数据库设计

课程名称：大学生社团管理

学　　院：计算机科学与工程学院

专　　业：软件工程　　班级：2014 级 3 班

设 计 人：张正锟　201401061038

山　东　科　技　大　学

2017 年　10 月　22 日

目 录

小组分工：

张正锟：需求分析、数据库设计、数据库实施与维护、高级程序设计语言系统实现、编写报告

1.概述

1.1 设计目标

- 熟悉数据库设计基本原理
- 熟悉数据库设计基本过程及方法
- 掌握数据库设计基本技巧及设计工具
- 掌握数据库设计相应的 SQL 语句操作
- 增强数据库建模能力和分析能力
- 通过大学生社团系统的实现增强特定领域的编程能力

1.2 设计要求

对大学生社团管理系统进行数据库设计，要包括以下内容：

- 分析大学生社团管理应包含的实体、实体包含的属性。分析实体之间的关系，如强制参与、可选参与等，实体数应不低于 6 个；
- 对实体中相应的数据项给出详细的数据字典描述，语义要合理；
- 以 PowerDesigner 为建模工具，对数据库进行逻辑设计，图中含实体、属性、多样性、实体联系、主键、外键等；
- 设计的关系模式需进行规范化处理，每个关系模式应能达到 3NF；
- 针对选定的系统设计不低于 20 个事务，涉及到检索和更新等，事务要合理；
- 绘制事务图，使用路径指示 ER 模型支持的用户事务；
- 根据逻辑设计原则（多样性），转化为相应数据表，并标明主键、外键等；
- 相应的事务要求用 SQL 语言实现，并用到触发器、完整性约束、存储过程、视图、索引等技术和方法，以及查询、插入和修改等数据操作；
- 应用自己熟悉的高级程序设计语言，实现系统主要功能。

2.需求分析

2.1 系统需求和必要性分析

通过对各大高校实地的调查可知，一般高校的学生社团信息管理主要是依据纸质和

手工作业处理，人工的对大量会员的基本资料进行档案式管理，此种处理方式数据量大，管理模式和方法滞后，存放时间不能长久和数据更新速度慢。

考虑现存的情况，建立一个系统化的学生社团管理系统是十分必要的。比如由原来的档案式保存会员信息变为将信息存入数据库中进行系统管理；利用海报或者板报宣传社团活动变为直接在网上发布社团活动的时间和地点等；文件式申请社团的创建变为规范格式的网上申请，提交表格，再由系统管理员审核、批准；此系统还可以方便会员对社团动态进行查看、为社团评价打分、进行留言和会员之间的交流等。

2.2 系统描述

学生社团管理信息系统是一个操作简单、使用方便的系统。它的建立既是为了更加高效、规范地实现对社团动态进行管理，又是为了方便用户及时查看社团信息，保证信息的时效性和高效性。

此学生社团管理信息系统应达到以下七个目标：

- 系统采用人机对话操作模式，界面设计简单大方，操作简单，效率高，安全性能高，同时便于维护和管理；
- 在登录界面，可根据界面中的权限来选择不同用户可以对系统调用不同的功能。
- 迅速发布社团动态，对社团活动申请、会费管理、校外社团合作进行规范化和程序化管理；
- 能够大量存储社团会员信息，方便会员查看社团信息、留言、进行评分等操作；
- 社团管理员可以通过后台登录，对社团信息、活动、留言板和注册的用户进行查看和管理，同时对用户加入社团的申请进行审核。
- 通过查询、添加、修改等操作，对社团信息、用户资料、财务管理等模块进行管理。
- 系统管理员可以修改个人密码，可以对社团的各项活动进行管理和监督。

