



哈爾濱工業大學 (深圳)
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期: 2021 春季
课程名称: 数据库系统
实验名称: DPSV 系统设计与实现
实验性质: 设计型
实验学时: 2 地点: T2210
学生班级: 1801105
学生学号: 180110516
学生姓名: 潘延麒
评阅教师:
报告成绩:

实验与创新实践教育中心制

2021 年 1 月

1 实验环境

- 操作系统: Window 10 Professional
- 开发环境: VS Code V1.552
- 开发框架: Flask + React JS + Mysql

2 实验过程

2.1 系统功能

DPSV 是 Deadpool Short Video 的缩写, 意为 Deadpool 短视频。该项目参考抖音功能需求进行制作, 主要囊括了 5 个板块: 用户管理模块、视频管理模块、评论相关模块、*私信相关模块以及*搜索相关模块。

- 用户管理模块

该模块负责处理用户登录注册逻辑, 响应用户更新请求, 以及处理用户关注的相关事宜。如图 1 所示:

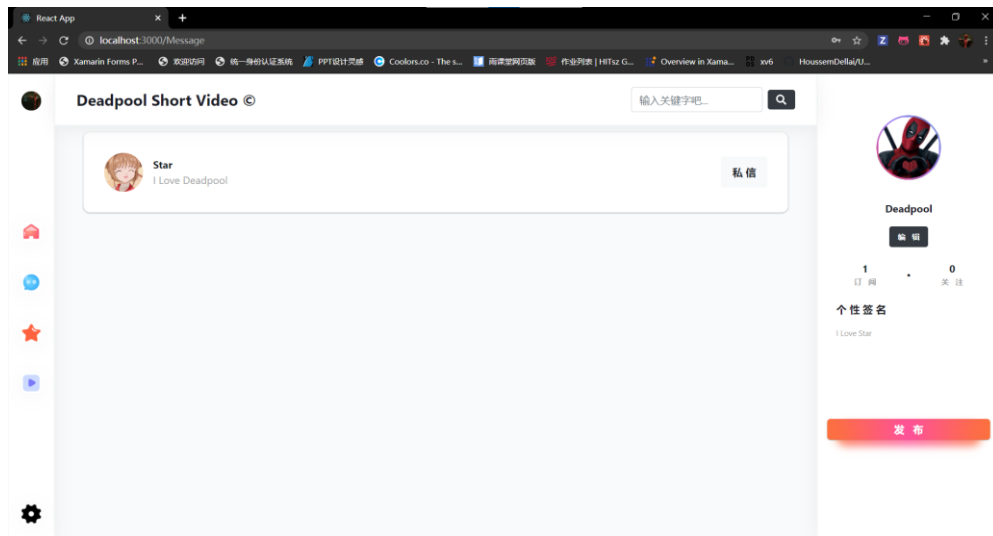


图 1 用户管理模块

- 视频管理模块

视频管理模块主要负责视频的增、删、查、改, 以及一些用户互动功能, 例如: 点赞、收藏等, 视频管理模块在系统多个界面均有体现, 接下来进行一一讲解。

图 2 是 DPSV 主页, 用于将数据库中的视频展示给用户。

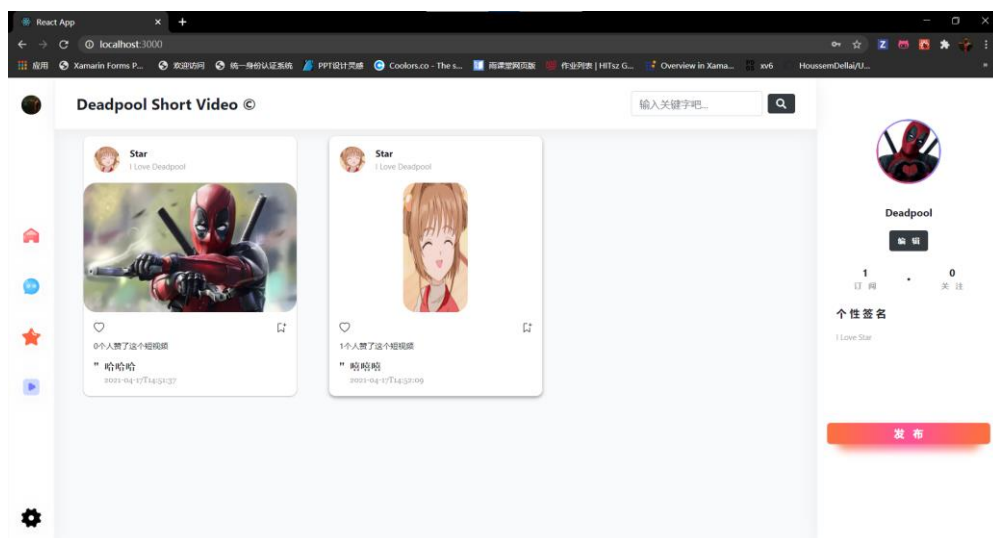


图 2 DPSV 主页

图 3 是 DPSV 用户视频管理界面，即个人作品页面，用于用户对自己发布的作品进行**查看或修改**。从图中我们可以看到，与主页不同的是，在个人作品页面的卡片右上角多出了代表**修改和删除**的按钮。

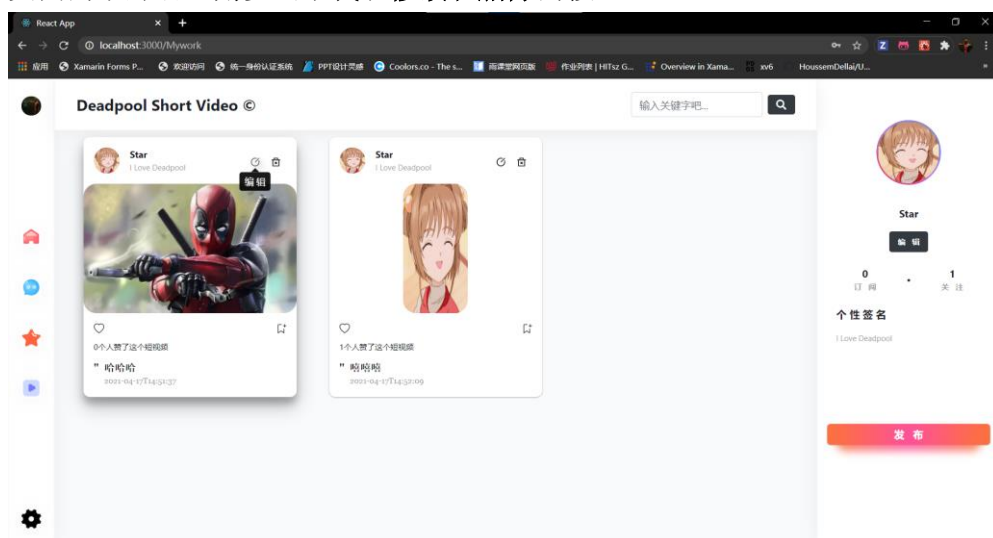


图 3 DPSV 个人作品页面

图 4 是 DPSV 视频发布界面，点击位于右侧的发布按钮即可进入该界面，该页面的主要功能是上传视频。每个视频都必须包含一个**视频元素**、一个**视频封面**、**标题**以及**描述**。

