《数据库系统原理》课程设计

系统实现报告

题目名称： GRE(美国研究生入学考试)学习系统

学号及姓名：15231031 张楠

15231036 杨旺旺

2017年 12月 28 日

1. 系统功能需求分析

经过分析与设计，我们的GRE学习系统的功能主要包括用户登陆与注册、用户做历次考试的试卷、自动评判、试题评论浏览与添加、用户错题库管理、用户生词库管理等五个主要部分。详细需求如下：

①用户登陆与注册：

管理用户注册账户的操作，并验证用户登录的信息。注册时，会对数据库中的用户名进行查重；登录时，会对用户密码进行校验。

②用户完成历次GRE考试的试卷：

用户可以从系统提供的历次GRE考试的试卷中下载所需试卷，并完成试卷中的试题。系统首先会为用户展示历次考卷的时间，由用户选择试题；而后系统会将该次考试的题目展示给用户。

③自动评判机制：

当用户完成试卷提交后，可以自动判断用户提交答案的正误。同时，系统还将帮助用户回顾每一道题——在这个过程中，用户可以看到自己的答案与正确答案，同时看到其他人对每一道题的评论与讲解。此外，用户还可以选择是否将某一道题加入自己的错题库。

④试题评论浏览与添加管理：

在用户完成试卷后，可以浏览试卷中每道题目下的评论区中其他用户对该题的评论，自己也可以添加或回复评论。同时，用户在浏览自己的错题库时也可以随时浏览和添加评论。此外，如果用户认为某一个评论值得一看，还可以为该评论点赞。在这个过程中，系统会限制用户的点赞数，即不能超过一个。

⑤用户错题库管理：

生成用户专属的错题库，管理用户做错的题目，也支持用户将需要记录的题目添加到错题库中，方便用户日后复习回顾。

⑥用户生词库管理：

管理用户需要记忆的生词，提供生词的拼写以及中文解释等信息，方便用户复习回顾。

1. 系统功能结构设计

我们的GRE学习系统App采用了广泛使用的C/S架构(即Client/Server架构，客户端/服务器架构)，将任务合理地分配到了Android App形式的客户端和Servlet架构的服务端。服务器端主要提供数据管理、数据共享、数据及系统维护和[并发控制](https://baike.baidu.com/item/%E5%B9%B6%E5%8F%91%E6%8E%A7%E5%88%B6" \t "https://baike.baidu.com/item/CS%E6%9E%B6%E6%9E%84/_blank)等服务，客户端主要完成用户的具体的业务，C/S架构降低了系统的通讯开销，能够向用户提供及时有效的服务。

我们的GRE学习系统的前端采用了Android客户端的形式。首先采用两个Activity(Android程序的基本单元)分别作为用户账号的注册和登陆模块，新用户需要注册账户并设置登录密码，老用户则可以直接通过账户和密码登陆，这两个模块都需要想服务器发送请求，验证用户身份。登陆成功后就到了GRE学习系统的主界面，由于我们的APP功能模块采用了树状的结构设计，主界面可以看做树的根节点，其他功能模块都由主界面的入口进入。下面将详细介绍我们APP的前端功能结构设计。

主要功能模块有学习计时器、用户登录密码重置、查看日历、我的单词本、试题库、我的题库等6大模块。

1. 学习计时器

实现了拥有“开始计时”、“暂停计时”、“重置计时器”三个按钮的计时器，用户可以使用计时器来记录自己学习的时间。

1. 用户登录密码重置

用户可以通过该模块重新设置自己的账户登录密码，并通知服务器更新数据库中该用户的登录密码信息。

1. 查看日历

该模块实现了个能够查询日期，规划日程的万年历，便于用户查看GRE考试时间，规划复习日程等。

1. 我的单词本

该模块实现了用户对个人单词本中单词的增、删、改、查功能。主界面是一个包含用户单词本中所有单词的列表，列表的每一项是一个英文单词，包括他的英文拼写以及中文释义。用户可以向单词本中手动添加一个新的单词，也能编辑单词本中原来就有的单词信息，除此之外，用户也可以将单词本中的单词删除。在这些操作后，用户都会将更新后的信息发送给服务器，以更新服务器中的单词本数据。

