Mini-CS Ranking 项目设计报告

周芯怡 17307130354
张作柏 17300240035

2019 年 5 月 6 日

目录

1 概览

本项目的所有源代码公开于 https://github.com/Oxer11/Mini-CSRanking。

1.1 项目要求

本项目需要我们设计一个数据库应用，采用客户端/服务器的结构，具体要求如下：

- 有完整的前后端架构（前端界面 + 后台数据库）
- 有用户注册登录系统
- 内容有实际意义且完整
- 支持前端对数据的增删查改操作

1.2 设计动机

这个项目的 idea 是受 CS Rankings[1]这个网站的启发。这个网站根据发表论文的数量对所有的学校进行了排名，这为学校提供了较为客观的评价指标。同时，在学校下方，还可以看到近年发表论文数量较多的学者，可以直接进入学者的主页，这也为不少出国党的同学提供了便利，可以更方便的选择自己中意的导师。

然而，这个网站还有一些小小的问题，就是对检索功能的支持还不够流畅。它无法直接按老师或学校的名字来检索，而是需要在排行榜上进行寻找，这是很费时的一件事。相比之下，GoogleScholar 的检索功能就更加友好，可以直接检索学者或论文。并且，GoogleScholar 可以浏览到学者近年来发表的论文的详细信息，可以说是一个较为全面的学术网站。

同时，若我们想了解某个领域的知名学者或顶级会议，仍缺乏比较合理的手段。虽然 DBLP 中提供了按会议检索的功能，但是没有根据定量的指标指出某领域的知名学者。

我们的应用旨在开发一个更全面的学术网站：结合三者的长处，选取其中核心功能，并增加一些个性化的设计与美观的可视化界面。

1.3 技术栈

开发语言	JavaScript / HTML / CSS / Python / SQLite
浏览器环境	Chrome / Firefox / Safari
第三方库	jQuery / Bootstrap

[1]http://csrankings.org

1.4 工作流程

我们两人在上学期的 ICS 课程中已经合作过一次，所以也就延续了上学期的“不分工”策略。所谓“不分工”，是指不明确区分前后端开发，而是两人共同开发，遇到具体任务时再分清两人的工作。之所以不区分前后端，一是因为项目较小，区分前后端会影响两人的交互性，导致开发效率降低；二是因为前后端的工作量差异较大，不便分工。实践证明，“不分工”策略是成功的，两人不仅顺利地完成了 PJ，且两人分得的工作量与工作难度也都较为平均。我们的具体分工如下：

周芯怡：

- 搜索功能前端页面的搭建与链接
- 用户登录/注册功能
- 用户个人中心的前后端设计
- 评论消息提醒的功能设计
- 网站的美化与运行测试

张作柏：

- 数据的搜集与处理
- 搜索功能关系模式的设计与后端搭建
- 使用外部元素，美化前端
- Ranklist 的前后端设计
- 新论文的添加与消息提醒功能

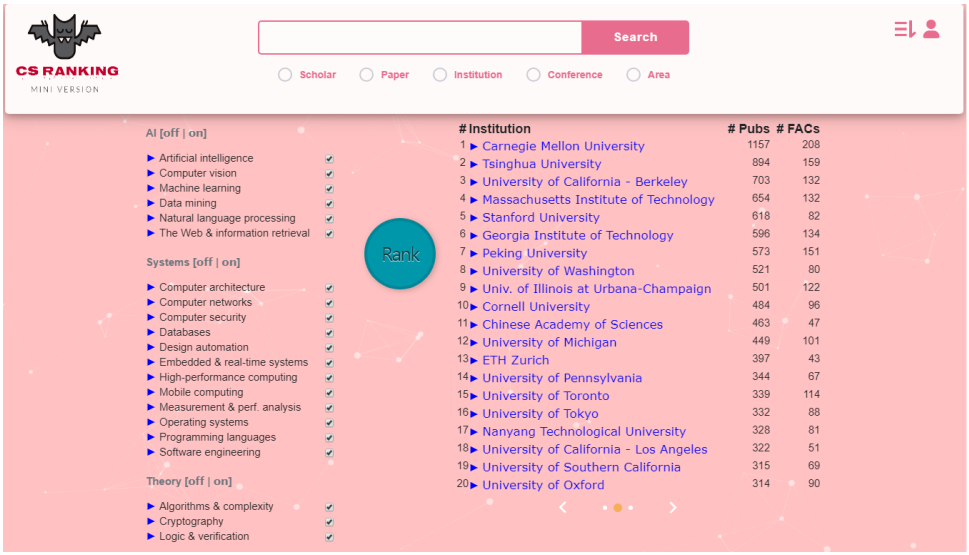
1.5 阅读指南

本报告长达近 30 页，要读完实属不易，在此整理概括各部分内容，以帮助读者阅读。在第二节中，我们将简单介绍 Mini-CS Ranking 项目的功能特点，并展示我们的前端界面。在第三节中，我们将列举分析数据库的关系模式。数据量大是本项目的特点之一，所以第四节我们将介绍数据的采集与导入流程。在第五节中，我们将详尽地分析项目中所涉及的增删查改操作。第六节中，我们将展开讨论在项目实现过程中遇到的困难和值得留意的实现细节。最后，在第七节中，我们简单总结了完成整个项目的感想与体会。