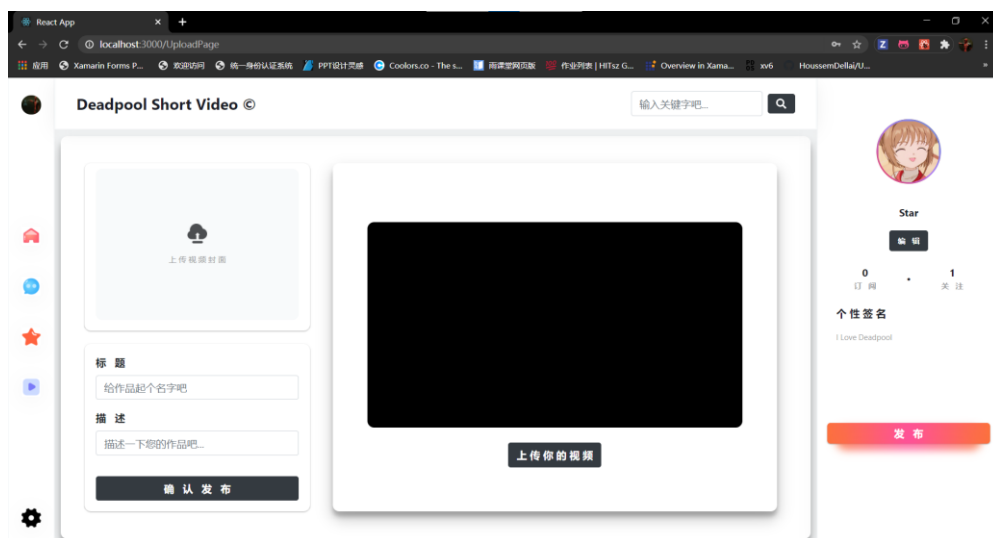


图 4 DPSV 视频上传页面

图 5 是 DPSV 视频详情页面，在该页面中可以进行关注、点赞、收藏和评论，在视频模块中，我们暂且不讨论评论功能。其中，点击关注后，该用户便将出现在私信列表，如图 1 所示。

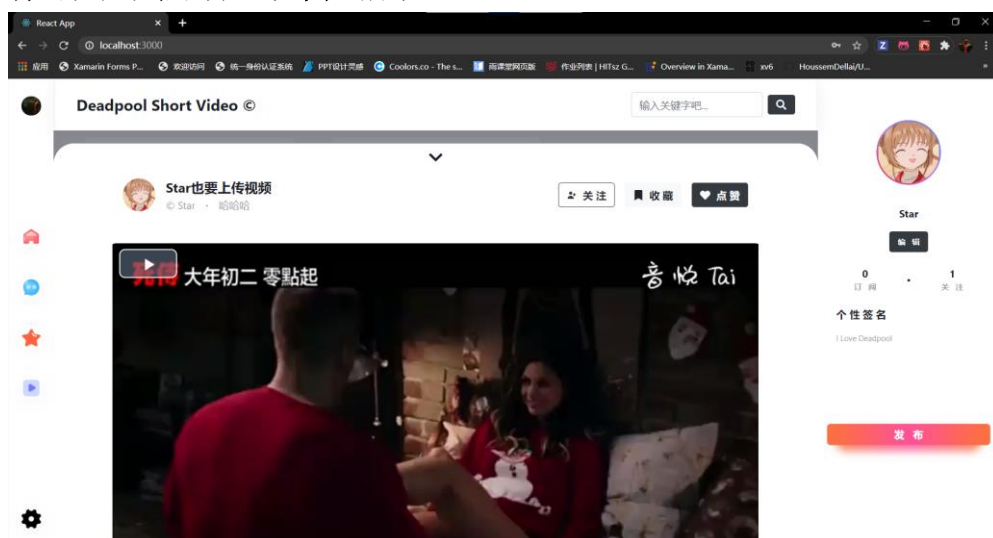


图 5 DPSV 视频详情页

当我们点击**收藏**后，被收藏的视频便将出现在 **DPSV 收藏**页面，如图 6 所示。这里由于时间原因，暂时没有考虑重复收藏的因素，因此，**演示视频**中出现了重复收藏两次的情况，不过这其实非常容易解决，只需要在插入数据前检查是否存在重复即可。

