

深圳大学

实验报告

课程名称: 数据库系统

实验序号: 实验 4

实验名称: 数据库设计

学 号: 2022155028

姓名: 黄亮铭

实验完成日期: 2024 年 12 月 15 日

目录

1 实验目的.....	3
2 实验要求.....	3
3 实验设备.....	3
4 实验内容.....	3
5 实验过程.....	4
6 问题分析（碰到什么问题，如何解决）	21
7 实验心得.....	21
8 诚信承诺.....	22

1 实验目的

- 1) 掌握数据库设计方法;
- 2) 了解概念模型、逻辑数据模型和物理数据模型之间的关系和不同;
- 3) 掌握使用高级语言访问、操作数据库,加深对前后台数据交互的理解。

2 实验要求

- 1) 确定选题,并进行需求分析,用高级语言实现一个小型数据库应用系统;
- 2) 完成一个小型系统的数据库设计,绘制 E-R 图;
- 3) 将E-R 图转成逻辑数据模型和物理数据模型,导出 sql 脚本,创建数据库,E-R 图至少包括 8 个实体和 7 个联系; 设计至少 1 个视图、1 个索引(非主键、非外键索引)、1 个触发器、1 个 存储过程或存储函数(视图、索引、触发器、存储过程或存储函数可以直接在 DBMS 中创建,不一定要在 pdm 图的时候创建);
- 4) 需要考虑关系完整性约束:主键约束、外键约束、空值约束;
- 5) 数据库使用 MySQL;
- 6) 编程语言不限;
- 7) 考察重点是数据库设计、前后台数据交互。图形界面是必须的,以简单明了为上。

3 实验设备

计算机、数据库管理系统如 php, mysql 等软件。

4 实验内容

- 1) 按要求完成数据库实验指导书,完成一个小型系统的设计与实现,除了指导书的明确要求,其他可以按照自己的思路拓展功能。(数据库设计 30 分,CDM 绘制及转换 20 分、可执行对象 20 分,前端 20 分,文档 10 分)
- 2) 附上相关关键问题的解决方案,以及运行结果,运行屏幕截图。
- 3) (可选)录一小段视频,3 分钟左右,对自己的亮点进行解说展示。

5 实验过程

本次实验选题我选择校园活动征招平台。

5.1 数据库设计

5.1.1 确定实体和联系

根据实验文档提供的需求，可以确定以下主要实体。

- **用户 (user):** 包括普通用户和管理员。
- **活动 (activity)**
- **申请记录 (application)**
- **审核记录 (audit)**
- **学生 (student)**

此外，还需要一些辅助的实体如下所示。

- **用户类型 (user_type):** 确定用户的类型，用于区分用户的权限。
- **状态 (status):** 确定活动、申请记录和审核记录的当前状态。对于活动来说，状态存在未开始、进行中和已结束等状态；对于申请记录来说，存在待审核、已通过和已拒绝等状态；对于审核记录来说，存在待审核和已审核等状态。
- **日志 (log):** 记录用户对数据库的表的操作，包括普通用户提交申请和管理员发布活动、审核申请等操作。
- **日志类型 (log_type):** 确定日志的类型，用于明确日志记录的操作是否存在问题，以及存在的问题的严重性。

上述实体之间存在如下的联系（包括但不限于）。

- **学生注册普通用户账号:** 一对多的关系。一个学生只能拥有一个普通用户账号。
- **普通用户提交申请记录, 申请活动:** 一对多的关系。一个用户可以提交多条申请记录，用于申请不同的活动。
- **管理员发布(编辑)活动:** 一对多的关系。一个管理员可以发布(编辑)多项活动。
- **管理员审核申请记录:** 多对多的关系。一个管理员可以审核多条申请记录，一条申请记录也可以被多个管理员进行审核。但是该系统为小型系统，暂定为一个管理员，因此也可以看成是一对多的关系，即一个管理员可以发布(编辑)多项活动。
- **活动与申请的关联:** 一对多的关系。一项活动对应多条申请记录。
- **日志与申请的关联:** 一对一的关系。一条申请记录会对应一条日志。
- **日志与活动的关联:** 一对一的关系。一项活动会对应一条日志。
- **日志与审核的关联:** 一对一的关系。一条审核记录会对应一条日志。
- **日志与用户的关联:** 一对一的关系。一项用户相关的操作会对应一条日志。

5.1.2 确定实体的字段

在确定了实体和实体之间的联系之后，我们需要确定每个实体的字段和属性。接下来我将以实体的英文名降序展示每个实体的字段以及属性设置。

对于活动表, 我们首先需要考虑活动标题、活动时间(开始、结束)、活动人数、活动地点、活动内容、活动负责人以及作为主键的活动编号, 因此我设置了如下字段(图1)。

Entity Properties - activity (activity)

General Attributes Identifiers Notes Rules

	Name	Code	Data Type	Length	Precision	N	F	C
1	activity_no	activity_no	Integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2	activity_start_time	activity_start_time	Date & Time			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3	activity_location	activity_location	Variable characters (64)	64		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
4	activity_total_spots	activity_total_spots	Integer			<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
5	activity_content	activity_content	Text			<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
6	activity_title	activity_title	Variable characters (64)	64		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
7	activity_spots	activity_spots	Integer			<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
8	activity_end_time	activity_end_time	Date & Time			<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
→	activity_leader	activity_leader	Variable characters (64)	64		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

More >>

确定

取消

图 1 活动表

对于申请表,我们需要考虑申请人的基本信息(如姓名、学号等)、申请的活动编号、申请时间和作为主键的申请编号等。此外,我还考虑到在 CDM=》LDM=》LDM=》Database 的过程中会自动生成具有外键限制的字段。因此我在申请表设置了如下字段(图2)。

Entity Properties - application (application)

General | Attributes | Identifiers | Notes | Rules

	Name	Code	Data Type	Length	Precision	N	F	C
1	application_no	application_no	Integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2	application_time	application_time	Date & Time			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

