1 实验环境

1.1 实验所用的操作系统

Windows 11 家庭中文版

1.2 主要开发工具

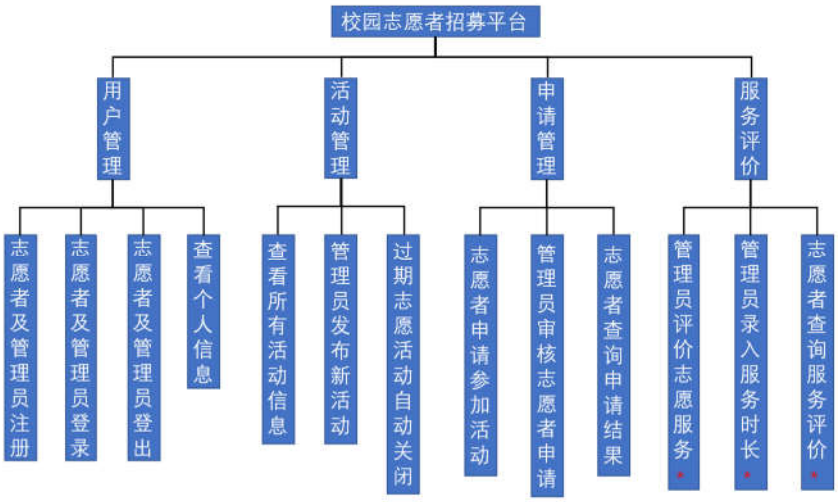
- 数据库设计：PowerDesigner（版本为 16.5）
- 数据库管理系统：MySQL Workbench（版本为 8.0）
- 前后端交互方案：使用 Django 搭建 Web 应用
- 前后端交互编程语言：Python
- 前端开发所用的 IDE：PyCharm 2022.2 (Community Edition)

2 实验过程

2.1 系统功能

2.1.1 系统设计功能（即设计数据库时所考虑的功能）

可根据本次实验所**设计**出的校园志愿者招募平台，绘制出系统的功能层次图，如下图所示（亮点功能用*标志）。



不难看出，所**设计**出的校园志愿者招募平台主要有**用户管理**、**活动管理**、**申请管理**、**服务评价**等 4 大类功能。

（1）用户管理

该平台有志愿者（即普通用户）、管理员这两种类型的用户。使用者可根据自身需要完成这两种不同类型用户的**注册**、**登录**、**登出**操作，并**查询**到自己的用户名、义工组织、学院等**个人信息**。

（2）活动管理

该平台允许使用者（**志愿者登录状态、管理员登录状态、未登录状态下均可**）**查看全部已有活动的信息**。管理员可根据实际需要，填写活动名称、时间、地点、人数、要求等活动信息，完成**新活动的发布**。此外，系统会判断活动时间与当前时间之间的先后关系，从而**自动关闭已经过期的活动**。

（3）申请管理

志愿者可**申请参加**尚未结束且正处于招募状态的活动。志愿者提交申请后，管理员可查看到自己所发布活动的申请人清单，并对申请人进行**审核（通过或拒绝）**。与此同时，志愿者可以**查看审核结果**，并观察到已提交的活动申请是**待审核、通过还是拒绝状态**。

（4）服务评价（亮点功能，已在功能层次图用*标志）

志愿者完成某次志愿活动的服务后，由**发布该活动的管理员**对志愿者的实际服务情况进行评价。**服务评价**具体包括：①录入该名志愿者在本次活动中的**实际服务小时数**；②对志愿者的整体服务表现进行**评分**（满分 10 分，最低不能低于 0 分）；③为志愿者的本次服务撰写**评语**。

