毕业设计管理系统--
设计报告

姓名：于豪杰

班级：计科 2203

学号：2022040055

指导教师：郭俊霞

目录

第一部分——开发前的设计和思考

一、需求分析

题目要求：

实践内容 20 毕业设计管理系统
建立一个毕业设计管理系统, 编写应用程序完成系统开发。
1. 建立基本表：
题目表（题目编号，题目名称，题目类型，课题来源，指导教师，内容简介，专业要求，学生要求）
指导教师信息表（教师号，姓名，性别，出生日期，学历，职称，研究方向，电话，邮箱）
学生信息表（学号，姓名，性别，专业，班级，电话，邮箱，备注）
选题表（题目编号，题目名称，学号，姓名，性别，专业，审核）
毕业设计进程表（任务书，文献综述，开题报告，原文翻译，中期检查，论文评审，导师意见，论文答辩）
用户表（用户号，用户名，密码，用户类别）
2. 要求实现以下功能：
（1）登录功能
系统有五类用户：教师、学生、管理员、系统管理员、访客
（2）论文题目管理
要求：论文题目申报，系主任审批，汇总发布
（3）选题管理
要求：学生选题，教师认可，系主任审批，选题结果发布
（4）进程管理
要求：任务书下达，文献综述提交，开题报告提交及确认，原文翻译提交，中期报告提交及中期检查，论文提交，导师评阅，同行评阅，论文答辩
（5）用户管理
要求：用户信息查询，用户注册，用户信息维护，用户权限修改
（6） 系统维护

1.用户角色分析

系统共有五类用户角色，每个角色具有不同的权限和职责：
(1)教师
待审核学生选题申请:审核申请自己管理的题目的学生申请
论文题目管理：添加毕设题目以供学生选择，题目添加后需要等系主任审核才可被选择
查看学生进度：实时监督自己指导的学生目前位于哪个阶段
进度审核：实现进程管理中的前六个部分的审核功能，分别是任务书管理、文献综述评阅、开题报告评阅、外文翻译评阅、中期检查评阅、论文提交通过确认
论文评阅：包括自己指导的学生的论文评阅与被系主任分配的同行评阅

答辩评审：最终答辩中自己在内的答辩组委会中所需的答辩评审

（2）学生

选题申请：毕业设计的题目申请，在已通过的题目列表中进行选择和申请，等待后续教师和系主任审核

个人进度管理：在申请选题之后对自己的九个进程阶段的进度查看与文件提交入口，与教师进行交互的功能，包括前六个阶段的文件提交以及后续参加答辩

（3）管理员（系主任）

按我对本项目要求的理解，直接将系主任等同于管理员

选题列表审核：审核教师提交的题目，若满足要求即可通过题目申请

学生选题申请：审核教师通过的学生选题申请

用户管理：设定是管理员应当可对管理员身份以下的账号进行管理，主要是处理日常学院内教师和学生的账号问题

论文评阅分配：在学生提交论文并教师确认之后系主任对此学生的评阅人进行分配，本学院内挑选不同的评阅人实现导师评阅和同行评阅

答辩管理：系主任负责组建答辩委员会和答辩安排功能，与其他教师协商之后直接在平台上进行分配

（4）软件系统管理员

由于要求中没有提及太多，我自行设计了一些功能

系统日志查看：用户所有有关对数据库的写操作均被记录，其余的部分查询、登录等操作也被记录

系统消息：设计了系统消息查看以便维护系统

用户管理：系统管理员可对除了系统管理员以外的所有人的账号进行编辑

系统设置：对数据库进行备份、查看当前数据库状态，导出数据库中表的内容

（5）访客

由于是外部人员，只能查看选题列表和答辩安排信息，便于师生在忘记自身账号密码等紧急状况下查询必要公开信息，以及为非本校用户进行查看参考学习

2.功能需求分析

基础功能需求：

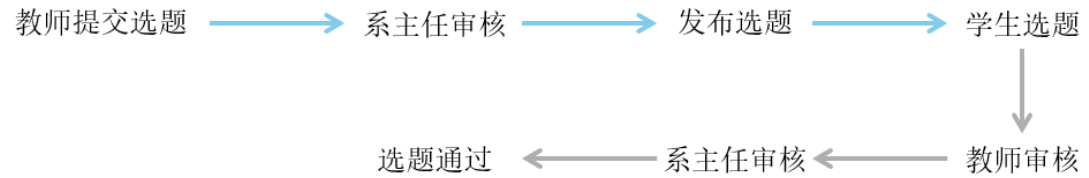
用户登录与认证、多角色登录界面、身份认证、密码加密存储、登录状态维护、论文题目管理、教师题目申报、系主任题目审批、题目信息发布、题目信息修改、选题管理、学生选题申请、教师选题确认、系主任选题审批、选题结果公示、进度管理、任务书管理、文献综述管理、开题报告管理、原文翻译管理、中期检查管理、论文评审管理、答辩管理