2 项目特点

Mini-CS Ranking 的主要设计目的是创建一个便捷的学术检索平台，并提供友好的用户交互界面。其主要特征有：

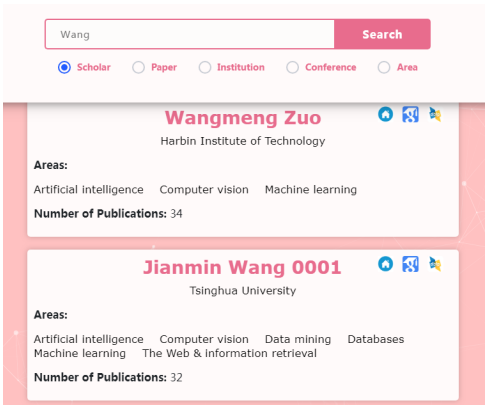
- **学校排名功能**：Mini-CS Ranking 模仿了 CS Rankings 的主要功能，即根据发表论文数量对学校进行排名。

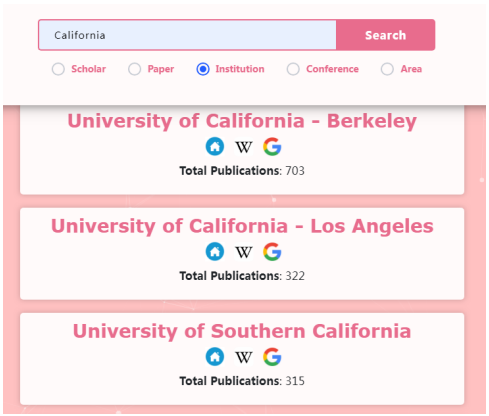


在排行榜中，可以看到学校的排名、发表论文数量和教师数量。点击左侧蓝色三角，即可看到学校下的老师排名。所有排名均是根据老师、学校发表的论文数量来降序排序的。老师的信息中还显示了老师所在的领域、老师的首页、老师的 GoogleScholar 页面和 DBLP 页面。

左侧是领域的信息，点击领域旁的蓝色三角，可看到属于该领域的顶会。勾选所关心的领域，点击 Rank 按钮，即可在排行榜中筛选出相关领域的老师。

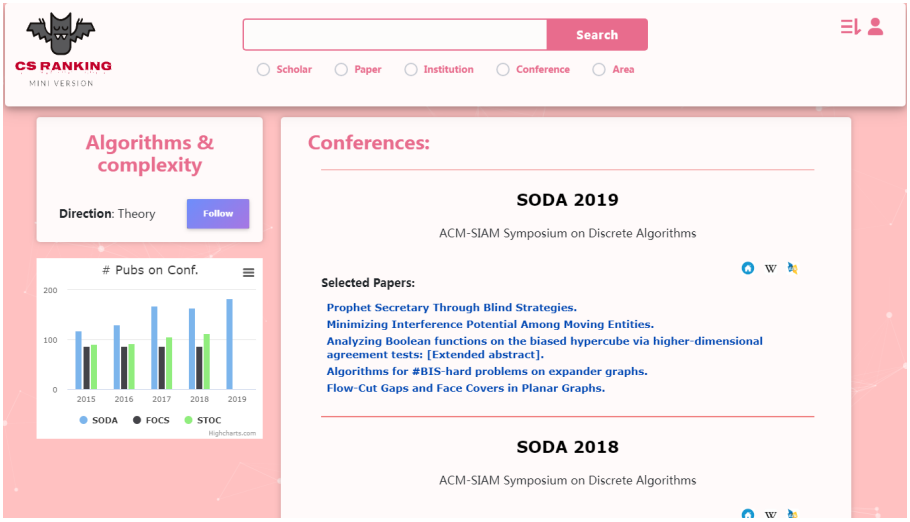
- **信息检索功能**：





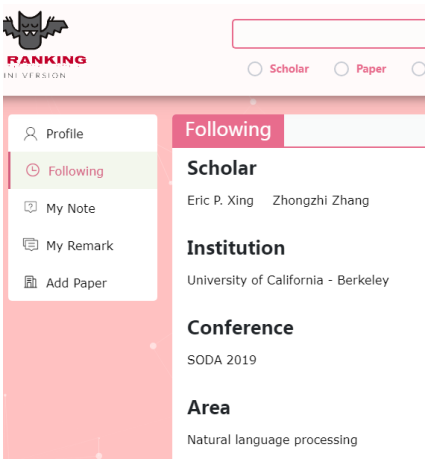
为了便于检索自己感兴趣的老师，我们设计了类似 GoogleScholar 的搜索功能。可以通过勾选搜索框下的选项，并输入搜索关键词来检索学校、论文、老师。并且，我们为每个学校和学者都设计了自己的主页。学者的主页中包含其近几年发表的论文、每年发表的论文数量和他的合作者。学校的主页中，则详细的列举了该校的学者信息，并统计了近几年中各个领域发表论文的数量。

- **领域会议信息**：除了上面提到的几项，我们还模仿 DBLP，提供了会议和领域的检索功能，为相关领域的学者提供便利。

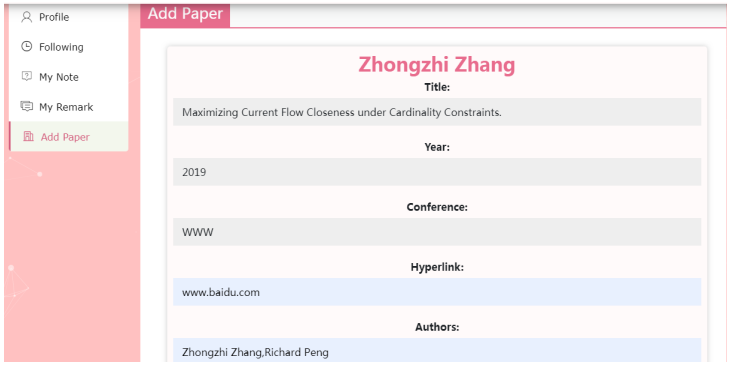


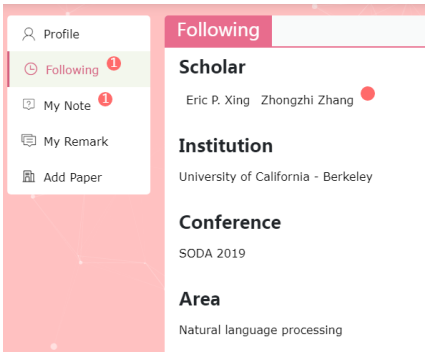
在会议的相关主页中，我们给出了该会议的所有论文，并提供了访问会议首页与 DBLP 的链接。在领域的主页中，我们则列举了该领域的顶级会议，统计了近几年来会议中的论文数量，并给出了几位近五年来领域最活跃的学者。

