 客户端

无特殊限制。只要安装相应的exe文件就可以运行。

### 支持软件

编译器：Qt5.3.1

操作系统：Windows，Linux或其他。

服务程序：MySQL服务。

### 接口

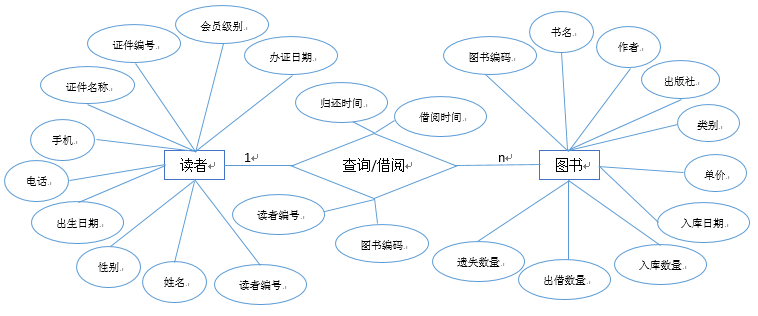
接口：该软件系统可以进行图书管理，包括新书入库，编辑图书，借阅等等。

### 控制

点击运行exe文件，之后登陆等操作。

# 概要设计

## ER图



## 系统功能设计

### 总体功能

#### 功能结构图



#### 与不同用户角色的交互

##### 验证功能

系统将验证用户身份，输入用户名LibraryID和密码，随后的交互取决于用户的身份。

##### 读者功能

* 查询图书的各种信息
* 查询本人的借阅信息
* 查看的基本信息
* 修改本人的基本信息
* 进行图书的借阅的归还

##### 管理员功能

* 查询图书的信息
* 查询读者的信息
* 编辑图书信息
* 图书入库
* 人员添加
* 办理图书证
* 图书借阅和归还
* 挂失处理
* 查看并添加图书类别字典表
* 查看并添加人员类别字典表

### 各模块具体功能

#### 查询系统

##### 图书信息查询

系统可以查询图书的信息（按图书的书名，作者，还有出版社等等）。

##### 人员信息查询

查询系统中所有客人的各种信息。

##### 借阅信息查询

查询系统中图书的借阅信息，可以通过人员的信息或是图书的信息来查询。

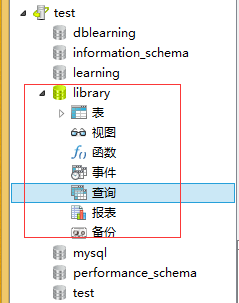
# 详细设计

## 数据库设计

### 创建数据库

本次课程设计需要一个独立的数据库，为此创建一个名为library的数据库。

如图所示：



### 数据字典

#### 数据结构

|  |  |
| --- | --- |
| 名称 | 系统用户 |
| 含义 | 登录系统进行相关操作的用户 |
| 组成 | 用户id，用户编码，用户姓名，性别。出生日期，电话，手机，邮箱，用户角色，用户备注，最近登录，最近登录时间 |

|  |  |
| --- | --- |
| 名称 | 用户字典 |
| 含义 | 用户类型字典 |
| 组成 | 用户类型id，用户类型名称 |

|  |  |
| --- | --- |
| 名称 | 管理员信息 |
| 含义 | 管理员的基本信息 |
| 组成 | id，管理员的id |

|  |  |
| --- | --- |
| 名称 | 读者信息 |
| 含义 | 读者对象 |
| 组成 | id，读者id，证件名称，证件编码，会员级别，办证时间 |

|  |  |
| --- | --- |
| 名称 | 图书信息 |
| 含义 | 图书的基本信息 |
| 组成 | 图书的id，书名，作者，出版社，类别，单价，入库日期，入库数量，遗失数量 |

|  |  |
| --- | --- |
| 名称 | 图书唯一标识 |
| 含义 | 书的唯一标识 |
| 组成 | id，图书编号，图书的id，是否借出 |

|  |  |
| --- | --- |
| 名称 | 图书类别字典 |
| 含义 | 图书类别字典表 |
| 组成 | 图书类别的id，图书类别的名称 |

|  |  |
| --- | --- |
| 名称 | 借阅信息 |
| 含义 | 书的借阅信息 |
| 组成 | 借阅id，读者，图书编号，出借时间，应还日期，遗失 |

|  |  |
| --- | --- |
| 名称 | 会员级别字典 |
| 含义 | 会员级别的所有集合 |
| 组成 | 会员级别的id，级别名称，最长借阅时间，最多借书数，会费 |

|  |  |
| --- | --- |
| 名称 | 挂失 |
| 含义 | 挂失信息 |
| 组成 | id，读者编号，挂失编号，备注信息 |

#### 处理过程

|  |  |  |  |
| --- | --- | --- | --- |
| 过程吗 | 输入数据 | 输出数据 | 处理逻辑 |
| 用户登录 | 用户名，密码 | 登录是否承购 | 如果用户输入的用户名和密码有效，则根据用户类型跳转到相应管理页面，否则返回登录 |
| 图书入库 | 图书的各项信息 | 入库是否成功 | 根据输入的数据检测系统检测输入的图书信息是否符合要求，若符合，则入库成功，否则，则不成功 |
| 用户注册 | 用户注册信息 | 用户注册是否成功 | 根据输入数据检测，判断是否符合要求，若符合，则注册成功 |
| 图书借阅 | 借阅图书的编号 | 借阅是否成功 | 如该图书还在馆中，则根据检测系统，符合条件则可以借阅。 |
| 图书挂失 | 图书编号，读者id | 挂失是否成功 | 根据图书信息，找到相关图书，并进行挂失 |
| 图书查询 | 相关查询条件 | 查询结果 | 若馆中有该图书，则查询成功，若没有，则查询失败 |

### E-R图向关系模型的转换

从表的设计中可以看出，系统一共存在9个关系，4个1对多关系，1个多对多关系。

实体型转换为关系模式。实体的属性就是关系的属性，实体的码就是关系的码。对于实体间的联系则有以下不同的情况：

一个m ：n 联系转换为一个关系模式。与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为各实体码的组合。

一个1 ：n 联系可以转换为一个独立的关系模式，也可以与n 端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为n端实体的码。

一个1 ：1 联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。三个或三个以上实体间的一个多元联系可以转换为一个关系模式。与该多元联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为各实体码的组合。