活动管理员完成服务评价后，志愿者可以**查询评价结果**。志愿者的累计服务时长、平均服务得分等属性也会随之更新。

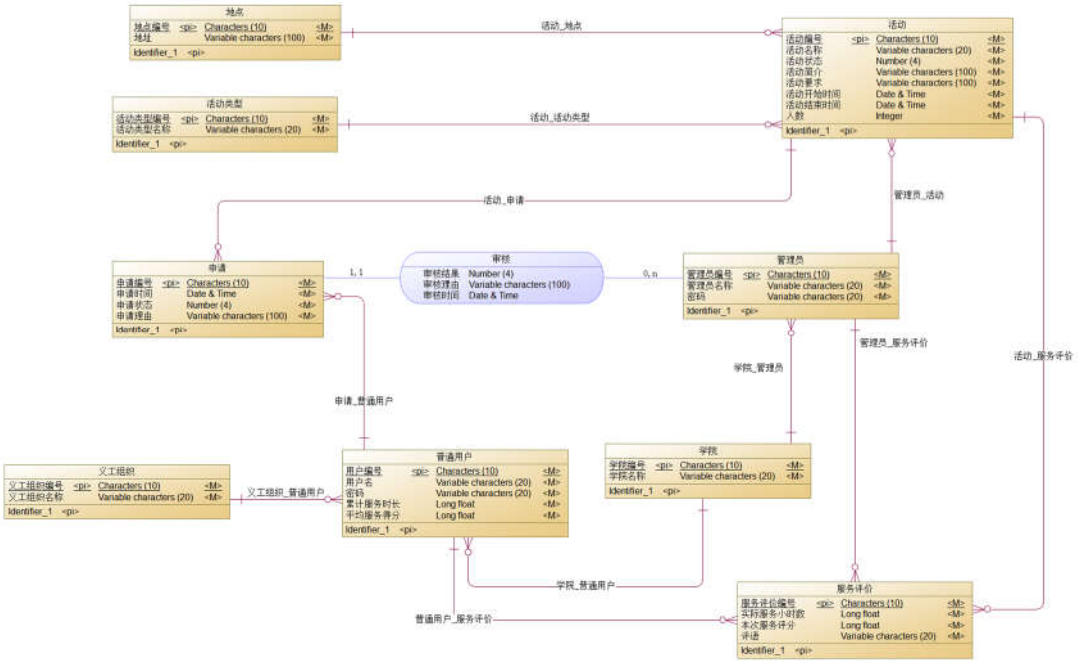
2.1.2 系统实现功能（即实际编程时所实现的功能）

完成了该校园志愿者招募平台的设计后，实际编程**实现**了如下功能：

- 在**未登录状态、志愿者登录状态**或**活动管理员登录状态**下**均能查看**全部已有活动的信息；
- 志愿者、活动管理员两种用户进行**注册**、**登录**、**登出**，并**查看个人信息**；
- 活动管理员输入时间、地点、人数、要求等信息，**发布新活动**；
- 志愿者**申请**参加活动；
- 活动管理员**查询自己所发布活动的申请人清单**，对志愿者所提交的申请进行**审核**，并**通过或拒绝**志愿者的申请；
- 志愿者**查询申请结果**，并能观察到自己提交的申请是**待审核**、**通过**还是**拒绝**状态。