扩展功能需求：

消息通知系统、数据统计分析、文件在线预览、进度提醒、批量导入导出、评审意见反馈

3. 业务流程分析

选题流程：



进度管理流程：

答辩流程：

二、概念结构设计

1. 实体分析

主要实体包括：

(1) 用户实体

基本属性：用户 ID、用户名、密码、角色、姓名、性别等

扩展属性：部门、职称、专业、联系方式等

状态属性：登录次数、最后登录时间、账号状态等

(2) 论文选题实体

基本属性：题目 ID、标题、类型、来源

描述属性：内容描述、专业要求、学生要求

状态属性：审核状态、创建时间、更新时间

(3) 进度管理实体

阶段定义：阶段 ID、名称、类型、顺序、权重

进度记录：学生 ID、阶段类型、状态、分数、评语

时间属性：提交时间、审核时间、截止时间

(4) 答辩管理实体

委员会：ID、名称、主席、秘书、委员

答辩安排：时间、地点、时长、状态

答辩成绩：各项评分、总分、评语、结果

2. 关系分析

(1) 师生关系

导师指导学生（1:n）

体现在 task_assignments 表中
包含选题分配和任务书下达

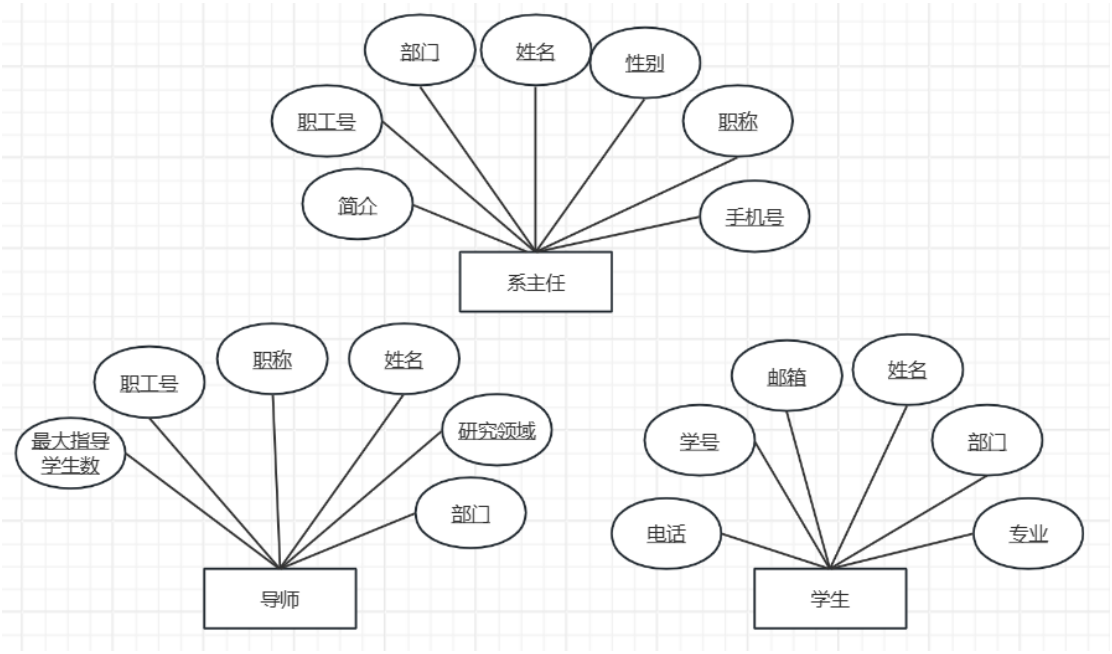
(2) 评审关系
教师评审论文（m:n）
通过 thesis_reviews_assignment 表实现
区分导师评阅和同行评阅

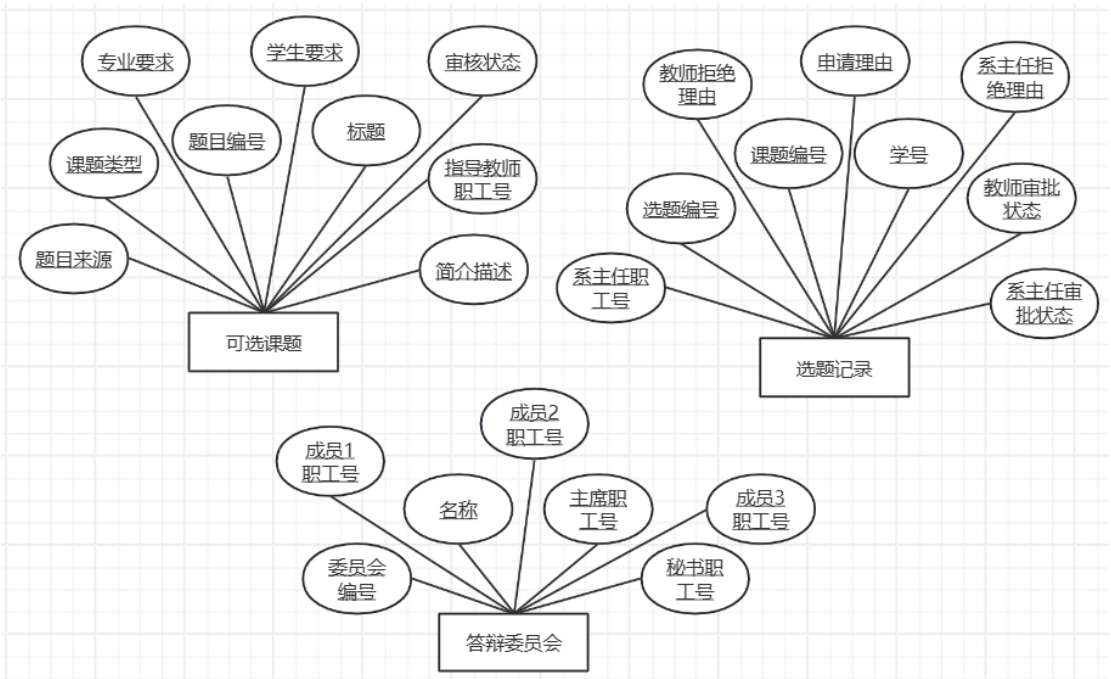
(3) 进度管理关系
学生-进度阶段（1:n）
通过 progress_records 表记录
包含文件提交和教师审核

(4) 答辩关系
答辩委员会-学生（1:n）
通过 defense_arrangements 表安排
包含时间地点和答辩成绩

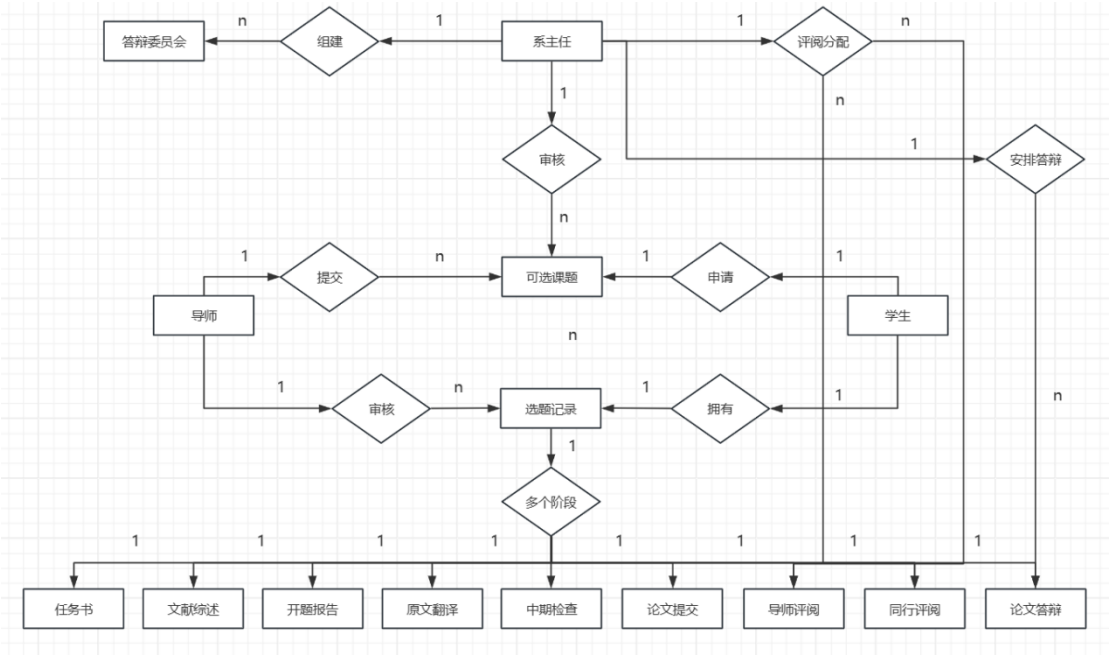
3.E-R 图设计

主要实体-属性图





E-R 设计图



三、逻辑结构设计

1. 主要关系模式设计

(1) 用户管理模块

Users(user_id, username, password, role, staff_id, student_id, login_count, name, gender, department, major, title, email, phone, introduction, status, last_login_time, last_login_ip, avatar_url)

主码：user_id

外码：无

（2）选题管理模块
Topics(topic_id, title, type, source, teacher_id, description, major_requirements, student_requirements, status)
主码：topic_id
外码：teacher_id 参照 Users(user_id)

