



華東師範大學  
EAST CHINA NORMAL  
UNIVERSITY

《数据库系统及应用实践》课程项目报告  
CovidLit Search(Final Report)

小组成员：

李鹏达	10225101460
武泽恺	10225101429
王 力	10225101434

2024 年 5 月 - 6 月

# 目 录

<b>1</b>	<b>项目简介</b>	<b>1</b>
<b>2</b>	<b>数据库设计</b>	<b>1</b>
2.1	概述 .....	1
2.2	表结构 .....	2
2.2.1	实体集 .....	2
2.2.2	联系集 .....	3
2.3	E-R 图 .....	4
2.4	schema .....	5
<b>3</b>	<b>数据集</b>	<b>6</b>
3.1	数据来源 .....	6
3.2	数据处理 .....	6
3.2.1	数据集结构 .....	6
3.2.2	处理方式 .....	7
3.3	数据量 .....	8
<b>4</b>	<b>功能设计</b>	<b>8</b>
4.1	用户功能 .....	8
4.2	搜索功能 .....	8
4.3	文献功能 .....	8
<b>5</b>	<b>用户界面设计</b>	<b>9</b>
5.1	首页 .....	9
5.2	搜索页 .....	9
5.3	文献详情页 .....	10
5.4	注册与登录页 .....	11
5.5	用户页面 .....	12
<b>6</b>	<b>SQL</b>	<b>13</b>
6.1	用户相关功能 .....	14
6.2	搜索相关功能 .....	15
6.3	文献相关功能 .....	16
<b>7</b>	<b>测试与性能</b>	<b>19</b>
7.1	测试语句 .....	19
7.2	测试结果 .....	21
7.3	性能分析与优化 .....	22
<b>8</b>	<b>平台</b>	<b>23</b>
8.1	硬件环境 .....	23
8.2	软件环境 .....	24
8.2.1	操作系统 .....	24
8.2.2	编程语言及库 .....	24
8.2.3	开发环境 .....	25
<b>9</b>	<b>调优</b>	<b>25</b>
9.1	数据库调优 .....	25

9.2 SQL 语句优化 .....	27
9.2.1 LIKE 语句优化 .....	27
9.2.2 非空判断优化 .....	29
<b>10 总结</b>	<b>29</b>
<b>附录 A 所附代码文件概述</b>	<b>30</b>
<b>附录 B Milestone 2 在 Milestone 1 基础的变化</b>	<b>31</b>
<b>附录 C Final Report 在 Milestone 2 基础的变化</b>	<b>32</b>

# 1 项目简介

本项目 (“CovidLit Search”) 致力于为研究人员提供一个方便的、用户友好的 COVID-19 相关文献检索工具，以帮助他们更快地找到相关文献进行参考研究。

本项目的目标是通过提供一个简单友好的界面，使用户能够快速搜索到与 COVID-19 相关的文献，并且能够根据作者、时间和期刊等信息进行检索。此外，本项目也允许用户通过研究方向、研究对象和研究问题等信息准确检索部分文献并对其他文献进行模糊搜索，以便用户查找到最相关的文献。

另外，本项目还提供了一个用户注册和登录系统。用户可以通过注册登录后，将自己的搜索历史保存在云端，以便在不同设备上查看自己的搜索历史。用户还可以将自己感兴趣的文献加入到自己的收藏夹中或订阅期刊，以便在以后查看。

本项目名称为”CovidLit Search”，意为”COVID-19 Literature Search”，即 COVID-19 相关文献检索。

# 2 数据库设计

## 2.1 概述

为了实现文献检索系统，我们需要设计一个数据库来存储文献、期刊、作者和引用等信息。

为了实现相关功能，我们考虑使用 Entity-Relation 模型来设计数据库。我们考虑设计文章 (article)、作者 (author)、期刊 (journal) 和用户 (user) 等实体集，以及撰写 (write)、引用 (cite)、订阅 (subscribe)、收藏 (collect) 和浏览历史 (history) 等联系集。

其中，文章 (article) 与作者 (author) 通过撰写 (write) 联系集相连，表示作者撰写了文章；文章 (article) 与期刊 (journal) 通过发表 (publish) 联系集相连，表示文章在期刊上发表；文章 (article) 与文章 (article) 通过引用 (cite) 联系集相连，标识不同文章之间的引用与被引用关系；用户 (user) 与文章 (article) 通过收藏 (collect) 浏览历史 (history) 联系集相连，表示用户收藏了文章和用户的历史浏览文章；用户 (user) 与期刊 (journal) 通过订阅 (subscribe) 联系集相连，表示用户订阅了该期刊，可能希望获取该期刊的最新文章。

起初，我们考虑使用 MySQL 数据库来存储数据，并成功地将小数据集的数据导入了数据库中，完成了基本的测试。但在完成实验 5 后，我们发现 PostgreSQL 在多表的复杂查询上的性能要明显优于 MySQL，在数据规模较大的完整数据集上，PostgreSQL 的性能可能会更优。因此，我们决定改用 PostgreSQL 数据库来存储数据。

## 2.2 表结构

### 2.2.1 实体集

用户 (user) 表存储用户的基本信息，其结构如下：

字段名	类型	主键	外键	说明
<i>id</i>	INT	是		用户 ID，自动递增
<i>nickname</i>	VARCHAR(100)			用户名（昵称）
<i>email</i>	VARCHAR(200)			邮箱
<i>password</i>	VARCHAR(100)			密码（加密后）
<i>avatar</i>	VARCHAR(500)			头像
<i>motto</i>	VARCHAR(1000)			座右铭
<i>collage</i>	VARCHAR(100)			学院（学校）
<i>subscribe_email</i>	BOOLEAN			是否订阅邮件
<i>save_history</i>	BOOLEAN			是否保存历史记录

表 2.1 用户 (user) 表

文章 (article) 表存储文章的基本信息，其结构如下：

字段名	类型	主键	外键	说明
<i>id</i>	VARCHAR(50)	是		文章 ID
<i>title</i>	VARCHAR(1000)			文章标题
<i>abstract</i>	TEXT			摘要
<i>doi</i>	VARCHAR(50)			数字对象唯一标识符
<i>license</i>	VARCHAR(50)			许可
<i>publish_time</i>	DATETIME			发表时间
<i>url</i>	VARCHAR(800)			文章 URL
<i>study_type</i>	VARCHAR(500)			研究类型
<i>addressed_population</i>	VARCHAR(1000)			研究对象人群
<i>challenge</i>	VARCHAR(2000)			挑战/研究问题
<i>focus</i>	VARCHAR(100)			研究重点
<i>authors</i>	text			作者

表 2.2 文章 (article) 表

期刊 (journal) 表存储期刊的基本信息，其结构如下：

字段名	类型	主键	外键	说明
<i>name</i>	VARCHAR(100)	是		期刊名称
<i>description</i>	VARCHAR(1000)			期刊描述

表 2.3 期刊 (journal) 表

作者 (author) 表存储作者的基本信息，其结构如下：

字段名	类型	主键	外键	说明
<i>name</i>	VARCHAR(100)	是		作者姓名
<i>email</i>	VARCHAR(1000)			邮箱
<i>lab</i>	VARCHAR(1000)			所在实验室
<i>institution</i>	VARCHAR(1000)			所在机构
<i>country</i>	VARCHAR(100)			国家
<i>post_code</i>	VARCHAR(100)			邮政编码
<i>settlement</i>	VARCHAR(100)			定居点 (城市)

表 2.4 作者 (author) 表

### 2.2.2 联系集

撰写 (write) 联系集存储文章与作者之间的联系，其结构如下：

字段名	类型	主键	外键	说明
<i>author_name</i>	VARCHAR(100)	是	<i>author(name)</i>	作者姓名
<i>article_id</i>	VARCHAR(50)	是	<i>article(id)</i>	文章 ID

表 2.5 撰写关系 (write) 表

发表 (publish) 联系集存储文章与期刊之间的联系，其结构如下：

字段名	类型	主键	外键	说明
<i>journal_name</i>	VARCHAR(100)	是	<i>journal(name)</i>	期刊名称
<i>article_id</i>	VARCHAR(50)	是	<i>article(id)</i>	文章 ID
<i>volume</i>	VARCHAR(50)			卷号
<i>pages</i>	VARCHAR(200)			页码

表 2.6 发表关系 (publish) 表

引用 (cite) 联系集存储文章与文章之间的引用关系，其结构如下：

字段名	类型	主键	外键	说明
<i>citing_id</i>	VARCHAR(100)	是	<i>article(id)</i>	引用文章 ID
<i>cited_id</i>	VARCHAR(100)	是	<i>article(id)</i>	被引用文章 ID

表 2.7 引用关系 (cite) 表

收藏 (collect) 联系集存储用户与文章之间的收藏关系，其结构如下：

字段名	类型	主键	外键	说明
<i>user_id</i>	INT	是	<i>user(id)</i>	用户 ID
<i>article_id</i>	VARCHAR(50)	是	<i>article(id)</i>	文章 ID