系统采用 MySQL 数据库，开发语言为 Java，数据库的存储容量足够大，而且比较稳定，能够较长时间保存数据。

2.3 系统功能需求分析

根据系统描述可以初步总结出系统的基本功能。

系统开发任务树和功能模块图如下：

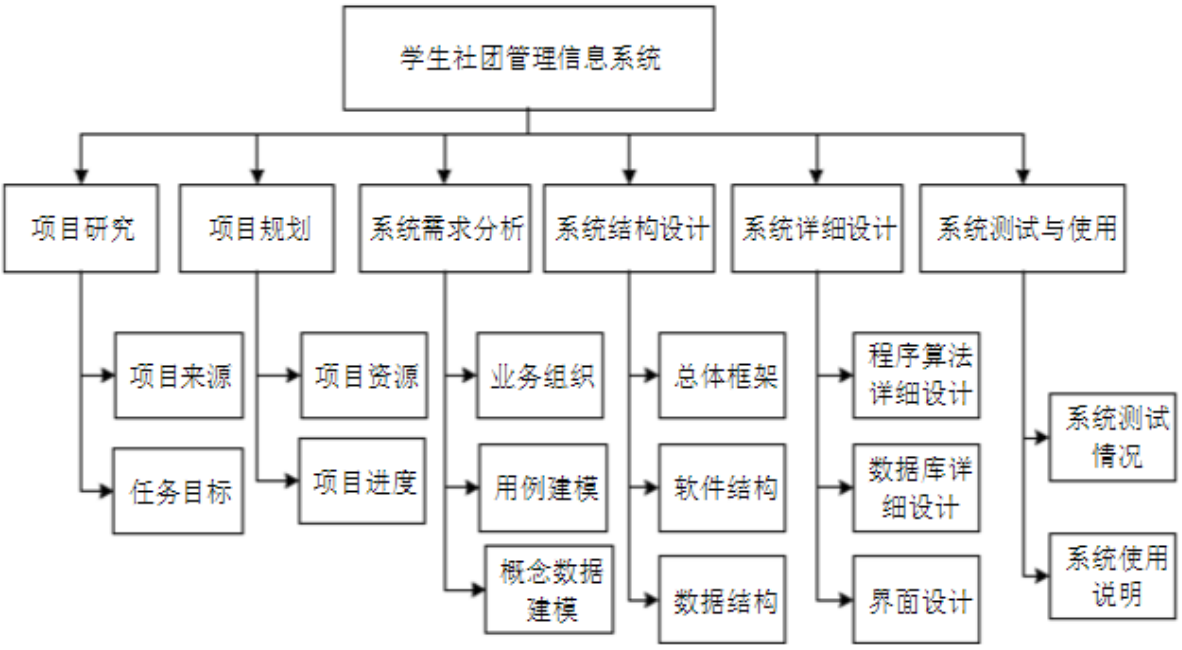


图 1 学生社团管理系统开发任务树

图 2 数据库设计阶段任务

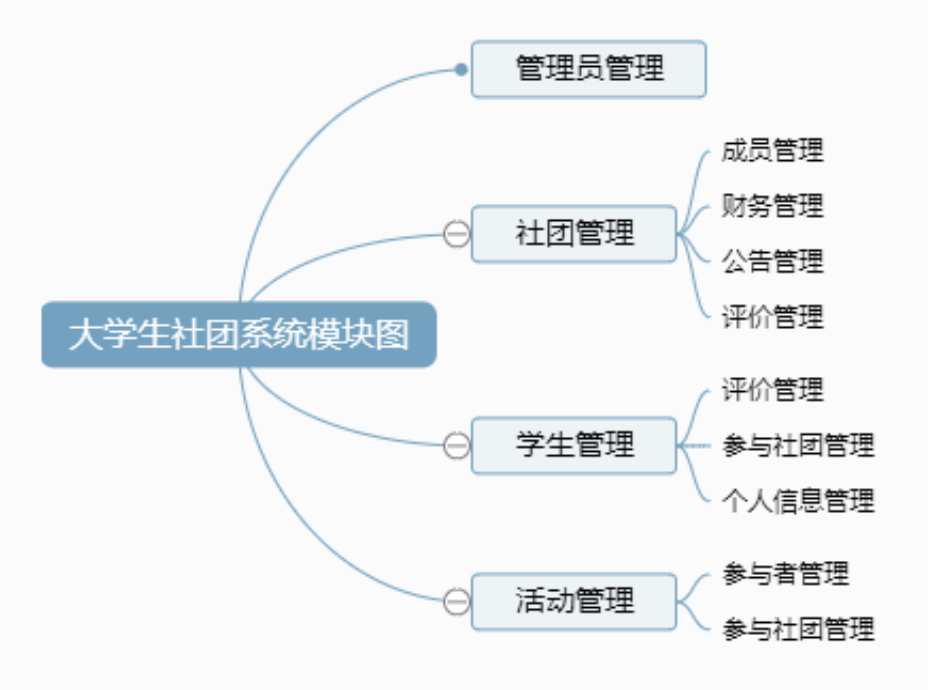


图 3 大学生社团系统模块图

2.4 数据字典

2.3.1 数据项

描述格式为：数据项描述={数据项名，数据项含义说明，数据类型，长度}

① 系统管理用户

- 编号描述={编号，系统用户的编号，int，11}
- 登录用户名描述={登陆用户名，系统用户的用户名，varchar，25}
- 登录密码描述={登录密码，系统用户登陆系统的密码，varchar，10}

② 大学学校学生

- 学生编号描述={学生编号，大学生的学号编号，int，11}
- 专业描述={专业，学生的专业班级，varchar，32}
- 地址描述={地址，学生的学校和家庭地址，varchar，128}
- 姓名描述{姓名，大学生的姓名，varchar，12}
- 密码描述={密码，大学生登陆系统查询信息的登录密码，varchar，32}

③ 社团

- 社团编号描述={社团编号，特定社团的编号，int，11}
- 社团名称描述={社团名称，社团的名字，varchar，32}

- 所属学院描述={所属学院，社团所属的学院名称，varchar，32}
- 社费描述={社费，加入社团需要交纳的费用，double，64}
- 社团建立时间描述={建立时间，社团开始运行的时间，tinyblob，32}

④ 活动

- 活动编号描述={活动编号，活动的编号，int，11}
- 地址描述={地址，活动开展的地址，varchar，64}
- 标题描述={标题，活动开展的标题描述，varchar，64}
- 内容描述={内容，活动开展的具体内容描述，varchar，256}
- 开始时间描述={开始时间，活动开展的开始时间，tinyblob，32}
- 结束时间描述={结束时间，活动开展的结束时间，tinyblob，32}

⑤ 社团评价

- 评价编号描述={评价编号，社团评价的编号，int，11}
- 评价星级描述={评价星级，对社团评价的星级，int，11}
- 时间描述={时间，做出评价的时间，tinyblob，32}
- 内容描述={内容，评价的具体内容描述，varchar，256}
- 用户编号描述={用户编号，做出评价的用户编号，int，11}
- 社团编号描述={社团编号，评价针对的社团编号，int，11}

⑥ 社团公告

- 社团公告编号描述={社团公告编号，某一条社团公告记录的编号，int，11}
- 社团公告标题描述={社团公告标题，某一条社团公告纪录的标题，varchar，32}
- 社团公告详情描述={社团公告详情，某一条社团公告纪录的详情，varchar，255}
- 社团公告发表时间描述={社团公告发表时间，某一条社团公告纪录的发表时间，datetime，10}
- 社团公告发表社团编号描述={社团公告的发表社团编号，某一条社团公告的发表社团的编号，int，11}