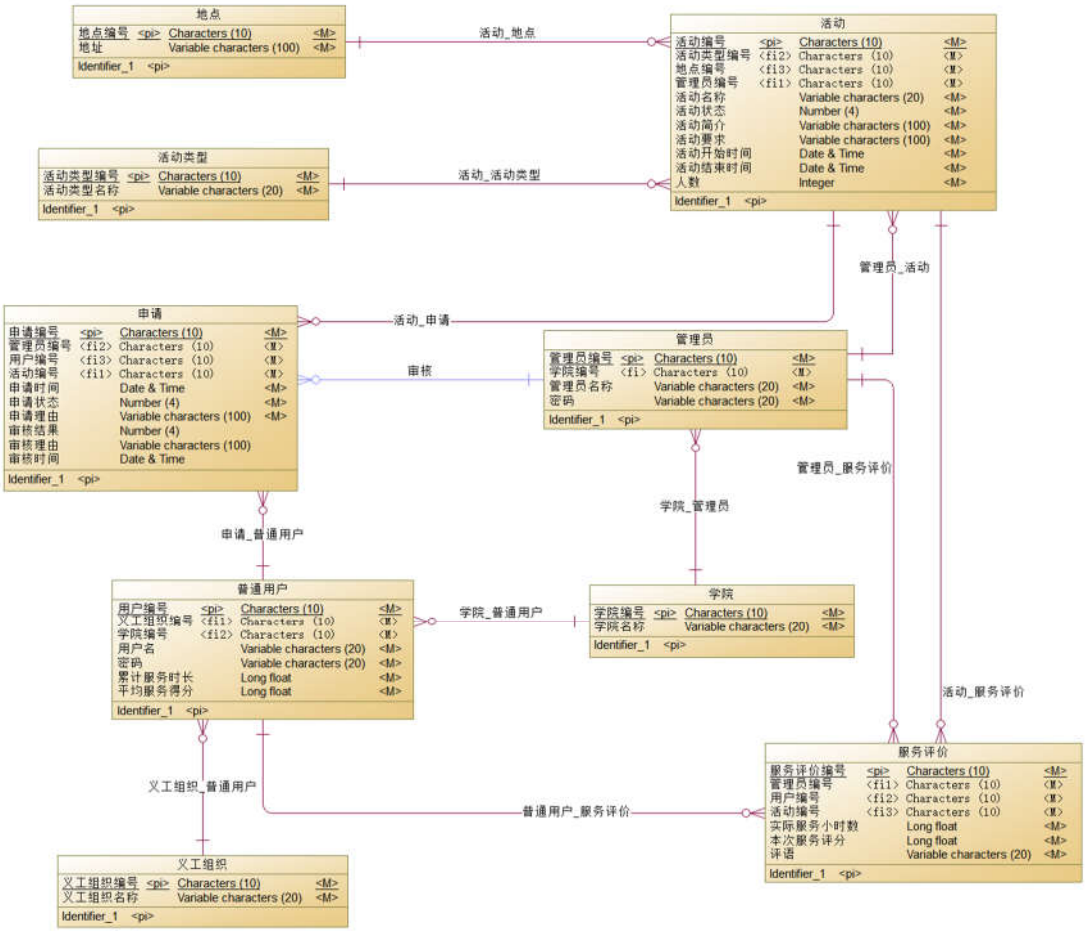
2.2 数据库设计

2.1.1 ER 图

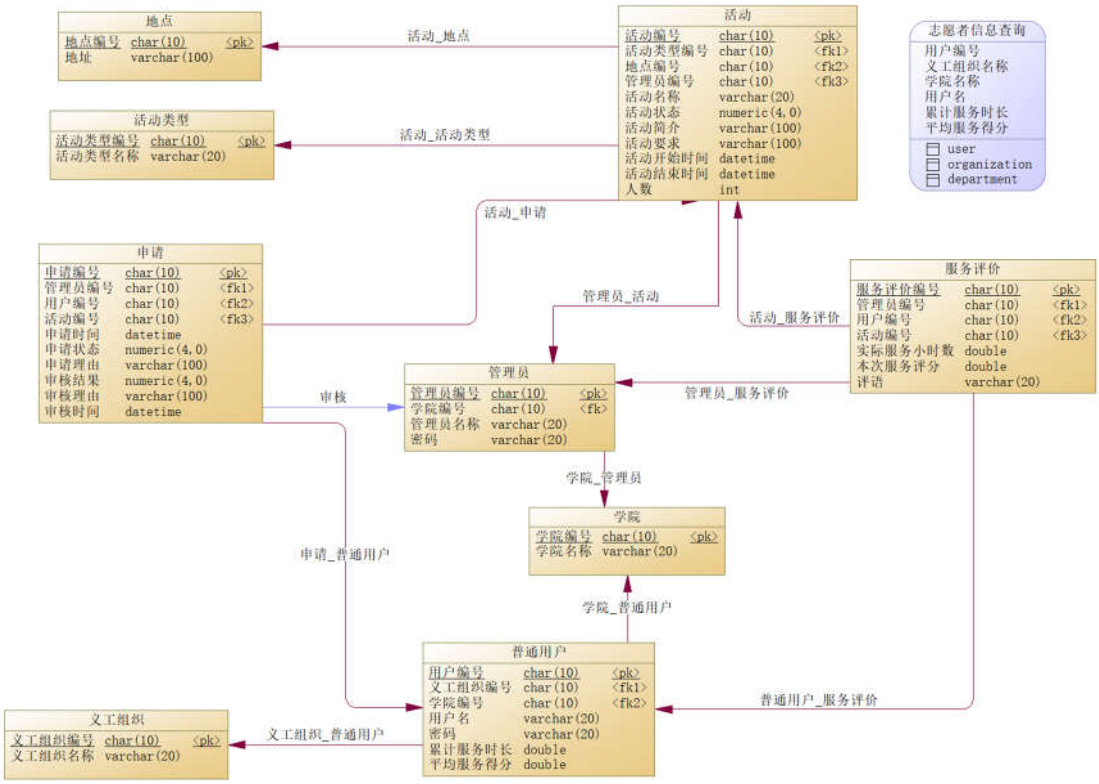


所设计的 ER 图总共有地点、活动类型、活动、申请、管理员、义工组织、普通用户、学院、服务评价这 **9 个实体**，并包含有 **12 个联系**（其中有 1 个是本身包含了属性的**关联**“审核”）。

