软件系统分析与设计作业

**学术成果分享平台**

**数据库设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **项目组成员信息** | | | | |
| **小组编号** | **13** | | | |
| **小组名称** | **MSI** | | | |
| **小组联系人** | **周星达** | | | |
| **学号** | **姓名** | **联系电话** | **本次实践中主要承担的工作内容** | **本次作业占比**  **(95%-105%之间)** |
| 21373339 | 周星达 | 15269992619 | 数据库设计 | 100% |
| 20101021 | 梁坤 | 18712085680 |  | 100% |
| 21373330 | 吴浩宇 | 13552318175 | 数据库设计与数据表建立 | 100% |
| 21373019 | 曾尔文 | 18979618851 | 数据库设计 | 100% |
| 21373278 | 赵乾宇 | 18355750747 | 数据库设计 | 100% |
| 21373326 | 秦茂凯 | 15611759462 | 数据库关系模型图绘制 | 100% |
| 21373328 | 王陆昊 | 13273673856 | 类图绘制与完成文档 | 100% |
| 21371220 | 杨硕 | 18098715927 |  | 100% |
| 21371134 | 田琦 | 13225685550 |  | 100% |
| 71066002 | 温舒盈 | 19658727170 |  | 100% |

2023年

版本变更历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 提交日期 | 主要编制人 | 审核人 | 版本说明 |
| v1.0 | 2023/11/15 | 吴浩宇 | 周星达 | 完成数据库设计与数据表绘制 |
| v1.1 | 2023/11/18 | 王陆昊，秦茂凯 | 周星达 | 类图的绘制与文档完成 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 引言

## 项目概述

本数据库设计说明书主要说明了学术成果分享平台的数据库设计。主要包括学术成果的管理、科研人员门户的建立、科研人员交流、学术成果的收藏、推荐等功能。其中以学术功能为主，个性推荐功能为辅，来帮助通过入驻认证的科研从业人员来维护其个人学术成果。

当然，本网站作为一个学术成果网站的主要目标是借助网站来帮助学者顺利完成其学术目标，社交、推荐功能只是为了帮助使用者交流，而不会影响学者的主要学术目标。

## 文档概述

本文档旨在说明数据相关的设计方案，文档内容依次包括数据库架构、数据库存储方案、数据库物理模型以及数据库表详细设计。数据库架构细致地说明了采用主从数据库分布部署架构的原因及优势。数据库物理模型及表设计详细说明了数据库的数据存储结构，可以以此为依据确定后续业务逻辑的具体实现并作为后续开发的重要参照。数据库存储方案和部署方案则可以作为后续搭建数据库的过程参考和指导，便于项目上线。非功能性需求保证了数据库运行时的并发访问性能以及数据传输和保存的安全性，确保良好的用户体验，使系统的安全性、稳定性和有效性得到保障。