More >>

确定 取消

图 2 申请表

对于审核表，我们需要考虑的内容和申请类似。本质上审核是管理员对申请表做的一个动作，但是为了方便后续的日志处理，我将其拆分成一个表。因此审核表的设置和申请表类似。

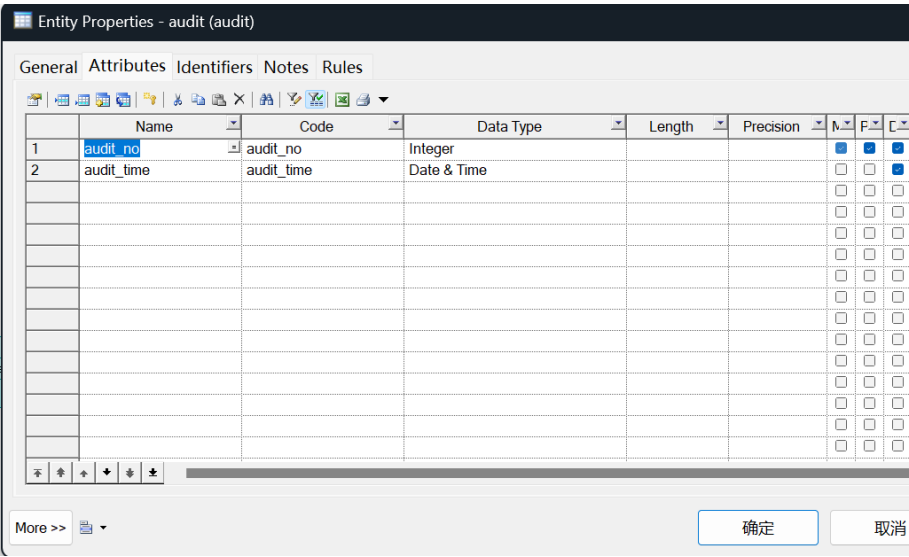


图 3 申请表

对于学生表，我们需要考虑的是学生的信息，如学号、姓名等。学生表的字段设置情况如图 4 所示。

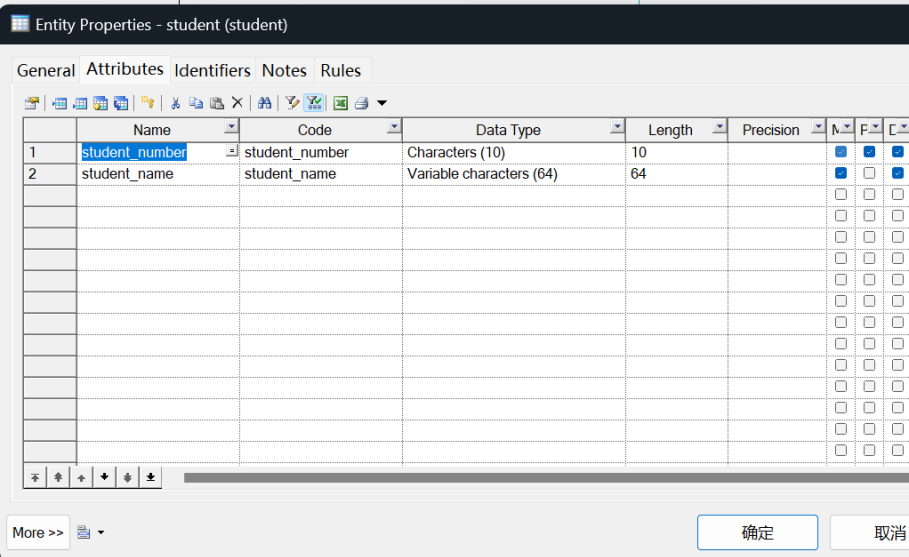


图 4 申请表

对于用户表，我们需要考虑的是用户编号、账号、密码、账号类型、学生信息等字段。此外，我还考虑到在 CDM=》LDM=》LDM=》Database 的过程中会自动生成具有外键限制的字段。因此，我在用户表只设置了如下的字段（图 5）。

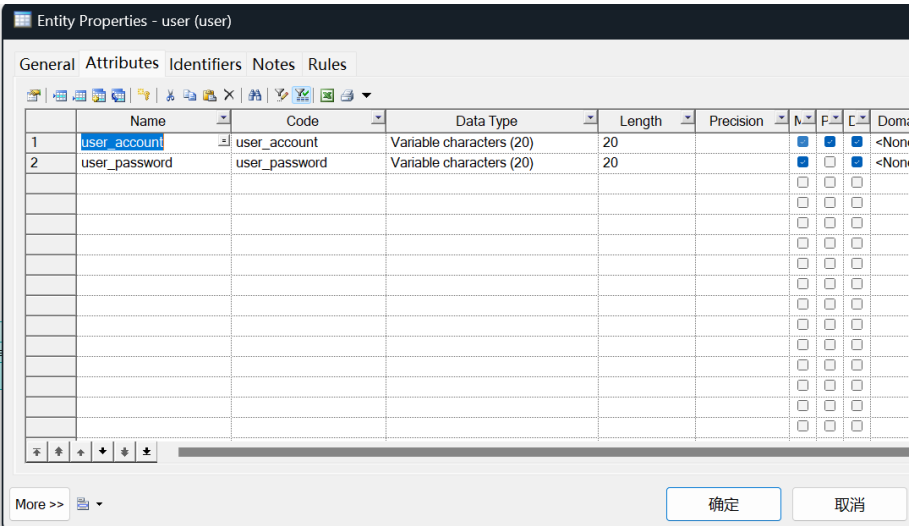


图 5 用户表

除了上述主要的实体的字段以外，我们还有一些辅助实体的字段。如用户类型表（图 6）中存储用户类型编号和用户类型名称（管理员、普通用户）；状态类型表（图 7）中存储状态编号和状态名称（未开始、进行中、已结束、待审核、已审核、已通过、已拒绝）；日志表（图 8）包含日志编号、日志内容；日志类型表（图 9）包含日志类型编号和日志类型名称（info、error、panic）。

