



毕业设计（论文）

中文题目：数据库教学网站建设

英文题目：Database CAI Site

学 院：_____

专 业：_____

学生姓名：_____

学 号：_____

指导教师：_____

____2007____年____6____月____10____日

题 目： 数据库原理课程教学网站建设

适合专业： 计算机科学与技术

指导教师（签名）： _____ 提交日期： 2007 年 3 月 16 日

毕业设计（论文）基本内容和要求：

数据库原理课程教学网站建设内容是建成一个可以供学生和老师使用的网上教学平台，拥有课程介绍，在线学习，课件下载，论坛，在线答题，作业处理等功能。该项目由两名同学共同完成。梅丹同学承担的是在线答题模块和作业处理模块两个主要部分。

其中在线答题模块要求同学通过用户名和密码在网站在线答题（主要是选择题），提交后能够立刻得到分数并看到正确答案。教师可以以管理员身份建立题库，插入题库，编辑题库和删除题库。

作业处理模块要求同学能够通过用户名和密码在网站即时看到作业的说明和要求，在线提交作业。教师可以以管理员身份编辑作业题目和修改作业内容，并能够下载学生的作业。教师离线评阅学生的作业后可以在线对学生作业评分。每轮作业结束后网站可以显示正确答案。

作业系统要求有学生的成绩记录并最后给出总成绩。

毕业设计（论文）重点研究的问题：

1. 学生和老师身份的认证。
2. 系统拥有较高的安全性能。
3. 选择题自动评分。
4. 作业的上传和下载。

毕业设计（论文）应完成的工作：

1. 提交能够实现内容和要求所述的全部功能的 Web 应用包。
2. 提交程序全部源码。
3. 完成功能测试。
4. 提交相应技术文档。
5. 毕业论文。

参考资料推荐：

《数据库原理及 Oracle 应用》冯凤娟 清华大学出版社 2006

《SQL 与 PL/SQL 程序设计基础》冯凤娟 清华大学出版社 2002

Dave Thomas, Chad Fowler, Andy Hunt. 《Programming Ruby: The Pragmatic Programmers' Guide, Second Edition》Pragmatic Programmers, LLC 2004

Dave Thomas, David Heinemeier Hansson. 《Agile Web Development with Rails》Pragmatic Programmers, LLC 2005

其他要说明的问题：

无

题 目： 数据库原理课程教学网站建设

学院： 计算机 专业： 计科 学生姓名： ?? 学号： ×××××

文献综述：

通过阅读相关文献并查阅相关资料，本人对毕业设计的内容和要求有了充分理解和认识。目前国内外均有相关功能的网站，大部分是用 php, asp, jsp 等语言实现的，且都为拥有自主知识产权的收费软件。这些同类产品有着高收费，难开发，难维护等特点。不适于本校自己使用。

开发拥有自主知识产权的数据库原理教学网站对我校有着重要意义。目前我校软件学院用的教学网站是 Carnegie Mellon 大学的收费网站，为实现此网站的功能我校需要每年向该大学支付高额的使用费用。本数据库原理课程教学网站的建成可以为我校节省大量资金。又因是我校自主开发的产品所以可以应老师要求随时对网站内容和功能进行修改。同时本教学网站在建设时就考虑到了通用性和扩展性。在条件允许的情况下可以稍加修改变成商业网站，为我校创造利润。

本人在数据库原理课程教学网站项目中主要承担在线答题模块和作业处理模块两个主要部分，要研究并实现如下功能：

其中在线答题模块题要求同学通过用户名和密码在网站在线答题（主要是选择题），提交后能够立刻得到分数并看到正确答案。教师可以

以管理员身份建立题库，插入题库，编辑题库和删除题库。

作业处理模块要求同学能够通过用户名和密码在网站即时看到作业的说明和要求，在线提交作业。教师可以以管理员身份编辑作业题目和修改作业内容，并能够下载学生的作业。教师离线评阅学生的作业后可以在线对学生作业评分。每轮作业结束后网站可以显示正确答案。

经查阅资料及分析，认为 Ruby on Rails + MySQL 的框架有着开源，功能强大，开发速度快，适于动态网站建设等适合实现项目要求的功能，故计划用该框架进行项目开发。

主要参考文献:

冯凤娟.《数据库原理及 Oracle 应用》清华大学出版社 2006

冯凤娟.《SQL 与 PL/SQL 程序设计基础》清华大学出版社 2002

Dave Thomas, Chad Fowler, Andy Hunt. 《Programming Ruby: The Pragmatic Programmers' Guide, Second Edition》Pragmatic Programmers, LLC 2004

Dave Thomas, David Heinemeier Hansson. 《Agile Web Development with Rails》Pragmatic Programmers, LLC 2005

Ruby Programming Language <http://www.ruby-lang.org/>

Ruby in Twenty Minutes

<http://www.ruby-lang.org/en/documentation/quickstart/>

Ruby on Rails <http://www.rubyonrails.org/>

研究方案:

数据库用 MySQL，网站用 Ruby on Rails 框架进行项目开发。

计划研究方案:

根据要求建立数据字典。

然后建立数据库。

用 Ruby on Rails 框架完成网页全部功能。

对网站的功能及安全进行测试。

因为要实现同学通过在线答题，提交后能够立刻得到分数并看到正确答案及教师建立题库，插入题库，编辑题库和删除题库功能需要稳定的数据库支持，所以数据库的设计是需要研究的主要问题。另一方面由于需要同学有上传作业和老师下载并批改作业的功能，在文件存储上需要做一定的研究以确定作业文件在服务器端的格式和状态。

毕业设计（论文）进度安排：			
序号	毕业设计（论文）各阶段内容	时间安排	备注
1	准备资料	4.1~4.9	
2	建立数据库	4.10~4.19	
3	建立网站	4.20~5.14	
4	测试	5.15~5.19	
5	撰写论文	5.20~6.10	

指导教师意见：

该毕业设计需要完成整个教学网站的在线答题模块和作业处理模块两个主要部分两部分，有较大的工作量。该开题报告中技术路线清晰，计划进度合理，有一定的参考价值和实用价值。同意该开题报告。

指导教师签名：_____ 审核日期：_____年____月____日

中文摘要

数据库原理课程教学网站建设内容是建成一个可以供学生和老师使用的网上教学平台，拥有课程介绍，在线学习，课件下载，论坛，在线答题，作业处理等功能。本人在数据库原理课程教学网站项目中主要承担在线答题模块和作业处理模块两个主要部分，要研究并实现在线答题功能和作业处理功能。其中在线答题功能要求同学通过用户名和密码在网站在线答题（主要是选择题），提交后能够立刻得到分数并看到正确答案。教师可以以管理员身份建立题库，插入题库，编辑题库和删除题库。作业处理功能要求同学能够通过用户名和密码在网站即时看到作业的说明和要求，在线提交作业。教师可以以管理员身份编辑作业题目和修改作业内容，并能够下载学生的作业。教师离线评阅学生的作业后可以在线对学生作业评分。每轮作业结束后网站可以显示正确答案。整个项目基于 Ruby On Rails 架构实现，数据库用 MySQL。开发拥有自主产权的数据库原理教学网站对我校有着重要意义。目前我校软件学院用的教学网站是 Carnegie Mellon 大学的收费网站，为实现此网站的功能我校需要每年向该大学支付高额的使用费用。本数据库原理课程教学网站的建成可以为我校节省大量资金。又因是我校自主开发的产品所以可以应老师要求随时对网站内容和功能进行修改。同时本教学网站在建设时就考虑到了通用性和扩展性。在条件允许的情况下可以稍加修改变成商业网站，为我校创造利润。

关键词：

数据库教学 网站 Ruby On Rails MySQL

Abstract

Database CAI SITE is web site which can be used on CAI of Database education. It has a lot of function, for e.g. On Line Studying, lecture downloading, forum, online multiple choices test, as well as homework upload and check.

I did the model of online multiple choices and homework upload & check. These two models both supply role control which could recognize administrator and student. And a administrator could edit quizzes, categories, classes, database class and also the a student's information. As a administrator, one could also change the student's password. As a student, one could do multiple choices and upload one's homework. If a student has upload his homework, the teacher could download his home work and give a mark.

All Project are Based on Ruby on Rails and MySQL.

It is general acknowledgement that our university is using the Carnegie Mellon web site for database education and it costs a lot. As long as our project Database CAI SITE released, we can say good bye to Carnegie Mellon. And Database CAI SITE could also be a comical product.

Key words:

Database CAI web site Ruby on Rails MySQL

目录

第一章	项目介绍	12
1.1	项目背景	12
1.2	预期实现功能	12
1.2.1	在线答题功能	12
1.2.2	作业处理功能	12
第二章	项目架构及开发平台介绍	13
2.1	Ruby 语言介绍	13
2.1.1	Ruby 概述	13
2.1.2	Ruby 的特长	13
2.2	Ruby On Rails 的介绍	15
2.2.1	Rails 是敏捷的	15
2.2.2	Rails 的 MVC	16
2.2.3	生成代码	17
2.3	MySQL 介绍	18
2.3.1	MySQL 是一个数据库管理系统	18
2.3.2	MySQL 是一个关系数据库管理系统	18
2.3.3	MySQL 是开源的	18
2.3.4	MySQL 是快速的、可靠的和易于使用的	19
2.3.5	MySQL 在客户/服务器或嵌入系统中的应用	19
2.3.6	有大量的 MySQL 软件可以使用	19
2.4	开发平台简介	19
2.5	Intellij IDEA	20
2.5.1	Ruby Plug-in	22
2.5.2	Navicat SQL 简介	24
第三章	需求分析	26
3.1	任务概述	26
3.2	功能需求	26
3.2.1	在线答题模块	27
3.2.2	作业处理模块	27
3.2.3	学生管理模块	28
第四章	数据库设计	29
4.1	数据库表结构	29
4.1.1	管理员列表（administrators）	29
4.1.2	学生列表（students）	29
4.1.3	章节列表（categories）	30

4.1.4	选择题列表（choices）	31
4.1.5	作业的题目及答案列表（quizzes）	32
4.1.6	学生作业列表（homeworks）	32
4.1.7	学生自然班列表（original_classes）	33
4.1.8	学生年级列表（grades）	34
4.1.9	数据库班级列表（dbclasses）	34
4.2	数据库表间关系	35
第五章	功能模块的设计和实现	37
5.1	在线答题模块	38
5.1.1	教师端	38
5.1.2	学生端	53
5.2	作业提交模块	58
5.2.1	教师出题功能	59
5.2.2	学生做题功能	61
5.2.3	学生更新作业功能	65
5.2.4	教师批改作业功能	67
5.2.5	学生查看作业批改结果功能	70
5.3	学生管理模块	71
5.3.1	编辑学生基本情况	72
5.3.2	编辑数据库班	73
5.3.3	编辑自然班	73
5.3.4	编辑年级	74
5.3.5	查看成绩卡	75
5.3.6	计算平均成绩	77
第六章	技术难点	79
6.1	触发器	79
6.1.1	触发器功能描述	79
6.1.2	触发器实现方法	79
6.2	设置题目和答案显示时间功能	81
6.2.1	设置题目和答案显示时间功能描述	81
6.2.2	设置题目和答案显示时间实现方法	85
6.3	访问控制	86
6.3.1	访问控制功能描述	86
6.3.2	访问控制实现方法	86
6.4	选择题的特定答案顺序	90
6.4.1	选择题的特定答案顺序功能描述	90
第七章	项目前景	92

第一章 项目介绍

数据库原理课程教学网站建设内容是建成一个可以供学生和老师使用的网上教学平台，拥有课程介绍，在线学习，课件下载，论坛，在线答题，作业处理等功能。整个项目计划成为我校数据库教改项目之一，由两名同学进行网站开发。

1.1 项目背景

开发拥有自主知识产权的数据库原理教学网站对我校有着重要意义。目前我校软件学院用的教学网站是 Carnegie Mellon 大学的收费网站，为实现此网站的功能我校需要每年向该大学支付高额的使用费用。本数据库原理课程教学网站的建成可以为我校节省大量资金。又因是我校自主开发的产品所以可以应老师要求随时对网站内容和功能进行修改。同时本教学网站在建设时就考虑到了通用性和扩展性。在条件允许的情况下可以稍加修改变成商业网站，为我校创造利润。

1.2 预期实现功能

本人在数据库原理课程教学网站项目中主要承担在线答题模块和作业处理模块两个主要部分，要研究并实现如下功能：

1.2.1 在线答题功能

其中在线答题功能要求同学通过用户名和密码在网站在线答题（主要是选择题），提交后能够立刻得到分数并看到正确答案。教师可以以管理员身份建立题库，插入题库，编辑题库和删除题库。

1.2.2 作业处理功能

作业处理功能要求同学能够通过用户名和密码在网站即时看到作业的说明和要求，在线提交作业。教师可以以管理员身份编辑作业题目和修改作业内容，并能够下载学生的作业。教师离线评阅学生的作业后可以在线对学生作业评分。每轮作业结束后网站可以显示正确答案。

第二章 项目架构及开发平台介绍

2.1 Ruby 语言介绍

2.1.1 Ruby 概述

Ruby，一种为简单快捷物件导向编程（面向对象程序设计）而创的脚本语言，由日本人松本行弘（まつもとゆきひろ）（英译：Yukihiro Matsumoto）开发，并且是一款遵守 GPL 协议和 Ruby License 的免费软件。。

Ruby 是一种功能强大的面向对象的脚本语言，她可以使您方便快捷地进行面向对象编程。有时使用像 Smalltalk、Eiffel 或 C++ 这样正式的面向对象语言来开发一些小项目显得有点"小题大做"，而 Ruby 刚好可以满足这些面向对象编程的需求。当然，开发者也可以使用 Ruby 进行普通的面向过程编程。

Ruby 的文本处理能力极强，与 Perl 不分伯仲。同时，Ruby 的语法简单，还有异常处理以及迭代器等构造，使编程变得简单明了。简而言之，开发着可以像使用 Perl 一样方便快捷地进行面向对象的开发。

Ruby 是由松本行弘开发的免费软件。

2.1.2 Ruby 的特长

作为一种比较新的脚本语言，Ruby 有着如下特长：
解释器

Ruby 是解释型语言，因此执行程序时无需编译。

变量无类型(动态地确定类型)

Ruby 的变量可以接收各种类型的数据，因此没有必要担心变量类型的问题。另一方面，这弱化了编译检查的功能。

无需声明变量

Ruby 中无需声明变量即可使用。可以根据变量名分辨变量的种类(局部变量，全局变量，实例变量等)。

语法简单

受 Eiffel 影响，Ruby 的语法十分简单。

内存管理无需用户干预

Ruby 自动进行内存管理。解释器内置的垃圾回收器会自动回收不再使用的对象。

一切都是对象

从一开始 Ruby 就被设计成为纯粹的面向对象语言。包括整数这种基本的数据类型在内所有数据都被看是对象，进而得到统一的处理。

类，继承，方法

Ruby 当然具有面向对象语言的基本功能，包括类，继承和方法等。

特殊方法

可向某对象添加方法。例如，可以把 GUI 按钮被按下时的动作作为方法记述下来，还可以用它来进行原型库（prototype base）的面向对象编程（只要您想这样的话）。

使用模块进行糅合（Mix-in）

Ruby 认为多重继承会导致问题复杂化，因此故意舍弃了多重继承，但可以使用模块超越类的界限来共享数据和方法等。这就是“Mix-in”糅合功能

迭代器

迭代器功能可以将流程控制结构抽象化。

闭包

可以将某过程片段对象化，对象化后的该过程片段就称作闭包。

功能强大的字符串操作 / 正则表达式

以 Perl 为样板创造出了功能强大的字符串操作和正则表达式检索功能。

超长整数

内置了处理超长整数的功能，所以只要内存允许就可以计算非常大的整数。例如计算 400 的阶乘等也轻而易举。

异常处理功能

异常处理功能可以使您编写代码处理异常情况。

可以直接访问 OS

Ruby 可以使用（UNIX 的）绝大部分的系统调用。即使单独使用 Ruby 也可以进行系统编程。

动态加载

若 OS 支持的话，可以在运行时读入对象文件。

2.2 Ruby On Rails 的介绍

Ruby on Rails, 也称 RoR 或简称 Rails, 是一个使用 Ruby 语言写的开源网络应用框架, 它是严格按照 Model-View-Controller (MVC) 结构开发的。它努力使自身保持简单, 来使实际的应用开发时的代码更少, 使用最少的配置。

Rails 的设计由一组关键的概念来驱动着: “不要重复自己” (Don't Repeat Yourself) 和 “约定优于配置” (Convention Over Configuration)。

DRV—系统内每个部分应该只在一个地方被表达。Rails 使用强大的 Ruby 开始自己的生命。你会发现在 Rails 应用程序中少有重复; 你在一个地方说你需要什么—这个地方通常由 MVC 体系来暗示。

配置约定也是至关重要的。它意味着 Rails 能判断出你的应用程序交织在一起的每个部分的缺省值。下面是约定, 并且你可以用比使用 XML 配置的, 典型的 Java Web 应用程序更少的代码来写一个 Rails 应用程序。如果你想覆写这些约定, Rails 也可很容易地做到。

此外 Rails 还包括完整的支持 Web 服务的材料, 接受邮件, AJAX(高级交互式 Web 应用程序), 完整的单元测试框架(包括对 mock 对象的透明支持), 和对开发, 测试, 生产环境的隔离。或者我们谈论 Rails 具有的代码产生器。它们产生 Ruby 代码框架, 然后由你填充应用程序的逻辑部分。

最后, Rails 的区别是源于它的起源—Rails 是被精选出来的商业应用程序。它倾向于创建一个框架的最好途径是找到特定应用程序的中心主题, 然后在平常的代码基础内使用它们。当你开发你的 Rails 应用程序时, 你是在现有的一个很不错的应用程序上开始的。

2.2.1 Rails 是敏捷的

Rails 的所有部分都是单独的和可交互的。没有笨重的工具集, 没有复杂的配置, 也没有难懂的程序。只有少数开发者, 它们喜爱的编辑器, 和一组 Ruby 代码。这会产生这样的透明度, 开发者做的事情会立即被反射, 并让用户看到所做的事情。这是根本的交互式过程。

Rails 的文档无可指责。Rails 使它可轻易地为你代码创建 HTML 文档。但是 Rails 开发处理并不受文档驱动。在 Rails 的工程中，你几乎找不到 500 页以上说明书。相反，你会发现一组用户和开发者会共同地探究它们的需要，以及对这个需要的可能的处理方式。你也会发现开发者和用户会变得对解决它们曾试着解决的问题更有经验。你可以在开发周期内找到早期的软件框架。这个软件可能粗糙但实用，它让用户一接触它就知道你拿来的是什么。

2.2.2 Rails 的 MVC

了解 Rails，首先需要理解的是它的模型/视图/控制器（model/view/controller, MVC）架构。虽然这种技术不是 Rails 所特有的——甚至不是 Web 应用程序所特有的（相对于其他程序），但是 Rails 具有非常清晰而专一的 MVC 思维方式。如果您并不使用 MVC 方法，那么 Rails 的用处将大为降低（与遵循其模式的情况相比）。

模型

Rails 应用程序的模型部分主要是它所使用的底层数据库。实际上，在很多情形中 Rails 应用程序正是以一种受管理的方式对关系型数据库管理系统（RDBMS）中的数据执行操作的一个途径。

ActiveRecord 类是 Rails 的一个核心组成部分，它将关系型表映射为 Ruby 对象，使其成为控制器可以操作并能在视图中显示的数据。Rails 应用程序特别倾向于使用广为应用的 MySQL 数据库，不过也有与很多其他 RDBMS 的绑定，比如 IBM DB2。

如果您愿意，您可以添加 Ruby 代码来在应用程序模型中执行额外的验证，加强数据关联，或者触发其他操作。应用程序的 `app/models/` 目录中的 Ruby 文件能够调用 ActiveRecord 的多种验证方法。不过，您也可以将模型代码留作一个存根，而只是依赖保存数据的 RDBMS 的约束。

控制器

控制器以其抽象形式执行应用程序的逻辑。也就是说，应用程序的 `app/controllers/` 目录中的 Ruby 脚本能把模型数据导入为变量，保存回去，或对其进行修改和处理。不过，控制器不关心用户如何适当地显示或者输入数据。在通常的 MVC 模型中，这可以让用户能够以多种方式与同一控制器进行交互：本地 GUI，Web 界面，以及视力较弱的人使用的语音界面都可以与相同的控制器进行交互。

不过，Rails 不像那样非常通用；相反，它仅局限于在 Web 页中提供和收集

数据。虽然如此，但是您可以修改那些 Web 页的布局 —— 颜色、字体、表格、样式表单，等等 —— 与控制器代码无关。

视图

Rails 视图是我们编写 Ruby 代码的地方。Rails 包含有一门用于 .rhtml 的非常好的模板语言，它将纯粹的 HTML 与嵌入的 Ruby 代码组合起来。Rails 应用程序界面的表层外观通常是由 CSS 样式表单控制的。.rhtml 格式是一种增强的 HTML。实际上，一个简单的 HTML 文件本身也是一个合法的 RHTML 模板，不过，不应该忽略 RHTML 为您提供的脚本控制。

RHTML 是真正的模板格式 —— 不仅是在 HTML 中嵌入代码的方式 —— 这是一种更为有效的方法。如果您熟悉 PHP，那么可以考虑 PHP 本身与 Smarty 模板之间的对照。也就是说，嵌入的脚本只是将代码与未被解释的 HTML 混合在一起；当需要向客户机输出某些内容时，代码部分仍要负责执行 print 语句。

与之不同的是，模板引擎向 HTML 添加了一组定制的标签，让您能够将条件、循环以及其他逻辑作为增强的 HTML 标记的一部分来表示。

2.2.3 生成代码

Rails 所提供的工具主要是一组代码生成器。相对于那些强迫我使用严格的工作空间和 IDE 的开发环境，我更喜欢这种方法。Rails 不会妨碍您，但是却会为您省去大部分手工编程的工作 —— 或者，通过提供“可自由获得的”初步（first-pass）支架（scaffolding），至少帮助您轻松将需要手工编码的工作分为多个部分。

支架的概念是 Rails 中的核心概念。非常简单的应用程序可能完全不用编码，让 Rails 在运行时动态地生成客户机 HTML 页面。第一遍生成代码时创建的只是粗略的支架；接下来您可以生成更详细的能够定制的控制器的视图和模型。不过在开始时不需要生成太多。

Rails 对其文件的组织是固定的而且非常普通的，不过这种组织相对严格。如果您试图强行使用其他文件和代码组织方式，那么您可能得付出努力去修改 Rails 环境。再者说，我找不到不使用 Rails 所提供的组织方式的理由；在大部分情况下，它“fits your brain”（Ruby 的支持者喜欢这样讲）。例如，如果您从头开始设计一个框架（至少如果您以“Ruby 方式”思考），那么这些目录名称及其组织可能与您的选择非常接近。

2.3 MySQL 介绍

MySQL 是最受欢迎的开源 SQL 数据库管理系统,它由 MySQL AB 开发、发布和支持。MySQL AB 是一家基于 MySQL 开发人员的商业公司,它是一家使用了一种成功的商业模式来结合开源价值和方法论的第二代开源公司。MySQL 是 MySQL AB 的注册商标。

MySQL 是一个快速的、多线程、多用户和健壮的 SQL 数据库服务器。MySQL 服务器支持关键任务、重负载生产系统的使用,也可以将它嵌入到一个大配置(mass-deployed)的软件中去。MySQL 网站(<http://www.mysql.com>)提供了关于 MySQL 和 MySQL AB 的最新的消息。

2.3.1 MySQL 是一个数据库管理系统

一个数据库是一个结构化的数据集合。它可以是从一个简单的销售表到一个美术馆、或者一个社团网络的庞大的信息集合。如果要添加、访问和处理存储在一个计算机数据库中的数据,你就需要一个像 MySQL 这样的数据库管理系统。从计算机可以很好的处理大量的数据以来,数据库管理系统就在计算机处理中和独立应用程序或其他部分应用程序一样扮演着一个重要的角色。

2.3.2 MySQL 是一个关系数据库管理系统

关系数据库把数据存放在分立的表格中,这比把所有数据存放在一个大仓库中要好得多,这样做将增加你的速度和灵活性。“MySQL”中的 SQL 代表“Structured Query Language”(结构化查询语言)。SQL 是用于访问数据库的最通用的标准语言,它是由 ANSI/ISO 定义的 SQL 标准。SQL 标准发展自 1986 年以来,已经存在多个版本:SQL-86, SQL-92, SQL:1999, SQL:2003, 其中 SQL:2003 是该标准的当前版本。

2.3.3 MySQL 是开源的

开源意味着任何人都可以使用和修改该软件,任何人都可以从 Internet 上下载和使用 MySQL 而不需要支付任何费用。如果你愿意,你可以研究其源代码,并根据你的需要修改它。MySQL 使用 GPL(GNU General Public License, 通用公共许可),在 <http://www.fsf.org/licenses> 中定义了你在不同的场合对软件

可以或不可以做什么。如果你觉得 GPL 不爽或者想把 MySQL 的源代码集成到一个商业应用中去，你可以向 MySQL AB 购买一个商业许可版本。

2.3.4 MySQL 是快速的、可靠的和易于使用的

如果这是你正在寻找的，你可以试一试。MySQL 服务器还包含了一个由用户紧密合作开发的实用特性集。你可以在 MySQL AB 的 <http://www.mysql.com/it-resources/benchmarks/> 上找到 MySQL 服务器和其他数据库管理系统的性能比较。

MySQL 服务器原本就是开发比已存在的数据库更快的用于处理大的数据库的解决方案，并且已经成功用于高苛刻生产环境多年。尽管 MySQL 仍在开发中，但它已经提供一个丰富和极其有用的功能集。它的连接性、速度 and 安全性使 MySQL 非常适合访问在 Internet 上的数据库。

2.3.5 MySQL 在客户/服务器或嵌入系统中的应用

MySQL 数据库服务器是一个客户/服务器系统，它由多线程 SQL 服务器组成，支持不同的后端、多个不同的客户程序和库、管理工具和广泛的应用程序接口(APIs)。