## 相关文档

[1] 需求调研报告

[2] 需求模型

[3] 分析模型报告

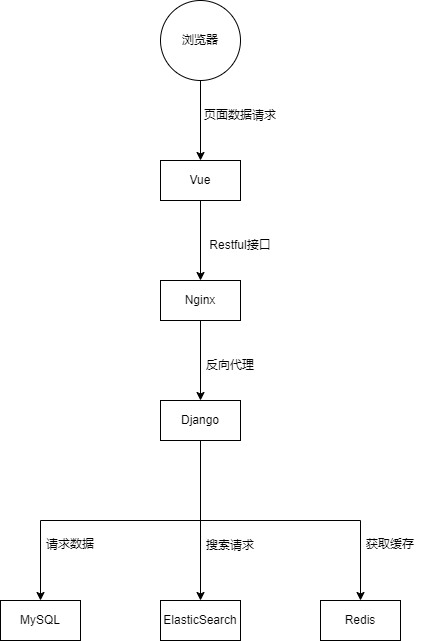
[4] 架构调研报告

[5] 架构设计报告

# 数据库架构

## 宏观架构

整体的系统架构图示如下：



**图2-1** 整体系统架构图

结构化数据：结构化数据是按照预定模型进行组织和存储的数据类型，其具有明确定义的数据模式或格式。这种数据通常以表格形式存储在关系型数据库中，使用行和列的结构。

非结构化数据：非结构化数据则是指不符合传统结构的数据形式，它没有固定的模式或格式，难以直接放入表格中或以常规方式进行组织和存储。

在本系统中，数据量很大而且数据类型、数据内容都比较的复杂，为了方便处理各种复杂数据，我们将所有数据分为结构化数据和非结构化数据两类：结构化数据数据量很大但是主要访问方式为读数据，我们使用Elasticsearch进行数据管理；非结构化数据的数据量较少，但是逻辑功能比较复杂，对于此类数据我们使用MySQL数据库进行管理。

## 数据库管理系统

本平台的数据量在百万级范围。为了确保平台的可用性并兼顾性价比，我们选择了结构化数据与非结构化数据并存的管理形式。对于不同特征的数据，我们选用Elasticsearch与MySQL对其进行管理。

针对结构化数据，我们选用MySQL数据库进行管理，MySQL数据库是比较流行的关系型数据库，在WEB开发中经常被使用，有十分丰富的接口工具并形成了完整的生态，与相关工具集成能够大幅提升开发质量与效率，非常适用于本平台结构化数据管理。

针对非结构化数据中的学术成果数据部分，我们选择使用Elasticsearch进行管理，Elasticsearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。Elasticsearch是用Java语言开发的，并作为Apache许可条款下的开放源码发布，是一种流行的企业级搜索引擎。Elasticsearch用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便；针对非结构化数据中被频繁修改调用的部分（例如热度，阅读量），我们使用 Redis 进行管理，通过启用 Redis 服务来快速管理这部分数据。

## 容量规划

对于非结构化数据，比如海量的学术成果文本数据，根据之前的需求分析，我们确定学术成果的总容量大概50G左右，为了提高平台效率。我们使用Elasticsearch搭建集群，因此不需要依赖第三方协调管理组件，自身内部就实现了集群的管理功能。ES集群由一个或多个Elasticsearch节点组成，每个节点配置相同的cluster.name即可加入集群，默认值为 “elasticsearch”。确保不同的环境中使用不同的集群名称，否则最终会导致节点加入错误的集群。 最终我们选定200G的存储容量，数据引擎配置：4核32G宽表引擎结点。

对于结构化的数据，我们使用MySQL数据库进行存储中，包括平台用户的账户信息、学者特有信息，用户对学者的关注记录、用户对学术成果的收藏记录，以及平台功能数据例如消息数据、各种申请和申诉数据。根据需求规格说明书，为了提高数据的安全性，我们预计本平台需要存储50G的基本数据，并为之预留了100G的数据。由于后期数据变动会比较大，因此本存储容量需求可能还会发生变化。最终我们选定的Elasticsearch存储容量为480G，数据引擎的配置为：2个4核8G的宽表引擎节点，2个4核16G的时序引擎节点，2个4核16G的搜索引擎节点。