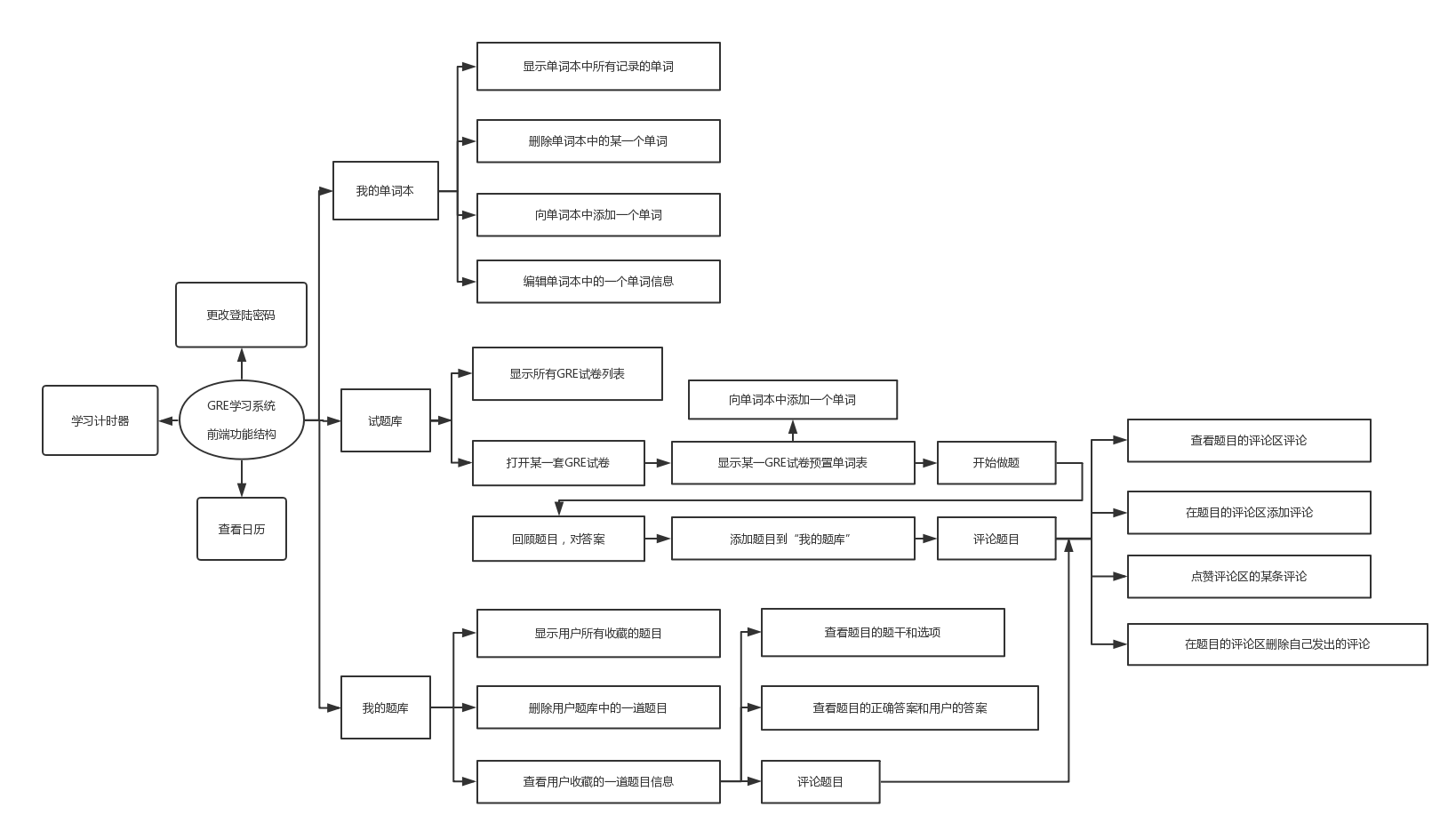
1. 试题库

该模块实现了用户做题的功能，也是本软件的主干功能模块。主界面是一个包含所有GRE考卷的列表，列表的每一项是一张GRE考卷，当用户选择某一张考卷后，就进入了完成一张考卷的流程。该流程具体过程如下：(1)弹出该试卷的预置单词表，让用户先记忆该卷子涉及的GRE重点词汇，这里用户也可以将其中的若干词汇加入“我的单词本”(2)开始进入做题步骤，依次显示该卷子包含的所有题目，每道题目包括题干，题目的选项，用户做出自己的选择后，点击“下一题”继续完成下一题，直到最后一题(3)当用户完成该卷子的全部题目后，会将用户的答案上传到服务器记录到数据库中，并进入对答案、复习阶段(4)在复习阶段可以查看每道题目下的评论区中的评论，并对评论进行增删改查。(5)用户可以在复习阶段将题目收藏，加入”我的题库”中，以便之后复习之需。