2.1.2 LDM 图


地点
地点编号 <pi> Characters (10) <M>
地址 Variable characters (100) <M>
Identifier_1 <pi>
活动_地点
活动
活动编号 <pi> Characters (10) <M>
活动类型编号 <fi2> Characters (10) <M>
地点编号 <fi3> Characters (10) <M>
管理员编号 <fi1> Characters (10) <M>
活动名称 Variable characters (20) <M>
活动状态 Number (4) <M>
活动简介 Variable characters (100) <M>
活动要求 Variable characters (100) <M>
活动开始时间 Date & Time <M>
活动结束时间 Date & Time <M>
人数 Integer <M>
Identifier_1 <pi>
活动类型
活动类型编号 <pi> Characters (10) <M>
活动类型名称 Variable characters (20) <M>
Identifier_1 <pi>
活动_活动类型
管理员_活动
申请
申请编号 <pi> Characters (10) <M>
管理员编号 <fi2> Characters (10) <M>
用户编号 <fi3> Characters (10) <M>
活动编号 <fi1> Characters (10) <M>
申请时间 Date & Time <M>
申请状态 Number (4) <M>
申请理由 Variable characters (100) <M>
审核结果 Number (4)
审核理由 Variable characters (100)
审核时间 Date & Time
Identifier_1 <pi>
活动_申请
审核
管理员
管理员编号 <pi> Characters (10) <M>
学院编号 <fi> Characters (10) <M>
管理员名称 Variable characters (20) <M>
密码 Variable characters (20) <M>
Identifier_1 <pi>
管理员_服务评价
学院_管理员
申请_普通用户
普通用户
用户编号 <pi> Characters (10) <M>
义工组织编号 <fi1> Characters (10) <M>
学院编号 <fi2> Characters (10) <M>
用户名 Variable characters (20) <M>
密码 Variable characters (20) <M>
累计服务时长 Long float <M>
平均服务得分 Long float <M>
Identifier_1 <pi>
学院_普通用户
学院
学院编号 <pi> Characters (10) <M>
学院名称 Variable characters (20) <M>
Identifier_1 <pi>
活动_服务评价
义工组织_普通用户
普通用户_服务评价
服务评价
服务评价编号 <pi> Characters (10) <M>
管理员编号 <fi1> Characters (10) <M>
用户编号 <fi2> Characters (10) <M>
活动编号 <fi3> Characters (10) <M>
实际服务小时数 Long float <M>
本次服务评分 Long float <M>
评语 Variable characters (20) <M>
Identifier_1 <pi>
义工组织
义工组织编号 <pi> Characters (10) <M>
义工组织名称 Variable characters (20) <M>
Identifier_1 <pi>

2.1.3 PDM 图



2.1.4 数据库表结构

1、 表结构

（1）表 1：普通用户（即志愿者）表

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
user_id	CHAR(10)	☑	☑	☐	☐	☐	☐	☐	☐	
organization_id	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
department_id	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
username	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐	☐	
password	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐	☐	
total_hours	DOUBLE	☐	☑	☐	☐	☐	☐	☐	☐	
average_score	DOUBLE	☐	☑	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	

主键约束：普通用户（即志愿者）表的主键为 user_id，即每个志愿者都会有一个能

唯一标识其记录的用户编号。

外键约束：包括有义工组织编号 organization_id、学院编号 department_id 等。这是因为义工组织名称、学院名称等信息被分别保存在义工组织表和学院表中，而每个志愿者都有其对应的义工组织、学院信息，因此需要在普通用户（即志愿者）表中使用义工组织表、学院表的主键来作为外键。

空值约束（非空约束）：图中“NN”选项被勾选的属性都被设置成了非空属性（not null）。这表明用户编号、义工组织编号、学院编号、用户名、密码、累计服务时长、平均服务得分等属性的取值都不能为空。

（2）表 2：活动表

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
activity_id	CHAR(10)	☑	☑	☐	☐	☐	☐	☐	☐	
activity_type_id	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
location_id	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
admin_id	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
activity_name	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐	☐	
activity_state	DECIMAL(4,0)	☐	☑	☐	☐	☐	☐	☐	☐	
activity_introduction	VARCHAR(100)	☐	☑	☐	☐	☐	☐	☐	☐	
activity_requirements	VARCHAR(100)	☐	☑	☐	☐	☐	☐	☐	☐	
activity_start_time	DATETIME	☐	☑	☐	☐	☐	☐	☐	☐	
activity_end_time	DATETIME	☐	☑	☐	☐	☐	☐	☐	☐	
headcount	INT	☐	☑	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	

主键约束：活动表的主键为 activity_id，即每个活动都会有一个能唯一标识其记录的活动编号。

外键约束：包括有活动类型编号 activity_type_id、地点编号 location_id、管理员编号 admin_id 等。这是因为活动类型名称、地址、管理员名称等信息被分别保存在活动类型表、地点表、管理员表中，而每个活动都有其对应的活动类型、地点、管理员信息，因此需要在活动表中使用活动类型表、地点表、管理员表的主键来作为外键。

空值约束（非空约束）：图中“NN”选项被勾选的属性都被设置成了非空属性（not null）。这表明活动编号、活动类型编号、地点编号、管理员编号、活动名称、活动状态、活动简介、活动要求、活动开始时间、活动结束时间、人数等属性的取值都不能为空。