- **用户关注功能**：为了提升用户友好性，我们设计了关注功能，用户可以在相应的界面关注自己感兴趣的学者、老师、领域等等。这样一来，访问学者的主页就便捷了许多，不需要每次重新进行搜索；同时，当关注的学者发表新的论文时，会收到系统发来的消息提醒。

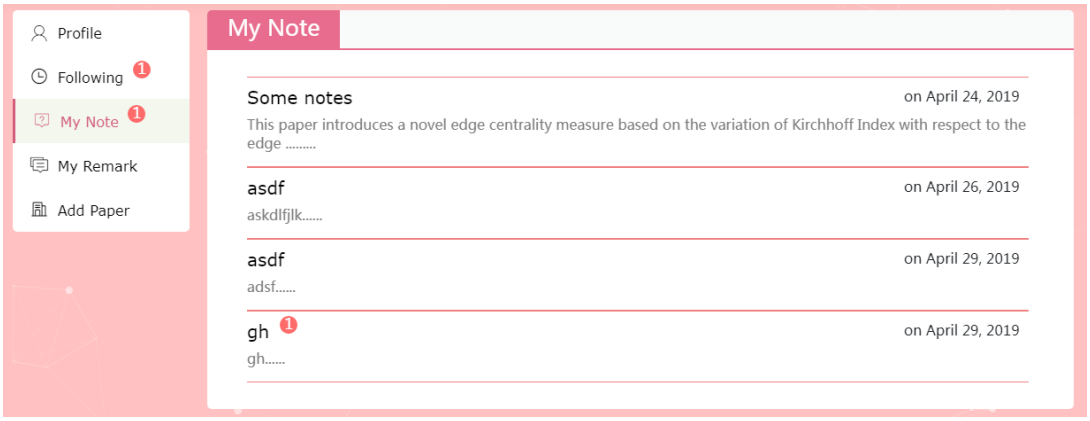


- **认证发表论文**：用户注册时，若系统检测到数据库中有与您同名的学者，则会提醒您可以进行学者认证。认证后，学者的信息会自动导入用户信息中。并且获得了发表新论文的权限，成功发表论文后，则会通知该学着的所有关注者。





- **书写笔记评论**：登录后，用户可以对自己感兴趣的论文书写笔记，并在论文主页下方留言。其他用户看到后，可在笔记下发表评论，相关用户则会接收到新添加评论的提醒信息。

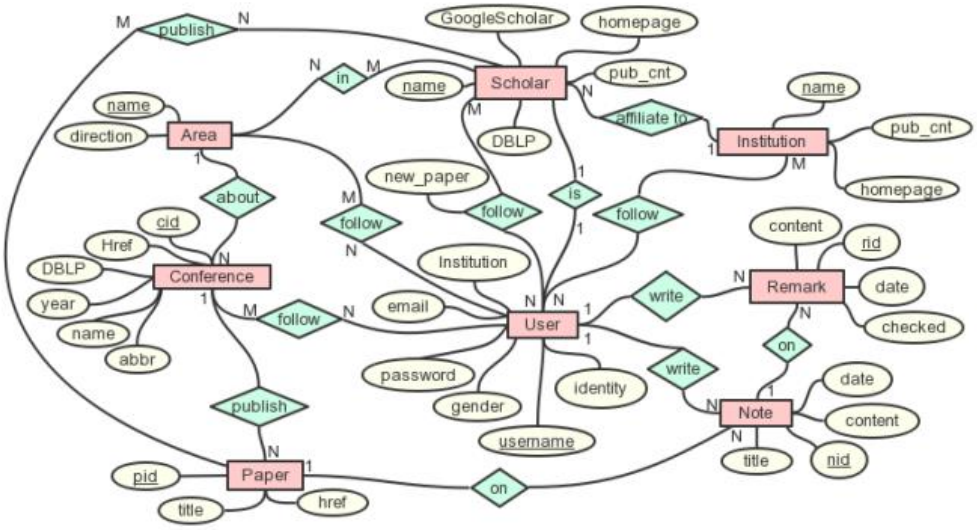


因此，可以说，Mini-CS Ranking 集合了 CSRankings，GoogleScholar，DBLP 三者的核心功能，又增加了友好的用户登录功能，并提供了更为美观交互界面。

3 ER 图与关系模式设计

3.1 ER 图

本项目设计的 ER 图如下：



3.2 关系模式与建表语句

根据以上 ER 图，我们建立以下关系模式：

- Institution(name, homepage, pub_cnt)

```
create table Institution(
    name char(30),
    homepage char(200),
    pub_cnt integer default 0,
    primary key(name)
    )
```

- Scholar(name, homepage, DBLP, GoogleScholar, affliliation, pub_cnt)

```
create table Scholar(
    name char(30),
    homepage  char(200),
    DBLP char(200),
    GoogleScholar char(200),
```

```
    affiliation char(30)
    pub_cnt integer default 0,
    primary key(name),
    foreign key(affiliation) references Institution(name)
    )
```