1. 我的题库

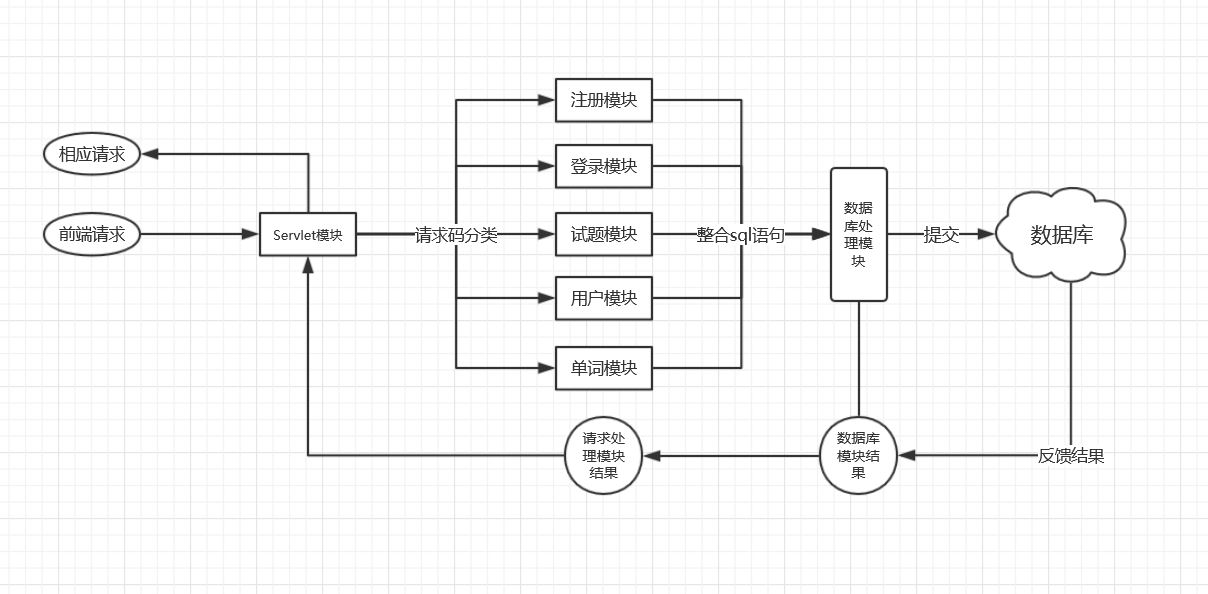
该模块实现了用户回顾收藏的题目的功能。主界面是一个包含所有收藏题目的列表，列表的每一项是一道题目，用户选择一套题目后就可以查看题目的详细信息,包括题干、选项、正确答案、曾经自己的答案、评论区等。用户也可以将该题目从“我的题库”中删除。

前端功能结构图如下图所示:



在后端程序设计中，我们使用的是“面向对象，统一处理”的思想。程序中有三个主要的模块：Servlet服务器模块、请求处理模块、数据库处理模块。整个程序的架构如下：首先主函数位于Servlet模块中，它将以post方法接收请求；接收到这些请求后，它会按照请求的类型将它们下发至不同的请求处理模块。在各个请求处理模块中，我们均预留了连接数据库处理模块的接口，通过分析请求内容，处理模块会生成相应的sql语句，并将它们提交给数据库处理模块进行操作。在数据库处理模块中，程序会连接至数据库并执行sql语句；同时，该模块还将捕获数据库发生的错误（或获得正确的结果），并将其返回至请求处理模块中，而请求处理模块则会通过Servlet模块将得到的信息发送回前端程序。通过这样的设计，后端程序可以很好地处理前端的请求并发送反馈（不论是正确结果还是错误信息），从而使程序正常、鲁棒地运行。

对于后端即收到的前端请求，处理流程可以使用下图表示：



在前后端数据传输部分，我们使用了常用的Http协议POST传输方式。在大多数情况下，数据传输都由客户端发起，将操作请求码和相关的请求信息打包成成JSON格式，并通过POST请求发送给服务器。之后Servlet服务器通过判别请求码分发客户端请求，解析JSON格式的数据信息，接着完成相应的数据处理、数据存储、数据库维护等工作，并将用户请求的数据通过JSON格式打包，最后通过POST报文返还给客户端。客户端在接收到服务器返回的数据后，解析JSON数据并在把所用数据处理后显示到Android界面中。这样，一次完整的前后端数据传输过程就完成了。

1. 数据库基本表的定义

#=============================

#用户表

#属性 用户ID：int USID

# 账号：char[50] USACCOUNT

# 密码：char[20] USPASSWORD

create table US

(

USID int not null unique auto\_increment,

USACCOUNT char(50) not null unique,

USPASSWORD char(20) not null,

primary key(USID)

);