MySQL 也可以是一个嵌入的多线程库，你可以把它连接到你的应用中而得到一个小、快且易于管理的产品。

2.3.6 有大量的 MySQL 软件可以使用

幸运的是，你可以找到你所喜爱的已经支持 MySQL 数据库服务器的软件和语言。

2.4 开发平台简介

整个项目的代码开发全部有 IntelliJ IDEA6 with Ruby plug-in 开发。MySQL 的 GUI Client 选用 Navicat 出品的 Navicat SQL 7。和同类产品比较，IDEA 和 Navicat SQL 都有显著的优势。

2.5 IntelliJ IDEA

IntelliJ IDEA 是一个高智能化的 java IDE。他为开发者提供强大的辅助功能和开发工具，并让开发者在开发过程中体验到真正的乐趣。他有如下令人爱不释手的特點。

图 2.1 所示为 IDEA 工作窗口的截图。

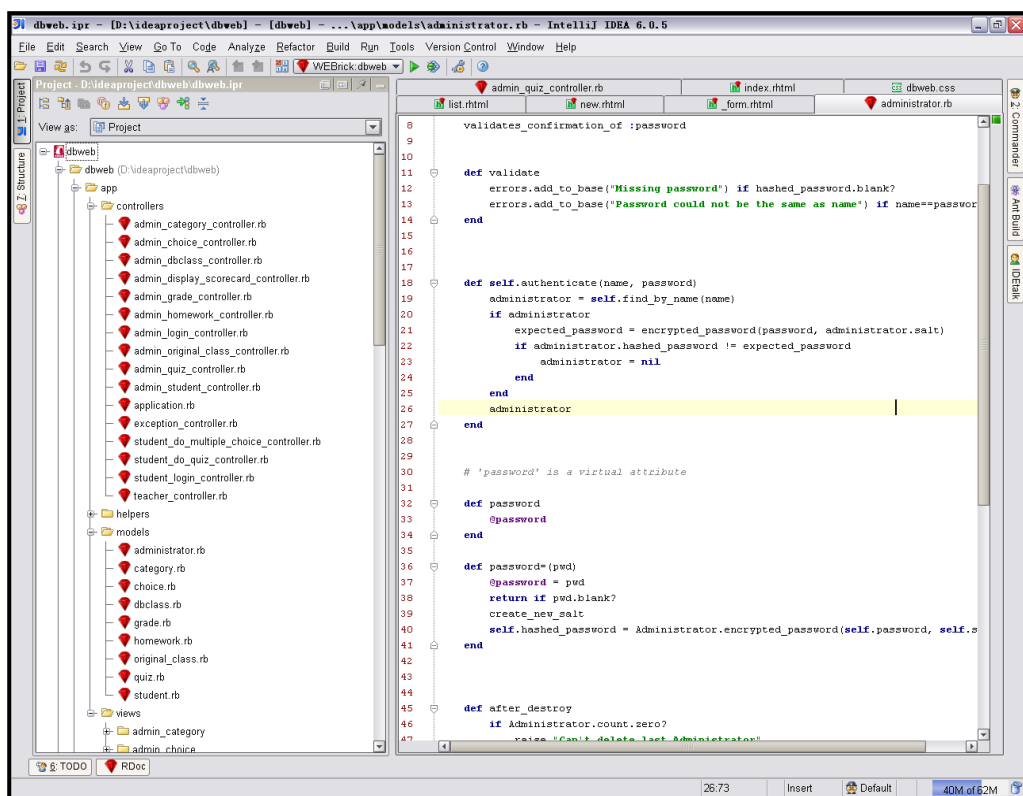


图 2.1 IDEA 工作窗口的截图

1. 感知语法词选择

反复使用 **Ctrl-w** 使所选表达式逐步增大直到选取整个文件。但是这一点听起来并不是很重要，它在与其它特性如“introduce variable” refactoring（“引入变量”重整）相结合使用时才真正好用。我可以把脱字符置于复杂表达式上，按 **Ctrl-w** 直到选取所需范围然后按 **Ctrl-Alt-v** 为表达式引入一个新的变量。如果我要替换这个表达式的其他事件 Idea 甚至也可以自动做到。

2. 多种导航形式

在一个工程里，Idea 有很多种方法用于转换。除鼠标之外所有的命令都可以通过键盘实现（这一点对于像我这样难以忍受不停地切换到鼠标的人来说绝

对不错)。Ctrl-n 可以通过键入类名查找一个类。Ctrl-shift-n 可以查找文件。Ctrl-e 得到最近编辑的文件，通过键入文件名或者鼠标键选择。命令不仅可以容易地在你的工程两个视图里导航并且使整合操作变得轻而易举。Alt-f1 可以是在你在任何一个其他视图里选择当前文件。工程导航地方法太多在这儿不能全部列举。所有命令没必要像观点一样都给出概念。

3. 本地历史

开启本地历史，你源代码中的每一个改变（在 tool 和 externally 里）都被跟踪。这个特性救了我很多次所以我推荐它。不同的地方在一个绝对漂亮的差异查看器里高亮显示出来。

4. 设计优良的整合支持

Idea 支持很多整合功能。更重要的使它们设计的好容易使用。有一些你会一直使用，也有一些几乎用不到。当你发现你需要它们时它们就成了一个大的时间节约器。

5. 代码助手

在 Idea 里有一些真正强大的代码编写助手。在这里我只能介绍给你一点它的好处。一个比较好的例子是“委托方法”助手。它允许你在你的类里面为一个对象实例委托方法调用。它并不是你每天都使用的东西，但当你需要它的时候你就会发现自己节省了大量的时间。另一个例子是产生冗余代码和等价特性。你会发现这是一个自己经常使用的特性。其他一些代码助手特性为：产生 Getters/Setters，产生构造器与“surround with...”（译者注：被什么保护）等特性。

6. 灵活的代码重格式化

关于代码怎样重格式化的选择有许多。它可以为每一个工程单独设置，所以没必要担心自己是否必须支持不同的代码标准。重格式化一个文件或者一个完整的工程眨眼间就可以完成。

7. 直观简洁的 GUI

更为好玩的是每一个程序中的函数都可以通过键盘访问的到。在一个开发工具里这是应该有的但可悲的是大多数 IDE 对键盘导航并不注意。为了完成工作不必要在不同的视图之间切换，也不必要在不同的窗口之间选择了。

8. 与文件系统自动同步

不管你的 IDE 是多么优秀，通常都需要在工具之外作一些工作。Idea 在这一点上做的很好。大多数工程都可以在 Idea 里不用改变他们的结构就可以使用。Idea 还可以检查文件在外部的更改情况。这不需要人工刷新以告诉 Idea 一个文件已被更改。

9. 动态的错误高亮显示

在键码时 Java 代码，XML 与 Java 文档标签被动态解析，错误也会被报告。Ant 建立文件甚至可为 Ant 工程句法提供额外支持。

10. 检查

运行代码检查以报告大量的代码中潜在的错误点。它可以编制成脚本以作为一个批量工作运行，结果也可以以网页的形式提交。

11. 灵巧的编辑功能

一些小技巧如键入匹配的引号及括号使得生活更加容易:)。

12. 几乎没有向导

从前许多厂家都绑定大量的向导而不是设计一个简单易用的接口。虽然有向导但只是一个滥用的工具。Idea 只包含了一个用于帮助创建新工程的向导，并且仍然能够创建一个对新手来讲容易使用的程序。

13. 灵巧的模板

这些灵巧的模板是代码的片断，它们被用来做各种各样令人惊讶的事情。我第一次使用“iterate over collection”模板时就被震惊了。它不仅挑拣出正确的默认收集变量而且还知道我已经放进去了什么以及默认的方法。这个特性很难在纸上描述。下载一个测试版本试验一下吧。

14. 最好的代码实现支持

Idea 现在有三种代码实现支持包括实现默认得变量名称，JavaDoc 以及其他代码细节。

15. 未使用代码高亮显示

如果 Idea 检查出某一个变量或者方法没有被使用，它会把它作为一个不明显的警告显示出来。检查可用来做更详细的死代码分析。甚至 JavaDoc 标签也能被查出来。

16. 有规律的查找与替换表现

规律查找是可能的，但更美妙的事实是你也可以做规律替换。这一点可节约大量的时间。

17. 意识动作

在 Idea 觉察到它可以通过某种方法帮助你时它会提供一些有用地选择。例如你要使用一个表达式使它可以与指定地变量相匹配，Idea 会察觉到这一点然后给你一些选择，要么使用表达式要么改变你在使用地变量地类型。

2.5.1 Ruby Plug-in

专门为 Ruby 编程开发的 Ruby Plug-in 除了拥有 IDEA 的众多优秀功能外还针对 ruby 语言及 Ruby On Rails 架构实现了很多方便的功能。

专门的 Rails 功能视图，自动补全 Ruby 的关键字，Ruby 语言的高亮提示，基于 Ruby 优化的自动排版，即使错误检查和代码校验，为 Ruby 文件优化的智能视图，快速查看 Ruby 文档 (RDoc)，Rails 的脚本语言支持，自动传见 Rails 框架，在视图和控制器 View-Controller 间智能跳转，通过菜单快速访问 Rails

的 generators 脚本和测试功能。

图 2.2 所示为加入 Ruby Plugin 的 IDEA 工程结构截图。

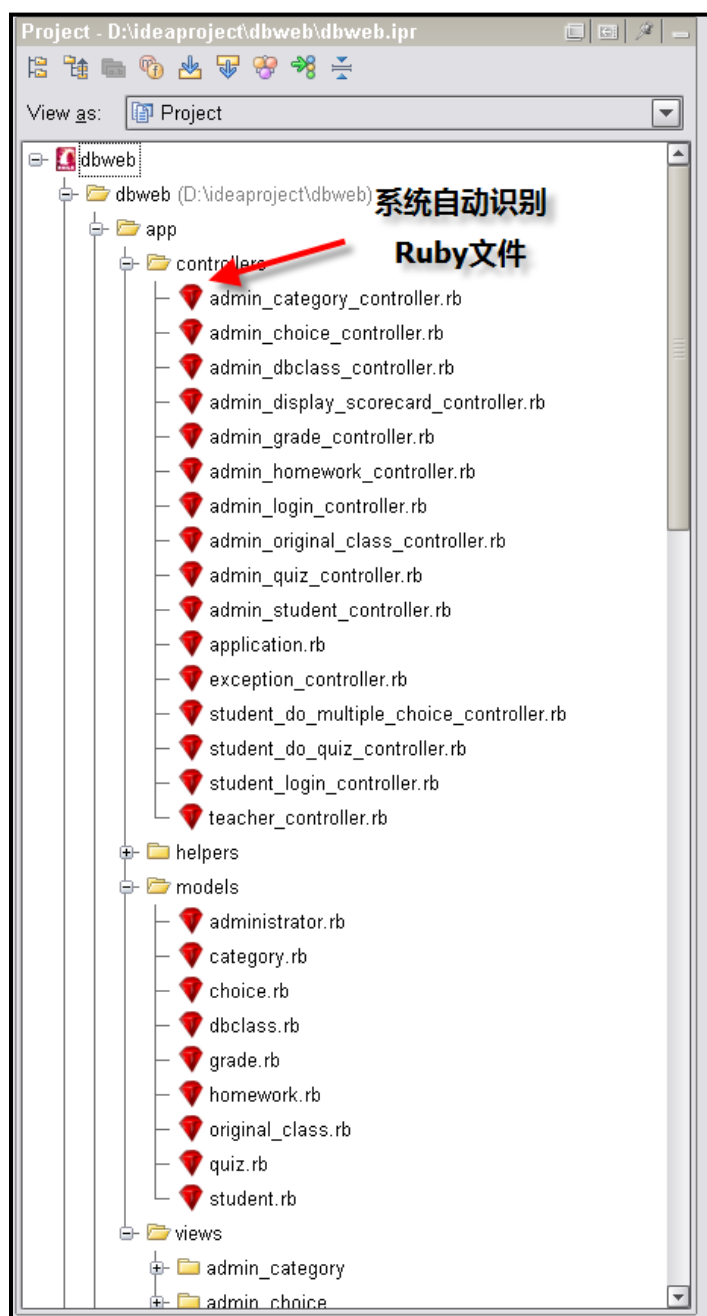


图 2.1 加入 Ruby Plugin 的 IDEA 工程结构截图

2.5.2 Navicat SQL 简介

Navicat SQL 是 PremiumSoft 出品的一套功能强大的 MySQL 数据库系统管理及开发工具。它不仅适合资深的专业研发者，同时适合新手轻松的学习。友善的图形化使用接口，Navicat 可以让您快速且简易的建立、查询、组织、存取，并在安全及方便的环境下共享信息。

图 2.3 所示为 Navicat SQL 的操作界面。

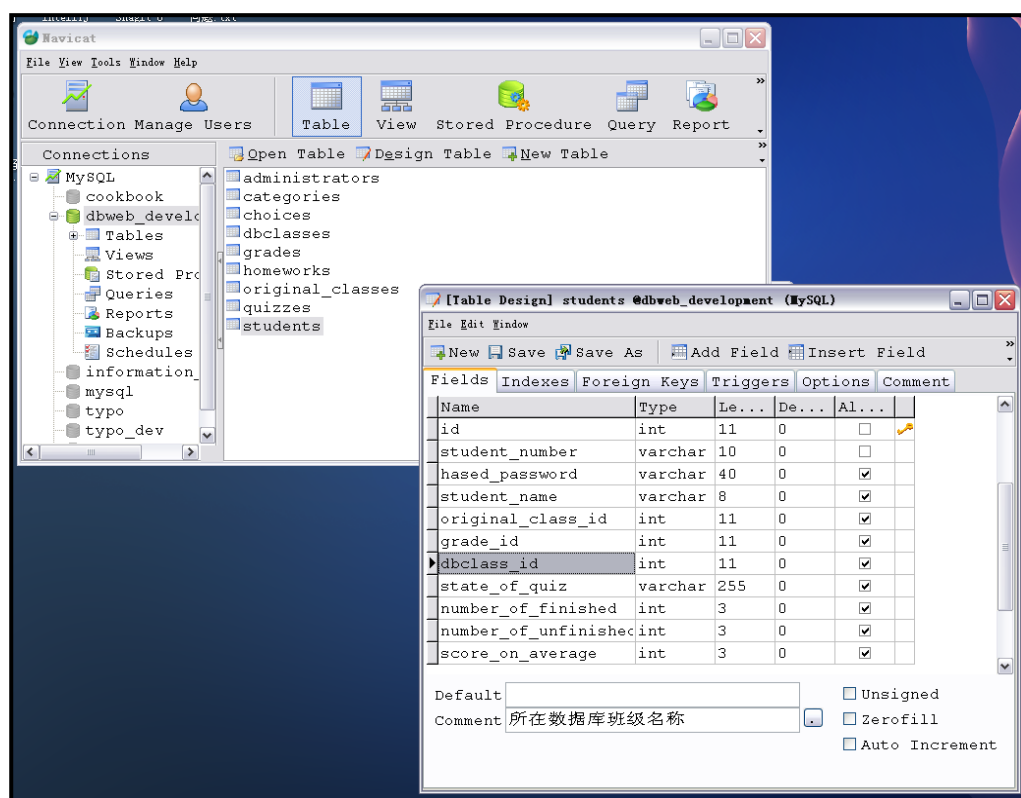


图 2.2 Navicat SQL 的操作界面

Navicat SQL 支持多重 MySQL 服务器的联机，同时可连接数个远程及本地数据库。远程 MySQL 服务器可以是在 Linux、Unix、Mac OS X 或 Windows 平台中执行。

虽然 Navicat SQL 在国内普及并不高，但在国外已经成为 MySQL 的 GUI Client 的首选，对比 phpmyadmin 和 SQLyog 等国内较流行的产品可以立刻看出 Navicat SQL 的优势：

普及率较高的 phpmyadmin 基于 B/S 模式，优势是远程操做，一旦配置好服务器即可在任意主机启动网页浏览器进行操作。但其安装十分繁琐，首先需

要假设 apache 然后需要安装 php，对于对 apache 和 php 不熟悉的开发人员来说是一个难点。另外其瘦客户端的机制无形中降低了对数据库的访问速度。

刚刚推出社区版的 SQLyog 有很多的用户群，但是目前 SQLyog 仍然没有开发出对 UTF8 的支持，这使得网站和数据库在中文的编码和统一问题无法轻松解决。也是很少用用户利用 SQLyog 开发中文应用的原因。

Navicat SQL 充分弥补了以上两款软件的缺点。首先基于 C/S 模式的 Navicat SQL 解决了 phpmyadmin 安装复杂及速度慢的缺点，并提供良好的 UTF8 编码支持，可以轻松对数据库的 UTF8 编码的中文进行编辑和修改，使得网页，数据库，数据库 GUI 的编码得以统一。

Navicat SQL 更有如下强大功能及快捷方式：超时自动重新连接 SQL server，数据及结构同步，新查询建立程序 – 可从不同的数据库建立查询，查询参数，SQL 中控台，检视建立程序，建立检视、预存程序及触发器，SSH 通道私钥，支持所有 MySQL 版本，SSH 通道，外键，Unicode 及字符集支援，编辑 Blob 字段文字，打印数据表结构，从 ODBC 汇入数据，汇入 / 导出超过 18 种最常见的格式数据，包括 MS Access, MS Excel, XML, PDF 及 TXT，建立备份排程表、汇入 / 导出数据传输、储存查询及数据同步作业，以可视化报告建立程序建立报告，报告归档 – 报告可存成一个压缩文件 (.raf) 以便备份及往后增加，从命令行建立报告归档。

第三章 需求分析

3.1 任务概述

《数据库原理课程教学网站建设》的目的是建成一个可以供学生和老师使用的网上教学平台，拥有课程介绍，在线学习，课件下载，论坛，在线答题，作业处理等功能。整个项目计划成为我校数据库教改项目之一。项目完成后，不但可以供本校师生使用，更可以针对国内外大专院校的需求进行升级最终成为一个课程类型无关的公共教学网站框架。

3.2 功能需求

为实现在线答题功能和作业处理功能，计划将整个子项目分成如图 3.1 所示的如下三个模块进行开发：

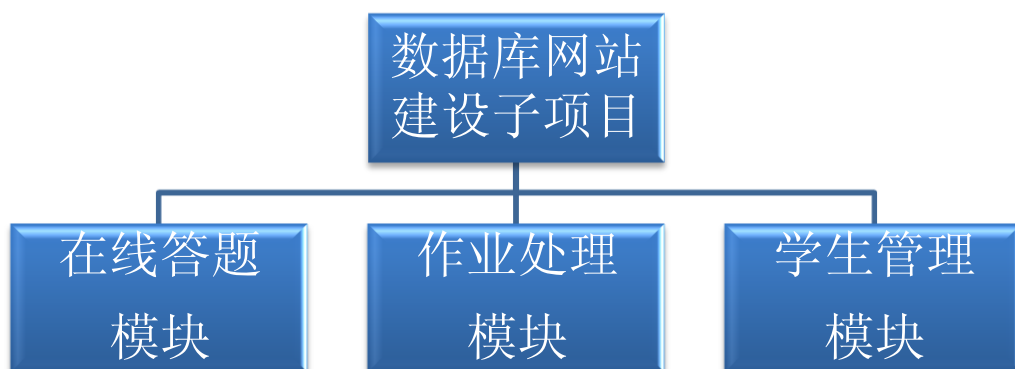


图 3.1 项目结构

3.2.1 在线答题模块

其中在线答题模块要求同学通过用户名和密码在网站在线答题(主要是选择题)，提交后能够立刻得到分数并看到正确答案。教师可以以管理员身份建立题库，插入题库，编辑题库和删除题库。具体需要实现如下功能：

教师端

1. 教师需通过管理员登录窗口进行登录后才能进行各项操作。
2. 无权限或非法用户无法进行操作。
3. 教师可以对章节进行新建，编辑，插入，删除等操作。
4. 教师可以对选择题进行新建，编辑，插入，删除等操作。
5. 选择题要求实现文字和图像的共同描述功能。

学生端

1. 学生需通过学生登录窗口进行登录后方能进行各项操作。
2. 学生可以按章节查看习题。
3. 学生可以按章节答题。
4. 每道题的选项顺序实现随机化。
5. 不同的学生的选项顺序不同。
6. 学生答题结束并提交后系统自动评。
7. 系统评分同时需显示正确答案。
8. 系统对学生的错误答案进行突出显示以帮助学生改正。

3.2.2 作业处理模块

作业处理模块要求同学能够通过用户名和密码在网站即时看到作业的说明和要求，在线提交作业。教师可以以管理员身份编辑作业题目和修改作业内容，并能够下载学生的作业。教师离线评阅学生的作业后可以在线对学生作业评分。每轮作业结束后网站可以显示正确答案。具体需要实现如下功能：

教师端

1. 教师需通过管理员登录窗口进行登录后才能进行各项操作。
2. 无权限或非法用户无法进行操作。
3. 教师可以对章节进行新建，编辑，插入，删除等操作。
4. 教师可以对作业题进行新建，编辑，插入，删除等操作。

5. 作业题要求实现文字和图像的共同描述功能。
6. 作业题能够设置问题显示时间，即学生只有在规定时间之后方能看到题目并答题。
7. 作业题能够设置答案显示时间，即学生只有在规定时间之后方能看到答案。
8. 学生对题目作答并上传作业后教师可以下载学生的作业并批改。
9. 教师对学生作业能够评分并加入评语。

学生端

1. 学生需通过学生登录窗口进行登录后方能进行各项操作。
2. 学生可以查看作业题。
3. 学生能够对问题作答并上传作业。
4. 学生上传作业后，如教师未对作业批改，则学生可以更新自己的作业。
5. 教师批改后学生可以看到分数，评语并下载自己曾经提交的作业。
6. 学生能够即时查看自己的作业状态。
7. 学生无法查看别人的作业及状态。

3.2.3 学生管理模块

学生管理模块能完全针对教师端开发，使教师能够对学生进行管理，如设置学生密码，自然状况等。学生管理模块还包含能够查看学生作业情况的汇总表格系统，并计算学生的平均成绩的功能。具体需要实现如下功能：

1. 教师需通过管理员登录窗口进行登录后才能进行各项操作。
2. 无权限或非法用户无法进行操作。
3. 教师可以对学生进行新建，编辑，插入，删除等操作。
4. 教师可以对数据库班进行新建，编辑，插入，删除等操作。
5. 教师可以对学生自然班进行新建，编辑，插入，删除等操作。
6. 教师可以对年级进行新建，编辑，插入，删除等操作。
7. 教师可以查看各数据库班的人数。
8. 教师可以查看学生自然班的人数。
9. 教师可以查看各年级的人数。
10. 教师可以按数据库课程的不同班级查看学生作业状况。
11. 系统能够提供计算学生作业平均分的功能。
12. 通过汇总表格，能够方便地管理学生信息，批改学生作业。

第四章 数据库设计

通过对项目的需求分析研究,实现需求分析的全部功能需要建立管理员列表,学生列表,章节列表,选择题列表,作业的题目及答案列表,学生作业列表,学生自然班列表,学生年级列表,数据库班级列表等共九个列表。本章将分别从数据库表结构和各间关系两方面说明数据库的结构。在部分应用触发器对密码和人数等属性进行自动控制,具体内容会在第六章中详细介绍。

4.1 数据库表结构

所有表中的 id 均为惟一识别序列号,类型为 int,自动增加。id 也是各表的主键。表中的主键是 id 也是 Ruby On Rails 架构的强制性要求。

各表结构及键值意义如下所示:

4.1.1 管理员列表 (administrators)

管理员列表的表结构如表 4.1 所示:


Name	Type	Length	Decimals	Allow Null	
id	int	11	0	<input type="checkbox"/>	
name	varchar	100	0	<input type="checkbox"/>	
hashed_password	char	40	0	<input checked="" type="checkbox"/>	
salt	varchar	255	0	<input checked="" type="checkbox"/>	

表 4.1 管理员列表的表结构

各列含义及功能分别是:

name: 管理员名字

hashed_password: 加 salt 后的密码的 sha1 散列运算

salt: 在密码原文后面加入随机数字,防止使用字典破解。

4.1.2 学生列表 (students)

学生列表的表结构如表 4.2 所示:


Name	Type	Length	Decimals	Allow Null	
id	int	11	0	<input type="checkbox"/>	
student_number	varchar	10	0	<input type="checkbox"/>	
hased_password	varchar	40	0	<input checked="" type="checkbox"/>	
student_name	varchar	8	0	<input checked="" type="checkbox"/>	
original_class_id	int	11	0	<input checked="" type="checkbox"/>	
grade_id	int	11	0	<input checked="" type="checkbox"/>	
dbclass_id	int	11	0	<input checked="" type="checkbox"/>	
state_of_quiz	varchar	255	0	<input checked="" type="checkbox"/>	
number_of_finished	int	3	0	<input checked="" type="checkbox"/>	
number_of_unfinished	int	3	0	<input checked="" type="checkbox"/>	
score_on_average	int	3	0	<input checked="" type="checkbox"/>	

表 4.2 学生列表的表结构

各列含义及功能分别是：

Student_number: 学号

Hased_password: sha1 散列运算后的密码

Student_name: 姓名

Original_class_id: 自然班 id

Grade_id: 年级 id

Dbclass_id: 数据库班 id

State_of_quiz: 作业完成状态 0—未做 1—已提交 2—已批改 默认 0

Number_of_finished: 已完成作业数

Number_of_unfinished: 未完成作业数

Score_on_average: 平均分

4.1.3 章节列表（categories）

章节列表的表结构如表 4.3 所示：


Name	Type	Length	Decimals	Allow Null	
id	int	11	0	<input type="checkbox"/>	
category_name	varchar	30	0	<input checked="" type="checkbox"/>	

表 4.3 章节列表的表结构

各列含义及功能分别是：

Category_name: 章节名

4.1.4 选择题列表（choices）

选择题列表的表结构如表 4.4 所示：


Name	Type	Length	Decimals	Allow Null	
id	int	11	0	<input type="checkbox"/>	
category_id	int	11	0	<input checked="" type="checkbox"/>	
quiz_trunk	text	0	0	<input checked="" type="checkbox"/>	
image	varchar	255	0	<input checked="" type="checkbox"/>	
answer	text	0	0	<input checked="" type="checkbox"/>	
false1	text	0	0	<input checked="" type="checkbox"/>	
false2	text	0	0	<input checked="" type="checkbox"/>	
false3	text	0	0	<input checked="" type="checkbox"/>	
selected	text	0	0	<input checked="" type="checkbox"/>	
show_red	varchar	11	0	<input checked="" type="checkbox"/>	