## 部署方案

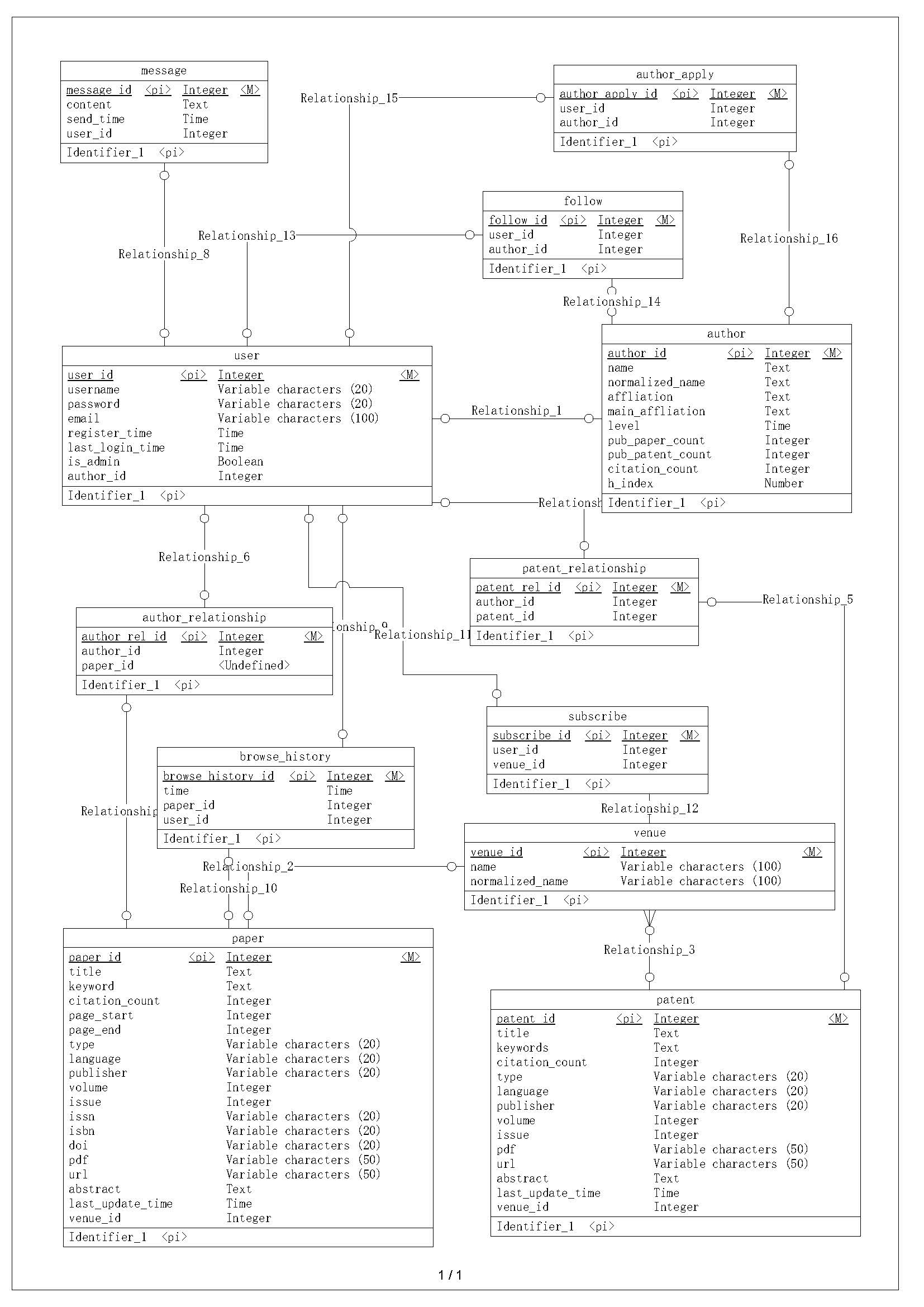
MySQL的部署十分方便，按照官网提示的安装步骤就可以安装使用。为了提高数据的安全性以及可用性，我们采用复制的方式同步数据，然后通过读写分离提升数据库的并发负载能力，从而在运行过程中数据库切换时，不会因为数据的缺失而影响业务的进展。

Elasticsearch使用非常的方便，官网对它的描述十分详细，可以充分了解它的运行机制。Elasticsearch 是一个高度可扩展的开源全文搜索和分析引擎。它允许快速，近实时地存储，搜索和分析大量数据。它通常用作底层引擎、技术，为具有复杂搜索功能和要求的应用程序提供支持。Redis 同样是开箱即用的，在启动服务时设置为守护进程启动即可。其默认的字段过期检测会保证数据库不会占用过多的磁盘空间。