#=============================

#考卷表

#属性 试卷ID：int PAID

# 时间：char[11] PATIME

# 题目编号：int\*10 PAQUES(n)

create table PAPER

(

PAID int not null unique,

PATIME char(11) not null,

PAQUES1 int not null,

PAQUES2 int not null,

PAQUES3 int not null,

PAQUES4 int not null,

PAQUES5 int not null,

PAQUES6 int not null,

PAQUES7 int not null,

PAQUES8 int not null,

PAQUES9 int not null,

PAQUES10 int not null,

primary key(PAID)

);

#================================

#题目类型表

#属性 题目ID：int

create table QUESTYPE

(

QUESID int not null unique,

ttype char(3) not null,

primary key(QUESID)

);

insert into QUESTYPE (QUESID, ttype) VALUES(1, '51');

insert into QUESTYPE (QUESID, ttype) VALUES(2, '51');

insert into QUESTYPE (QUESID, ttype) VALUES(3, '62');

insert into QUESTYPE (QUESID, ttype) VALUES(4, '62');

insert into QUESTYPE (QUESID, ttype) VALUES(5, '62');

insert into QUESTYPE (QUESID, ttype) VALUES(6, '93');

insert into QUESTYPE (QUESID, ttype) VALUES(7, '62');

insert into QUESTYPE (QUESID, ttype) VALUES(8, '62');

insert into QUESTYPE (QUESID, ttype) VALUES(9, '62');

insert into QUESTYPE (QUESID, ttype) VALUES(10, '62');

#==============================

#单词表

#属性 单词ID：int WOID

# 英文：char[20] WOENG

# 中文：char[20] WOCHI

create table WORD

(

WOID int not null unique auto\_increment,

WOENG char(20) not null unique,

WOCHI char(20) not null,

primary key(WOID, WOENG)

);

#==============================

#用户错题表

#属性 用户ID：int USID

# 题目ID：int QUESID

# 题目答案 : char(4) QUESANS

create table USER\_FAULT

(

USID int not null,

QUESID int not null,

QUESANS char(4) not null,

primary key(USID, QUESID)

);

#==============================

#五选一题目表

#属性 题目编号：int QUID

# 题目内容：varchar QUCON

# 五个选项：char[50] QUOPT\*5

# 一个答案: char[2] QUANS

create table QUES51

(

QUID int not null unique,

QUCON varchar(500) not null unique,

QUOPTA char(50) not null,

QUOPTB char(50) not null,

QUOPTC char(50) not null,

QUOPTD char(50) not null,

QUOPTE char(50) not null,

QUANS1 char(2) not null,

primary key(QUID)

);

#==============================

#六选二题目表

#属性 题目编号：int QUID

# 题目内容：char[500] QUCON

# 六个选项：char[50] QUOPT\*6

# 两个答案: char[2] QUANS\*2

create table QUES62

(

QUID int not null unique,

QUCON varchar(500) not null unique,

QUOPTA char(50) not null,

QUOPTB char(50) not null,

QUOPTC char(50) not null,

QUOPTD char(50) not null,

QUOPTE char(50) not null,

QUOPTF char(50) not null,

QUANS1 char(2) not null,

QUANS2 char(2) not null,

primary key(QUID)

);

#==============================

#九选三题目表

#属性 题目编号：int QUID

# 题目内容：char[500] QUCON

# 九个选项：char[50] QUOPT\*6

# 三个答案: char[2] QUANS\*2

create table QUES93

(

QUID int not null unique,

QUCON varchar(500) not null unique,

QUOPTA char(50) not null,

QUOPTB char(50) not null,

QUOPTC char(50) not null,

QUOPTD char(50) not null,

QUOPTE char(50) not null,

QUOPTF char(50) not null,

QUOPTG char(50) not null,

QUOPTH char(50) not null,

QUOPTI char(50) not null,

QUANS1 char(2) not null,

QUANS2 char(2) not null,

QUANS3 char(2) not null,

primary key(QUID)

);

#================================

#评论表

#属性 评论编号 int COID

# 评论时间 char[30] COTIME

# 评论内容 char[300] COCON

# 点赞数量 int CONICE

# 题目编号 int COQUESID

# 发送用户编号 int COFROM

create table COMENT

(

COID int not null unique auto\_increment,

COTIME varchar(30) not null,

COCON varchar(300) not null,

CONICE int not null,

COQUESID int not null,

COFROM int not null,

primary key(COID)

);

#===============================

#考卷-生词表

#属性 考卷编号：int PWPAPERID

# 单词编号：int PWWORDID

create table PAPER\_WORD

(

PWPAPERID int not null,

PWWORDID int not null,

primary key(PWPAPERID, PWWORDID)

);

#===============================

#用户生词表

#属性 用户ID：int USERID

# 单词英文：char[20] UWENG

# 单词中文：char[20] UWCHI

# 记录时间：char[11] UWTIME

create table USER\_WORD

(

USERID int not null,

UWENG varchar(30) not null,

UWCHI varchar(100) not null,

UWTIME varchar(50) not null,

primary key(USERID, UWENG)