- Profile(username, scholar, gender, identity, institution, password, email)

```
create table Profile(
    username char(200),
    email char(200),
    password char(200),
    scholar char(30),
    gender char(10),
    identity char(10),
    institution char(30),
    primary key(username),
    foreign key(scholar) references Scholar(name)
    )
```

- Conference(cid, name, abbr, year, DBLP, Href)

```
create table Conference(
    cid integer,
    name char(50),
    abbr char(10),
    year integer,
    DBLP char(200),
    Href char(200),
    primary key(cid)
    )
```

- Paper(pid, title, href, conf_id)

```
create table Paper(
    pid integer,
    title char(100),
    href char(500),
    conf_id integer,
    primary key(pid),
    foreign key(conf_id) references Conference(cid)
```

```
    )
```

- Area(name, direction)

```
create table Area(
    name char(50),
    direction char(50),
    primary key(name)
    )
```

- Scholar_Paper(scholar_name, paper_title)

```
create table Scholar_Paper(
    scholar_name char(30),
    paper_title integer,
    primary key(scholar_name, paper_title),
    foreign key(scholar_name) references Scholar(name),
    foreign key(paper_title) references Paper(pid)
    )
```

- Scholar_Area(scholar_name, area)

```
create table Scholar_Area(
    scholar_name char(30),
    area char(50),
    primary key(scholar_name, area),
    foreign key(scholar_name) references Scholar(name),
    foreign key(area) references Area(name)
    )
```

- Conference_Area(conf_id, area)

```
create table Conference_Area(
    conf_id integer,
    area char(50),
    primary key(conf_id),
    foreign key(conf_id) references Conference(cid),
    foreign key(area) references Area(name)
    )
```

- User_Institution(user, ins)

```
create table User_Institution(
    user char(200),
    ins char(30),
    primary key(user, ins)
    foreign key(user) references Profile(username),
    foreign key(ins) references Institution(name)
    )
```

- User_Conference(user, conf)

```
create table User_Conference(
    user char(200),
    conf integer,
    primary key(user, conf),
    foreign key(user) references Profile(username),
    foreign key(conf) references Conference(cid)
    )
```

- User_Area(user, area)

```
create table User_Area(
    user char(200),
    area char(50),
    primary key(user, area),
    foreign key(user) references Profile(username),
    foreign key(area) references Area(name)
    )
```

- User_Scholar(user, sch, new_paper)

```
create table User_Scholar(
    user char(200),
    sch char(30),
    new_paper boolean default False,
    primary key(user, sch),
    foreign key(user) references Profile(username),
    foreign key(sch) references Scholar(name)
    )
```

- Note(nid, title, content, date, author, paper)

```
create table Note(
```

```
    nid integer,
    title char(100),
    content text,
    date date,
    author char(200),
    paper integer,
    primary key(nid),
    foreign key(author) references Profile(username),
    foreign key(paper) references Paper(pid)
    )
```

- Remark(rid, content, date, author, note, checked)

```
create table Remark(
    rid integer,
    content text,
    date date,
    author char(200),
    note integer,
    checked boolean default FALSE,
    primary key(rid),
    foreign key(author) references Profile(username),
    foreign key(note) references Note(nid)
    )
```

3.3 函数依赖与范式分析

关系模式 Conference 中除了非主属性对主键 cid 的函数依赖，还有：name→abbr，abbr→name。形成 abbr 和 name 的传递依赖，所以只是第二范式。为了提升查询效率，我们保留了此处冗余。如果按照以下方式分解，则可以达到第三范式：

Conference(name, year, DBLP, Href)

name_abbr(name, abbr)

其余关系模式中均只存在非主属性对主键的函数依赖，因此都是第三范式。

4 数据集获取与导入

数据量大是本项目的特征之一，单是数据库中的论文数量就达到了 5 万的数量，而论文作者关系更是达到了 20 万的级别，而获取这些数据并导入数据库，着实花费了不少精力，因此本节将重点介绍数据集的获取过程，以及根据关系模式导入数据的流程。

本节所介绍的所有代码与文件均包含在 DataProcess 文件夹下，独立于整个项目。

4.1 CS Rankings 开源数据

因为 CS Rankings 本身是开源项目，其数据集也是公开的[2]，所以我们的一部分数据直接使用了这些开源数据，以减少工作量。

这一类数据主要包括 csrankings.csv 和 institution.csv 两个文件。csrankings.csv 包含了学者姓名、所在学校、学者主页和 ScholarID；institution.csv 包含了学校的全称。其中学者与所在学校之间的关系，如果自己获取的话会比较麻烦。因为要到相应的学校网站上去爬取老师的数据，而这类通用性较强的爬虫是比较难做的，因此我们选择直接使用 CS Rankings 已经整理好的开源数据。

4.2 DBLP 开源数据集解析

众所周知，DBLP 因其便利的开源数据集而闻名，许多数据挖掘相关的研究也是在 DBLP 的数据集上展开的，所以我们在此就使用了 DBLP 的开源数据集[3]，来获取论文信息与作者信息。

但是因为 DBLP 的数据集是 xml 格式，所以无法直接使用的，需要先对其进行解析。解析的代码在 parser.py 中，作用是将 XML 格式的 DBLP 数据集，转换为便于我们处理的 paper.csv 和 paper_author.csv 文件。

解析的过程我们参考了 https://blog.csdn.net/shuaishuai3409/article/details/54316214 一文。我们调用了 Python 标准库中的 SAX 解析器进行解析，其原理是在解析 XML 的过程中，会触发事先由用户定义的事件。我们要做的工作就是重新定义一个 xml.sax.ContentHandler 的类，其中包含三种事件：