表 2.8 收藏关系 (*collect*) 表

订阅 (*subscribe*) 联系集存储用户与期刊之间的订阅关系，其结构如下：

字段名	类型	主键	外键	说明
<i>user_id</i>	INT	是	<i>user(id)</i>	用户 ID
<i>journal_name</i>	VARCHAR(200)	是	<i>journal(name)</i>	期刊名称

表 2.9 订阅关系 (*subscribe*) 表

浏览历史 (*history*) 联系集存储用户与文章之间的浏览历史关系，其结构如下：

字段名	类型	主键	外键	说明
<i>user_id</i>	INT	是	<i>user(id)</i>	用户 ID
<i>article_id</i>	VARCHAR(50)	是	<i>article(id)</i>	文章 ID
<i>time</i>	DATETIME	是		浏览时间

表 2.10 浏览历史 (*history*) 表

## 2.3 E-R 图

根据上述设计，我们绘制了数据库的 E-R 图，如图 2.1 所示。

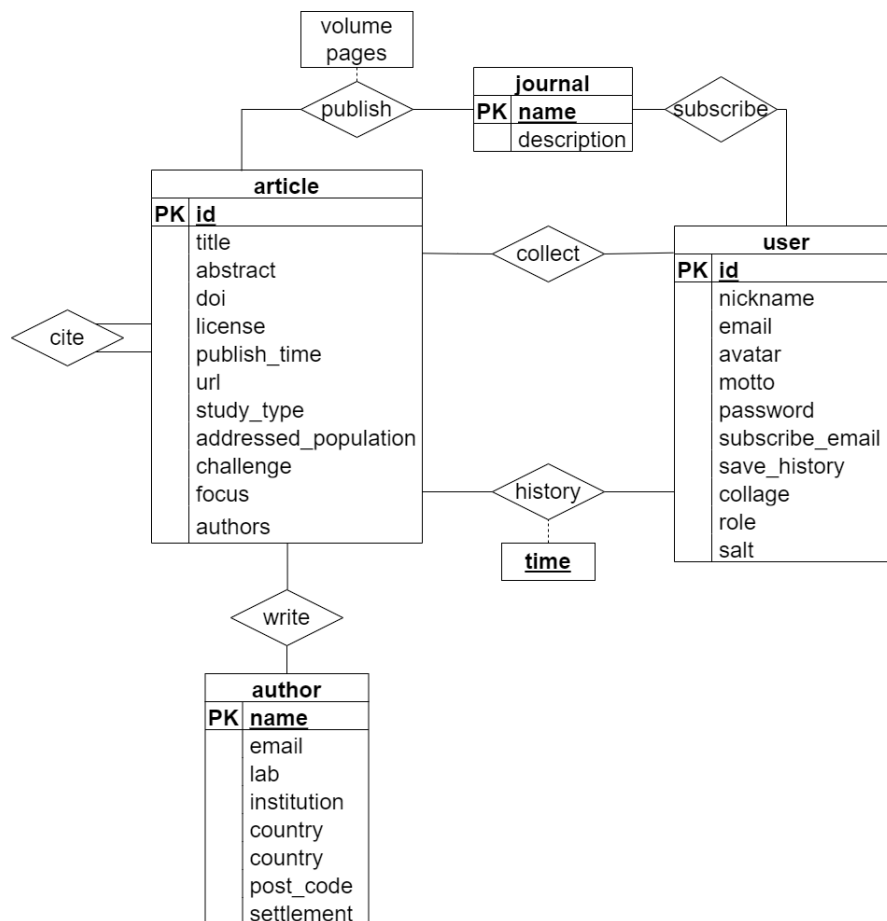


图 2.1 数据库 E-R 图

其中，PK 表示主键，FK 表示外键。

## 2.4 schema

根据上述设计，我们可以写出数据库的 `schema`，如下所示：

```

article(id, title, abstract, doi, license, publish_time, url, study_type,
        addressed_population, challenge, focus, authors)
author(name, email, lab, institution, country, post_code, settlement)
write(author_name, article_id)
journal(name, description)
publish(journal_name, article_id, volume, pages)
cite(citing_id, cited_id)
user(id, nickname, email, password, avatar, motto, collage, subscribe_email,
     save_history, role, salt)
collect(user_id, article_id)
subscribe(user_id, journal_name)
history(user_id, article_id, time)
  
```



## 3 数据集

### 3.1 数据来源

本项目使用的数据集是由美国白宫联合一系列顶尖研究机构提供的 COVID-19 Open Research Dataset (CORD-19)。该数据集可以在 Kaggle 上下载<sup>1</sup>，它包含了超过 1,000,000 篇来自 PubMed、PMC、bioRxiv 和 medRxiv 等来源的 COVID-19 相关文献的元数据，其中 400,000 篇提供全文。此外，该数据集还包括超过 6,000 篇按研究方向分类的文献元数据，包括研究方向、研究对象和研究问题等信息。

该数据集的元数据包括文献标题、作者、摘要、发布时间、期刊、全文链接等信息。我们将使用这些信息来构建我们的文献检索系统。

### 3.2 数据处理

#### 3.2.1 数据集结构

我们首先对数据集进行了初步的探索，其结构如图 3.1 所示。数据集主要包括以下几个部分：

1. `metadata.csv`: 包含了文献的元数据，包括文献标题、作者、摘要、发布时间、期刊、全文链接等信息。
2. `metadata.readme`: 包含数据集内容的更新日志。
3. `json_schema.txt`: 包含了数据集中的 JSON 文件的结构。
4. `COVID.DATA.LIC.AGMT.pdf`: 包含了数据集的使用许可协议。
5. `document_parses/`: 文件夹，包含了数据集中的文献全文，以 JSON 格式存储。
6. `Kaggle/target_tables/`: 文件夹，包含了数据集中的研究方向分类的文献元数据，包括研究方向、研究对象和研究问题等信息。
7. `cord_19_embeddings/`: 文件夹，包含了数据集中的文献的嵌入向量。

---

<sup>1</sup><https://www.kaggle.com/datasets/allen-institute-for-ai/CORD-19-research-challenge>

archive	
├── metadata.csv.....	1
├── metadata.readme.....	2
├── json_schema.txt.....	3
├── COVID.DATA.LIC.AGMT.pdf.....	4
├── document_parses.....	5
│   ├── pdf_json	
│   │   └── ...	
│   ├── pmc_json	
│   │   └── ...	
└── Kaggle	
├── target_tables.....	6
│   ├── 0_table_formats_and_column_definitions	
│   ├── 1_population	
│   │   └── ...	
│   └── ...	
├── cord_19_embeddings.....	7
│   └── cord_19_embeddings-2022-06-02.csv	

图 3.1 数据集结构

### 3.2.2 处理方式

由于数据集较大，包含了大量无用信息，且格式不是我们期望的格式，我们需要对数据集进行预处理，提取出我们需要的信息，并插入数据库中。

我们使用 Python 脚本进行处理，处理方式如下：

1. 创建一些存储过程或函数，用于简化后续大量数据的插入。
2. 根据 json\_schema.txt 文件，对 document\_parses/ 文件夹中的文献全文的 JSON 结构使用 Python 类进行建模，以便后续提取信息。
3. 读取 metadata.csv 文件，对每一条文献的元数据进行处理。
  - (a) 提取文献的基本信息，包括标题、摘要、发布时间、期刊等。
  - (b) 根据原数据中的文献全文地址，读取对应的全文文件。
    - i. 提取文献的详细作者信息，包括作者单位、邮箱、国籍等。
    - ii. 提取文献引用信息，包括引用文献的标题、作者、期刊、发布时间等。
  - (c) 将提取的信息存储到 .tbl 格式的本地文件中。
  - (d) 将 .tbl 文件导入数据库中。
4. 读取 Kaggle/target\_tables/ 文件夹中的研究方向分类的文献元数据。
  - (a) 提取文献的研究方向、研究对象和研究问题等信息。
  - (b) 将提取的信息存储到数据库中。

起初，我们采取的方案是使用 Python 脚本直接读取数据集中的文件并进行处理，每处理一篇文献就插入一次数据库，并使用数据库连接池与多线程来加速处理。但在使用全部数据集进行测试时，我们发现随着数据的插入，数据库的性能会逐渐下降，导致处理速度变慢。因此，我们改变了策略，将数据处理与数据插入分开，先将数据处理后存储到本地文件中，再使用数据库的 COPY 命令批量插入数据，以提高性能。

### 3.3 数据量

对于前期开发测试，我们使用数据集中的 1000 + 5000 余篇文献进行测试。对于后期开发，我们将使用全部数据集（1000000 + 5000 余篇文献）进行测试。最终，我们将使用全部数据集进行部署。

## 4 功能设计

### 4.1 用户功能

1. 用户注册：用户可以通过邮箱注册账号。
2. 用户登录：用户可以通过邮箱和密码登录账号。
3. 用户修改密码：用户可以通过邮箱验证或旧密码修改密码。
4. 用户信息修改：用户可以修改自己的昵称、头像、座右铭等信息。
5. 用户订阅：用户可以订阅感兴趣的期刊。
6. 用户收藏：用户可以收藏感兴趣的文献。
7. 用户浏览历史：用户可以查看自己的浏览历史。
8. 邮件订阅：用户可以订阅邮件，以从邮件中获取订阅的文献的更新。