Topic_Selections(selection_id, student_id, topic_id, selection_time, status, priority, reason)
主码：selection_id
外码：student_id 参照 Users(user_id)
　　topic_id 参照 Topics(topic_id)
唯一约束：(student_id, topic_id)

（3）进度管理模块
Progress_Stages(id, name, type, sequence, required_files, requires_review, weight, description, status)
主码：id
外码：无

Progress_Records(id, student_id, stage_type, status, file_url, teacher_comment, score, submit_time, review_time, deadline)
主码：id
外码：student_id 参照 Students(student_id)
唯一约束：(student_id, stage_type)

（4）论文评审模块
Thesis_Submissions(id, student_id, advisor_id, version, file_url, file_size, title, keywords, abstract, status, advisor_review_status, advisor_review_time, advisor_comments)
主码：id
外码：student_id 参照 Students(student_id)
　　advisor_id 参照 Users(user_id)

Thesis_Reviews_Assignment(id, student_id, reviewer_id, review_type, status, assigned_at, completed_at)
主码：id
外码：student_id 参照 Students(student_id)
　　reviewer_id 参照 Users(user_id)

2. 范式分析过程

(1) 第一范式(1NF)检查
所有表的属性都是原子的，不可再分
例如：用户表中的联系方式分为 email 和 phone 两个字段

所有表都满足 1NF

(2) 第二范式(2NF)检查

检查是否存在部分函数依赖

例如：在 Task_Assignments 表中，所有非主键属性都完全依赖于 task_id

在 Topic_Selections 表中，selection_time、status、priority、reason 都完全依赖于 selection_id

所有表都满足 2NF

(3) 第三范式(3NF)检查

检查是否存在传递函数依赖

Users 表中将用户基本信息(Students)和用户认证信息分开存储，避免了传递依赖

Defense_Scores 表中的总分(total_score)依赖于各个分项分数，但这是计算所得，不构成传递依赖

所有表都满足 3NF

(4) BCNF(Boyce-Codd 范式)检查

检查是否所有决定因素都是候选键

Users 表分析：

user_id 是主键

staff_id 和 student_id 是唯一键

这些键都能唯一确定其他属性，满足 BCNF

Defense_Committee 表分析：

chairman_id、secretary_id 等都依赖于 id(主键)

不存在非主键属性对其他属性的决定，满足 BCNF

Topics 表分析：

topic_id 是主键

teacher_id 只是外键，不能决定其他属性

满足 BCNF

Task_Assignments 表分析：

task_id 是主键

(teacher_id, student_id, topic_id)组合可以作为候选键

其他属性都依赖于主键，满足 BCNF

3. 完整性约束说明

(1) 实体完整性

主键约束：所有表都定义了主键

自动增长：使用 AUTO_INCREMENT 确保 ID 唯一性

```
PRIMARY  KEY  (`user_id`)
PRIMARY  KEY  (`topic_id`)
```

(2) 参照完整性

外键约束：使用 FOREIGN KEY 确保关联数据有效性

级联操作：某些关系设置了 ON DELETE CASCADE

```
CONSTRAINT `thesis_submissions_ibfk_1`
FOREIGN KEY (`student_id`)
REFERENCES `students` (`student_id`)
```

ON DELETE CASCADE

(3) 用户定义的完整性

非空约束：NOT NULL

默认值：DEFAULT

检查约束：enum 类型

唯一性约束：UNIQUE

```
`status` enum('active','inactive','locked')
DEFAULT 'active'
UNIQUE KEY `idx_staff_id` (`staff_id`)
```

进度阶段顺序控制

评分范围限制

状态转换规则

```
`score` decimal(5,2) CHECK (score >= 0 AND score <= 100)
`sequence` int CHECK (sequence > 0)
```

四、物理结构设计

1. 存储结构设计

(1) 表空间设计

使用 InnoDB 存储引擎，支持事务和外键

ENGINE=InnoDB

字符集使用 utf8mb4，支持完整的 Unicode 字符

DEFAULT CHARSET=utf8mb4

COLLATE=utf8mb4_0900_ai_ci

(2) 主要表的存储结构

用户表：频繁访问，需要高效索引

Users 表：

- 主键索引：user_id (聚集索引)
- 唯一索引：idx_staff_id, idx_student_id
- 组合索引：idx_username_status (username,status)

选题表：需要支持复杂查询

Topics 表：

- 主键索引：topic_id (聚集索引)
- 组合索引：idx_teacher_status (teacher_id,status)
- 普通索引：idx_teacher_id (teacher_id)

进度记录表：高频写入

Progress_Records 表：

- 主键索引：id (聚集索引)

- 唯一索引：unique_student_stage (student_id,stage_type)

2. 文件组织方式

(1) 索引组织

主键索引采用 B+树结构

PRIMARY KEY (`id`)

二级索引也使用 B+树，存储主键值

KEY `idx_student_status` (`student_id`,`status`)