### 详细数据表设计

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 用户信息表（user） | | | | | | |
| 字段名 | 名称 | 字符类型 | 主键 | 外键 | 允许空 | 备注 |
| user\_id | 用户id | int | 是 |  | NOT NULL |  |
| user\_code | 用户编号 | varchar |  |  | NOT NULL |  |
| user\_name | 用户姓名 | varchar |  |  | NOT NULL |  |
| user\_psw | 用户密码 | varchar |  |  | NOT NULL |  |
| sex | 性别 | tinyint |  |  | NOT NULL |  |
| birthday | 出生日期 | date |  |  |  |  |
| phone | 电话 | varchar |  |  |  |  |
| mobile | 手机 | VARCHAR |  |  |  |  |
| mail | 邮箱 | VARCHAR |  |  |  |  |
| user\_type | 用户角色 | int |  | 是 | NOT NULL |  |
| user\_remark | 用户备注 | varchar |  |  |  |  |
| user\_lastlogtime | 最近登录时间 | datetime |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 用户角色字典表(user\_type) | | | | | | |
| 字段名 | 名称 | 字符类型 | 主键 | 外键 | 允许空 | 备注 |
| user\_type\_id | user\_type的id | int | 是 |  | NOT NULL |  |
| user\_type\_name | user\_type的名称 |  |  |  | NOT NULL |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 管理员信息表（managers） | | | | | | |
| 字段名 | 名称 | 字符类型 | 主键 | 外键 | 允许空 | 备注 |
| id | 表本身的id | int | 是 |  | NOT NULL |  |
| manager\_id | 管理员的id | int |  | 是 | NOT NULL |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 读者信息表（readers） | | | | | | |
| 字段名 | 名称 | 字符类型 | 主键 | 外键 | 允许空 | 备注 |
| id | 表本身的id | int | 是 |  | NOT NULL | 此字段只是为了更好的开展 |
| reader\_id | 读者的id | int |  | 是 | NOT NULL | 关联user表的user\_id字段 |
| card\_name | 证件名称 | varchar |  |  | NOT NULL |  |
| card\_id | 证件编号 | varchar |  |  | NOT NULL |  |
| level | 会员级别 | int |  | 是 | NOT NULL | 会员级别的id |
| day | 办证日期 | date |  |  | NOT NULL |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 图书信息表（books） | | | | | | |
| 字段名 | 名称 | 字符类型 | 主键 | 外键 | 是否为空 | 备注 |
| book\_id | 图书的id | int | 是 |  | NOT NULL |  |
| book\_name | 书名 | varchar |  |  | NOT NULL |  |
| author | 作者 | varchar |  |  | NOT NULL |  |
| publishing | 出版社 | varchar |  |  | NOT NULL |  |
| category\_id | 类别 | int |  | 是 | NOT NULL | 图书类别字典表的id |
| price | 单价 | varchar |  |  |  |  |
| date\_in | 入库日期 | date |  |  | NOT NULL |  |
| quantity\_in | 入库数量 | int |  |  | NOT NULL |  |
| quantity\_out | 出库数量 | int |  |  | NOT NULL |  |
| quantity | 遗失数量 | int |  |  | NOT NULL |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 图书唯一标识表（unique\_book） | | | | | | |
| 字段名 | 名称 | 字符类型 | 主键 | 外键 | 是否为空 | 备注 |
| id | id | int | 是 |  | NOT NULL |  |
| book\_code | 图书的编号 | int |  |  | NOT NULL |  |
| book\_id | 图书的id | int |  | 是 | NOT NULL | 指向books表的book\_id |
| isborrow | 是否借出 | int |  |  | NOT NULL |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 图书类别字典表(book\_category) | | | | | | |
| 字段名 | 名称 | 字符类型 | 主键 | 外键 | 是否为空 | 备注 |
| category\_id | 图书类别的id | int | 是 |  | NOT NULL |  |
| category\_name | 图书类别的name | varchar |  |  | NOT NULL |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 借阅表（borrow） | | | | | | |
| 字段名 | 名称 | 字符类型 | 主键 | 外键 | 是否为空 | 备注 |
| id | 借阅表的id | int | 是 |  | NOT NULL |  |
| reader\_id | 读者编号 | int |  | 是 | NOT NULL |  |
| book\_id | 图书编号 | int |  | 是 | NOT NULL |  |
| date\_borrow | 出借日期 | date |  |  | NOT NULL |  |
| date\_return | 应还日期 | date |  |  | NOT NULL |  |
| loss | 遗失 | bit |  |  | NOT NULL |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 会员级别（member\_level） | | | | | | |
| 字段名 | 名称 | 字符类型 | 主键 | 外键 | 是否为空 | 备注 |
| level\_id | 会员级别 | int | 是 |  | NOT NULL |  |
| level\_name | 会员级别的名称 | varchar |  |  | NOT NULL |  |
| days | 最长出借天数 | int |  |  | NOT NULL |  |
| numbers | 最多借书册数 | int |  |  | NOT NULL |  |
| fee | 会费 | double |  |  | NOT NULL |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 挂失(loss\_reporting) | | | | | | |
| 字段名 | 名称 | 字符类型 | 主键 | 外键 | 是否为空 | 备注 |
| id | 挂失的id | int | 是 |  | NOT NULL |  |
| reader\_id | 读者编号 | int |  | 是 | NOT NULL | 关联readers |
| loss\_date | 挂失日期 | datetime |  |  | NOT NULL |  |
| remark | 备注信息 | varchar |  |  |  |  |

### 约束

用户表——用户类型必须在用户字典里面的

图书表——图书类型必须在图书字典里面的

借阅表——借阅时间必须小于归还时间

### 对数据库安全性的研究。

#### 对用户的密码等重要信息的管理

由于用户的密码，证件等信息非常重要，虽然本系统采用的是C/S架构，在安全上有了很大的保障，但是对于用户的机密信息，需要采用加密的方式，MD5是一个非常常用的加密方式，但是现在来说，只是一次加密还远远不够，会出现对应的密码，所以，本系统采用了两次加密，使得加密大大提高。还有很多其他方面，不一一详述。

#### 用户权限分配

不同的用户类型具有不一样的权限，例如，只有超级管理员才具有添加管理员的权限，图书入库等一般的管理员就可以，读者只有借阅，归还，修改个人信息等基本权限。权限的控制使得系统在安全上大大提高，同时也保障了数据库的安全。

### 对数据库效率优化的探究

#### 规范化设计与适量冗余的权衡

一般来说，逻辑数据库设计会满足规范化的前3级标准：第1规范：没有重复的组或多值的列。第2规范：每个非关键字段必须依赖于主关键字，不能依赖于1个组合式主关键字的某些组成部分。第3规范：1个非关键字段不能依赖于另一个非关键字段。

遵守这些规则的设计会产生较少的列和更多的表，因而也就减少了数据冗余，也减少了用于存储数据的页。但表关系也许需要通过复杂的合并来处理，这样会降低系统的性能。

在课程设计中，如果要查询图书借阅的具体情况，需要将借阅表，唯一标识的图书表，图书表，人员表和读者表功5张表进行连接查询，而这种查询在系统中的使用率极高，如果每次这样查询，必然会造成效率低下。因此，为了提高响应速度，降低数据库负载，可以考虑适当的冗余。