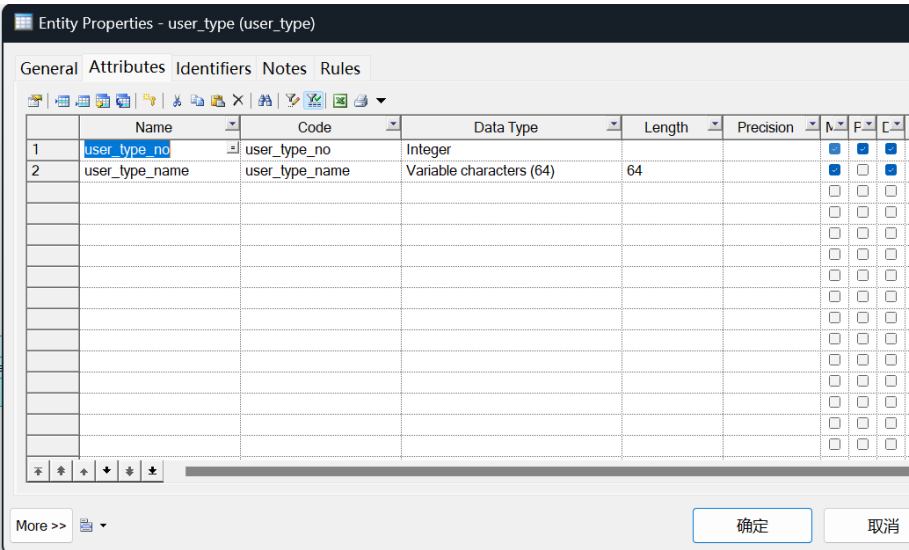


图 6 用户类型表

Entity Properties - status (status)

General Attributes Identifiers Notes Rules

	Name	Code	Data Type	Length	Precision	N	F	C
1	status_no	status_no	Integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2	status_name	status_name	Variable characters (64)	64		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
						<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

[Icons]

More >> [Icon] | [Button: 确定] | [Button: 取消]

[illegible][illegible]

5.1.3CDM 绘制

在 5.1.1 中我们确定了实体和实体之间的类型，在 5.1.2 中我们确定了每个实体的字段和字段属性。我们可以根据这些内容绘制出 E-R 图如图 10 所示。

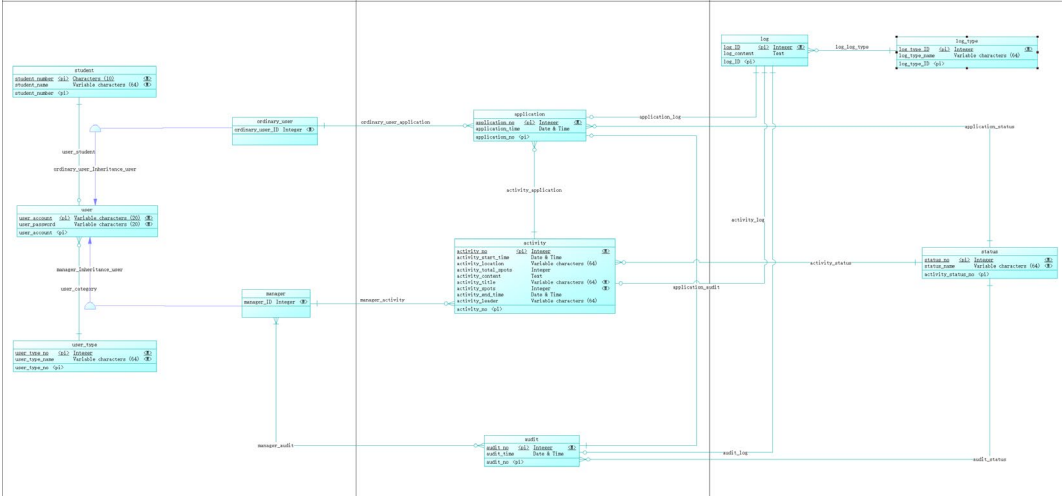


图 10E-R 图

5.1.4CDM 转换 LDM

CDM 转换为逻辑数据模型的步骤如下所示。

- 1) 在 Tools 菜单中单击 **Generate Logical Data Model** 命令，打开逻辑数据模型设置窗口。
- 2) 在设置窗口修改信息（修改命名等），进行个性化配置，一般来说保持默认即可。
- 3) 将生成的 LDM 修改成我们习惯的 Entity/Relationship 类型。

