

（深圳）

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

# 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

# 实验过程

## 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：

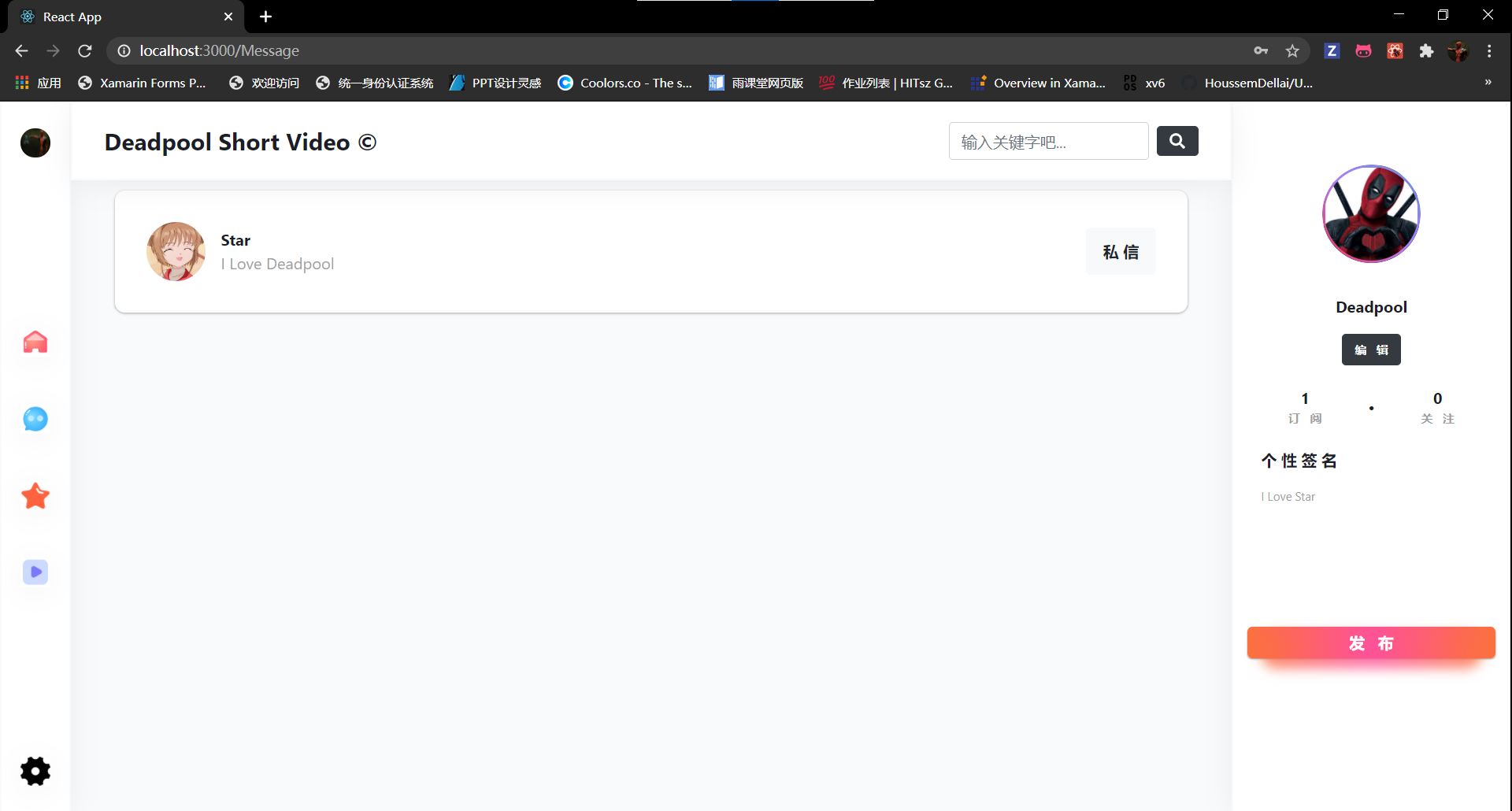


图1 用户管理模块

* **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。

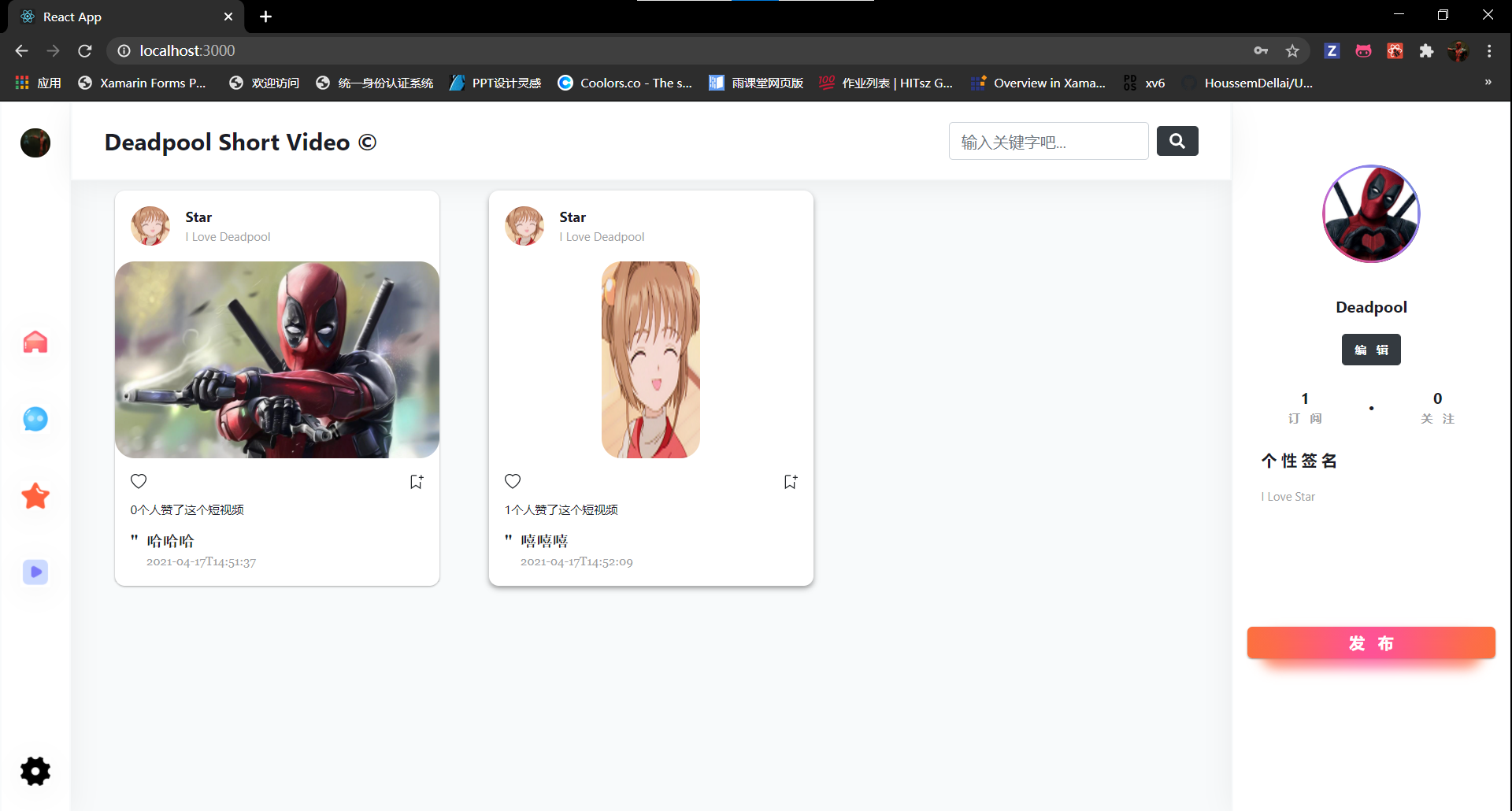


图2 DPSV主页

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。

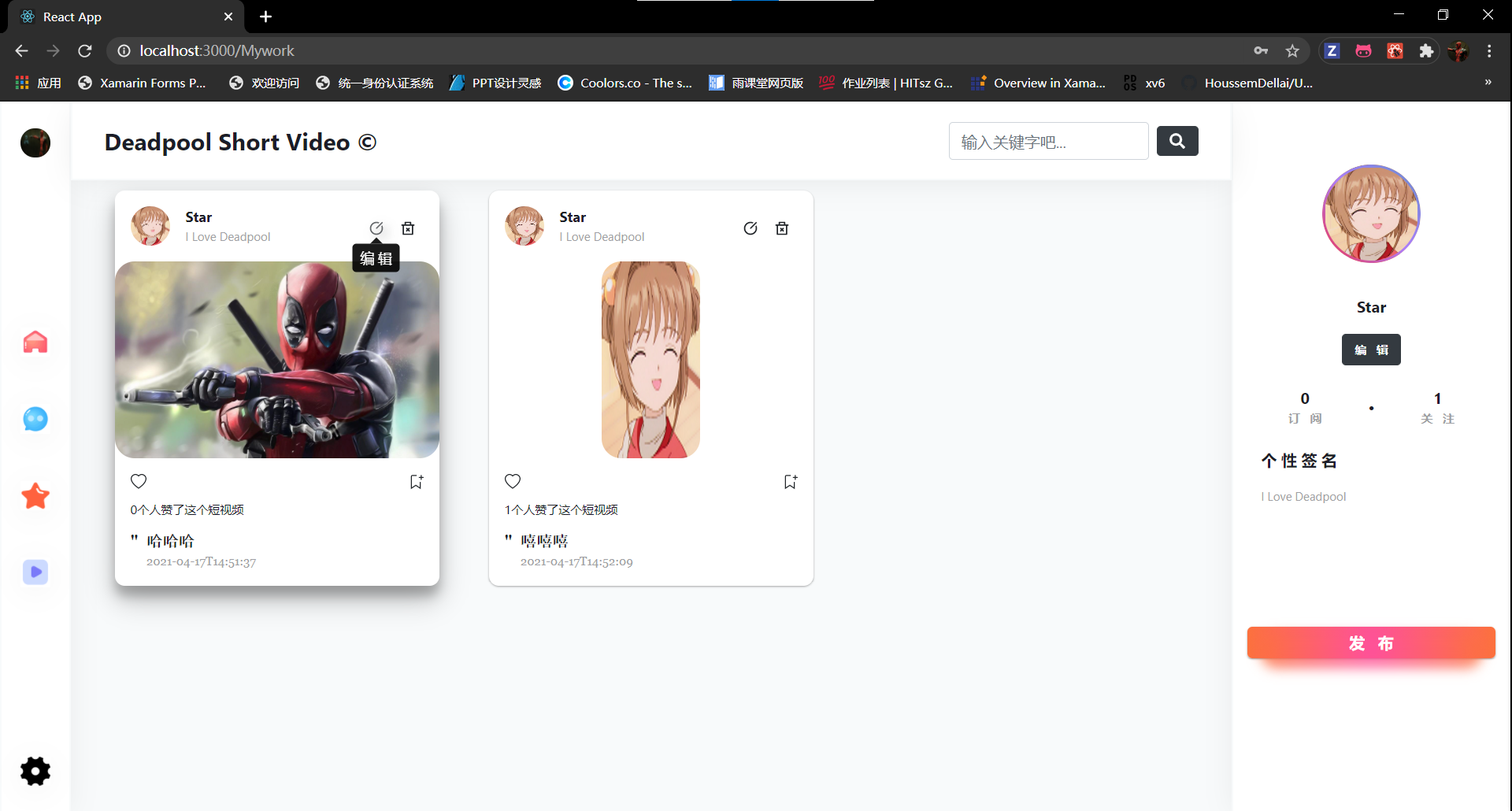


图3 DPSV个人作品页面

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。每个视频都必须要包含**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。