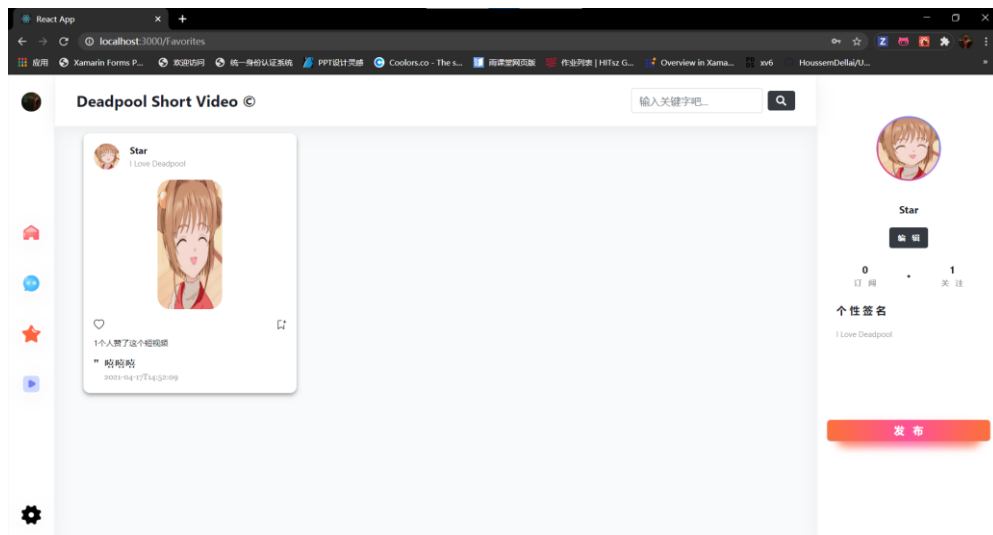


图 6 DPSV 收藏页面

- **评论相关模块**

评论相关模块由两个部分构成：**评论视频**以及**回复评论**。如图 7 所示。

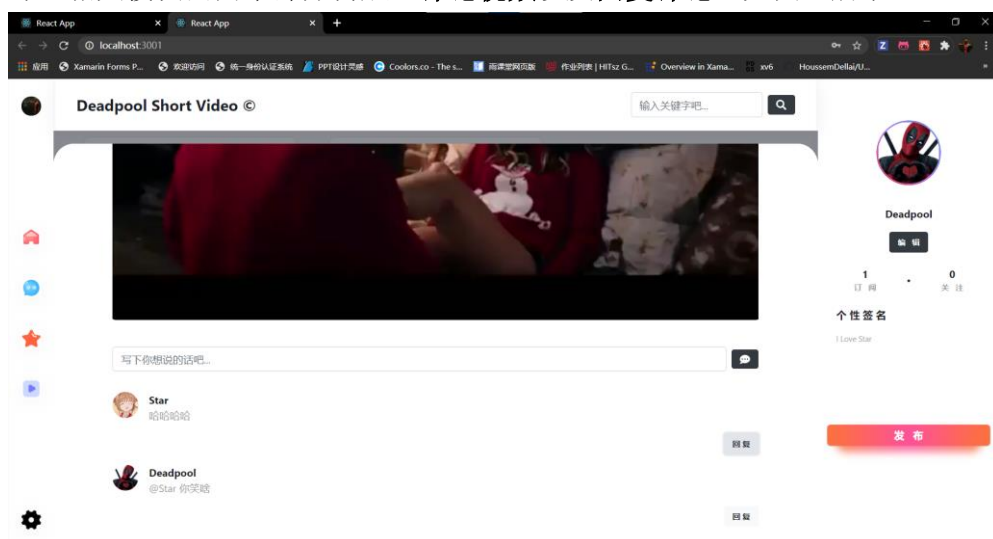


图 7 DPSV 评论及回复

- ***私信模块**

图 8 展示了私信模块的页面。在该页面中，我们能够看到所有关注我的人以及我关注的人，并且可以与他们进行私信。

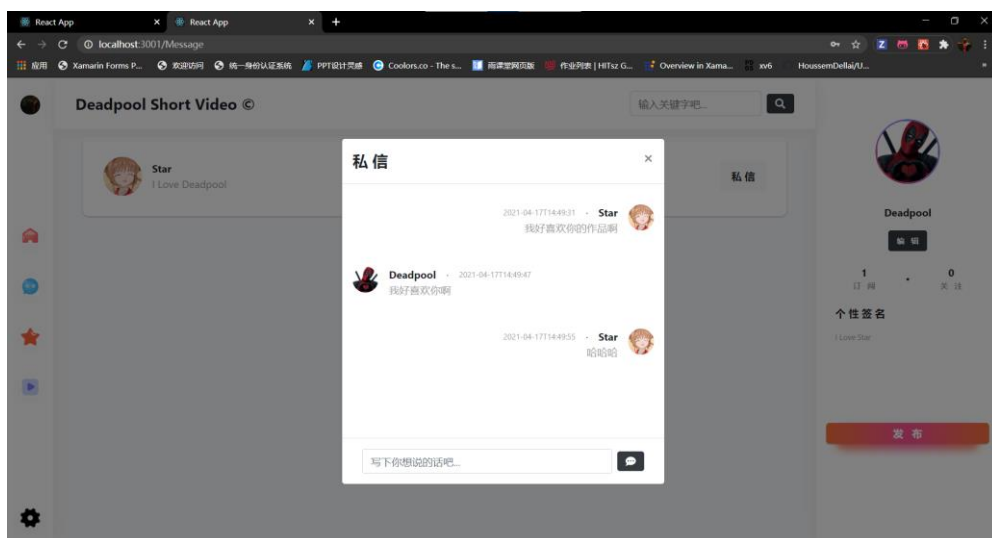


图 8 DPSV 私信页面

● *搜索模块

无论在哪个页面，点击搜索后，都能跳转到搜索页面，搜索模块能够存储已登录用户的全部搜索记录，并能够在后期对其进行数据分析，从而推送合适的视频。如图 9 所示。

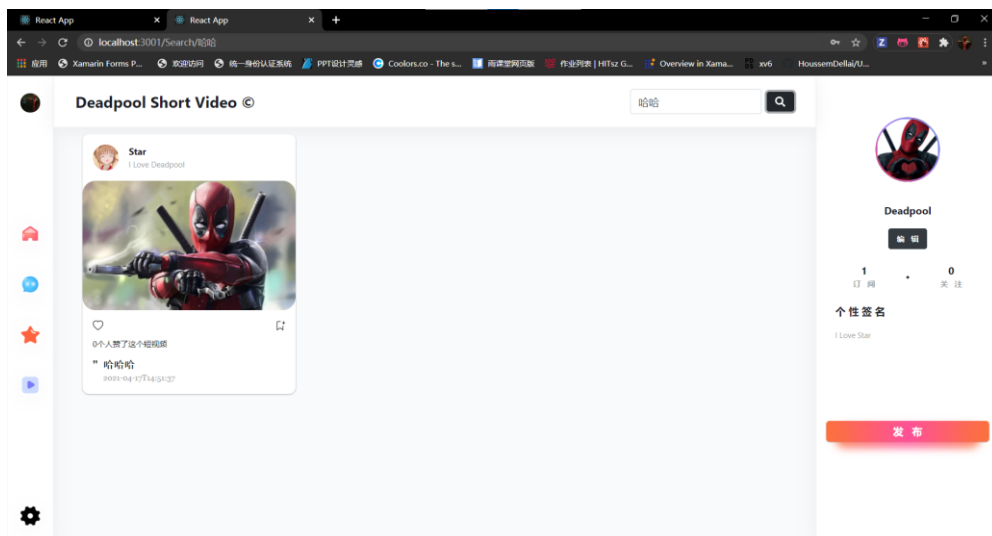


图 9 DPSV 搜索展示

2.2 数据库设计

2.1.1 ER 图

作为一个短视频项目，DPSV 的核心便是说明用户与视频、用户与用户的关系。在 CDM 的架构过程中，我从最直接的用户与视频关系入手，构建了整个 CDM 的主干 ER 网络，如图 10 所示。该结构阐释了用户与视频间的四种关系：

- **拥有关系：**一个用户可拥有多个视频；
- **收藏关系：**一个用户可收藏多个视频；

- **点赞关系：**一个用户可点赞多个视频；
- **观看关系：**一个用户可有多条观看记录；

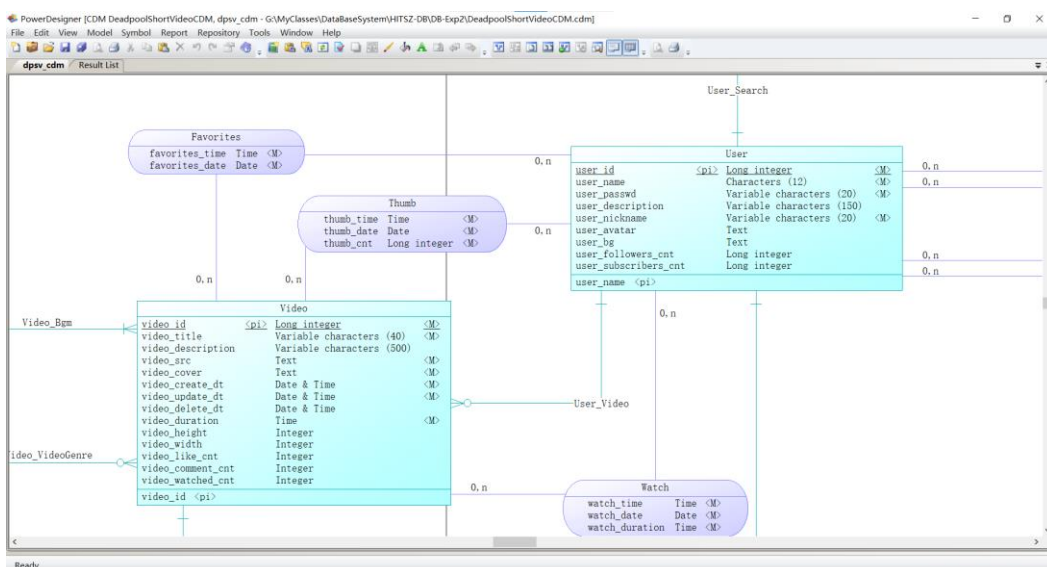


图 10 用户与视频主干 ER 网络结构

其次，我们需要处理**用户与用户间**的关系——关注、私信，图 11 显示了这个关系的构建方案：一个用户可以关注多个用户，亦可不关注用户；一个用户可以和多个用户发送私信，亦可不发送私信。

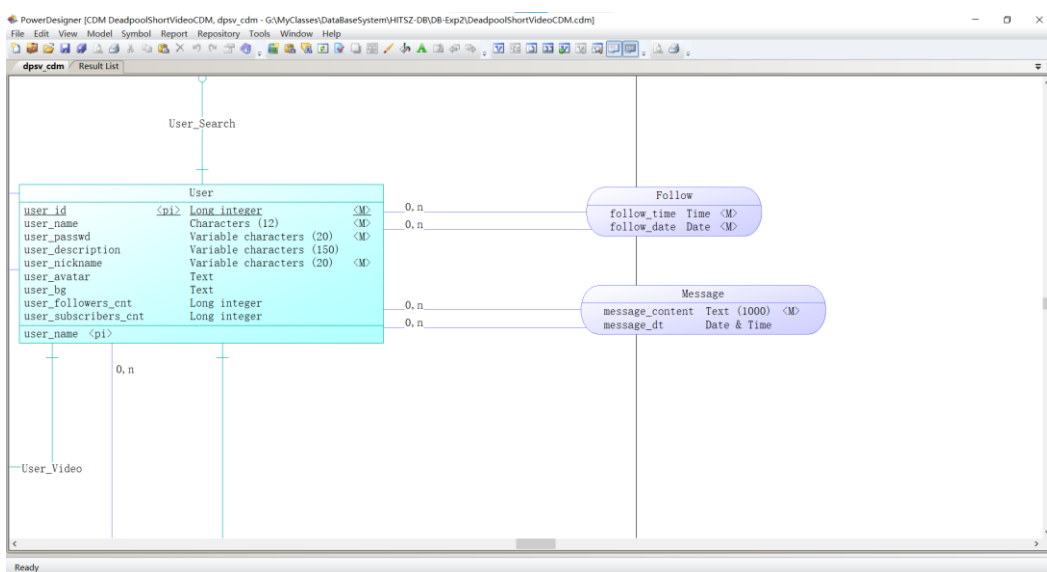


图 11 用户与用户间的关注、私信关系

接下来是处理最为复杂的**用户、视频、评论间**的关系——一个用户可评论多个视频，一个用户亦可回复一个评论，该回复依然为一个评论，因此设计了如图 12 的递归结构来表述这一关系。