);

#================================

#点赞记录表

#属性 用户ID：int NIUSERID

# 评论ID：int NICOMID

# 点赞时间(精确到分钟)：char[17] NITIME

create table NICE

(

NIUSERID int not null,

NICOMID int not null,

NITIME varchar(50) not null,

primary key(NIUSERID, NICOMID)

);

#==================================

#做题痕迹表

#属性 用户ID：int

# 试卷ID：int

# 答案：char[4]

create table USANS

(

USID int not null,

PAID int not null,

ANS1 char(4),

ANS2 char(4),

ANS3 char(4),

ANS4 char(4),

ANS5 char(4),

ANS6 char(4),

ANS7 char(4),

ANS8 char(4),

ANS9 char(4),

ANS10 char(4),

primary key(USID, PAID)

);

1. 触发器的定义与实现

在数据库中，触发器主要维护了数据库的完整性，它使得数据库的一些操作会可以同时发生。我们认为，在一般的系统中，这些系列性的操作其实可以由后端程序进行编写。但是，使用触发器，可以使我们在直接操作数据库时避免错误，同时也可以减少后端的编码量。

经过分析我们发现，本系统中主要需要使用触发器维护“评论”与“点赞”之间的关系，其表现为：删除一条评论的同时需要删除关于这条评论的全部点赞信息，给一个评论点赞后需要将这个评论的点赞数自增。这样的操作在后端程序中可以实现，但使用触发器就显得格外简单，其定义与实现如下：

#================================

# 当评论被删除时，同时删除一切与之相关的点赞信息

create trigger nice\_delete

after delete on COMENT

for each row

delete from NICE where NICOMID=old.COID;

#================================

# 用户点赞时更新评论的点赞数

create trigger nice\_add1

after insert on NICE

for each row

update coment set CONICE = CONICE + 1 where COID=new.NICOMID;

此外，我们还使用了触发器维护卷子、预置单词与用户做题痕迹之间的关系：

#================================

# 删除试卷时同时删除试卷单词关系与所有用户在该卷子上的做题记录

create trigger paper\_word

after delete on paper

for each row begin

delete from paper\_word where paper\_word.PWPAPERID = old.PAID;

delete from USANS where USANS.PAID = old.PAID;