1. 元素开始事件 startElement：遇到开始标签时触发，此时可将类中的变量清零，并记录当前标签的类别。

2. 内容处理事件 characters：处理一个标签中的内容，将标签中的内容保存到局部变量中。

3. 元素结束事件 endElement：遇到结束标签时触发，将局部变量中的内容保存到全局的字典中。

[2]https://github.com/emeryberger/CSrankings

[3]http://dblp.uni-trier.de/xml/

需要注意的是，在处理时要区分父标签和子标签，即在父标签结束时才将局部变量清零，子标签结束时则只需要更新局部变量的属性。

因为 DBLP 的数据集实在太大，所以我们只取了近五年中发表在部分顶会上的论文。

具体解析过程，请参见 parser.py。

4.3 爬虫获取数据

除去以上两个数据来源，还有一部分数据是由我们自己爬取的，例如 institution_homepage.csv，其中包含了每所学校的主页网址。

爬虫的原理是，通过一个 Python 脚本，不断地对某些 URL 发送 request，接受返回的 html 代码，并通过分析处理获取需要的内容。为了获取每所学校的首页，我们可以在 Google 上搜索“CS 学校名字”，一般搜索结果的第一项就是该校计算机系的首页，因此核心思路就是：利用 request 向 Google 发送搜索请求，然后从返回的第一个搜索结果中解析出学校首页的 URL。

1. Google 提供了比较简洁的搜索 URL 接口：
   https://www.google.com.hk/search?q=<keyword>&num=1
   表示设置搜索关键词为 <keyword>，返回结果数量为 1。利用 Python 的 requests 库进行访问，并通过.text 属性获取 html 内容。

2. 对 html 代码的解析，我们可以先利用 request 获取一份 html 代码，并保存在本地后用浏览器打开，在调试器中找到我们想要的元素的标签。之后，我们使用了 Python 的 BeautifulSoup 库，这里我们可以很便捷地通过 find 方法根据标签类型和标签 id 找到相应元素，然后通过正则表达式进行字符串的匹配和处理。

需要注意的是，Google 的反爬虫机制做的相当智能，我们必须通过减慢访问速度和切换 UserAgent 来避免出现人机验证界面。

具体爬虫过程，请参见 scraper.py。

4.4 数据导入

因为调试阶段需要多次重写数据库，而数据的导入工作又十分繁琐，所以我们自己写了一个脚本专门用于导入数据。

所有已经预处理好的数据，都在 DataProcess 目录下的.csv 文件中。我们的任务就是从.csv 文件中，按表中格式解析数据，并通过 get_or_create 方法创建新数据。这里之所以使用 get_or_create 而非 create，主要是因为一次性导入全部数据耗时较长，有时我们需要多次运行脚本，而这样就有重复导入数据的情况出现，通过 get_or_create 就可以避免因为插入重复元组而报错的问题。

5 设计细节

本节主要介绍各个页面的设计细节，包括其中的涉及的难点与 SQL 语句。在实际的设计中，借助 Django 提供的便利的接口和 Python 对列表、字典等数据结构的支持，我们并不需要直接使用 SQL 语句进行查询，这样在一定程度上可以提高效率。但为了将查询功能展示的更清晰，我们还是将所涉及的 SQL 语句列在这里。

5.1 信息检索

5.1.1 论文主页

我们为每篇论文设计了单独的主页，其中包括的功能有显示论文作者、所属会议、所属领域：

- 作者：

```
select scholar_name
from Scholar_Paper
where paper_title = :pid;
```

- 所属领域：

```
select area
from Conference_Area, Paper
where pid = :pid and Paper.conf_id = Conference_Area.conf_id;
```

- 所属会议：因为会议是论文的一个属性，所以无需使用 select 语句筛选。

```
select conf_id
from Paper
where pid = :pid;
```

5.1.2 学者主页

在每位学者的主页中，我们实现了如下功能：

- 显示所属学校、DBLP、GoogleScholar、Homepage 等信息.

```
select nameaffiliation, DBLP, GoogleScholar, homepage
from Scholar
where name = :sid;
```

- 列举其发表的所有论文，按时间顺序降序排序，使用如下 SQL 语句进行查询，查询出的表记作 Papers。

```
select P.title, P.href, P.conf_id
from Scholar_Paper as SP, Paper as P,  Conference as C
where SP.scholar_name = :sid and SP.paper_title = P.pid and P.conf_id
    = C.cid
order by C.year DESC;
```

- 显示其所属领域

```
select distinct name
from Area as A, Conference_Area as CA, Papers as P
where CA.conf_id = P.conf_id and CA.area = A.name;
```

- 显示与其合作过的学者

```
select distinct A.name
from Scholar_Paper as A, Scholar_Paper as B
where A.paper_title = B.paper_title and B.scholar_name = :sid;
```

- 显示其近五年中每年发表的论文数量

```
select C.year, count(*)
from Scholar_Paper as SP, Paper as P, Conference as C
where SP.scholar_name = :sid and SP.paper_title = P.pid and P.conf_id
    = C.cid
group by C.year;
```

- 关注功能

```
insert into User_Scholar(user, sch, new_paper)
values(:user, :sid, False);
```

- 取消关注功能

```
delete from User_Scholar
where user = :user and sch = :sid;
```

注：在导入学者数据时，可以预先将学者的 pub_cnt 计算导入。因为论文的增加和删除数量非常少，pub_cnt 很少变化，所以我们可以预先将其计算好存储起来。计算使用的 SQL 语句如下：

```
update Scholar
set pub_cnt = (
select count(*)
from Scholar_Paper as SP
where SP.scholar_name = Scholar.name
);
```