在课程设计中，我并没有添加冗余的字段，而是通过应用层的优化将表连接数量降低到3张以内，从这里我懂得，不是数据库的规范化程度越高，系统的效率就越高的。

#### 字段设计空间占用的权衡

数据库中表字段应尽量简化，因为一张表可能有很多记录，在不影响设计含义的情况下，应该选取占用空间少的数据类型。

本次课程设计中遇到的最多的数据类型是字符串，尤其是在char，varchar和nvarchar中的选择问题。结果查阅资料，最后得出以下结论：

char(n)是长度为n个字节的定长的非Unicode的字符数据。n为一个介于1到8000之间的值。其存储大小为输入数据的实际字节长度，而不是n个字节。如果你输入的实际字节长度少于n那么其他位置会被空格填充。在数据库存储中英文字母和数字站一个字节，汉字站两个字节。那么char(n)最多可以存储n个英文字母或数字，或者n/2个汉字。Varchar(n)是长度为n 个字节的可变长度且非 Unicode 的字符数据。n 必须是一个介于1 和8,000 之间的数值。存储大小为输入数据的字节的实际长度，而不是 n 个字节。

所以，定长的字段可以用char(n)提高效率，长度范围变化大的字段可以采用varchar节省空间，而为了国际化需要时可以考虑nvarchar。

#### 复合主键与自增主键的选择

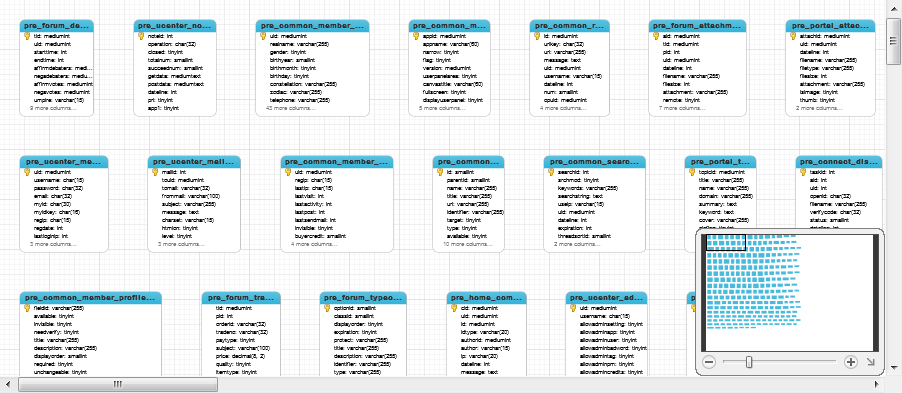
之所以设计复合主键，是因为一条记录可以有多个主属性共同决定。设计复合主键可以有效的避免数据的不一致或重复。但复合主键无论在查询时，还是应用层很可能会降价效率。

所以在课程设计中，比如借阅表，等等，我都使用了自增id，自增型的id效率最高，同时简化了应用层。

#### 外键参照关系不一定非得设计

关于数据库外键的是否应该设计，这个问题争议很多。从严格的数据一致性来讲，应该设计外键。但这不是绝对的，取决于应用的要求。

以下是目前中国最流行的论坛开发框架discuz数据库的截图。



让我惊讶的是，271张表没有一个外键参照关系，部分表甚至没有主键。而discuz却获得了巨大的市场，所有逻辑全交给了应用层。整个论坛运行十分流畅，原因就是数据库存取是不用检查主外键约束。

如果要在这个数据库添加外键参照关系，将会出现一个严重的问题，所有的表几乎都会涉及用户表，也就是说，几乎每个表的插入修改都会去用户表检查完整性，而一个论坛用户数量巨大，数据插入频繁，这些外键约束会严重降低效率。相反，在应用层，由于会话的连续性，只要应用层逻辑正确，基本不会出现数据不一致的情况。

本次课程设计由于表的数量少，还是将基本的外键参照关系进行了添加。

#### I/O瓶颈解决

数据库在进行操作时，不仅进行数据的变化，同时还要写日志。如果硬件允许，可以考虑将这些文件放在不同服务器上，防止并发量过大时I/O瓶颈制约了效率。

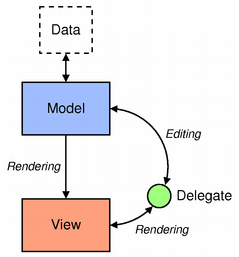
## 应用层设计

### Qt Model/View 模式

Model-View-Controller（MVC），是从Smalltalk发展而来的一种设计模式，常被用于构建用户界面。经典设计模式的著作中有这样的描述：

MVC有三种对象组成。Model是应用程序对象，View是它的屏幕表示，Controller定义了用户界面如何对用户输入进行响应。在MVC之前，用户界面设计倾向于三者糅合在一起，MVC对它们进行了解耦，提高了灵活性与重用性。

假如把view和controller结合在一起，结果就是model/view结构。这个结构依然是把数据存储与数据表示进行了分离，它与MVC都基于同样的思想，但它更加简单一些。这种分离使得在几个不同的view上同时显示同一个成为可能，也可以重新实现新的view，而不必改变底层的数据结构。为了更灵活的对用户输入进行处理，引入delegate这个概念。它的好处是，数据项的渲染与编程可以进行定制。



如上图所示，model与数据源通讯，并提供接口给结构中的别的组件使用。通讯的性质依赖于数据源的种类  
 与model实现的方式。view从model获取model indexes,后者是数据项的引用。通过把model indexes提供给model,view可以从数据源中获取数据。

在标准的views中，delegate会对数据项进行渲染，当某个数据项被选中时，delegate通过model indexes与model直接进行交流。总的来说，model/view 相关类可以被分成上面所提到的三组：models,views,delegates。这些组件通过抽象类来定义，它们提供了共同的接口，在某些情况下，还提供了缺省的实现。抽象类意味着需要子类化以提供完整的其他组件希望的功能。这也允许实现定制的组件。models,views,delegates之间通过信号，槽机制来进行通讯：

从model发出的信号通知view数据源中的数据发生了改变。  
从view发出的信号提供了有关被显示的数据项与用户交互的信息。  
从delegate发生的信号被用于在编辑时通知model和view关于当前编辑器的状态信息。

### 为了更好的扩展性，还用了很多设计模式的思想

在项目的设计中，我还用了很多设计模式的思想，例如，单例模式，抽象工厂模式，观察者模式等等。例如，在数据库的连接中，我使用了单例模式，使得数据库的连接能够保障唯一性，不产生错误的数据。在初始化系统中，一样。部分代码样例如下：

#ifndef SYSENVIROMENT\_H

#define SYSENVIROMENT\_H

#include <QtSql/QSqlDatabase>

class User;

class Readers;

class Managers;

#define GLobal\_DB (SysEnviroment::getInstance()->getDB())

class SysEnviroment

{

private:

SysEnviroment();

static SysEnviroment \*m\_Instance;

SysEnviroment(const SysEnviroment&) {}

SysEnviroment & operator = (const SysEnviroment &)

{

return \*this;

}

class Garbo

{

public:

