数据科学与计算机学院

数据库课程设计报告

姓名 杨麒平

学号 15336220

专业 信息与计算科学

完成日期： 2018 年 1 月 7 日

目　录

第 1 章

艇员管理系统

A crew management system can store the training record and provide a great feed back for the training crew. Our crew is training on a daily basis, one finds that by comparing and contrast, would make a positive and huge impact on the confidence and momentum of he training individuals as well as crew members. In the old school ways, our crew members have to use pen and paper to write down their training results, and later need to be typed into computers. Therefore developing a crew management system would greatly enhance the efficiency in terms of designing training plans and logging training results . 艇员管理系统是一个能够发布训练计划，统计训练记录的系统。并能通过统计得到的训练记录进行反馈，为队员提供参考。一个体育团队通过收集每天的训练情况来定期给队员提供积极科学的反馈，并以此能够修正训练计划，从而使训练更加高效有序。可是现实团队里面在即时通讯软件（wechat）里面发布和收集训练记录，或者说用纸来记录数据。这样的做法费时费力而且数据也不完整，即使通信软件里面有相当强大的搜索功能。但是如果每次进行数据收集都是对聊天记录进行搜索的话，数据难免丢失（数据格式不一致等原因）因此开发一个数据库应用来进行队内事务管理是相当有必要的。

§1.1 开发环境与开发工具

§1.2 系统需求分析

1. 系统的用户管理
   包括用户的添加、删除、修改信息、修改密码等。

2. 艇员信息管理
   艇员信息的添加、删除、修改、查询。

3. 训练计划管理
   训练计划的添加，修改，删除，查询。

4. 训练记录管理
   训练记录的添加，修改，删除，查询。

5. 船艇信息
   艇的信息添加，修改，删除，查询。

6. 开支信息
   开支记录的添加，修改，删除，查询。

§1.3 功能需求分析

艇员管理系统按照上面所述，管理功能主要在艇员信息、训练计划和训练记录的管理上面。下面还有一些主要的功能说明

1. 其中用户分为两个级别。普通用户级别和特权用户级别。特权用户级别的操作是普通用户级别的超集。

2. 其中添加、删除操作输入特权用户级别，AKA 只有特权用户能够添加新的成员。而每个用户需要自己的艇员信息被录入才能注册新的用户。每个用户能够对自己的信息进行查询，对部分信息能够进行修改。