5.1.3 学校主页

在每所学校的主页中，我们实现了如下功能：

- 检索该校的所有学者，按发表论文数降序排序。

```
select name
from Scholar
where affiliation = :ins
order by pub_cnt DESC;
```

- 显示学者的基本信息。查询语句同 5.1.2节。
- 显示学者所属的领域。查询语句同 5.1.2节。
- 显示学者最近发表的几篇论文，若查询结果数目大于 5，则仅取前 5 项。查询语句同 5.1.2节。
- 统计每个领域中发表论文的数量

```
select CA.area, count(*)
from Paper as P, Conference as C, Conference_Area as CA
where P.conf_id = C.cid and CA.conf_id = C.id
group by CA.area;
```

- 关注功能

```
insert into User_Institution(user, ins)
values(:user, :ins);
```

- 取消关注功能

```
delete from User_Institution
where user = :user and ins = :ins;
```

注：学校的 pub_cnt 不是指发表的论文数，而是每人每篇论文统计一次。也就是说，假如一篇论文中有两位来自 A 校的作者，那么 A 校的 pub_cnt 会增加 2。之所以这样设置，是因为考虑到每位作者对论文都有其贡献，假如 A 校的 5 位学者和 B 校的 1 位学者合著了一篇论文，那么如果说该论文对 A、B 校的 pub_cnt 贡献都为 1，未免对 A 校有失公平，因为 A 校在论文的发表中明显贡献会更大，按论文人次统计很好的解决了这一问题。

与学者的 pub_cnt 类似，学校的 pub_cnt 也可以在最初计算后存储。该操作涉及的 SQL 语句为：

```
update Insititution
set pub_cnt = (
select sum(S.pub_cnt)
from Scholar as S
where S.affiliation = Insitution.name
);
```

5.1.4 领域主页

在每个领域的主页中，我们实现了如下功能：

- 筛选该领域中的顶会

```
select conf_id
from Conference_Area
where area = :area;
```

- 列举每个顶会的几篇论文

```
select title, href
from Paper
where conf_id = :cid;
```

- 筛选该领域最顶尖的学者，取 pub_cnt 最高的五位

```
select name, homepage, DBLP, GoogleScholar, affiliation, pub_cnt
from Scholar as S
where exists (
select *
from Scholar_Papae as SP, Paper as P, Conference as C, Conference_Area
    as CA
where SP.scholar_name = S.name and SP.paper_title = P.pid and P.
   conf_id = C.cid and CA.conf_id = C.cid and CA.area = :area
```

```
)
order by pub_cnt DESC;
```

- 显示这些学者的相关信息，查询语句同 5.1.3节。
- 统计每个顶会每年的论文发表量

```
select cid, count(*)
from Conference as C, Paper as P
where P.conf_id = C.cid
group by cid
```

- 关注功能

```
insert into User_Area(user, area)
values(:user, :area);
```

- 取消关注功能

```
delete from User_Area
where user = :user and area = :area;
```

5.1.5 会议主页

在每个领域的主页中，我们实现了如下功能：

- 列举会议基本信息

```
select name, abbr, year, DBLP, Href
from Conference
where cid = :cid;
```

- 列举论文列表

```
select pid, title, href
from Paper
where conf_id = :cid;
```

- 关注功能

```
insert into User_Conference(user, conf)
values(:user, :cid);
```

- 取消关注功能

```
delete from User_Conference
where user = :user and conf = :cid;
```

5.1.6 检索页面

我们设计了一个搜索框，用于检索相关信息，涉及到的功能有：

检索学者 按照关键词检索学者，按 pub_cnt 降序排序：

```
select name, homepage, DBLP, GoogleScholar, affiliation, pub_cnt
from Scholar
where name like '%<key>%'
order by pub_cnt DESC;
```

检索论文

```
select title, href, conf_id
from Paper
where title like '%<key>%';
```

检索学校

```
select name, homepage, pub_cnt
from Institution
where name like '%<key>%'
order by pub_cnt DESC;
```

检索会议

```
select name, abbr, year, DBLP, Href
from Conference
where name like '%<key>%' or abbr like '%<key>%';
```

检索领域

```
select name, direction
from Area
where name like '%<key>%';
```

5.2 排行榜

在 ranklist.html 中，我们对所有的学校进行了排名，根据其 pub_cnt，涉及的功能与 SQL 语句如下：

- 对所有学校按 pub_cnt 降序排名

```
select name, homepage, pub_cnt
from Institution
order by pub_cnt DESC;
```

- 统计各所学校的教职工人数

```
select I.name, count(*)
from Institution as I, Scholar as S
where I.name = S.affiliation
group by I.name;
```

- 根据所选领域，筛选出每所学校相应的老师

```
select I.name, S.name
from Institution as I, Scholar as S
where I.name = S.affiliation and exists (
select *
from Scholar_Paper as SP, Paper as P, Conference_Area as CA
where SP.scholar_name = S.name and SP.paper_title = P.pid and P.
    conf_id = CA.conf_id and CA.area in :area_list
)
group by I.name;
```

注：受查询效率所限，我们没有实现按领域排名的功能，实际上这个功能的查询语句可以如下表示：

```
with Result(ins, cnt) as (
select I.name, count(*)
from Institution as I, Scholar as S, Scholar_Paper as SP, Paper as P,
   Conferece_Area as CA
where I.name = S.affiliation and S.name = SP.scholar_name and SP.
   paper_title = P.pid and P.conf_id = CA.conf_id and CA.area in :area_list
group by I.name
)
select I.name, R.cnt
from Institution as I, Result as R
where I.name = R.ins
```

```
order by R.cnt DESC;
```