~Garbo()

{

if (SysEnviroment::getInstance())

{

delete SysEnviroment::getInstance();

}

}

};

static Garbo Garbo;

public:

static SysEnviroment \*getInstance()

{

if (m\_Instance == NULL)

{

m\_Instance = new SysEnviroment();

}

return m\_Instance;

}

QSqlDatabase\* getDB();

void setUser(User \*user);

User\* getUser() const;

Readers\* getReader() const;

Managers\* getManager() const;

private:

QSqlDatabase\* db;

User\* user;

Readers\* reader;

Managers\* manager;

};

#endif // SYSENVIROMENT\_H

# 系统实现

## 系统部署

### 数据库部署

系统使用的是MySQL数据库，所以必须保障数据库启动。

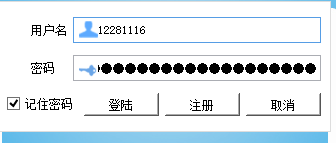
## 基本功能

### 用户登录

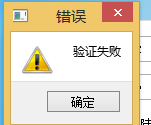
点击exe文件，运行，进入登录界面：



如果之前用户选择了记住密码，系统将会自动填充密码，若用户选择取消，则立即将本地的记住密码缓存给擦除。由于密码涉及到安全性，所以选择了在本地的磁盘上建一个XML文件，而不是保存在数据库或是其他地方，并进行加密，保障了数据的安全性。



当输入无效的用户信息时，系统会有相应的提示：



系统根据已经通过验证的用户信息自动判断用户类型，加载不同的内容，进入管理主页：





以上为管理员的界面，分两部分，一个是图书管理部分，一个是人员管理部分。



以上是普通用户管理模块。

### 用户注销

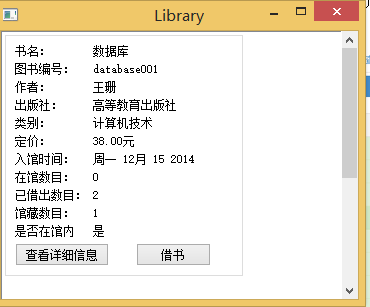
点击上方的退出标志，即可安全退出。

### 图书信息查询

因为在主界面中已经对图书进行了分类等等功能，所以还有一个非常重要的功能是高级检索，当用户想搜索的时候，可以在搜索Label里面输入相应的内容，进行搜索。搜索功能可以模糊搜索，做到了分词等效果