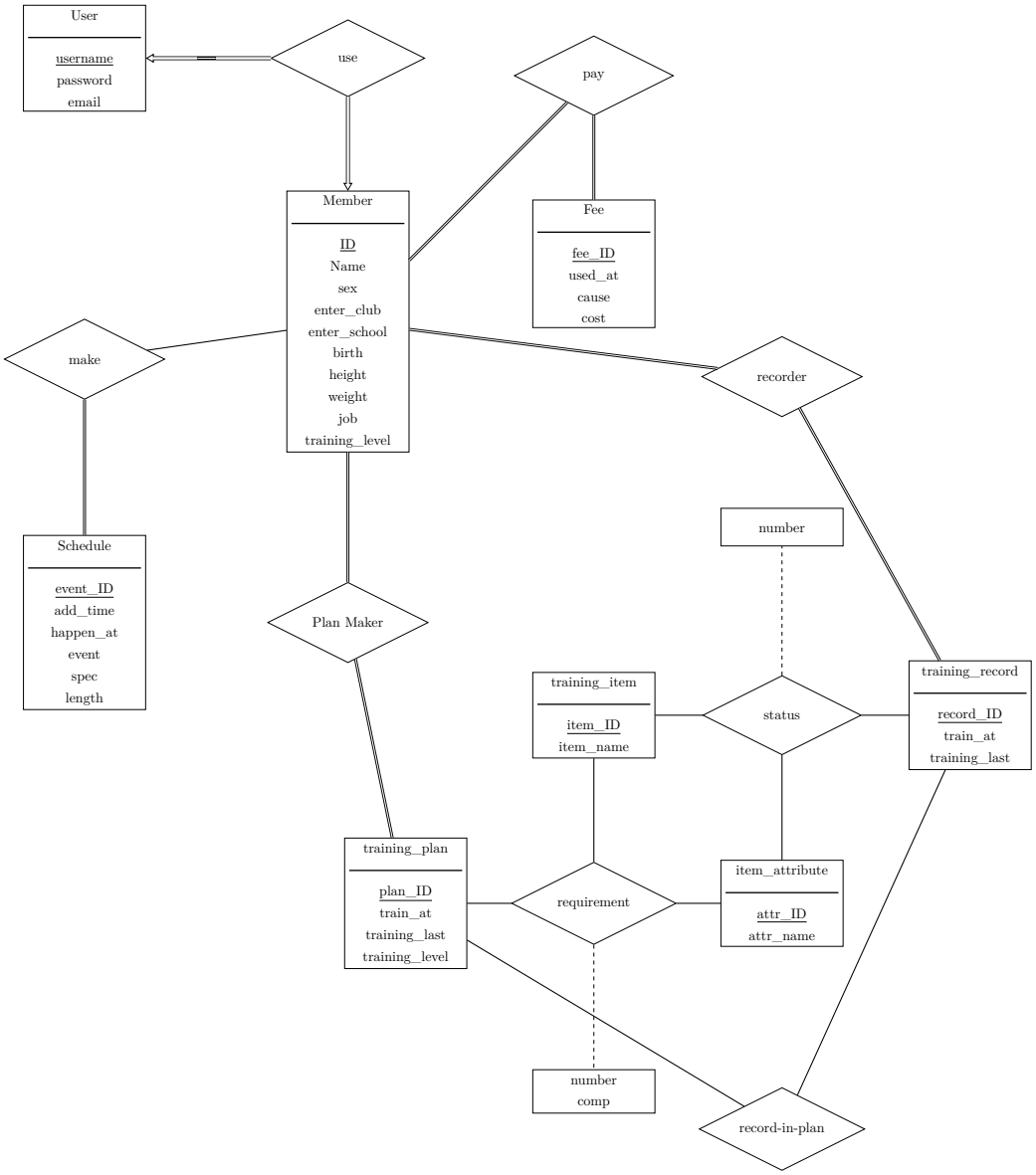
3. 其中添加，修改，删除属于特权用户级别。

4. 每个用户只能对自己的训练记录进行 (CUD–Create, Update, Delete)。每个用户可以查询其他用户的训练记录。