经过上述步骤后，我们得到的 LDM 模型如图 11 所示。

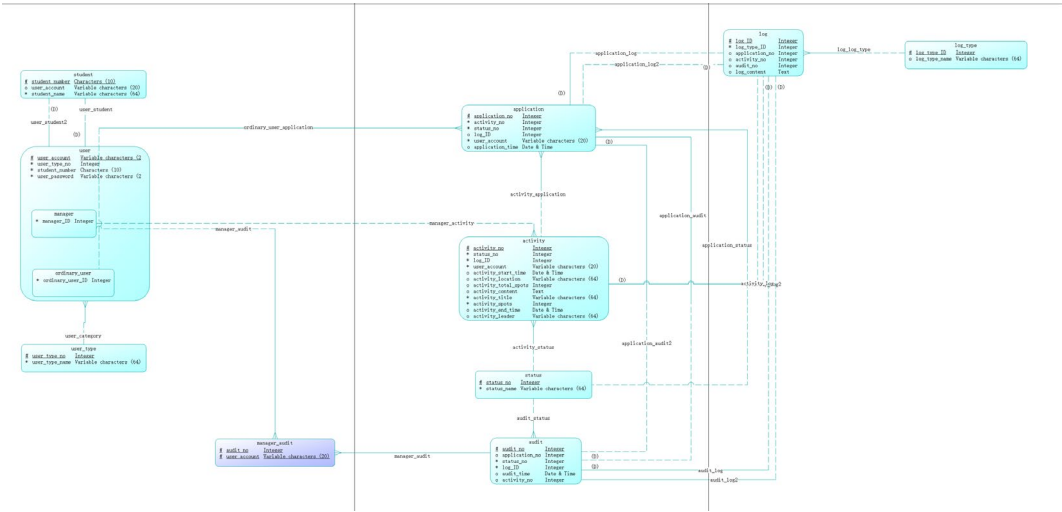


图 11LDM 图

5.1.5LDM 转换 PDM 再导出 SQL 脚本

在将 LDM 转换为 PDM 之前，我们需要在 LDM 中删除一些字段，因为 CDM 转换 LDM 的时候，一对一关系会在两个实体中都生成一个有外键限制的字段，在后续生成的数据库会造成无法删除数据库记录的错误。此外，我们还需要处理 CDM 转换 LDM 过程中将 ID、ACCOUNT 字段设置为联合主键的问题，即在 LDM 中管理员和普通用户与其他主体的联系中删除 ID 字段，如图 12 所示。

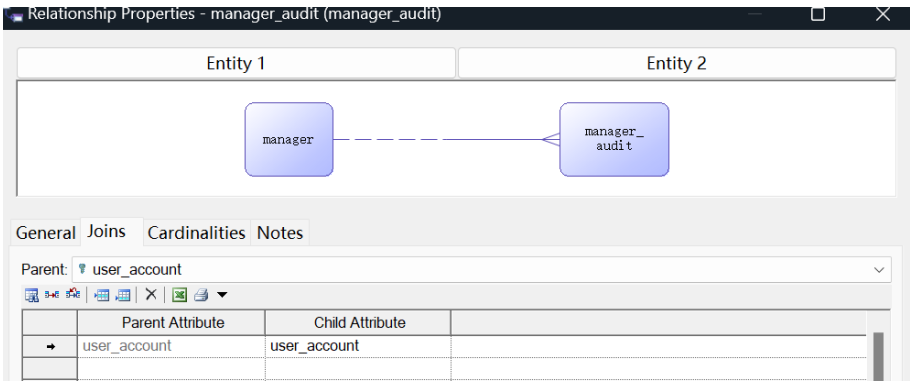


图 12 删除字段

在解决上述问题之后，我们将进行 LDM 转换为 PDM 的步骤。

- 1) 在 Tools 菜单中单击 Generate Physical Data Model 命令，打开物理数据模型设置窗口。
- 2) 在设置窗口修改信息（修改命名、数据库等），进行个性化配置，除了修改信息中提到的以外一般可以保持默认。

经过上述步骤后，我们得到的 PDM 模型如图 13 所示。

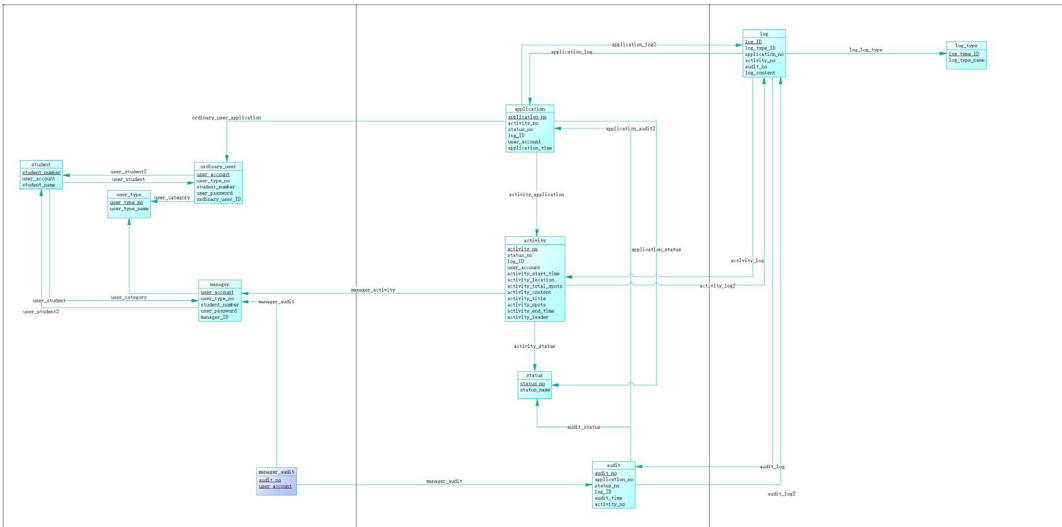


图 13PDM 图

完成上述步骤之后，我们根据实验手册的步骤将 PDM 导出为 SQL 脚本，部分内容如图 14 所示，完整脚本放在代码压缩包中。

```

1  /*=====*/
2  /* DBMS name:      MySQL 5.0 */
3  /* Created on:     2024/12/13 19:18:30 */
4  /*=====*/
5
6
7  drop table if exists activity;
8
9  drop table if exists application;
10
11 drop table if exists audit;
12
13 drop table if exists log;
14
15 drop table if exists log_type;
16
17 drop table if exists manager;
18
19 drop table if exists manager_audit;
20
21 drop table if exists ordinary_user;
22
23 drop table if exists status;
24
25 drop table if exists student;
26
27 drop table if exists user_type;
28
29 /*=====*/
30 /* Table: activity */
31 /*=====*/
32 create table activity
33 (
34     activity_no          int not null auto_increment unique,
35     status_no            int not null,
36     log_ID               int,
37     user_account          varchar(20) not null,
38     activity_start_time   datetime,
39     activity_location     varchar(64),
40     activity_total_spots  int,
41     activity_content      text,
42     activity_title        varchar(64) not null,
43     activity_spots        int not null,
44     activity_end_time     datetime,
45     activity_leader       varchar(64),
46     primary key (activity_no)
47 );
48

```

图 14SQL 脚本（部分内容）

5.1.6 添加可执行对象

该部分我没有在 PDM 中添加，而是在 SQL 脚本中手动添加。

5.1.6.1 存储过程

在管理员通过普通用户的申请的时候，会将活动报名成功人数加 1。我设计了一个存储过程，将这个过程封装在函数中，方便调用。存储过程如图 15 所示。

```

/* 存储过程：活动人数+1 */
DELIMITER $$

CREATE PROCEDURE updateActivitySpots(IN activityNo INT)
BEGIN
    UPDATE activity
    SET activity_spots = activity_spots + 1
    WHERE activity_no = activityNo;
END$$

DELIMITER ;