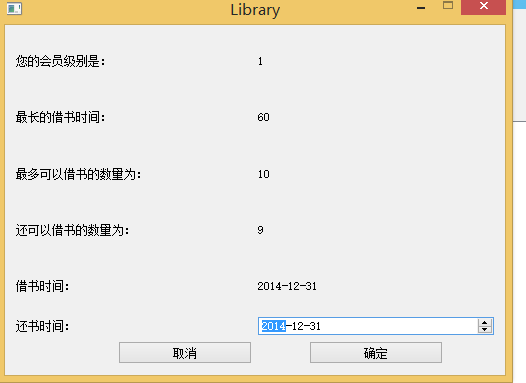


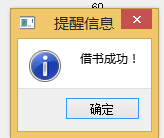




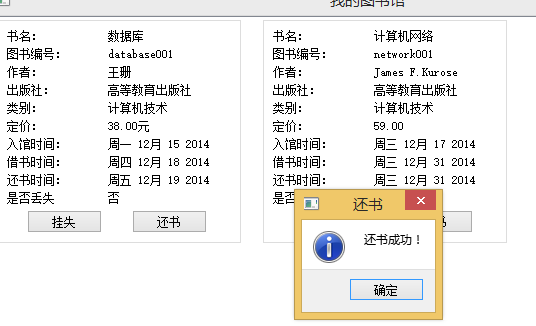
### 借出还书功能

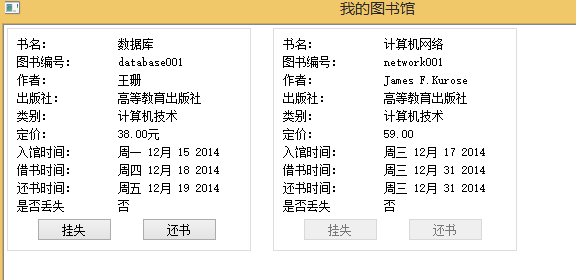
点击借书





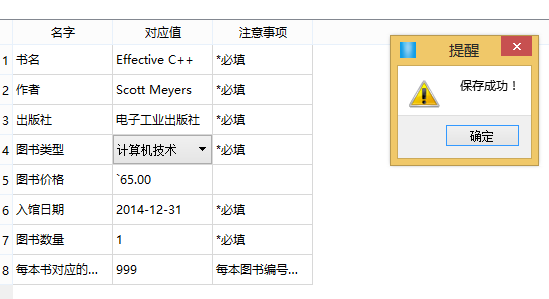
点击还书

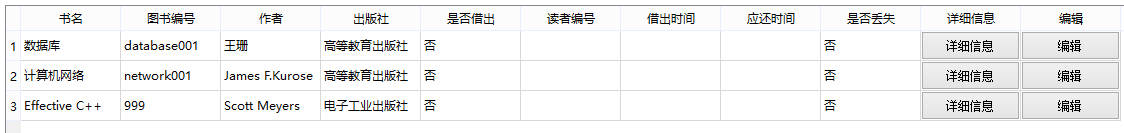




### 添加新书，编辑旧书

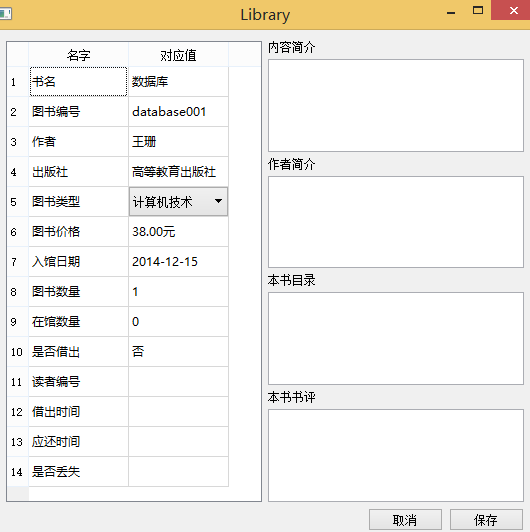






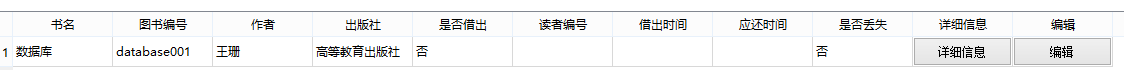
书成功地添加到书籍栏里面。

对旧书进行编辑

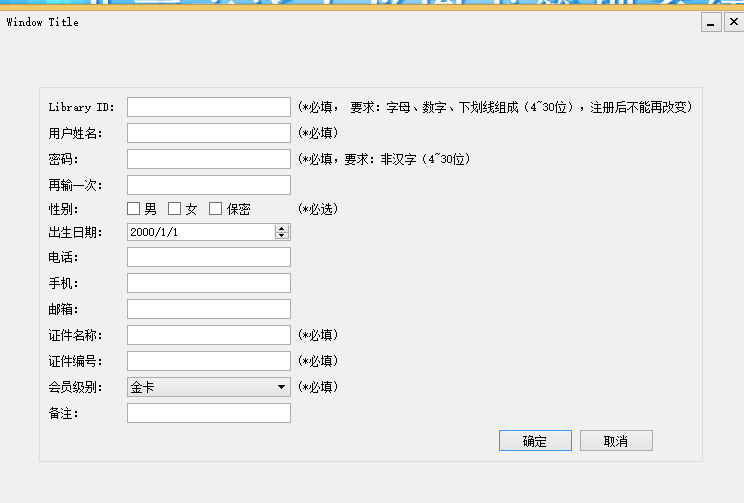


将书改为医学类型后，数就不在计算机栏目里，而是出现在医学栏目里

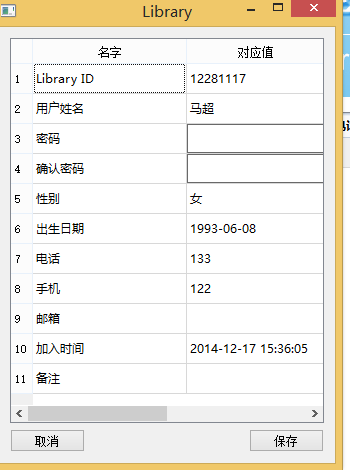




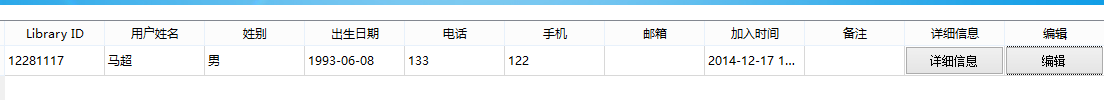
### 人员信息注册



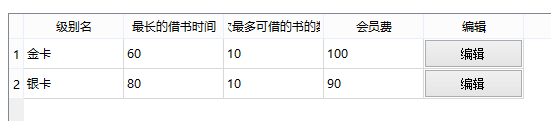
### 人员信息的编辑



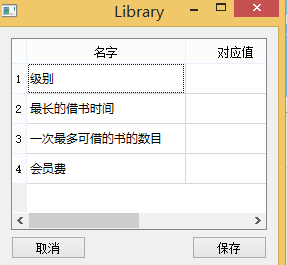
将性别从女变为男后，栏目里出现以下：



### 会员级别字典表



编辑



### 图书类别字典表



以上是一些基本的功能，没有一一列举，只能使部分截图。

# 附录：数据库设计的优点说明

### 建立存储过程

存储过程（Stored Procedure）是在大型数据库系统中，一组为了完成特定功能的SQL 语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象，任何一个设计良好的数据库应用程序都应该用到存储过程。

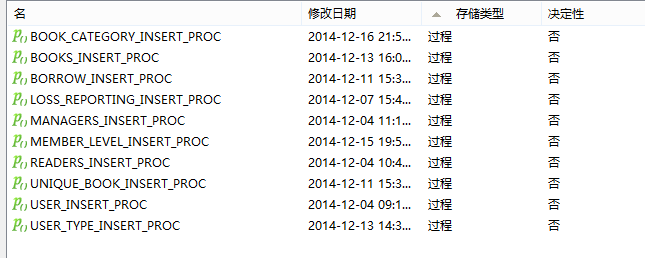
在本课程设计中，我也使用了不少的存储过程，存储过程相对直接操作sql已经来说有以下优点：

①重复使用。存储过程可以重复使用，从而可以减少数据库开发人员的工作量。

②提高性能。存储过程在创建的时候在进行了编译，将来使用的时候不再重新翻译。一般的SQL语句每执行一次就需要编译一次，所以使用存储过程提高了效率。

③减少网络流量。存储过程位于服务器上，调用的时候只需要传递存储过程的名称以及参数就可以了，因此降低了网络传输的数据量。

④安全性。参数化的存储过程可以防止SQL注入式攻击，而且可以将Grant、Deny以及Revoke权限应用于存储过程。



### SQL拼接

利用Qt的反射机制，通过使用字符串，来实现函数的通用化调用，例如你可以利用反射把很多函数放置到数组中，实现一次遍历，全部调用。实现类似于hibernate的功能。

反射机制是基于moc(meta object compiler)实现的,在这里多插一句（可以说Qt所有C++没有的特性，几乎都和这个有关系）。但是需要注意的是Qt提供的反射式基本的反射，不支持类的反射，这个与Java，C#还是有差别的。以下是SqL拼接的一些C++代码：

#include "basedao.h"

#include <QDebug>

#include <QSqlRecord>

#include <QSqlQuery>

#include <QSqlError>

#include <QVariant>

#include <QSqlResult>

#include <QVector>

#include <QSql>

#include <QMessageBox>

#include <QMetaObject>

#include <QMetaProperty>

#include <iostream>

#include "model/sysenviroment.h"

//开启数据库连接

#define DB\_BEGIN(x) \

bool ismyOpen = true; \

if (GLobal\_DB->isOpen()) \

{\

ismyOpen = false;\

} else {\

if (!GLobal\_DB->open())\

{\

Q\_ASSERT\_X(false, #x, "database can not open ");\

}\

}\

//关闭数据库连接

#define DB\_END()\

if(ismyOpen)\

GLobal\_DB->close();\

BaseDao::BaseDao()

{

}

/\*

\* @brief

\* 提高Dao层打开数据库

\* @name openDB()

\* @arg

\* void

\* @return bool

\* @author

\* QiumingLu Email:12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::openDB()

{

if (!GLobal\_DB->isOpen())

return GLobal\_DB->open();

return true;

}

bool BaseDao::closeDB()

{

if (GLobal\_DB->isOpen())

{

GLobal\_DB->close();

}

return true;

}

/\*

\* @brief

\* 创建事物

\* @name transaction()

\* @arg

\* void

\* @return bool

\* @author

\* QiumingLu Email:12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::transaction()

{

return GLobal\_DB->transaction();

}

/\*

\* @brief

\* 创建事物

\* @name

\* commit()

\* @arg void

\* @return bool

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::commit()

{

return GLobal\_DB->commit();