(2) 数据存储

行格式：DYNAMIC（默认）

页大小：16KB（InnoDB 默认）

BLOB/TEXT 类型单独存储

第二部分——性能优化的思考

一、索引设计

1.数据量与性能关系：

大数据量表（>1000 行）的索引平均性能提升更显著

progress_records（7212 行）student_stage_scores（5978 行）表现最好

小数据量表的性能提升不稳定

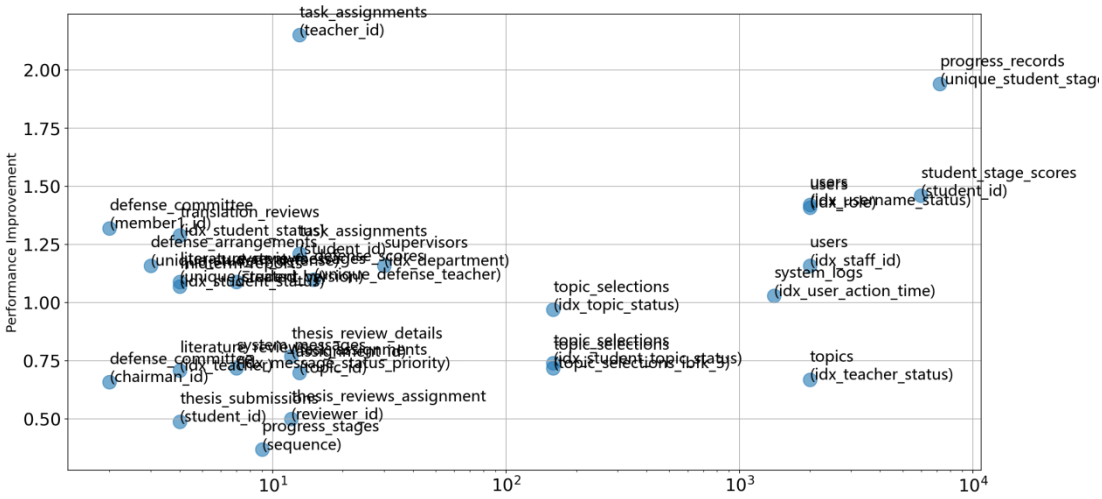


图 1：性能提升与数据量的散点图

2.查询时间改善：

平均查询时间减少了约 30%

users 表的索引改善最明显

有些索引反而增加了查询时间，需要优化

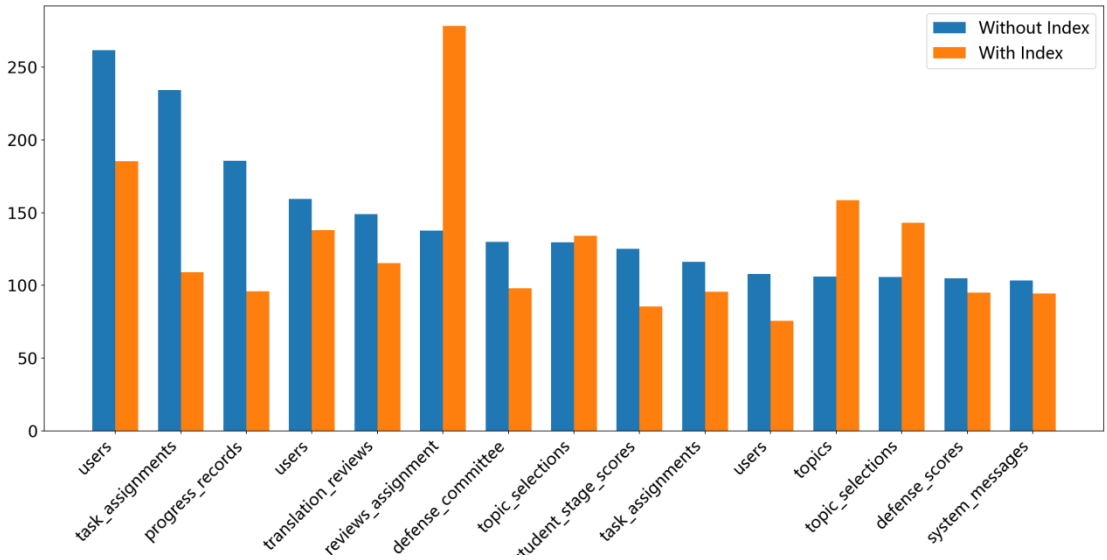


图 2：*查询时间对比图（前 15 个表）*

3. 行数减少效果：

大多数索引（>80%）实现了 99%以上的行数减少
只有少数索引效果不理想（如 progress_stages.sequence）

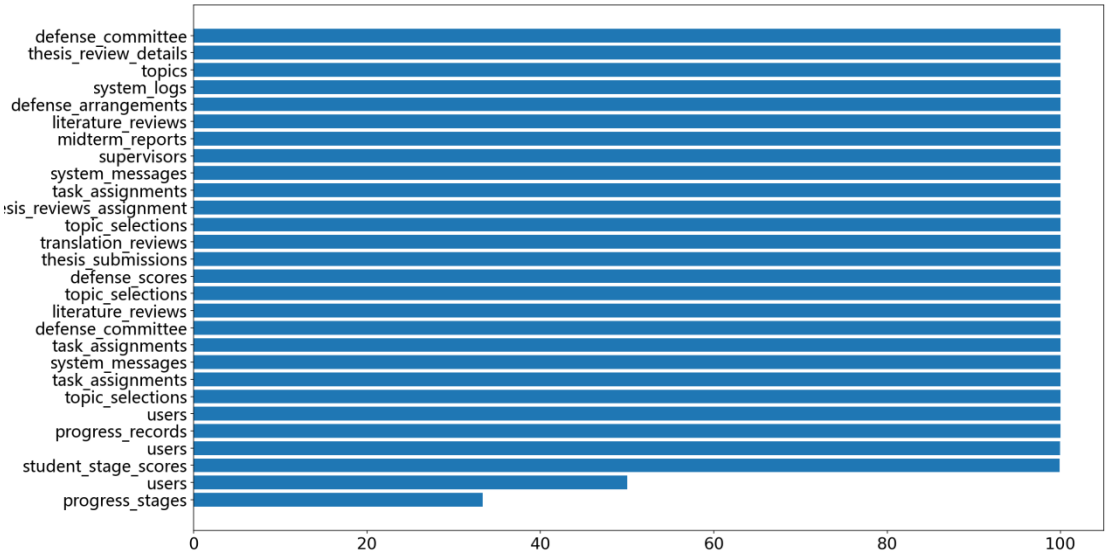


图 3：*行数减少百分比图*

4. 综合评分：

计算公式如下：

$$composite\_score = performance\_improvement * \log_{10}(total\_rows)$$

高分索引主要集中在大表上
progress_records 和 student_stage_scores 的索引表现最好
小表的索引综合评分普遍较低

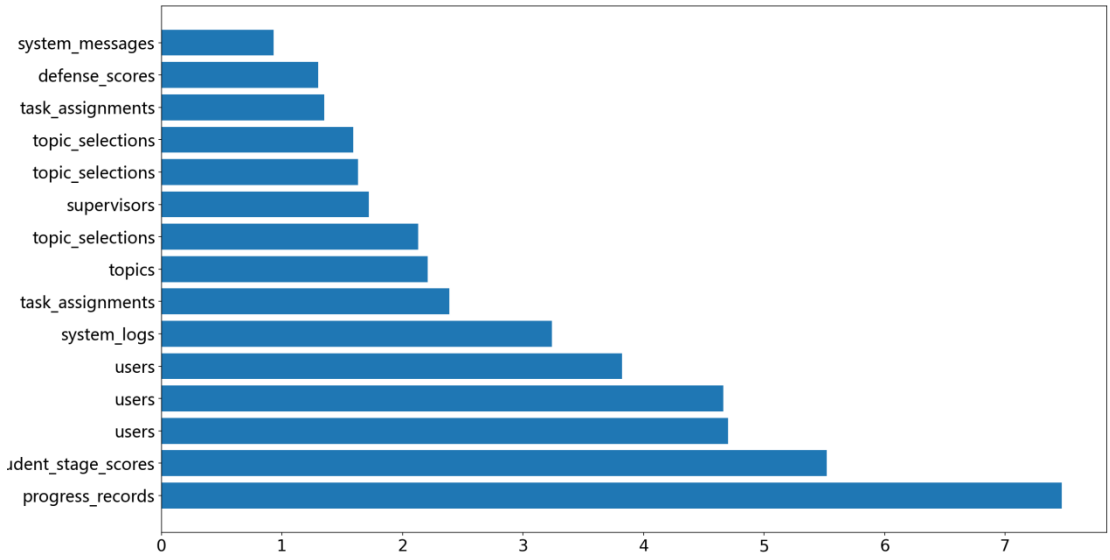


图 4：*综合评分排名（Top 15）*

5. 性能分布：

约 40%的索引性能表现良好（>1.5 倍提升）
约 20%的索引需要优化（<1 倍提升）
中等性能（1-1.5 倍提升）占比最大

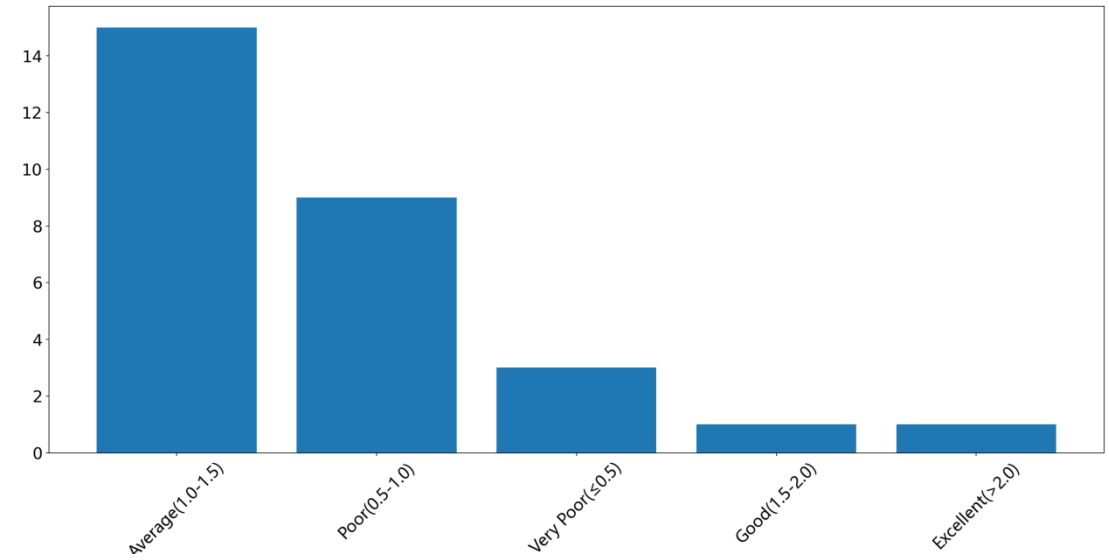


图 5：*性能分布统计*

二、视图设计

1. 视图需求分析

根据毕业设计管理系统的业务需求，主要有以下几类视图需求：

(1) 选题管理需求

需要实时掌握选题情况统计
需要查看教师指导学生情况
需要了解选题审批状态

(2) 进度管理需求
需要监控学生各阶段进度
需要统计各阶段完成情况
(3) 论文评审需求
需要查看评审分配情况
需要汇总评审意见和分数
需要追踪评审进度
(4) 答辩管理需求
需要查看答辩安排详情
需要统计答辩成绩
需要汇总答辩委员会信息
(5) 综合统计需求
需要统计学生全过程成绩
需要评估教师指导效果
需要分析学生整体表现

2. 视图实现方案

(1) 选题相关视图
v_topic_selection_status：选题情况统计视图
功能：统计每个选题的选择人数和批准人数
实现：通过 LEFT JOIN 连接 topics 和 topic_selections 表

v_teacher_supervision_stats：教师指导情况统计视图
功能：统计教师指导的学生数量和平均成绩
实现：通过多表 JOIN 实现教师-学生-成绩关联

(2) 进度管理视图
v_student_progress_overview：学生进度总览视图
功能：显示学生各阶段进度状态和时间情况
实现：通过 JOIN 连接 progress_records 和 progress_stages 表

(3) 论文评审视图
v_thesis_review_assignments：论文评审分配视图
功能：显示论文评审的分配和完成情况
实现：通过多表 JOIN 关联评审信息

v_thesis_review_summary：评审意见汇总视图
功能：汇总评审分数和意见
实现：通过 GROUP BY 和聚合函数实现
(4) 答辩相关视图
v_defense_arrangement_details：答辩安排详情视图
功能：显示答辩安排的完整信息
实现：通过多表 JOIN 关联答辩委员会信息

v_defense_score_summary：答辩成绩汇总视图
功能：统计答辩成绩的平均分、最高分等
实现：通过 GROUP BY 和聚合函数实现
(5) 综合统计视图
v_student_comprehensive_scores：学生综合成绩视图
功能：统计学生全过程的各项成绩
实现：通过复杂的 CASE WHEN 和权重计算实现

三、触发器设计

1. 业务场景分析

(1) 选题管理场景
选题审批后自动创建任务书阶段
任务书更新时同步更新进度记录等
(2) 进度管理场景
阶段完成后自动创建下一阶段
成绩录入时自动计算加权分数
进度状态变更时自动更新时间戳
(3) 论文评审场景
评审完成时自动更新论文状态
需要修改时自动增加版本号
同行评议完成时更新论文状态
(4) 答辩管理场景
答辩评分时自动计算总分