5. 普通用户只能部分修改，能够查询。

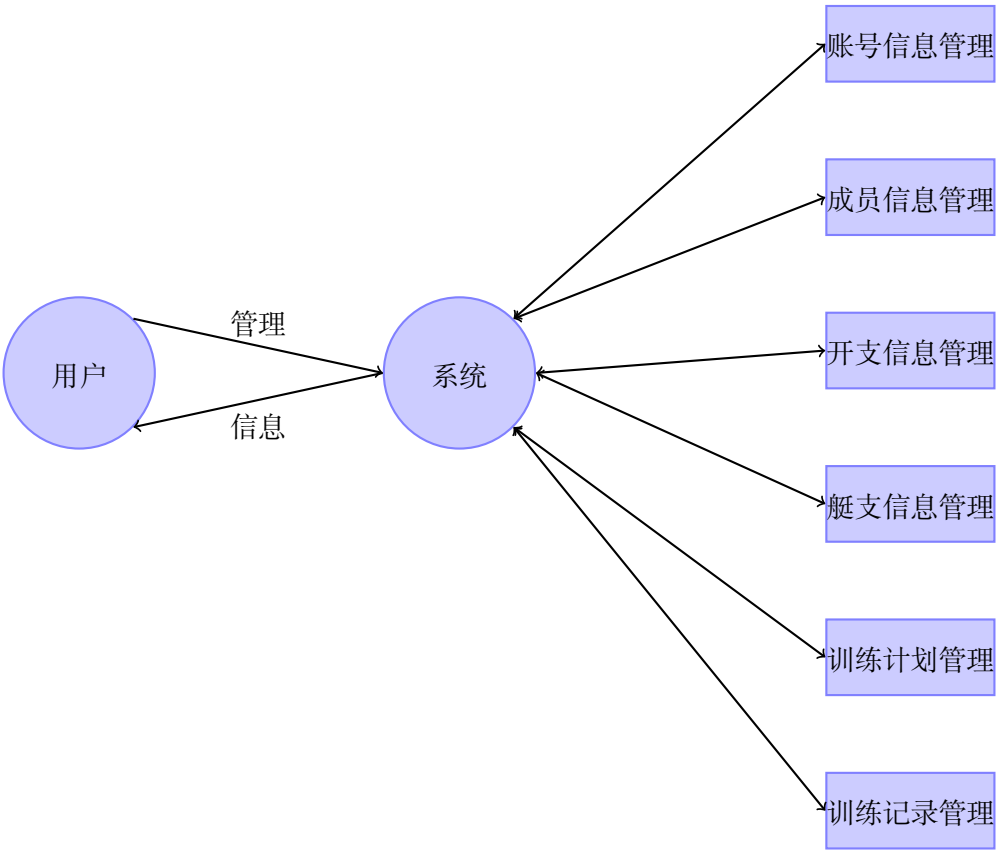
§1.4 系统设计

§1.4.1 系统 ER 图

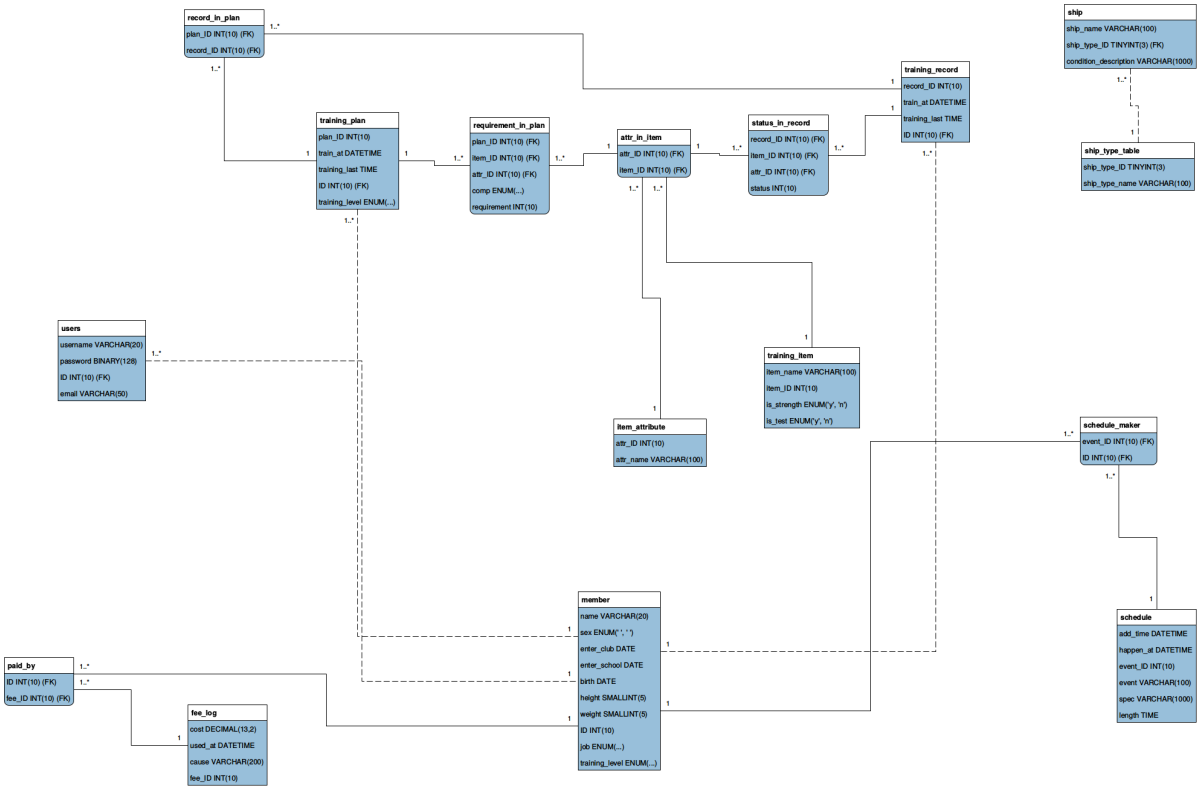