end

值得指出的是，对于每一个系列性的操作组，我们均经过了仔细地分析，从而判断这些操作是应该使用触发器还是应该手动实现。以上的触发器操作均是在分析后得到的结果。

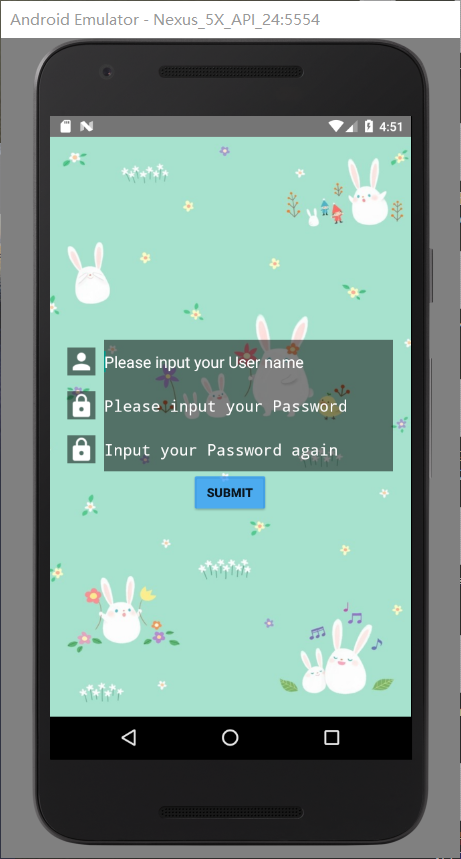
1. 存储过程的定义与实现

对于本系统的存储过程来说，可以先将其分为两类：系统管理者的数据预存储与用户数据的存储。下面从这两个方面进行阐述。

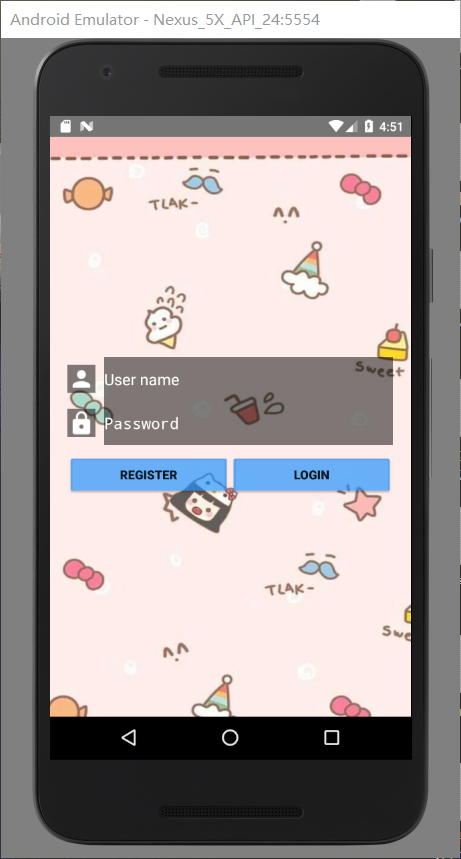
从系统管理者的角度来说，本系统没有在Java脚本中为其留出添加数据的接口，因为我们认为目前整个系统并没有真正投入和使用，预存的数据量并不大，因此相较于使用Java脚本对数据库进行修改，不如直接使用针对Mysql数据库的编辑软件进行修改——在这样的修改过程中，管理者可以通过可视化的复制粘贴方式对数据库直接进行操作，非常方便。当然，如果本系统真正投入使用，也完全可以使用脚本来导入数据——在操作时只需要指明以下信息即可：(1)卷子的发布时间 (2)卷子中的题目内容 (3)一套卷子包含的题目ID (4)一套卷子中包含的预置单词。在创建脚本时，还需要根据题目的类型进行区分，将每一道题目存储到不同的表中即可。以上步骤只需要将需要导入的数据按某种特定格式排列，由脚本读入及分析即可，在实现方面非常简单。

从用户的角度来说，可能涉及到的存储过程只有以下几个：注册时输入的密码、修改密码、做题时提交的答案、自行添加的错题与单词、自己编写的评论、为他人的评论点赞这六个。在后台程序中，我们对这些存储过程都是同样对待的，其体现如下：在DataBase类中，我们定义了insert函数，它可以接受任何sql语句来向数据库中进行插入操作。因此，在后端程序中，只需解析前端发送来的请求，将其翻译为sql语句，而后通过DataBase类提交到数据库中即可。一旦数据库的存储操作发生了问题，后端程序会向前端返回错误报告，通过数据包的某些字段来提示前端所遇到的问题，而后前端将会将此问题反馈给用户。此外，由于数据库基础表中数据项的一些复杂关系，我们还使用了触发器来辅助存储，比如为评论点赞时，不仅会增加用户点赞记录，而且也会为评论表里的点赞数量自增。通过这种方式，我们不仅维护了数据库的完整性约束，而且也减少了后端的代码量。