2.3.2 数据结构

描述格式为：数据结构描述={数据结构名，含义说明，组成:{数据项或数据结构}}

① 系统管理员用户数据结构描述={管理员用户，管理员用户，组成：{编号 登录用

户名 登录密码}}

② 大学学生结构描述={学生，学生，组成：{专业编号 姓名 性别 密码 学院 专业}}

③ 社团结构描述={社团，社团和成员活动等信息，组成：{社团编号 社团名称 所属专业 建立时间 社团费 社团成员 社团活动 社团公告 社团评价}}

④ 活动结构描述={活动，社团所举行的活动描述，组成：{活动编号 活动标题 活动内容 活动开始时间 活动结束时间 活动地址 活动参与社团}}

⑤ 社团评价结构描述={社团评价，对社团的评价信息管理，组成：{社团评价编号 评价星级 评价内容 评价时间 做出评价的学生 被评价的社团}}

⑥ 社团公告结构描述={社团公告，社团发出的公告信息管理，组成：{社团公告编号 公告标题 公告内容详情 公告发出时间 公告发出社团}}

2.5 事务需求分析

根据系统需求设计了 31 个事务：

① 数据输入事务

- 添加社团信息
- 添加社团评价信息
- 添加社团公告信息
- 添加社团
- 添加社团活动项
- 添加社团用户
- 添加系统管理者
- 添加普通大学生用户

② 数据更新删除

- 更新/删除社团信息
- 更新/删除社团评价信息
- 更新/删除社团公告信息
- 更新/删除社团
- 更新/删除社团活动项
- 更新/删除社团用户

- 更新/删除系统管理者
- 更新/删除普通大学生用户

③ 数据输出事务

- 管理员查询所有社团活动
- 管理员查询某个社团所拥有的活动
- 管理员查询某活动对应的所有参与社团
- 管理员通过起止时间查询社团活动信息
- 管理员查看某社团的社团评价信息
- 管理员查查看特定星级的社团评级信息
- 管理员查看某用户所有发表的评价信息
- 管理员查看所有社团基本信息
- 管理员查看某社团的用户列表
- 管理员查询所有管理员信息
- 管理员查询某社团的公告
- 管理员查询某学生所参加的社团
- 学生查看所有的社团基本信息
- 学生查看所有社团的评价信息
- 学生查看加入社团的公告

3．概念结构设计

3.1 实体关系

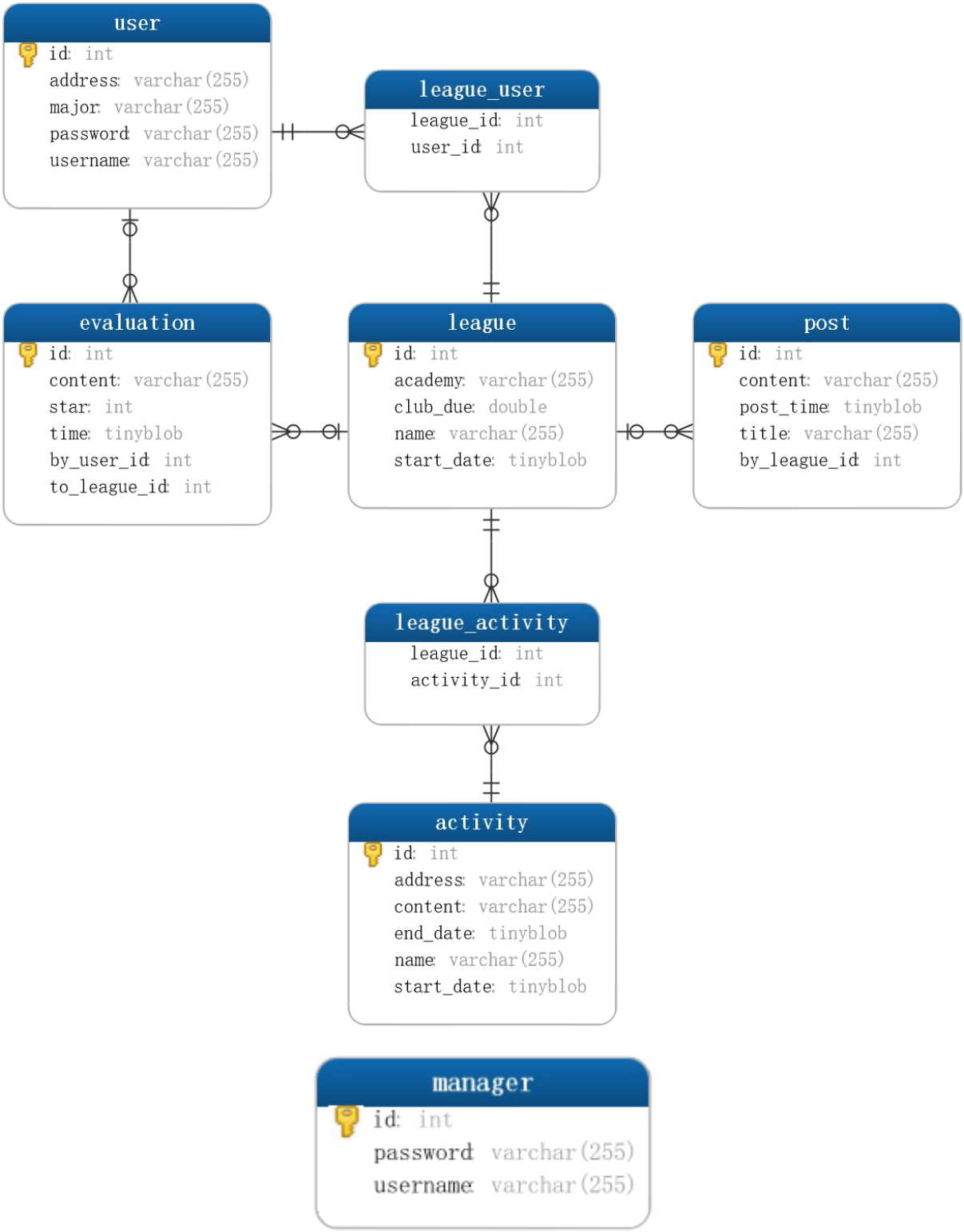
- 一个社团会有多个活动，一个活动会有多个参与社团，所以社团和活动的关系是 N:N
- 一个社团评价只针对一个社团，一个社团会有多条评价，所以社团和评价的关系是 1:N
- 一个社团评价只能是一个人提出，一个人可以提多条评价信息，所以社团评价和用户的关系是 N:1
- 一个社团公告只可能属于一个社团，一个社团可能会发表多条公告，所以社团和公告的关系是 1:N

- 一个学生可能参加多个社团，一个社团可能有多个学生，所以社团和学生的关系是 N:N

3.2 实体间依赖关系

- 添加一条社团评价时，必须指定做出评价的学生，同时必须制定评价针对的社团
- 添加一条社团公告时，必须指定做出公告的社团
- 添加一条社团参与记录时，必须指定学生，同时必须指定对应社团

3.3 E-R 图



4．逻辑结构设计

4.1 关系模型

1:N 关系的转换方法：在 N 端实体集中增加新属性，新属性由联系对应的 1 端实体

集的码和联系自身的属性构成，新增属性后原关系的码不变。具体 E-R 图转换为关系模型如下：

- 大学生：（学号，专业班级，登录密码，姓名，地址）
- 管理员：（管理员编号，登陆用户名，密码）
- 社团：（社团编号，社团所属学院，社团费，名称，建立时间）
- 活动：（活动编号，活动地址，活动名称，活动内容，开始时间，结束时间）
- 评价：（评价编号，评价星级，评价内容，评价时间，评价用户编号，针对社团编号）
- 公告（公告编号，公告标题，公告内容，公告发表时间，公告发表社团编号）
- 活动参与（活动编号，社团编号）
- 社团用户（社团编号，学生学号）