（3）表 3：管理员表

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
admin_id	CHAR(10)	☑	☑	☐	☐	☐	☐	☐	☐	
department_id	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
admin_name	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐	☐	
admin_password	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	

主键约束：管理员表的主键为 admin_id，即每个管理员都会有一个能唯一标识其记录的管理员编号。

外键约束：包括有学院编号 department_id。这是因为学院名称信息被保存在学院表中，而每个管理员都有其对应的学院信息，因此需要在管理员表中使用学院表的主键来作为外键。

空值约束（非空约束）：图中“NN”选项被勾选的属性都被设置成了非空属性（not null）。这表明管理员编号、学院编号、管理员名称、密码等属性的取值都不能为空。

2、 索引

1) 索引截图

活动表中定义了 **2 个索引** idx_start_time、idx_end_time（非主键、非外键索引），分别对应活动表中的 activity_start_time、activity_end_time 属性，如下所示。

```
30    /*==============================================================*/
31    /* Index: idx_start_time                                        */
32    /*==============================================================*/
33    create index idx_start_time on activity
34    (
35       activity_start_time
36    );
37
38    /*==============================================================*/
39    /* Index: idx_end_time                                          */
40    /*==============================================================*/
41    create index idx_end_time on activity
42    (
43       activity_end_time
44    );
```

Column	Type	Nullable	Indexes
activity_id	char(10)	NO	PRIMARY
activity_type_id	char(10)	NO	FK_activity_activity_type
location_id	char(10)	NO	FK_activity_location
admin_id	char(10)	NO	FK_admin_activity
activity_name	varchar(20)	NO	
activity_state	decimal(4,0)	NO	
activity_introduction	varchar(100)	NO	
activity_requirements	varchar(100)	NO	
activity_start_time	datetime	NO	idx_start_time
activity_end_time	datetime	NO	idx_end_time
headcount	int	NO	

2) 使用场景（用途）

这两个索引分别针对活动表的活动开始时间 activity_start_time、活动结束时间 activity_end_time 字段建立，可**加快根据活动开始时间、活动结束时间来查找活动的速度**。实际中有可能发生这样的场景。例如，志愿者如果更希望参加位于某一个具体时间段而非其它时间段的活动，就可以根据活动开始时间、活动结束时间来查找位于这个时间段之内的活动。

此外，为活动开始时间、活动结束时间建立索引还意味着可以**加快让活动根据开始时间、结束时间来进行排序的速度**。实际中同样可能发生这样的场景，因为活动除了需要按照名称大小进行排序外，同样可能需要按照其开始时间、结束时间的先后顺序来进行排序，从而能让用户查询到最新的、最早的活动都有哪些。

3、 视图

1) 视图截图

普通用户（即志愿者）表中定义了一个可以用于查询志愿者信息的视图，如下所示。

2) 使用场景（用途）

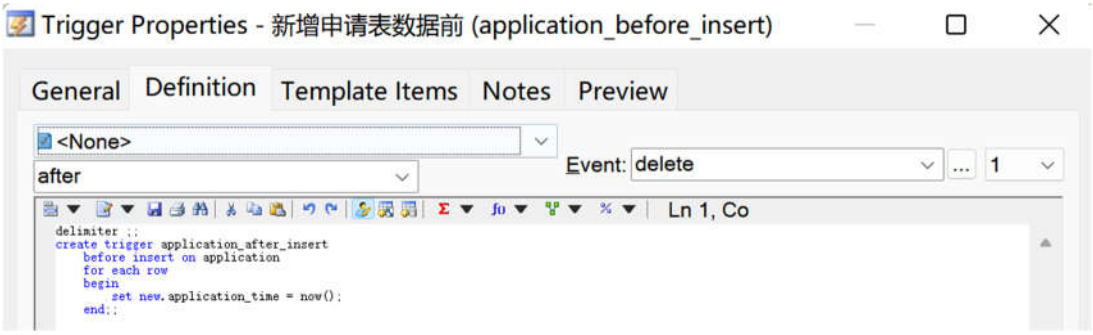
该视图可以用于**显示全体志愿者的个人信息（但是不包括密码 password 属性）**。实际中有可能发生这样的场景，因为确实有可能出现需要展示全体志愿者的编号、义工组织、学院、用户名、累计服务时长、平均服务得分等属性的情况。当然，志愿者的密码 password 属性不能公开，因此在定义的视图中并没有这一属性。

4、 触发器

1) 触发器截图

【触发器 1】自动设置申请时间为函数 now()的返回值

```
183     delimiter ;;
184 •   create trigger application_after_insert
185         before insert on application
186         for each row
187         begin
188             set new.application_time = now();
189         end;;
```