5.3 用户信息

5.3.1 用户个人中心

Profile 显示用户的基本信息。涉及的 SQL 语句是：

```
select username, email, scholar, gender, identity, institution
from Profile
where username=:name;
```

Edit Profile 提供用户修改个人信息的途径。涉及的 SQL 语句是：

```
update Profile
set identity=:identity, gender=:gender, email=:email, institution=:ins,
   password=:pw
where username=:name;
```

Follow

- 显示用户关注的学者、会议、领域及机构。涉及的 SQL 语句有：

```
select sch
from User_Scholar
where user=:user;
select ins
from User_Institution
where user=:user;
select area
from User_Area
where user=:user;
select conf
from User_Conference
where user=:user;
```

- 当有关注的学者发表新的论文时，会在此页面上显示提醒。涉及的 SQL 语句有：

```
select sch
from User_Scholar
where user=:request.user and new_paper=True;
```

My Note

- 显示用户写过的所有论文笔记。涉及的 SQL 语句是：

```
select title, content, date
from Note
where author=:user;
```

- 当有笔记被其他用户评论时，会在此页面上显示新消息的数量。涉及的 SQL 语句是：
  统计总消息数

```
select count(*)
from Remark, Note
where Remark.note=Note.nid and Note.author=:user and checked=False;
```

  统计每篇笔记消息数

```
select count(*)
from Remark
where note=:note and checked=False;
```

My Remark 显示用户写过的所有评论

```
select content, date
from Remark
where author=:user;
```

Add Paper 为已经认证过的用户提供添加新论文的途径。用户可以填写论文的标题、年份、会议、超链接及作者名单并提交。当用户已经通过了认证、填写的会议和学者都已经在数据库中存在并且用户是作者之一时，论文才可以添加成功。所涉及的主要 SQL 语句是：

- 更新 Paper

```
insert into Paper(title, conf_id, href)
select :title, cid, :href
from Conference
where abbr=:abbr and year=:year;
```

- 更新 Schoalr_Paper

```
insert into Scholar_Paper(scholar_name, paper_title)
values(:name, :pid);
```

- 更新 Scholar_Area

```
insert into Scholar_Area(scholar_name, area)
select :name, area
from Conference_Area
where conf_id=:conf;
```

- 更新 User_Scholar，使得关注学者的用户收到提醒

```
update User_Scholar
set new_paper=True
where sch=:a;
```

- 更新论文作者的 pub_cnt

```
update Scholar
set pub_cnt = pub_cnt + 1
where name = :name;
```

- 更新论文作者所属学校的 pub_cnt，对于每个论文作者，

```
update Insititution
set pub_cnt = pub_cnt + (
select count(*)
from Scholar
where Scholar.affiliation = Insitution.name and Scholar.name in :
    name_list
);
```

5.3.2 用户登录注册

我们实现了用户的登入、登出和注册功能，另外添加了身份认证。登入登出主要依赖 Django 自带的用户验证系统完成，比较简单，在此略去。

注册 用户输入用户名邮箱和密码，经验证没有重名后即可注册成功

- 检查是否有重名

```
select conut(*)
from Profile
where username=:username;
```

- 创建新用户

```
insert into Profile(username, email, password)
values(:username, :email, :password);
```

认证 注册成功后，系统会筛选姓名包含用户名的学者，让用户选择自己是否是其中之一；经过认证的用户拥有添加论文的额外权限

- 筛选姓名相似的学者

```
select name, affiliation
from Scholar
where name like “%:username%”;
```

- 认证后，更新用户信息

```
update Profile
set scholar=:name, institution=:affiliation
where username=:username;
```

5.3.3 论文及笔记

我们为每篇论文制作了单独的主页，展示论文信息以及这篇论文的所有笔记，登陆后可以添加笔记；点击笔记进入笔记主页，展示了笔记信息和相关评论。同样，登陆后可以添加评论。

论文主页 展示论文信息及笔记

- 筛选笔记

```
select title, content
from Note
where paper=:pid;
```

- 添加笔记

```
insert into Note(title, content, author, paper)
values(:title, :content, :user, :paper);
```

笔记主页 显示笔记、评论，及添加评论

- 筛选笔记信息及评论

```
select Note.title, content, date, Paper.title
from Note, Paper
where Note.paper=Paper.pid and Note.nid=:nid;
select title, content, author, date
from Remark
where note=:nid
order by date desc;
```

- 如果笔记作者是当前登录作者，笔记的所有评论设置为已读

```
update Remark
set checked=True
where note=:nid;
```

- 添加评论

```
insert into Remark(content, author, note)
values(:content, :user, :note);
```

6 技术难点

本节主要列举一下我们在完成项目过程中遇到的问题和所涉及的技术难点。

1. **网站搭建**：我们使用的是 Django 框架，整个网站的搭建流程如下：首先，我们在 urls.py 中设置好每个页面的 URL 网址；然后，创建若干 html 模板，对应于这些页面；之后，书写后端函数，通过渲染模板完成响应。为了实现各个页面之间的跳转，我们在每个页面中加入超链接，然后在访问某个页面时，通过 GET 请求将相应的参数传给后端。这里使用 GET 请求的原因是，参数可以直接在 URL 中看到，方便调试，也方便直接访问。

2. **字符编码**：因为在 url 中显示的字符会进行 url 编码，前后端交互时如果不进行编码和解码就会出现问题。比如当 url 参数中带有 & 时，就必须在前端的变量后加上 |urlencode；前端发送给后端的字符串，有时也需要用 urllib.parse.unquote() 进行解码。