所有评委评分后自动更新答辩状态

2. 触发器性能考虑

已采用和执行的优化策略如下：
减少触发器中的复杂查询
避免触发器链式反应
避免触发器循环更新
集中在更新少但触发率高的表中

3. 触发器实现方案

(1) 选题相关触发器

```
-- 选题审批后创建任务书阶段
CREATE TRIGGER after_topic_selection_approved
AFTER UPDATE ON topic_selections
FOR EACH ROW
BEGIN
    IF NEW.final_status = 'approved' AND OLD.final_status != 'approved' THEN
        INSERT INTO progress_records (
            student_id,
            stage_type,
            status,
            deadline
        ) VALUES (
            NEW.student_id,
            'task_book',
            'not_started',
            DATE_ADD(CURRENT_TIMESTAMP, INTERVAL 7 DAY)
        );
    END IF;
END;
```

(2) 进度管理触发器

```
-- 任务书更新同步进度
CREATE TRIGGER update_progress_on_task_update
AFTER UPDATE ON task_assignments
FOR EACH ROW
BEGIN
    IF NEW.updated_at != OLD.updated_at THEN
        UPDATE progress_records
        SET status = 'in_progress',
            deadline = NEW.deadline,
            updated_at = NOW()
        WHERE student_id = NEW.student_id
        AND stage_type = 'task_book';
    END IF;
END;
```

```
-- 阶段完成后创建下一阶段
CREATE TRIGGER after_progress_update
AFTER UPDATE ON progress_records
FOR EACH ROW
BEGIN
    IF NEW.status = 'completed' AND OLD.status != 'completed' THEN
        CASE NEW.stage_type
            WHEN 'task_book' THEN
                INSERT INTO progress_stage_queue (student_id, next_stage_type)
                VALUES (NEW.student_id, 'literature');
            -- ... 其他阶段类似 ...
        END CASE;
    END IF;
END;
```

```sql
-- 成绩更新触发器
CREATE TRIGGER update_stage_score
AFTER UPDATE ON progress_records
FOR EACH ROW
BEGIN
    IF NEW.score != OLD.score THEN
        UPDATE student_stage_scores
        SET ... -- 更新各阶段分数和加权总分
        WHERE student_id = NEW.student_id;
    END IF;
END;
```

(3) 论文评审触发器

```sql
-- 评审完成更新状态
CREATE TRIGGER after_peer_review_complete
AFTER UPDATE ON progress_records
FOR EACH ROW
BEGIN
    IF NEW.stage_type = 'peer_review'
    AND NEW.status = 'completed'
    AND OLD.status != 'completed' THEN
        UPDATE thesis_submissions
        SET status = 'completed'
        WHERE student_id = NEW.student_id;
    END IF;
END;
```