4.2 规范化

数据库逻辑设计的结果不是唯一的。为了进一步提高数据库应用系统的性能，还应该根据应用需要适当的修改、调整数据模型的结构，这就是数据模型的优化。数据库的设计范式是数据库设计所需要满足的规范，满足这些规范的数据库是简洁的、结构明晰的，同时，不会发生增删改查（CRUD）操作异常。反之则是乱七八糟，不仅给数据库的编程人员制造麻烦，而且面目可憎，可能存储了大量不需要的冗余信息。

① 1NF-无重复的列

所谓第一范式（1NF）是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。

经检查，上面所得到的表的属性都是不可分割的。

② 2NF-属性完全依赖于主键

第二范式（2NF）要求数据库表中的每个实例或行必须可以被惟一地区分。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。这个惟一属性列被称为主关键字或主键、主码。

我们所设计的 8 个表中，每个表都只有一个主键，所以每个表的属性都完全依赖于主键，即满足 2NF。

③ 3NF-属性不依赖于其他非主属性

如果关系模式 R 是第二范式，且每个非主属性都不传递依赖于 R 的候选键，则称 R

为第三范式模式。

我们所设计的 8 个表中，没有非主属性对于码的传递函数依赖，即满足 3NF。

4.3 数据表

- 社团活动：

名	类型	长度	小数点	不是 null
id	int	11	0	☑
address	varchar	255	0	☐
content	varchar	255	0	☐
end_date	tinyblob	0	0	☐
name	varchar	255	0	☐
start_date	tinyblob	0	0	☐

- 社团评价：

名	类型	长度	小数点	不是 null
id	int	11	0	☑
content	varchar	255	0	☐
star	int	11	0	☐
time	tinyblob	0	0	☐
by_user_id	int	11	0	☐
to_league_id	int	11	0	☐

- 社团：

名	类型	长度	小数点	不是 null
id	int	11	0	☑
academy	varchar	255	0	☐
club_due	double	0	0	☐
name	varchar	255	0	☑
start_date	tinyblob	0	0	☐

- 系统管理员：

名	类型	长度	小数点	不是 null
id	int	11	0	☑
password	varchar	255	0	☑
username	varchar	255	0	☑

- 社团公告：

名	类型	长度	小数点	不是 null
id	int	11	0	☑
content	varchar	255	0	☐
post_time	tinyblob	0	0	☐
title	varchar	255	0	☐
by_league_id	int	11	0	☐

- 社团用户：

名	类型	长度	小数点	不是 null
id	int	11	0	☑
address	varchar	255	0	☐
major	varchar	255	0	☐
password	varchar	255	0	☑
username	varchar	255	0	☑

- 社团活动参与：

名	类型	长度	小数点	不是 null
league_id	int	11	0	☑
activity_id	int	11	0	☑

- 社团用户参与：

名	类型	长度	小数点	不是 null
league_id	int	11	0	☑
user_id	int	11	0	☑

5．物理结构设计

5.1 索引

- 由于社团名经常出现在查询和表连接中，并且系统中社团名唯一不重复，所以在该属性上建立 B-tree 索引
- 由于管理员经常登录，经常出现在表查询中，并且系统中管理员登录名唯一不重复，所以在该属性上建立 B-tree 索引
- 由于大学生信息表中的学生学号经常出现在查询和表连接中，并且作为主键，它的值唯一，所以在这个属性上建立 B-tree 索引。

B-tree 索引物理原理：

B+树是数据库系统实现索引的首选数据结构。评价一个数据结构作为索引的优劣最重要的指标是在查找过程中磁盘 IO 操作次数的渐进复杂度。

根据 BTree 的定义，可知检索一次最多需要访问 h 个节点。数据库系统的设计者巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次 IO 就可以完全载入。为了达到这样的目的，每次新建节点时，直接申请一个页的空间，这样就保证一个节点无力上也存储在一个页里，加之计算机硬盘存储分配都是按照页对齐的，就实现了一个 node 只需要一次 IO。

6．数据库实现与维护

6.1 数据库表结构建立

6.1.1 活动表 Activity

```
-- ----------------------------
-- Table structure for activity
-- ----------------------------
DROP TABLE IF EXISTS `activity`;
CREATE TABLE `activity` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `address` varchar(255) DEFAULT NULL,
  `content` varchar(255) DEFAULT NULL,
  `end_date` tinyblob,
  `name` varchar(255) DEFAULT NULL,
  `start_date` tinyblob,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6.1.2 社团评价表 evaluation

```
-- ----------------------------
-- Table structure for evaluation
-- ----------------------------
DROP TABLE IF EXISTS `evaluation`;
CREATE TABLE `evaluation` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `content` varchar(255) DEFAULT NULL,
  `star` int(11) DEFAULT NULL,
  `time` tinyblob,
  `by_user_id` int(11) DEFAULT NULL,
  `to_league_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `FKh2hatsg8lg8ydngqgql0unxpx` (`by_user_id`),
```

```
    KEY `FK89ib103ndnixgyus1ut5jr02e` (`to_league_id`),
    CONSTRAINT `FK89ib103ndnixgyus1ut5jr02e` FOREIGN KEY
(`to_league_id`) REFERENCES `league` (`id`),
    CONSTRAINT `FKh2hatsg8lg8ydngqgql0unxpx` FOREIGN KEY
(`by_user_id`) REFERENCES `user` (`id`)
  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6.1.3 社团表 league

```
  -- ----------------------------
  -- Table structure for league
  -- ----------------------------
  DROP TABLE IF EXISTS `league`;
  CREATE TABLE `league` (
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `academy` varchar(255) DEFAULT NULL,
    `club_due` double DEFAULT NULL,
    `name` varchar(255) NOT NULL,
    `start_date` tinyblob,
    PRIMARY KEY (`id`),
    UNIQUE KEY `UK_n8qcbpi2pjf8bbenfm9le3v36` (`name`)
  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6.1.4 社团活动参与纪录表 league_activity