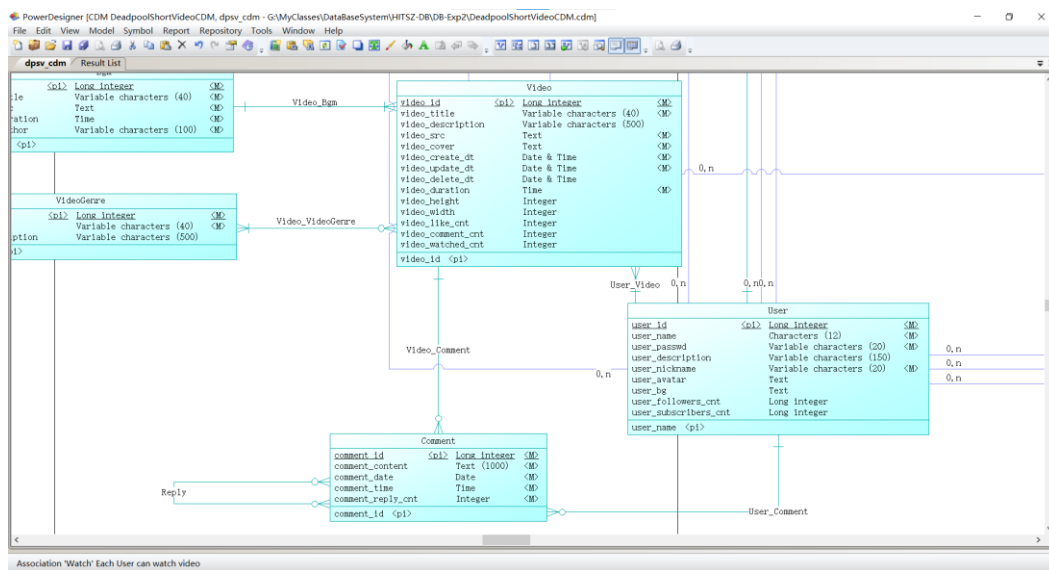


图 12 用户、视频、评论间的关系

2.1.2 LDM 图

与 CDM 图类似，首先逻辑描述用户与视频间的关系，如图 13 所示。

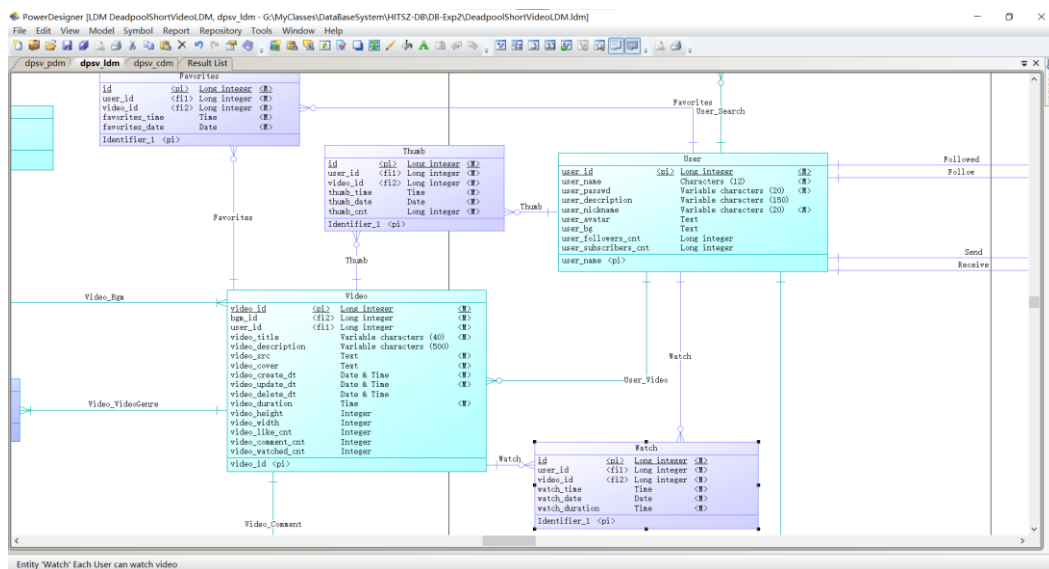


图 13 用户与视频部分 LDM 图

其次是逻辑描述用户与用户间的关系，如图 14 所示。

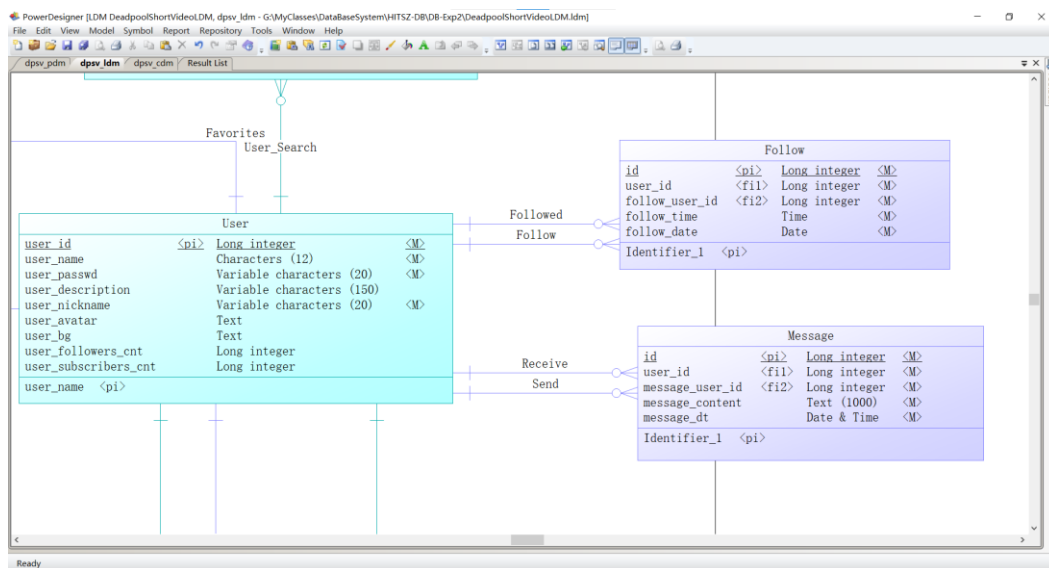


图 14 用户与用户的关注、私信关系

最后描述用户、视频、评论间的关系。如图 15 所示

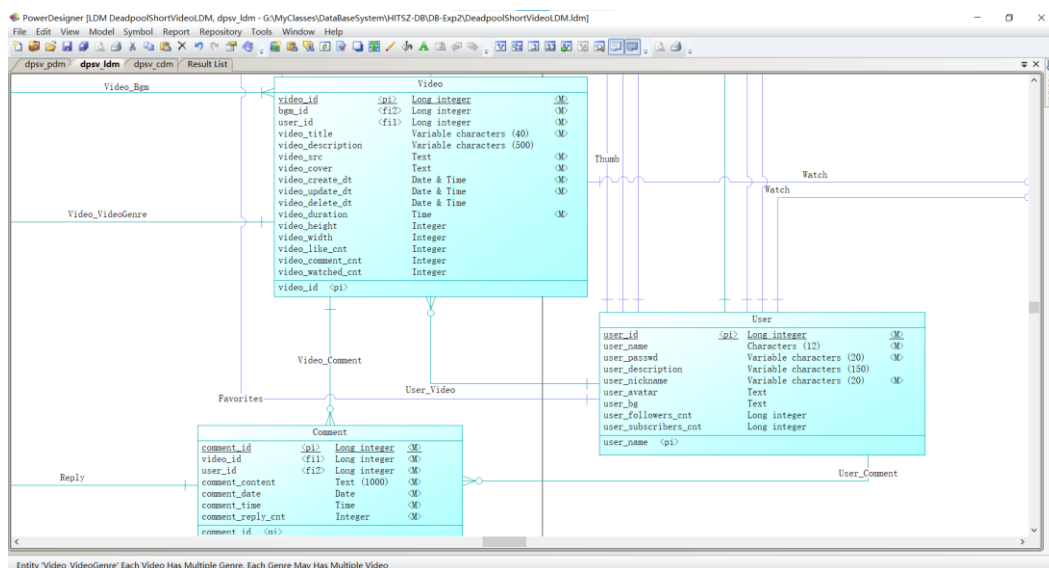
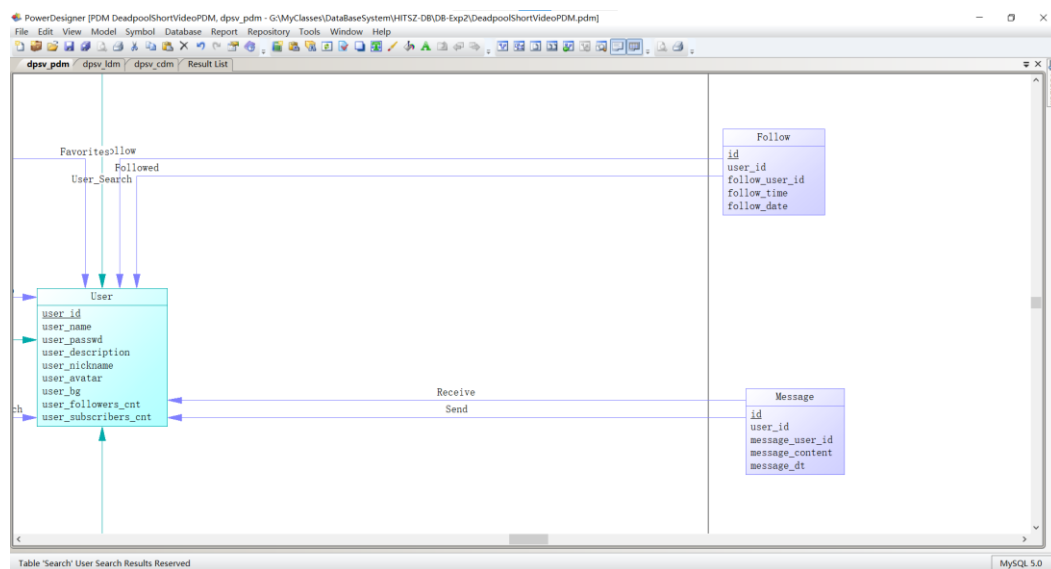
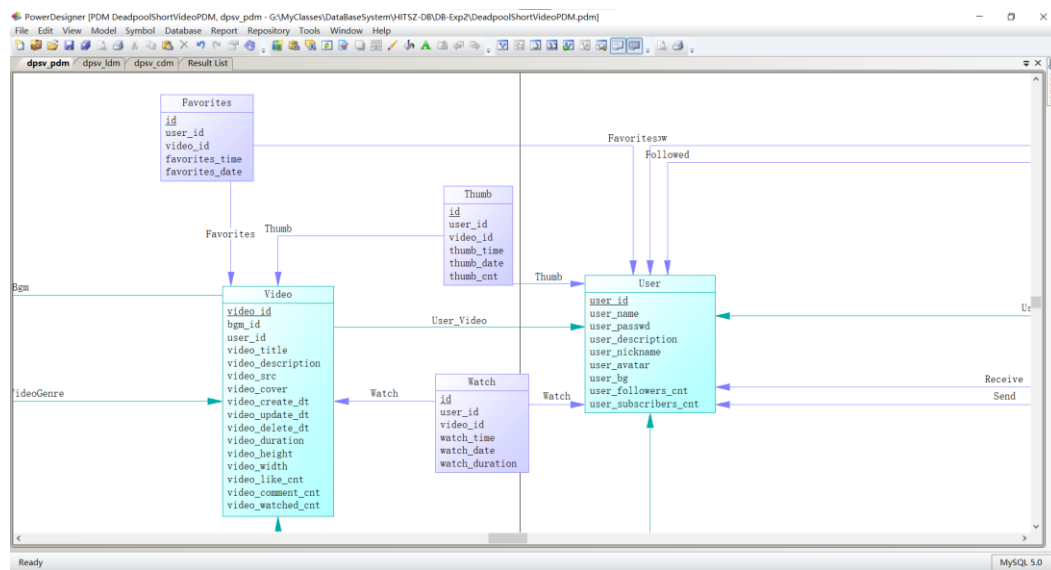


图 15 用户、视频、评论间的关系

2.1.3PDM 图

相应的 PDM 见图 16、17、18 所示，在此不做赘述。



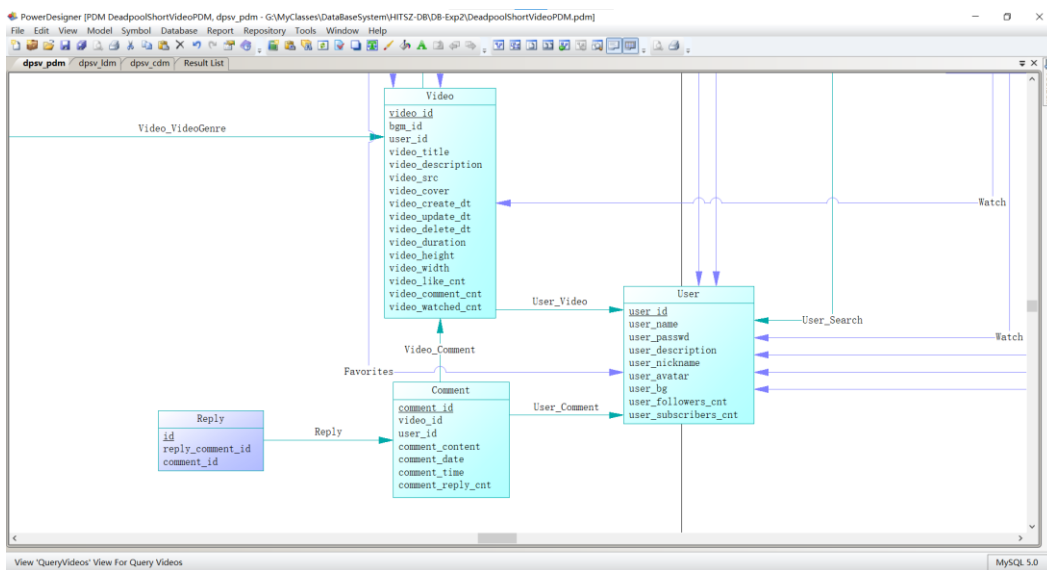


图 18 用户、视频与评论间的物理表设计

2.1.4 数据库表结构

1、 表结构

我们主要讲述视频、用户以及评论三个表，因为它们构成了 DPSV 的基础大厦。

- Video

Video 表结构如图 19 所示。一个 Video 由众多字段组成。但在实际开发过程中，发现有些字段比较冗余。表 1 阐释了 Video 中的重要字段。

图 19 Video 表结构

表 1 Video 重要字段描述

编号	字段名	意义	约束	主键
----	-----	----	----	----

1	video_id	视频唯一 ID	非空	✓
2	user_id	持有该视频的用户 ID	用户 ID	
3	video_src	视频 URL 地址	非空	
4	video_cover	视频封面 URL 地址	非空	
5	video_title	视频标题	非空	
6	video_description	视频描述	非空	
7	video_comment_cnt	视频评论数	无	
8	video_like_cnt	视频获赞次数	无	

- User

User 表结构如图 20 所示，其记录了用户的所有关键信息。因为用户是 DPSV 基石。因此，它并不需要外键约束。表 2 给出了 User 中的关键字段的解释。