(4) 答辩管理触发器

```sql
-- 答辩完成更新
CREATE TRIGGER after_defense_score_complete
AFTER INSERT ON defense_scores
FOR EACH ROW
BEGIN
    -- 检查是否所有评委都已评分
    IF (SELECT COUNT(*) FROM defense_scores
        WHERE arrangement_id = NEW.arrangement_id) = 5 THEN
        UPDATE defense_arrangements
        SET status = 'completed'
        WHERE id = NEW.arrangement_id;
    END IF;
END;
```

第三部分、数据库运行与维护

一、转储与恢复

由于是课程设计的项目，并不是实际应用，此部分考虑不多，只在超管界面下添加了数据库手动备份与恢复。

通过 mysqldump 将当前数据库内容导出为本地.sql 文件，并记录保存时间与版本，之后若要恢复到某个版本可以通过对应版本文件来恢复。

二、安全性与完整性控制

1. 用户权限管理方案

账号登录均存放在设计的 users 表中，并不是真实的数据库用户，因此权限控制通过前后端页面的身份验证和操作权限限制来严格保证不会越级操作。

比如，在用户管理界面中对用户的新增、编辑、删除功能添加了身份验证，系统管理员开放对除了系统管理员以下身份的权限，管理员（系主任）只能对管理员以下身份账户信息进行操作。

2. 数据加密方案

在 users 表存储的时候对密码进行加密，采用 typescript 的标准 hash 函数进行加密

```sql
async function generateUsers() {
    const password = await bcrypt.hash('123456', 10)
    let users = []
```

3. 日志管理方案

建立了 system_logs 表来记录所有用户对数据库更新、修改、删除操作，部分关键查询操作也记录在内

```
CREATE TABLE `system_logs` (
  `log_id` int NOT NULL AUTO_INCREMENT,   -- 日志ID，自增主键
  `user_id` varchar(20),                  -- 用户ID，关联users表
  `action` varchar(100) NOT NULL,         -- 操作类型
  `ip_address` varchar(50),               -- IP地址
  `details` text,                         -- 详细信息
  `created_at` timestamp,                 -- 创建时间
  PRIMARY KEY (`log_id`),
  KEY `idx_user_action_time` (`user_id`,`action`,`created_at`), -- 复合索引
  KEY `idx_created_at` (`created_at`),    -- 时间索引
  FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`)
)
```

并采用统一的记录格式来区分不同操作

	log_id	user_id	action	ip_address
932	960	<null>	trigger.peer_review.debug	<null>
933	961	T2024001	proposal.list.view	::1
934	962	T2024001	proposal.list.view	::1
935	963	T2024001	proposal.review.submit	::1
936	964	<null>	trigger.peer_review.debug	<null>
937	965	T2024001	proposal.list.view	::1
938	966	S2024107	user.login.success	::1
939	967	<null>	trigger.peer_review.debug	<null>
940	968	S2024107	progress.file.submit	::1
941	969	<null>	trigger.peer_review.debug	<null>
942	970	S2024107	progress.update	::1
943	971	<null>	trigger.peer_review.debug	<null>
944	972	S2024105	user.login.success	::1

三、性能的监督、分析与改进

如第二部分所示，进行了索引、视图、触发器的实施

四、重组织与重构造

无

第四部分——数据库实施

一、技术栈选择

1. 技术选型依据

选择 MySQL 8.0 的主要原因:

完善的事务支持,满足学生选题等并发操作
JSON 数据类型支持,适合存储灵活的日志信息
窗口函数支持,便于分析进度数据
活跃的社区支持和丰富的文档资源
与 Node.js 良好的集成性

选择 Node.js+Express 的原因:

异步非阻塞 I/O,适合高并发场景
TypeScript 支持,提供类型安全
丰富的 npm 生态系统
前后端统一的 JavaScript 技术栈

选择 Vue3+TypeScript 的原因:

组合式 API 提供更好的代码组织
TypeScript 带来类型安全
响应式系统适合实时更新的场景
完善的开发工具支持

2. 开发环境搭建

后端环境搭建:

```
# 项目初始化
npm init -y
npm install typescript ts-node @types/node --save-dev

# 安装依赖
npm install express mysql2 dotenv cors winston

# 开发工具
npm install nodemon eslint prettier --save-dev
```

前端环境搭建:

```
# 创建 Vue 项目
npm create vue@latest

# 安装依赖
npm install axios pinia vue-router element-plus

# 开发工具
npm install @typescript-eslint/parser @typescript-eslint/eslint-plugin --save-dev
```

二、前端页面设计

1. 用户界面设计

1.1 视觉层次

```css
/* 颜色系统 */
:root {
  --primary-color: #1890ff;       // 主色调
  --success-color: #52c41a;       // 成功状态
  --warning-color: #faad14;       // 警告状态
  --error-color: #f5222d;         // 错误状态
  --heading-color: #2c3e50;       // 标题文字
  --text-color: #333333;          // 正文文字
  --disabled-color: #909399;      // 禁用状态
}
```

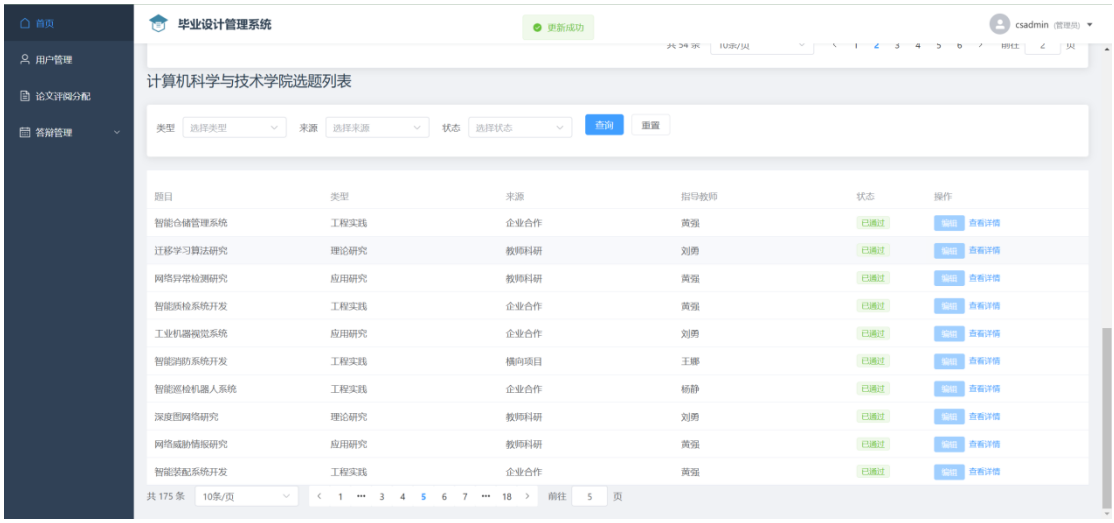
```css
/* 字体系统 */
.title {
  font-size: 18px;
  font-weight: 600;
}

.text-regular {
  font-size: 14px;
}

.text-small {
  font-size: 12px;
}
```