```

图 15 存储过程

5.1.6.2 触发器

我一共设置了 5 个触发器，每个触发器的作用分别为：

- 1) 在插入活动记录前根据时间判断活动的状态。

```

/* 触发器：在插入活动记录前根据时间判断活动的状态 */
DELIMITER $$

CREATE TRIGGER before_activity_insert
BEFORE INSERT ON activity
FOR EACH ROW
BEGIN
    DECLARE currentTime DATETIME;
    SET currentTime = NOW();

    IF currentTime < NEW.activity_start_time THEN
        SET NEW.status_no = 1;
    ELSEIF currentTime >= NEW.activity_start_time AND currentTime <= NEW.activity_end_time THEN
        SET NEW.status_no = 2;
    ELSE
        SET NEW.status_no = 3;
    END IF;
END$$

DELIMITER ;

```

图 16 触发器 1

- 2) 在更新活动记录前根据时间判断活动的状态。

```

/* 触发器：在更新活动记录前根据时间判断活动的状态 */
DELIMITER $$

CREATE TRIGGER after_activity_update
BEFORE UPDATE ON activity
FOR EACH ROW
BEGIN
    DECLARE currentTime DATETIME;
    SET currentTime = NOW();

    IF currentTime < NEW.activity_start_time THEN
        SET NEW.status_no = 1;
    ELSEIF currentTime >= NEW.activity_start_time AND currentTime <= NEW.activity_end_time THEN
        SET NEW.status_no = 2;
    ELSE
        SET NEW.status_no = 3;
    END IF;
END$$

DELIMITER ;

```

图 17 触发器 2

- 3) 在插入申请记录之前设置申请时间，申请状态，日志。

```

/* 触发器：在插入申请记录之前设置申请时间，申请状态，日志 */
DELIMITER $$

CREATE TRIGGER before_application_insert
BEFORE INSERT ON application
FOR EACH ROW
BEGIN
    SET NEW.status_no = 6;
    SET NEW.application_time = NOW();
    SET NEW.log_ID = NULL;
END$$

DELIMITER ;

```

图 18 触发器 3

- 4) 在插入申请记录后会同时生成一条审核记录。

```
/* 触发器：在插入申请记录后会同时生成一条审核记录 */
DELIMITER $$

CREATE TRIGGER after_application_insert
AFTER INSERT ON application
FOR EACH ROW
BEGIN
    INSERT INTO audit (application_no, status_no, log_ID, audit_time, activity_no)
    VALUES (NEW.application_no, 6, NULL, NULL, NEW.activity_no);
END$$

DELIMITER ;
```

图 19 触发器 4

- 5) 修改申请记录的状态，同时根据修改后的申请记录状态修改活动报名人数和审核状态。

```
/* 触发器：修改申请记录的状态，同时根据修改后的申请记录状态修改活动报名人数和审核状态 */
DELIMITER $$

CREATE TRIGGER after_application_update
AFTER UPDATE ON application
FOR EACH ROW
BEGIN
    IF OLD.status_no = 6 AND NEW.status_no = 4 THEN
        CALL updateActivitySpots(NEW.activity_no);
    END IF;
END$$

DELIMITER ;
```

图 20 触发器 5

5.1.6.3 视图

我一共创建了两个视图，第一个视图是用于授权普通用户以活动的开始时间和结束时间降序浏览活动；第二个是视图用于管理员以申请时间升序审核普通用户的申请。

```
/* 视图：在Activity表和status表中建立 */
CREATE VIEW activity_with_status AS
SELECT
    a.activity_no,
    a.activity_start_time,
    a.activity_location,
    a.activity_total_spots,
    a.activity_content,
    a.activity_title,
    a.activity_spots,
    a.activity_end_time,
    a.activity_leader,
    s.status_name
FROM
    activity a
LEFT JOIN
    status s ON a.status_no = s.status_no;
```

图 21 视图 1 浏览活动

```

/* 视图：包含前端审核记录需要的各种信息 */
CREATE VIEW audit_record_view AS
SELECT
    s.student_name AS student_name,
    s.student_number AS student_number,
    app.user_account AS user_account,
    app.application_time AS application_time,
    ac.activity_title AS activity_title,
    st1.status_name AS application_status_name,
    st2.status_name AS audit_status_name,
    au.activity_no AS activity_no,
    au.application_no AS application_no,
    au.audit_no AS audit_no
FROM
    audit au
JOIN application app ON au.application_no = app.application_no
JOIN activity ac ON app.activity_no = ac.activity_no
JOIN ordinary_user o_user ON app.user_account = o_user.user_account
JOIN status st1 ON app.status_no = st1.status_no
JOIN status st2 ON au.status_no = st2.status_no
JOIN student s ON o_user.student_number = s.student_number;

```

图 22 视图 2 审核申请

5.1.6.4 索引

我在活动表的活动开始时间和活动结束时间上创建了索引，加速视图的查询。

```

/* 索引：降序排序开始时间和结束时间 */
CREATE INDEX idx_activity_start_end_time ON activity (activity_start_time DESC, activity_end_time DESC);

```

图 23 索引

5.2 前端设计

前端是 MVC 架构中的 VIEW 层（视图层），我使用 uniapp 实现，开发环境为官方工具 HBuilderX。

项目的总体框架如图 24 所示。

框架中有一些比较重要的文件（夹）的介绍如下：

- 文件 main.js: Vue 初始化入口文件。
- 文件 App.vue: 应用配置，用来配置 App 全局样式以及监听。
- 文件 pages.json: 配置页面路由、导航条、选项卡等页面类信息。
- 文件 manifest.json: 配置应用名称、appid、logo、版本等打包信息。
- 文件夹 pages: 业务页面文件存放的目录。
- 文件夹 static: 存放应用引用的本地静态资源（如图片、视频等）的目录。