}

/\*

\* @brief

\* 执行存储过程

\* @name call\_proc

\* @arg

\* pObj QObject\* [In]需要进行数据库操作的对象，需要继承与QObject

\* proc\_name const [In] QString& 存储过程的名字

\* inFields const [In] QStringList& 需要输入的信息

\* outFields const [In] QStringList& 需要输出的信息

\* @return bool

\* true 执行成功

\* false 执行失败

\* @remark

\* 本函数默认是为insert写的存储过程，默认是插入所有的信息，out是信息主要是返回id

\* 输出的信息，通过pObj返回

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::call\_proc(QObject \*pObj, const QString &proc\_name,

const QStringList &inFields, const QStringList &outFields, bool isOutputDebug)

{

QString sql = QString("call %1 (").arg(proc\_name);

sql += makeQuestionMarkList(inFields.size() + outFields.size());

sql += ")";

DB\_BEGIN(call\_proc);

QSqlQuery query( (\*GLobal\_DB) );

query.prepare(sql);

int idx = 0;

for (int i = 0; i < inFields.size(); ++i)

{

QVariant var = pObj->property(inFields[i].toStdString().c\_str());

std::cout << inFields[i].toStdString() << " " << var.toString().toStdString() << std::endl;

if (var.type() == QVariant::Double) {

float tmp = var.toDouble();

query.bindValue(idx++, tmp);

}

//ID 外键

else if (var.type() == QVariant::Int && inFields[i].indexOf("\_ID", 0, Qt::CaseInsensitive)

&& pObj->property(inFields[i].toStdString().c\_str()).toInt() == -1)

{

query.bindValue(idx++, QVariant());

}

else

{

query.bindValue(idx++, var);

}

}

for (int i = 0; i < outFields.size(); ++i)

{

query.bindValue(idx++, "0", QSql::Out);

}

bool ret = query.exec();

if (ret)

{

idx = idx - outFields.size();

for (int i = 0; i < outFields.size(); ++i)

{

pObj->setProperty(outFields[i].toStdString().c\_str(), query.boundValue(idx + i));

}

} else {

if (isOutputDebug) {

qDebug() << "[DATABASE][\*INSERT\*][PROC\_NAME: " << proc\_name << "]ERROR: "

<< query.lastError().text() << "\n";

}

}

DB\_END();

return ret;

}

/\*

\* @brief

\* 执行存储过程

\* @name call\_proc

\* @arg

\* pObj QObject\* [In]需要进行数据库操作的对象，需要继承与QObject

\* proc\_name const [In] QString& 存储过程的名字

\* outFields const [In] QStringList& 需要输出的信息

\* @return bool

\* true 执行成功

\* false 执行失败

\* @remark

\* 本函数默认是为insert写的存储过程，默认是插入所有的信息，out是信息主要是返回id

\* 输出的信息，通过pObj返回

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::call\_proc(QObject \*pObj, const QString &proc\_name,

const QStringList &outFields, bool isOutputDebug)

{

QStringList excludeList("objectName");

excludeList.append(outFields);

QStringList inFields = getAvaProperties(pObj->*metaObject*(), excludeList);

return call\_proc(pObj, proc\_name, inFields, outFields, isOutputDebug);

}

/\*

\* @brief

\* 插入整个对象

\* @name Insert()

\* @arg

\* pObj QObject\* [In] 需要进行数据库操作的对象，需要继承与QObject

\* tableName [In] const QString& 表名

\* @return bool

\* true 执行成功

\* false 执行失败

\* @author

\* QiumingLu Email:12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::Insert(QObject \*pObj, const QString &tableName)

{

QStringList excludeList("objectName");

QStringList inFields = getAvaProperties(pObj->*metaObject*(), excludeList);

return Insert(pObj, tableName, inFields);

}

/\*

\* @brief

\* 插入制定的属性

\* @name Insert

\* @arg

\* pObj QObject\* [In] 需要进行数据库操作的对象，需要继承与QObject

\* tableName [In] const QString& 表名

\* inFields [In] const QStringList& 插入的属性信息列表

\* @return bool

\* true 执行成功

\* false 执行失败

\* @author

\* QiumingLu Email:12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::Insert(QObject \*pObj, const QString &tableName, const QStringList &inFields)

{

QString sql = QString(" insert into %1(").arg(tableName);

QString joins = inFields.join(",");

sql += joins + ") values (";

sql += makeQuestionMarkList(inFields.size());

sql += ")";

DB\_BEGIN(Insert);

QSqlQuery query( \*(GLobal\_DB) );

query.prepare(sql);

for (int i = 0; i < inFields.size(); ++i)

{

//ID外键

if (inFields[i].indexOf("\_ID", 0, Qt::CaseInsensitive)

&& pObj->property(inFields[i].toStdString().c\_str()).toInt() == -1) {

query.bindValue(i, QVariant());

} else {

query.bindValue(i, pObj->property(inFields[i].toStdString().c\_str()));

}

}

bool ret = query.exec();

if (!ret)

{

qDebug() << query.lastError().text();

}

DB\_END();

return ret;

}

/\*

\* @brief

\* 更新制定的属性

\* @name Update

\* @arg

\* pObj QObject\* [In] 需要进行数据库操作的对象，需要继承与QObject

\* tableName [In] const QString& 表名

\* conditions [In] const QStringLis& 条件

\* @return bool

\* true 执行成功

\* false 执行失败

\* @author

\* QiumingLu Email:12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::Update(QObject \*pObj, const QString &tableName, const QStringList &conditions)

{

QStringList excludeList("objectName");

excludeList.append(conditions);

QStringList updateFields = getAvaProperties(pObj->*metaObject*(), excludeList);

return Update(pObj, tableName, updateFields, conditions);

}

/\*

\* @brief

\* 更新制定的属性

\* @name Update

\* @arg

\* pObj QObject\* [In] 需要进行数据库操作的对象，需要继承与QObject

\* tableName [In] const QString& 表名

\* updateFields [In] const QStringList& 更新属性信息列表

\* conditions [In] const QStringLis& 条件

\* @return bool

\* true 执行成功

\* false 执行失败

\* @author

\* QiumingLu Email:12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::Update(QObject \*pObj, const QString &tableName, const QStringList &updateFields, const QStringList &conditions)

{

QString sql = QString(" update %1 set ").arg(tableName);

int nSize = updateFields.size();

for (int i = 0; i < nSize; ++i)

{

if (i != nSize - 1) {

sql += updateFields[i] + " = ?, ";

} else {

sql += updateFields[i] + " = ? ";

}

}

sql += " where 1 = 1";

for (int i = 0; i < conditions.size(); ++i)

{

sql += " and " + conditions[i] + " =? ";

}

DB\_BEGIN(Update);

QSqlQuery query( \*(GLobal\_DB) );

query.prepare(sql);

int nIdx = 0;

for (int i = 0; i < nSize; ++i)

{

QVariant variant = pObj->property(updateFields[i].toStdString().c\_str());

if (updateFields[i].indexOf("\_ID", 0, Qt::CaseInsensitive)

|| updateFields[i].indexOf("locker", 0, Qt::CaseInsensitive)) {

int tmp = pObj->property(updateFields[i].toStdString().c\_str()).toInt();

if (tmp == -1)

variant = QVariant();