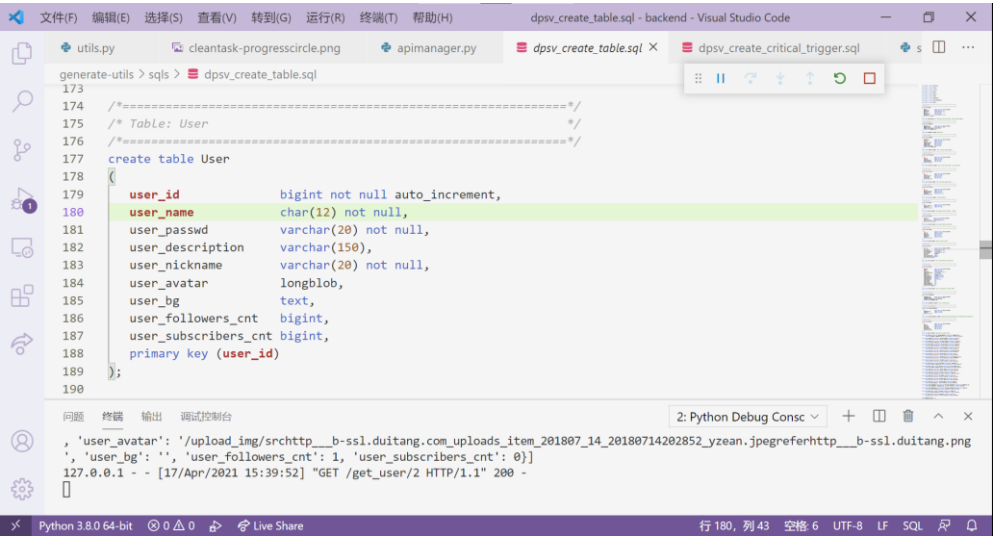


图 20 User 表结构

表 2 User 重要字段描述

编号	字段名	意义	约束	主键
1	user_id	用户唯一 ID	非空	✓
2	user_name	用户名	非空	
3	user_passwd	密码	非空	
4	user_description	用户描述	无	
5	user_nickname	用户昵称	非空	
6	user_avatar	用户头像 URL	无	
7	user_follower_cnt	用户关注的数量	无	
8	user_subscribers_cnt	用户被关注的数量	无	

- Comment

Comment 表结构如图 21 所示，其有 video_id 和 user_id 两个外键约束，表明该评论发生在哪个视频下、由哪个用户发起的。

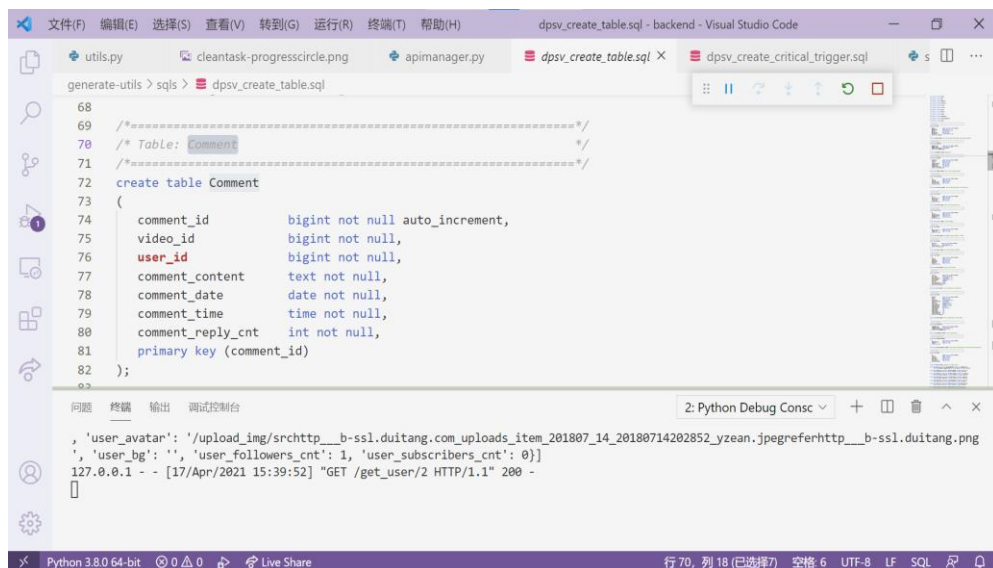


图 21 Comment 表结构

表 3 阐释了 Comment 的重要字段意义。

表 3 Comment 重要字段描述

编号	字段名	意义	约束	主键
1	comment_id	评论唯一 ID	非空	√
2	video_id	所属视频的 ID	视频 ID	
3	user_id	该评论所属用户 ID	用户 ID	
4	comment_content	评论的内容	非空	

2、索引

我为 user_name 设置了非主索引，如图 22 所示。建立该索引的根本目的还是在于不希望发生用户名重复的事情，因此，在每次插入用户数据时，都会先查询 User 表是否存在该用户名的用户，如果没有，插入请求才能够发生。而由于查询用户名是字符比较，相对来说比较慢，因此，为它创建索引将能够加速用户名的查询。

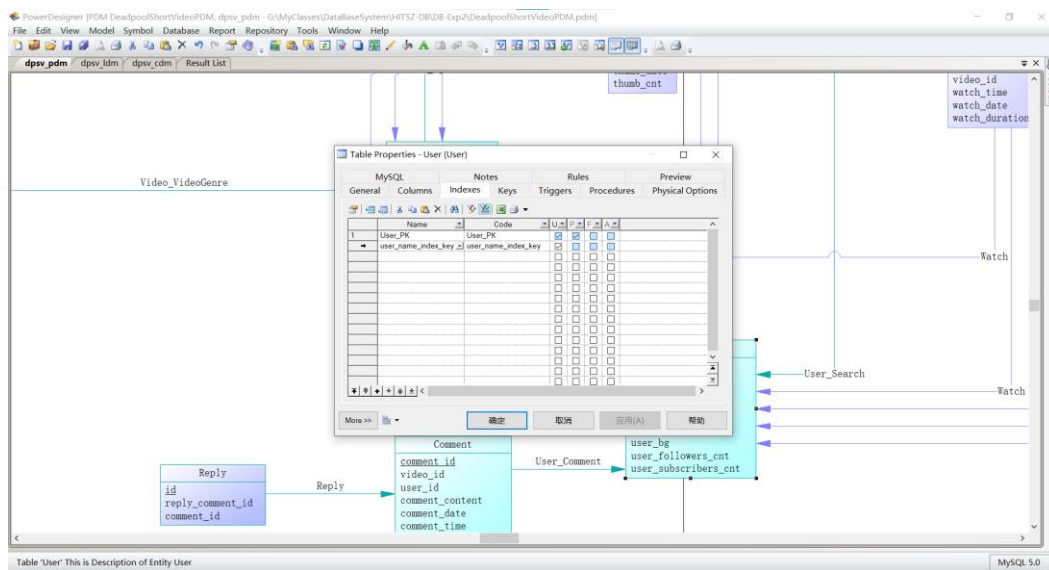


图 22 user_name 的非主索引

3、 视图

目前仅构建了三个视图，如图 23 所示。这三个视图分别是对用户、视频以及用户评论进行查询。主要目的在于测试数据库功能的正确性。在正式开发的过程中没有用到视图。

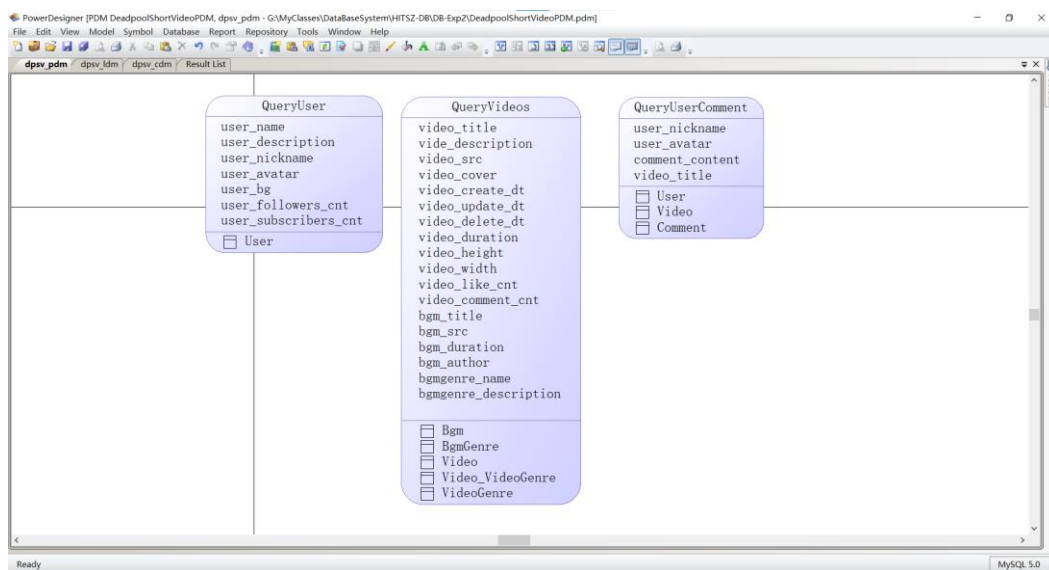


图 23 DPSV 中的视图

4、 触发器（选做）

触发器的目标是：当一个表中的元素改变，则应该通知另一个表，使其相关值亦发生改变。在 DPSV 中，我们有相当多的字段是 xxx_cnt，例如：video_like_cnt，当我们点赞一个视频的时候，video_like_cnt 便应该自增 1。事实上，点赞操作是对 Thumb 表进行操作的，因此，每当 Thumb 表插入一条记录后，我们便应该使 Video 表中对应 Video 的 video_like_cnt 字段自增 1。由此，我们可以构建如下触发器：

```
delimiter
/*=====*/
```

```

/* 点赞某个Video 后, 该Video 的点赞数++ */
/*=====*/
create trigger thumbTrigger
after insert on thumb
for each row
begin
    update Video
    set Video.video_like_cnt = Video.video_like_cnt + NEW.thumb_cn
t
    where Video.video_id = NEW.video_id;
end;