3. **分页处理**：像搜索结果、排行榜之类的数据条目较多的页面，我们需要使用分页技术来控制每页显示的条目数量。这里我们使用的是 Django 自带的 paginate 方法，我们只需要传入相应的参数（进行分割的查询，每个页面的条目个数，当前是第几页）就可以接受当前页面的结果。使用 Django 提供的接口，这种分页方式可以不需要每次都将所有的结果都返回，而只返回需要显示的结果，大大提升了查询效率。

4. **局部刷新**：在设计 follow 功能时，我们在每个学者的主页添加了 follow 按钮。当用户未登录或未关注该学者时，按钮显示文字为 follow；当用户已关注该学者时，按钮显示文字为 unfollow。那么，当用户点击 follow 时，我们要发送请求到后端，为用户添加关注，然后将 follow 按钮变成 unfollow 选项。

   这一过程看似简单，但实现起来却相当麻烦。一般而言，我们点击按钮后，会发送一个 GET 或 POST 请求给后端，后端处理后会重新渲染页面，这时网页会自动刷新。然而，整个页面的显示只有 follow 文字这一处发生了变化，刷新整个页面效率低下，导致使用不便捷。在这里，我们使用了 **jquery** 的局部更新工具。

   我们自行书写了一小段 js 代码来处理这一操作，大致思路是：首先我们为 follow_button 添加事件响应函数，在 follow_button 被点击时触发。接着，我们会获取按钮中文字信息，并将此作为参数，向后端发送 POST 请求。之后，后端处理该请求，并将相应的返回值以 JSON 格式传给前端。最后，前端接受后端的响应，并将相应的文字信息写入按钮中。

   与之相适应的，后端的代码也需要做相应的调整：当后端接收到请求后，首先判断请求的类型是 GET 还是 POST，若是 GET，这说明是常规的请求；若是 POST，则说明是点击 follow 按钮后，jquery 发送的请求。若为 POST，则首先判断用户

是否登录，在根据登录情况对表进行相应修改，最后将需要在按钮中显示的信息返回。

5. **排行榜显示**：排行榜页面的显示中有三个难点：**显示/隐藏教师信息**、**自动勾选领域信息**和**排行榜的局部刷新**。

   - 访问 ranklist 页面时，后端会将学校的排名连同该校老师的排名一同返回给前端，而为了不然前端显示的过于冗杂，我们默认会隐藏老师的信息。当用户点击学校旁的三角按钮时，才会显示出该校的老师。这一操作是通过 html 中 display:none 的属性完成的。在 html 中，将一个元素的 display 属性设置为 none 后，它将完全隐藏，不占任何位置，这样后端将老师信息传递给前端后，所有的显示任务就全部交给前端了。前端要做的就是利用 js，为三角按钮设置事件，当点击该按钮时，切换老师模块的 display 属性，并更改三角按钮的样式。
   - 自动勾选领域信息是指，为方便用户选择领域，用户可以通过 on/off 按钮直接勾选/取消勾选某一方向的所有领域。这一功能利用 jquery 的元素选择器可以简便地完成，添加事件：当点击 on 时，将相应方向的每个 checkbox 属性都设置为勾选。
   - 排行榜的局部刷新基本原理与上一条相同。两个不同点是：1. 提交 POST 请求时，需要获取所有 checkbox 的勾选状况，这个我们要通过 jquery 获取每个 checkbox 的情况写入一个数组，之后将数组作为参数传入。2. 因为 ranklist 变化较大，所以后端返回时，我们需要按照模板将 ranklist 的部分重新渲染一遍，然后在前端通过.html() 方法重新该部分的代码。

6. **大数据处理**：因为我们的网站中涉及的数据量较大，所以查询的效率难免会受到限制，这也是在设计网站时困扰我们的一点。一种比较合理的处理方式是，允许一些数据的冗余，比如提前计算并维护 pub_cnt 这种修改次数较少，但经常被用于查询，且查询效率很低的属性。虽然会增大所占用的空间，但是考虑到其对查询效率的提升，这点空间还是值得的。

7 结语

本项目到此便已圆满完成，在这个项目中确实学到了许多课堂上学不到的数据库知识，也提升了自己的项目开发能力。有几个简单的感想：

- 现实生活中的关系模式远比课本中的更加复杂。在创建关系模式的时候，为了达到更高的范式要求，我们不得不将一些本来非常自然地关系拆分出来，就比如 Conference_Area 等等。这类拆分虽然能减少局部依赖、传递依赖的产生，但也为我们的查询带来了不便，甚至可能会降低查询的效率。

- 查询效率与数据冗余之间往往存在权衡 (tradeoff)。虽然关系的合理拆分能提高关系模式的范式，但是过多的表导致我们不得不在查询中大量使用连接操作，这是效率非常低下的。相比之下，如果将冗余的程度放宽，或许能减少一些连接操作，进而提高我们的查询效率。

- 我们所做的项目主要以搜索功能为主，所以对增删改操作支持的并不是很多，这是因为“搜索引擎”类的网站天然地不支持增删改操作，这些操作涉及到的需要维护的量太多，且会造成许多不协调的问题出现，所以最好的方法还是从后端批量导入数据。尽管如此，这类“搜索引擎”项目在查询的复杂度上却是大大地超出了一般地管理系统，这一点也可以从我们第 5 节如此长的篇幅中体现。

最后，感谢队友愿意再次和我组队！面对工作量如此巨大的 Project，也正是队友的鼓励与支持，才给了我坚持下来的动力！

以及，十分感谢助教愿意检查我们的项目，提出宝贵的建议，并不厌其烦地读完这份冗长的报告！