### 4.2 搜索功能

1. 文献搜索：用户可以通过关键词搜索文献。
2. 高级搜索：用户可以通过作者、时间、期刊等信息进行高级搜索。
3. 研究方向搜索：用户可以通过研究方向、研究对象和研究问题等信息进行搜索。
4. 文献推荐：系统可以根据用户的搜索历史推荐相关文献。

### 4.3 文献功能

1. 文献详情：用户可以查看文献的详细信息。
2. 文献引用：用户可以查看文献的引用信息，包括引用与被引用，直接引用和间接引用。
3. 文献跳转：用户可以跳转到文献全文所在的网址。

4. 文献推荐：系统可以根据文献的内容推荐相关文献。
5. 文献分享：用户可以将文献分享到社交媒体或邮件。

## 5 用户界面设计

### 5.1 首页

首页包含搜索、用户注册、登录、文献推荐等功能，如图 5.1 所示。

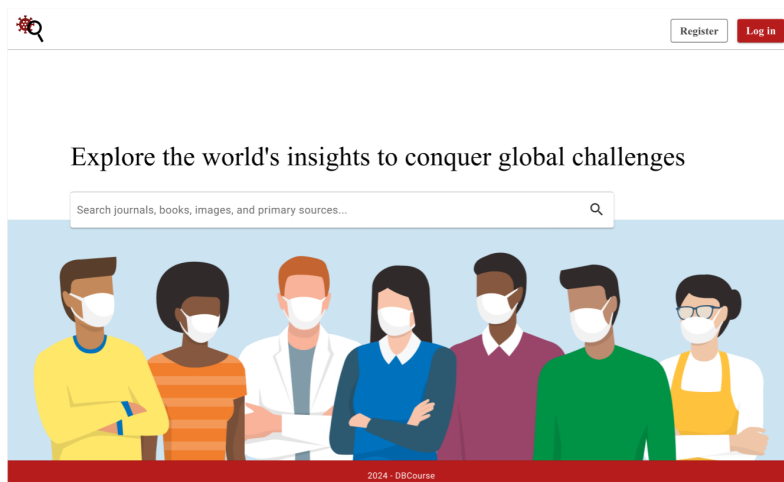


图 5.1 首页

用户在登入后，首页会显示用户昵称、头像等信息，并且可以跳转至用户个人主页、收藏、安全设置等页面，如图 5.2 所示。

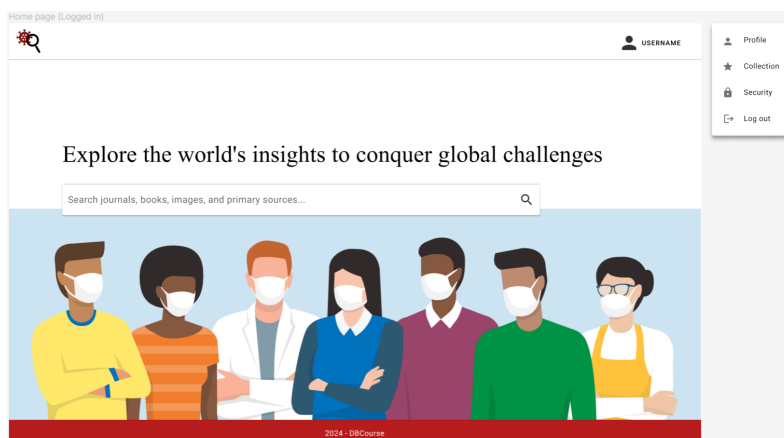


图 5.2 首页（登录后）

### 5.2 搜索页

用户在任意界面都可以进行搜索，搜索结果会显示在搜索页。搜索页包含搜索框、高级搜索、研究方向搜索等功能，用户可以对搜索结果进行排序，或对搜索结果进行筛选，如图

5.3 所示。

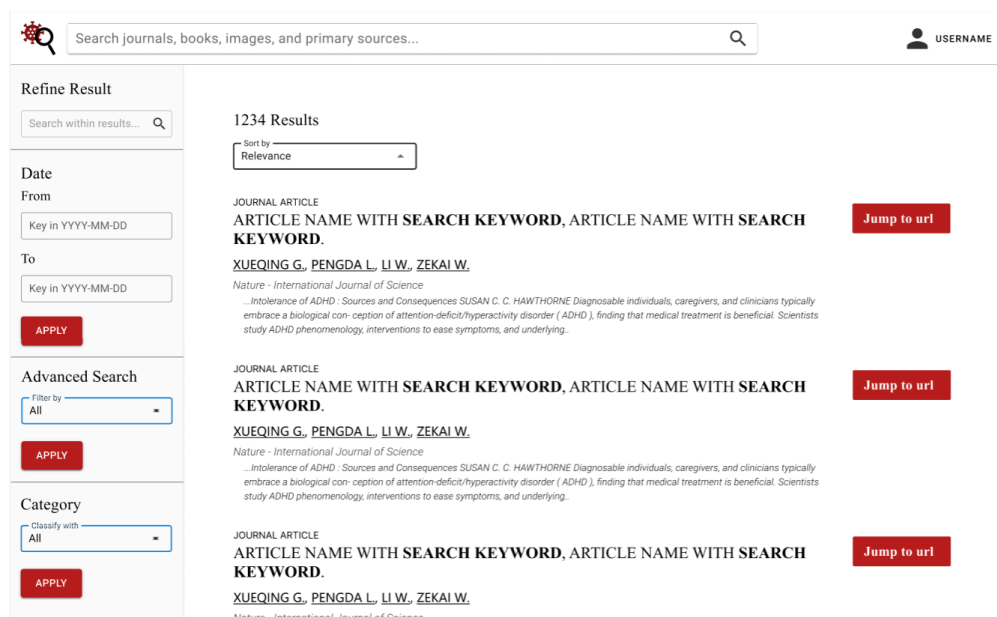


图 5.3 搜索页

## 5.3 文献详情页

用户点击文献，将跳转至文献页。用户可以查看文献的详细信息，包括标题、作者、摘要、发布时间、期刊等信息。用户可以查看文献的引用信息，包括引用与被引用，直接引用和间接引用。用户也可以跳转到文献全文所在的网址。如图 5.4 所示。