【触发器 2】新增服务评价后自动更新志愿者的累计服务时长、平均服务得分

```
192     delimiter ;;
193 •   create trigger service_evaluation_after_insert after insert on service_evaluation
194         for each row
195         begin
196             update user
197             set
198                 user.average_score = (user.average_score * user.total_hours + new.service_hours * new.score) / (user.total_hours + new.service_hours),
199                 user.total_hours = user.total_hours + new.service_hours
200             where
201                 user.user_id = new.user_id;
202         end;;
```

2)　　使用场景（用途）

【触发器 1】自动设置申请时间为函数 now()的返回值

定义该触发器后，可以在新增一条申请记录时将 application_time 属性的值设置为函数 now()的返回值，从而实现**获取当前时间并自动记录志愿者申请时间**的功能。

【触发器 2】新增服务评价后自动更新志愿者的累计服务时长、平均服务得分

定义该触发器后，可以在志愿评价表新增一条服务评价记录（即管理员完成一次服务评价）后自动更新志愿者的 total_hours、average_score 等属性。其中，累计服务时长 total_hours 被更新为**旧值加上本次的实际服务时长**，而平均服务得分 average_score 则是由**单次服务评分按服务时长大小加权平均计算得出**。

3)　　验证触发器

【触发器 1】自动设置申请时间为函数 now()的返回值

触发前：

application_id	admin_id	user_id	activity_id	application_time	application_state	application_reason	check_result	check_reason	check_time
0	2	0	0	2023-01-16 21:32:25	1	无	1	同意	2022-12-10 06:00:00
1	4	1	1	2023-01-16 21:32:25	1	无	1	无	2022-12-11 04:00:00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

新增一条记录：

```
1  Insert Into application
2      (application_id, admin_id, user_id, activity_id, application_state, application_reason)
3      values(2, 2, 4, 0, 0, "无")
4
```

触发后：

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

application_id	admin_id	user_id	activity_id	application_time	application_state	application_reason	check_result	check_reason	check_time
0	2	0	0	2023-01-16 21:32:25	1	无	1	同意	2022-12-10 06:00:00
1	4	1	1	2023-01-16 21:32:25	1	无	1	无	2022-12-11 04:00:00
2	2	4	0	2023-01-16 22:51:11	0	无	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

application 1

Output

Action Output

#	Time	Action	Message
1	22:51:11	Insert Into application (application_id, admin_id, user_id, activity_id, application_...	1 row(s) affected

观察不难发现，虽然新增记录的 Insert 语句并没有指明 application_time 的值，但这条新增记录的 application_time 的值还是被设置为了系统的当前时间。这说明**所定义的触发器确实生效了**。

【触发器 2】新增服务评价后自动更新志愿者的累计服务时长、平均服务得分
触发前的 user 表：

```
1 •  SELECT * FROM volunteer_recruitment.user;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell

user_id	organization_id	department_id	username	password	total_hours	average_score
0	0	5	张三	zhangsan	0	0
1	1	0	李四	lisi	0	0
2	0	1	王五	wangwu	0	0

新增一条服务评价记录：