表 4.4 选择题列表的表结构

各列含义及功能分别是：

Category_id: 章节 id

Quiz_trunk: 题干

Image: 图像（图像数据直接存储在文件系统中，此处存储仅为本地地址）

Answer: 正确答案

False1: 错误答案 1

False2: 错误答案 2

False3: 错误答案 3

Selected: 学生选择的答案

Show_red: 0—学生此题做对 1—学生此题做错 默认 0

4.1.5 作业的题目及答案列表（quizzes）

题目列表的表结构如表 4.5 所示：



Name	Type	Length	Decimals	Allow Null	
id	int	11	0	<input type="checkbox"/>	
serial_number	int	11	0	<input type="checkbox"/>	
category_id	int	11	0	<input checked="" type="checkbox"/>	
question_name	varchar	255	0	<input checked="" type="checkbox"/>	
question_abstract	text	0	0	<input checked="" type="checkbox"/>	
question_detail	text	0	0	<input checked="" type="checkbox"/>	
image	varchar	255	0	<input checked="" type="checkbox"/>	
answer	text	0	0	<input checked="" type="checkbox"/>	
date_question_available	datetime	0	0	<input checked="" type="checkbox"/>	
date_answer_available	datetime	0	0	<input checked="" type="checkbox"/>	

表 4.5 题目列表的表结构

各列含义及功能分别是：

- Serial_number: 题目惟一序列号
- Category_id: 章节 id
- Question_name: 题目名称
- Question_abstract: 题目摘要
- Question_detail: 题目内容
- Image: 图表（图表数据直接存储在文件系统中，此处存储仅为本地地址）
- Answer: 答案
- Date_question_available: 题目显示时间
- Date_answer_available: 答案显示时间

4.1.6 学生作业列表（homeworks）

作业列表的表结构如表 4.6 所示：




Name	Type	Length	Decimals	Allow Null	
id	int	11	0	<input type="checkbox"/>	
student_number	varchar	10	0	<input type="checkbox"/>	
quiz_serial_number	int	11	0	<input type="checkbox"/>	
file	varchar	255	0	<input checked="" type="checkbox"/>	
score	int	3	0	<input checked="" type="checkbox"/>	
teacher_suggestion	text	0	0	<input checked="" type="checkbox"/>	
check_flag	int	1	0	<input checked="" type="checkbox"/>	
update_time	datetime	0	0	<input type="checkbox"/>	

表 4.6 作业列表的表结构

各列含义及功能分别是：

Student_number: 学号

Quiz_serial_number: 题目序列号

File: 作业（作业数据直接存储在文件系统中，此处存储仅为本地地址）

Score: 作业成绩

Teacher_suggestion: 教师评语

Check_flag: 标记位 0—未提交 1—已提交 2—已批改

Update_time: 更新时间

4.1.7 学生自然班列表（original_classes）

学生自然班列表的表结构如表 4.7 所示：


Name	Type	Length	Decimals	Allow Null	
id	int	11	0	<input type="checkbox"/>	
original_class_name	varchar	20	0	<input checked="" type="checkbox"/>	
num_of_student	int	11	0	<input checked="" type="checkbox"/>	

表 4.7 学生自然班列表的表结构

各列含义及功能分别是：

Original_class_name: 自然班名

Num_of_student: 学生人数

4.1.8 学生年级列表（grades）

学生年级列表的表结构如表 4.8 所示：


Name	Type	Length	Decimals	Allow Null	
id	int	11	0	<input type="checkbox"/>	
grade_name	varchar	20	0	<input checked="" type="checkbox"/>	
num_of_student	int	11	0	<input checked="" type="checkbox"/>	

表 4.8 学生年级列表的表结构

各列含义及功能分别是：

Grade_name: 年纪名称，如 03 级，04 级，05 级

Num_of_student: 年级人数

4.1.9 数据库班级列表（dbclasses）

数据库班表的表结构如表 4.9 所示：


Name	Type	Length	Decimals	Allow Null	
id	int	11	0	<input type="checkbox"/>	
dbclass_name	varchar	40	0	<input type="checkbox"/>	
num_of_quiz	int	4	0	<input checked="" type="checkbox"/>	
num_of_student	int	11	0	<input checked="" type="checkbox"/>	

表 4.9 数据库班表的表结构

各列含义及功能分别是：

Dbclass_name: 数据库班级名

Num_of_quiz: 本班作业总数

Num_of_student: 学生总数

4.2 数据库表间关系

上一节小节中描述的各表间有如图 4.1 的引用关系

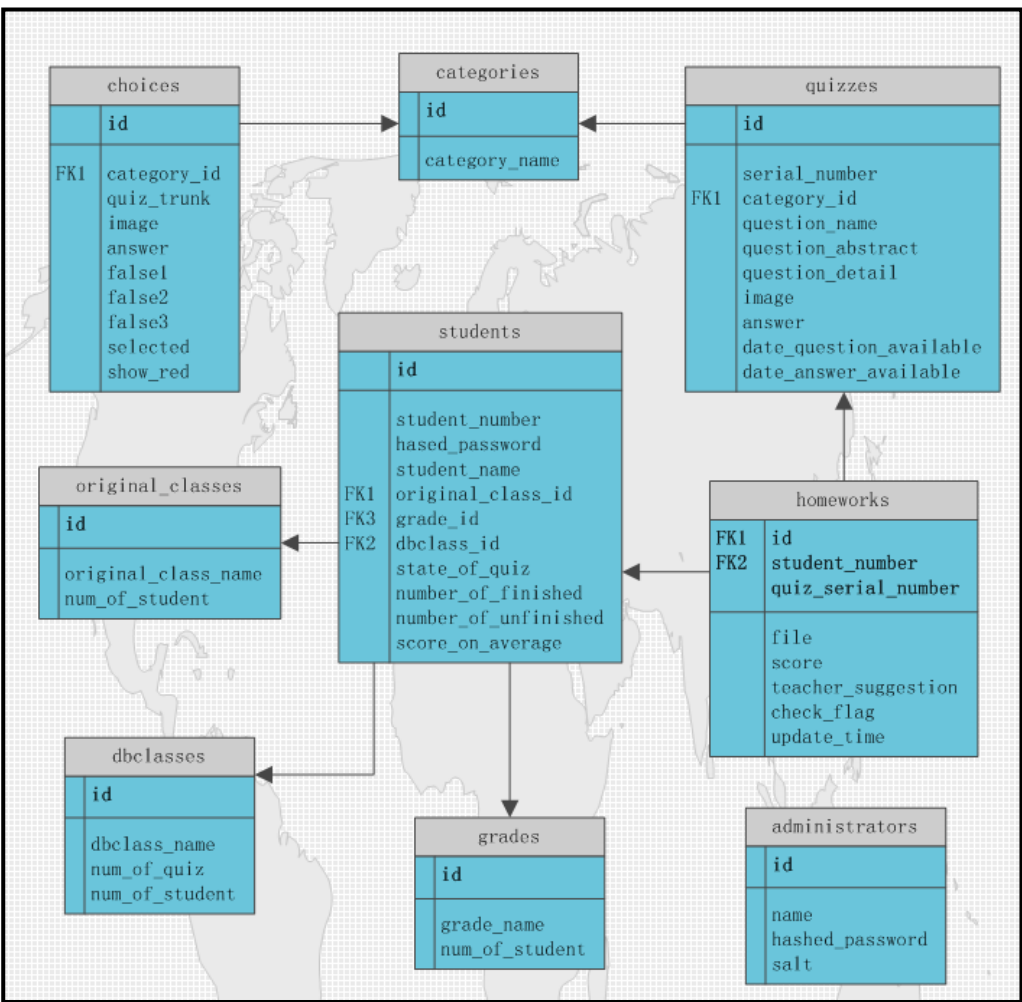


图 4.1 各表间的引用关系

- 其中 choices.category_id
其中 quizzes.category_id
其中 students.original_class_id
其中 students.grade_id
其中 students.dbclasses_id
- 引用 categories.id
引用 categories.id
引用 original_classes.id
引用 grades.id
引用 dbclasses.id

其中 homeworks.student_number	引用 students.student_number
其中 homeworks.quiz_serial_number	引用 quizzes.serial_number

通过上述引用可以在各表间方便地进行查找，也保证了数据的唯一性，在大大降低了存储空间的同时，实现了数据间的紧密结合和精度的提高。表间的引用关系也为题目录入时精确的下拉菜单选项提供了基础，关于此部分内容会在第五章详细论述。

第五章 功能模块的设计和实现

本人承担的数据库教学网站部分分三个功能模块：在线答题模块，作业提交模块和学生管理模块。三个模块共用管理员列表，学生列表，章节列表，学生自然班列表，学生年级列表，数据库班级列表，使这个项目各模块联系紧密。又因在线答题模块和作业提交模块虽有相似但目的和实现方法不尽相同，故在控制器和视图方面各自为营大大降低耦合，为今后的升级和扩展打下良好基础。学生管理模块专门为管理员即教师提供学生管理和信息汇总功能。在学生汇总查询视图中，系统加入动态链接，使教师可以轻松掌握各学生的基本状况，并能通过链接查看或批改学生的作业。三个模块及内部功能子模块结构如图 5.1 所示：

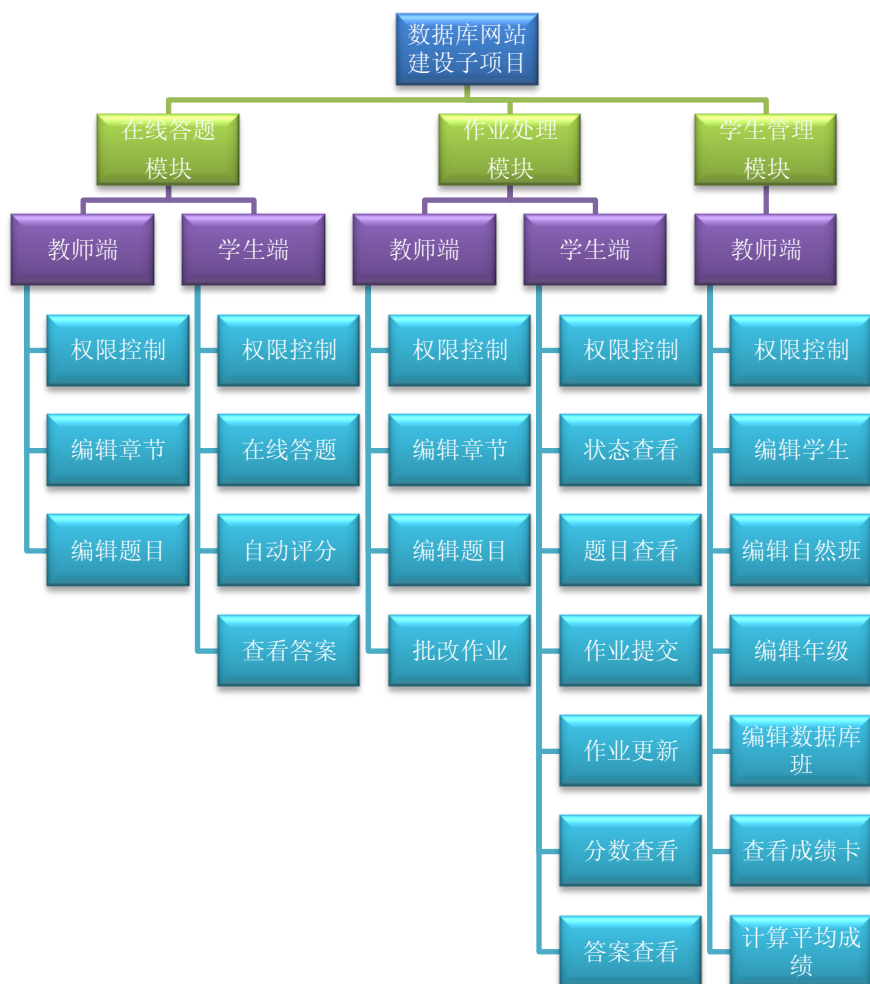


图 5.1 三个模块及内部功能子模块结构图

5.1 在线答题模块

在线答题模块题要求同学通过用户名和密码在网站在线答题,提交后能够立刻得到分数并看到正确答案。教师可以以管理员身份建立题库,插入题库,编辑题库和删除题库。

5.1.1 教师端

图 5.2 所以为管理员即教师登录后显示界面。

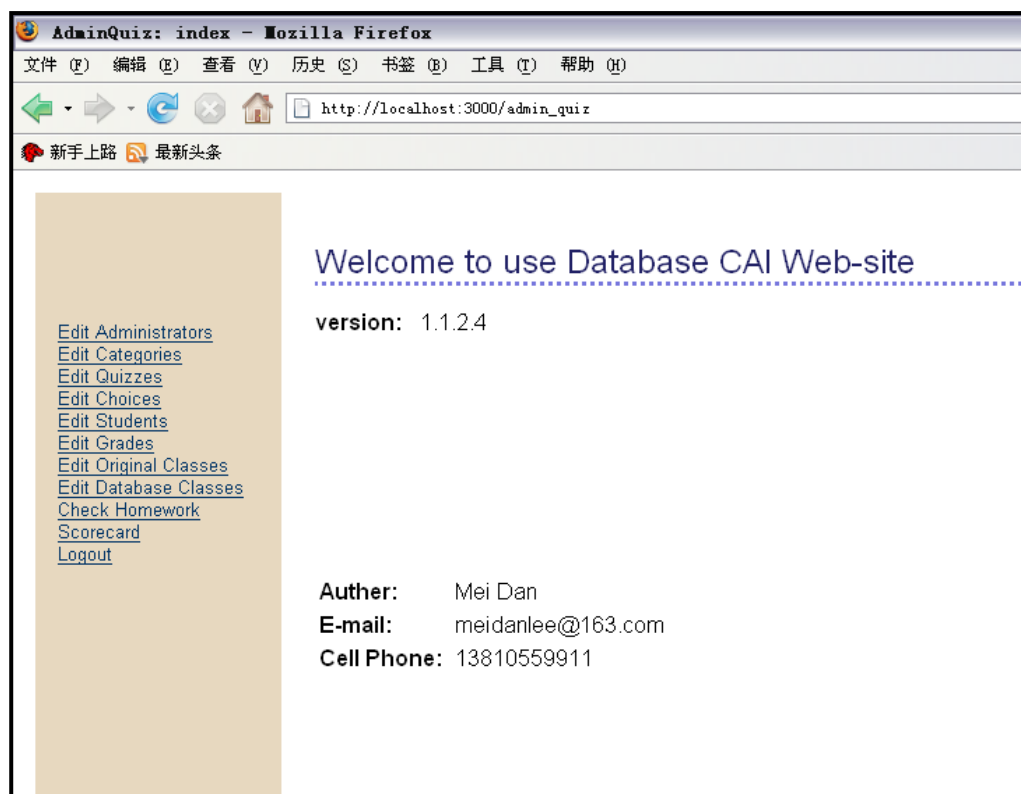


图 5.2 管理员欢迎界面

从登录后左侧工具菜单中可以看出,教师端有如下功能(按从上至下顺序):

-
-
1. 编辑管理员
 2. 编辑章节
 3. 编辑作业题
 4. 编辑选择题
 5. 编辑学生
 6. 编辑年级
 7. 编辑自然班
 8. 编辑数据库班
 9. 批改作业
 10. 查看学生成绩卡
 11. 登出

5.1.1.1 编辑管理员

编辑管理员的界面如图 5.3 所示：

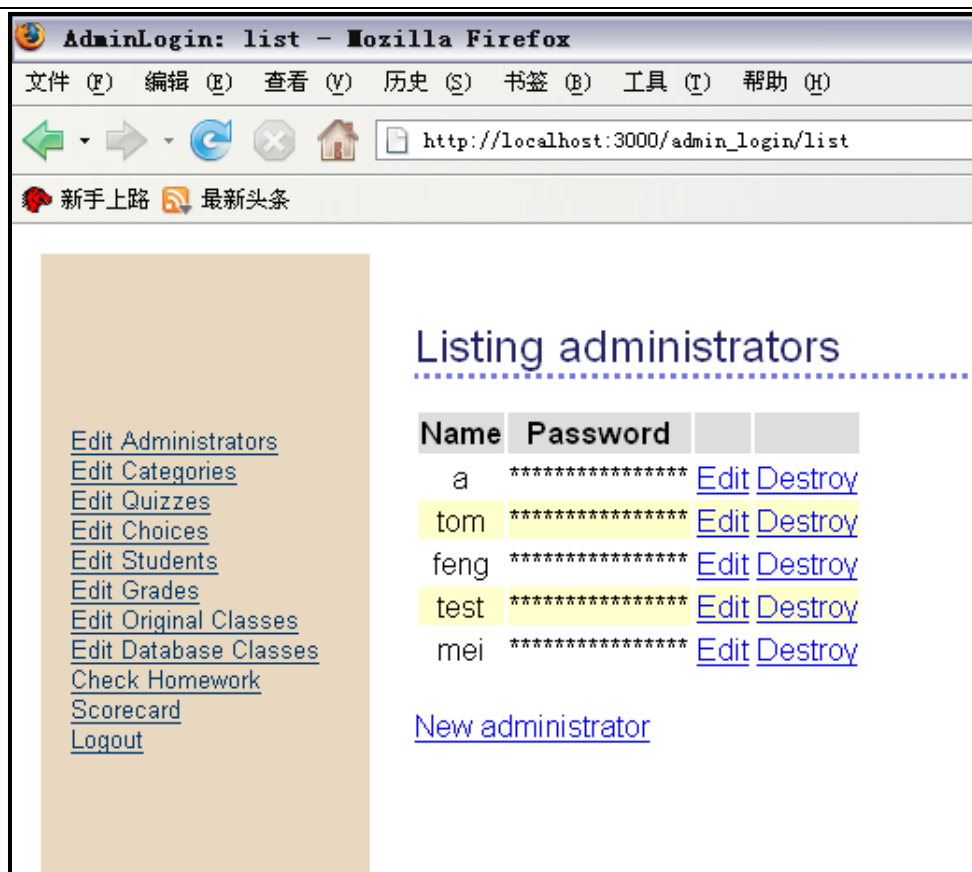


图 5.3 编辑管理员的界面

管理员可以通过主功能页面查看当前管理员列表，新建管理员，编辑管理员，或者删除管理员。

图 5.4 所示为新建管理员界面，在输入管理员密码时要求输入两次以防止输入错误。

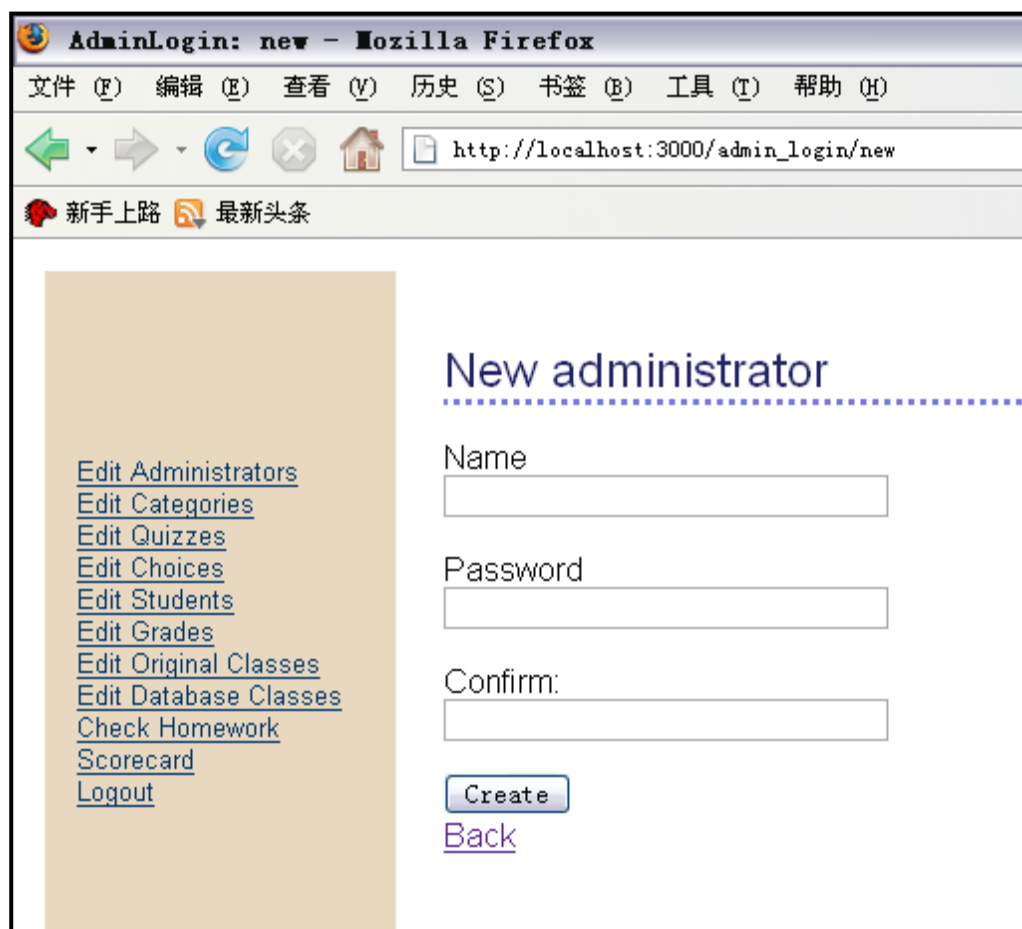


图 5.4

管理员界面加入了特别验证机制, 要求 password 与 name 必须相同以提高密码强度。图 5.5 所示为密码和用户名相同 (均为 BJTU) 时的提示用户无法新建管理员。

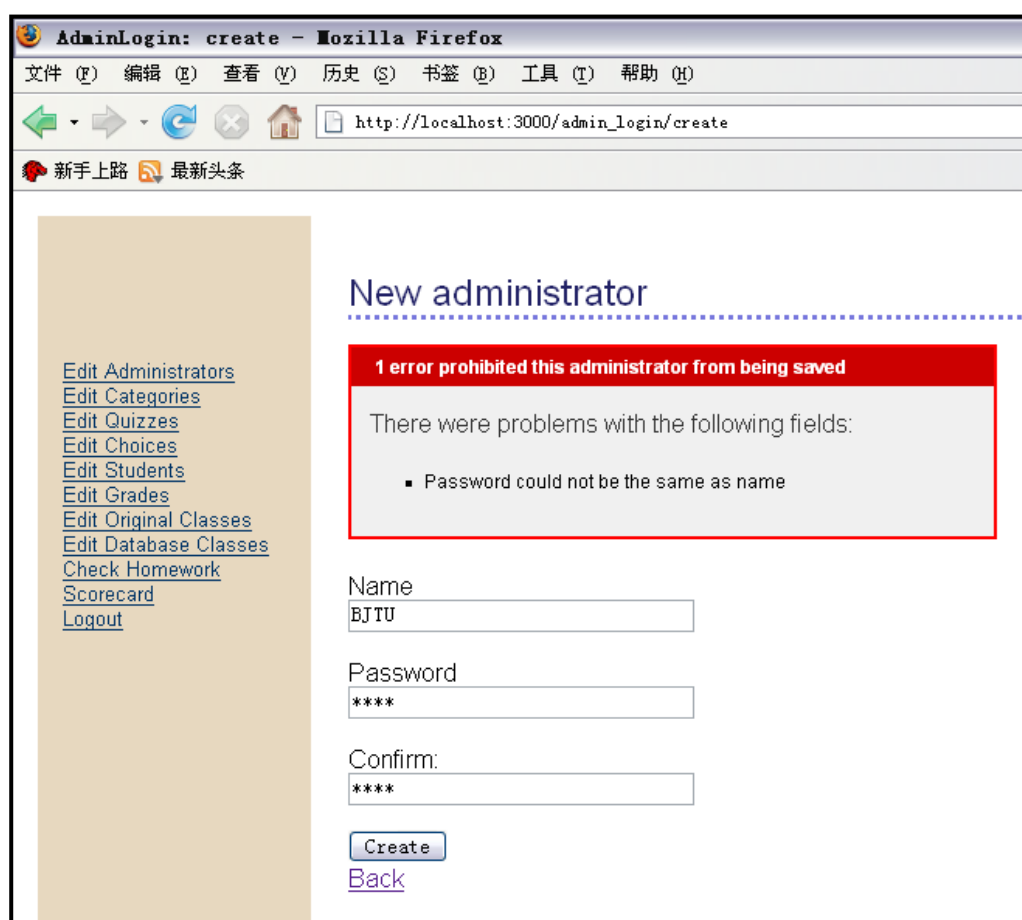


图 5.5 无法新建管理员的错误提示

实现上述功能的文件为

/dbweb/app/models/administrators.rb

核心代码如图 5.6 所示：

```
class Administrator < ActiveRecord::Base

  validates_presence_of :name
  validates_uniqueness_of :name

  attr_accessor :password
  attr_accessor :password_confirmation
  validates_confirmation_of :password

  def validate
    errors.add_to_base("Missing password") if hashed_password.blank?
    errors.add_to_base("Password could not be the same as name") if name==password
  end
end
```

图 5.6 管理员模型验证核心代码

以下两句分别验证 name 不能为空和在列表内必须惟一。

```
validates_presence_of :name
validates_uniqueness_of :name
```