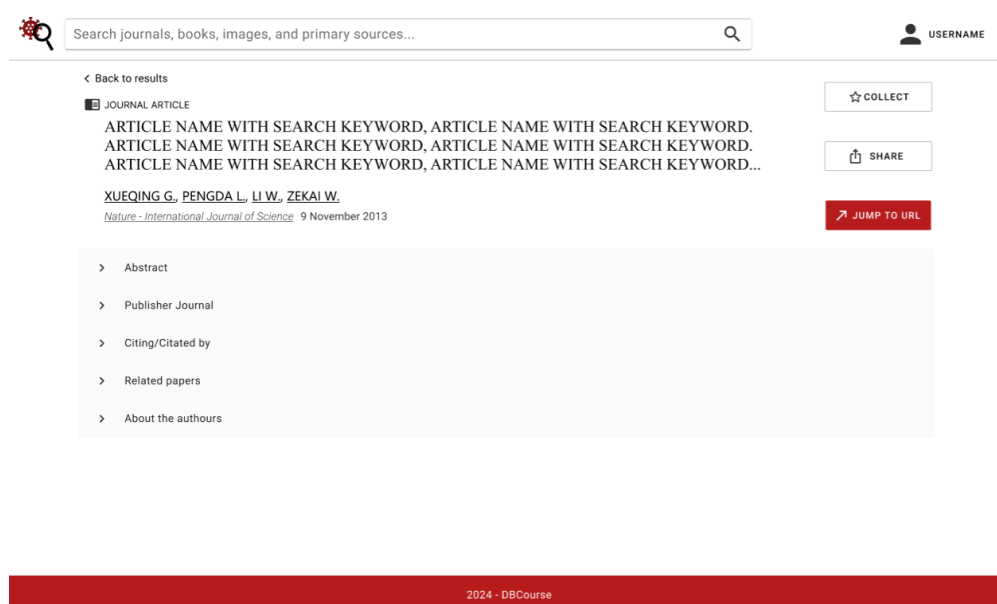


图 5.4 文献详情页

用户点击摘要、期刊、作者等栏目，可以展开查看更多相关信息，如图 5.5 所示。

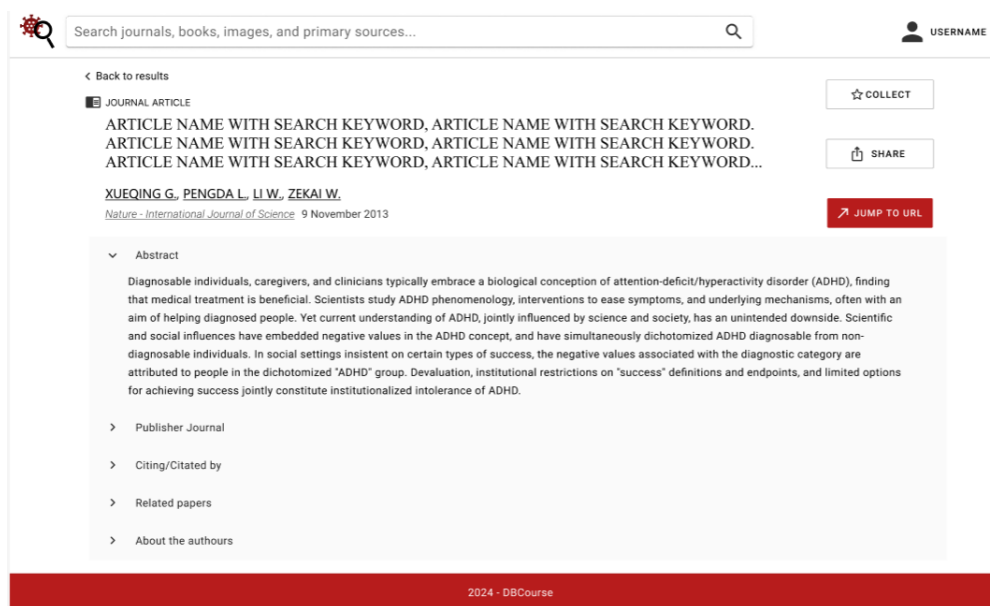


图 5.5 文献详情页（更多信息）

## 5.4 注册与登录页

用户在任意界面点击注册，将跳转至注册页。用户在注册页输入邮箱，通过验证码认证后，设置密码完成注册，如图 5.6 所示。

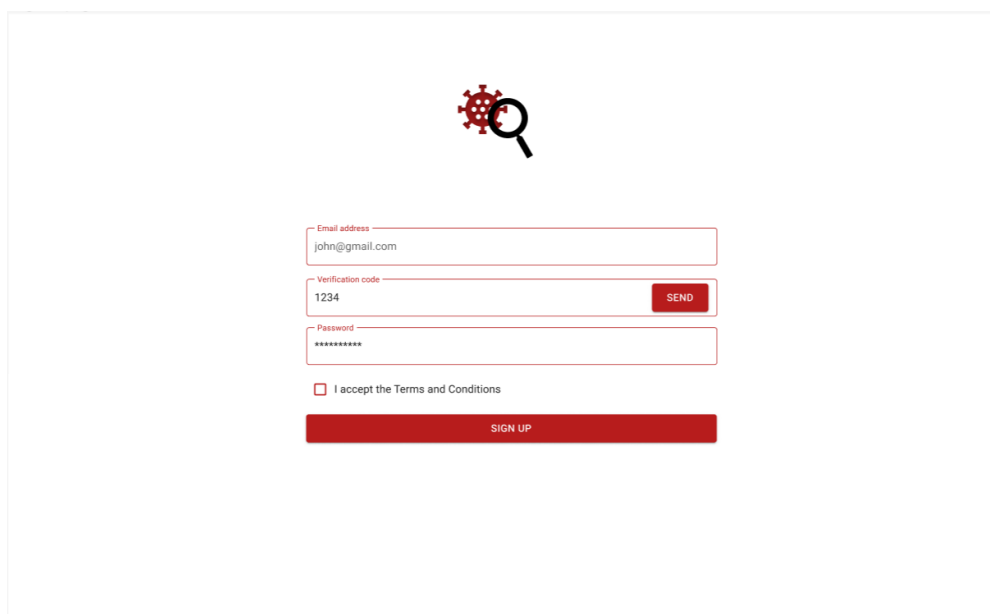
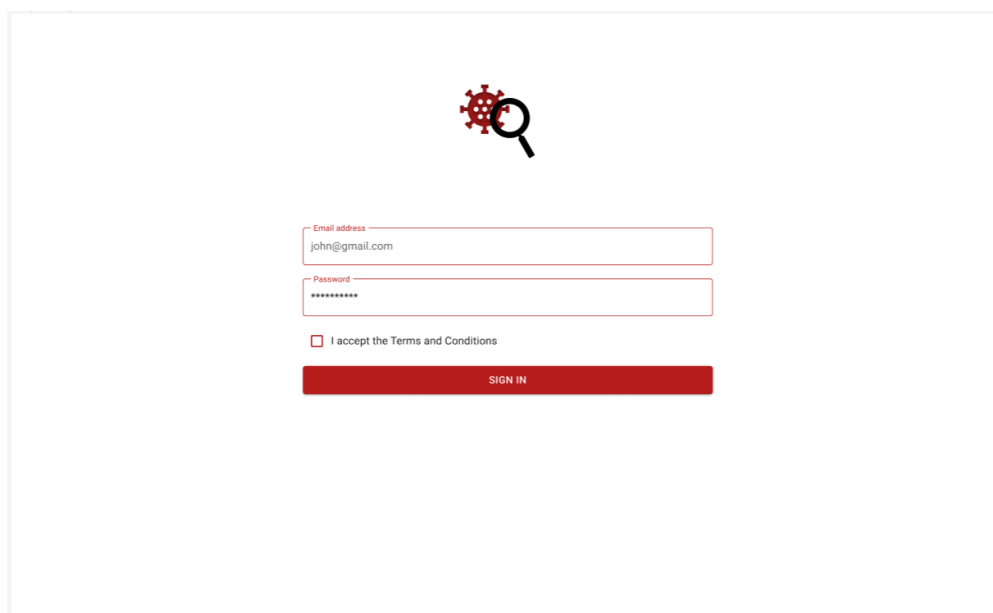


图 5.6 注册页

用户在任意界面点击登录，将跳转至登录页。用户在登录页输入邮箱和密码，完成登录，如图 5.7 所示。

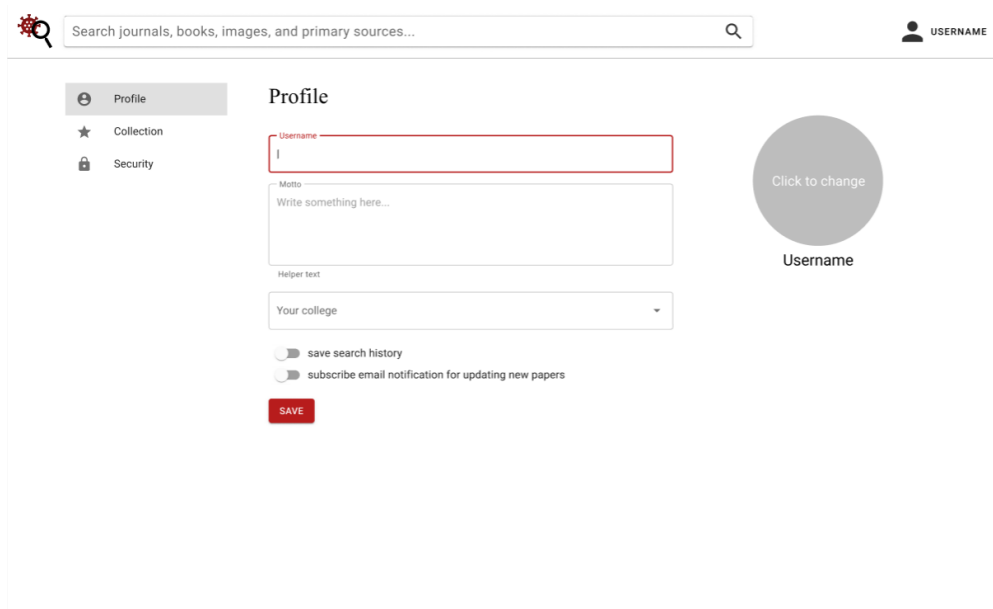


The login page features a red virus icon with a magnifying glass. Below it are two input fields: 'Email address' with the value 'john@gmail.com' and 'Password' with masked characters. A checkbox for 'I accept the Terms and Conditions' is present, followed by a red 'SIGN IN' button.

图 5.7 登录页

## 5.5 用户页面

用户在任意界面点击用户信息按钮，将跳转至用户信息页面。在用户信息页面，用户可以查看和修改自己的个人信息，包括昵称、头像、座右铭、学院等信息。用户也可以修改个人信息偏好，如是否存储浏览历史等。如图 5.8 所示。



The user profile page has a top navigation bar with a search bar and a user icon labeled 'USERNAME'. A left sidebar contains links for 'Profile', 'Collection', and 'Security'. The main 'Profile' section includes a 'Username' input field, a 'Motto' text area, a 'Your college' dropdown menu, and two toggle switches for 'save search history' and 'subscribe email notification for updating new papers'. A red 'SAVE' button is at the bottom. On the right, there is a circular placeholder for a profile picture with the text 'Click to change' and 'Username' below it.