1. 系统实现结果
2. 注册界面



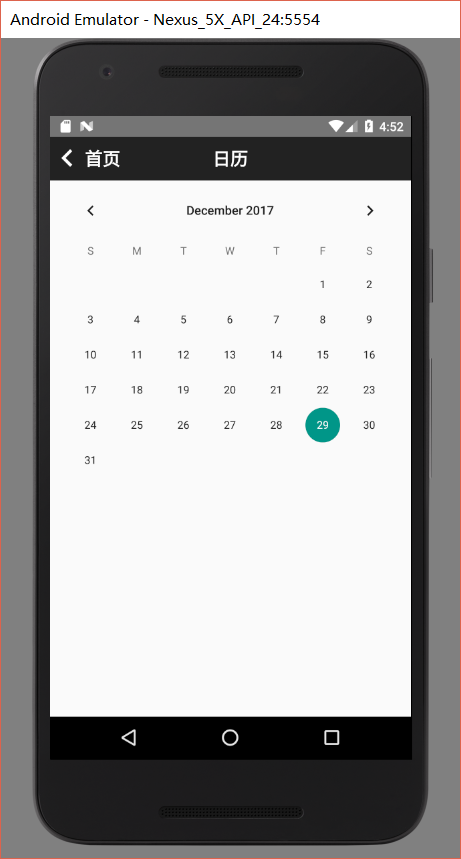
1. 登陆界面



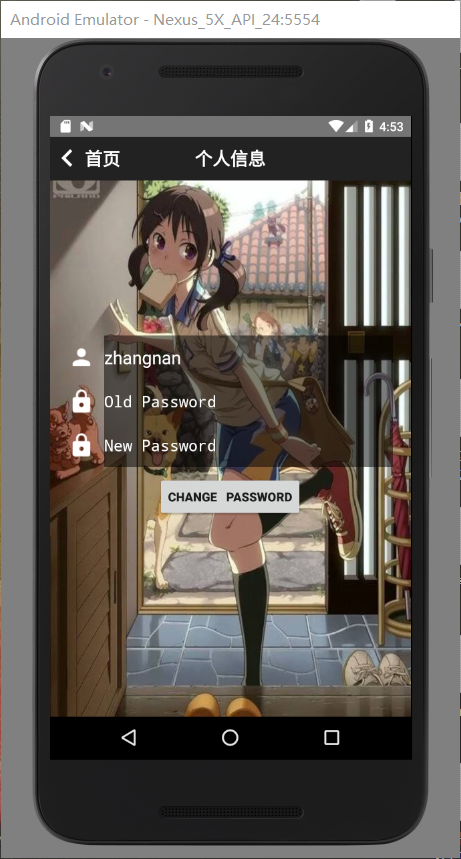
1. 主界面



1. 日历界面



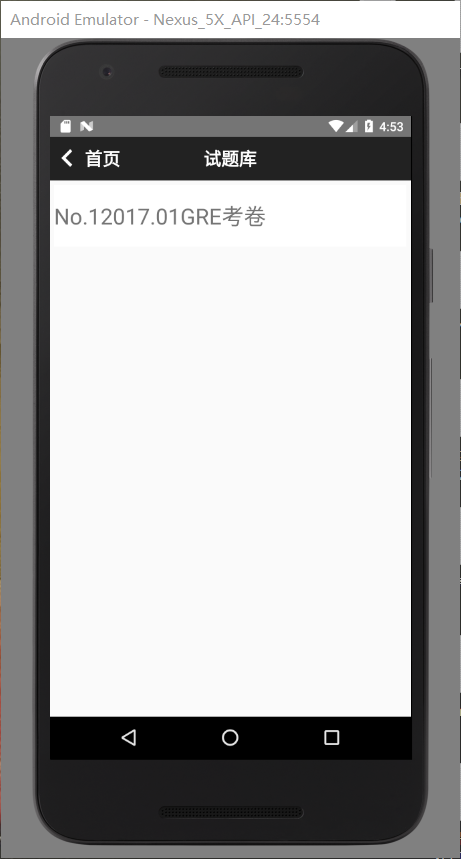
1. 更改登陆密码界面



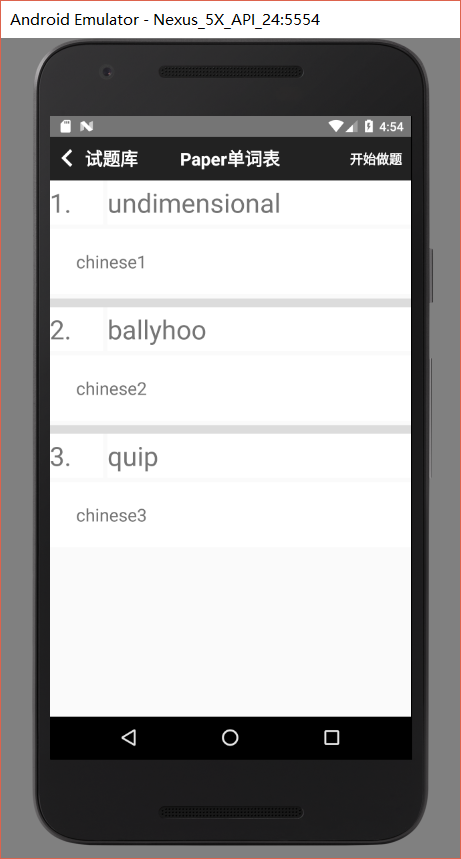
1. 学习计时器



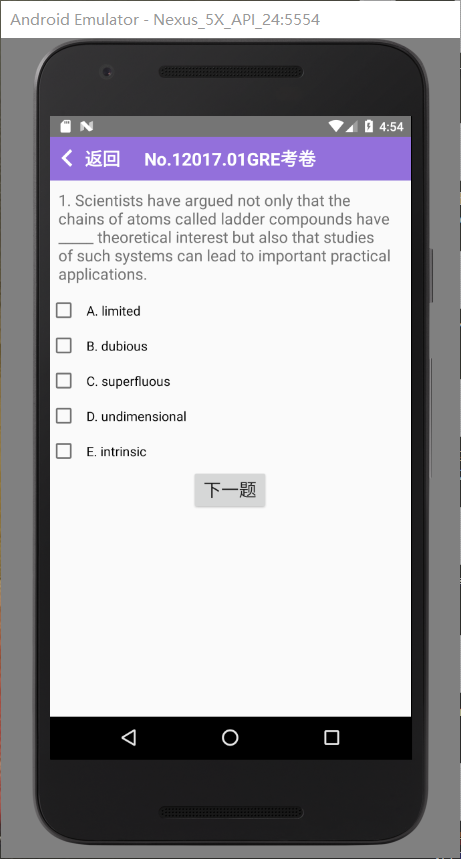
1. 进入试题库



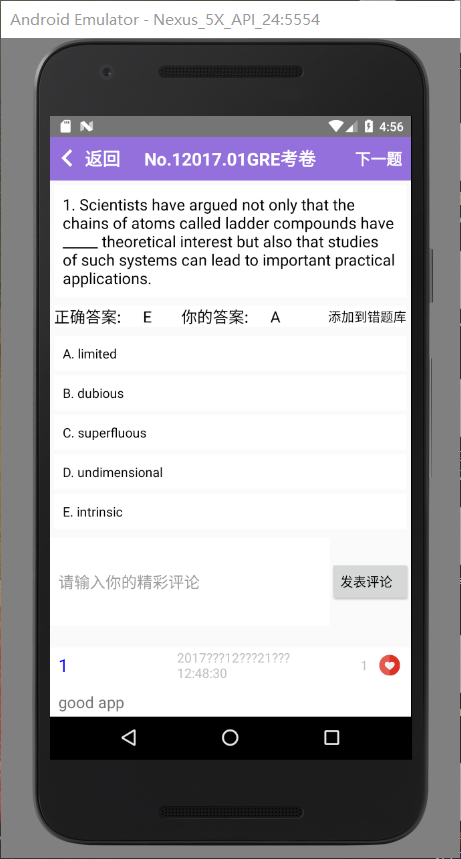
1. 预置单词表



1. 开始做题