论文题目	申请学生	所属院系	审批状态	操作
test1	赵磊	计算机科学与技术学院	审核通过	已处理或教师未审核 查看详情
智能招聘系统开发	李静	计算机科学与技术学院	审核通过	已处理或教师未审核 查看详情
asfdfd	李静	计算机科学与技术学院	系主任已拒绝	已处理或教师未审核 查看详情
基于深度学习的图像识别系统设计与实现	杨敏	计算机科学与技术学院	审核通过	已处理或教师未审核 查看详情
asfdfd	杨敏	计算机科学与技术学院	教师已拒绝	已处理或教师未审核 查看详情
基于深度学习的图像识别系统设计与实现	杨敏	计算机科学与技术学院	系主任已拒绝	已处理或教师未审核 查看详情
asfdfd	杨敏	计算机科学与技术学院	系主任已拒绝	已处理或教师未审核 查看详情
基于深度学习的图像识别系统设计与实现	杨敏	计算机科学与技术学院	系主任已拒绝	已处理或教师未审核 查看详情
5G通信中的人工智能应用研究	周强	计算机科学与技术学院	审核通过	已处理或教师未审核 查看详情
基于深度学习的图像识别系统设计	李同学	计算机科学与技术学院	系主任已拒绝	已处理或教师未审核 查看详情

共 54 条 10条/页 1 2 3 4 5 6 前往 1 页

1.2 交互反馈

```vue
// 操作反馈
const handleOperation = async () => {
  try {
    ElMessage.success('操作成功')
  } catch (error) {
    ElMessage.error('操作失败')
  }
}

// 加载状态
const loading = ref(false)
<el-button :loading="loading">提交</el-button>
```



2. 页面布局设计

2.1 整体布局

```
<!-- Layout/index.vue -->
<template>
  <div class="app-wrapper">
    <!-- 顶部导航 -->
    <Header class="header" />

    <!-- 侧边栏 -->
    <Sidebar class="sidebar" />

    <!-- 主内容区 -->
    <div class="main-content">
      <router-view />
    </div>
  </div>
</template>
```

3. 交互设计

3.1 导航交互

```vue
<!-- Sidebar.vue -->
<template>
  <el-menu
    :default-active="activeMenu"
    :collapse="isCollapse"
    @select="handleSelect"
  >
    <!-- 动态菜单项 -->
    <template v-for="menu in menuItems">
      <el-menu-item
        v-if="!menu.children"
        :key="menu.path"
        :index="menu.path"
      >
        <el-icon><component :is="menu.icon" /></el-icon>
        <span>{{ menu.title }}</span>
      </el-menu-item>

      <el-sub-menu
        v-else
        :key="menu.path"
        :index="menu.path"
      >
        <!-- 子菜单项 -->
      </el-sub-menu>
    </template>
  </el-menu>
</template>
```

3.2 表单交互

```vue
<!-- StudentView.vue -->
<template>
  <div class="upload-section">
    <!-- 文件上传 -->
    <el-upload
      :before-upload="beforeUpload"
      :on-success="handleSuccess"
      :on-error="handleError"
    >
      <el-button type="primary">
        上传文件
      </el-button>
    </el-upload>

    <!-- 进度展示 -->
    <el-steps :active="currentStep">
      <el-step
        v-for="stage in stages"
        :key="stage.id"
        :title="stage.name"
        :description="stage.status"
      />
    </el-steps>
  </div>
</template>
```

4. 响应式设计

4.1 断点设计

```scss
// styles/variables.scss
$breakpoints: (
  'xs': 480px,
  'sm': 768px,
  'md': 992px,
  'lg': 1200px,
  'xl': 1920px
);

@mixin respond-to($breakpoint) {
  @media screen and (max-width: map-get($breakpoints, $breakpoint)) {
    @content;
  }
}
```

4.2 响应式布局

```vue
<template>
  <el-row :gutter="20">
    <!-- 响应式列 -->
    <el-col
      :xs="24"
      :sm="12"
      :md="8"
      :lg="6"
    >
      <el-card>
        <!-- 内容 -->
      </el-card>
    </el-col>
  </el-row>
</template>
```

三、后端对接数据库实现

1. 接口设计

1.1 RESTful API 规范

```javascript
// 路由设计示例
const router = express.Router()

// 资源命名规范
router.get('/users')                    // 获取用户列表
router.get('/users/:id')                // 获取单个用户
router.post('/users')                   // 创建用户
router.put('/users/:id')                // 更新用户
router.delete('/users/:id')             // 删除用户

// 查询参数规范
// GET /api/thesis?status=reviewing&page=1&limit=10
```

1.2 统一响应格式

```ts
// utils/response.ts
export interface ApiResponse<T> {
  code: number
  message: string
  data?: T
}

export const success = <T>(data: T, message = '操作成功'): ApiResponse<T> => ({
  code: 200,
  message,
  data
})

export const error = (message: string, code = 500): ApiResponse<null> => ({
  code,
  message
})
```

2. 数据访问层设计

2.1 数据库连接池优化

```ts
// utils/db.ts
import mysql from 'mysql2/promise'

const pool = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0,
  // 添加连接池配置
  enableKeepAlive: true,
  keepAliveInitialDelay: 0
})

// 数据库操作封装
export const query = async <T>(sql: string, params?: any[]): Promise<T> => {
  const connection = await pool.getConnection()
  try {
    const [rows] = await connection.query(sql, params)
    return rows as T
  } finally {
    connection.release()
  }
}
```

2.2 DAO 层实现

```typescript
// models/thesis.ts
export interface Thesis {
  id: number
  title: string
  student_id: string
  status: string
  created_at: Date
}

export class ThesisDAO {
  // 创建论文
  static async create(thesis: Partial<Thesis>): Promise<number> {
    const sql = 'INSERT INTO thesis SET ?'
    const result = await query<any>(sql, [thesis])
    return result.insertId
  }

  // 查询论文
  static async findById(id: number): Promise<Thesis | null> {
    const sql = 'SELECT * FROM thesis WHERE id = ?'
    const [thesis] = await query<Thesis[]>(sql, [id])
    return thesis || null
  }

  // 带事务的更新操作
  static async updateWithTransaction(id: number, data: Partial<Thesis>): Promise<boolean> {
    const connection = await pool.getConnection()
    try {
      await connection.beginTransaction()

      // 更新论文
      await connection.query('UPDATE thesis SET ? WHERE id = ?', [data, id])

      // 记录日志
      await connection.query(
        'INSERT INTO system_logs (user_id, action, details) VALUES (?, ?, ?)',
        [data.student_id, 'thesis.update', JSON.stringify(data)]
      )

      await connection.commit()
      return true
    } catch (error) {
      await connection.rollback()
      throw error
    } finally {
      connection.release()
    }
  }
}
```