```
  -- ----------------------------
  -- Table structure for league_activity
  -- ----------------------------
  DROP TABLE IF EXISTS `league_activity`;
  CREATE TABLE `league_activity` (
    `league_id` int(11) NOT NULL,
    `activity_id` int(11) NOT NULL,
    KEY `FK7g2y827hegkltuycaffg8sl5c` (`activity_id`),
    KEY `FKgsuul4o7bix397rlv0nh2g6wm` (`league_id`),
    CONSTRAINT `FK7g2y827hegkltuycaffg8sl5c` FOREIGN KEY
(`activity_id`) REFERENCES `activity` (`id`),
    CONSTRAINT `FKgsuul4o7bix397rlv0nh2g6wm` FOREIGN KEY
(`league_id`) REFERENCES `league` (`id`)
  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6.1.5 社团用户参与纪录表 league_user

```
  -- ----------------------------
  -- Table structure for league_user
  -- ----------------------------
  DROP TABLE IF EXISTS `league_user`;
```

```
CREATE TABLE `league_user` (
  `league_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  KEY `FK2a17bp9p71ipbpwotxuhc5q6h` (`user_id`),
  KEY `FKh7auyvlq2pg4ms5q011220v25` (`league_id`),
  CONSTRAINT `FK2a17bp9p71ipbpwotxuhc5q6h` FOREIGN KEY (`user_id`)
REFERENCES `user` (`id`),
  CONSTRAINT `FKh7auyvlq2pg4ms5q011220v25` FOREIGN KEY
(`league_id`) REFERENCES `league` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6.1.6 管理员表 manager

```
-- ----------------------------
-- Table structure for manager
-- ----------------------------
DROP TABLE IF EXISTS `manager`;
CREATE TABLE `manager` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `password` varchar(255) NOT NULL,
  `username` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `UK_o0yekye62maw0eia889dcxyk7` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6.1.7 社团公告表 post

```
-- ----------------------------
-- Table structure for post
-- ----------------------------
DROP TABLE IF EXISTS `post`;
CREATE TABLE `post` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `content` varchar(255) DEFAULT NULL,
  `post_time` tinyblob,
  `title` varchar(255) DEFAULT NULL,
  `by_league_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `FKdkqk8bryl5tlyugncimdkxg0s` (`by_league_id`),
  CONSTRAINT `FKdkqk8bryl5tlyugncimdkxg0s` FOREIGN KEY
(`by_league_id`) REFERENCES `league` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6.1.8 大学生用户信息表 user

```
-- ----------------------------
```

```
-- Table structure for user
-- ----------------------------
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `address` varchar(255) DEFAULT NULL,
  `major` varchar(255) DEFAULT NULL,
  `password` varchar(255) NOT NULL,
  `username` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `UK_sb8bbouer5wak8vyiiy4pf2bx` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6.2 数据库视图建立

6.2.1 五星级评价社团视图

创建五星级评价社团查询的视图，简化了数据库查询操作，将表连接的操作隐藏起来。当管理员查询时，可以直接查询视图。

```
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost`
SQL SECURITY DEFINER VIEW `five_star` AS
select `evaluation`.`to_league_id` AS `to_league_id`,
`league`.`id` AS `id`,
`evaluation`.`star` AS `star`
from (`evaluation` join `league`
on((`evaluation`.`to_league_id` = `league`.`id`)))
where (`evaluation`.`star` = 5)
```

6.2.2 某学院所有社团发表公告视图

创建某学院所有社团发表公告的视图，简化了数据库查询操作，将表连接的操作隐藏起来。当管理员查询记录时，可以直接查询视图。

```
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost`
SQL SECURITY DEFINER VIEW `academy_post` AS
select `post`.`by_league_id` AS `by_league_id`,
`league`.`id` AS `id`,
`league`.`academy` AS `academy`,
`post`.`title` AS `title`,
`post`.`content` AS `content`,
`post`.`post_time` AS `post_time`
from (`post` join `league` on(
```

```
    (`post`.`by_league_id` = `league`.`id`))
  )
```

6.3 数据库存储过程建立

6.3.1 社团公告查询存储过程

为了方便查询某社团所有的公告信息和社团本身的信息，并且方便进行过滤，所以创建存储过程。

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `v2`()
BEGIN
   SELECT *
   FROM post, league
   WHERE post.by_league_id = league.id;
END
```

7. 社团系统 Java 部分代码和运行

7.1 系统实体类实现

7.1.1 社团活动类 Activity 实现

```
@Entity
public class Activity implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String name;
    private String content;
    private String address;
    private LocalDate startDate;
    private LocalDate endDate;

    @ManyToMany(mappedBy = "activityList")
    private List<League> leagueList = new ArrayList<>();

    @Override
    public String toString() {
        return "Activity{" +
               "id=" + id +
```

```
                ", name='" + name + '\'' +
                ", content='" + content + '\'' +
                ", address='" + address + '\'' +
                ", startDate=" + startDate +
                ", endDate=" + endDate +
                '}';
    }
}
```

7.1.2 社团评价 Evaluation 类实现

```
@Entity
public class Evaluation implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private Integer star;
    private String content;
    private LocalDateTime time;

    @ManyToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "to_league_id")
    private League toLeague;

    @ManyToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "by_user_id")
    private User byUser;

    @Override
    public String toString() {
        return "Evaluation{" +
                "id=" + id +
                ", star=" + star +
                ", content='" + content + '\'' +
                ", time=" + time +
                '}';
    }
}
```

7.1.3 社团 League 类实现

```
@Entity
public class League implements Serializable {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(unique = true, nullable = false)
    private String name;

    private String academy;
    private Double clubDue;
    private LocalDate startDate;

    @ManyToMany(cascade = CascadeType.PERSIST)
    @JoinTable(name = "league_activity",
        joinColumns = {@JoinColumn(name = "league_id", referencedColumnName =
"id")},
        inverseJoinColumns = {@JoinColumn(name = "activity_id",
referencedColumnName = "id")})
    private List<Activity> activityList = new ArrayList<>();

    @OneToMany(mappedBy = "toLeague")
    private List<Evaluation> evaluationList = new ArrayList<>();

    @OneToMany(mappedBy = "byLeague")
    private List<Post> postList = new ArrayList<>();