图 5.8 用户信息

用户点击收藏按钮，将跳转至收藏页面。在收藏页面，用户可以查看自己收藏的文献，如图 5.9 所示。

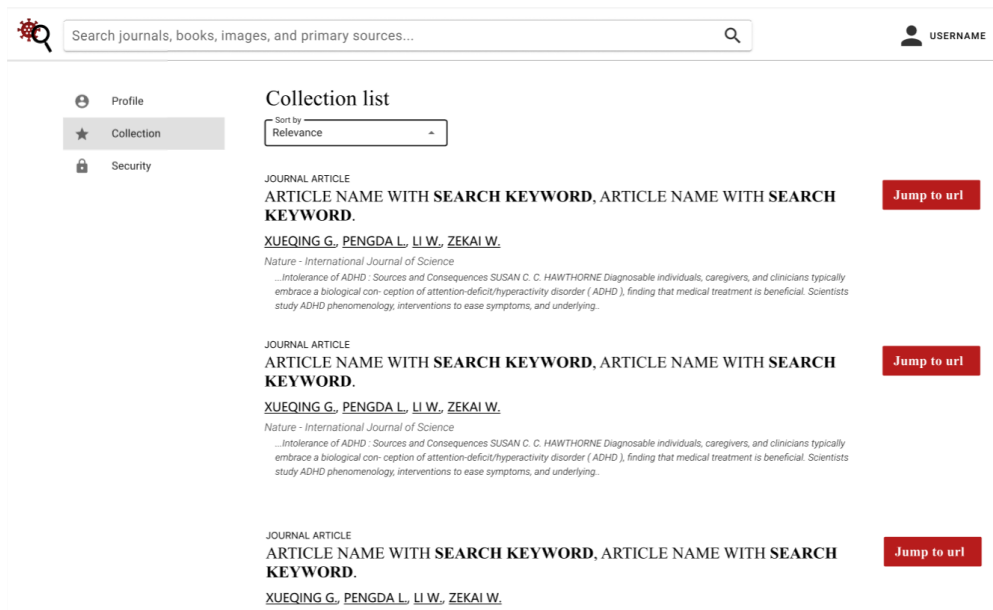


图 5.9 收藏

用户点击安全设置按钮，将跳转至安全设置页面。在安全设置页面，用户可以修改密码，如图 5.10 所示。

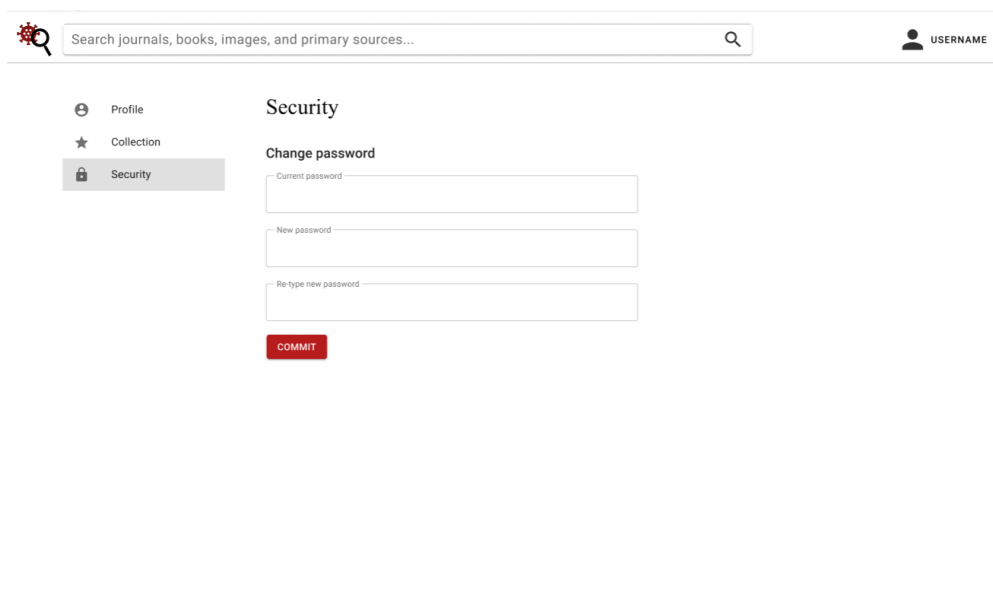


图 5.10 安全设置

## 6 SQL

在此部分，我们将展示部分 SQL 语句，用于实现系统相关功能。

由于数据量较大，我们考虑在查询时应该使用分页查询，以减少查询时间。我们使用 LIMIT 和 OFFSET 关键字来实现分页查询，例如，LIMIT n OFFSET m 或 LIMIT m, n 表示



从第  $m + 1$  行开始取  $n$  行数据。为了使报告中的 SQL 语句更加简洁，我们在此省略了分页查询的部分。

在下面的代码中，SQL 语句中的 ? 表示占位符，用于接收用户输入的参数。

## 6.1 用户相关功能

根据用户邮箱和密码进行注册:

```
1 INSERT INTO "user" ("email", "password") VALUES (?, ?);
```

根据邮箱查询用户:

```
1 SELECT * FROM "user" WHERE email = ?;
```

根据用户传入的新密码和用户的 id 来修改密码:

```
1 UPDATE "user" SET "password" = ? WHERE "id" = ?;
```

根据用户传入修改信息来对原信息进行修改:

```
1 UPDATE "user" SET "nickname" = ?, "avatar" = ?, "motto" = ?,  
2 "collage" = ?, "subscribe_email" = ? "save_history" = ?  
3 WHERE "id" = ?;
```

根据用户的 ID 和期刊的名称实现用户订阅感兴趣的期刊:

```
1 INSERT INTO "subscribe" VALUES(?,?);
```

根据用户的 ID 和文献的 ID 用户实现收藏感兴趣的文献:

```
1 INSERT INTO "collect" VALUES(?,?);
```

根据用户的 ID 来实现查看用户的浏览历史:

```
1 SELECT * FROM "history" WHERE "user_id" = ?;
```

**定时任务**——删除过期的浏览历史<sup>1</sup>:

```
1 CREATE EXTENSION pg_cron;  
2 SELECT cron.schedule(  
3 'daily_clean_history',  
4 '0 0 * * *',  
5 $$
```

<sup>1</sup>需要安装 pg\_cron 插件

```
6 DELETE FROM history
7 WHERE time < NOW() - INTERVAL '1 month';
8 $$
9 );
```

## 6.2 搜索相关功能

根据作者姓名（模糊）搜索作者信息：

```
1 SELECT * FROM "author" WHERE "name" LIKE "%?%";
```

根据作者姓名（模糊）搜索其撰写的文献：

```
1 SELECT "author_name", "id", "title", "abstract", "doi",
2       "license", "publish_time", "url"
3 FROM "write" JOIN "article"
4 ON "article_id" = "id"
5 WHERE "author_name" LIKE "%?%";
```

根据文献标题（模糊）搜索文献：

```
1 SELECT "id", "title", "abstract", "doi", "license"
2 FROM (
3     SELECT *
4     FROM "article"
5     WHERE "title" LIKE "%?%"
6 ) AS "art"
7 JOIN "publish" ON "id"="publish"."article_id"
8 JOIN "write" ON "id"="write"."article_id";
```

根据期刊名模糊搜索期刊：

```
1 SELECT *
2 FROM "journal"
3 WHERE "journal"."name" LIKE "%?%";
```

根据期刊名模糊搜索该期刊刊登的文章：

```
1 SELECT *
2 FROM "publish" JOIN "article" ON "article_id" = "id"
3 WHERE "journal_name" LIKE "%?%";
```

根据研究方向、研究对象和研究问题搜索文献：

```
1 SELECT "id", "title", "abstract", "doi", "license",
2       "publish_time", "url", "journal_name"
```

```

3  FROM (
4      SELECT *
5      FROM "article"
6      WHERE "study_type" LIKE "%?%"
7      OR "addressed_population" LIKE "%?%"
8      OR "challenge" LIKE "%?%"
9      OR "focus" LIKE "%?%"
10 ) AS "art"
11 JOIN "publish" ON "id"="publish"."article_id"
12 JOIN "write" ON "id"="write"."article_id";

```

### 6.3 文献相关功能

根据文献 ID 查询文献的详细信息：

```

1  SELECT "id","title","abstract","doi","license","publish_time","url"
2      ", "study_type",
3      "addressed_population","challenge","focus","journal_name","volume"
4      ", "pages"
5  FROM "article" JOIN "publish" ON "id"="article_id"
6  WHERE "id" = ?;

```

根据文献 ID 查询作者：

```

1  SELECT "write".author_name
2  FROM "article" JOIN "write" ON article_id = "id"
3  WHERE "id" = ?;

```

**函数**——插入被引用的文献：

```

1  CREATE OR REPLACE FUNCTION "public"."insert_cited_art"(
2      "aid" varchar,
3      "in_title" varchar,
4      "in_doi" varchar,
5      "in_publish_time" timestampz,
6      "in_journal" varchar,
7      "in_volume" varchar,
8      "in_pages" varchar,
9      "authors" varchar
10 )
11 RETURNS "pg_catalog"."void" AS $BODY$
12 DECLARE
13     cnt INT;
14     current_author VARCHAR;
15     position INT;
16 BEGIN
17     SELECT COUNT(*) INTO cnt FROM article WHERE title = in_title;
18     IF cnt = 0 THEN