我们重点关注文件夹 pages 和文件 pages.json。我们完成的所有页面内容均保存在文件夹 pages 中，而页面路由信息、导航栏信息还有选项卡等内容军保存在文件 pages.json 中。文件夹 pages 包含的所有页面均需要在文件 pages.json 里面的页面路由注册才能被正常地使用。

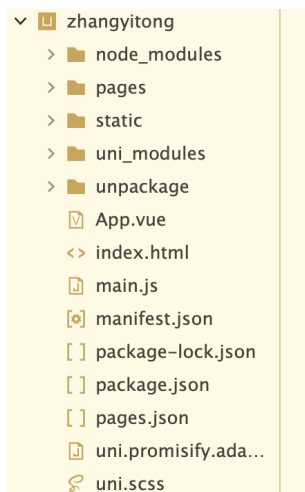


图 24 前端框架

该项目前端一共有 9 个页面，分别为普通用户登录页面、普通用户注册页面、管理员登录页面、普通用户的活动列表页面、普通用户的活动详情页面、普通用户的关于我页面、管理员的活动列表页面、管理员的活动详情页面、管理员的审核页面。

接下来我将以核心之一的管理员的活动列表页面为例，讲述前端是如何实现的。其他页面的实现方法与该页面的实现方法类似。

我将从 script 即脚本部分和 template 即模版部分介绍该页面的实现思路，而 style 即样式部分主要是页面美化，与核心逻辑无关，因此直接忽略。

5.2.1 脚本

脚本部分负责处理页面的数据和交互逻辑。

- **数据定义：**在 `data()` 函数中定义了页面所需的数据属性，如 `events`（活动列表）、`showModal`（控制浮窗显示）、`newEvent`（新活动对象）。

```
data() {  
  return {  
    events: [],  
    showModal: false,  
    newEvent: [],  
  };  
},
```

图 25

- **生命周期钩子：**`onShow()` 方法在页面显示时调用，用于获取活动列表。

```
onShow() {  
  this.fetchEvents();  
},
```

图 26

➤ 方法定义:

1) fetchEvents(): 向后端请求活动数据并更新 events 数组。

```
fetchEvents() {
  uni.request({
    url: 'http://127.0.0.1:8080/event/activity/query',
    method: 'GET',
    success: (res) => {
      if (res.statusCode === 200) {
        this.events = [];
        res.data.activity.forEach(activity => {
          this.events.push({
            no: activity.no,
            title: activity.title,
            leader: activity.leader,
            spots: activity.spots,
            totalSpots: activity.totalSpots,
            location: activity.location,
            startTime: String(activity.startTime).split(/[+]/)[0] +
              " " + String(activity.startTime).split(/[+]/)[1],
            endTime: String(activity.endTime).split(/[+]/)[0] + " " +
              String(activity.endTime).split(/[+]/)[1],
            content: activity.content,
            statusName: activity.statusName,
          });
          console.log(activity.startTime)
        });
      } else {
        console.error('Failed to fetch events:', res);
      }
    },
    fail: () => {
      console.error('Request failed');
    }
  });
},
```

图 27

2) showAddEventModal(): 显示新增活动浮窗，初始化新活动对象 newEvent。

```
showAddEventModal() {
  const currentDate = new Date();
  const currentYear = currentDate.getFullYear();
  const currentMonth = currentDate.getMonth() + 1; // 月份从0开始计数
  const currentDay = currentDate.getDate();
  const currentTime = currentDate.getHours() + ':' + currentDate.getMinutes();

  this.newEvent = {
    title: '',
    leader: '',
    spots: '0',
    totalSpots: '',
    location: '',
    startDate: `${currentYear}-${currentMonth < 10 ? '0' + currentMonth : currentMonth}-`,
    startTime: currentTime,
    endDate: `${currentYear}-${currentMonth < 10 ? '0' + currentMonth : currentMonth}-${`,
    endTime: currentTime,
    content: '',
  };
  this.showModal = true;
},
```

图 28

3) closeAddEventModal(): 关闭浮窗。

```
closeAddEventModal() {
  this.showModal = false;
},
```

图 29

- 4) addEvent(): 提交新活动信息到后端, 成功后关闭浮窗并刷新活动列表。

```
addEvent() {
  // 向后端发送新增活动的消息
  console.log(uni.getStorageSync("userAccount"));
  uni.request({
    url: 'http://127.0.0.1:8080/event/activity/add', // 替换为你的后端API地址
    method: 'POST',
    data: { ...
  },
  success: (res) => {
    if (res.statusCode === 200 && res.data.success === true) {
      uni.showToast({
        title: '活动添加成功',
        icon: 'success'
      });
      this.closeAddEventModal(); // 关闭浮窗
      // 刷新活动列表
      this.fetchEvents();
    } else {
      uni.showToast({
        title: '添加失败',
        icon: 'none'
      });
    }
  },
  fail: () => {
    uni.showToast({
      title: '请求失败',
      icon: 'none'
    });
  }
});
this.fetchEvents();
},
```

图 30

- 5) onStartDateChange() 、 onStartTimeChange() 、 onEndDateChange() 、 onEndTimeChange(): 处理日期和时间选择器的变化。

```
onStartDateChange(e) {
  this.newEvent.startDate = e.detail.value;
},
onStartTimeChange(e) {
  this.newEvent.startTime = e.detail.value;
},
onEndDateChange(e) {
  this.newEvent.endDate = e.detail.value;
},
onEndTimeChange(e) {
  this.newEvent.endTime = e.detail.value;
},
```

图 31

- 6) goToEventDetail(): 返回到活动详情页面, 传递活动对象作为参数。

```
goToEventDetail(event) {
  uni.navigateTo({
    url: '/pages/manager_activity_detail/manager_activity_detail?event=' + JSON
      .stringify(event)
  });
},
```