    @ManyToMany(cascade = CascadeType.PERSIST)
    @JoinTable(name = "league_user",
        joinColumns = {@JoinColumn(name = "league_id", referencedColumnName =
"id")},
        inverseJoinColumns = {@JoinColumn(name = "user_id", referencedColumnName =
"id")})
    private List<User> userList = new ArrayList<>();

    @Override
    public String toString() {
        return "League{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", academy='" + academy + '\'' +
                ", clubDue=" + clubDue +
                ", startDate=" + startDate +
                '}';
```

```
    }
}
```

7.1.4 系统管理员 Manager 实现

```
@Entity
public class Manager implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public String toString() {
```

```
        return "Manager{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                '}';
    }
}
```

7.1.5 社团公告 Post 类实现

```
@Entity
public class Post implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String title;
    private String content;
    private LocalDateTime postTime;

    @ManyToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "by_league_id")
    private League byLeague;

    @Override
    public String toString() {
        return "Post{" +
                "id=" + id +
                ", title='" + title + '\'' +
                ", content='" + content + '\'' +
                ", postTime=" + postTime +
                '}';
    }
}
```

7.1.6 学生用户 User 类实现

```
@Entity
public class User implements Serializable, Comparable<User> {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
```

```
    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    private String major;

    private String address;

    @OneToMany(mappedBy = "byUser")
    private List<Evaluation> evaluationList = new ArrayList<>();

    @ManyToMany(mappedBy = "userList")
    private List<League> leagueList = new ArrayList<>();

    @Override
    public int compareTo(User o) {
        return id.compareTo(o.id);
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", major='" + major + '\'' +
                ", address='" + address + '\'' +
                '}';
    }
}
```

7.2 系统界面设计与实现

7.2.1 启动类实现

```
package com.zzkun;

import com.zzkun.view.MainView;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class LeagueManagerApplication implements CommandLineRunner {

   @Autowired private MainView mainView;

   public static void main(String[] args)  {
      SpringApplication.run(LeagueManagerApplication.class, args);
   }

   @Override
   public void run(String... strings) throws Exception {
      mainView.run();
   }
}
```

7.2.2 主界面实现

```
package com.zzkun.view;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.Scanner;

@Component
public class MainView {

    @Autowired private Scanner cin;
    @Autowired private ActivityView activityView;
    @Autowired private EvaluationView evaluationView;
    @Autowired private LeagueView leagueView;
    @Autowired private ManagerView managerView;
    @Autowired private PostView postView;
    @Autowired private UserView userView;

    public void run() {
        System.out.print("请输入管理员账号：");
        String username = cin.nextLine();
        System.out.print("请输入管理员密码：");
        String password = cin.nextLine();
        while (!("root".equals(username) && "root".equals(password))) {
            System.out.println("用户名或密码输入错误，请重新输入(默认 root, root)");
```

```
        }
        mainControl();
    }

    private void mainControl() {
        while (true) {
            printMainMenu();
            int num = cin.nextInt();
            switch (num) {
                case 1:
                    userView.run();
                    break;
                case 2:
                    leagueView.run();
                    break;
                case 3:
                    activityView.run();
                    break;
                case 4:
                    postView.run();
                    break;
                case 5:
                    evaluationView.run();
                    break;
                case 6:
                    managerView.run();
                    break;
                case 0:
                    return;
            }
        }
    }

    private void printMainMenu() {
        System.out.println("========主菜单========");
        System.out.println("1.学生用户管理");
        System.out.println("2.社团管理");
        System.out.println("3.社团活动管理");
        System.out.println("4.社团公告管理");
        System.out.println("5.社管质量(评价体系)管理");
        System.out.println("6.管理员账号管理");
        System.out.println("0.退出社团管理系统");
        System.out.println("====================");
```

```
    }
}
```

7.2.3 学生管理功能界面实现

```
package com.zzkun.view;

import com.zzkun.dao.UserDao;
import com.zzkun.model.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.List;
import java.util.Scanner;

@Component
public class UserView {

    @Autowired private Scanner cin;
    @Autowired private UserDao userDao;

    public void run() {
        while (true) {
            printMenu();
            int num = cin.nextInt();
            if (num == 1) {
                showAllUser();
            } else if (num == 2) {
                addUser();
            } else {
                return;
            }
        }
    }

    private void showAllUser() {
        List<User> users = userDao.findAll();
        System.out.println("所有用户信息：");
        for (User user : users) {
            System.out.println(user);
        }
    }

    private void addUser() {
```

```
        User user = new User();
        System.out.print("输入新用户学号：");
        user.setUsername(cin.next());
        System.out.print("输入新用户密码：");
        user.setPassword(cin.next());
        System.out.print("输入新用户专业：");
        user.setMajor(cin.next());
        System.out.print("输入新用户家庭地址：");
        user.setAddress(cin.next());
        try {
            userDao.save(user);
        } catch (Exception e) {
            System.out.println("添加新用户失败！");
        }
    }

    private void printMenu() {
        System.out.println("-----------------------");
        System.out.println("1.查看当前所有学生信息");
        System.out.println("2.添加新的学生用户");
        System.out.println("3.修改学生用户信息");
        System.out.println("4.修改学生密码");
        System.out.println("5.修改学生专业信息");
        System.out.println("6.修改学生家庭地址信息");
        System.out.println("0.返回上一层");
        System.out.println("-----------------------");
    }

}
```

7.2.4 Bean 配置类实现

```
package com.zzkun.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.Scanner;