```
1 •  Insert Into service_evaluation
2        (service_evaluation_id, admin_id, user_id, activity_id, service_hours, score, comment)
3        values(0, 2, 0, 0, 1, 9.8, "无")
```

触发后的 user 表：

user_id	organization_id	department_id	username	password	total_hours	average_score
0	0	5	张三	zhangsan	1	9.8
1	1	0	李四	lisi	0	0
2	0	1	王五	wangwu	0	0

Output

Action Output

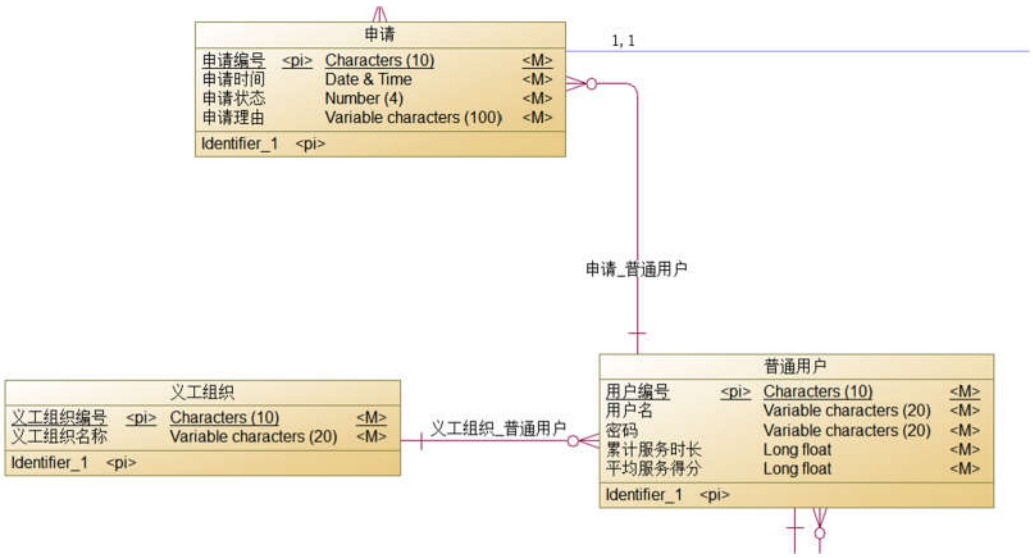
#	Time	Action	Message
1	23:03:47	Insert Into service_evaluation (service_evaluation_id, admin_id, user_id, activity...	1 row(s) affected

观察不难发现，新增一条服务记录后，user 表中相应志愿者的**累计服务时长、平均服务得分也随之被更新为了正确的值**。这说明**所定义的触发器确实生效了**。

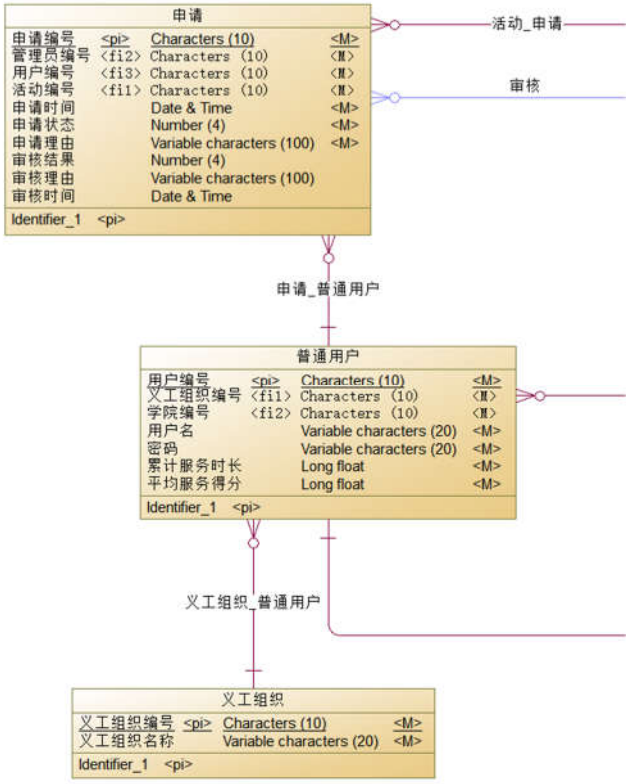
2.1.5 分析

选取了三个实体：申请、普通用户（即志愿者）、义工组织。这三者之间一共存在两种联系：申请_普通用户、义工组织_普通用户。

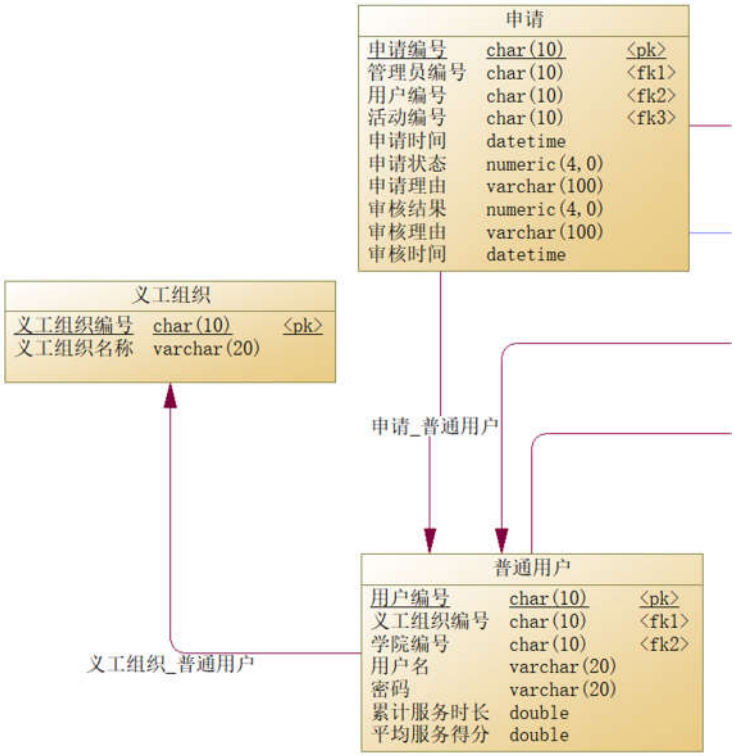
ER 图：


申请
申请编号 <pi> Characters (10) <M>
申请时间 Date & Time <M>
申请状态 Number (4) <M>
申请理由 Variable characters (100) <M>
Identifier_1 <pi>
1, 1
申请_普通用户
义工组织
义工组织编号 <pi> Characters (10) <M>
义工组织名称 Variable characters (20) <M>
Identifier_1 <pi>
义工组织_普通用户
普通用户
用户编号 <pi> Characters (10) <M>
用户名 Variable characters (20) <M>
密码 Variable characters (20) <M>
累计服务时长 Long float <M>
平均服务得分 Long float <M>
Identifier_1 <pi>

LDM 图：


申请
申请编号 <pi> Characters (10) <M>
管理员编号 <fi2> Characters (10) <M>
用户编号 <fi3> Characters (10) <M>
活动编号 <fi1> Characters (10) <M>
申请时间 Date & Time <M>
申请状态 Number (4) <M>
申请理由 Variable characters (100) <M>
审核结果 Number (4)
审核理由 Variable characters (100)
审核时间 Date & Time
Identifier_1 <pi>
活动_申请
审核
申请_普通用户
普通用户
用户编号 <pi> Characters (10) <M>