图 32

5.2.2 模板

模板部分定义了页面的结构和内容。

- **活动列表：**使用 v-for 指令遍历 events 数组，为每个活动创建一个列表项。
- **新增活动按钮：**一个按钮，点击时调用 showAddEventModal()方法。
- **浮窗组件：**用于收集新活动的信息。包含多个输入字段和一个提交按钮。
- **日期时间选择器：**使用 picker 组件允许用户选择日期和时间。

因为模板代码部分过于冗长，不方便将完整的代码进行截图展示，因此这里直接给出该页面的效果图。



图 33 管理员活动列表页面（左）管理员新增活动子页面（右）

5.3 后端设计

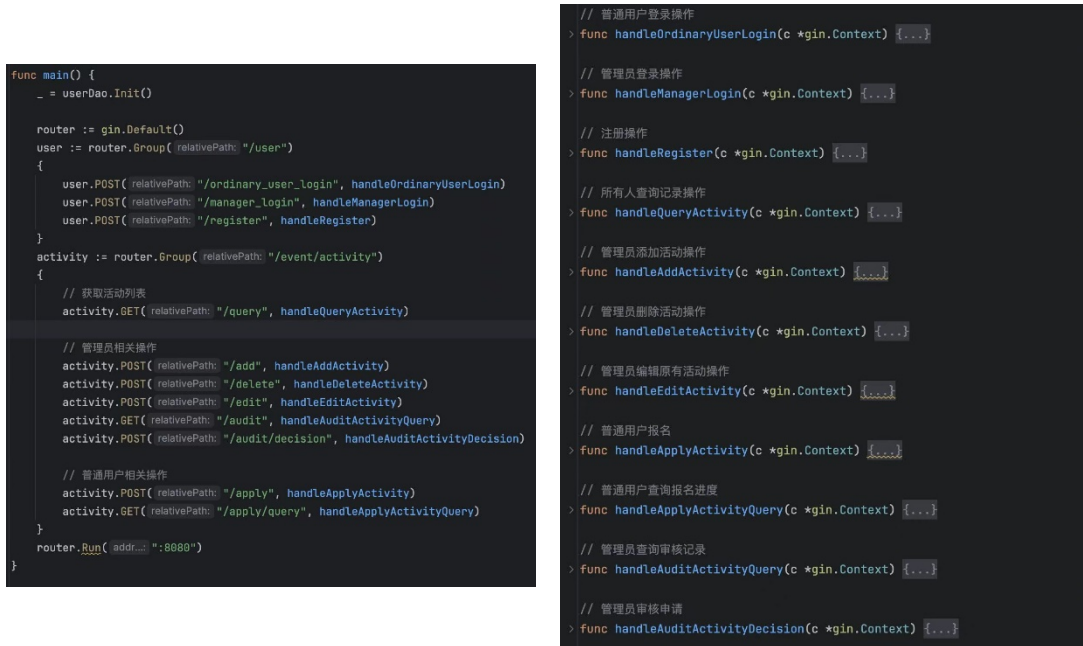
后端是 MVC 架构中的 Controller 层（控制器层）和 Model（模型层），前者的任务是实现消息的转发，后者的任务是实现对应的业务逻辑处理。后端我使用 Go 实现，开发环境为官方工具 Golang，使用的框架为 Gin、Gorm。

后端数据处理流程均为控制器层接收前端数据请求和数据提交，然后控制器层将数据转

发给模型层中对应的 API 接口，最后由该接口进行业务逻辑处理并和数据库进行交互。

5.3.1 Controller 层

首先我使用 Gin 框架的路由组功能接收前端发送的数据请求和数据提交，然后转发给不同的处理函数进行简单的处理，最后把数据请求和数据提交转发给 Model 层。



```
func main() {
    _ = userDao.Init()

    router := gin.Default()
    user := router.Group(relativePath: "/user")
    {
        user.POST(relativePath: "/ordinary_user_login", handleOrdinaryUserLogin)
        user.POST(relativePath: "/manager_login", handleManagerLogin)
        user.POST(relativePath: "/register", handleRegister)
    }
    activity := router.Group(relativePath: "/event/activity")
    {
        // 获取活动列表
        activity.GET(relativePath: "/query", handleQueryActivity)

        // 管理员相关操作
        activity.POST(relativePath: "/add", handleAddActivity)
        activity.POST(relativePath: "/delete", handleDeleteActivity)
        activity.POST(relativePath: "/edit", handleEditActivity)
        activity.GET(relativePath: "/audit", handleAuditActivityQuery)
        activity.POST(relativePath: "/audit/decision", handleAuditActivityDecision)

        // 普通用户相关操作
        activity.POST(relativePath: "/apply", handleApplyActivity)
        activity.GET(relativePath: "/apply/query", handleApplyActivityQuery)
    }
    router.Run(addr: ":8080")
}

// 普通用户登录操作
> func handleOrdinaryUserLogin(c *gin.Context) {...}

// 管理员登录操作
> func handleManagerLogin(c *gin.Context) {...}

// 注册操作
> func handleRegister(c *gin.Context) {...}

// 所有人查询记录操作
> func handleQueryActivity(c *gin.Context) {...}

// 管理员添加活动操作
> func handleAddActivity(c *gin.Context) {...}

// 管理员删除活动操作
> func handleDeleteActivity(c *gin.Context) {...}

// 管理员编辑原有活动操作
> func handleEditActivity(c *gin.Context) {...}

// 普通用户报名
> func handleApplyActivity(c *gin.Context) {...}

// 普通用户查询报名进度
> func handleApplyActivityQuery(c *gin.Context) {...}

// 管理员查询审核记录
> func handleAuditActivityQuery(c *gin.Context) {...}

// 管理员审核申请
> func handleAuditActivityDecision(c *gin.Context) {...}
```

图 34 路由组（左）和处理函数（右）

完整的代码放在代码压缩包中。这里只给出查询活动列表的代码。



```
// 所有人查询记录操作
func handleQueryActivity(c *gin.Context) { 1 usage
    respActivity, err := userDao.QueryActivity()
    if err != nil {
        c.JSON(http.StatusOK, gin.H{
            "message": err.Error(),
            "success": false,
        })
    } else {
        c.JSON(http.StatusOK, gin.H{
            "success": true,
            "activity": respActivity,
        })
    }
}
```