# 3.数据模型



## 数据关系模型



**图3-1 数据关系模型图**

## 数据库表设计详情

user（用户）

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| username | varchar(20) |  | 用户名 |
| password | varchar(20) |  | 用户的账户密码 |
| email | varchar(100) |  | 用户注册所用的邮箱 |
| register\_time | time |  | 注册时间 |
| last\_login\_time | time |  | 上次登录时间 |
| is\_admin | bool |  | 是否为管理员 |
| author\_id | integer | 外键（author表） | 认证为哪位学者 |

author（学者）

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| name | text |  | 名字 |
| normalized\_name | text |  | 规格化名字 |
| affliation | text |  | 所属单位 |
| main\_affliation | text |  | 主要所属单位 |
| level | time |  | 职称 |
| pub\_paper\_count | integer |  | 发表文献数 |
| pub\_patent\_count | integer |  | 发表专利数 |
| citation\_count | integer |  | 被引数 |
| h\_index | real |  | h指数 |

paper（文献）

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| title | text |  | 标题 |
| keywords | text |  | 关键词 |
| citation\_count | integer |  | 所属单位 |
| page\_start | integer |  | 起始页 |
| page\_end | integer |  | 结束页 |
| type | varchar(20) |  | 文献类型 |
| language | varchar(20) |  | 语言 |
| publisher | varchar(20) |  | 发行机构 |
| volume | integer |  | 卷号 |
| issue | integer |  | 期号 |
| issn | varchar(20) |  | 国际标准刊号 |
| isbn | varchar(20) |  | 国际标准书号 |
| doi | varchar(20) |  | 数字对象标识符 |
| pdf | varchar(50) |  | 可供下载pdf的网址 |
| url | varchar(50) |  | 其它网站对此文献的收录 |
| abstract | text |  | 摘要 |
| last\_update\_time | time |  | 上次更新时间 |
| venue\_id | integer | 外键（venue表） | 发表场所 |

patent(专利)

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| title | text |  | 标题 |
| keywords | text |  | 关键词 |
| citation\_count | integer |  | 所属单位 |
| type | varchar(20) |  | 文献类型 |
| language | varchar(20) |  | 语言 |
| publisher | varchar(20) |  | 发行机构 |
| volume | integer |  | 卷号 |
| issue | integer |  | 期号 |
| pdf | varchar(50) |  | 可供下载pdf的网址 |
| url | varchar(50) |  | 其它网站对此文献的收录 |
| abstract | text |  | 摘要 |
| last\_update\_time | time |  | 上次更新时间 |
| venue\_id | integer | 外键（venue表） | 发表场所 |

author\_relationship(学者-文献关系)

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| author\_id | integer | 外键 | 作者编号 |
| paper\_id | integer | 外键 | 文献编号 |

patent\_relationship(学者-专利关系)

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| author\_id | integer | 外键 | 作者编号 |
| patent\_id | integer | 外键 | 专利编号 |

venue（发表场所）

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| name | varchar(100) |  | 场所名 |
| normalized\_name | varchar(100) |  | 规格化场所名 |

browse history（浏览历史）

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| time | time |  | 浏览时间 |
| paper\_id | integer | 外键（paper表） | 浏览的文献 |
| user\_id | integer | 外键（user表） | 用户 |

message（消息）

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| content | text |  | 消息内容 |
| send\_time | time |  | 发送时间 |
| user\_id | integer | 外键（user表） | 发送给哪个用户 |

author apply（学者门户认领申请）

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| user\_id | integer | 外键（user表） | 申请的用户 |
| author\_id | integer | 外键（author表） | 所要认领的  学者门户的学者id |

subscribe（订阅）

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| user\_id | integer | 外键（user表） | 用户 |
| venue\_id | integer | 外键（venue表） | 订阅的发表场所 |

follow（关注）

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **属性** | **主键/外键** | **属性说明** |
| id | integer | 主键 | 编号 |
| user\_id | integer | 外键（user表） | 用户 |
| author\_id | integer | 外键（author表） | 所关注的学者 |

# 缓存数据库模型

对于缓存的使用主要体现在查询功能。一般流程是查询，如果文章存在于缓存中，则直接从缓存中获取，否则从数据库查找。本网站的数据库使用到了Elasticsearch和MySQL。针对两者的特性，对其缓存有不同的使用策略。