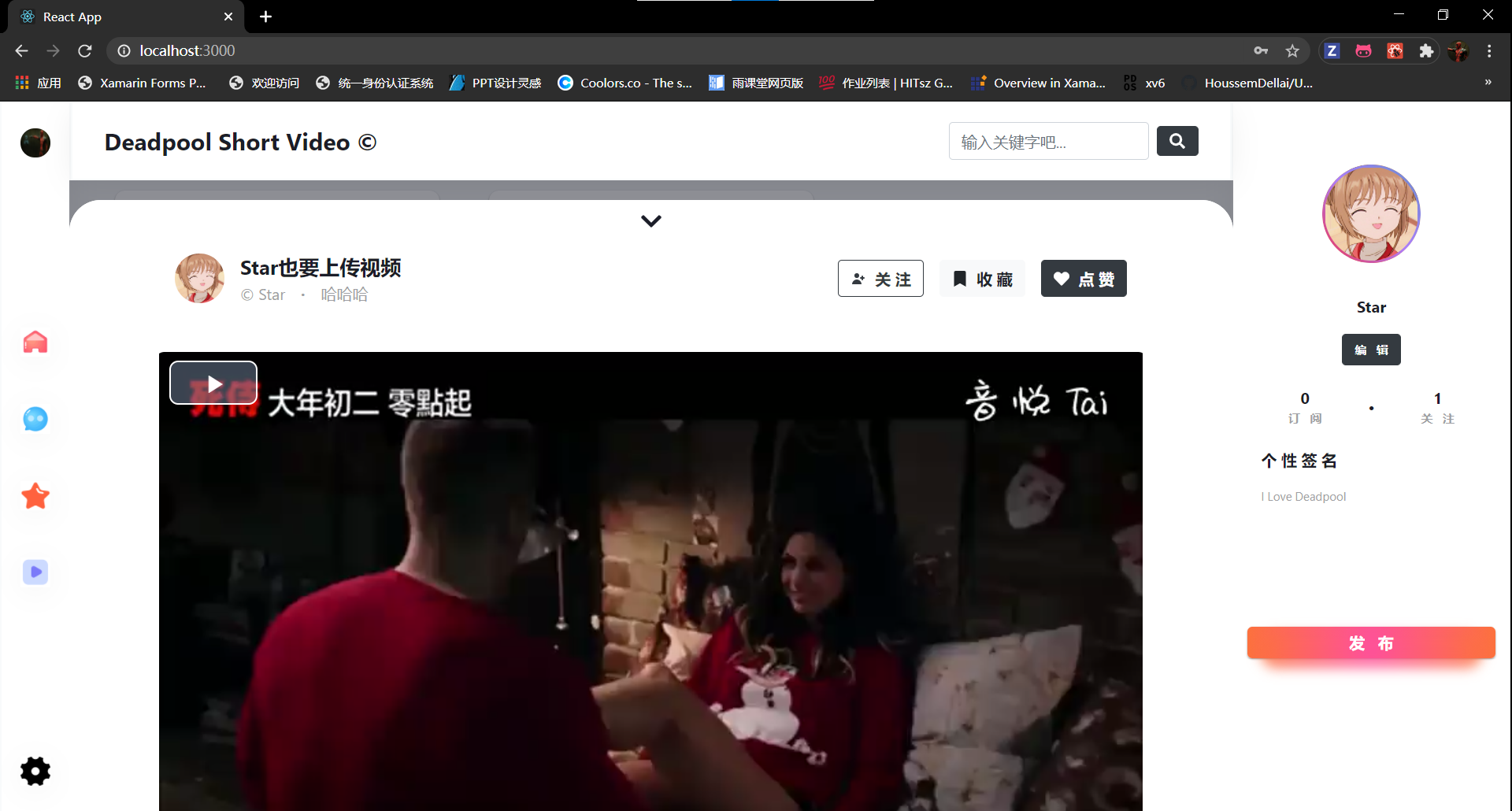


图5 DPSV视频详情页

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。

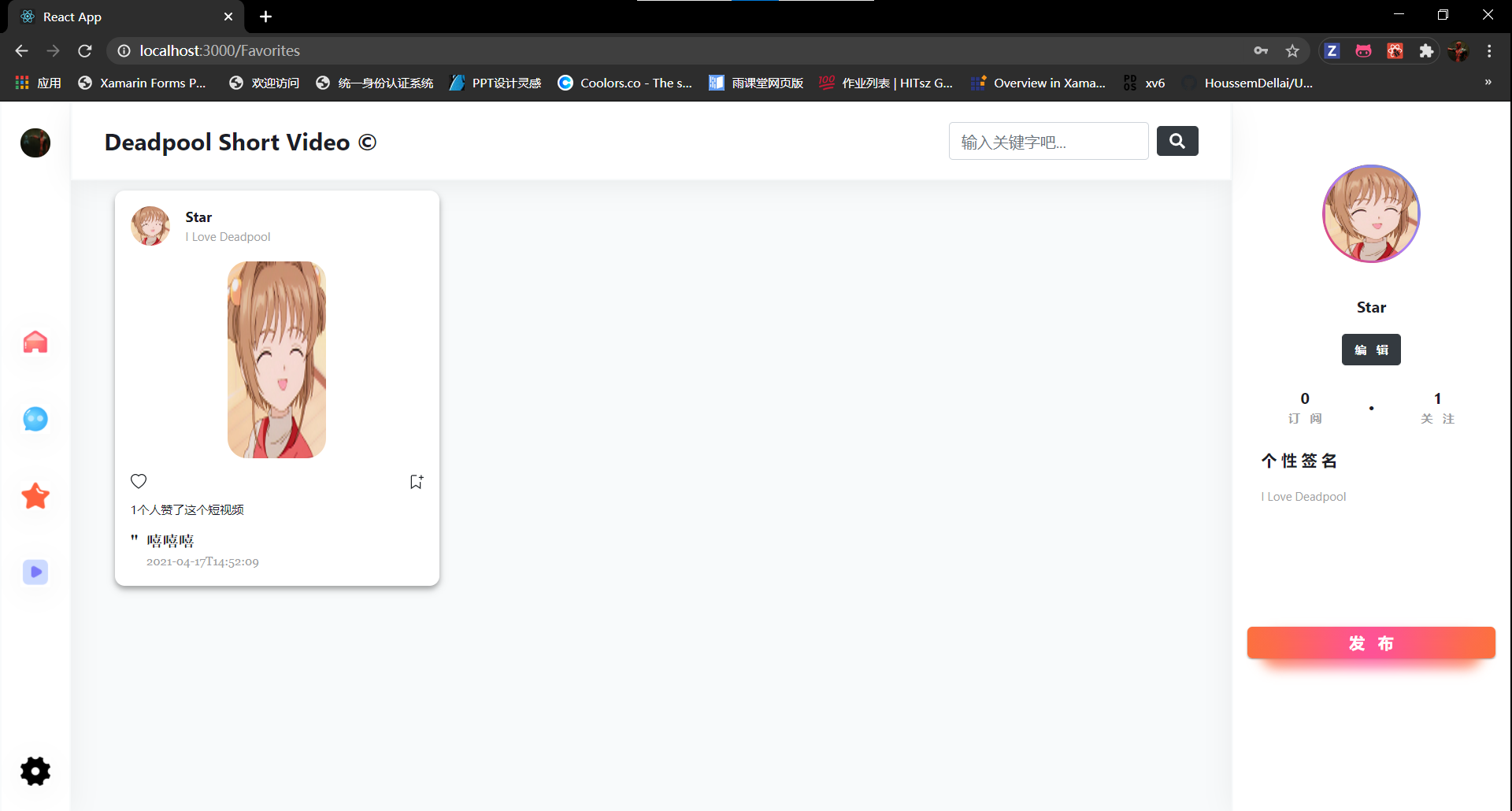
13.12 厘米

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。

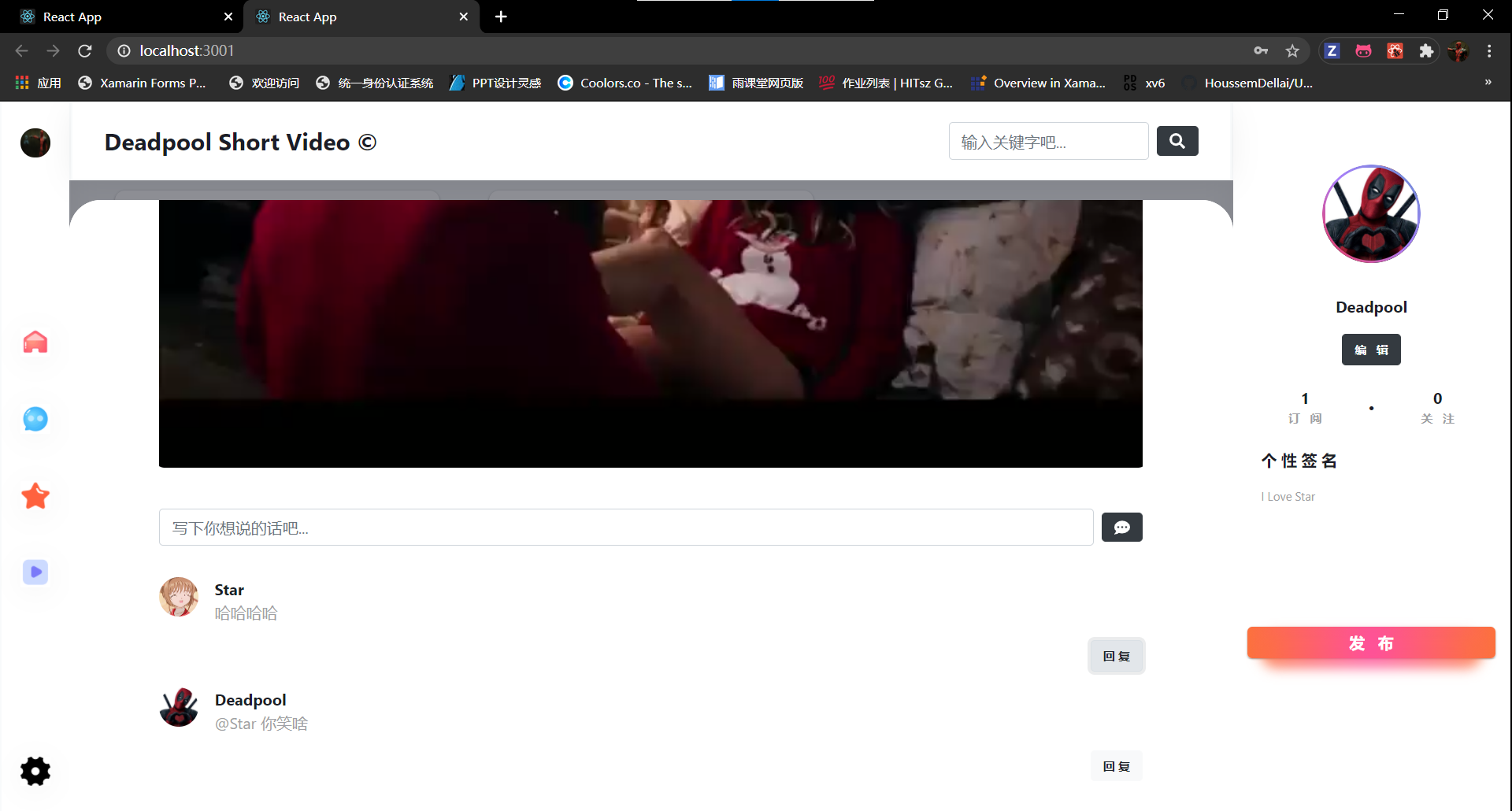


图7 DPSV评论及回复

* **\*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。

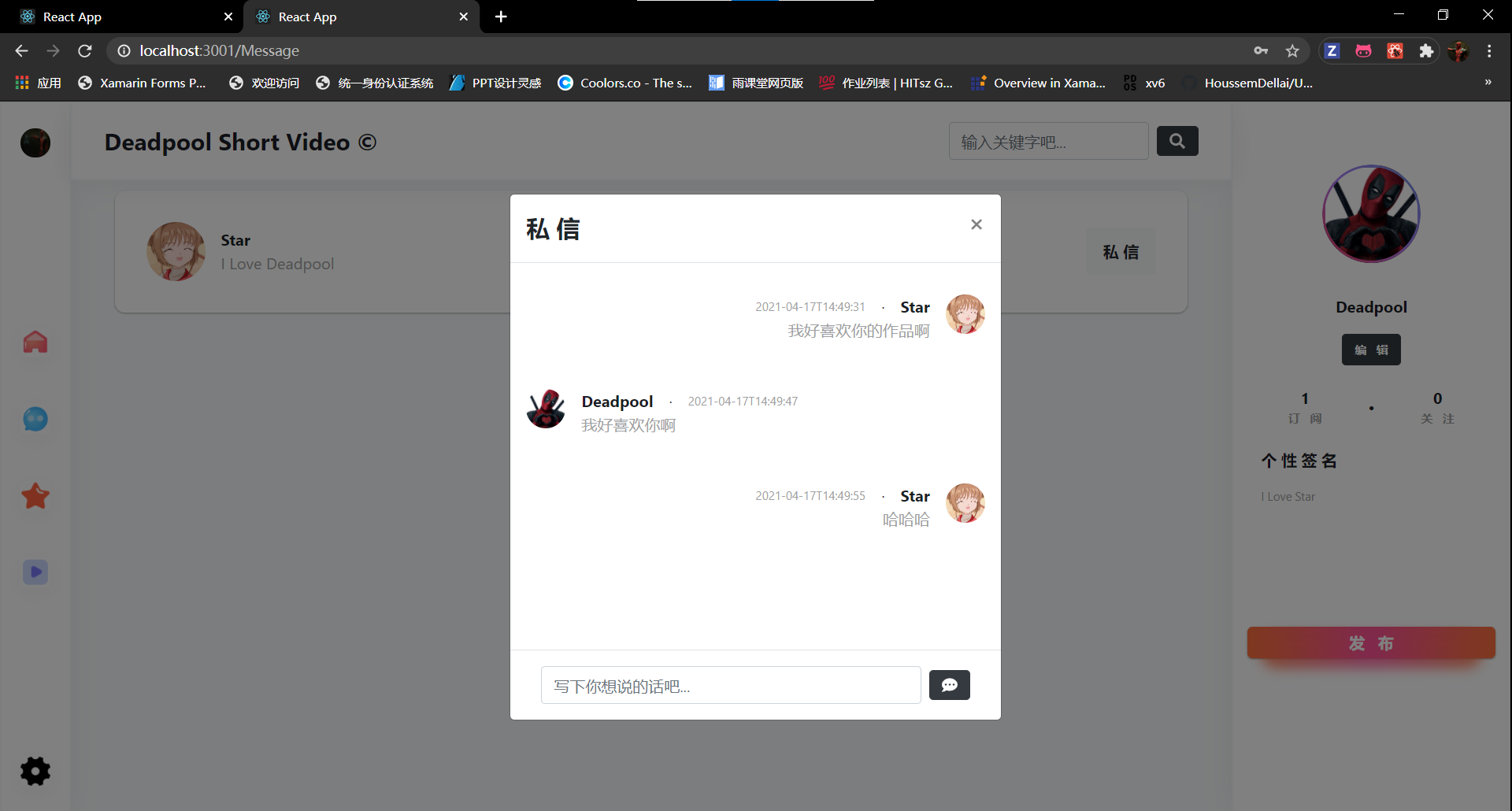