@Configuration
public class BaseConfiguration {

    @Bean
    public Scanner scanner() {
```

```
        return new Scanner(System.in);
    }
}
```

7.2.5 MySQL 连接配置文件

```
spring.datasource.url=jdbc:mysql://localhost:3306/league_manager?characterEncoding
=utf8&useSSL=true
spring.datasource.username=root
spring.datasource.password=123456
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.initialize=true
spring.jpa.properties.hibernate.hbm2ddl.auto=create
logging.level.root=error
```

7.3 系统部分功能截图

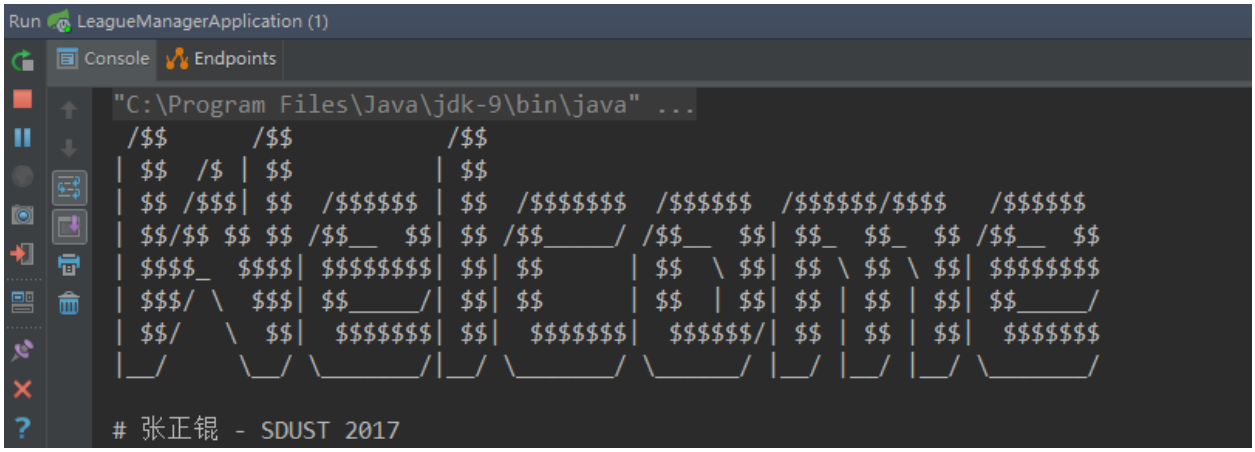


图 1 系统启动

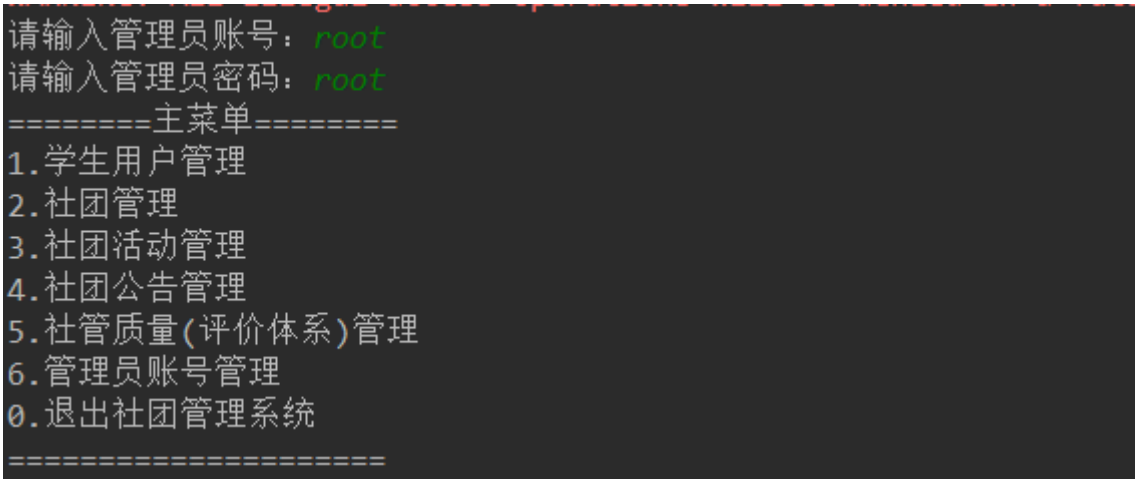


图 2 管理员登陆和系统主要功能菜单

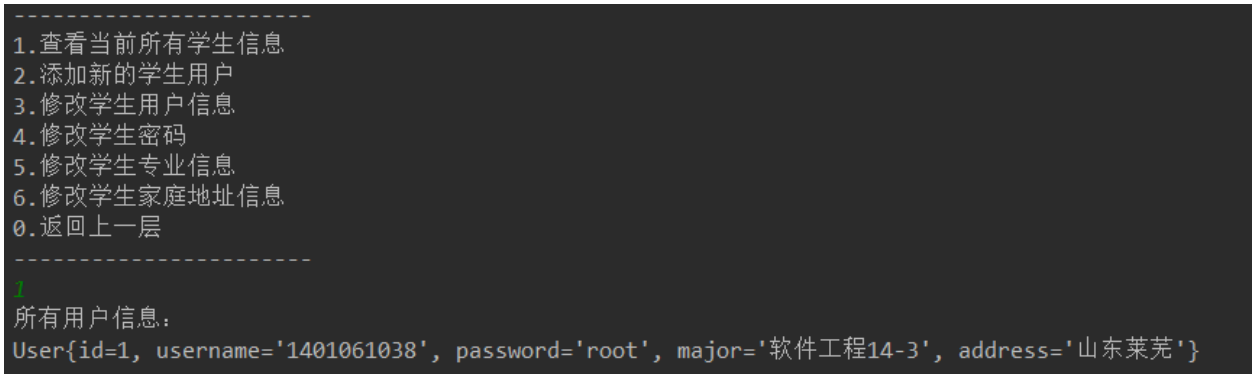


图 3 用户管理功能

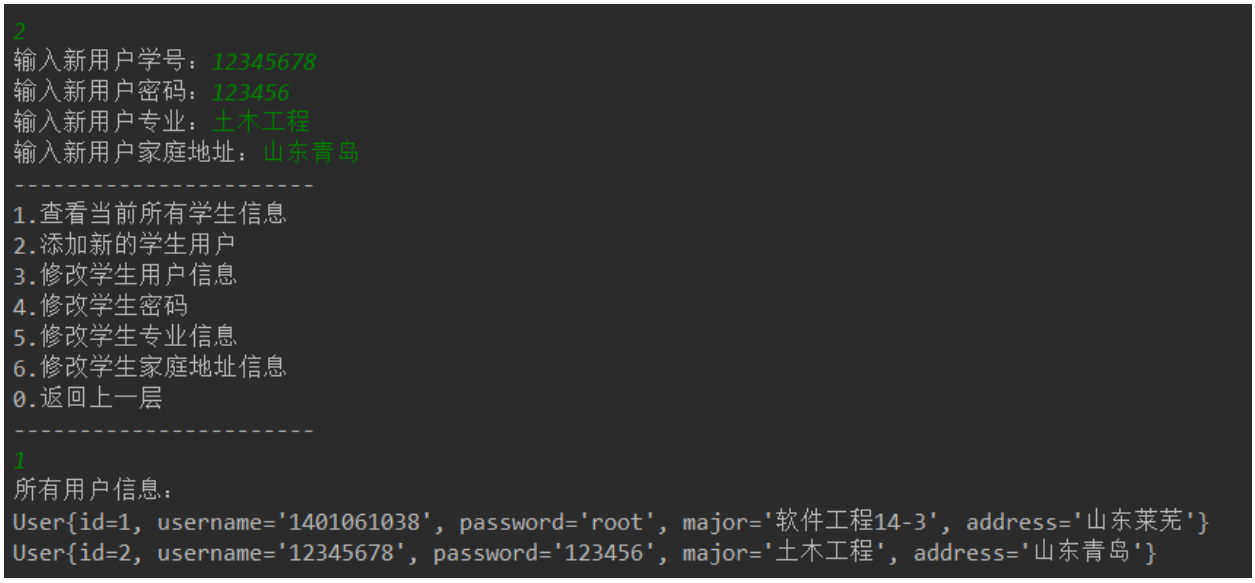


图 4 添加学生用户

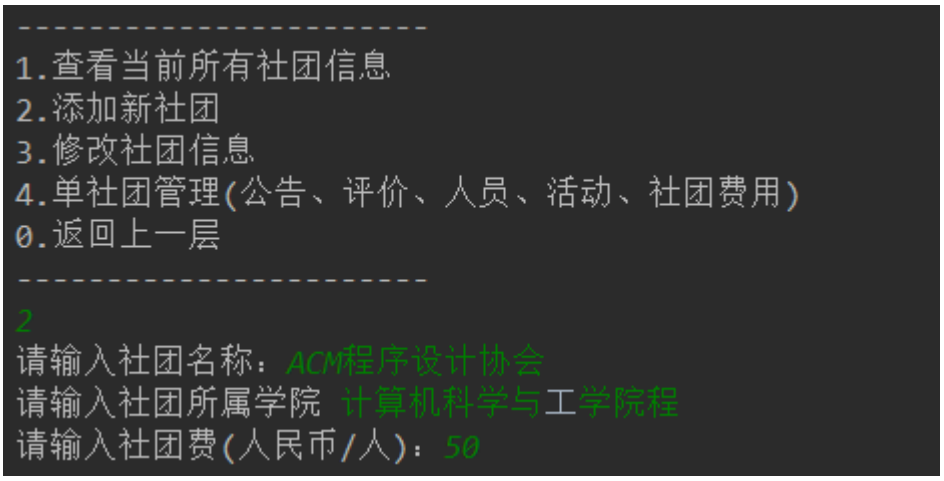


图 5 社团管理和社团添加

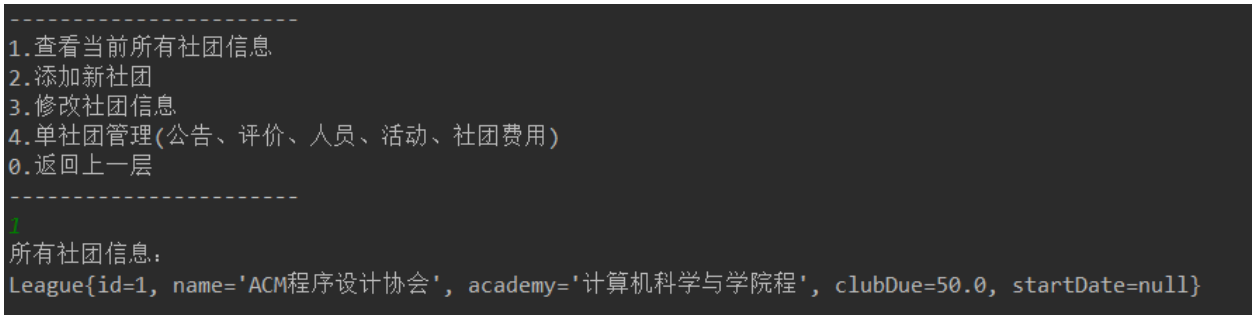


图 6 社团查看