}

query.bindValue(nIdx++, variant);

}

for (int i = 0; i < conditions.size(); ++i)

{

QVariant variant = pObj->property(conditions[i].toStdString().c\_str());

if (conditions[i].indexOf("\_ID", 0, Qt::CaseInsensitive)) {

int tmp = pObj->property(conditions[i].toStdString().c\_str()).toInt();

if (tmp == -1)

variant = QVariant("NULL");

}

query.bindValue(nIdx++, variant);

}

bool ret = query.exec();

QString exedState = ret? "SUCCESS" : "\*\*FAILURE\*\*";

qDebug() << "[DATABASE][\*UPDATE\*][CLASSNAME:" << pObj->*metaObject*()->className() << "]"

<< "[TABLENAME:" << tableName << "]" << "EXECUTEDSTATE:" << exedState << "]";

if (!ret)

qDebug() << query.lastError().text();

DB\_END();

return ret;

}

/\*

\* @brief

\* 删除满足条件的属性

\* @name Delete

\* @arg

\* pObj QObject\* [In] 需要进行数据库操作的对象，需要继承与QObject

\* tableName [In] const QString& 表名

\* conditions [In] const QStringLis& 条件

\* @return bool

\* true 执行成功

\* false 执行失败

\* @remark

\* 本函数只是为只有一个条件做准备的

\* @author

\* QiumingLu Email:12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::Delete(QObject \*pObj, const QString &tableName, const QString &condition)

{

QStringList conditions(condition);

return Delete(pObj, tableName, conditions);

}

/\*

\* @brief

\* 删除满足条件的属性

\* @name Delete

\* @arg

\* pObj QObject\* [In] 需要进行数据库操作的对象，需要继承与QObject

\* tableName [In] const QString& 表名

\* conditions [In] const QStringLis& 条件

\* @return bool

\* true 执行成功

\* false 执行失败

\* @remark

\* 本函数支持多条件拼接

\* @author

\* QiumingLu Email:12281116@bjtu.edu.cn

\* @test

\* no

\*/

bool BaseDao::Delete(QObject \*pObj, const QString &tableName, const QStringList &conditions)

{

QString sql = QString(" delete from %1 where 1 = 1 ").arg(tableName);

foreach (QString item, conditions) {

sql += " and " + item + " = ? ";

}

DB\_BEGIN(Delete);

QSqlQuery query(\*(GLobal\_DB));

query.prepare(sql);

for (int i = 0; i < conditions.size(); ++i)

{

query.bindValue(i, pObj->property(conditions[i].toStdString().c\_str()));

}

bool ret = query.exec();

QString exedState = ret? "SUCCESS" : "\*\*FAILURE\*\*";

if (!ret)

qDebug() << query.lastError().text();

else

qDebug() << "[DATABASE][\*DELETE\*][CLASSNAME:" << pObj->*metaObject*()->className()

<< "]" << "[TABLENAME:" << tableName << "]" << "[EXECUTEDSTATE:" << exedState <<"]";

DB\_END();

return ret;

}

/\*

\* @brief

\* 查询满足条件的数据

\* @name Select

\* @arg

\* pObj QObject\* [In] 需要进行数据库操作的对象，需要继承与QObject

\* tableName [In] const QString& 表名

\* outFields [In] const QStringList& outFields 要查询输出的信息

\* conditions [In] const QStringList& 条件

\* @return QRetVec

\* 返回查询的信息

\* @remark

\* 本函数仅支持多个条件的查询

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\*/

QRetVec BaseDao::Select(QObject \*pObj, const QString &tableName,

const QStringList &outFields, const QStringList &conditions)

{

QRetVec ret;

Q\_ASSERT(outFields.size() != 0);

QString sql("select ");

int nSize = outFields.size();

for (int i = 0; i < nSize; ++i)

{

if (i != nSize - 1)

{

sql += outFields[i] + ",";

} else {

sql += outFields[i];

}

}

sql += QString(" from %1 where 1 = 1").arg(tableName);

int nCnt = conditions.size();

for (int i = 0; i < nCnt; ++i)

{

sql += " and " + conditions[i] + " = ? ";

}

DB\_BEGIN(Select);

QSqlQuery query( (\*GLobal\_DB) );

query.prepare(sql);

for (int i = 0; i < nCnt; ++i)

{

query.bindValue(i, pObj->property( conditions[i].toStdString().c\_str() ) );

}

if (query.exec())

{

while (query.next()) {

QVector< CPair > oneRet(outFields.size());

for (int i = 0; i < outFields.size(); ++i)

{

int idx = query.record().indexOf( outFields[i] );

oneRet[i] = CPair( outFields[i], query.value(idx));

}

ret.append(oneRet);

}

}

DB\_END();

return ret;

}

/\*

\* @brief

\* 查询满足条件的数据

\* @name Select

\* @arg

\* pObj QObject\* [In] 需要进行数据库操作的对象，需要继承与QObject

\* tableName [In] const QString& 表名

\* condition [In] const QString& 条件

\* @return QRetVec

\* 返回查询的信息

\* @remark

\* 本函数仅支持一个条件的查询

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\*/

QRetVec BaseDao::Select(QObject \*pObj, const QString &tableName, const QString &condition)

{

QStringList conditions(condition);

return Select(pObj, tableName, conditions);

}

/\*

\* @brief

\* 查询满足条件的数据

\* @name Select

\* @arg

\* pObj QObject\* [In] 需要进行数据库操作的对象，需要继承与QObject

\* tableName [In] const QString& 表名

\* conditions [In] const QStringList& 条件

\* @return QRetVec

\* 返回查询的信息

\* @remark

\* 本函数仅支持多个条件的查询

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\*/

QRetVec BaseDao::Select(QObject \*pObj, const QString &tableName, const QStringList &conditions)

{

QStringList excludeList("objectName");

QStringList outFields = getAvaProperties( pObj->*metaObject*(), excludeList );

return Select(pObj, tableName, outFields, conditions);

}

/\*

\* @brief

\* 查询满足条件的数据

\* @name Select

\* @arg

\* pObj QObject\* [In] 需要进行数据库操作的对象，需要继承与QObject

\* tableName [In] const QString& 表名

\* outFields [In] const QStringList &outFields 要查询输出的信息

\* condition [In] const QString& 条件

\* @return QRetVec

\* 返回查询的信息

\* @remark

\* 本函数仅支持多个条件的查询

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\*/

QRetVec BaseDao::Select(QObject \*pObj, const QString &tableName,

const QStringList &outFields, const QString &condition)

{

QStringList conditions(condition);

return Select(pObj, tableName, outFields, conditions);

}

/\*

\* @brief

\* 执行sql语句，进行增删改操作

\* @name callADUSql

\* @arg

\* pObj QObject\* [In] 操作的对象

\* sql const QString& [In] 待执行的sql语句

\* fields const QStringList [In] 对应于sql语句中需要填充的?字段

\* @return QRetVec

\* 需要注意本函数仅能处理曾删改操作

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\*/

bool BaseDao::callADUSql(QObject \*pObj, const QString &sql, const QStringList &fields)