图8 DPSV私信页面

* **\*搜索模块**

无论在任何页面，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。

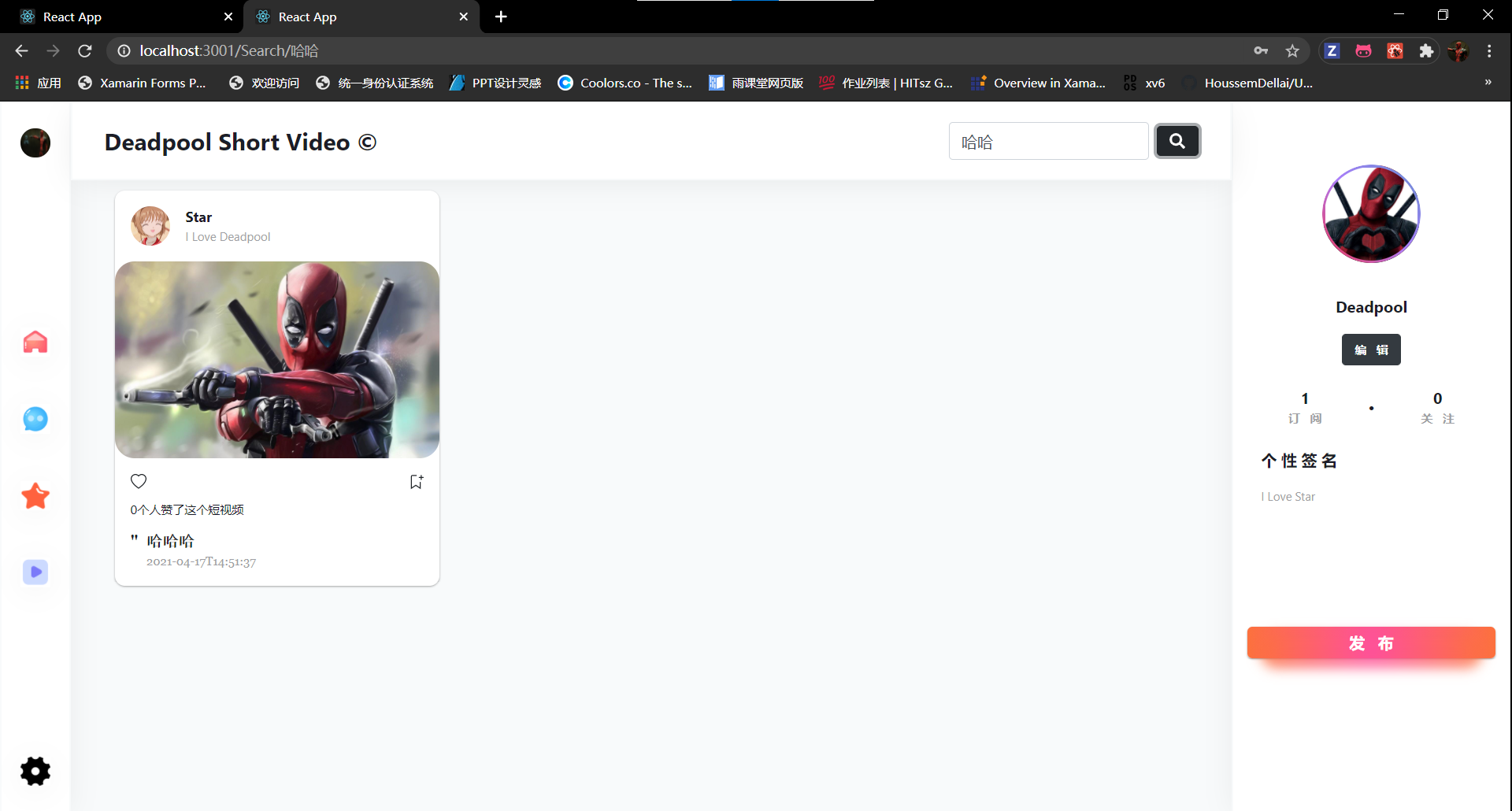


图9 DPSV搜索展示

## 数据库设计

### 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；

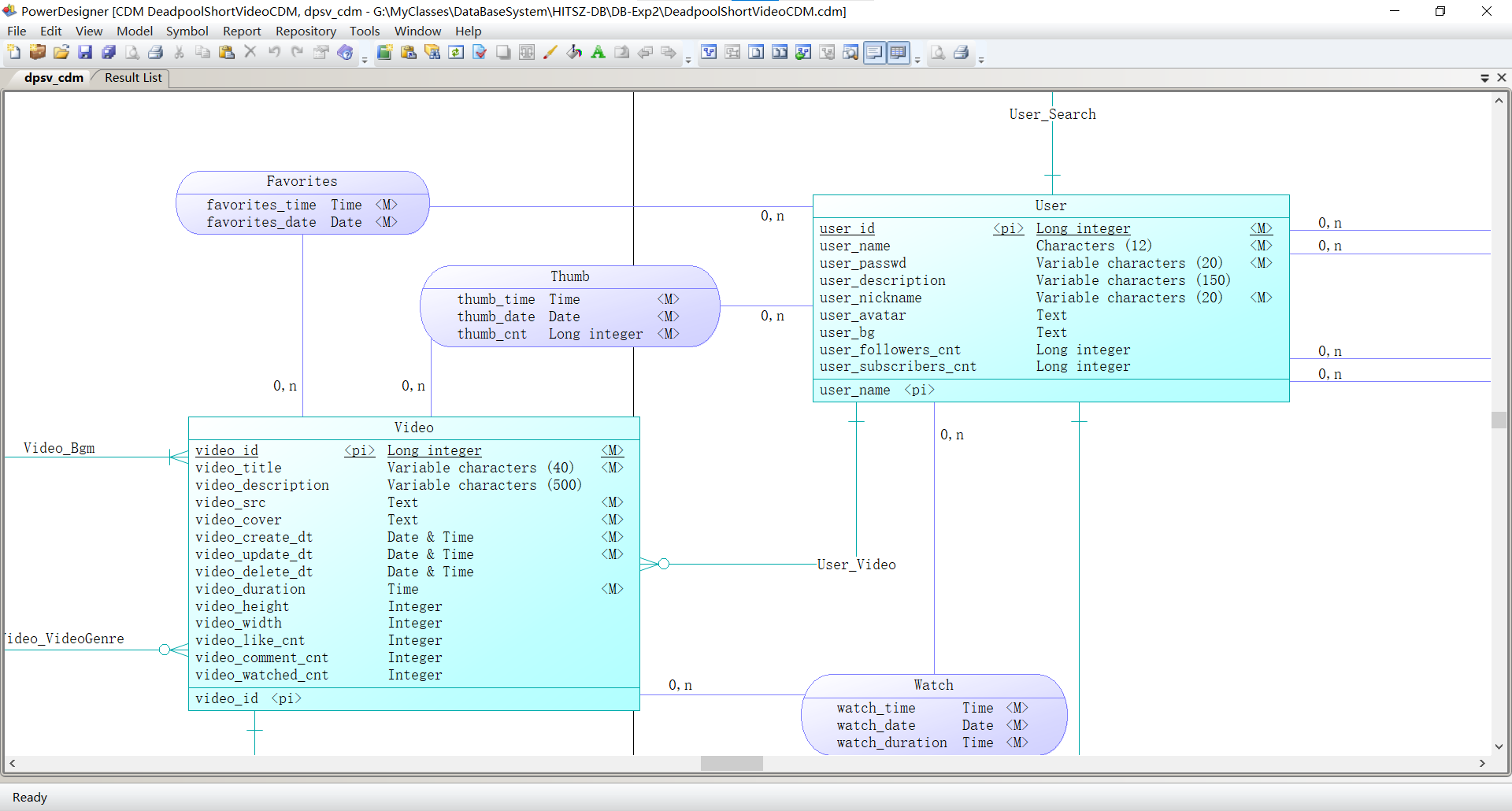


图10 用户与视频主干ER网络结构

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。

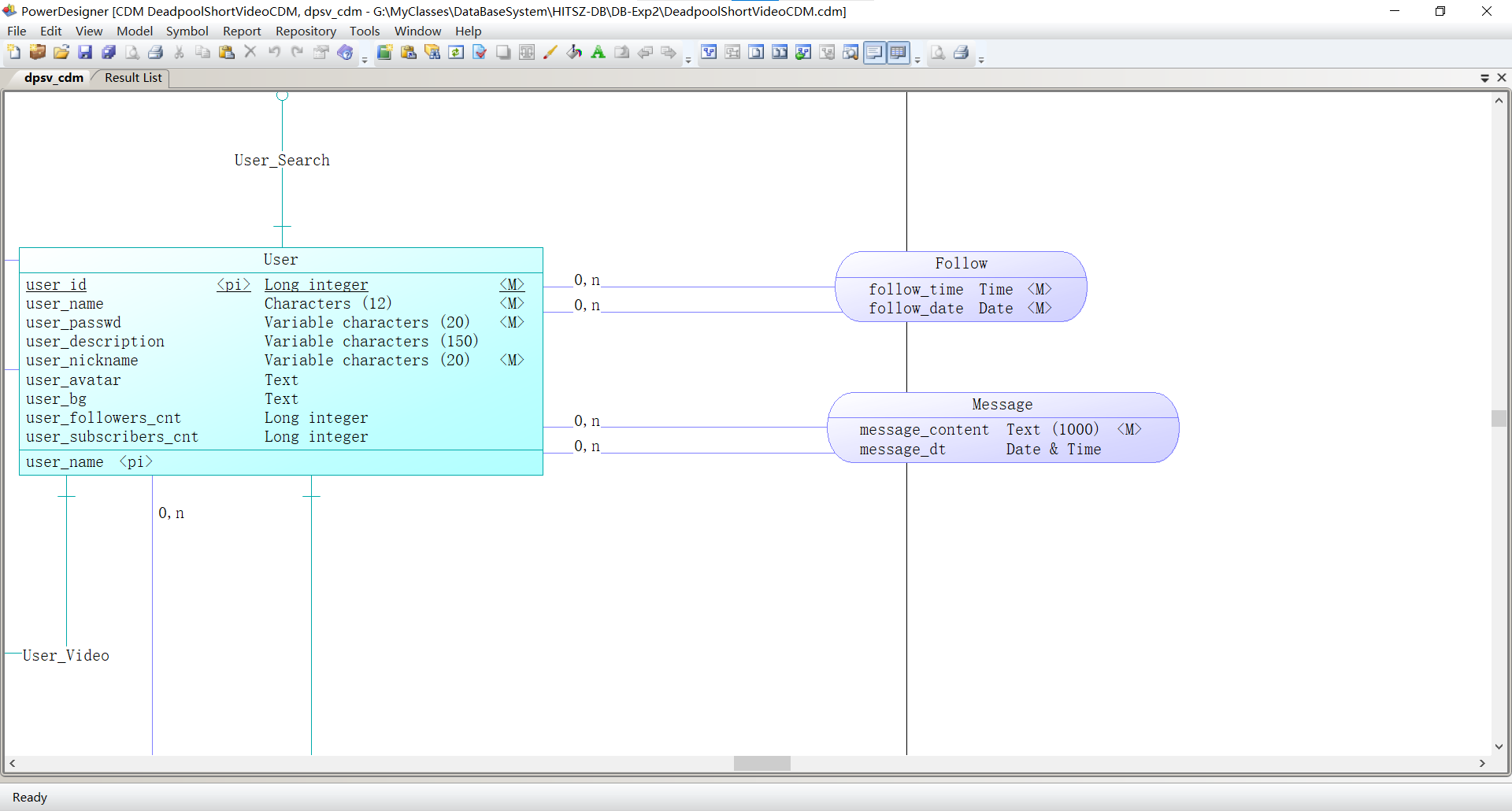