以下三句实现了密码的重复输入验证。

```
attr_accessor :password
attr_accessor :password_confirmation
validates_confirmation_of :password
```

validate 函数用于验证密码不能为空以及 name 和 password 不能相同。

5.1.1.2 编辑章节

编辑章节的页面如图 5.7 所示：

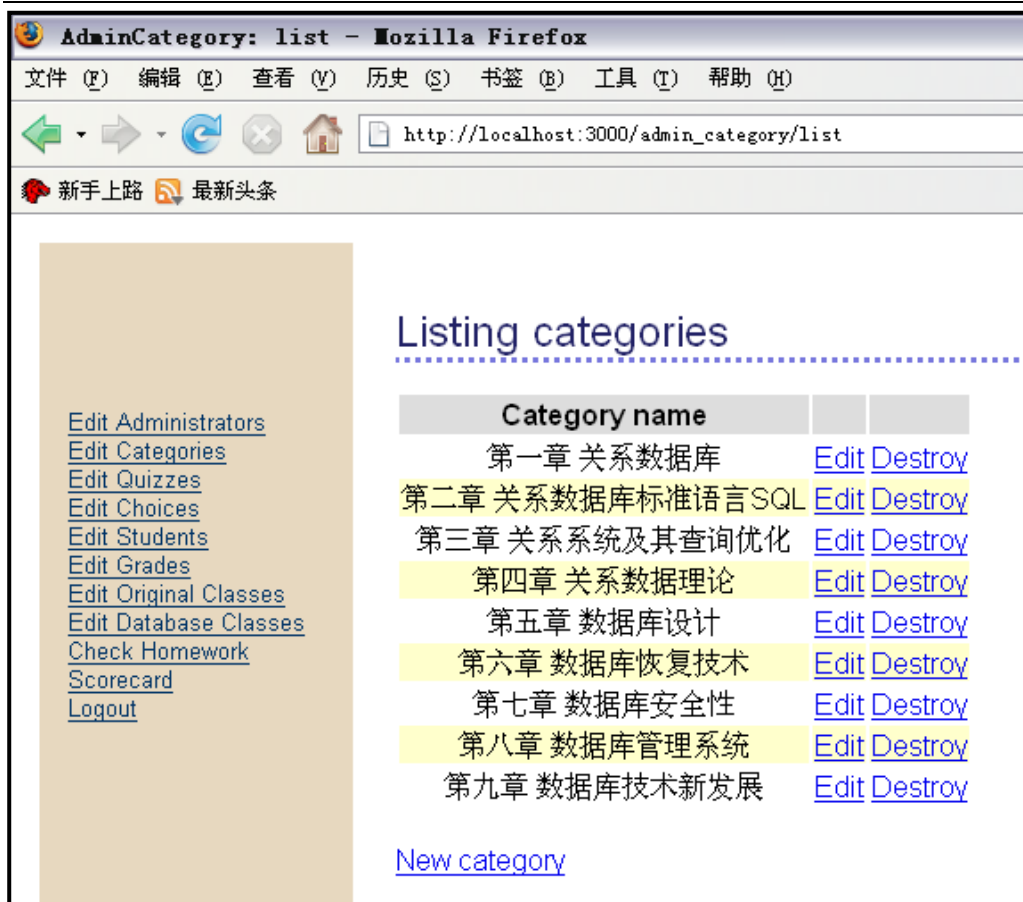


图 5.7 编辑章节的页面

通过列表及每行最后的超级链接可以方便的对章节进行编辑和删除。一旦章节删除，与被删除章节相关的所有题目均会被自动删除。

整个 category 的 action 都由 dbweb\app\controllers\admin_category_controller.rb 进行控制。

列表控制器代码如下：

```
def list
  @category_pages, @categories = paginate :categories, :per_page
=> 10
end
```

代码在实现列表的同时利用 Ruby On Rails 的便捷机制建立起分页功能，`:per_page => 10` 规定了每页 10 个条目。

新建章节控相关制器代码如下：

```
def new
  @category = Category.new
end
```

控制器 AdminCategoryController :: new 自动调用视图 dbweb/app/views/admin_category/new.rhtml 显示如图 5.8 的输入页面：

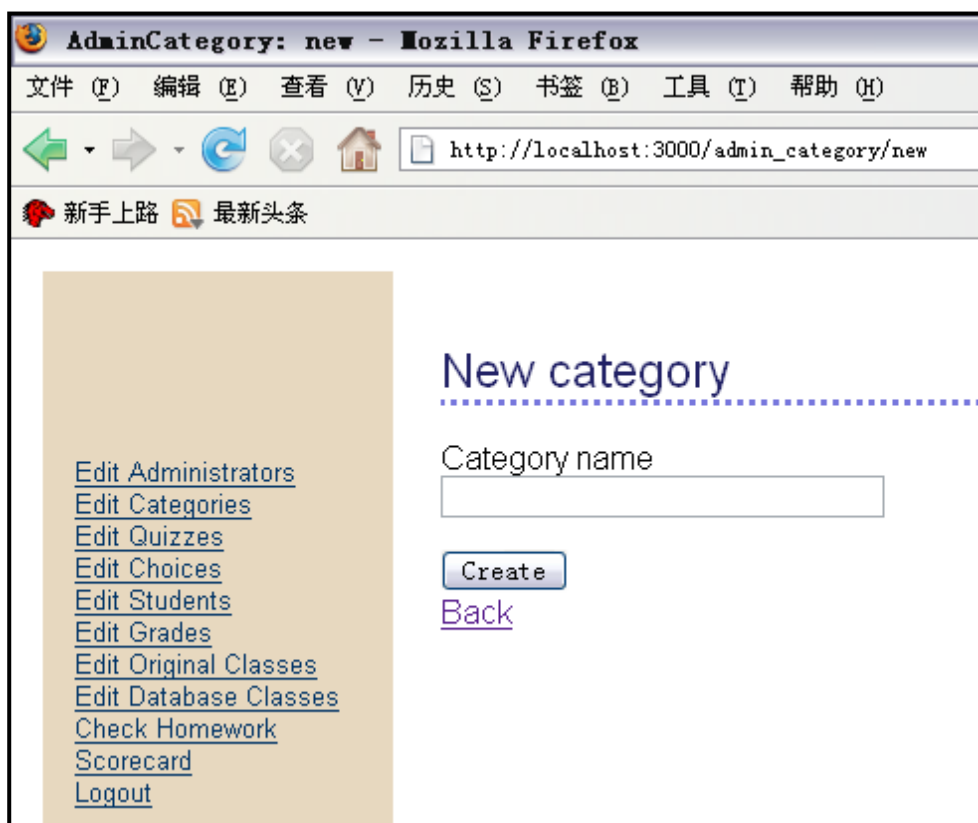


图 5.8 新建章节

New.rhtml 文件内容如下：

```
<h1>New category</h1>
```

```
<% form_tag :action => 'create' do %>
  <%= render :partial => 'form' %>
  <%= submit_tag "Create" %>
<% end %>
```

```
<%= link_to 'Back', :action => 'list' %>
```

<%= render :partial => 'form' %> 一句调用 `_form.rhtml` 通用 form，显示输入提示和输入框。`_form.rhtml` 的内容如下：

```
<%= error_messages_for 'category' %>

<!--[form:category]-->
<p><label for="category_category_name">Category name</label><br/>
<%= text_field 'category', 'category_name' %></p>
<!--[eoform:category]-->
```

`New.rhtml` 最后提交按钮直接将 action 转至 `create`，`create` 的定义如下：

```
def create
  @category = Category.new(params[:category])
  if @category.save
    flash[:notice] = 'Category was successfully created.'
    redirect_to :action => 'list'
  else
    render :action => 'new'
  end
end
```

AdminController :: create 其中通过 @category.save 将新输入的章节存入数据库。整个存储过程完成。

Category 的更新过程和新建过程相似，核心更新 action 为 AdminCategoryController :: update，代码如下：

```
def update
  @category = Category.find(params[:id])
  if @category.update_attributes(params[:category])
    flash[:notice] = 'Category was successfully updated.'
    redirect_to :action => 'show', :id => @category
  else
    render :action => 'edit'
  end
end
```

Category 的删除较简单，在 list.rhtml 的链接中调用 AdminCategoryController :: destroy，通过 id 找到需要删除的 category。Destroy 的定义如下：

```
def destroy
  Category.find(params[:id]).destroy
  redirect_to :action => 'list'
end
```

5.1.1.3 编辑选择题

选择题列表页面如图 5.9 所示：

Listing choices

Category	Trunk	Answer	False 1	False 2	False 3	
第一章 关系数据库	关系数据库中，实现实体之间的联系是通过表与表之间的公共属性 公共索引 公共存储 公共元组					Show Edit Destroy
第一章 关系数据库	题干 2	对	错 1	错 2	错 3	Show Edit Destroy
第一章 关系数据库	题干 3	对	错 1	错 2	错 3	Show Edit Destroy
第一章 关系数据库	题干 4	对	错 1	错 2	错 3	Show Edit Destroy

New choice

图 5.9 选择题列表页面

为验证系统的准确性和可行性，第一行显示的是真正的题目即在项目应用中使用的题目，后三行显示的是系统测试用的题目，没有真正的题干，而只是标号和显示答案的对错。

多选题的列表、分页显示、新建、编辑和删除机制与 category 十分相似，这里不再重复论述。下面就选择题所特有的一些功能的实现进行介绍。

新建题目提供章节选择：

每一个选择题在新建的时候都可以通过智能下拉列表进行章节的选择，效果如图 5.10 所示：



图 5.10 智能下拉列表

此处的下拉列表完全根据 category 自动生成，无须人工控制，未建立的章

节不会在此显示，教师可以通过本功能方便地进行题目归类。实现章节自动选择的视图代码如下：

```
<p>
  <label for="choice_category">Category</label><br/>
  <select name="choice[category_id]">
    <% $categories.each do |category| %>
      <option value="<%= category.id %>">
        <%= category.category_name %>
      </option>
    <% end %>
  </select>
</p>
```

通过第四行的代码，可以看到，下拉列表内的章节全部来自于数据库内的章节。`$categories` 作为一个全局变量用于存储章节，而用户选中的 `value` 是 `category.id`，这就实现了选择题的章节参照章节列表的机制。`$categories` 由下列代码初始化：

```
$categories = Category.find :all
```

此代码保存在 `dbweb\app\controllers\application.rb` 中，是调用所有 `action` 前都会自动调用的代码。

上传图表：

教师可以为自己出的选择题上传图表，上传图表按钮截图 5.11 下所示：

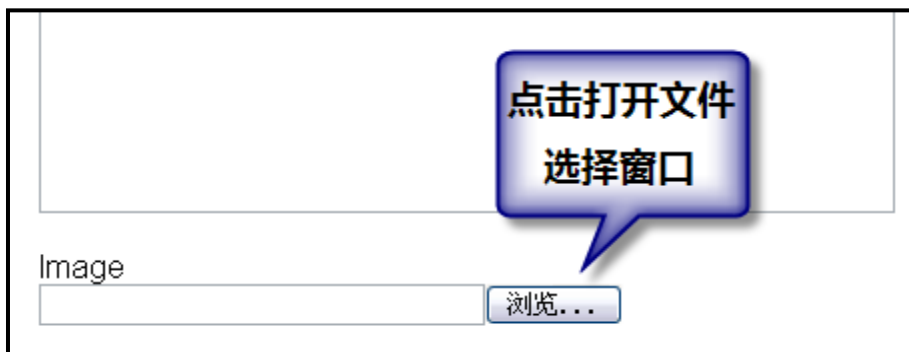


图 5.11 上传图片按钮

点击浏览按钮可以弹出图 5.12 所示的文件选择窗口：



图 5.12 弹出的文件选择窗口

此处图像上传应用 Ruby On Rails 专用的 `file_column` 插件。只要在选择题的 `model` 文件 `dbweb\app\models\choice.rb` 中用 `file_column` 函数声明要用于存储文件的键，即可在视图文件中方便的实现图片的上传和显示。`choice.rb` 内容如图 5.13 所示：

```
1 class Choice < ActiveRecord::Base
2   belongs_to :category
3
4   file_column :image
5
6   validates_presence_of :category_id
7   validates_presence_of :quiz_trunk
8   validates_presence_of :power
9   validates_presence_of :se1
10  validates_presence_of :se2
11  validates_presence_of :se3
12
13  def get_category_name(choice)
14    choice.category.category_name
15  end
16 end
```

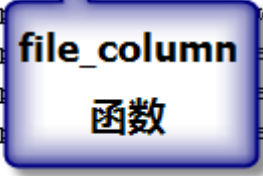
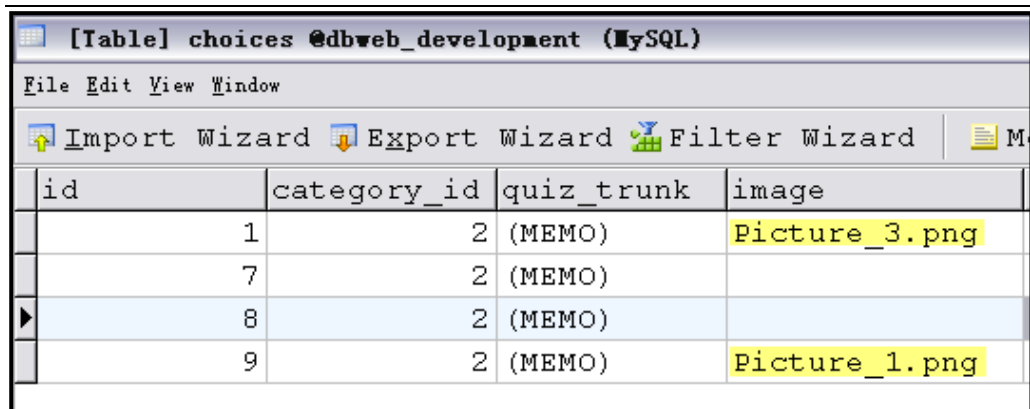


图 5.13 file_column 列

在新建视图中，实现文件选择器的代码如下：

```
<p>
  <label for="quiz_image">Image</label><br/>
  <%= file_column_field "choice", "image" %>
</p>
```

上传的文件会自动存储在服务器的文件系统内而非数据库内，数据库存储的只是文件在服务器公共文件目录下的文件名，如图 5.14 所示：



id	category_id	quiz_trunk	image
1	2	(MEMO)	Picture_3.png
7	2	(MEMO)	
8	2	(MEMO)	
9	2	(MEMO)	Picture_1.png

图 5.14 数据库内容截图

用 `file_column` 上传的文件可以方便地用 `url_for_file_column` 函数来取得图片的 `url`，并用于显示。图 5.15 即为上传后的图片的显示效果：



图 5.15 图片显示效果

5.1.2 学生端

一旦学生登录，就可以从点击左侧的快捷工具栏进行操作，图 5.16 所示为点击做多选题后的效果：

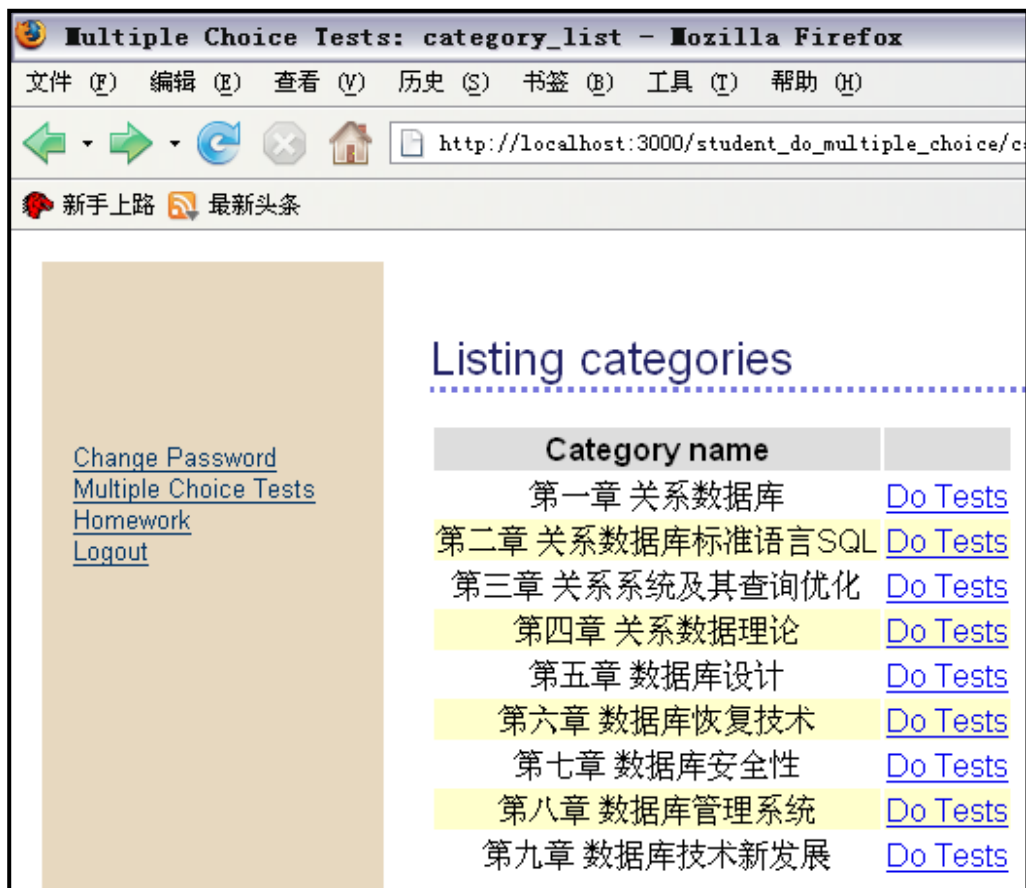


图 5.16 多选题的章节选择页面

左侧的快速链接工具栏提供如下功能：

1. 更改密码
2. 做多选题
3. 做作业
4. 登出

5.1.2.1 做选择题并查看结果

点击章节后面的 do tests 按钮即可查看进入相关章节的题目并作答，选择题做题界面截图如图 5.17 所示：

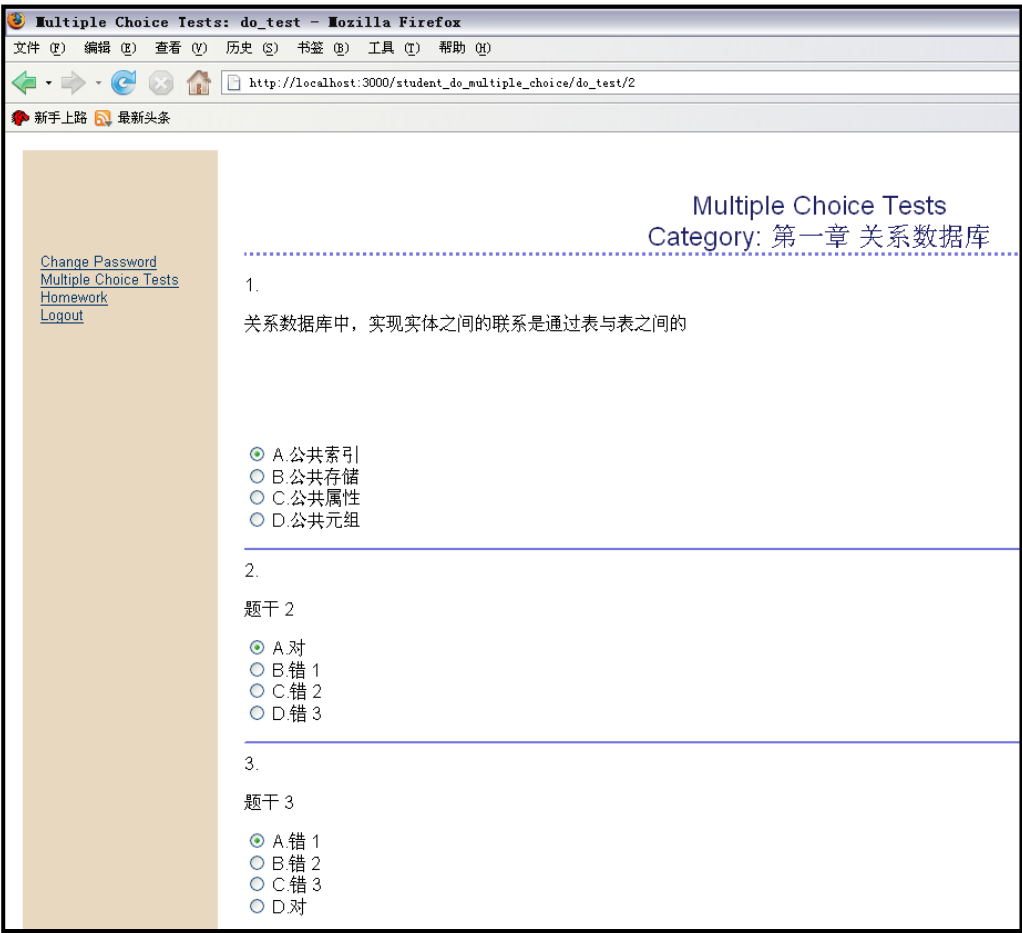


图 5.17 做选择题页面

所有的选项无须人工干预，均随机出现，学生在做完题目后可以点击提交按钮进行提交，然后查看答案界面。答案界面如图 5.18 所示：

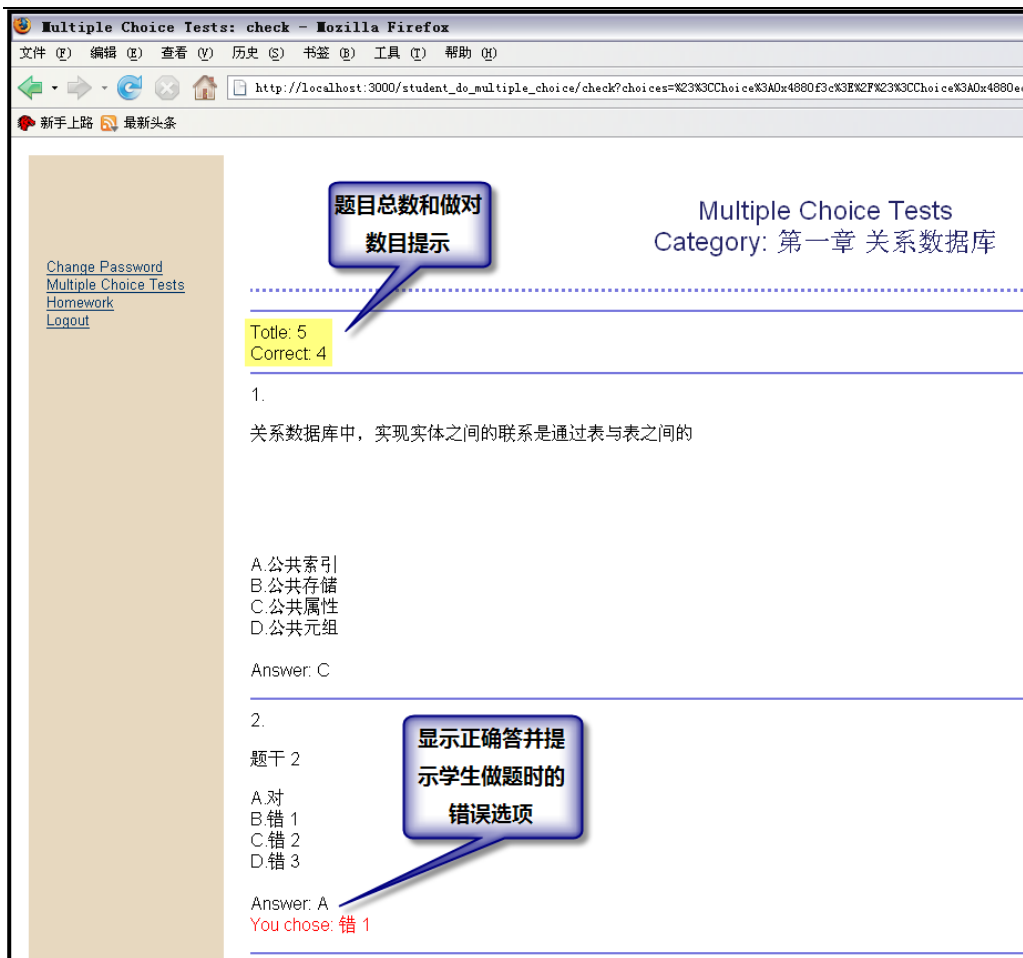


图 5.18 显示答案截图

关于选择题题目随机出现的功能将放在第六章项目亮点中详细讲解，此处的重点是阐述题目的自动批改并提示错误功能。

文件 `dbweb\app\controllers\student_do_multiple_choice_controller.rb` 实现了学生做选择题的全部 action。`StudentDoMultipleChoiceController` 的 `check` 实现了自动批改的功能。原理是学生提交作业后选项将被保存在一组 hash 集合中，并由系统自动判断是否正确，如果正确则正确答案总数加 1，否则标记此题做错并在答案显示页面以红色提示，相关代码如下所示：

def check

```
@category = Category.find_by_id(params[:category][:id])
@choices = Choice.find :all,
:conditions => "category_id = #{@category.id}",
```

```
:order => "id asc"

@total = Choice.count :all,
:conditions => "category_id = #{@category.id}"

@correct_counter = 0

@selected = params[:q]

i = -1
for choice in @choices do
  i = i + 1
  if @selected["#{choice.id}"] == choice.answer
    @correct_counter = @correct_counter + 1
    choice.show_red = "he chose the answer"
  else
    choice.show_red = @selected["#{choice.id}"]
  end
end
end
```

在显示作业答案之后，视图会自动检测此题学生是否做错，如果做错责用红色提示学答题时的选项，实现此功能的代码如下所示：

```
<%if choice.show_red != "he chose the answer"%>
  <font color="red">
    You chose: <%=choice.show_red%>
  </font>
  <br/>
<%end%>
```

5.2 作业提交模块

作业提交模块是本项目的重点也是本项目的难点，要求同学能够通过用户名和密码在网站即时看到作业的说明和要求，在线提交作业。教师可以以管理员身份编辑作业题目和修改作业内容，并能够下载学生的作业。教师离线评阅学生的作业后可以在线对学生作业评分。每轮作业结束后网站可以显示正确答案。

鉴于本模块的逻辑性和顺序性较强，本小节的论述顺序将按一道题的周期进行。首先是教师编辑作业；然后学生在线查看作业并给出答案；教师没有批改作业时学生可以修改即更新自己的作业；教师看到有学生提交作业后可以下载学生的作业，然后查看作业并给出成绩和评语；教师批改作业后学生可以查看分数和评语并下载自己的作业。整个流程的五个步骤如流程图 5.19 所示：