§1.4.2 数据流程图

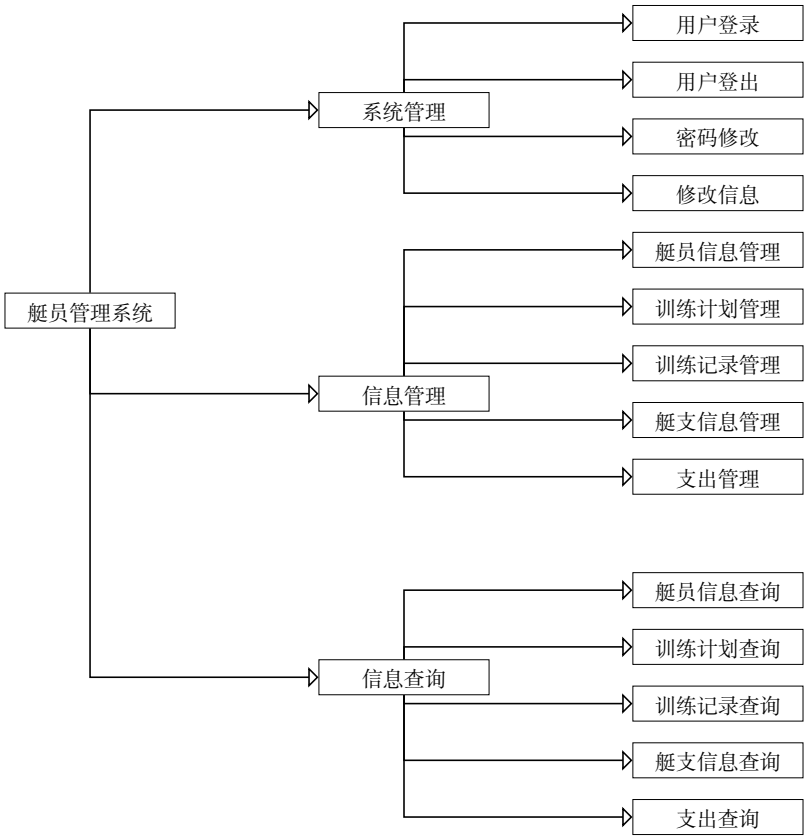
图中的系统层就是我们设计的网页应用。他简化了用户和数据库的交互，并提供一定的等级分层，以及维护系统的数据完整性。