图11 用户与用户间的关注、私信关系

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。

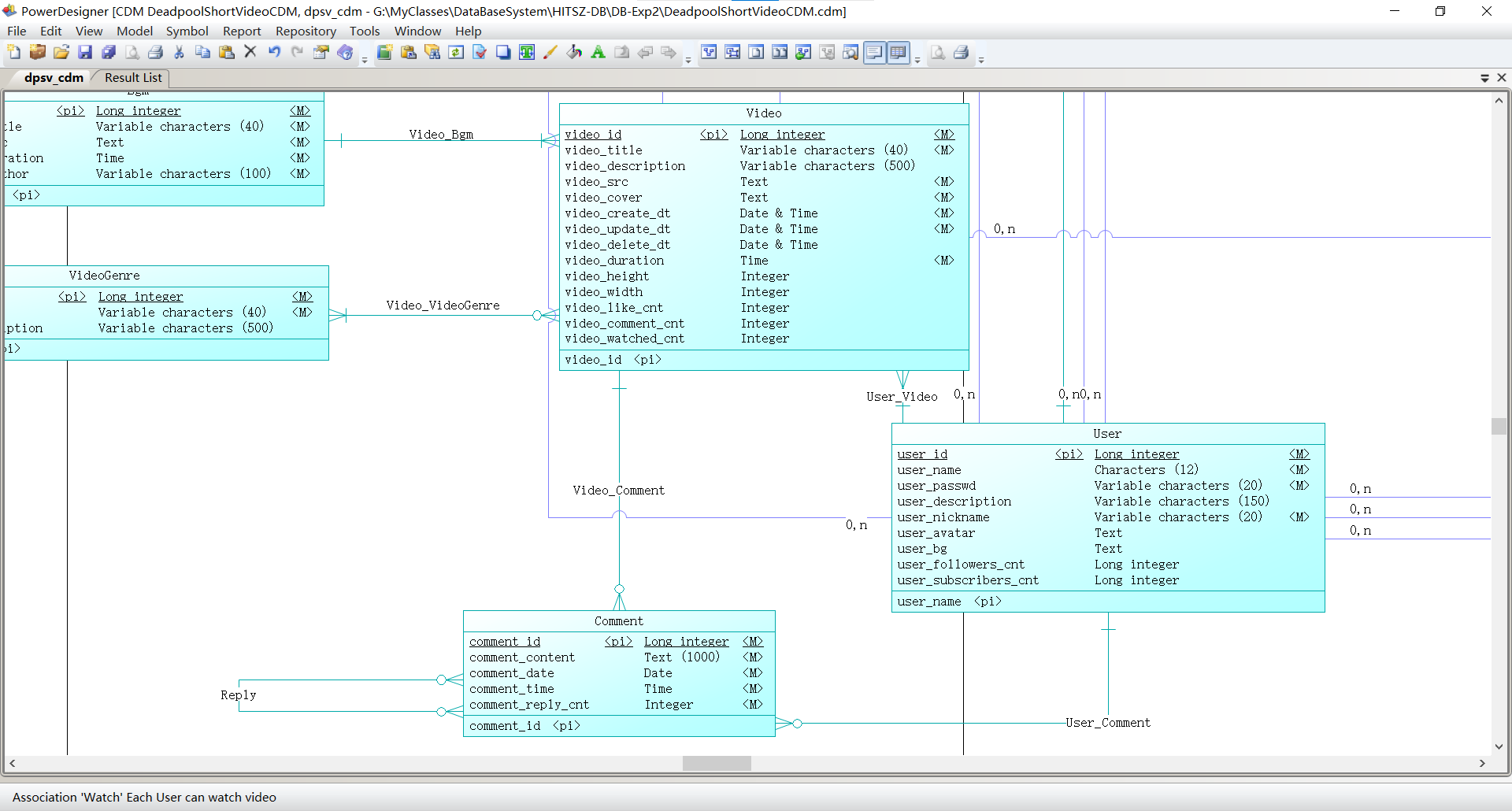


图12 用户、视频、评论间的关系

### 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。

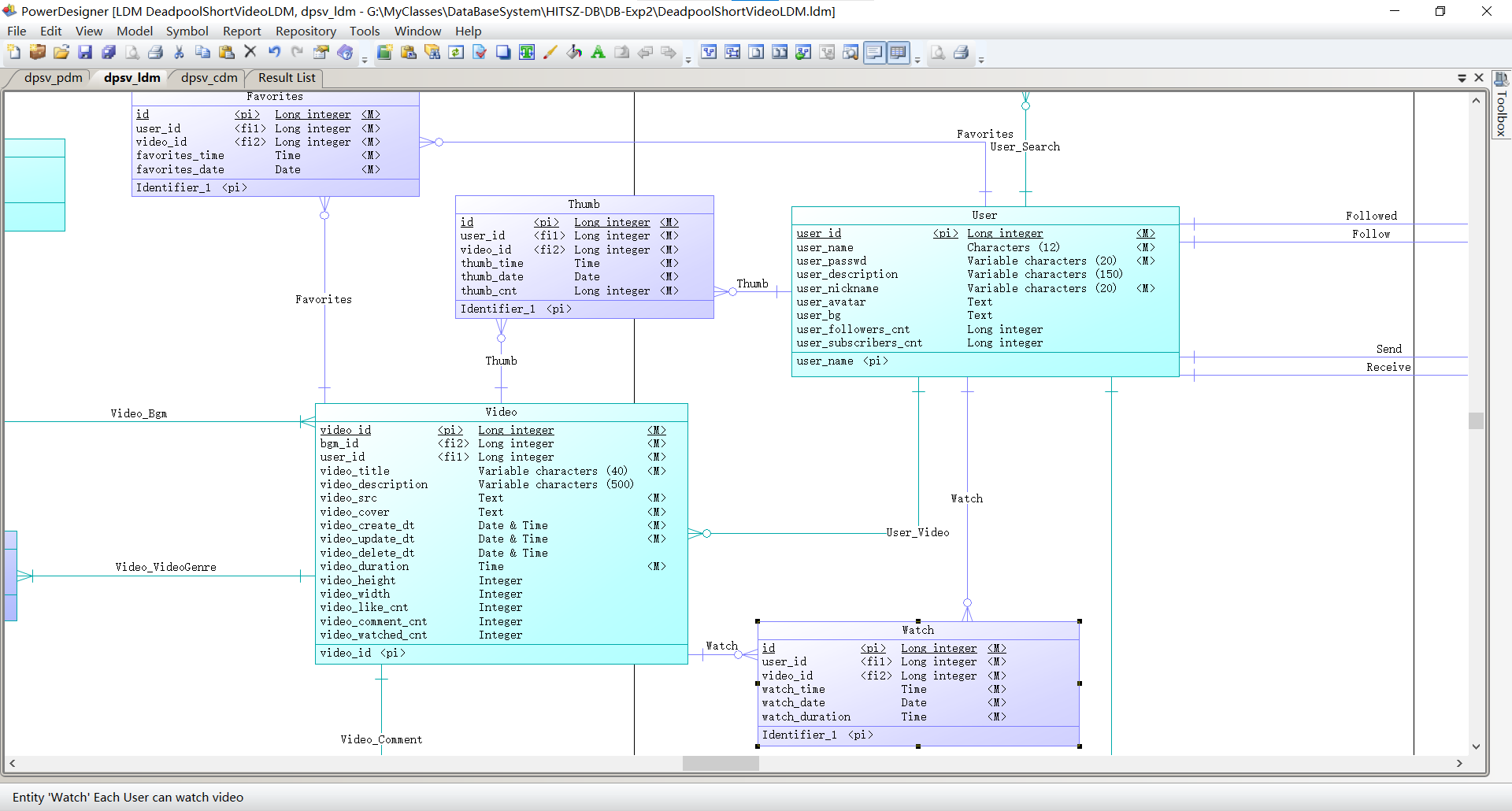


图13 用户与视频部分LDM图

其次是逻辑描述用户与用户间的关系，如图14所示。

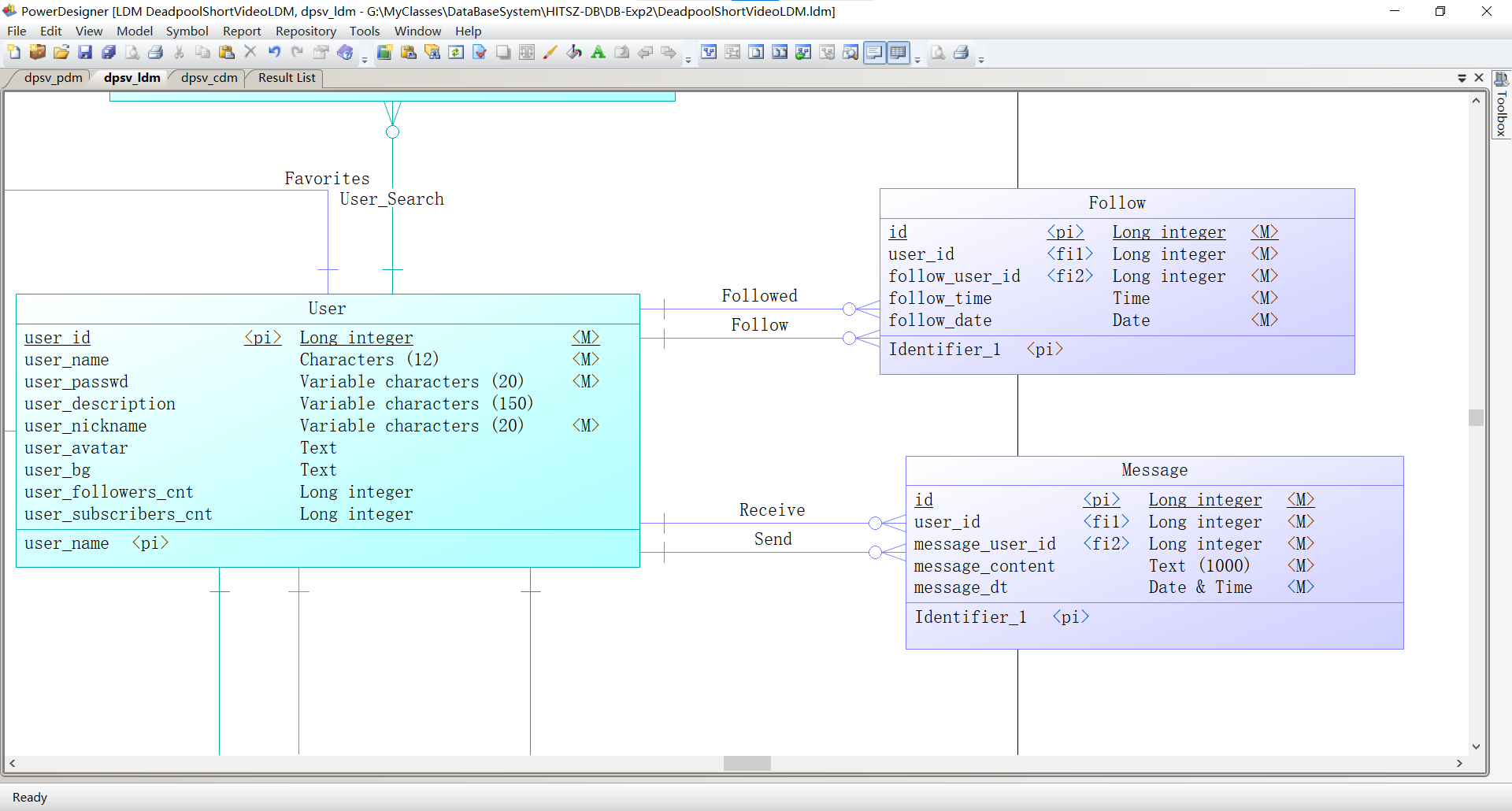


图14 用户与用户的关注、私信关系