## Elastic Search

### 自动缓存

Elasticsearch有一个自动缓存机制，称为Field Data Cache（字段数据缓存）。它用于加速经常访问的字段数据的检索，比如对某些字段进行聚合、排序或者计算。当执行搜索、聚合或排序等操作时，Elasticsearch会从磁盘加载数据到内存中进行处理。为了提高性能，Elasticsearch使用Field Data Cache来缓存这些字段数据。但是大量的字段数据缓存可能会占用大量内存，影响整体的性能。我们需要根据需要自动管理和清理这些缓存，以避免内存占用过高。

### 过滤器缓存

Elasticsearch的过滤器缓存（Filter Cache）用于存储和重复使用经常被使用的过滤器（Filters）。它的作用类似于Field Data Cache，但它主要针对查询中使用的过滤器条件。过滤器缓存能够显著提高查询的性能，特别是当相同的过滤器被频繁使用时。但并非所有的过滤器都适合缓存。某些过滤器可能是动态的或者结果集变化频繁，缓存这些过滤器可能不够有效。因此，在使用过滤器缓存时，我们需要对使用的过滤器类型和数据特性进行提前评估，以便最大化性能提升。

## MySQL

### 缓存使用策略

本网站对数据库缓存的使用主要体现在MySQL。在网站最初的设计想法，使用MySQL默认的缓存机制，当MySQL收到传入的sql语句时，它首先和先前已经解析过的sql语句进行比较，如果发现相同，则返回已缓存数据。由于架构采用Django进行数据库查询，并不存在直接sql查询。所以无需考虑sql语句是否一致导致的缓存是否使用的问题。

### 数据一致性问题

按照当前设计，如果使用MySQL默认的缓存功能，MySQL可以自动保证数据一致性问题，在有对数据库进行修改时，也会对缓存进行修改，因此不用特别处理数据一致性问题。

## Django

### 缓存配置

关于缓存的应用，不仅仅体现在数据库层面，对于Django框架本身，我们使用from django.core.cache import cache可以在把查询后的数据放入内存中，并通过@cache\_page (缓存时间)进行设置Django缓存。

### 缓存使用策略

对于Django缓存，将主要用于对论文的查询，作为比数据库缓存的互补，但是考虑到文献查询范围大，缓存命中率可能不高的情况，需要实际参照缓存对资源的使用与对速度的提升的比较进行选择，可能会调低缓存时间甚至关闭Django缓存。

## 其他缓存数据库

本网站在计算热门文献与热门领域时，使用了Redis服务。因为这些数据会被频繁的修改与更新，且数据之间关系性较弱，适用于非关系型快速读写数据库。Redis 是最为合适的选择。

# 5.安全保密设计

数据库作为管理系统的基础，通常保存着重要的文献信息与用户信息。系统中的数据一旦丢失，将需要很长时间进行恢复，有时甚至使信息系统不得不从系统初始化阶段重新开始运行。进行数据备份是保障系统安全的重要手段。数据备份需要严格按照事先制定的备份与故障恢复策略进行，并落实备份登记和检查措施。另外，系统设置用户的标识以鉴定是否是合法用户，并要求合法用户设置其密码，保证用户身份不被盗用，并在进行访问是进行Token验证；系统对不同的数据设置不同的访问级别，限制访问用户可查询的处理数据类别和内容；系统对不同用户设置不同的权限，区分不同的用户，如区分管理员和普通用户。具体的系统配置应当根据系统实际运行情况做进一步的调整。

# 6.总结

本在数据设计阶段，我们采用了MySQL作为数据库管理系统，并进行了数据库的设计和管理。本文档中详细阐述了选择MySQL的原因，并且对于数据库的架构设计做出了详尽的说明。文档中还简要介绍了部署方案，方便后续部署时进行参考。并且文档中包含了数据库表的详细设计，阐述了数据库的物理模型，为后续开发提供了重要的指导。