图 35 前端获取活动列表的请求的处理函数

5.3.2 Model 层

首先使用工厂模式创建一个 Dao 对象，然后对其进行初始化操作，用于后端连接数据库。然后实现不同的 API 接口进行对应的业务逻辑处理。

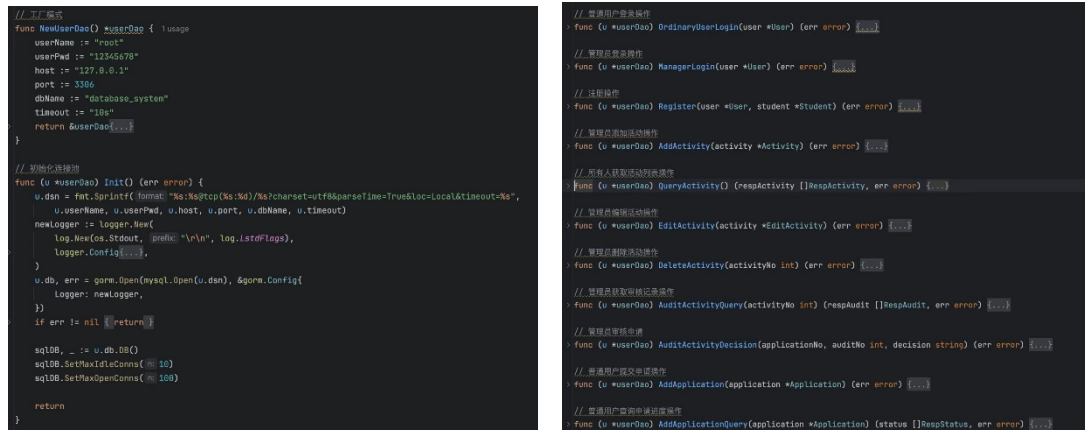


图 36 初始化（左）和 API 接口（右）

完整的代码放在代码压缩包中。这里只给出查询活动列表的代码。

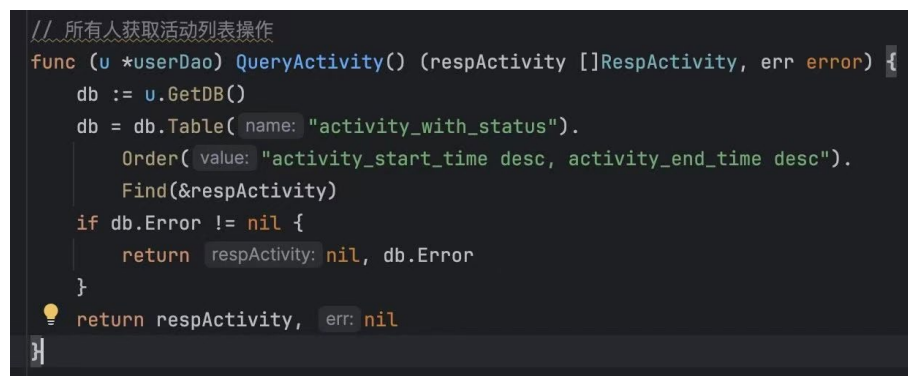


图 37 查询活动列表

6 补充内容

补充的内容为实现过期的活动系统自动设置为关闭状态。

实现的思路为：1) 首先创建一个存储过程，将活动的结束时间和当前时间进行对比，如果当前时间大于活动的结束时间，则将活动状态设置为已结束，否则不改变。2) 然后创建一个事件，每过一个小时就检查是否有活动过期(调用 1 中提到的存储过程)。3) 将 MySQL 的事件调度器设置为打开状态。

```

/* 补充内容 */
DELIMITER $$

CREATE PROCEDURE UpdateActivityStatus()
BEGIN
    DECLARE currentTime DATETIME;
    SET currentTime = NOW();
    UPDATE activity
    SET status_no = CASE
        WHEN activity_end_time < currentTime THEN 3
        ELSE status_no
    END;
END;
DELIMITER ;

DELIMITER $$

CREATE EVENT IF NOT EXISTS UpdateActivityStatusEvent
ON SCHEDULE EVERY 1 HOUR -- 每小时检查一次
DO CALL UpdateActivityStatus()$$

DELIMITER ;

SET GLOBAL event_scheduler = ON;

```

图 38 自动设置关闭状态

7 问题分析（碰到什么问题，如何解决）

- 1) 前端以 GET 请求发送数据时后端可以正确地接收到数据，但是前端以 POST 请求发送数据时后端无法正确接收到数据。

解决办法：默认的请求头的 content-type 为 application/json，需要将其修改为 'application/x-www-form-urlencoded' 后端才能正确接收。或者不修改 content-type，后端使用对应的结构体（需要对每个字段设置 json 别名）进行接收。

- 2) 在实现删除活动功能时发现数据库无法正确删除相关表的相关字段。

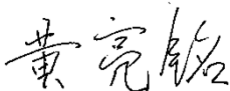
解决办法：查看数据库对应的表，发现数据库中申请表和审核表相互之间存在外键限制，将外键限制以及其中一个表的外键字段删除即可解决该问题。

8 实验心得

- 1) 本次实验我完成了小型系统的设计与实现的全部要求。
- 2) 通过本次实验，我熟悉了如何从零开始设计一个具有关系完整性的数据库。
- 3) 通过本次实验，我了解了如何实现前后端之间的交互。
- 4) 通过本次实验，我熟悉了 MySQL 的视图、触发器、索引和存储过程等相关的内容。

9 诚信承诺

本组成员郑重承诺在项目实施的过程中不发生任何不诚信现象，一切不诚信所导致的后果均由本组成员承担。

签名（手签，数字上传）：

指导教师批阅意见：

成绩评定：

指导教师签字：
年 月 日

备注：