1. 对答案，复习



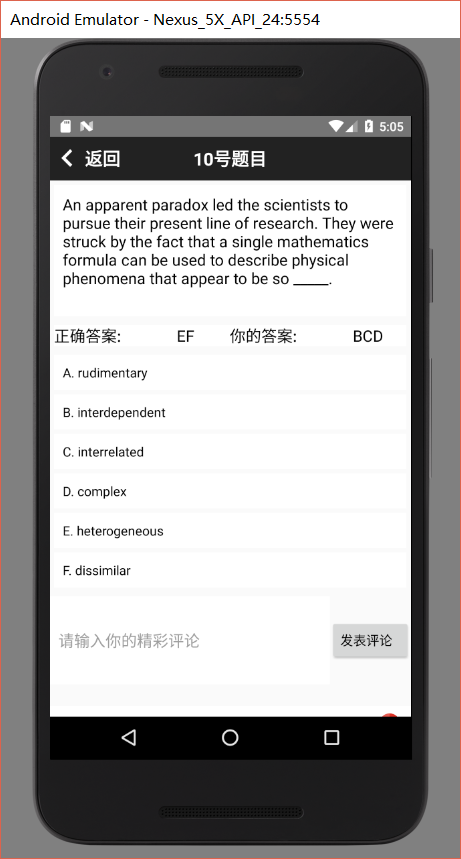
1. 评论区



1. 我的题库和从我的题库中删除题目



1. 查看题库中的题目

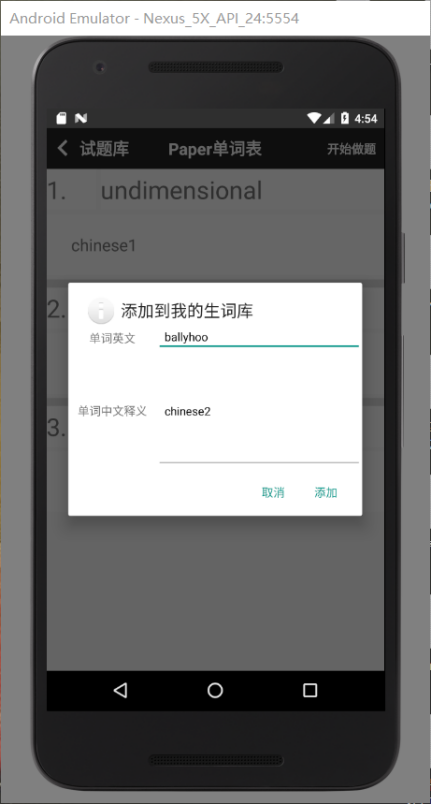


1. 我的单词本



1. 添加单词和删除单词





1. 总结

经过一学期的数据库课程的学习之后，我们不仅从理论上学习到了一个关系数据库的构成与设计，还亲自动手，完成了一个中等规模的关系数据库应用，收益颇多。在完成这次数据库课程设计后，我们真正认识到了一个关系数据库不仅仅是一个提供数据来源的简单数据文件，他更像一个操作系统，能够管理数据、调度数据并维护数据的完整性。一个优秀的应用不仅仅前端要设计的好，后端数据库的管理也是十分重要的，二者缺一不可。