§1.4.3 数据库关系模式设计



§1.4.4 系统功能模块图



§1.4.5 物理结构设计

```
-- with debug if and if not
drop database if exists crewmen;
create database crewmen character set utf8 collate utf8_general_ci;
use crewmen;

drop table if exists users;

drop table if exists schedule_maker;
drop table if exists paid_by;
drop table if exists schedule;
drop table if exists fee_log;

drop table if exists record_in_plan;
drop table if exists requirement_in_plan;
drop table if exists status_in_record;

drop table if exists attr_in_item; -- referenced by requirement and status

drop table if exists training_record; -- by status by record_in_plan
drop table if exists training_plan; -- by requirment by record_in_plan
drop table if exists item_attribute; -- referenced by attr_in_item
drop table if exists training_item; -- referenced by attr_in_item
```

```
drop table if exists member; -- referenced by item plan record paid_by schedule_maker users
drop table if exists ship;
drop table if exists ship_type_table;


create table if not exists member
       (name varchar(20) not null,
       -- big chunk for information
       sex enum(' 男', ' 女') default ' 男',
       enter_club date, -- the crew
       enter_school date, -- school
       birth date,
       height smallint unsigned,
       weight smallint unsigned,
       ID int unsigned,
       -- big chunk for information
       job enum('couch', 'crew leader', 'crew member') default 'crew member',
       training_level enum('newbie', 'medium', 'old bird') default 'newbie',
       primary key(ID)
       );-- character set utf8 collate utf8_general_ci;

create table if not exists users
       (username varchar(20) not null,
       password binary(128) not null, -- half byte each, so 128 * 0.5 * 8 is 512 using sha2('pass', 512)
       -- priority tinyint unsigned not null defalut 2, -- the lower the better
       ID int unsigned, -- don't have to be unique, since it's PK for member
       email varchar(50) not null unique,
       primary key(username),
       foreign key(ID) references member(ID) on delete cascade
       );

-- how to enforce constraint on full participation, every schedule should have at least a maker...
create table if not exists schedule
       (add_time datetime default current_timestamp,
       happen_at datetime, -- time could be unsure of
       event_ID int unsigned auto_increment,
       event varchar(100) not null,
       spec varchar(1000),
       length time,
       primary key(event_ID)
       );

create table if not exists schedule_maker
       (event_ID int unsigned,
       ID int unsigned,
       primary key(event_ID, ID),
       foreign key(event_ID) references schedule(event_ID),
       foreign key(ID) references member(ID) on delete cascade
       );
```

```
-- same question as schedule, every fee should have a payer
create table if not exists fee_log
       (cost decimal(13,2), -- cost could range from cent to 10^13 which is 10^4 billion yuan, should be enough
        used_at datetime,
        cause varchar(200),
        fee_ID int unsigned,
        primary key(fee_ID)
       );

create table if not exists paid_by
       (ID int unsigned,
       fee_ID int unsigned,
       primary key(ID, fee_ID),
       foreign key(ID) references member(ID) on delete cascade,
       foreign key(fee_ID) references fee_log(fee_ID)
       );

-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

-- It's divided into three main parts:
-- The plan and record are symmetrical
-- ignore! no longer supported (although plan can have multiple makes?? weird functionality though)


-- The main similarity lies in the relation of (record, item, attribute, status/requirement)
-- when it indicates that a record can have multiple items
-- when an item can have multiple attributes
-- Therefore this relation can be able to indicate a collection of status of a collection of training items in

-- notice how attr_in_item is referenced by (requirement_in_plan and status_in_record)
-- may need to define training item
create table if not exists training_item
       (item_name varchar(100),
       item_ID int unsigned auto_increment,
       is_strength enum('y', 'n'), -- strength or aerobic
       is_test enum('y', 'n'),     -- test or regular
       primary key(item_ID)
       );

create table if not exists training_record
       (record_ID int unsigned auto_increment,
       train_at datetime,
       training_last time,
       ID int unsigned,
```

```
       primary key(record_ID),
       foreign key(ID) references member(ID)
       );

create table if not exists training_plan
       (
       plan_ID int unsigned auto_increment,
       train_at datetime,
       training_last time,
       ID int unsigned,
       training_level enum('newbie', 'medium', 'old bird', 'all') default 'all',
       primary key(plan_ID),
       foreign key(ID) references member(ID)
       );

create table if not exists item_attribute
       (attr_ID int unsigned,
       attr_name varchar(100) unique,
       primary key(attr_ID)
       );

create table if not exists attr_in_item
       (attr_ID int unsigned,
       item_ID int unsigned,
       primary key(attr_ID, item_ID),
       foreign key(attr_ID) references item_attribute(attr_ID),
       foreign key(item_ID) references training_item(item_ID)
       );

create table if not exists requirement_in_plan
       (
       plan_ID int unsigned,
       item_ID int unsigned,
       attr_ID int unsigned,
       comp enum('larger', 'smaller', 'no requirement') not null default 'no requirement',
       requirement int unsigned,
       primary key(plan_ID, item_ID, attr_ID),
       foreign key(item_ID, attr_ID) references attr_in_item(item_ID, attr_ID),
       foreign key(plan_ID) references training_plan(plan_ID)
       );

create table if not exists status_in_record
       (record_ID int unsigned,
       item_ID int unsigned,
       attr_ID int unsigned,
       status int unsigned,
       primary key(record_ID, item_ID, attr_ID),
       foreign key (item_ID, attr_ID) references attr_in_item(item_ID, attr_ID),
       foreign key (record_ID) references training_record(record_ID)
```

```
       );

-- below is record and plan relation
create table if not exists record_in_plan
       (plan_ID int unsigned,
       record_ID int unsigned,
       primary key(plan_ID, record_ID),
       foreign key(plan_ID) references training_plan(plan_ID),
       foreign key(record_ID) references training_record(record_ID)
       );

-- Obsoleted
-- create table if not exists plan_maker
--        (ID int unsigned,
--        plan_ID int unsigned,
--        primary key(ID, plan_ID),
--        foreign key(ID) references member(ID) on delete cascade,
--        foreign key(plan_ID) references training_plan(plan_ID)
--        );

-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

create table if not exists ship_type_table
       (ship_type_ID tinyint unsigned,
       ship_type_name varchar(100),
       primary key(ship_type_ID)
       );

create table if not exists ship
       (ship_name varchar(100),
       ship_type_ID tinyint unsigned,
       condition_description varchar(1000),
       primary key(ship_name),
       foreign key(ship_type_ID) references ship_type_table(ship_type_ID)
       );
```

§1.5 系统功能的实现

具体实现是利用 **python-flask** 框架，以及 mysql 数据库。
这个项目最主要的模块我认为是训练计划和训练记录，还有用户权限的实现。

§1.5.1 用户系统

用户登录 提供用户名和密码的表单。数据库储存密码这个设计利用的是 sha-512 的单向 hash 方式储存。以后估计会改用 bcrypt。因此每次收到密码，在后端要把字符串用 hash 得到对应的编码与数据库相应的用户名的密码进行比对。如果正确，从数据库中的成员 (member) 信息读出权限，

并存放到 session 里面，并把用户的登录状态还有用户名存放到 session 里面。

以下是主要的代码：

特权机制 对于某些页面进行限制，比如添加新成员，添加修改训练计划等。如果用户访问受限制的 url，先从登录后存放的 session 中找到对应的权限。如果权限不足，返回重定向，如果权限足够，返回对应的页面。

一下是主要的代码：

§1.5.2 训练计划

计划查询 每个人都能访问今日的训练计划。

更改计划 只有队长或者教练能够修改，添加或者删除训练计划。对于返回的表单进行检查。

§1.5.3 训练记录

§1.6 课程设计心得体会