```

```
19 BEGIN
20     INSERT INTO article(id, title, doi, publish_time) VALUES(aid
      , in_title, in_doi, in_publish_time)
21     ON CONFLICT (id) DO NOTHING;
22 EXCEPTION WHEN unique_violation THEN
23 END;
24 position := 1;
25 WHILE POSITION('; ' IN authors) > 0 LOOP
26     current_author := SUBSTRING(authors, 1, POSITION('; ' IN
      authors) - 1);
27 BEGIN
28     INSERT INTO author(name) VALUES(current_author)
29     ON CONFLICT (name) DO NOTHING;
30 EXCEPTION WHEN unique_violation THEN
31 END;
32 BEGIN
33     INSERT INTO write(author_name, article_id) VALUES(
      current_author, aid)
34     ON CONFLICT (author_name, article_id) DO NOTHING;
35 EXCEPTION WHEN unique_violation THEN
36 END;
37     authors := SUBSTRING(authors, POSITION('; ' IN authors) + 1);
38 END LOOP;
39 current_author := authors;
40 BEGIN
41     INSERT INTO author(name) VALUES(current_author)
42     ON CONFLICT (name) DO NOTHING;
43 EXCEPTION WHEN unique_violation THEN
44 END;
45 BEGIN
46     INSERT INTO write(author_name, article_id) VALUES(
      current_author, aid)
47     ON CONFLICT (author_name, article_id) DO NOTHING;
48 EXCEPTION WHEN unique_violation THEN
49 END;
50 BEGIN
51     INSERT INTO journal(name) VALUES(in_journal)
52     ON CONFLICT (name) DO NOTHING;
53 EXCEPTION WHEN unique_violation THEN
54 END;
55 BEGIN
56     INSERT INTO publish(journal_name, article_id, volume, pages)
      VALUES(in_journal, aid, in_volume, in_pages)
57     ON CONFLICT (journal_name, article_id) DO NOTHING;
58 EXCEPTION WHEN unique_violation THEN
59 END;
60 END IF;
61 END;
62 $BODY$
```

```

63  LANGUAGE plpgsql VOLATILE
64  COST 100

```

函数——插入扩展文章：

```

1  CREATE OR REPLACE FUNCTION "public"."insert_extend_art"(
2      "aid" varchar,
3      "in_title" varchar,
4      "in_publish_time" timestampz,
5      "in_url" varchar,
6      "in_study_type" varchar,
7      "in_addressed_population" varchar,
8      "in_challenge" varchar,
9      "in_focus" varchar,
10     "in_journal" varchar
11 )
12 RETURNS "pg_catalog"."void" AS $BODY$
13 DECLARE
14     cnt INT;
15 BEGIN
16     SELECT COUNT(*) INTO cnt FROM article WHERE title = in_title;
17     IF cnt = 0 THEN
18         INSERT INTO article(id, title, publish_time, url, study_type,
19             addressed_population, challenge, focus)
20         VALUES(aid, in_title, in_publish_time, in_url, in_study_type,
21             in_addressed_population, in_challenge, in_focus)
22         ON CONFLICT (id) DO NOTHING;
23         INSERT INTO journal(name) VALUES (in_journal)
24         ON CONFLICT (name) DO NOTHING;
25         INSERT INTO publish(journal_name, article_id, volume, pages)
26         VALUES(in_journal, aid, NULL, NULL)
27         ON CONFLICT (journal_name, article_id) DO NOTHING;
28     END IF;
29 END;
30 $BODY$
31 LANGUAGE plpgsql VOLATILE
32 COST 100

```

递归查询——查询文献的直接引用和间接引用：

```

1  WITH RECURSIVE "c" AS (
2      SELECT "citing_id", "cited_id", 0 AS "flag" FROM "cite"
3      WHERE "citing_id" = ?
4      UNION
5      SELECT "citing_id", "id", 1 AS "flag" FROM "c" JOIN "article"
6      ON "c"."cited_id" = "article"."id"
7  )
8  SELECT * FROM "c";

```

## 7 测试与性能

### 7.1 测试语句

在导入了全部数据集后，我们选取了 18 条 SQL 语句进行测试，如下所示：

- 1 根据邮箱和密码以及用户昵称注册

```
1  INSERT INTO "user" ("id", nickname, email, "password") VALUES  
    (0, 'test_name', 'test_email', '13456');
```

- 2 根据邮箱查询用户:

```
1  SELECT * FROM "user" WHERE email = 'test_email';
```

- 3 根据用户传入的新密码和用户的 id 来修改密码:

```
1  UPDATE "user" SET "password" = '456789' WHERE id = 0;
```

- 4 根据用户传入修改信息来对原信息进行修改:

```
1  UPDATE "user"  
2  SET nickname = 'test_name2', avatar = 'dsaw', motto = 'sawd',  
3    collage = 'col', subscribe_email = 'true', save_history =  
    true  
4  WHERE id = 0;
```

- 5 根据用户的 ID 和期刊的名称实现用户订阅感兴趣的期刊:

```
1  INSERT INTO subscribe (user_id, journal_name) VALUES (0, '  
    Respir Res');
```

- 6 根据用户的 ID 和文献的 ID 用户实现收藏感兴趣的文献:

```
1  INSERT INTO "collect" (user_id, article_id) VALUES (0, '  
    tn4ba9le');
```

- 7 根据用户的 ID 来实现查看用户的浏览历史:

```
1  SELECT * FROM "history" WHERE user_id = 0;
```

## 8 根据作者姓名（模糊）搜索作者信息：

```
1 SELECT * FROM author WHERE name LIKE '%' || 'aa' || '%';
```

## 9 根据作者姓名（模糊）搜索其撰写的文献：

```
1 SELECT w.author_name, ar.id, ar.title, ar.abstract, ar.doi, ar
   .license, ar.publish_time, ar.url
2 FROM "write" w
3 JOIN article ar ON w.article_id = ar.id
4 WHERE w.author_name LIKE '%' || 'aa' || '%';
```

## 10 根据文献标题（模糊）搜索文献：

```
1 SELECT ar.id, ar.title, ar.abstract, ar.doi, ar.license
2 FROM article ar
3 JOIN publish p ON ar.id = p.article_id
4 JOIN write w ON ar.id = w.article_id
5 WHERE ar.title LIKE '%' || 'aa' || '%';
```

## 11 根据期刊名模糊搜索期刊：

```
1 SELECT *
2 FROM journal
3 WHERE name LIKE '%' || 'Res' || '%';
```

## 12 根据期刊名模糊搜索该期刊刊登的文章：

```
1 SELECT ar.*
2 FROM publish p
3 JOIN article ar ON p.article_id = ar.id
4 WHERE p.journal_name LIKE '%' || 'aaa' || '%';
```

## 13 根据文献 ID 查询文献的详细信息：

```
1 SELECT ar.id, ar.title, ar.abstract, ar.doi, ar.license, ar.
   publish_time, ar.url,
2 ar.study_type, ar.addressed_population, ar.challenge, ar.
   focus,
3 p.journal_name, p.volume, p.pages
4 FROM article ar
5 JOIN publish p ON ar.id = p.article_id
6 WHERE ar.id = 'tn4ba91e';
```

## 14 根据文献 ID 查询作者：

```
1  SELECT w.author_name
2  FROM article ar
3  JOIN write w ON ar.id = w.article_id
4  WHERE ar.id = 'tn4ba91e';
```

#### 15 查询文献的直接引用和间接引用：

```
1  WITH RECURSIVE c AS (
2    SELECT citing_id, cited_id, 0 AS flag
3    FROM cite
4    UNION ALL
5    SELECT cite.citing_id, cite.cited_id, 1 AS flag
6    FROM c
7    JOIN cite ON c.cited_id = cite.citing_id
8  )
9  SELECT * FROM c;
```

#### 16 测试函数 insert\_cited\_art:

```
1  SELECT insert_cited_art('test_id', 'test_title', 'test_doi', '
    2024/6/5', 'test_journal', 'test_volume', 'test_in_pages',
    '0;1');
```

#### 17 测试函数 insert\_extend\_art:

```
1  SELECT insert_extend_art('test_id', 't_title', '2024/6/5', '
    t_url', 't_type', 't_addr', 't_cha', 't_focus', 't_journal
    ');
```

## 7.2 测试结果

我们使用 EXPLAIN 命令来查看每条 SQL 语句的执行计划，以评估其性能。我们将测试结果汇总如表 7.1 所示。



表 7.1 性能测试结果

序号	执行时间/ms
1	34
2	32
3	32
4	32
5	34
6	32
7	31
8	389
9	1028
10	2063
11	441
12	362
13	526
14	158
15	505
16	36
17	270.68
18	270.865

### 7.3 性能分析与优化

从表 7.1 中可以看出，大部分 SQL 语句的执行时间都在 1000ms 以内，性能较好。但是，有两条 SQL 语句的执行时间较长，分别为 2063ms 和 1028ms。这是因为这两条 SQL 语句中包含了多个表的连接查询，导致了性能下降。我们可以通过增加索引来提高性能。