最后描述用户、视频、评论间的关系。如图15所示

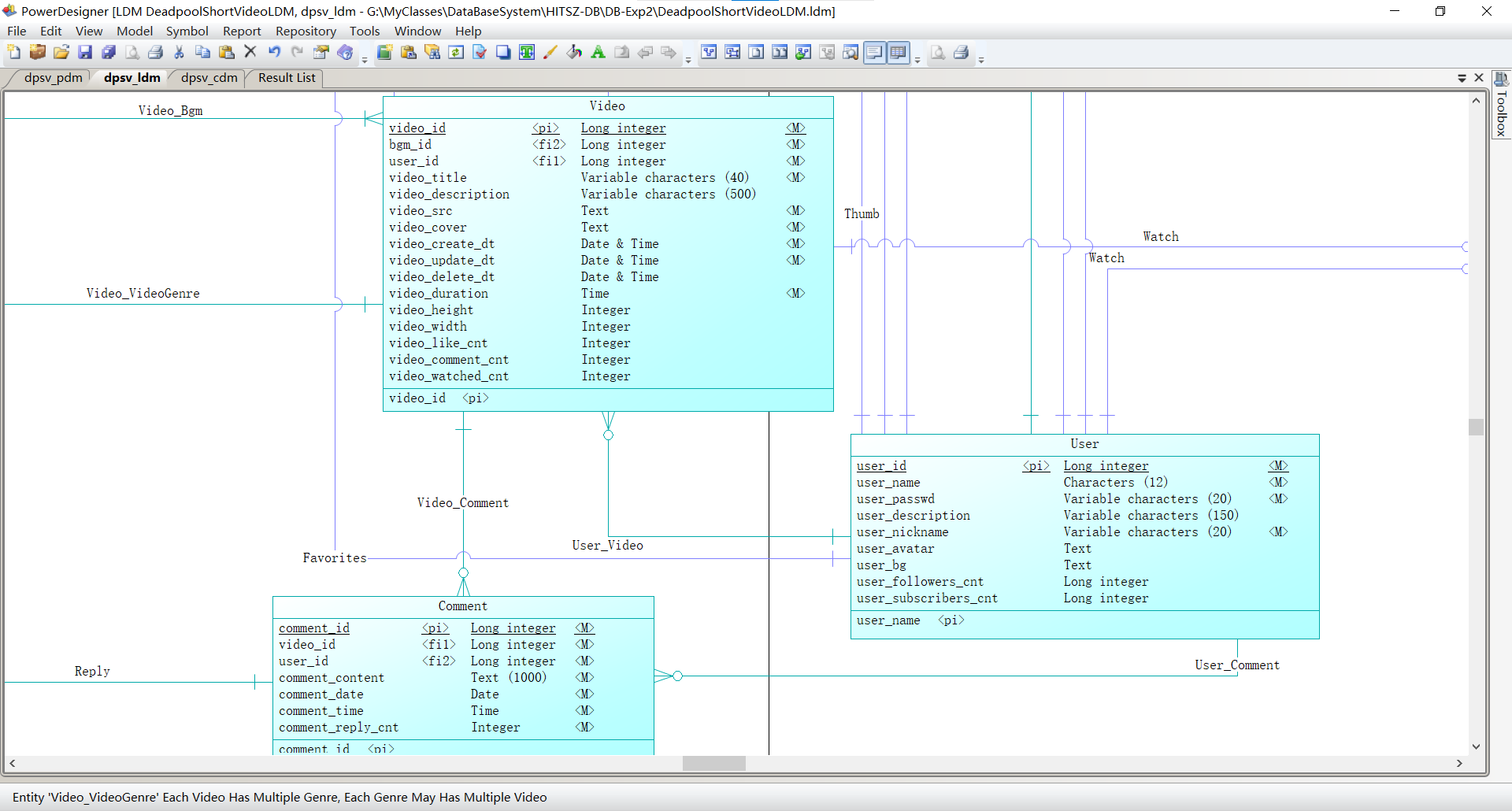


图15 用户、视频、评论间的关系

### PDM图

相应的PDM见图16、17、18所示，在此不做赘述。



图16 用户与视频之间的物理表设计

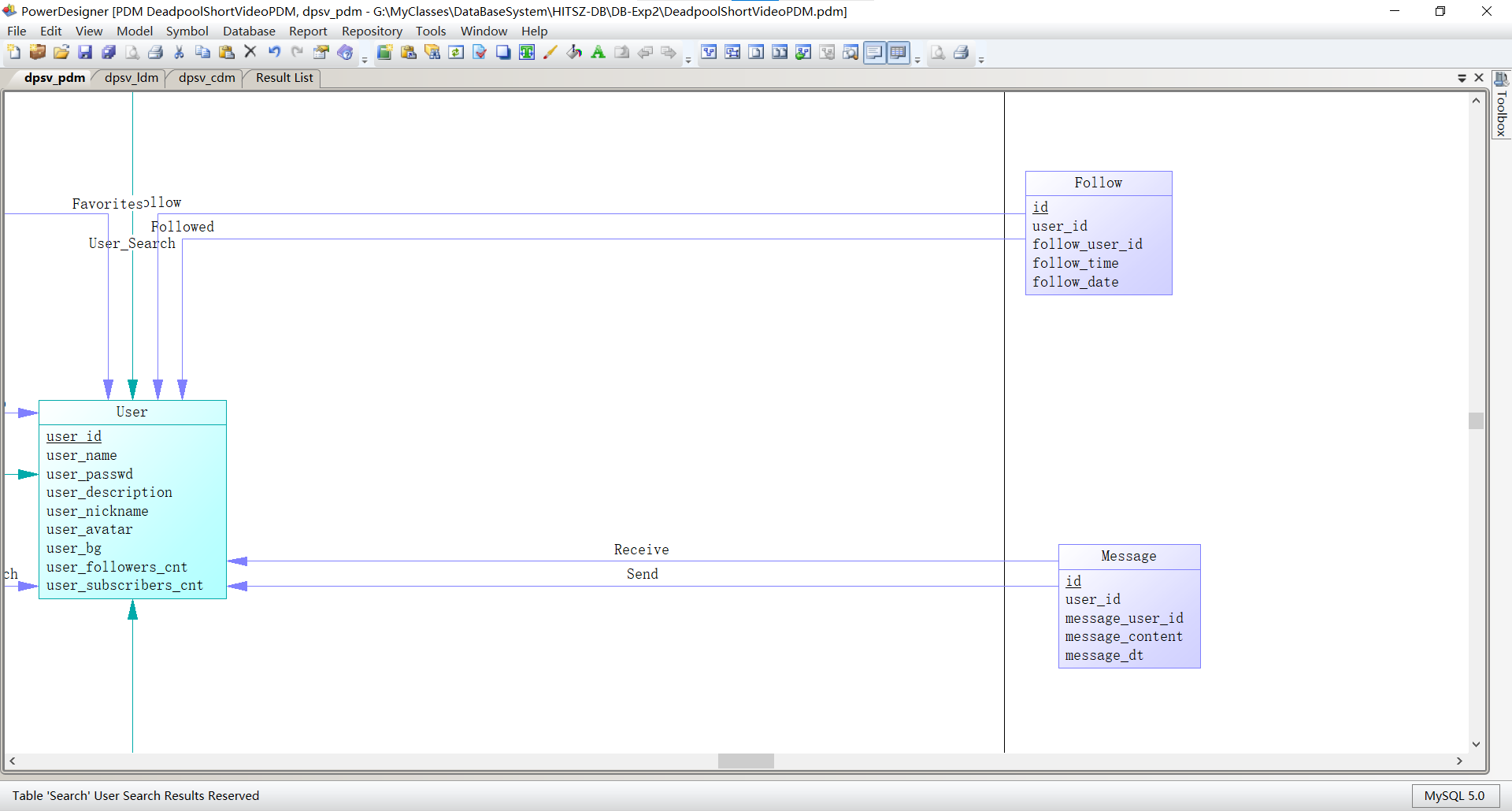


图17 用户与用户之间的物理表设计

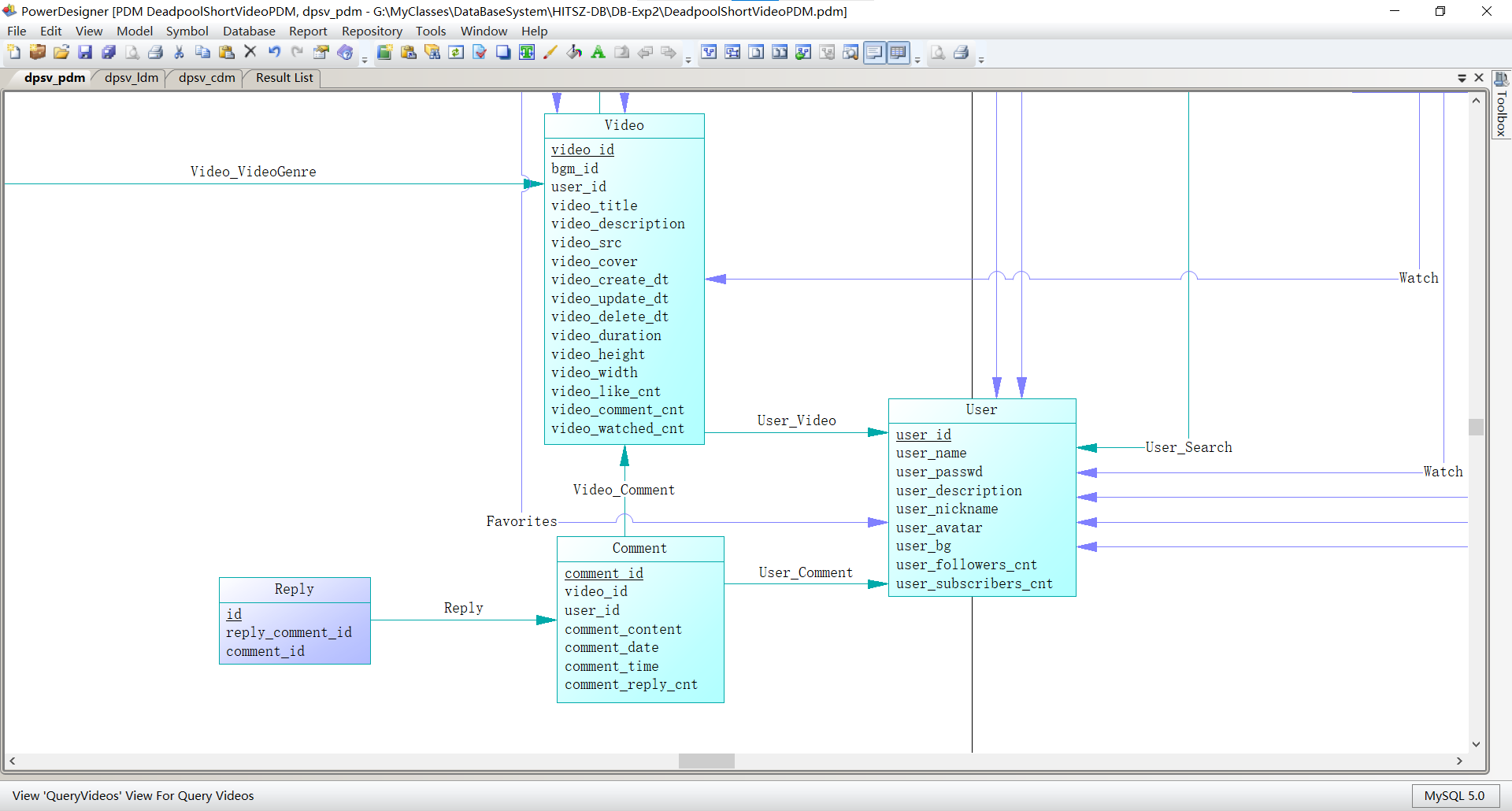


图18 用户、视频与评论间的物理表设计

### 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。

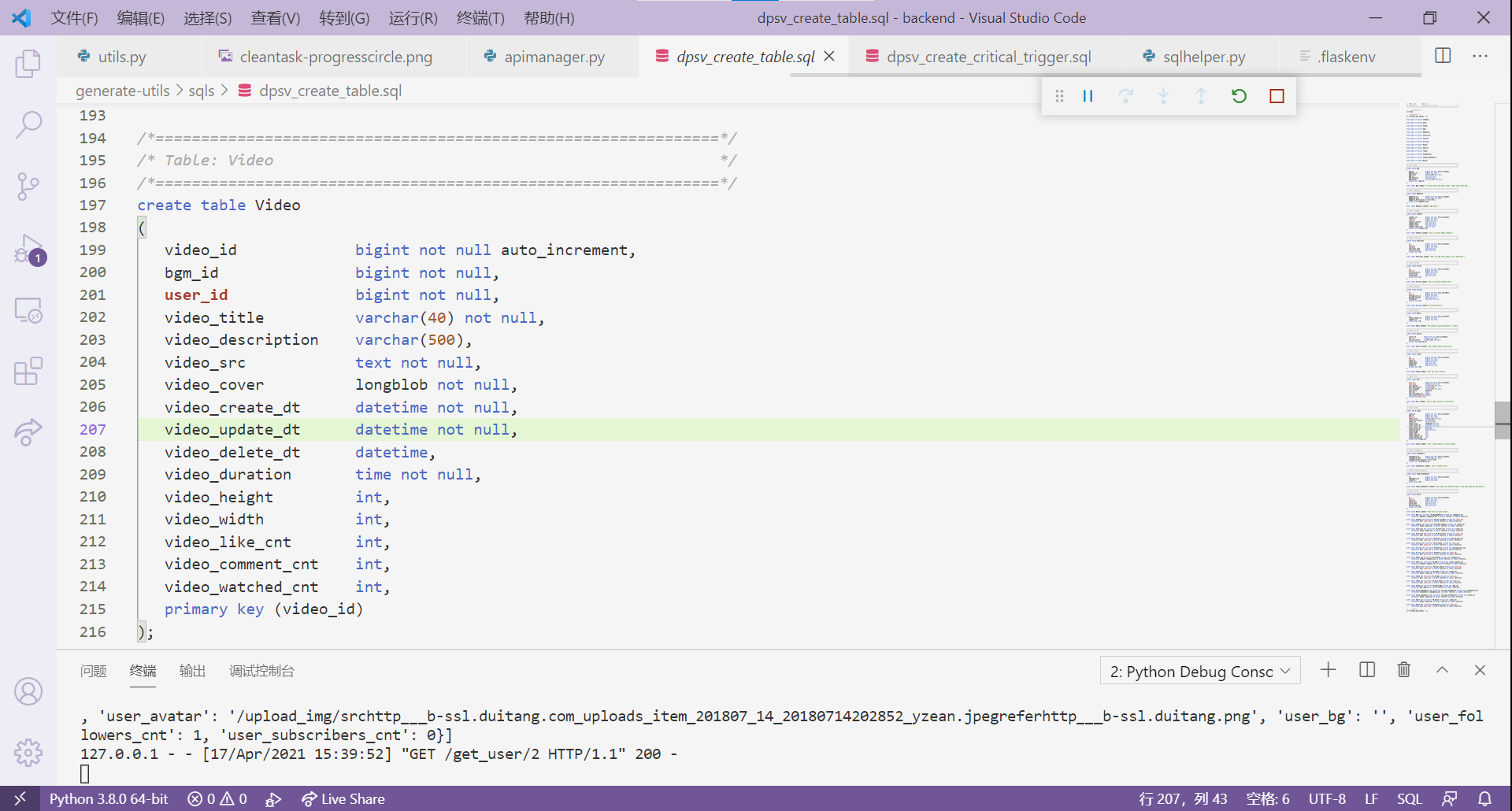