图 5.19 作业提交流程

5.2.1 教师出题功能

作业题出题的机制与选择题出题机制相似，图 5.20 所示截图为教师出题界面

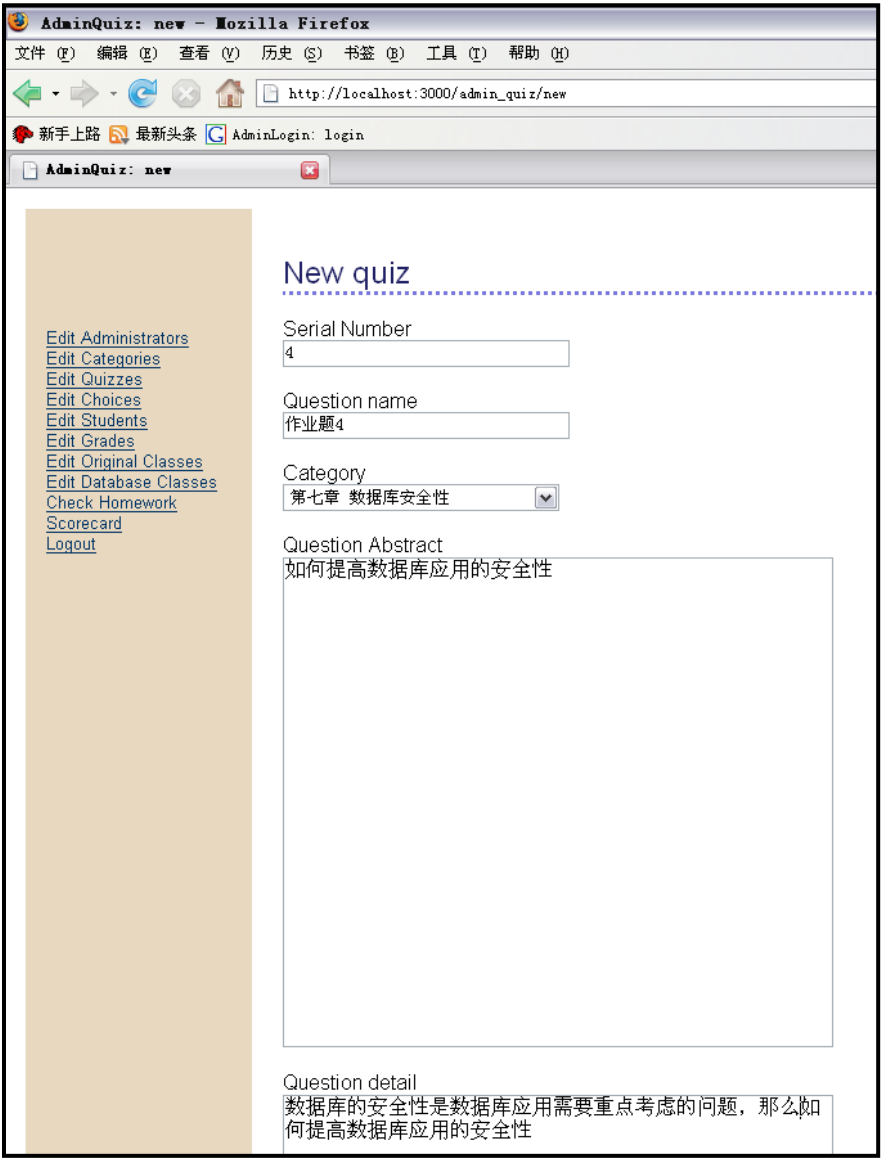


图 5.20 教师出题界面

因题目会教师的计划变化而编辑甚至删除，故用自增的 id 无法连续表示题目序列号。为解决上述问题，本应用特别在题目中加入 Serial Number，用于控制题目序列。此 Serial Number 可看作是管理员界面用于标识题目的主键，所以 Serial Number 有其惟一性。为保持 Serial Number 的惟一，在 model 里面特别限定，Serial Number 存储时检查并确保惟一才能存储。实现本功能的代码保存在 dbweb\app\models\quiz.rb 具体代码如下：

```
class Quiz < ActiveRecord::Base

  belongs_to :category

  validates_uniqueness_of :serial_number
  validates_presence_of :serial_number
  validates_presence_of :question_abstract
  validates_presence_of :question_name
  validates_presence_of :category_id
  validates_presence_of :question_detail
  validates_presence_of :answer

  file_column :image

  def self.available_quizzes
    find(:all,
      :conditions => "date_question_available <= now()",
      :order      => "date_question_available DESC")
  end
end
```

代码 validates_uniqueness_of :serial_number 专门用于验证 serial_number 在数据库内必须惟一。如数据库内已经有序列号为 1 的题目，那么再次新建序列号为 1 的数据库就会报错而达到防止重复的功能。图 5.21 即为新建题目的 serial_number 与数据库内的 serial_number 有重复时的报错效果：

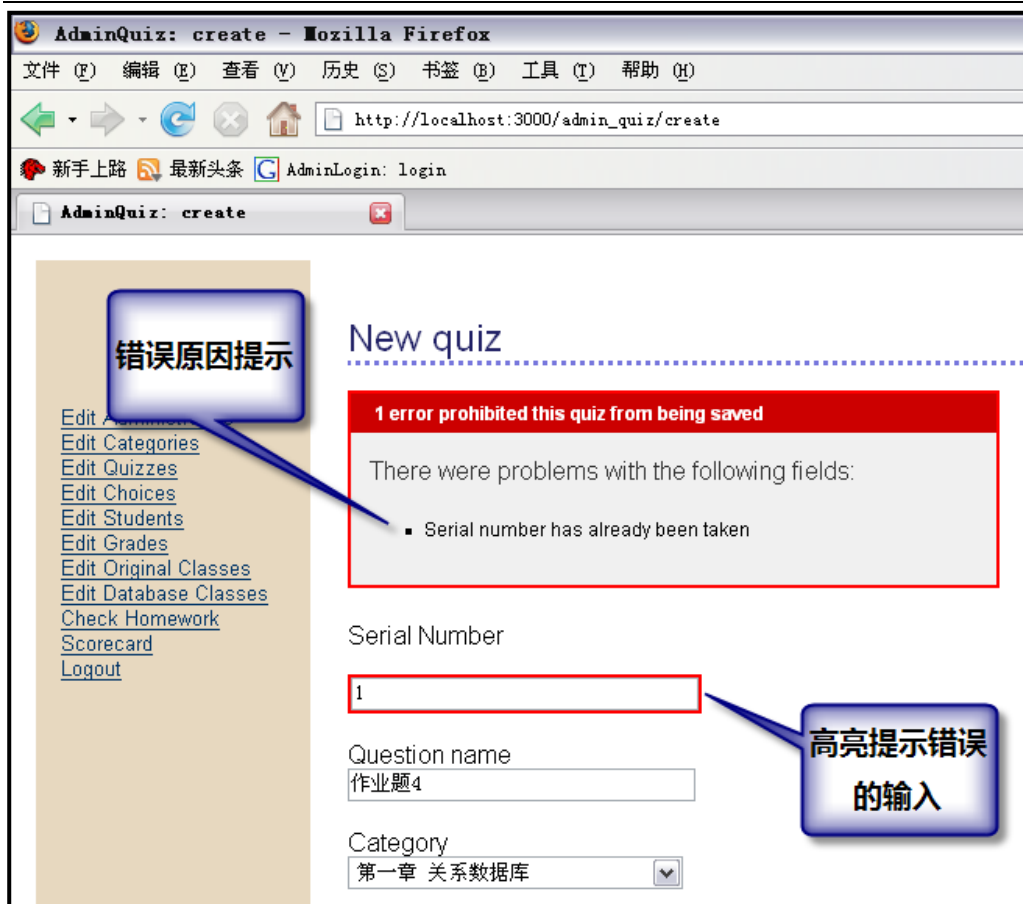


图 21 重复 serial number 错误提示

作业题与选择题相似，也利用 file_column 插件提供了图表显示功能，此功能的实现在在线答题模块已经详细阐述。

5.2.2 学生做题功能

学生上传的文件路径和作业相关信息全部保存在表 homeworks 中，在找作业题时通过键 Student_number（学号）和键 Quiz_serial_number（题目序列号）惟一确定。

学生登陆后可以通过左边的快速链接查看可做题目并选择作答，图 5.22 为某同学查看作业题截图，此同学当前的状态为没有做过任何题。

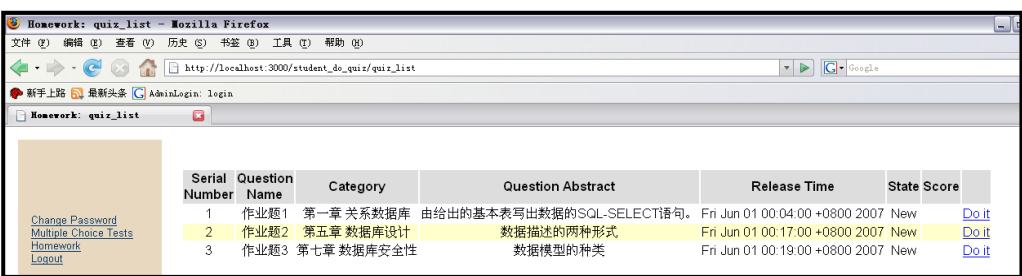


图 5.22 作业状态

列表表头从左至右分别为题目序列号，作业题名字，作业题所属章节，作业题摘要，发布时间，当前状态，分数。

其中状态（state）一栏用于显示此题相对学生的状态，如这里的三道题的state 都显示 new，就标识该生现在可做的题共有三道，目前一道未做。Score 一栏用于显示教师评定的成绩。因学生尚未做题，所以该栏为空。

学生在看好希望做的题目后可以点击 Do it 链接做题。该链接直接跳转到图 5.23 所示的做题页面：

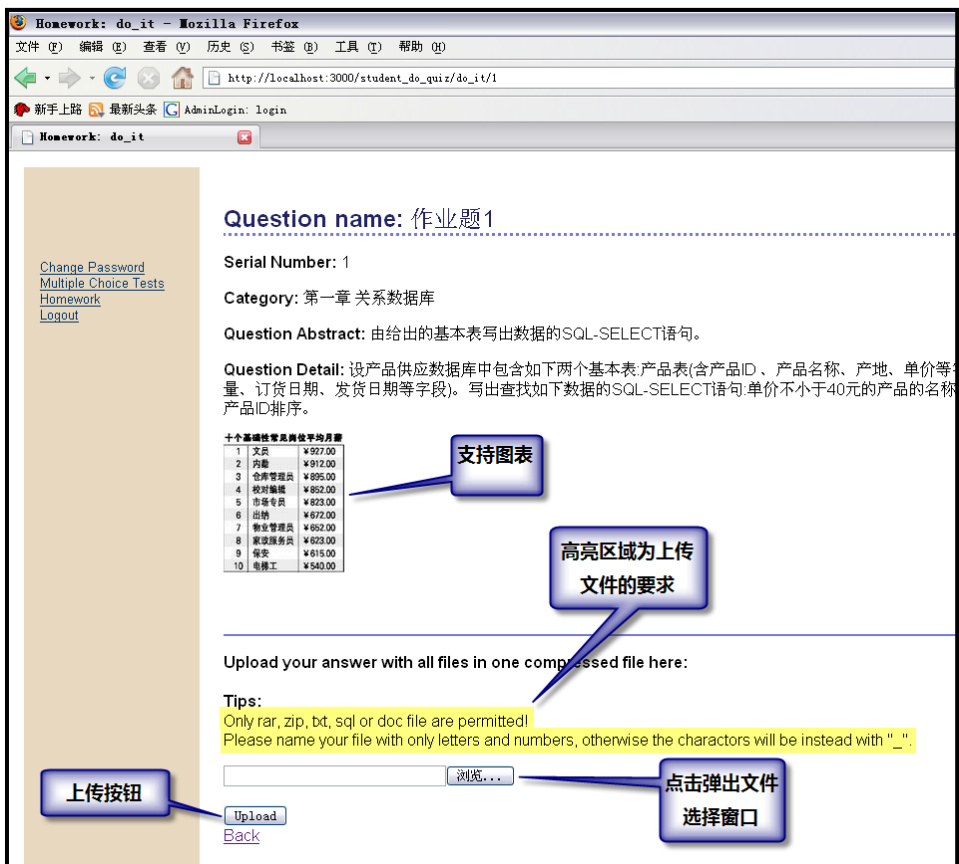


图 5.23 做题页面

为保证系统的安全和防止攻击，文件上传有一定限制。大小被限定在 0~2MB，上传文件的扩展名只允许是 rar, zip, txt, sql 或 doc（大小写无关），并且上传文件名只能是英文字母或数字，其它字母或或符号将被“_”代替。所有上传的文件都会被存放在服务器网站根目录的指定文件夹，是学生和老师只能在登录并拥有特定权限时可以通过按钮下载，而无法直接在浏览器网址栏输入 URL 或在下载工具中输入 URL 来下载。这样就防止了学生查看或下载其他学生已经上传的作业的行为。

下面的代码是 dbweb\app\models\homework.rb 的全部代码，主要就是规定了上传文件在本地的存放路径和对上传文件大小及格式的验证：

```
class Homework < ActiveRecord::Base
  file_column :file, :store_dir => "mei1984dan1124"
  validates_filesize_of :file, :in => 0.bytes..2000.kilobytes
  validates_file_format_of :file, :in =>
    ["rar", "zip", "txt", "sql", "doc", "RAR", "ZIP", "TXT", "SQL",
    "DOC"]
End
```

通过清晰的函数我们可以看到 file_column :file, :store_dir => "mei1984dan1124"规定了上传的文件存在于文件系统的 mei1984dan1124 目录下。根据 Ruby On Rails 的机制，这个目录无法通过 HTTP 直接访问而只能通过 form 的 action 调用地址并下载文件。validates_filesize_of :file, :in => 0.bytes..2000.kilobytes 一句规定了文件的大小最大为 2MB，这是文件上传时必须考虑的因素，因为如果有人恶意上传大文件将大大消耗服务器的空间并可能造成网络堵塞最终至服务器瘫痪。validates_file_format_of :file, :in => ["rar", "zip", "txt", "sql", "doc", "RAR", "ZIP", "TXT", "SQL", "DOC"]是对文件格式的限定，只允许同学提交有效的文件，其他扩展名的文件将无上传。

图 5.24 为上传文件时选择不被系统允许的格式时的截图：

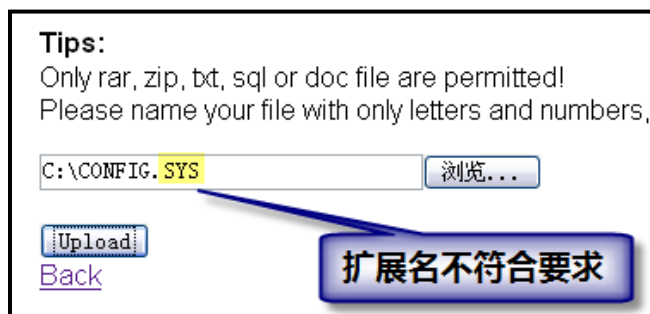


图 5.24 文件扩展名非法

点击上图的 Upload 按钮后会显示如图 5.25 上传文件不成功页面并提示错误:

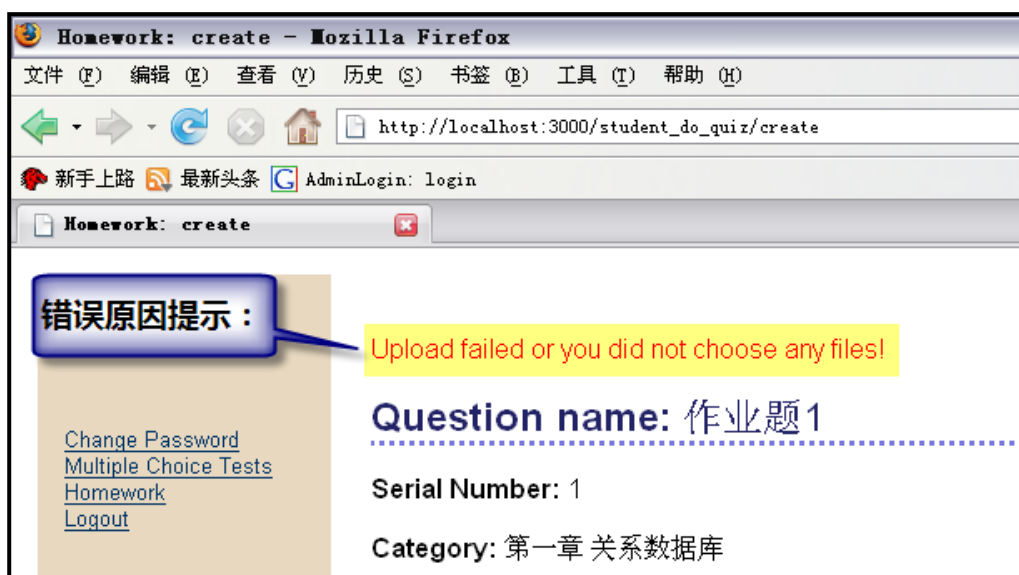


图 5.25 上传文件不成功错误的提示

只有学生选择正确类型的文件并大小在 2MB 之内才能被提交成功。图 5.26 所示即为学生提交成功后系统的通知:

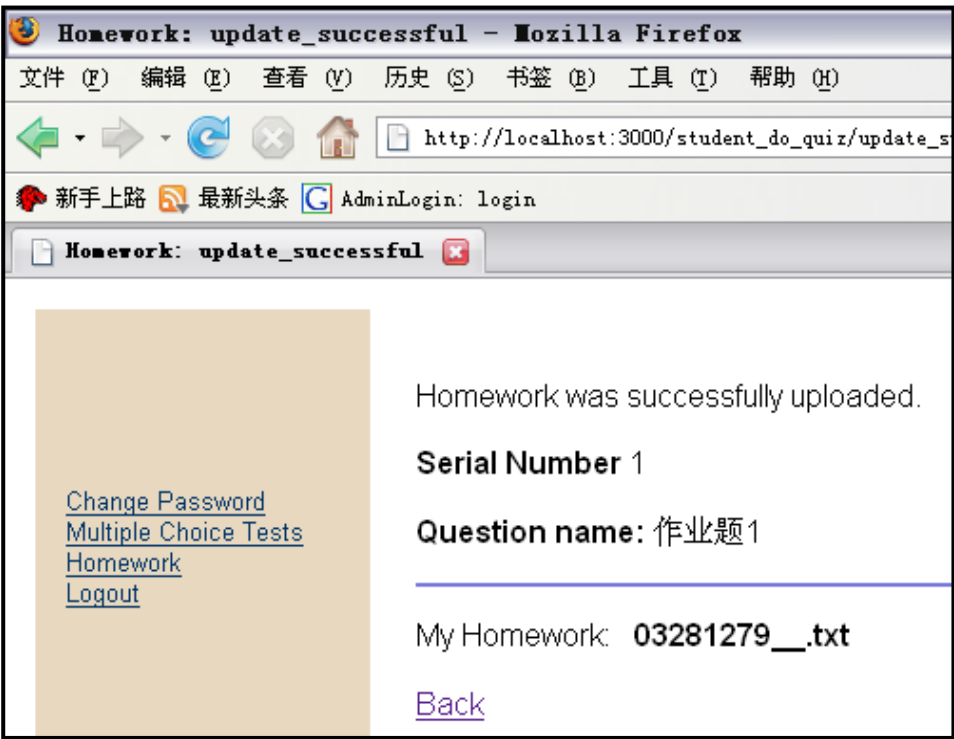


图 5.26 作业上传成功

系统把学生作业写入 homeworks 表的同时会更改学生的 students.state_of_quiz 相关标志位为 1，表示学生该作业已经提交，但尚未被批改。

5.2.3 学生更新作业功能

学生提交作业后如教师未对作业进行批改则仍可更新自己的作业，图 5.27 所示为作业已经提交，但教师未批改的学生端界面。

Serial Number	Question Name	Category	Question Abstract	Release Time	State	Score
1	作业题1	第一章 关系数据库	由给出的基本表写出数据的SQL-SELECT语句。	Fri Jun 01 00:04:00 +0800 2007	Submitted	Edit
2	作业题2	第五章 数据库设计	数据描述的两形式	Fri Jun 01 00:17:00 +0800 2007	New	Do it
3	作业题3	第七章 数据库安全性	数据模型的种类	Fri Jun 01 00:19:00 +0800 2007	New	Do it

状态显示作业
已经被提交

可以更改已提
交的作业

图 5.27 作业状态

学生点击 edit 后即可进入图 5.28 所示的更改作业界面：

Question name: 作业题1

Serial Number: 1

Category: 第一章 关系数据库

Question Abstract: 由给出的基本表写出数据的SQL-SELECT语句。

Question Detail: 设产品供应数据库中包含如下两个基本表:产品表(含产量、订货日期、发货日期等字段)。写出查找如下数据的SQL-SELECT语句
产品ID排序。

十个基础性常见岗位平均月薪

1	文员	¥927.00
2	内勤	¥912.00
3	仓库管理员	¥895.00
4	校对编辑	¥862.00
5	市场专员	¥823.00
6	出纳	¥672.00
7	物业管理员	¥662.00
8	家政服务员	¥623.00
9	保安	¥615.00
10	电梯工	¥540.00

提交最新版作业

Upload your **latest** answer with all files in one compressed file here:

浏览...

Upload

Back

图 5.28 更改作业

更新作业的 action 代码如下所示

```
def update
  @homework = Homework.find_by_id(params[:homework][:id])
  @quiz = Quiz.find_by_serial_number(@homework.quiz_serial_number)

  if params[:homework][:file] == ""
    $flash_coler = "color: red"
    flash[:notice] = "Upload failed or you did not choose any files!"
    render :action => 'edit'
  else
    if @homework.update_attributes(params[:homework])
```

```
        $flash_coler = "color: green"
        flash[:notice] = 'Homework was successfully updated.'
        redirect_to :action => 'update_successful',
        :h_id => @homework, :q_id => @quiz
    else
        $flash_coler = "color: red"
        flash[:notice] = "Upload failed or you did not choose
any files!"

        @quiz =
        Quiz.find_by_serial_number
        (@homework.quiz_serial_number)
        render :action => 'edit'
    end
end
end
end
```

此段代码的原理是首先判断学生是否有选择文件上传，如果学生没有选择文件则弹出 Upload failed or you did not choose any files!提示，并重新转到上传页面。如果有学生有选择文件并上传则用 @homework.update_attributes(params[:homework]) 对数据库进行更新，参数 params[:homework] 存储的是学生作业文件的路径和更新时间信息。如果更新成功则跳转到更新成功页面。

5.2.4 教师批改作业功能

全部学生提交的所有作业都可以在教师端进行查看并批改。未批改的作业 checked 标志位将显示 NO，表示未批改，批改后作业的 checked 标志位将显示 YES，表示已经批改过。图 5.29 显示的是当前有一位同学提交作业并尚未批改的界面：

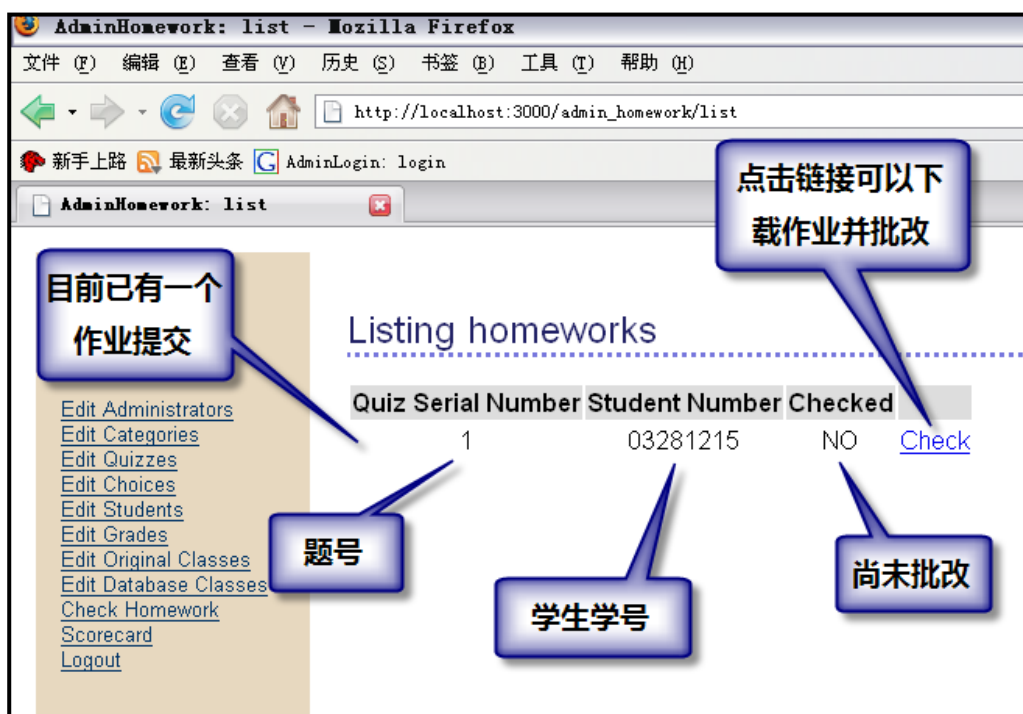


图 5.29 可批改作业列表

本机制通过调用 `dbweb\app\controllers\admin_homework_controller.rb` 的 `list` action，读取数据库内的全部作业并列表显示至视图完成。Checked 状态对应 `homeworks` 表的 `Check_flag` 列，相应标记和意义分别为 0—未提交 1—已提交 2—已批改。