在增加了索引后，我们再次进行了测试，结果如表 7.2 所示。

表 7.2 性能测试结果（优化后）

序号	执行时间/ms
1	36
2	34
3	37
4	36
5	38
6	36
7	36
8	66
9	956

续下页

表 7.2 性能测试结果（优化后）（续表）

序号	执行时间/ms
10	639
11	350
12	118
13	417
14	39
15	35
16	37
17	3.27
18	0.824

从表 7.2 中可以看出，经过优化后，性能有一定的提升。其中，第 10 条 SQL 语句的执行时间从 2063ms 降低到 639ms。这说明增加索引对性能有一定的提升作用。改进后的性能已经可以满足系统的需求。

两次测试的对比结果如下图所示。

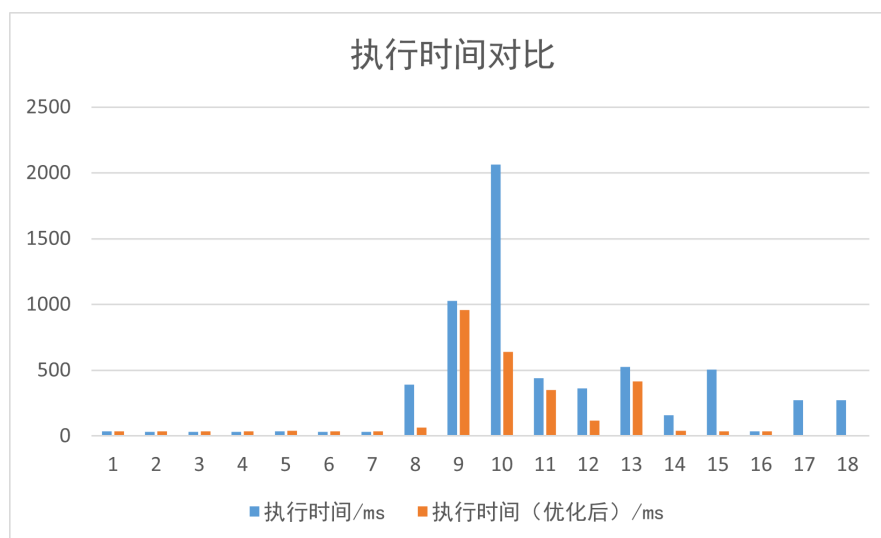


图 7.1 性能测试结果对比

## 8 平台

### 8.1 硬件环境

在项目进入开发后，我们使用了云服务器进行部署。我们选择并购置了一台阿里云的云服务器，具体配置如下：

- 规格：ecs.e-c1m4.xlarge
- CPU：4 cores (vCPU)

- 内存：16GiB
- 硬盘：100GiB ESSD Entry 云盘 (2600 IOPS)
- 网络：10Mbps 专有网络

## 8.2 软件环境

### 8.2.1 操作系统

我们考虑使用 Ubuntu 作为操作系统，具体配置如下：

- 系统：Ubuntu 22.04 64 位
- 内核：5.4.0-1045-aws
- 架构：x86

### 8.2.2 编程语言及库

我们采用数据处理、前端、后端分离的开发模式，以便于团队分工协作。具体使用的编程语言及库如下：

#### 数据

在开发前期，我们使用了 Python 进行数据处理，并将 SQL 语句内嵌到后端代码中，以实现了对数据进行增删改查。

在编写 SQL 时，我们惊奇地发现即使是效率极大程度上优于 MySQL 的 PostgreSQL，在面对千万级的数据集效率也显得十分低下。因此，我们考虑加入一些插件来提高 PostgreSQL 的效率，并拓展其功能。

具体使用的插件如下：

插件	用途
pg_cron	用于定时任务的插件
pg_trgm	用于模糊搜索的插件

表 8.1 PostgreSQL 插件

#### 前端

前端部分采用的编程语言为 TypeScript，开发框架为 React。

前端部分的技术栈如下：

库	用途
React	用于构建用户界面的库
Material-UI	用于构建 UI 组件的库
Axios	用于发送 HTTP 请求的库

表 8.2 前端技术栈

## 后端

后端部分采用的编程语言为 C#，开发框架为 ASP.NET Core。

后端部分的技术栈如下：

库	用途
ASP.NET Core	用于构建 Web 应用程序的框架
Entity Framework Core	用于访问数据库的对象关系映射（ORM）框架
AutoMapper	用于对象映射的库
JWT Token	用于认证的令牌
Swagger	用于生成 API 文档的库
MemoryCache	用于缓存数据的库
EmailSender	用于发送邮件的库

表 8.3 后端技术栈

### 8.2.3 开发环境

我们主要采用的开发环境如下：

工具	用途
Visual Studio Code	代码编辑器
Git	版本控制系统
Docker	应用程序平台
PostgreSQL	数据库管理系统
Navicat	数据库管理工具

表 8.4 开发环境

## 9 调优

### 9.1 数据库调优

首先，我们对于表结构进行了优化。在我们设计数据库时，我们尽量避免使用过多的冗余字段，以减少数据冗余。

但是，对于 *article* 表，我们采取了加入冗余字段来提高数据库的效率。*article* 表的设计起初是这样的：

字段名	类型	主键	外键	说明
<i>id</i>	VARCHAR(50)	是		文章 ID
<i>title</i>	VARCHAR(1000)			文章标题
<i>abstract</i>	TEXT			摘要
<i>doi</i>	VARCHAR(50)			数字对象唯一标识符
<i>license</i>	VARCHAR(50)			许可
<i>publish_time</i>	DATETIME			发表时间
<i>url</i>	VARCHAR(800)			文章 URL
<i>study_type</i>	VARCHAR(500)			研究类型
<i>addressed_population</i>	VARCHAR(1000)			研究对象人群
<i>challenge</i>	VARCHAR(2000)			挑战/研究问题
<i>focus</i>	VARCHAR(100)			研究重点

表 9.1 文章 (*article*) 表

考虑到 *article* 表数据量较大 ( $10^8$ )，为了避免 *article* 表与 *write* 表 JOIN 时消耗过长的时间，我们考虑为 *article* 加入冗余字段 *authors*。

字段名	类型	主键	外键	说明
...	...			...
<i>authors</i>	TEXT			作者

表 9.2 文章 (*article*) 续表

在查询文章或作者时，只需要利用 LIKE 语句查询 *article* 表，而不需要 JOIN 操作，从而期望提高查询效率。经过多次测试与验证，我们确信这样以空间换时间的方式是十分有必要的。

其次，在编写查询文章的代码时，我们发现 LIKE 的效率较低，在全数据集上经常超时 (超过 30s)，建立相应的索引后，我们发现仍然不能改善这一问题。但进行模糊查询不可避免要使用到 LIKE 语句。这不得不让我们思考如何提高 LIKE 的效率。

通过 EXPLAIN 语句检查这些语句的执行计划，发现它们采用顺序扫描而未曾使用索引。同时我们上网查询得知，LIKE 语句确实采用顺序扫描而非索引检索的方式获取内容。经过一系列网络资料的查询后，我们最终确定导入 *trgm* 插件，使用其提供的 GIN 索引来优化模糊查询。

表名	字段名	说明
<i>article</i>	<i>authors</i>	作者
<i>article</i>	<i>title</i>	文章标题
<i>author</i>	<i>name</i>	作者姓名
<i>journal</i>	<i>name</i>	期刊名
<i>publish</i>	<i>journal_name</i>	期刊名
<i>write</i>	<i>author_name</i>	作者姓名

表 9.3 采用 GIN 索引的属性

经过多次测试与验证，我们发现原本超时（超过 30s）的 SQL 语句，经过 GIN 索引的优化后，查询时间缩短到了 1s，效率得到了显著提高。因此，我们确信采用 GIN 索引的方式也是很有必要的。

最后，我们对于数据库中各个表的不同字段都建立了不同的索引，以期优化查询效率。具体的索引如下表所示。

表名	字段名	索引方法	说明
<i>article</i>	<i>id</i>	btree	文章 ID
<i>article</i>	<i>title</i>	btree	文章标题
<i>article</i>	<i>url</i>	btree	文章 URL
<i>author</i>	<i>name</i>	btree	作者姓名
<i>cite</i>	<i>cited_id</i>	btree	被引用的文章 id
<i>cite</i>	<i>citing_id</i>	btree	引用的文章 id
<i>journal</i>	<i>name</i>	btree	期刊名
<i>publish</i>	<i>article_id</i>	btree	文章 ID
<i>publish</i>	<i>journal_name</i>	btree	期刊名
<i>write</i>	<i>article_id</i>	btree	文章 ID