```
----------------------
1.查看当前所有社团信息
2.添加新社团
3.修改社团信息
4.单社团管理(公告、评价、人员、活动、社团费用)
0.返回上一层
----------------------
0
========主菜单========
1.学生用户管理
2.社团管理
3.社团活动管理
4.社团公告管理
5.社管质量(评价体系)管理
6.管理员账号管理
0.退出社团管理系统
====================
0

Process finished with exit code 0
```

图 7 社团系统退出

8．设计总结

通过本次课程设计，对 SQL 语言，数据库的创建、修改、删除方法有了一定的了解，通过导入表和删除表、更改表，学会了数据库的基本操作。

数据库设计是一门理论性和实践性都很强的专业基础课，也是一门综合性的技术基础学科。许多测试理论和方法只有通过实际验证才能加深理解并真正掌握。

我们做实验绝对不能人云亦云，要有自己的看法，这样我们就要有充分的准备，若是做了也不知道是个什么实验，那么做了也是白做。实验总是与课本知识相关的，实验中回顾课本的知识，知道实验时将要写什么 SQL 语句，知道怎么处理程序和数据库的联系。

我们做实验不要一成不变和墨守成规，应该有改良创新的精神。实际上，在弄懂了实验原理的基础上，我们的时间是充分的，做实验应该是游刃有余的，如果说创新对于我们来说是件难事，那改良总是有可能的。比如在设计数据库模式时候，从 1NF 到 2NF，从 2NF 到 3NF，甚至从 3NF 到 BCNF，每次改进都是很大的进步，都能学到很多知识。

回顾起此课程设计，至今我仍感慨颇多，从理论到实践，在这段日子里，可以说得是苦多于甜，但是可以学到很多很多的东西，同时不仅可以巩固了以前所学过的知识，而且学到了很多在书本上所没有学到过的知识。

此次设计也让我明白了思路即出路，有什么不懂不明白的地方要及时请教或上网查询，只要认真钻研，动脑思考，动手实践，就没有弄不懂的知识，收获颇丰。