点击 `check` 可以调用 `edit` action，通过这个作业题的 `id` 号，找到这个作业题并显示，如图 5.30 所示：

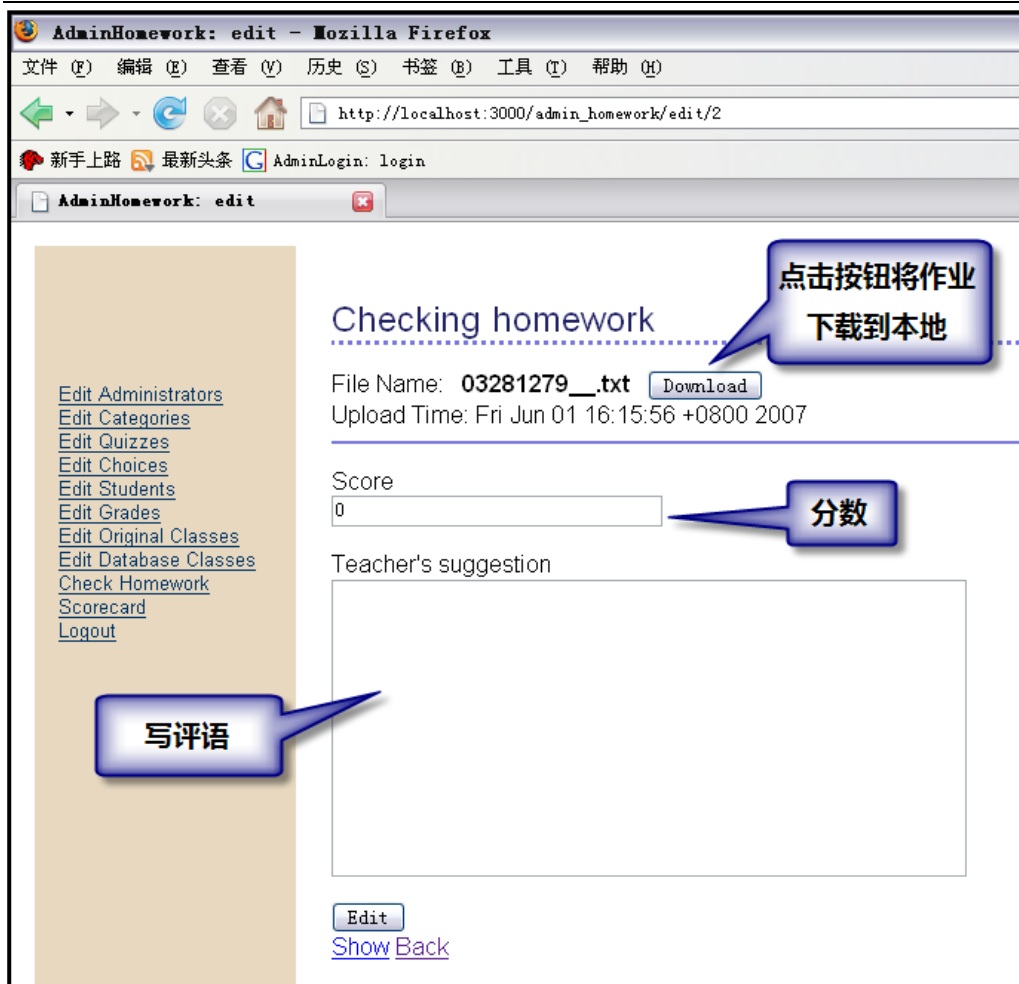


图 5.30 批改作业

题目和老师评语都是文本输入框，download 是通过 file_column 插件产生的文件下载链接，可以把作业下载到本地进行查看。

当老师查看作业并给出分数和评语后可以点击 Edit 按钮提交批改结果。按钮 Edit 调用 update action，将作业信息加入分数和评语并更新，核心代码如下所示：

def update

```
@updated_homework = Homework.new(params[:homework])
@homework = Homework.find(params[:id])
@homework.check_flag = 1
@homework.score = @updated_homework.score
```

```
@homework.teacher_suggestion=  
    @updated_homework.teacher_suggestion  
if @homework.update  
  
    begin  
        student = Student.find_by_student_number  
            (@homework.student_number)  
  
        student.state_of_quiz[@homework.quiz_serial_number] = "2"  
        student.save  
    rescue  
    end  
  
    flash[:notice] = 'Homework was successfully checked.'  
    redirect_to :action => 'show', :id => @homework  
else  
    render :action => 'edit'  
end  
end
```

上述代码在存储作业成绩的同时还更改了 students 表，学生相关行的学生作业状态信息位 students.state_of_quiz，把该位标志为 2，表示该学生这道作业已经被批改。

5.2.5 学生查看作业批改结果功能

教师批改作业后学生的界面就会显示作业已经被批改并可以查看教师的评语。查看成绩和教师评语的界面也可以下载自己的作业。图 5.31 显示学生的某作业被教师批改完后，学生作业列表的效果。

Serial Number	Question Name	Category	Question Abstract	Release Time	State	Score
1	作业题1	第一章 关系数据库	由给出的基本表写出数据的SQL-SELECT语句。	Fri Jun 01 00:04:00 +0800 2007	Checked	100
2	作业题2	第五章 数据库设计	数据描述两种形式	Fri Jun 01 00:17:00 +0800 2007	New	
3	作业题3	第七章 数据库安全性	数据模型的种类	Fri Jun 01 00:19:00 +0800 2007	New	

图 5.31 学生作业列表

系统在读取学生作业信息时看到该学生的这个作业的状态标志位已经标为 2，即表示批改过，责在 state 栏显示 checked 信息。分数栏为读取作业信息时发现作业已经被批改，故显示分数。学生点击 show 后可以查看老师的评语并下载自己的作业。

如图 5.32 所示

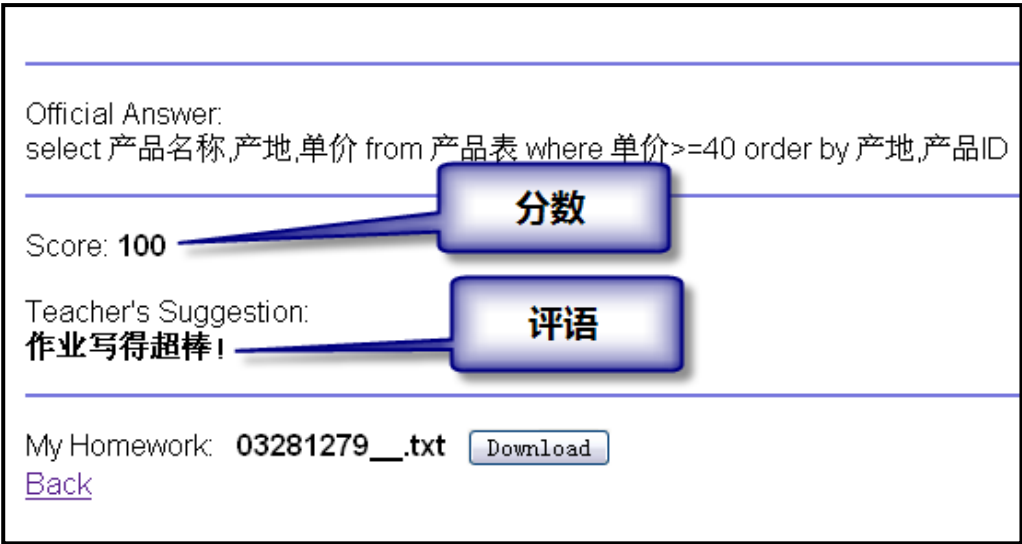


图 5.32 查看作业分数和评语界面

这个界面的信息全部由 homeworks 表的相关条目直接生成。

5.3 学生管理模块

教师端有管理学生的功能，点击左侧快捷链接中的 Edit Students，即可进行学生管理。通过学生列表的链接，可以进行新建学生，编辑学生，查看学生信息和删除学生等操作。图 5.33 为查看学生列表：

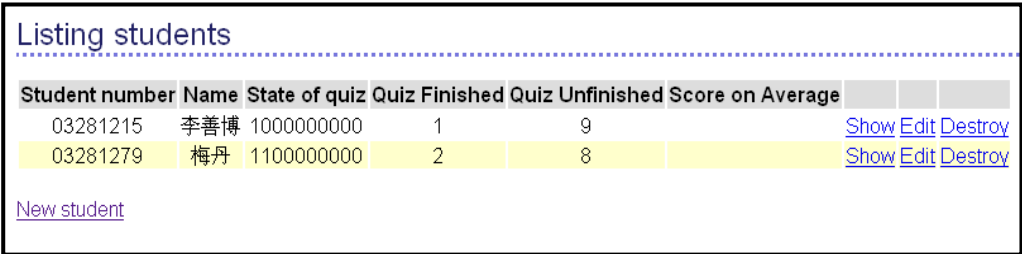


图 5.33 查看学生列表

里面分别有学生学号，姓名，题目状态，完成作业数，未完成作业数等信息。并可以通过后边的快速链接进行查看详细信息，编辑学生自然状况和删除学生操作。所有对学生的增加和删除操作都将触发学生表的触发器，使与学生相关的学生自然班列表，学生年级列表，数据库班列表的学生数目自动增减。关于触发器的实现将在第六章技术难点中进行详细讲解。

5.3.1 编辑学生基本情况

学生基本情况的编辑界面如图 5.34 所示：

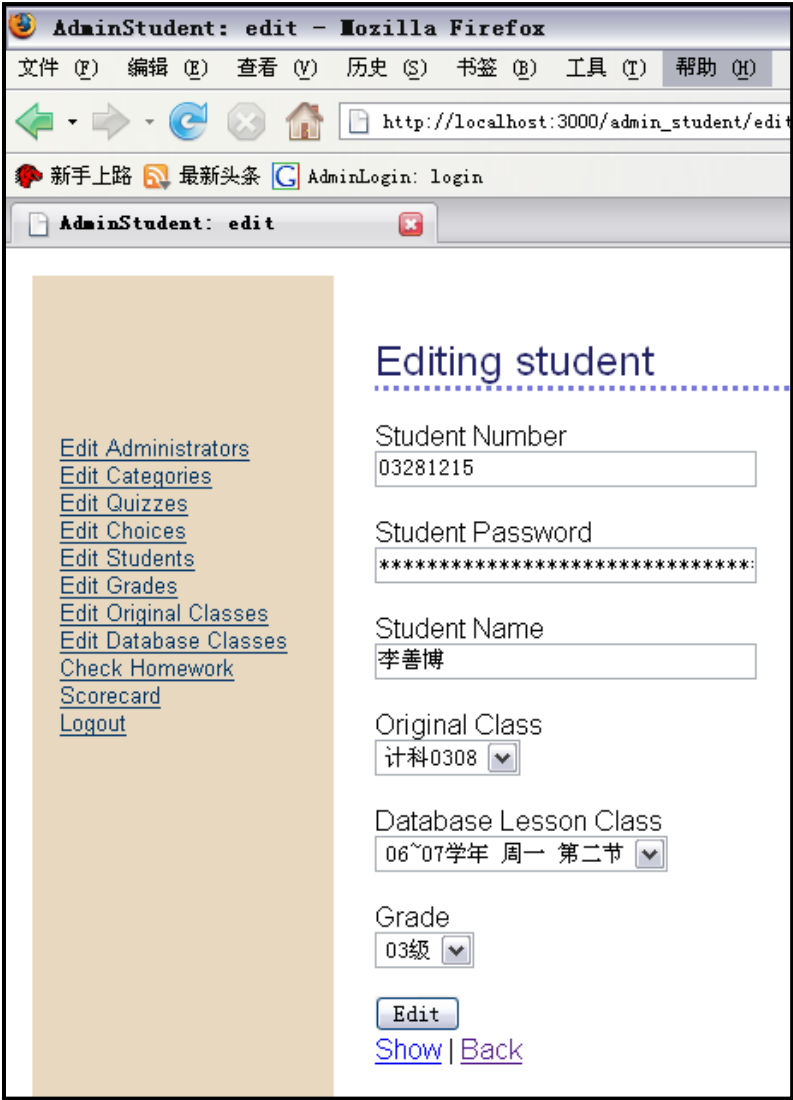


图 5.34 学生基本情况编辑

管理员可以更改学生的学号，密码，姓名，自然班，数据库班和年级等信息，其中学号，姓名通过输入框进行输入，密码框为加密属性，所有输入均用“*”代替。自然班，数据库班和年级智能下拉列表都是通过读取数据库的数据自动生成，不会出现某学生的班级不在班级列表之内等错误。这样做即提高了系统的稳定性同时也减少了管理员的手工劳动。

5.3.2 编辑数据库班

数据库班级列表用来分开不同的数据库课程班级，从图 5.35 可以看出，目前数系统内存储有两个头的数据库课程，分别是计算机学院和软件学院。

Listing dbclasses

Dbclass name	Num of quiz	Num of student			
06~07学年 周一 第二节 计算机学院	10	2	Show	Edit	Destroy
06~07学年 周二 第五节 软件学院	15	0	Show	Edit	Destroy

New dbclass

图 5.35 数据库班列表

从图中可以看到，在数据库课程班级中有 Num of Quiz 一项内容，该内容控制该数据库班的作业总数，下面将要讲到的计算平均成绩机制将用到该数据。

数据库班级的编辑和修改机制都与章节的编辑修改机制相似，该原理已在本章在线答题模块，教师端，编辑章节小节进行讲解，故不再重复。

5.3.3 编辑自然班

通过自然班列表，老师可以清楚地看见学生在各自然班的分布情况，如图 5.36 所示，目前数据库内有 8 个班级，每个班级的人数都一目了然。这里看到的学生人数并非人工输入，而是由系统自动生成。关于自动生成人数的功能介绍请详见第六章技术难点的触发器一节。

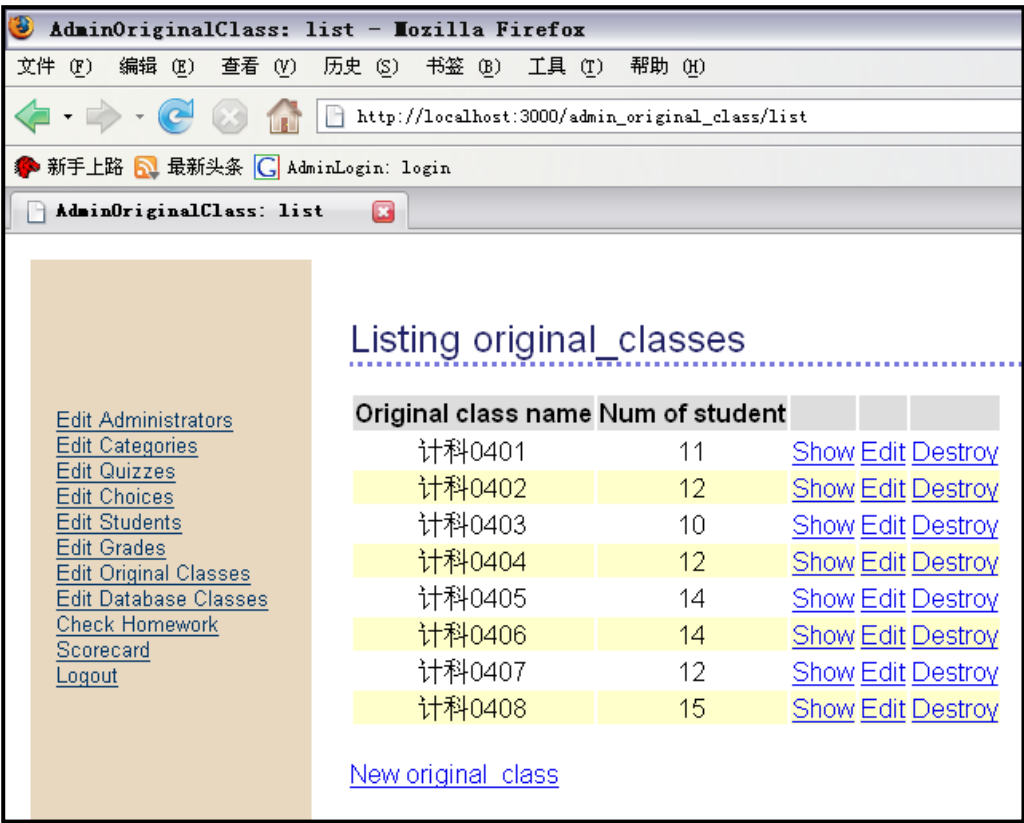


图 5.36 数据库自然班列表

自然班的编辑和修改机制都与章节的编辑修改机制相似，该原理已在本章在线答题模块，教师端，编辑章节小节进行讲解，故不再重复。

5.3.4 编辑年级

通过年级列表，老师可以清楚地看见学生在各年级的分布情况，如图 5.37 所示，目前数据库内有 7 个年级，各年级的人数可以通过该列表查看。这里看到的学生人数并非人工输入，而是由系统自动生成。关于自动生成人数的功能介绍请详见第六章技术难点的触发器一节。

[Edit Administrators](#)
[Edit Categories](#)
[Edit Quizzes](#)
[Edit Choices](#)
[Edit Students](#)
[Edit Grades](#)
[Edit Original Classes](#)
[Edit Database Classes](#)
[Check Homework](#)
[Scorecard](#)
[Logout](#)

Listing grades

Grade name	Num of student			
01级	0	Show	Edit	Destroy
02级	0	Show	Edit	Destroy
03级	0	Show	Edit	Destroy
04级	100	Show	Edit	Destroy
05级	0	Show	Edit	Destroy
06级	0	Show	Edit	Destroy
07级	0	Show	Edit	Destroy

[New grade](#)

图 5.37 年级列表

年级的编辑和修改机制都与章节的编辑修改机制相似，该原理也已在本章在线答题模块，教师端，编辑章节小节进行讲解，故不再重复。

5.3.5 查看成绩卡

查看成绩卡是本项目的重点，通过查看学生成绩卡，教师可以直观掌握当前学生的作业完成情况，也可以查看通过计算平均成绩按钮来计算学生的平均成绩。

列表的内容如图 5.38 所示，如某（X，Y）项为空白表示姓名为 X 的学生，第 Y 次作业没有做。如某（X，Y）项为 NG 表示姓名为 X 的学生，第 Y 次作业已经提交，但尚未批改。如某（X，Y）项为数字表示姓名为 X 的学生，第 Y 次作业已经提交并批改，数字为该次作业的得分。

各项内容均提供链接功能，如点击学生的名字即可进入该生的信息管理界面；点击 NG 项可以查看学生提交的作业并进行批改；点击数字项即分数，可以查看该次作业的分数和评语并下载该作业。

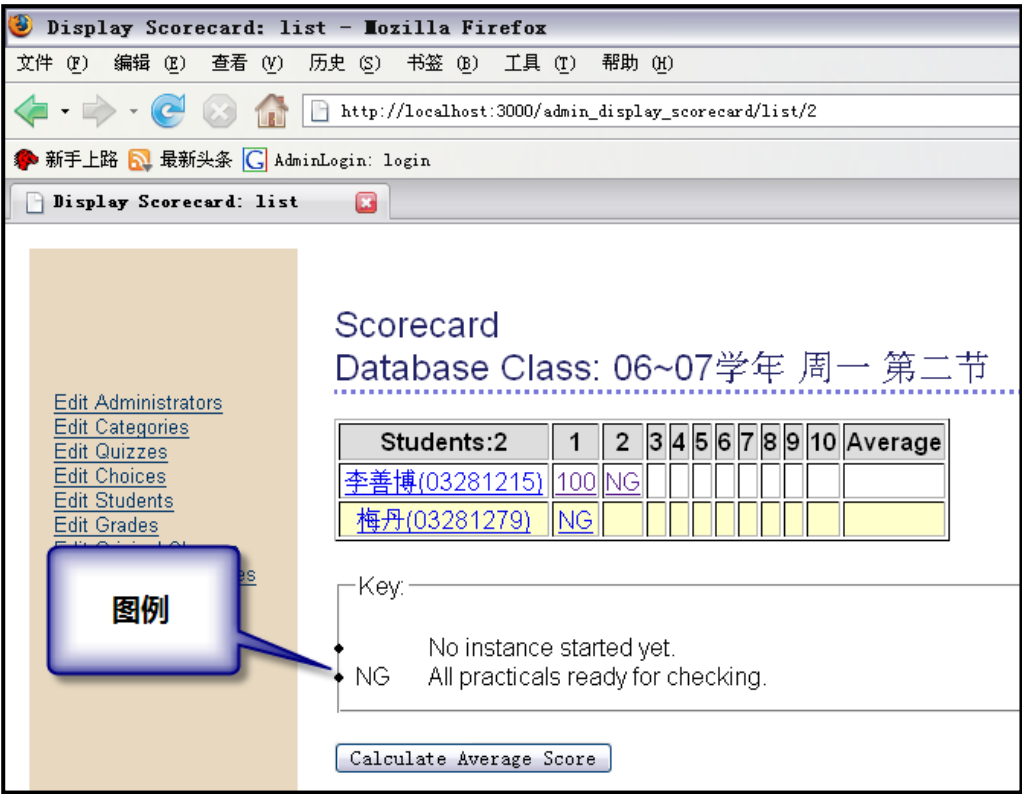


图 5.38 学生成绩卡

该页面的对应 action 通过循环迭代，列出各个学生的详细信息，其中作业数目由前面提到的数据库班的作业数目确定，状态由数组 `student.state_of_quiz` 的对应项确定，该项标志为 0 表示题目未做，该项标志为 1 表示作业已提交，该项标志为 2 表示作业已批改。生成信息项的核心代码如下所示：

```
<td>
<%=
case student.state_of_quiz[i,1]
when "0"
puts ""

when "1"
hw = Homework.find_by_student_number_and_quiz_serial_number
(student.student
_number,i)
```

```
link_to 'NG', :controller => 'admin_homework', :action => 'edit', :id =>
hw.id

when "2"
hw = Homework.find_by_student_number_and_quiz_serial_number
                                (student.student_
number,i)
link_to "#{hw.score}", :controller => 'admin_homework', :action =>
'edit', :id => hw.id
end
%>
</td>
```

5.3.6 计算平均成绩

计算平均成绩的核心代码如下所示：

```
def calculate_average
  @students = Student.find :all
  @homeworks = Homework.find :all

  ave = Hash.new
  for student in @students do
    ave["#{student.student_number}"] = 0
  end

  for homework in @homeworks do
    ave["#{homework.student_number}"] += homework.score
  end

  for student in @students do
    student.score_on_average=
                                ave[student.student_number]/student.dbclas
                                s.num_of_quiz
```

```

    student.save
end
flash[:notice] = "Calculate Finished!"
redirect_to :action => 'dbclass_list'

end

```

计算平均成绩的算法是这样的，首先，通过 `@students = Student.find :all` 和 `@homeworks = Homework.find :all` 两个语句取得全部学生的信息和全部作业的信息。然后新建一组散列表（hash 集合），然后建立一组以学号命名的键，并将起始值定为 0，用于存储平均成绩。接下来的一个循环是将数据库中作业列表的每一项成绩按照学号累加在相应学生的总成绩里。最后一个循环就是用学生的总成绩除以题目数而得到平均成绩。所有算完的平均成绩都会存储在学生列表的 `students.score_on_average` 项下面，读取学生信息时就可以从此处直接得到学生的平均成绩。

计算平均成绩后的截图如图 5.39 所示：

<h2>Scorecard</h2> <h3>Database Class: 06~07学年 周一 第二节</h3>											
Students:2	1	2	3	4	5	6	7	8	9	10	Average
李善博(03281215)	90										9
梅丹(03281279)	100	100									20

Key:

- No instance started yet.
- NG All practicals ready for checking.