表 9.4 所建立的索引

显然，这些索引在提高查询效率方面起到了重要作用。

## 9.2 SQL 语句优化

在开发过程中，我们对大量 SQL 语句进行了优化，以期提高查询效率。下面，我们将举例两个优化效果最显著的 SQL 语句进行说明。

### 9.2.1 LIKE 语句优化

编写后端代码时，我们考虑将模糊查询文章标题、作者、期刊合并为同一个接口，这对我们的数据库查询带来了挑战。最初，我们采用原始的 LIKE 语句进行模糊查询，SQL 语句如下：

```
1 SELECT
2     article.*,
```

```
3         publish.*
4 FROM
5         article
6         JOIN publish ON article.ID = publish.article_id
7 WHERE
8         ( article.title LIKE '%{?}%' OR article.authors LIKE '%{?}%'
9           OR journal_name LIKE '%{?}%' )
9 ORDER BY
10        publish_time DESC;
```

这一语句在全量数据集中的千万级表 *article* 和千万级表 *publish* 上执行时，效率较低，必然超时。我们考虑使用 GIN 索引优化语句，但效果不明显，结合 EXPLAIN 语句分析我们推测是因为索引未被使用，通过上网查询证实了我们的猜想。由具体资料指出，WHERE 子句中包含 OR 将导致语句不使用索引。因此，我们考虑采用 UNION 来代替实现 or 的功能，这样对于目标关键词的模糊查询将会使用索引，以期使效率提升。

```
1         SELECT
2         article.*,
3         publish.*
4 FROM
5         article
6         JOIN publish ON article.ID = publish.article_id
7 WHERE
8         article.title LIKE '%diabetes%' UNION
9 SELECT
10        article.*,
11        publish.*
12 FROM
13        article
14        JOIN publish ON article.ID = publish.article_id
15 WHERE
16        article.authors LIKE '%diabetes%' UNION
17 SELECT
18        article.*,
19        publish.*
20 FROM
21        article
22        JOIN publish ON article.ID = publish.article_id
23 WHERE
24        journal_name LIKE '%diabetes%'
25 ORDER BY
26        publish_time DESC;
```

经过多次测试与验证，我们发现优化后的 SQL 语句成功正确使用了 GIN 索引，原本超时 (超过 30s) 的 SQL 语句，经过索引及 UNION 的优化后，查询时间缩短到了 1s，效率得到了显著提高，这证明我们的优化工作是成功的。

### 9.2.2 非空判断优化

在编写 SQL 时，我们对于一个属性是否为空的判断，我们采用了 IS NOT NULL 的方式。比如，在实现对文章 url 是否存在时，我们使用了如下面所示的代码：

```
1 SELECT
2     article.*,
3     publish.*
4 FROM
5     article
6 JOIN publish ON article.ID = publish.article_id
7 WHERE
8     article.title LIKE {?} AND article.url IS NOT NULL;
```

但是，我们发现 IS NOT NULL 的效率较低，我们考虑使用 COALESCE 函数来代替 IS NOT NULL，以期提高查询效率。我们修改为如下的代码：

```
1 SELECT
2     article.*,
3     publish.*
4 FROM
5     article
6 JOIN publish ON article.ID = publish.article_id
7 WHERE
8     article.title LIKE {?} AND AND COALESCE(article.url, '111') <> '
    111';
```

经过多次测试与验证，我们发现优化后的 SQL 语句成功提高了将近  $\frac{1}{3}$  的效率，这证明我们对非空判断的优化工作是成功的。

## 10 总结

在此项目中，我们设计了一个有关 Covid(新冠病毒) 的论文查询网站，实现了对千万级别数据库的高效查询功能与用户的收藏订阅。我们首先设计了数据库，然后使用 Python 对数据进行处理，将数据导入到数据库中。数据库设计中，我们使用了 PostgreSQL 作为数据库，编写了数十个 SQL 查询语句，并使用 pgSQL 编写了存储过程和函数。开发方面，我们使用了 C# 作为后端语言，集成了 ASP.NET Core 框架；使用 TypeScript 作为前端语言，集成了 React 框架。平台方面，我们使用了阿里云服务器进行部署，使用 Ubuntu 作为操作系统，使用 Docker 作为应用程序平台，成功将可执行文件发布到 Docker Hub<sup>1</sup>，并集成 Github Actions 实现自动化部署。调优方面，我们对数据库进行了调优，包括对表结构的优化、对 SQL 语句的优化、对索引的优化等，以提高查询效率。最终，我们成功高效实现了整个网站<sup>2</sup>，并对其进行了合理的性能测试，达到了预期的效果。项目丰富了我们的数据库设计、数据处理、前后端开发、平台部署、调优等方面的经验，是一次十分有意义的实践。

<sup>1</sup><https://hub.docker.com/r/pdli/covidlit-search>

<sup>2</sup><https://covidlit-search.pdli.site>



## 附录 A 所附代码文件概述

随本报告所附代码文件结构如下：

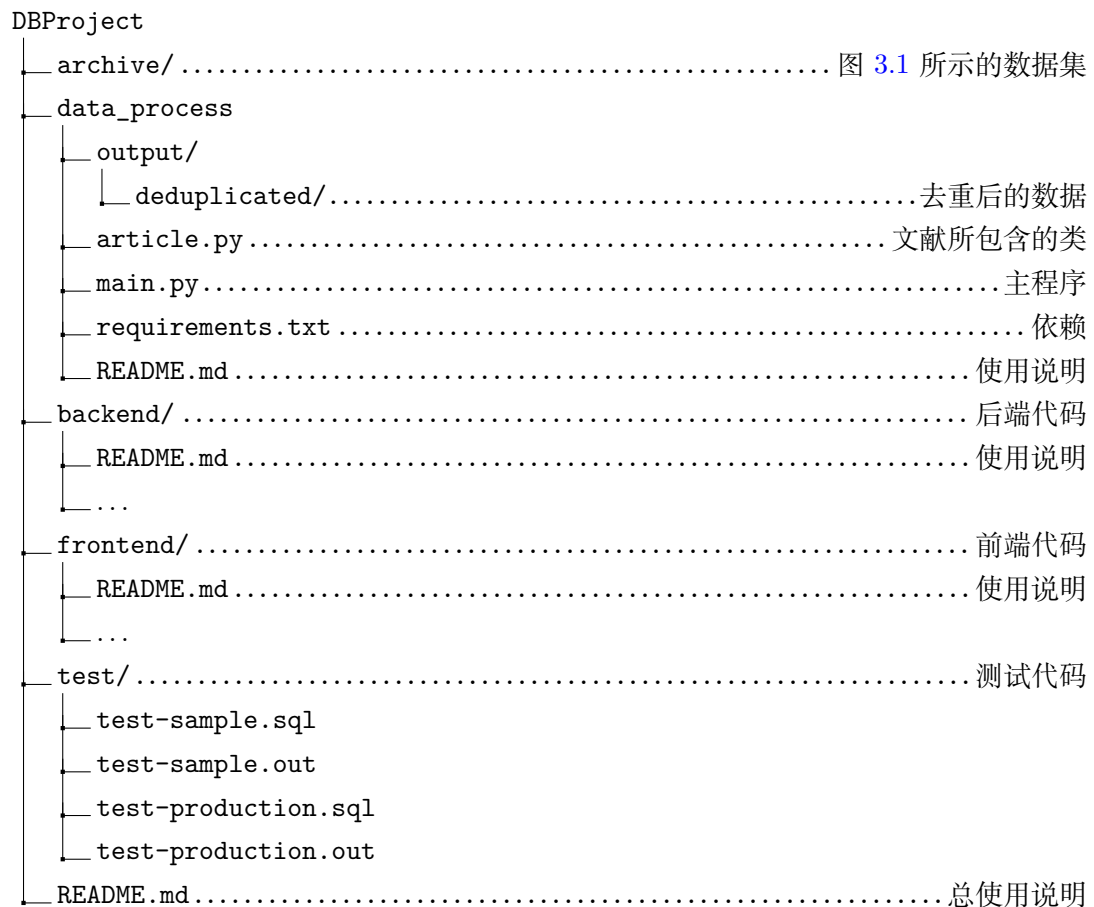


图 A.1 所附代码文件结构

## 附录 B Milestone 2 在 Milestone 1 基础的变化

1. 在 [2.1](#) 节中，我们更换了使用的数据库管理系统，从 MySQL 变为 PostgreSQL。
2. 在 [3.2.2](#) 节中，我们更改了数据处理的方式，先将数据处理后存储到本地文件中，再使用数据库的 COPY 命令批量插入数据，以提高性能。
3. 在第 [6](#) 章中，我们修改了函数/存储过程的代码和部分 SQL 语句，以符合 PostgreSQL 的语法。
4. 增加了第 [7](#) 章，对系统的性能进行了测试和分析。
5. 增加了附录[A](#)，展示了所附代码文件的结构和使用方法。

## 附录 C Final Report 在 Milestone 2 基础的变化

1. 在 2.2.1 节中，我们为 *article* 表增加了 *authors* 属性以存储作者信息。
2. 在 2.3 节中，我们修改了 ER 图，增加了 *authors* 属性。
3. 在 2.4 节中，我们修改了 *article* 表的模式，增加了 *authors* 属性。
4. 增加了第 8 章，展示了系统的硬件环境、软件环境和开发环境。
5. 增加了第 9 章，展示了数据库调优和 SQL 语句优化的过程。
6. 增加了第 10 章，总结了本次项目。