3. 业务逻辑层设计

3.1 Service 层实现

```typescript
// services/thesis.service.ts
export class ThesisService {
  // 提交论文
  static async submitThesis(data: {
    student_id: string
    title: string
    file_url: string
  }) {
    // 参数验证
    if (!data.title || !data.file_url) {
      throw new ValidationError('标题和文件不能为空')
    }

    // 业务逻辑处理
    const thesis = await ThesisDAO.create({
      ...data,
      status: 'pending'
    })

    // 发送通知
    await NotificationService.send({
      user_id: data.student_id,
      type: 'thesis_submitted',
      content: `论文《${data.title}》已提交成功`
    })

    return thesis
  }
```

4. 异常处理机制

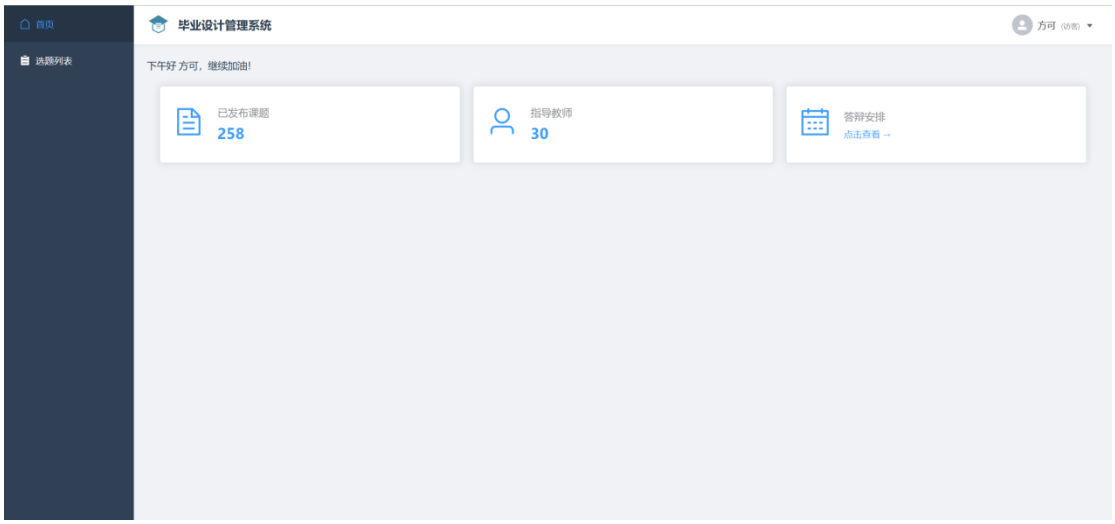
4.1 自定义错误类型

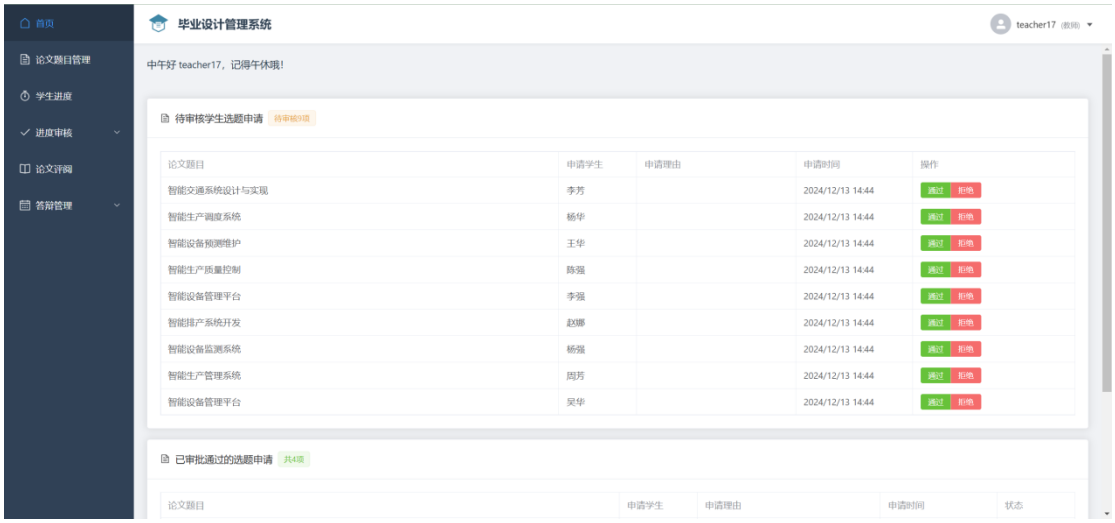
```typescript
// utils/errors.ts
export class BaseError extends Error {
  constructor(
    public message: string,
    public code: number,
    public status: number = 500
  ) {
    super(message)
    this.name = this.constructor.name
    Error.captureStackTrace(this, this.constructor
  }
}

export class ValidationError extends BaseError {
  constructor(message: string) {
    super(message, 400, 400)
  }
}

export class NotFoundError extends BaseError {
  constructor(message: string) {
    super(message, 404, 404)
  }
}

export class BusinessError extends BaseError {
  constructor(message: string) {
    super(message, 400, 400)
  }
}
```

4.2 全局错误处理中间件

```ts
// middleware/error.ts
export const errorHandler = (
  err: Error,
  req: Request,
  res: Response,
  next: NextFunction
) => {
  // 记录错误日志
  console.error('Error:', {
    name: err.name,
    message: err.message,
    stack: err.stack,
    path: req.path,
    method: req.method,
    body: req.body
  })
```

一些项目页面图片示例：





第五部分——总结与反思

一、目前存在的问题

1. 系统功能方面

缺乏数据导出功能，不支持批量数据处理
文件上传功能较为简单，缺乏预览和在线编辑功能
消息通知系统不够完善，缺乏实时推送功能
缺少移动端适配，用户体验有待提升

2. 技术实现方面

部分代码重复度较高，需要进一步抽象和复用
错误处理机制不够完善，日志记录不够详细
缺乏完整的单元测试和集成测试
数据库查询优化空间较大，部分查询效率较低

3. 安全性方面

密码策略相对简单，缺乏复杂度要求
敏感数据加密方式需要加强
缺乏完整的操作审计日志
XSS 和 SQL 注入防护需要加强

二、未来改进的方向

1. 功能扩展

增加数据分析和可视化功能
开发移动端应用或响应式界面
完善文件管理系统，支持在线预览和编辑
引入实时通知系统，支持即时消息推送

2. 技术优化

重构代码，提高代码复用率
优化数据库设计和查询性能
完善测试体系，提高代码覆盖率
引入缓存机制，提升系统响应速度

3. 安全加强

实现更严格的密码策略
加强数据加密和传输安全
完善权限控制和访问审计
增加防攻击措施

三、项目经验总结

1. 技术收获

掌握了前后端分离开发模式
深入理解了 Node.js 和 React 的开发特点
提升了数据库设计和优化能力
积累了项目架构和性能优化经验

2. 项目管理经验

学会了合理规划开发周期
掌握了版本控制和代码管理
理解了模块化设计的重要性
提高了问题分析和解决能力

3. 个人成长总结

提高了独立解决问题的能力
增强了技术文档阅读和编写能力
培养了持续学习的习惯
积累了实际项目经验