图19 Video表结构

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。

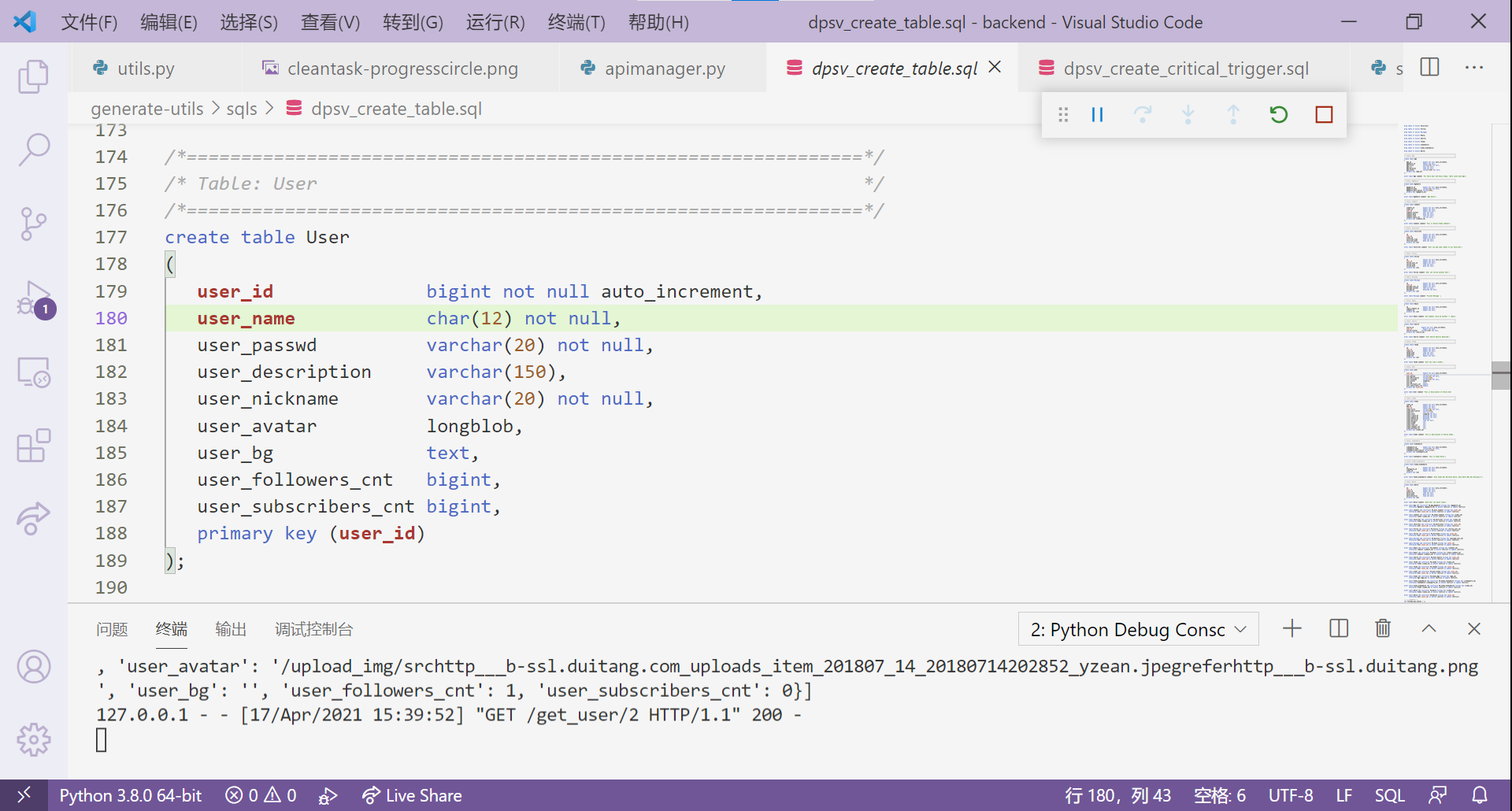


图20 User表结构

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。

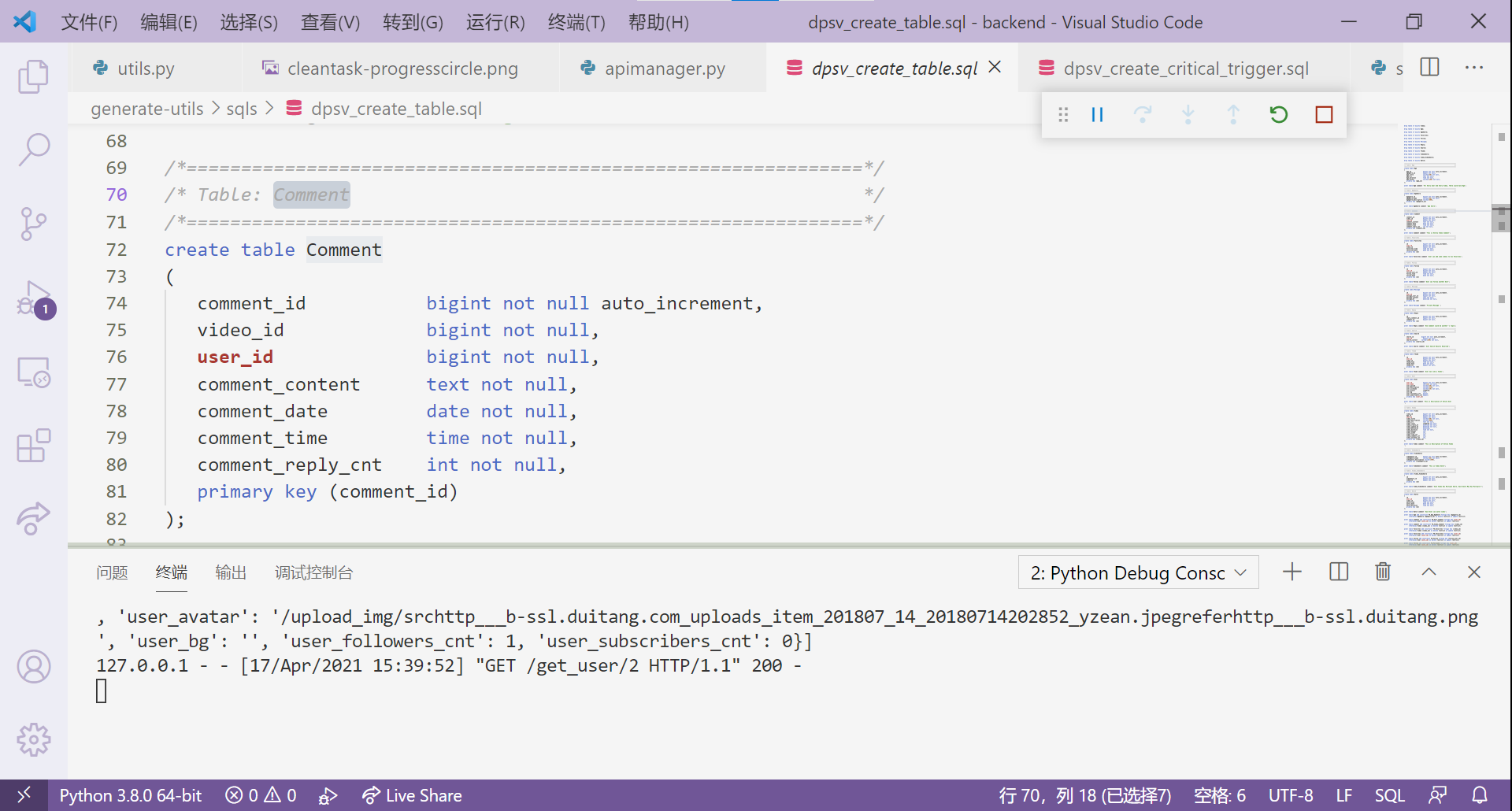


图21 Comment表结构

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 |  |

1. **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。