Calculate Average Score

图 5.39 平均成绩显示

第一位同学因只做了一道题，并且得 90 分，所以他的平均成绩是 $90/10=9$ 分。第二位同学做了两次作业，并且都得 100 分，所以她的作业得分是 $(100+100)/10=20$ 分。

第六章 技术难点

本项目在开发过程中遇到很多技术难点，通过作者不断努力，终于将问题全部解决。本章将详细论述触发器，题目和答案的显示时间功能，安全性，选择题的特定答案顺序等功能的设计和实现。

6.1 触发器

6.1.1 触发器功能描述

应需求分析的要求，学生管理模块应提过自然班，年级，数据库班的查询功能。其中一个重要信息就是人数，因学生人数众多，若通过手工增减学生人数不但工作量大而且很容易出现错误。

6.1.2 触发器实现方法

作者为解决上述问题，特别在数据库的 `students` 表加入了多个触发器，通过触发器来控制学生的人数，控制学生数目增加核心代码如下：

```
begin
update dbclasses set num_of_student = num_of_student + 1
  where id = new.dbclass_id;
update grades set num_of_student = num_of_student + 1
  where id = new.grade_id;
update original_classes set num_of_student = num_of_student + 1
  where id = new.original_class_id;
end
```

这段触发器代码在 `students` 表有新内容插入时会自动调用，应用

new.dbclass_id, new.grade_id, new.original_class_id 三项的内容来确定该生的数据库班，年级和自然班，然后使三个表的相关项的 num_of_student 各加 1，实现数据的一致。

同理，下段代码实现了在学生被删除时三学生信息表学生人数的自动减少：

```
begin
update dbclasses set num_of_student = num_of_student - 1
  where id = old.dbclass_id;
update grades set num_of_student = num_of_student - 1
  where id = old.grade_id;
update original_classes set num_of_student = num_of_student - 1
  where id = old.original_class_id;
end
```

上面这段触发器代码在 students 表有数据被删除时会自动调用，应用 old.dbclass_id, old.grade_id, old.original_class_id 三项的内容来确定被删除的学生的数据库班，年级和自然班，然后使三个表的相关项的 num_of_student 各减 1，实现数据的一致。

图 6.1 为触发器的定义界面：

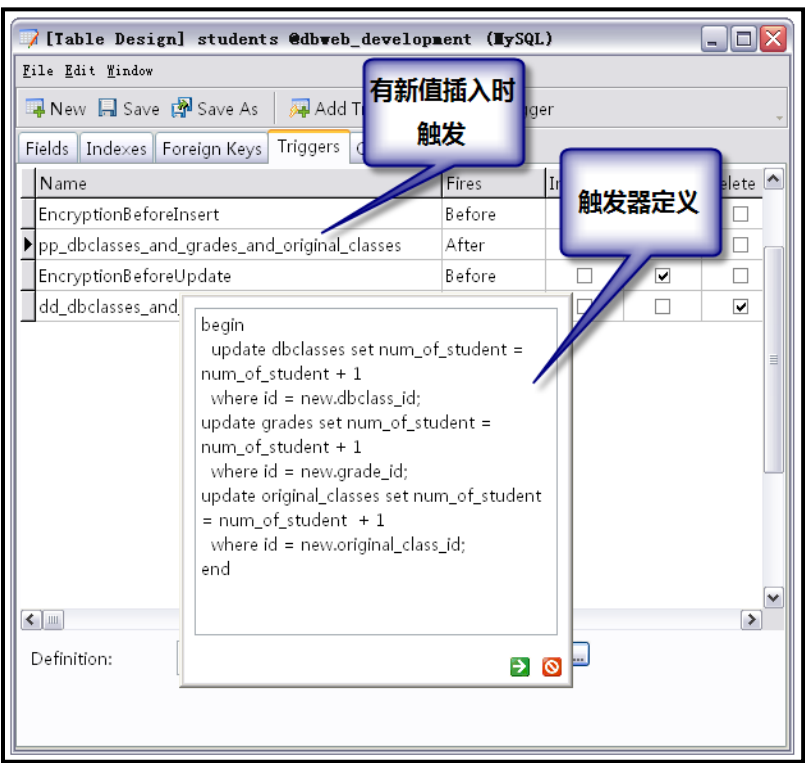


图 6.1 触发器

6.2 设置题目和答案显示时间功能

6.2.1 设置题目和答案显示时间功能描述

题目和答案的显示时间设置功能是本项目的难点，同时也是本项目的亮点。拥有题目和答案的显示时间设置功能的系统能够让教师只需出题时设置相应时间即可一劳永逸，无须后期管理。比如某教师在 6 月 4 日星期一上午有一节课，计划在课后留一道作业题，那么她可以在 6 月 3 日星期日晚上备课的时候就把题出好，并设置题目显示时间为 6 月 4 日中午 12 点，那么学生上午上完课回到寝室就可以看到题目在线上并进行作答了。该教师同时期望该周第次课，如 6 月 7 日第五节课讲解上节课布置的作业题，并希望课后学生回到寝室能够从网上看到详细答案。实现答案的定时显示只需要在出题的同时也就是 6 月 3 日设定答案显示时间为 6 月 7 日 18 点即可。也就是说教师只需要对题目进行一次编辑，就可以实现按计划发布作业和公布答案。流程图 6.2 直观的说明了这一便捷功能：

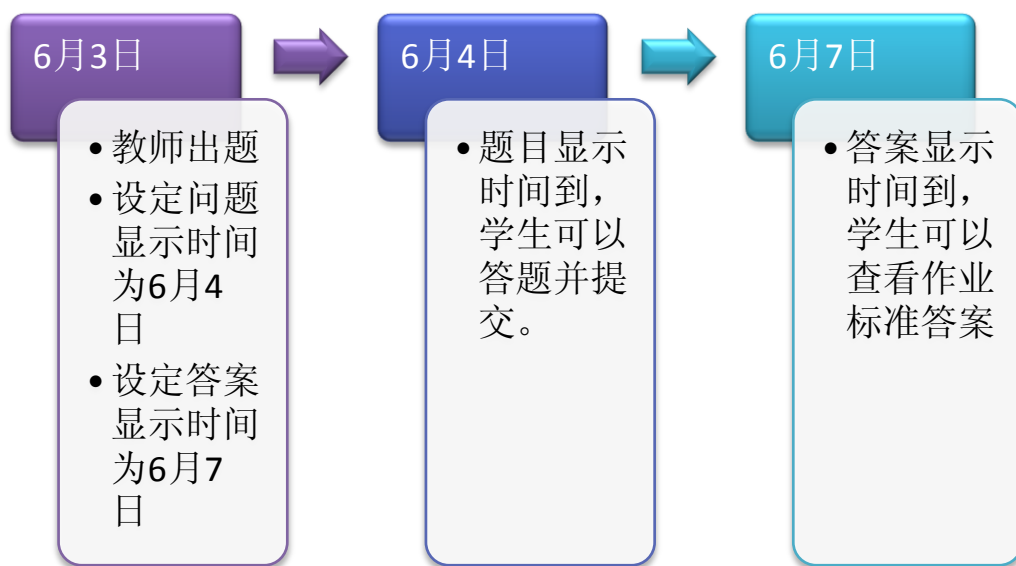


图 6.2 时间机制流程

图 6.3 所示为出题时设置题目显示时间和答案显示时间的功能窗口。其中设置的时间就是上面流程图涉及的时间。

The screenshot displays a web-based interface for managing questions. At the top, there is an 'Image' field with a '浏览...' (Browse...) button. Below this is an 'Answer' field containing a SQL query: `select 产品名称,产地,单价`, `from 产品表`, `where 单价>=40`, and `order by 产地,产品ID`. A blue callout bubble with the text '作业题显示时间' (Homework question display time) points to the 'Answer' field. Below the 'Answer' field, the 'Date question available' is set to 2007 June 4 at 12:00. The 'Date answer available' is set to 2007 June 7 at 18:00, with a blue callout bubble labeled '答案显示时间' (Answer display time) pointing to the '00' in the minutes field. At the bottom, there are 'Edit', 'Show', and 'Back' buttons.

Image

浏览...

Answer

```
select 产品名称,产地,单价
from 产品表
where 单价>=40
order by 产地,产品ID
```

作业题显示时间

Date question available

2007 June 4 12:00

Date answer available

2007 June 7 18:00

答案显示时间

Edit

Show | Back

图 6.3 题目显示时间和答案显示时间的设置

本题题目为“作业题 1”图 6.4 显示为 6 月 2 日学生登录时看到的可做题目，因上图的设置，此时“作业题 1”被隐藏，无法看到。

Serial Number	Question Name	Category	Question Abstract	Release Time	State	Score	
2	作业题2	第五章 数据库设计	数据描述的两种形式	Fri Jun 01 00:17:00 +0800 2007	New		Do it
3	作业题3	第七章 数据库安全性	数据模型的种类	Fri Jun 01 00:19:00 +0800 2007	New		Do it

图 6.4 学生 6 月 2 日登录时看到的可做题目

图 6.5 为学生看到的是学生 6 月 5 日登录时看到的可做题目：

Serial Number	Question Name	Category	Question Abstract	Release Time	State	Score	
2	作业题2	第五章 数据库设计	数据描述的两种形式	Fri Jun 01 00:17:00 +0800 2007	New		Do it
3	作业题3	第七章 数据库安全性	数据模型的种类	Fri Jun 01 00:19:00 +0800 2007	New		Do it
1	作业题1	第一章 关系数据库	由给出的基本表写出数据的SQL-SELECT语句。	Mon Jun 04 12:00:00 +0800 2007	New		Do it

图 6.5 学生 6 月 5 日登录时看到的可做题目

上图可以很明显看到，最后一行新多出了一道题目，这就是通过时间机制显示的题目。同时可以从列表看到，“作业题 1”的发布时间是 6 月 4 日中午 12 点。

图 6.6 为学生 6 月 8 日登录时看到的作业状态，此时可以看到答案已经公布了。

Official Answer:
select 产品名称,产地,单价 from 产品表 where 单价>=40 order by 产地,产品ID

Score: 100

Teacher's Suggestion:
好！！

My Homework: 03281279___.txt

[Back](#)

标准答案已经
可见

图 6.6 学生 6 月 8 日登录时看到的作业状态

6.2.2 设置题目和答案显示时间实现方法

此功能的实现原理是这样的，首先在 Quizzes 表定义两个 datetime 格式的键，分别为 Date_question_available（题目显示时间）和 Date_answer_available（答案显示时间），出题页面通过如下两段代码建立日期时间选择器，存储题目显示时间和答案显示时间。

```
<p>
  <label for="quiz_date_question_available">
    Date question available
  </label><br/>
  <%= datetime_select 'quiz', 'date_question_available' %>
</p>
```

```
<p>
  <label for="quiz_date_answer_available">
    Date answer available
  </label><br/>
  <%= datetime_select 'quiz', 'date_answer_available' %>
</p>
```

学生端通过在建立列表时只选择显示日期在当前日期之前的题目显示，并按题目发布时间排序。核心代码如下所示：

```
def quiz_list
  @student =
Student.find_by_student_number(session[:student_number])
  @quiz_pages, @quizzes = paginate :quizzes,
    :conditions => "date_question_available <=
now()",
    :order => "date_question_available ASC",
    :per_page => 10
End
```

其中第四行即为题目发布时间筛选功能。

同理，在学生查看作业的时候，在视图中设置判断语句，只有服务器时间在答案显示时间之后才能够显示答案。判断答案是否显示的代码如下所示：

```
<p>
  <% if @quiz.date_answer_available <= Time.now %>
  Official Answer:<br/>
  <%=h @quiz.answer %>
  <hr color="#77d">
  <%end%>
</p>
```

其中第二行<% if @quiz.date_answer_available <= Time.now %>即为判断答案是否可以显示。

6.3 访问控制

6.3.1 访问控制功能描述

由于本系统被设计为供全校学生使用的系统，故安全和访问控制问题必须重点考虑。本系统实现了高强度的访问控制机制。管理员的全系列操作功能只有登录后才能使用，同样，学生的功能只有通过才能使用，并且一个学生无法查看其他人的作业情况或下载他人的作业。数据库中所有密码均用 sha1 散列算法保存，即使数据库密码不幸外泄，黑客也无法得到用户们的原始密码。甚至管理员直接访问查看数据库也无法看到学生的原始密码。

6.3.2 访问控制实现方法

针对管理员和学生两类用户本系统有两套密码加密机制，管理员的加密机制十分复杂并由代码实现，学生加密机制相对简单但仍然很难破译，由触发器实现。

6.3.2.1 管理员加密机制

管理员的加密算法是这样的，新建管理员的时候要求输入的密码与管理员 name 不同，防止好事者试密码。输入后的管理员密码并非直接进行 sha1 加密，而是在管理员的密码后面加上系统一段特定标识代码，再加上系统随机生成的复杂随机数（salt），共三部分组成一个 String，然后将此 string 经 sha1 加密后存储在 administrators 表中。同时存储对应 salt，以备登录时验证。

腌制过的密码（即加了 salt 的密码）和不加 salt 的密码在强度上有很大的不同。腌制过的密码可以有效防止字典暴力破解。著名的 NT4 就是一个很好的例子，NT4 应用简单的 MD5 摘要算法存储密码，一旦用户黑入系统并把密码数据下载到本地就可以“翻开”他的字典进行破解了。互联网上到处都是 MD5 的字典库，如果用户使用的是常见单词或数字，那么有了密码文件的黑客通过一台超级计算机就可以瞬间得到用户的原始密码。而加入了 salt 的密码就显得不是那么容易，黑客需要把字典中的原始单词挨个加上 salt 并再次运行摘要算法，最后才能进行对比。这一步骤的工作量可要比仅仅做个循环比较大得多，如果用户数目很多，因为他们的 salt 都不同，那黑客就要多次重新建立密码字典来破解了。这可不是一名黑客希望看到的事情。

实现管理员密码加密的核心代码如下所示：

```
def self.encrypted_password(password, salt)
  string_to_hash = password + "MeiDan" + salt
  Digest::SHA1.hexdigest(string_to_hash)
End
```

其中 password 就是管理员的原始密码，中间的字符串“MeiDan”为特定标识代码，用于增加破解难度。Salt 的生成算法如下：

```
def create_new_salt
  self.salt = self.object_id.to_s + rand.to_s
end
```

从以上代码可知，salt 由系统运行时当前对象（即新建的这个 administrator 对象）的惟一 id 和一个随机数共同组成。这就保证了 salt 的随机性和唯一性。

系统验证时通过类似的加密过程加密管理员输入的密码，通过和数据库的 hashed_password 对比，来确定是否为合法用户。

6.3.2.2 学生加密机制

学生的加密机制相对较简单，全部加密过程在表内通过触发器实现，触发器代码如图 6.7 所示：

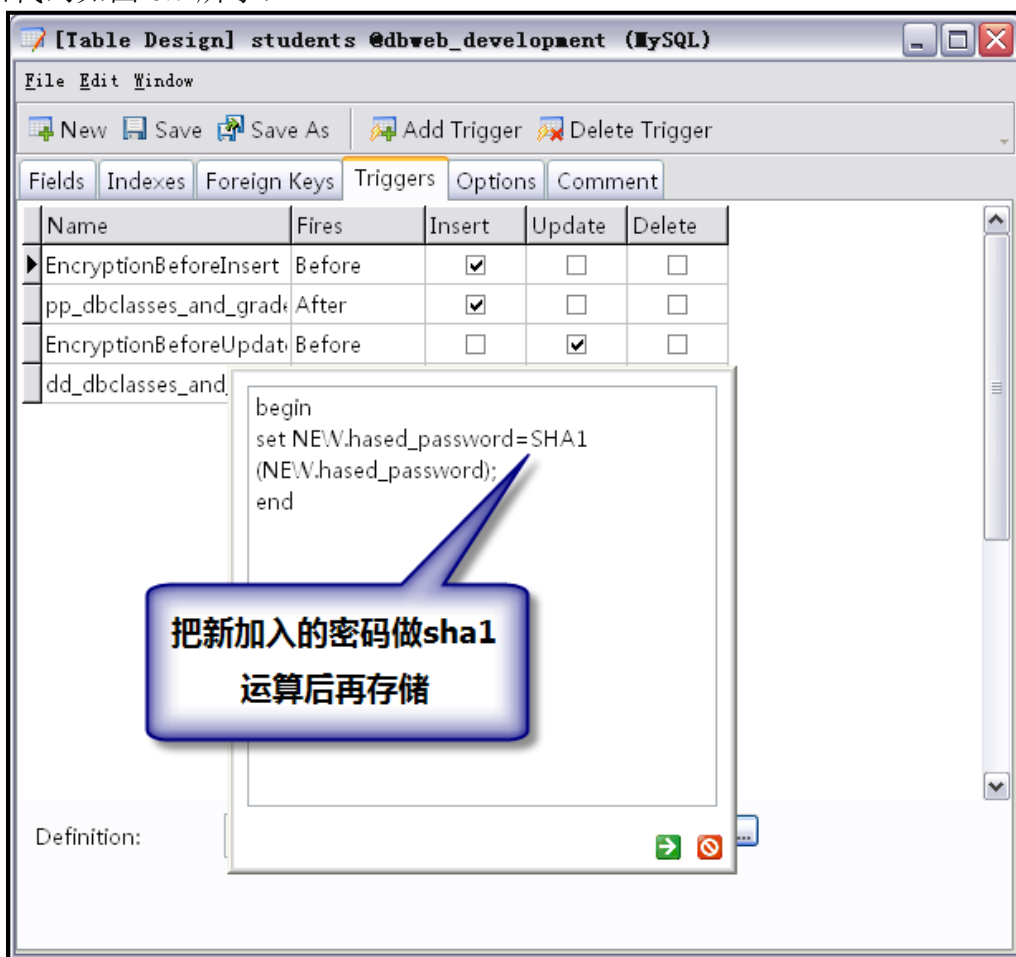


图 6.7 学生密码加密触发器

密码更新时也同样将新输入的密码进行 sha1 运算。学生登录时通过登录的 action 内部对输入的密码进行 sha1 运算，并与数据库内的 sha1 加密过的密码比较，如果相同即可通过登录认证。

6.3.2.3 action 过滤

每个管理员端的 action 第一行都是如下代码：

before_filter :authorize_administrator

类似的，每个学生端的 action 第一行都是如下代码：

before_filter :authorize_student

两个过滤器都是调用认证函数测试是否是合法用户，如果不是合法用户则跳转到登录框要求登录。

管理员或学生登录后都会通过 hash 在 session 内写入 id 或学号。如果是管理员，则会在登录成功后通过 session[:administrator_id] = administrator.id 一句在 session 中加入 administrator_id 变量，值为该管理员在 administrators 表内的 id。如果是学生，则会在登录成功后通过 session[:student_number] = student.student_number 一句在 session 中加入 student_number 变量，值为学生的学号。

教师和学生的过滤函数如下：

```
def authorize_administrator
  unless Administrator.find_by_id(session[:administrator_id])
    flash[:notice] = "Please log in"
    redirect_to(:controller => "admin_login", :action =>
"login")
  end
end

def authorize_student
  unless
Student.find_by_student_number(session[:student_number])
    flash[:notice] = "Please log in"
    redirect_to(:controller => "student_login", :action =>
"login")
  end
end
```

end

两个函数的第一句便是判断用户是否合法的代码，用户只有通过过滤才可以调用相关的 action，否则直接转到相应的登录页面。

6.4 选择题的特定答案顺序

6.4.1 选择题的特定答案顺序功能描述

为达到防止学生互相抄袭或交换答案的目的，选择题的选项都会随机出现。但随机出现并只永远都在随机永远都在变。而是同一个学生登录时看到的选项顺序始终不变，另一个同学虽然做相同的题，但选项顺序会完全不同。

图 6.8 显示为学号为 04281001 的同学做选择题时的选项页面：

2.

题干 2

☒ A.错 1

☐ B.错 2

☐ C.错 3

☐ D.对

3.

题干 3

☒ A.错 1

☐ B.对

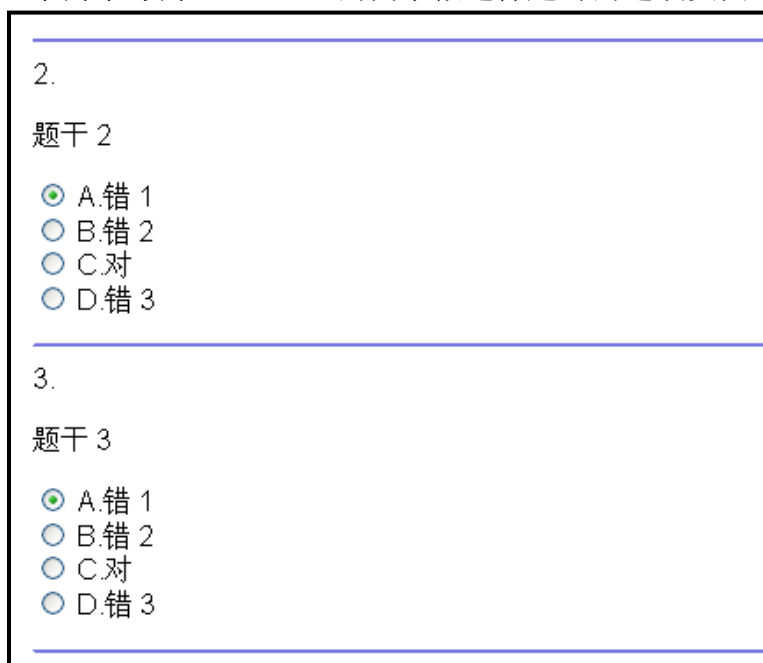
☐ C.错 2

☐ D.错 3

图 6.8 学号为 04281001 的同学做选择题时的选项页面

这位同学的第二题正确答案是 D，第三题正确答案是 B。

图 6.9 显示为学号为 04281005 的同学做选择题时的选项页面：



2.

题干 2

☒ A.错 1

☐ B.错 2

☐ C.对

☐ D.错 3

3.

题干 3

☒ A.错 1

☐ B.错 2

☐ C.对

☐ D.错 3

图 6.9 学号为 04281005 的同学做选择题时的选项页面

显然，这位同学两道题的正确答案都是 C。

这样就防止了学生偷懒，直接向同学要答案而得不到锻炼的情况。

6.4.2 选择题的特定答案顺序实现方法

系统生成选项时使用随机方法对选项进行排序，考虑到为了使同一名同学看到相同顺序的选项，特增加随机数种子为学生的学号加章节号。这样可以保证不同学生看到的答案选项顺序不同，同一名学生看到的选项顺序相同，并且不同章节的题目的选项顺序也不同的目的。

随机种子的代码如下：

```
<% srand session[:student_number].to_i + @category.id%>
```

具体的选项顺序分 4 种可能，分别是答案在第一项，答案在第二项，答案在第三项和答案在第四项。利用如下代码

```
<%randchoice = rand(4)%>
```

取得一个 0~3 之间的整数，如果 randchoice=0 则答案在第一项，如果 randchoice=1 答案在第二项，如果 randchoice=2 答案在第三项，如果 randchoice=3 答案在第四项。

第七章 项目前景

作为 Carnegie Mellon 大学的收费教学网站的取代者，我们任重而道远。数据库原理课程教学网站期待着老师们的认可和同学们满意。

在撰写这本论文时，本项目已经投入应用测试，并获得了预期的效果。下学期，即 07~08 学年上学期，本项目将正式投入运转，届时计算机学院和软件学院的全部数据库相关课程都将用此项目进行网上教学。同时，本项目也将作为数据库课程教改项目的一部分在学校立项，并参加评比。我们有理由相信，数据库原理课程教学网站必将成为数据库原理课程的重要部分。

项目开发之初我们就考虑将此项目转成商业化项目。整个数据库表格的关系清晰明了，没有使用任何专有项目或功能。稍加升级和扩展本项目即可成为一个公共教学网站平台，最终注册成商业产品并出售，为学校增加更多收入。

参考文献

1. 冯凤娟.《数据库原理及 Oracle 应用》清华大学出版社 2006
2. 冯凤娟.《SQL 与 PL/SQL 程序设计基础》清华大学出版社 2002
3. Dave Thomas, Chad Fowler, Andy Hunt. 《Programming Ruby: The Pragmatic Programmers' Guide, Second Edition》Pragmatic Programmers, LLC 2004
4. Dave Thomas, David Heinemeier Hansson. 《Agile Web Development with Rails》Pragmatic Programmers, LLC 2005
5. Ruby Programming Language <http://www.ruby-lang.org/>
6. Ruby in Twenty Minutes
<http://www.ruby-lang.org/en/documentation/quickstart/>
7. Ruby on Rails <http://www.rubyonrails.org/>

外文原文

Rest on Rails

<http://www.softiesonrails.com/2007/3/28/rest-101-part-1-understanding-resources>

Lesson 1 Understanding Resources

Square One

Let's begin at the beginning. How does a web browser work? Before trying to build websites with Rails, I used to think it was something like this:

- I type in a url into the address bar, or click on a link
- The browser makes an "http request" (whatever that means) and gets HTML code in response
- The browser renders the new HTML onto the page

And that was pretty much it. And I never really thought about how forms worked, either. I just figured they were a variant of the above.

In reality, the HTTP protocol includes exact instructions on how a browser should send its request to the server. HTTP is totally different from HTML, which happens to be the markup language chosen to represent the page content. It turns out that HTTP enables the browser to retrieve all kinds of information from the server, beyond simply getting an HTML page or submitting form data. In fact, HTTP actually allows for *eight different kinds of requests* to be made to the server. However, we are most familiar with these two kinds of requests:

- A "GET" request: get the contents of some resource on the web; the resource is uniquely identified by a URL;

- A "POST" request: send a package of data to a URL to create a new resource.

Square Two

See how I've slipped in the word "resource" when you weren't expecting it? Well, this is where my first, critical turning-point in thinking came in trying to understand REST: thinking of the web as a collection of resources, not "web pages". What do I mean by that?

Yesterday, I went on amazon.com and looked at some products I could buy. I also read some articles on Wikipedia. I read some news headlines on CNN. Then I checked the latest NHL standings to see just exactly how far Chicago is out of the playoffs.

If you want to understand REST, you have to stop thinking of these things as web pages. Let's take a Wikipedia article as an example. It's not just a web page. It's a resource - in my case, a short biography of Archimedes. I happened to use a web browser to access that resource, so the web browser requested an **HTML representation** of that resource, because rendering HTML is only way web browsers can display resources.

I know this is weird at first. Isn't the Archimedes article a web page? **No**. I *asked* for the HTML *representation* for a resource held by Wikipedia, a resource that can be identified by the URI `"//en.wikipedia.org/wiki/Archimedes"`. Wikipedia could have chosen to represent that resource in any number of ways - with a PDF, or perhaps as a .jpg picture. Maybe they offer all of those. But Firefox used a GET request that specified that the resource should be provided by the server as HTML, so that's what I got.

Another example: my plane reservation is a resource held by United Airlines. They let me get to that resource in many ways: as an HTML page, but they can also send my reservation information as a text message to my cell phone. Or they could send it to me in plain text in an email. Or I could call them up on this old device and they would tell it to me verbally. Same resource, multiple representations.

So I hope it's obvious now: my airline reservation is not just a web page. It's a real resource that (fortunately for me) can be rendered whenever I choose with HTML,

and I can use my browser to request that representation and render it onto my screen.

Square Three

Ok, once you can accept that the web is actually a huge collection of resources that can be rendered in multiple ways, and that an HTML rendering is just one of those, then there's one more concept I want you to grasp before we stop today. Resources aren't just single things like biographical articles, or an airline reservation, or a sports statistic. Some resources are *collections* of other resources. A list of holidays can be a resource. A list of friends can be a resource. A series of blog posts can be a resource.

Now that you understand what resources are, tune in next time when we talk about how HTTP enables the creation, reading, updating, and destruction of resources. (Rats, I think I just gave it away.)

Lesson 2 A Million APIs

Last time, I convinced you that the world wide web is a bunch of *resources*, not *web pages*. This time we will take another step closer to what REST is all about.

Object-Oriented Analysis and Design

If you've been doing object-oriented programming for some time, you probably learned to work like this:

1. Write a description of your problem domain - what your program is supposed to do.
2. The nouns are your classes
3. The verbs are methods on those classes
4. Have your nouns call methods on other nouns to accomplish a task

Sounds reasonable. That's how I did programming for a long time. This style of noun-verb programming is often called "RPC" style. I don't know what it has to do with "remote"-ness, but just know that "RPC-style" is the "classic" way of building

object-oriented software. It's actually a great way to develop software. But that's not how the web was designed to work. Why not?

Imagine it's 1992 (or just sometime before the web as you know it today). And suppose three different companies developed three different software applications: one to buy books, one to buy airline tickets, and one that can provide aerial photographs from any arbitrary distance above the earth. They each followed the classic object-oriented modeling techniques outlined above to develop their apps. And thinking that third-party software might someday want to interface with their software, they each implemented their own API as well.

Now suppose it became your job to write a tool that would front-end all three applications.

Ugh. You'd have to learn three different APIs. And I bet you would have UI controls that would correspond directly to the methods you can call in each API. But it sounds logical, right? I mean, how else could you do it? Besides, it's only three different APIs, and you know of some good books that give you some design patterns you could use to try to reduce the complexity (and look cool to your co-workers).