义工组织编号 <fi1> Characters (10) <M>
学院编号 <fi2> Characters (10) <M>
用户名 Variable characters (20) <M>
密码 Variable characters (20) <M>
累计服务时长 Long float <M>
平均服务得分 Long float <M>
Identifier_1 <pi>
义工组织_普通用户
义工组织
义工组织编号 <pi> Characters (10) <M>
义工组织名称 Variable characters (20) <M>
Identifier_1 <pi>

PDM 图：



【分析】

（1）ER 图中的联系

一个普通用户（即志愿者）可以提交多次申请，当然也可以一次申请都不提交；但是一个申请必须要有一个提交它的普通用户（即志愿者）。因此，普通用户、申请之间的关系是一个**一对多**的关系，并且普通用户是**部分参与联系**（**因为可以有用户暂时还没有提交活动申请**），申请则是**完全参与联系**（**因为所有已产生的申请都必定要有一个提交它的用户**）。

同理，普通用户（即志愿者）与义工组织之间也是一个**一对多**的关系。一个普通用户（即志愿者）必然要有其所属的义工组织（并且也只能有一个），因此普通用户是**完全参与联系**；但是一个义工组织可能暂时还没有其所属成员注册并被登记到数据库中，因此义工组织是**部分参与联系**。

（2）从 ER 图到 LDM 图

前面已经分析过，所选取的实体之间存在的关系都是**一对多**的关系。因此在从 ER 图变换到 LDM 图的过程中，只需要将**一对多关系**的一方的**主键**添加到另一方作为**外键**。普通用户、申请之间是**一对多**的关系，可以观察到普通用户的主键用户编号被作为了申请的外键，并且由于申请是**完全参与联系**，普通用户编号这一外键也被设置了**非空约束**。

同理，普通用户、义工组织之间是**一对多**的关系，义工组织的主键义工组织编号被作为了普通用户的外键，并且由于普通用户是**完全参与联系**，义工组织编号这一外键也被设置了非空约束。

（3）从 LDM 图到 PDM 图

从 LDM 图变换到 PDM 图，需要进行的改动并不大。可以观察到 LDM 图中实体的属性都被完整地保存到了 PDM 图中，不同之处在于实体之间的关系在 PDM 图中被直接简化成了箭头。**箭头由保存了外键的实体指向被引用的实体**。此外，PDM 中的数据类型也被转变成了更适合物理数据库存储的类型，<pi>、<fi>等也被转换成了<pk>、<fk>。

PDM 相比 LDM 而言要更接近最终所建立的数据库表。本次实验中索引、视图、触发器等的建立都是在 PDM 中进行的。

完成 PDM 的生成后，就可以导出 mysql 脚本新建数据库了。最终生成的数据库表结构也与 PDM 图高度相似。

3 收获和反思

收获：

本次实验让我充分体验了一个完整的数据库设计、实现以及小型系统设计、实现的过程。让我充分夯实了数据库设计和实现的相关知识，包括实体、联系、ER 图/LDM 图/PDM 图、触发器、索引、视图等，也让我进一步巩固了 sql 语句的相关用法。系统的实际实现也让我熟悉了前端开发的基本流程，掌握了一定的前端开发经验，提升了编程能力。

值得反思的问题及思考：

缺乏相应的前端开发经验，导致开发所需的时间较长。最后也只是在设计上完成了亮点功能（**服务评价部分**），实际编程时只是完成了本次实验所要求的必做部分。以后面对类似任务需要提前学习好相应的预备知识。