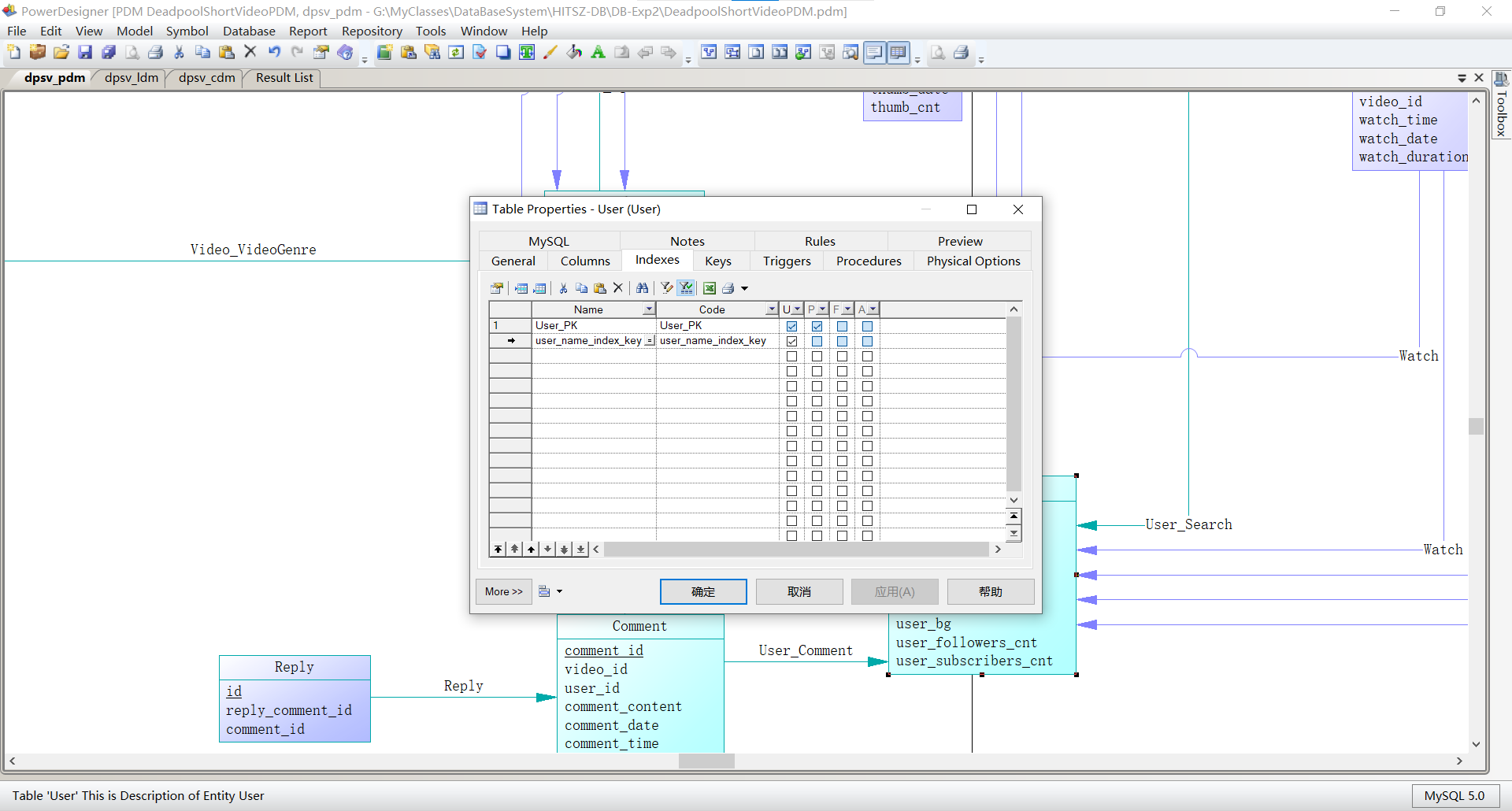


图22 user\_name的非主索引

1. **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。

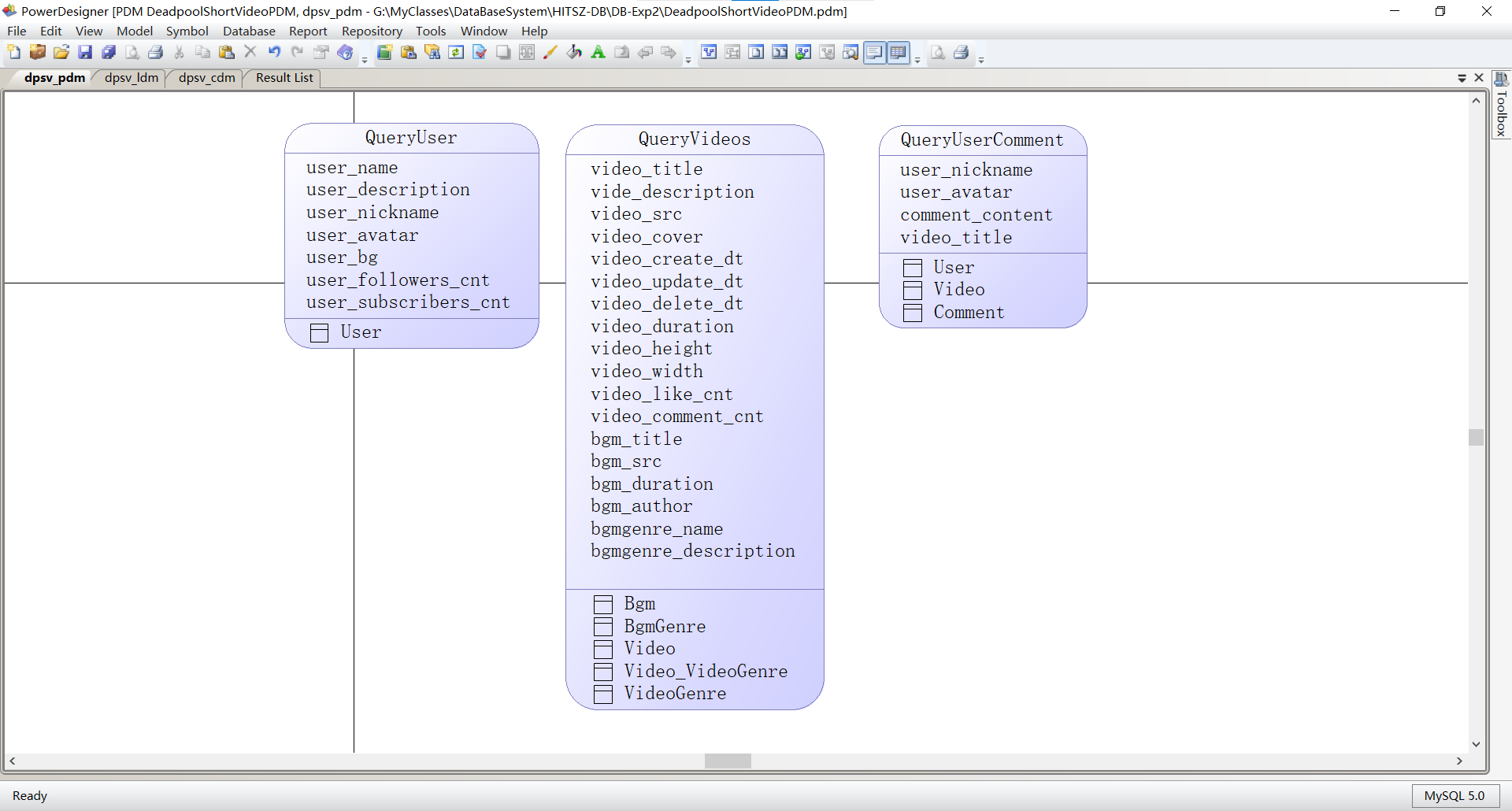
**

图23 DPSV中的视图

1. **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\_cnt      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\_cnt      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; |