```

此外, user_followers_cnt 等字段也可设置类似的触发器。DPSV 的所有触发器代码如下所示:

```

use dpsv;

/*
    来自: https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html

    it is necessary to redefine the mysql statement delimiter
    so that you can use the ; statement delimiter within the
    trigger definition
*/

drop trigger if exists thumbTrigger;
drop trigger if exists commentTrigger;
drop trigger if exists replyTrigger;
drop trigger if exists followTrigger;
drop trigger if exists watchTrigger;

delimiter
/*=====*/
/* 点赞某个Video 后, 该Video 的点赞数++ */
/*=====*/
create trigger thumbTrigger
after insert on thumb
for each row
begin
    update Video
    set Video.video_like_cnt = Video.video_like_cnt + NEW.thumb_cn
t
    where Video.video_id = NEW.video_id;

```

```

end;

/*=====*/
/* 评论某个Video后, 该Video的评论数++ */
/*=====*/
create trigger commentTrigger
after insert on `Comment`
for each row
begin
    update Video
    set Video.video_comment_cnt = Video.video_comment_cnt + 1
    where Video.video_id = NEW.video_id;
end;

/*=====
==*/
/* 评论逻辑: 首先插入一条Comment, 然后插入Reply, 被评论的Comment的被
*/
/* 评论数
/*=====
==*/
create trigger replyTrigger
after insert on Reply
for each row
begin
    update Comment
    set `Comment`.comment_reply_cnt = `Comment`.comment_reply_cnt
+ 1
    where `Comment`.comment_id = NEW.comment_id;
end;

/*=====*/
/* 关注某人, 关注的人++, 被关注的人, 被关注数量
/*=====*/
create trigger followTrigger
after insert on Follow
for each row
begin
    update `User`
    set `User`.user_followers_cnt = `User`.user_followers_cnt + 1
    where `User`.user_id = NEW.user_id;

    update `User`

```



```

set `User`.user_subscribers_cnt = `User`.user_subscribers_cnt
+ 1
where `User`.user_id = NEW.follow_user_id;
end;
/*=====*/
/* 观看视频，视频被观看数++ */
/*=====*/
create trigger watchTrigger
after insert on Watch
for each row
begin
update Video
set Video.video_watched_cnt = Video.video_watched_cnt + 1
where Video.video_id = NEW.video_id;
end;
show triggers;

```

5、 事务（选做）

暂未实现。

2.1.5分析

这里将着重分析**评论与回复**的转换 ER-LDM-PDM 转换关系。因为该部分关系最为复杂。首先看 ER 图，如图 24 所示。可以清晰的看见，Video 与 Comment 之间是**一对多**的关系，代表一个 Video 可以有多条评论，而评论与评论之间则是**多对多**的关系，因为评论可以被另一条所回复。

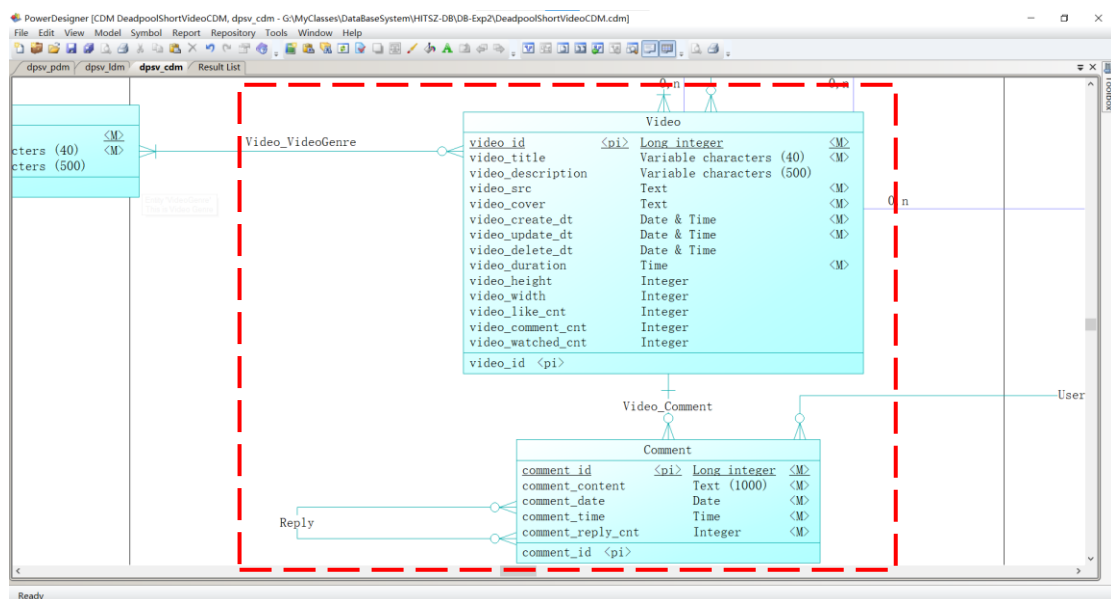


图 24 评论与回复 ER 图

转换成 LDM 后，结果如图 25 所示。

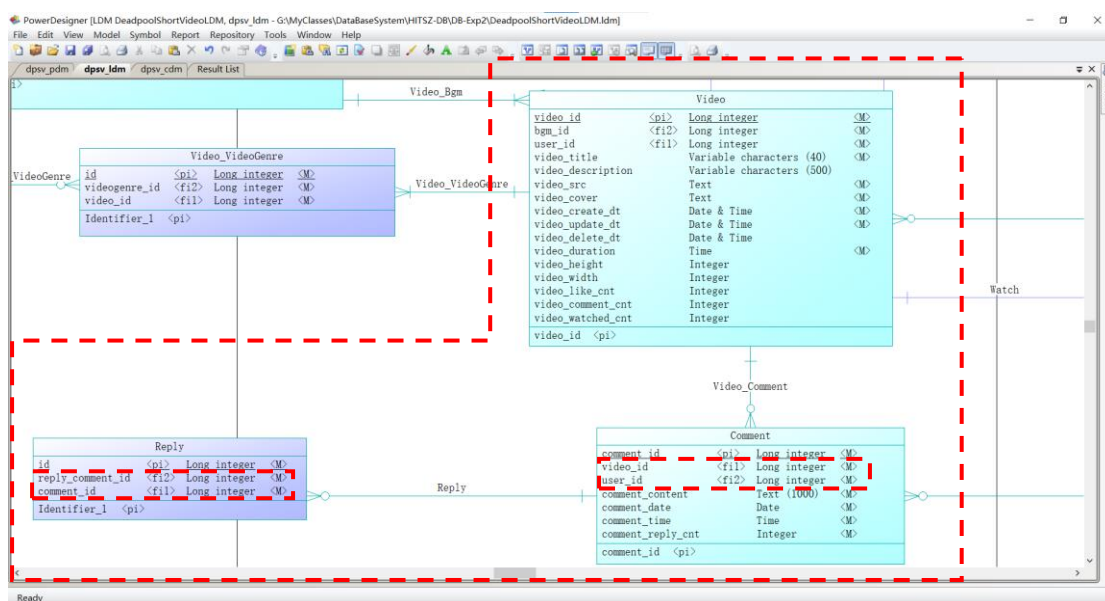


图 25 评论与回复 LDM 图

可以看见，评论到评论的回复关系被展开成了一个 Reply 实体，而评论与 Reply 之间是一对多的关系，代表一个评论可以有多条回复。事实上，Reply 实体建立了一个从 Comment 自身到自身的映射关系。此外，还可以看到 Comment 内多出了 video_id 以及 user_id 字段，这是因为不管是从 Video 到 Comment 或是 User 到 Comment 都是一对多的关系，这个一对多的映射信息只需被保存在多方即可。