{

DB\_BEGIN(callADUSql);

QSqlQuery query( \*(GLobal\_DB) );

query.prepare(sql);

for (int i = 0; i < fields.size(); ++i)

{

if (fields[i].indexOf("\_ID", 0, Qt::CaseInsensitive)

&& pObj->property(fields[i].toStdString().c\_str()).toInt() == -1) {

query.bindValue(i, QVariant());

} else {

query.bindValue(i, pObj->property(fields[i].toStdString().c\_str()) );

}

}

bool ret = query.exec();

DB\_END();

return ret;

}

/\*

\* @brief

\* 执行sql语句，进行查询操作

\* @name callSelectSql

\* @arg

\* pObj QObject\* [In] 操作的对象

\* sql const QString& [In] 待执行的sql语句

\* inFields const QStringList [In] 输入字段

\* outFields const QStringList [In]输出字段

\* @return QRetVec

\* 需要注意本函数仅能处理查询操作

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\*/

QRetVec BaseDao::callSelectSql(QObject \*pObj, const QString &sql,

const QStringList &inFields, const QStringList &outFields)

{

DB\_BEGIN(callSelectSql);

QSqlQuery query( \*(GLobal\_DB) );

query.prepare(sql);

int idx = 0;

for (int i = 0; i < inFields.size(); ++i)

{

query.bindValue(idx++, pObj->property(inFields[i].toStdString().c\_str()) );

}

for (int i = 0; i < outFields.size(); ++i)

{

query.bindValue(idx++, pObj->property(outFields[i].toStdString().c\_str()) );

}

QRetVec ret;

if (query.exec())

{

while (query.next())

{

QVector<CPair> oneRet;

foreach (QString field, outFields) {

int idx = query.record().indexOf(field);

oneRet.append( CPair (field, query.value(idx) ) );

}

ret.append(oneRet);

}

}

DB\_END();

return ret;

}

/\*

\* @brief

\* 执行sql语句

\* @name callSql

\* @arg

\* sql const QString& [In] 待执行的sql语句

\*

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\*/

QVariant BaseDao::callSql(const QString &sql)

{

DB\_BEGIN(callSql);

QSqlQuery query( (\*GLobal\_DB) );

if (!query.prepare(sql))

{

qDebug() << query.lastError().text();

return -1;

}

if (query.exec())

{

if (query.next()) {

return query.value(0);

}

return -1;

} else {

qDebug() << query.lastError().text();

}

DB\_END();

return -1;

}

QVariant BaseDao::callSql(const QString &sql, QList<QVariant> conditions)

{

DB\_BEGIN(callSql);

QSqlQuery query( (\*GLobal\_DB) );

query.prepare(sql);

for (int i = 0; i < conditions.size(); ++i)

{

query.bindValue(i, conditions[i]);

}

if (query.exec())

{

while (query.next()) {

return query.value(0);

}

}

DB\_END();

return -1;

}

/\*

\* @brief

\* 执行存储过程，自定义输入输出参数

\* @name callProc

\* @arg

\* proc\_name [In] const QString& 存储过程名字

\* inFields [In] QList<QVariant> & 需要输入的信息

\* outFields [In] QList<QVariant> & 需要输出的信息

\* @return bool

\* true 执行成功

\* false 执行失败

\* @author

\* QiumingLu Email: 12281116@bjtu.edu.cn

\*/

bool BaseDao::callProc(const QString &proc\_name, QList<QVariant> &inFields,

QList<QVariant> &outFields, bool isOutputDebug)

{

QString sql = QString("call %1 (").arg(proc\_name);

sql += makeQuestionMarkList(inFields.size() + outFields.size());

sql += ")";

DB\_BEGIN(call\_proc);

QSqlQuery query( (\*GLobal\_DB) );

query.prepare(sql);

int idx = 0;

for (int i = 0; i < inFields.size(); ++i)

{

QVariant var = inFields[i];

if (var.type() == QVariant::Double) {

float tmp = var.toDouble();

query.bindValue(idx++, tmp);

} else {

query.bindValue(idx++, var);

}

}

for (int i = 0; i < outFields.size(); ++i)

{

query.bindValue(idx++, QSql::Out);

}

bool ret = query.exec();

if (ret)

{

idx = idx - outFields.size();

for (int i = 0; i < outFields.size(); ++i)

{

outFields[i] = query.boundValue(idx + i);

}

} else {

if (isOutputDebug) {

qDebug() << "[DATABASE][\*INSERT\*][PROC\_NAME: " << proc\_name << "]ERROR: "

<< query.lastError().text() << "\n";

}

}

DB\_END();

return ret;

}

/\*

\* @brief

\* 根据需要创建问号的个数，输出由逗号分隔的字符串

\* @name makeQuestionMarkList(int nSize)

\* @arg

\* nSize int [In] 输入问号的数量

\* @return QString

\* @author

\* QiumingLu Email:12281116@bjtu.edu.cn

\* @test

\* no

\*/

QString BaseDao::makeQuestionMarkList(int nSize)

{

QString ret = "";

for(int i = 0; i < nSize; ++i)

{

if (i != nSize - 1)

ret += "?,";

else

ret += "?";

}

return ret;

}

/\*

\* @brief

\* 通过反射得到反射表中的属性

\* @name getProperties

\* @arg

\* pMeta const QMetaObject\* 元对象

\* @return QStringList

\* 返回列表属性

\* @author

\* QiumingLu Email:12281116@bjtu.edu.cn

\* @test

\* no

\*/

QStringList BaseDao::getProperties(const QMetaObject \*pMeta)

{

Q\_ASSERT(pMeta != NULL);

QStringList retList;

int properyCnt = pMeta->propertyCount();

for (int idx = 0; idx < properyCnt; ++idx)

{

retList.append(pMeta->property(idx).name());

}

return retList;

}

QStringList BaseDao::getAvaProperties(const QMetaObject \*pMeta, const QStringList &excludes)

{

Q\_ASSERT(pMeta != NULL);

QStringList retList;

int properyCnt = pMeta->propertyCount();

for (int idx = 0; idx < properyCnt; ++idx)

{

QString name = pMeta->property(idx).name();

if (excludes.contains(name))

continue;

retList.append(name);

}

return retList;

}

int BaseDao::getId(QString &tableName, QString &outFields)

{

QString sql = "select max(" + outFields + ") from " + tableName;

DB\_BEGIN(getId())

QSqlQuery query(\*GLobal\_DB);

query.prepare(sql);

int id;

bool flags = query.exec();

if (flags)

{

if (!query.next()) {

QMessageBox::warning(0, "错误", "失败", "确定");

} else {

id = query.value(0).toInt();

}

}

DB\_END();

return id;

}