1. **事务（选做）**

暂未实现。

### 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。可以清除的看见，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。

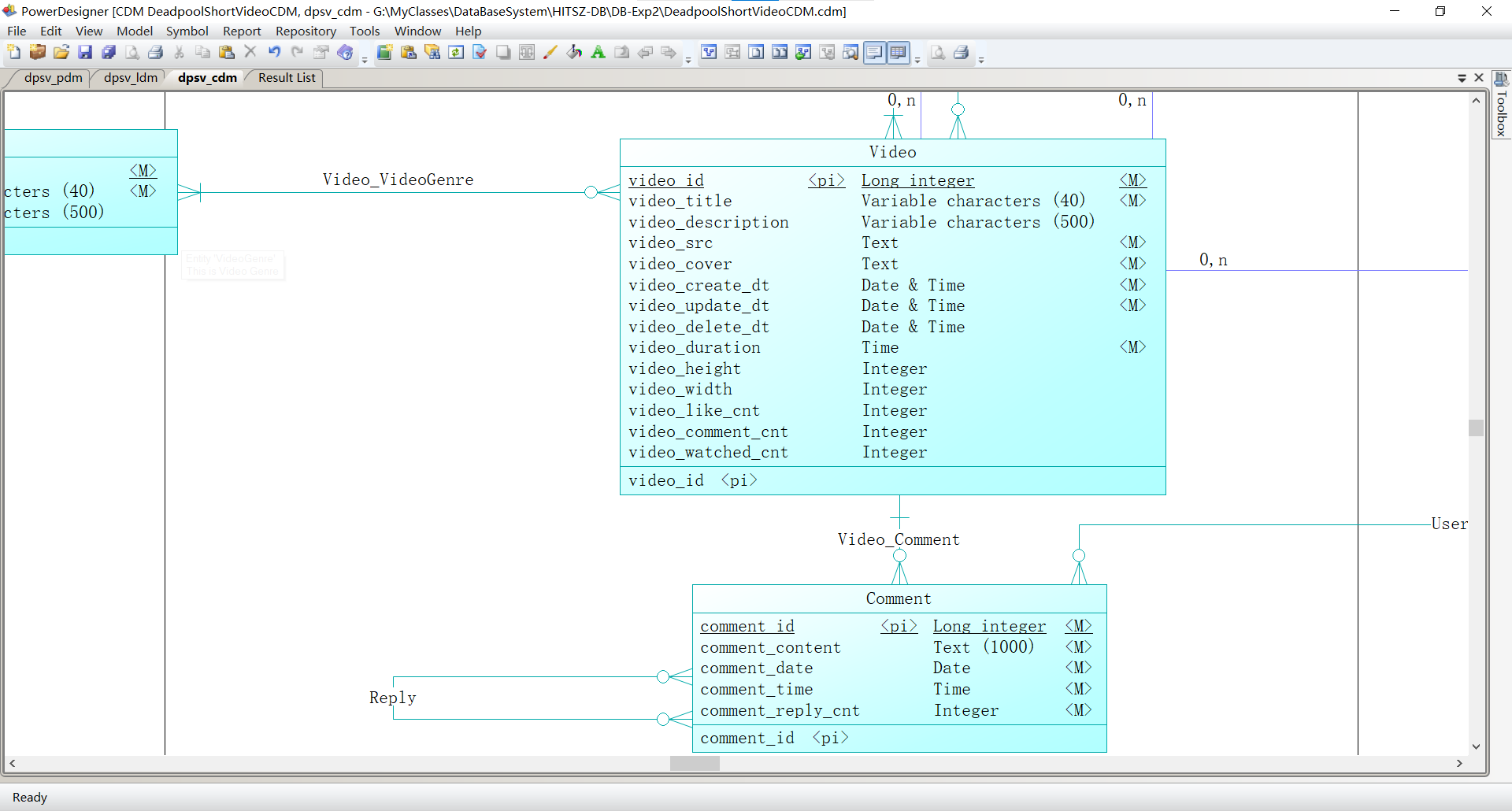


图24 评论与回复ER图

转换成LDM后，结果如图25所示。

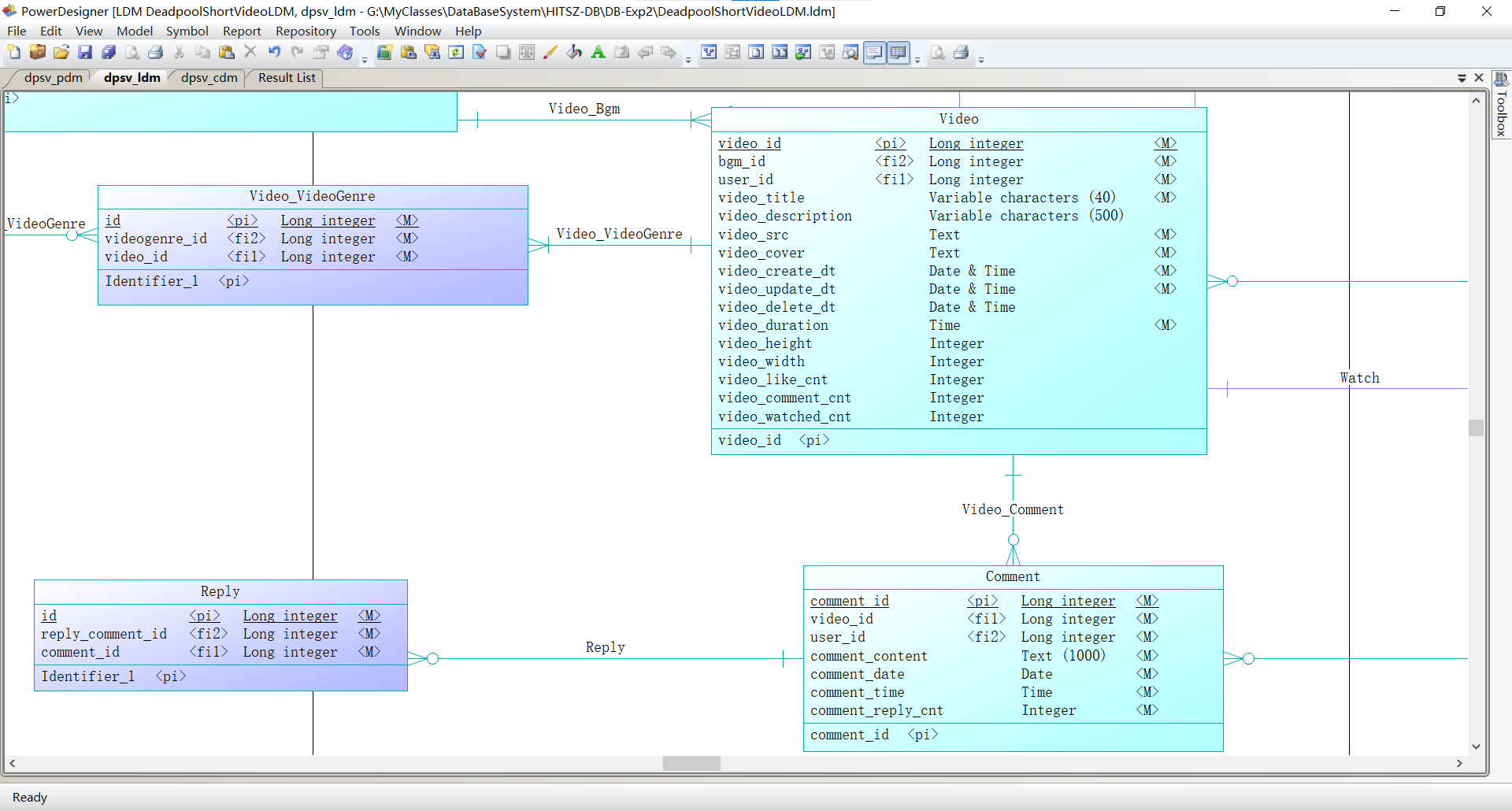


图25 评论与回复LDM图

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。

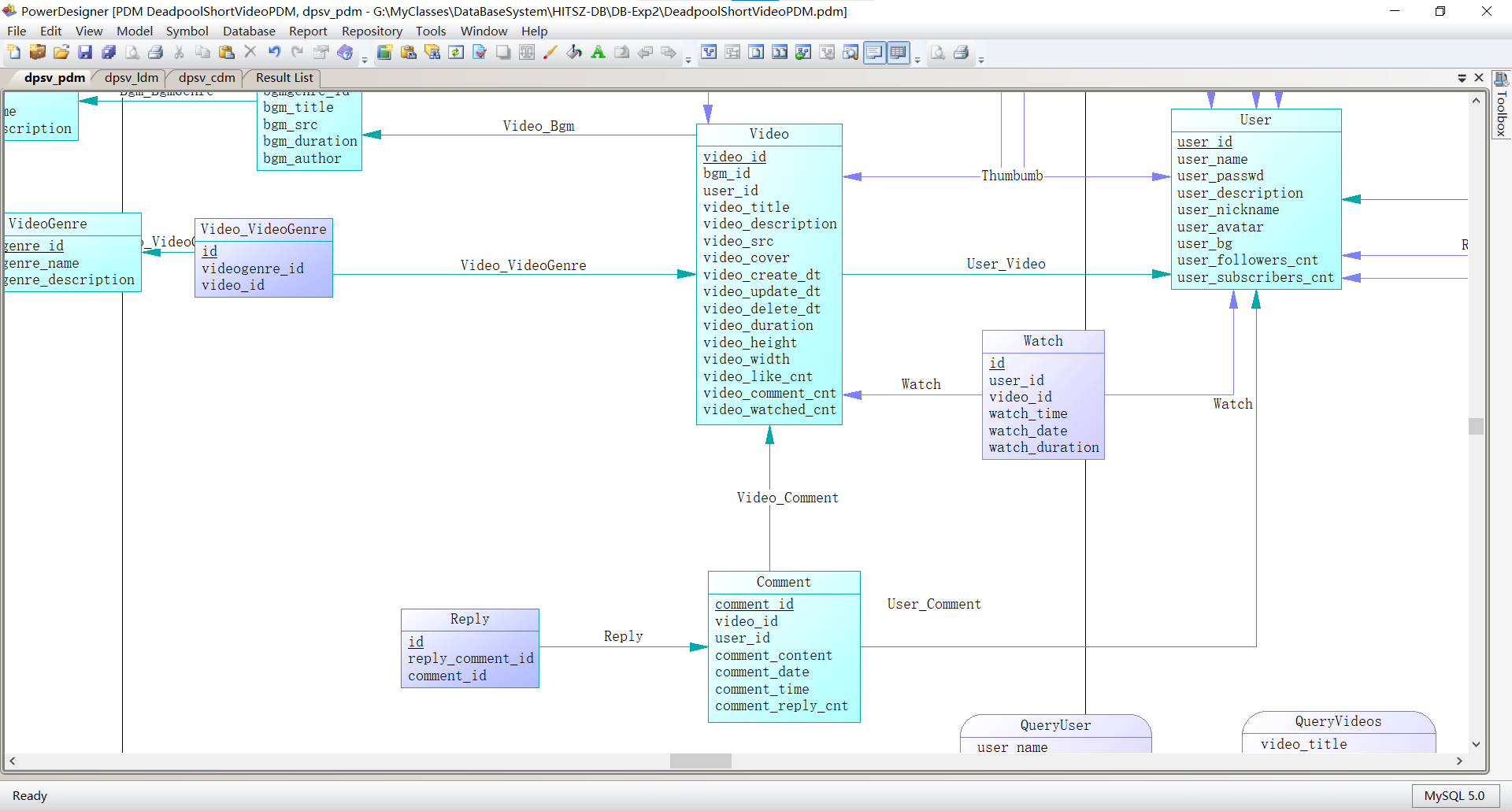


图26 评论与回复PDM图

# 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。

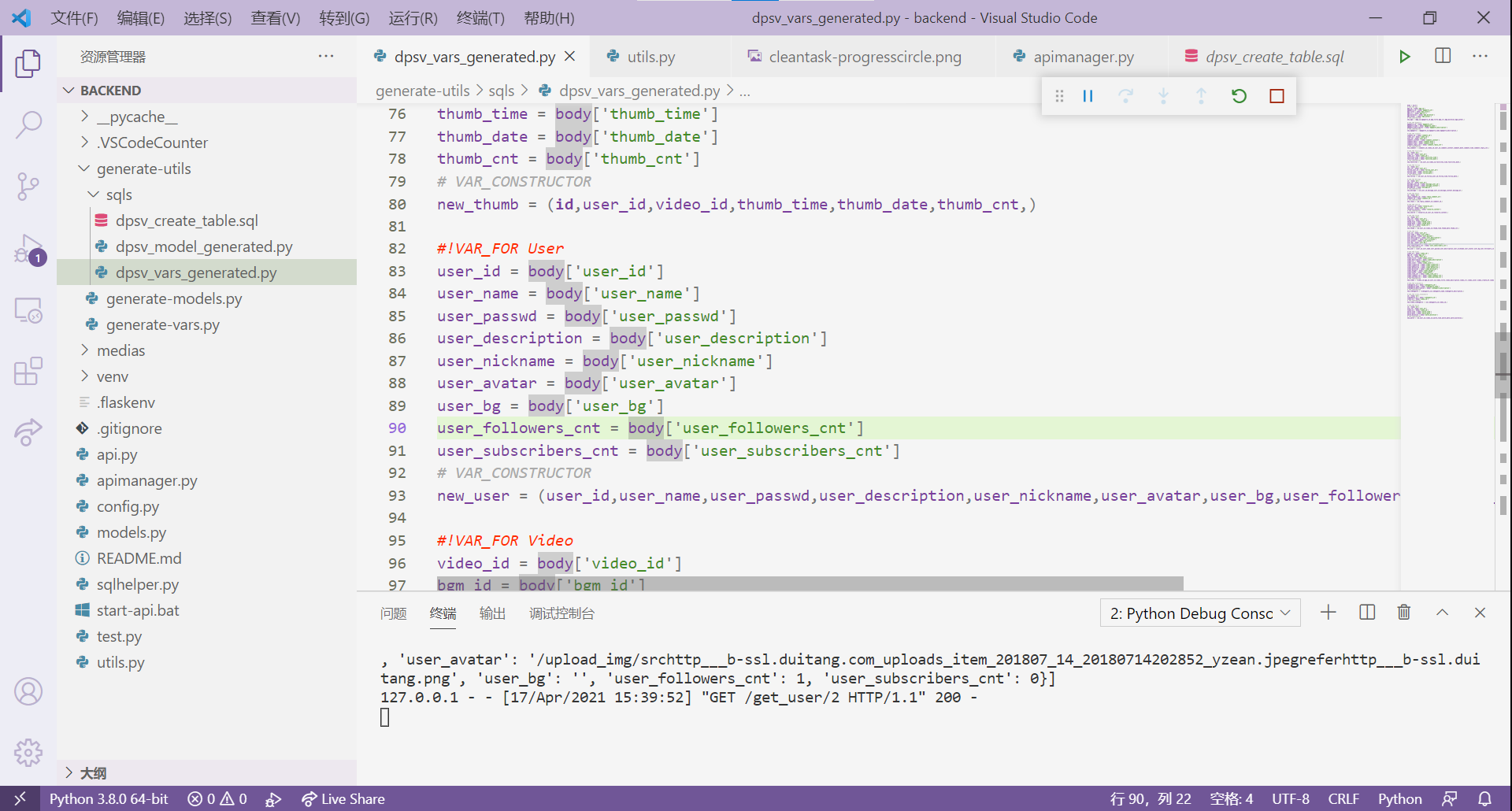


图27 自动生成的函数变量

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。