最后是将 LDM 图转化为 PDM 图，结果如图 26 所示。事实上，PDM 图与 LDM 图的差别并不是特别大，PDM 所做的工作是将 LDM 转化为数据库中的字段，最后再通过选择相应数据库引擎，即可将 PDM 图转换为基于该引擎的代码实现了。

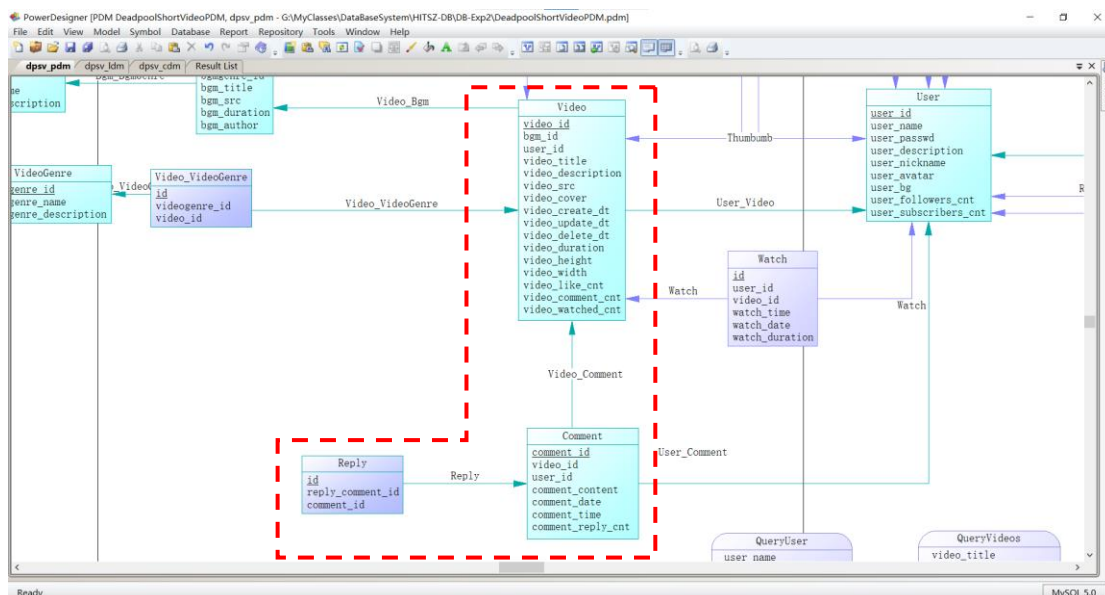


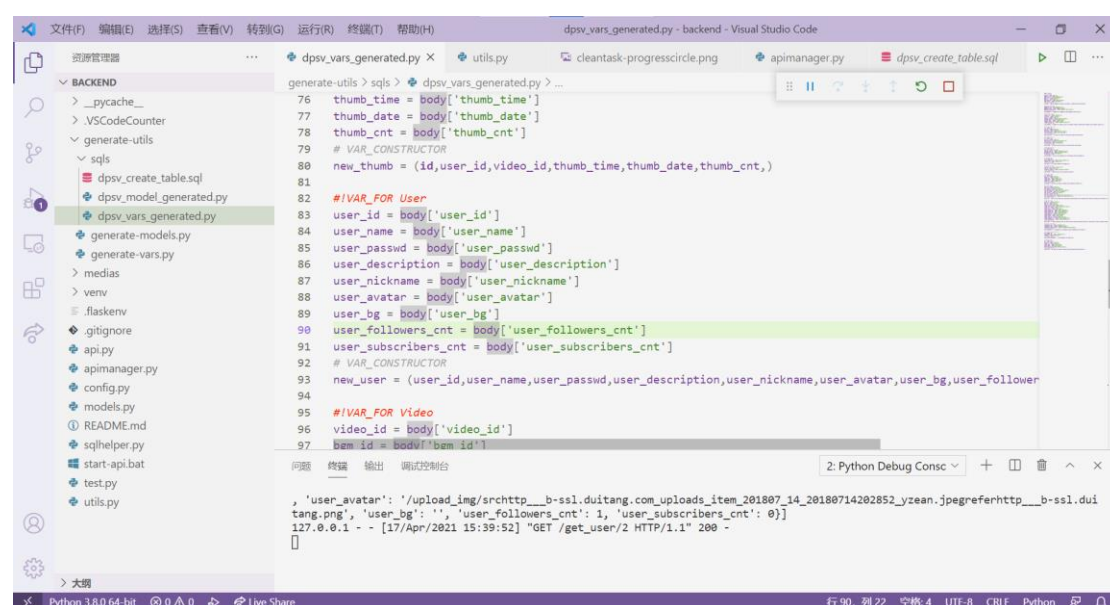
图 26 评论与回复 PDM 图

3 收获和反思

本次实验要求根据实验二中设计的数据库表结构来搭建一个全栈应用。事实上，本次实验的困难不在于 SQL 应该怎么写，而在于如何用**工程化**的方法去完成这样一个全栈应用。我们需要自行构建 API、自行构建访问函数、自行构建从前端到后端的整体架构。由于此前并未接触过 Flask 或 React，因此框架的学习也将占用一定的时间成本。

在项目的实现过程中，我对前端与后端都进行了较好的模块化，因此才能在如此短的开发周期内开发出一款基本符合要求又不失美观的产品。因此，**模块化的思想**我认为是非常重要的。

另一关键点在于体力工作的最小化。在项目的后端开发过程中，会写非常多重复的语句——例如编写数据模型、编写构造函数等，但这些语句都具有相似的模式，我们完全可以编写脚本来自动编写。图 27 展示了由脚本自动生成的 request 变量获取方式。然后我们仅需要做的便是从这里拷贝代码至需要的地方即可。



```
generate-utils > sqls > dpsv_vars_generated.py > ...
76 thumb_time = body['thumb_time']
77 thumb_date = body['thumb_date']
78 thumb_cnt = body['thumb_cnt']
79 # VAR_CONSTRUCTOR
80 new_thumb = (id,user_id,video_id,thumb_time,thumb_date,thumb_cnt,)
81
82 #!VAR_FOR User
83 user_id = body['user_id']
84 user_name = body['user_name']
85 user_passwd = body['user_passwd']
86 user_description = body['user_description']
87 user_nickname = body['user_nickname']
88 user_avatar = body['user_avatar']
89 user_bg = body['user_bg']
90 user_followers_cnt = body['user_followers_cnt']
91 user_subscribers_cnt = body['user_subscribers_cnt']
92 # VAR_CONSTRUCTOR
93 new_user = (user_id,user_name,user_passwd,user_description,user_nickname,user_avatar,user_bg,user_followers
94
95 #!VAR_FOR Video
96 video_id = body['video_id']
97 new_id = body['new_id']

, 'user_avatar': '/upload_img/srchttp__b-ssl.duitang.com/uploads_item_201807_14_20180714202852_yzean.jpegreferhttp__b-ssl.dui
tang.png', 'user_bg': '', 'user_followers_cnt': 1, 'user_subscribers_cnt': 0}]
127.0.0.1 - - [17/Apr/2021 15:39:52] "GET /get_user/2 HTTP/1.1" 200 -
```

图 27 自动生成的函数变量

最后，感谢老师能够提供这么一个全栈开发的机会，使得我们能够充分理解前后端的交接、后端与数据库交接的方式，期待本实验课程越办越好。