How about a million APIs?

Let's think about the browser again. When you install it on your computer, it has no foreknowledge of what websites you will want to visit. Yet it can communicate with any website you want. Not only that, you can take *actions* on those sites: you can buy music, book airline tickets, and view satellite images of your house.

Is it magic? Think for a moment how impossible things would be if each website had their own API. To buy a book on Amazon, the browser would have to know to call the `Amazon.Buy()` method. And it would need to call the `UnitedAirlines.CheckFlights()` method when I clicked a button that said "Check Flights" on it. You just can't build a tool that would have to talk to a million different APIs.

That's why the web wasn't designed from an RPC perspective. It was designed from a *REST* perspective.

Resource-centric Design

According to Wikipedia, REST stands for Representational State Transfer. Well... whatever. All it really means is that instead of having nouns with an arbitrary set of verbs attached to each one, every noun (from now on referred to as a *resource*) will have the same finite set of verbs. In other words, *every resource will expose the same API*. The essential methods in this API enable any client tool to:

1. Obtain a read-only view of a resource
2. Create a new resource
3. Update the attributes of a resource
4. Delete a resource

Wait a minute! Which one of these methods means "buy this book"? Which one means "search for flights from Chicago to New York next Tuesday"? Which one lets me zoom in from "city" level to "street" level on a map? And which one lets a user login to the site?

We'll find out the answer next time. But I think you might know the answer already, if you really can stop thinking about the "buy the book" *web page* and start thinking about what resource you might be *creating*. See you next time.

Lesson 3 RESTful Design

Last time, we talked about the differences between a web page and a resource. Now we need to understand how resource-based design will pay big dividends.

Repositories 'R' Us

From now on, think about your application as nothing more than a *resource repository*. What's a repository? Subversion is a good example of a *source code* repository. MySQL is a *relational data* repository. The NHL standings in the Chicago Tribune is a read-only repository of some hockey statistics. Can you think of some other examples?

At this point, I just need to pick a repository example we can use for the remainder of this series.

The Worst Example In The World

You work for SoftiesTransAir, and you get to build their software infrastructure from scratch.

- You need a rich-client, front-end tool to enter flight information. A flight is just the definition of an airplane going from one city to another at an assigned time).
- You want a web front end for airport employees to use. You want passengers to check their flight status from their cell phone.
- And your boss wants you to expose a nice XML-based API that third-party contractors can use to update those big arrivals/departures boards inside the airport.

Sounds like a fun project, right? Remember way back in part 1, we described traditional object-oriented design, and how one good way to get started is to look at a description of your problem domain and identify the nouns (which will be your objects) and the verbs (which will become methods). Just **think** of all the fancy design patterns you can use! You can't wait to draw your big UML diagram on the whiteboard and go to town with some interaction diagrams! Oooh, you can already imagine the `FlightSchedule.CancelFlight()` and `Flight.Delay()` methods.

Sorry Charlie. Since you decided to go with REST, most of your design is already done for you! In fact, the only thing left to do is to just identify your resources.

Let me say that again, just in case you didn't hear me over the squeak of your dry-erase markers, because it's kind of important.

Just identify your resources.

In our example, I can quickly identify some key nouns: airplanes, airports, and flights. I'll probably discover more along the way, but that's a good start for now.

How will client applications access each of these resources? That's what URLs were invented for. (By the way, you do know what the *L* in URL stands for, right?) The locator for each resource follows a simple pattern. For example:

-
- /airports is for the list of airports
 - /airplanes is for the list of airplanes
 - /flights is for the list of flights

and:

- /airports/ord is for Chicago O'Hare
- /airplanes/ZJ3543 is for the airplane identified by the serial number ZJ3543
- /flights/451 is for Flight #451

Four Methods

Ok, once you identify your resources, REST defines the four methods (or *verbs* in HTTP parlance) that each of your resources can implement:

- GET. Give me a read-only view of one instance of a resource from the repository.
- POST. I have new instance of a resource that I want to add to the repository.
- PUT. I want to update a resource that already exists in the repository.
- DELETE. I want to remove an instance of a resource from the repository.

Sound familiar? In Subversion, GET is like "checkout" or "update" command. POST is like the "add" command. PUT is like a "commit", and DELETE is like the "delete" command. In the MySQL example, GET is like doing a SELECT; POST is like INSERT; PUT is like UPDATE; and DELETE is, well, DELETE.

Wow. We know our resources and we know our methods. We can stop designing and start implementing.

Calling Those Methods with HTTP

Web applications use HTTP to let the client and server to talk to each other. Now we can tie up a loose end from Part 1. What really happens when you type a URL into your browser? The browser formats an HTTP message to call the GET method on the resource identified by the URL you gave it.

You can also call the POST method from your browser - but only if there's a <FORM> element on your web page. (Ok, not strictly true thanks to Ajax, but let's not go there right now). Forms can send data to a particular URL. When you submit a form, the browser calls the POST method, passing the URL *and also* some data representing the resource it wants to insert or update.

You can't call PUT or DELETE in your browser using HTML natives. Which is - as I'm told I like to say - *unfortunate*. Although every HTTP web server in the world is prepared to accept a PUT or DELETE method call, there's no standard way to make those calls with HTML. You have to invent your own way to signal to the server that you're trying to make a PUT or DELETE call.

Bottom line: You can map your nice REST API to nice HTTP verbs, but there are no HTML tags that will make the PUT or DELETE calls. The client and server have to agree ahead of time on some kind of convention they will use to fake the PUT and DELETE methods.

Lesson 4 Routing

Last time we learned that a REST design starts by identifying a few key resources. Today we need to talk about how Rails will route all of the REST-compliant requests, and how it expects you to handle those requests.

I want also take a step back for a moment, and mention that REST is not a Rails-only thing. REST was developed long before Rails, and is simply an approach that can be applied to many kinds of software applications. For example, now that I know about REST, I'll probably never write another WinForms app the same way again.

Resource == Controller

You already know about models and controllers. But how do you coordinate them to implement your newly-found RESTful design? First, let's get a few things out of the way:

- A resource is not always a one-to-one mapping to a Rails model or database table. Sometimes a resource is "virtual" and exists in your domain vocabulary and application logic, but is not backed by a database table.
- However, a resource is *always* mapped to a Rails controller.
- For you abstract-interface lovers out there (like me), you can consider your controller class to be the "concrete implementation" of your REST interface.
- Your controller will implement your resource regardless of which kind of client is talking to it (html browser, xml client, feed aggregator, etc.) As such, it will need to be able to respond accordingly, and send its responses and return values back to the client in the format that the client expects (html, xml, etc.) Back in part 3, I said we wanted different kinds of clients (web, cell phone, rich client, etc.) to be able to access the same resources. A single controller is responsible for rendering its resource, regardless of which client is making the request.

Seven

Most resources want to be accessible individually (an airport) and as a collection (the list of all airports). Your controller will have these seven actions (methods):

These four:

- **show** : this handles a GET request for the representation of one resource instance.
- **create** : this handles a POST request for creating a new resource instance.
- **update** : this handles a PUT request for updating an existing resource.
- **destroy** : this handles a DELETE request on a resource instance.

plus these three:

- **index** : this handles a GET request for a representation of the collection.
- **new** : this handles a GET request for a blank form useful for creating a

new resource. Usually makes sense only for HTML clients.

- **edit** : this handles a GET request for a form that is pre-filled with current values of an existing resource. Again, usually only for HTML clients.

The names of these methods are important, because the Rails routing infrastructure will be expecting them.

Of course, you can choose to implement fewer if you want. If your business rules say that a certain resource should only be rendered as read-only, then you might only need to implement the index and show methods, for example.

You may already be wondering how in the world you're going to develop a Rails application if those are the only actions you're allowed to have in your controllers. Well, first of all, don't worry. If you really, really think you have a situation that requires a few extra actions, then go right ahead and add them to your controller. Just remember that having any extra actions can be a warning sign that you haven't identified all of your resources yet.

The Traffic Cop

We need to connect some dots now. How do we map the URLs and HTTP verbs into method calls on our controller?

Normally, the action to be called in your controller is obvious just by looking at the url. I hope you remember how Rails normally translates incoming HTTP requests into calling methods in your controller code. If you don't remember, here's a five-second refresher course: your routes.rb file specifies how to translate URLs into controller classes and methods:

```
map.connect '/airports/:action/:id', :controller => 'airports'
```

This example maps a url like `www.mydomain.com/airports/open/45` to a controller class named `AirportsController` and a method named `open`. While the `open` method is running, `params[:id]` will yield `45`.

But to implement REST with the Rails framework, things would need to be done differently. First of all, we will want the same url to sometimes map to different

actions, depending on which HTTP verb has come along for the ride. In other words, this URL:

/airports/1

should map to our show action (for airport #1) if the HTTP verb was GET, but it should map to our destroy action if the HTTP verb was DELETE.

Prior to Rails 1.2, there wasn't a way to specify all that logic in the `map.connect` statement. Fortunately, 1.2 introduced a small but extremely important enhancement to the routing vocabulary. It's so cool, in fact, that unless you are aware of REST, you might have absolutely no idea how great this little enhancement really is to the Rails framework:

```
map.resources 'airports'
```

This code is short but it says a lot. It tells Rails all of these things:

- you have resources called *airports* (note the plural form, that's important)
- you want Rails to follow RESTful conventions and map incoming requests accordingly
- routing will not be based on URL pattern alone, but URL pattern *paired with* an HTTP verb
- it should route all requests to this resource through a controller named *AirportsController* (note, plural again)
- all URLs for this resource will follow a very specific convention (I'll describe those below)
- you want a set of pre-defined named routes for each action for this resource

The URL/verb pairs Rails will now automatically map for you look like this:

- /airports/ + POST = create
- /airports/1 + GET = show
- /airports/1 + PUT = update
- /airports/1 + DELETE = destroy
- /airports/ + GET = index
- /airports/new + GET = new
- /airports/1;edit + GET = edit

NOTE: That strange-looking semicolons are correct for Rails 1.2.3, but they will be changed to forward-slashes in Rails 2.0

Now all you have to do is implement the seven methods in your controller, and Rails handles the rest!

Don't Order Yet

But wait, there's more. There's a really neat generator that comes with Rails 1.2 and later called `scaffold_resource`, that will give a complete RESTful skeleton for your resource: it will add the routing code, a simple but functional controller implementing all seven methods, a migration for the underlying table, and RHTML templates for all of them! It's a great way to get started.

In fact, the best thing you can do to learn about REST is to start a fresh rails app and generate a resource. Create a database called `myapp_development` and `myapp_test` (for example), and then do this:

```
c:\dev> rails myapp  
c:\dev> cd myapp  
c:\dev\myapp> ruby script/generate scaffold_resource airport name:string  
designator:string  
c:\dev\myapp> rake db:migrate
```

Start up `ruby script/server` and go to `localhost:3000/airports`. You'll be able to create new airports, edit them, and delete them! How does it work?

Just open up `airport_controller.rb` - it's all right there. The Ruby code will be amazingly short. **Study the code.** It's very important that you understand the controller code before we move on.

Then, look at each of the `.rhtml` templates that were generated for you, and look for the use of named routes. Get a feel for how an incoming request gets routed to a controller action and which `.rhtml` will get rendered back to the client.

Now, there's probably one construct in the controller code that you haven't seen before: the strange-looking `respond_to` block. That's an important pillar of of our entire RESTful implementation, and we'll go into more details about it next time.

中文翻译

REST on Rails 指南

1: 理解资源

PART I

在理解 REST on Rails 之前，有必要先思考一下这个问题：浏览器是如何工作的？在开始使用 Rails 构建一个网站之前，我对这个问题是这么认为的：

- 首先我会在地址栏输入一个 URL，或者点击一个链接
- 然后浏览器会发送一个 HTTP 请求，并获取响应中的 HTML 代码
- 最后我会看到经过浏览器渲染的页面

就这么多，我甚至不知道 Form 是如何工作的，我觉得它跟点击链接没什么不同。

但在现实世界里，HTTP 协议有很严格的指令用于定义浏览器应该如何向服务器发送请求，HTTP 同 HTML 完全是两码事，HTML 只不过是一种用于表现页面内容的标记语言（Markup Language），而 HTTP 协议则允许浏览器从服务器获取各种类型的数据，HTML 只是其中之一。事实上，HTTP 协议定义了 8 中不同类型的请求，尽管如此，我们最熟悉的可能还是下面两种：

- GET，通过 GET 请求可以获取 Web 上的资源，每一个资源都由 URL 来唯一标识。
- POST，通过发送一组数据到特定的 URL 来创建一个新资源。

PART II

你可能对我使用“资源”（Resource）这个词感到迷惑不解，事实上我第一次看到这个词时也跟你一样迷惑，但这正是 REST 的精髓所在，在 REST 的世

界里，整个 Web 被看作一组资源的集合，而不是一张张的网页，这是什么意思呢？

昨天，我上当当买了几本书，又去维基百科查了几个词条，然后上新浪看了几条新闻，最后又在 NBA 网站上看了下骑士对马刺的比赛前瞻。

如果你想要理解 REST，那么你就要转变你的思维，不要再认为以上这些东西都是一张张的网页，让我们以维基百科为例，我查阅的 REST 词条事实上并不是一张网页，它是一个资源，我们使用 <http://zh.wikipedia.org/wiki/REST> 访问这个资源，并取得了它的 HTML 表示，之所以是 HTML，是因为浏览器只只是这种方式。

我承认这有些费解，<http://zh.wikipedia.org/wiki/REST> 怎么可能不是网页呢？事实上，确实不是，它是一个使用 URL 进行标识的资源，当我使用浏览器来访问它时，我得到了它的 HTML 表示，但维基百科可能还提供其他形式的表示，比如一个 PDF，一张 JPG 图片或者别的什么东西，而我之所以得到一个 HTML，是因为我的 Firefox 发送了一个 GET 请求，并明确的告诉了服务器，给我一个 HTML 表示。

再举个更浅显些的例子，比如我向南方航空定了张机票，他们可以通过 HTML 在浏览器中跟我确认，也可以发短信给我，或者发一封 Mail，当然也可以选择最稳妥的方式，打个电话告诉我。事实上就是一种资源，多种表示。

希望经过这番唠叨，你能够理解我所说的，我的机票订单不止是个网页，它是一个资源，当然我可以选择通过浏览器以 HTML 的方式来查看它。

PART III

一旦你接受了 Web 就是个巨大的资源集合，这些资源可以使用任意多的方式来表示，而 HTML 只是其中一种时，你离真正掌握 REST 已经不远了，但在结束今天的课程前，我还要在絮叨下：事实上，资源并不总是单个的东西，比如维基百科上介绍 REST 的文章，一张机票订单或者一堆 NBA 比赛数据，它也可以是一组资源的集合，比如中国传统节日列表，你最好的朋友等等，它们都是资源。

现在，你应该已经理解了什么是资源，在下一节，我们将讲解如何通过 HTTP 来创建（Creation），读取（Reading），更新（Updating）和销毁（Destruction）一个资源。

2: 无穷尽的 API

通过上一讲，我认为你树立了这个概念：即 Web 其实是一组资源而不是网页的集合（如果你还不这么认为，那请你先返回再次阅读第一讲）。这一讲我们将从另一个侧面来讲解为什么要有 REST？

面向对象设计与分析

如果你曾经学习过面向对象程序设计，那么你很可能会这样开始构建你的新程序：

- 首先，你需要定义你的问题域——你的程序要解决什么问题
- 然后，你会定义一个类，这个类的名字一般是名词
- 接着你会为这个类定义一些方法，方法的名字一般是动词
- 最用，通过调用其它类的方法，你的这个类顺利完成了它的使命

这看起来不错，事实上我曾经这么干了好多年，这种名词加动词的编程方法被成为“RPC”（远程过程调用），虽然我不明白那个 Remote（远程）是指什么，但 RPC 的确是构建面向对象软件的一个重要方法，不过这种方式却并不适合 Web 开发。

让我们回到远古，假设现在是 1992 年（或者 Web 出现之前的随便什么日子），假设有这样的三家公司，他们需要开发这样三种应用：书籍贩卖，机票贩卖以及 卫星地图浏览。并且他们都遵照了面向对象的设计思想，同时出于长远考虑，他们都认为总有一天会有第三方的软件需要同他们的系统进行交互，因此，他们都实现 了他们各自的 API。

现在，假设你的老板分配给你一个任务：为这三个系统设计一个统一的前端，你会怎么做呢？

我想你首先需要学习这三种完全不同的 API，然后为每一个 API 设计一个 UI 控件，当用户操作 UI 控件时，对应的 API 就会被调用，你可能会通过你学到的一些设计模式知识来简化你的工作量，并使你的代码看起来尽可能酷一些。

无穷尽的 API

当然这只是假设，但即使真的如此，在 Web 时代，你也不需要去学习那些无穷尽的糟糕 API，你所要做的就是你的电脑上安装一个浏览器，不是吗？浏览器对于你将要访问的网站一无所知，但它却能够准确的返回你想要的，你可以通过它购买音乐，预定机票，甚至从任意远的距离来欣赏你家的屋顶。

这很神奇，不是吗？但是让我们设想一下，如果每个网站都有它们自己的 API 会是什么样子？如果你想在 Amazon 买本书，浏览器必须知道如何调用 `Amazong.buy()`，如果你想查看航班信息，那么浏览器需要知道如何调用 `UnitedAirlines.CheckFlights()`，事实上，这样通吃所有 API 的程序永远也不可能被开发出来。

所以这就决定了 Web 不可能是 RPC 式的，它只能是 REST 式的。

以资源为中心的设计

那么 REST 究竟是什么呢？按照维基百科的解释，REST 是指 Representational State Transfer。这是什么意思呢，简单的说，就是现在每个名词都不再拥有它们各自独一无二的动词了，在 REST 的世界里，所有名词拥有的动词都是一样 的，并且数量也很有限。换句话说，也就是所有的资源都提供了一组相同的 API，这些 API 的实质就是允许随便什么客户端：

- 获取资源的某种表示
- 创建一个新资源
- 更新已存在的资源
- 销毁一个资源

等一下！那么究竟上面那个 API 可以让我“购买一本书”呢？搜索“下周二从纽约飞往洛杉矶的航班”又是哪个 API 完成的呢？

我们将在下一讲回答这个问题，但是如果你已经改变思维，不再认为”买书“就是一个网页，而是开始思考这其实是某个资源的创建，那么我想你其实应该已经知道答案了。

3: RESTful Design

通过上一讲，我们明白了为什么 Web 需要按照 REST 的方式来设计，而不是传统的面向对象编程的 RPC 方式，这一讲我们将通过一个实例来演示如何进行 REST 方式的 Web 设计，也就是让我们的设计变的 RESTful。

航空公司的需求

我们假设你为一家航空公司工作，你的任务是为他们设计一个航班管理系统，它的功能包括：

- 允许公司员工通过 **Web** 前端来输入航班信息。航班信息包括航班的起飞和降落城市，以及起飞时间等。
- 允许客户通过手机查询他的航班信息。
- 允许第三方通过我们提供的 **API** 来获取我们的航班信息。

很没有难度，不是吗？如果你是个急性子，你甚至可能都顾不上将你无懈可击的设计转换成 UML，就已经在你 IDE 的编辑框里输入了如下字符：

```
class FlightSchedule
def CancelFlight
.....
```

但是慢着，在 REST 的世界里，我们不再需要操心这些，我们需要做的只有一件事情：

定义你的资源！

是的，就这一件，因为 REST 已经为我们定义好了用于操作这些资源的方法。

在这个例子里，我们首先会想到这几个资源：airports, airplanes, flights。当然可能还会有其它，但就让我们先从这几个开始吧！

我们首先要做的就是为这些资源分配 URL，原则只有一个：尽可能的简单明了。

- /airports，通过这个 URL 可以访问所有的机场资源
- /airplanes，通过这个 URL 可以访问所有的飞机资源
- /flights，通过这个 URL 可以访问所有航班资源

还有：

- /airports/pudong，通过这个 URL 可以获取浦东机场的相关信息
- /airplanes/ZJ3543，通过这个 URL 可以获取编号为 ZJ3543 的飞机信息
- /flights/451，通过这个 URL 咋可以了解到航班 451 的起飞，降落城市已经起飞时间等信息。

方法已经准备好了

一旦你定义好了你的资源，整个设计也就完成了，因为，REST 已经为你准备好了以下四个方法（并且不再需要其它的了）：

- GET，获取资源
- POST，创建一个新资源
- PUT，更新已存在的资源
- DELETE，删除资源

通过 HTTP 调用这些方法

同样的，我们也不需要关心客户端如何来调用我们的方法，浏览器会帮我们搞定一切。

如果你仅仅只是在地址栏敲了个地址，然后按了下回车，浏览器会生成一条 HTTP 消息，并通过它来调用你输入的 URL 所代表的资源的 GET 方法。

如果你填写了一个表单，并点击了提交按钮，那么浏览器会把你填在表单中的信息组装成一条 HTTP POST 消息，并通过它来调用你想访问资源的 POST 方法。

但不幸的是，由于 HTML 的限制，目前你无法通过浏览器来调用资源的 PUT 和 DELETE 方法，不过这不重要，GET 和 POST 对我们已经足够了。

好了，我们的基于 REST 的设计就这么完成了，下一讲，我们将演示如何使用 Rest on Rails 来快速优美的实现我们的设计。

REST on Rails 指南 4：路由

通过上一讲我们了解到，RESTful 设计的关键就是定义系统中的资源，这一讲我们将学习在 Rails 中，如何将请求路由到我们的资源，以及我们应该如何处理它。

不过，有一点需要先说明：REST 并不是 Rails 的一部分，在 Rails 出现之前，REST 的概念已经存在很多年了，并且 REST 的应用也并不局限于 Web，事实上，它也可以应用到其它各种应用软件的开发中。

资源就是控制器

在我们正式开始之前，我们需要首先明确，在 Rails 中，资源和 model 并不总是一对一的关系，有时资源仅仅只是你应用逻辑中的一个实体的抽象，并不需要映射到你的数据库。但资源跟控制器总是一对一的，也就是每个资源都必须有一个与它相对应的控制器，并且你需要重新理解控制器，现在控制器只是 REST 接口 的具体实现，它的全部作用就是根据客户的请求返回资源的某种表示（HTML，XML 等）。

所以，就像第 2 章讲的，我们不再需要去设计那无穷尽的 API 了，现在我们的控制器只需要定义 7 个方法：

- show, 处理针对单个资源的 GET 请求
- create, 处理 POST 请求，并将创建一个新资源
- update, 处理 PUT 请求，并更新指定的资源
- destroy, 处理 DELETE 请求，销毁一个资源
- index, 处理针对多个资源的 GET 请求
- new, GET 请求，返回一个用于创建资源的表单，
- edit, GET 请求，返回一个用于更新资源的表单

Rails 会帮助我们将用户的请求路由到某个合适的方法，当然，你并不需要实现这全部的 7 个方法，如果你的系统不允许用户创建和修改资源，那么你只需要实现 `index` 和 `show` 方法就可以了。

不过更有可能的一种情况是你觉得这 7 个方法根本不够，你当然可以选择向控制器添加新的方法，但这其实是因为你的设计遗漏了一些资源，因为我建议，在你向控制器添加新方法之前，最好先重新考虑下你的设计。

方法已经定义好了，下一步的任务就是将用户的请求路由到指定的方法，在 `router.rb` 中，你可能会看到这样的路由：

```
map.connect '/airports/:action/:id', :controller = 'airports'
```

这条语句将映射 `/airports/open/45` 到 `airports` 控制器的 `open` 方法，你可以通过 `params[:id]` 获取 URL 中的参数 45。但是 REST 路由有些特殊，它需要同时考虑 URL 和请求的类型，因此同样是发往 `/airports/1` 的请求，如果是 GET 请求，它需要被路由到 `airports` 的 `show` 方法，而 DELETE 请求则需要被路由到 DELETE 方法。

不过幸运的是，从 Rails1.2 开始，我们不再需要通过 `map.connect` 来手动的配置 REST 路由，`map.resources` 会帮我们搞定一切：

```
map.resources 'airports'
```

这句话将创建如下的路由规则：

- 针对 `/airports/` 的 POST 请求将被路由到 `create` 方法
- 针对 `/airports/1` 的 GET 请求将被路由到 `show` 方法
- 针对 `/airports/1` 的 PUT 请求被路由到 `update` 方法
- 针对 `/airports/1` 的 DELETE 请求被路由到 `destroy` 方法
- 针对 `/airports/` 的 GET 请求被路由到 `index` 方法
- 针对 `/airports/new` 的 GET 请求被路由到 `new` 方法
- 针对 `/airports/1;edit` 的 GET 请求被路由到 `edit` 方法

注意：最后一条逗号分隔的 URL 看起来很丑陋，但它们在 Rails1.2.3 中是合法的，不过不用苦恼，它们将在 Rails2.0 中被去除

现在我们已经完成了 URL 的路由，下面我们需要做的就是实现这些方法：不过先别着急着码代码，从 Rails1.2 开始，我们有了一个新的生成器

(generator): scaffold_resource, 使用它我们可以很轻松的生成一个符合 REST 规范的 Rails 框架, 它包含:

- 资源所对应的 model
- 资源的 migration 文件
- 资源所对应的控制器, 控制器已经包含了 REST 所需的 7 个方法的实现
- 这 7 个方法所对应的 RHTML 文件
- 一条映射用户请求的路由

让我们仍然从第三讲的例子开始, 首先创建一个新程序, 然后为它添加一个 airport 资源:

```
D:\study>rails REST
D:\study>cd REST
D:\study\REST>ruby script/generate scaffold_resource airport
name:string designator:string
```

修改 database.yml 文件, 设置好你的数据库链接, 然后执行:

```
D:\study\REST>rake db:migrate
D:\study\REST>ruby script/server
```

现在定位你的浏览器到 <http://localhost:3000/airports/new>, 你应该已经可以创建一个新机场了, 是不是很神奇? 现在, 让我们来看看 airports_controller.rb, 所有的东西都在那了。

你应该会在控制器代码中看到一些奇怪的 respond_to 块, 这正是我们整个 REST 实现的关键所在, 我们将在下一讲详细探讨 respond_to 的细节。