



同濟大學
TONGJI UNIVERSITY

同济大学《大数据与数据 工程》 课程实验报告

数据库的搭建及应用

任课老师：余有灵，龚炜

学生：2151406 刘卓明

班级：10072901

2024 年 4 月

目 录

第一章 实验要求	1
1.1 问题描述	1
1.2 实验环境	1
第二章 ER 图分析以及表的建立	2
2.1 用户表的建立	2
2.2 影视表的建立	2
2.3 留言表的建立	3
2.4 关系表——收藏表的建立	3
第三章 主要任务	4
3.1 任务 a	4
3.2 任务 b	8
3.3 任务 c	12
第四章 总结与思考	14
4.1 实验总结	14
4.2 个人思考与心得	15

第一章 实验要求

1.1 问题描述

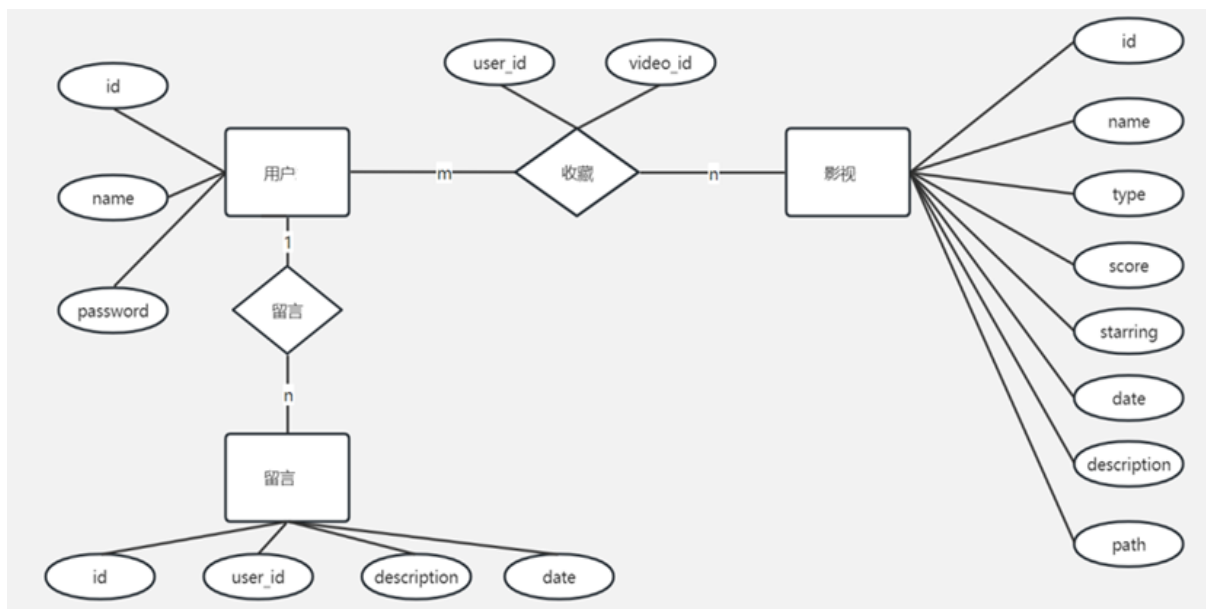


图 1.1: ER 图

要求：

1. 根据 ER 图，使用 create 等语句建立“影视管理”数据库和相关表
2. 使用 Mysql 语言语句，中文字符使用“utf8mb4”类型
 - a) 在数据库各个表中插入多行数据，其数量满足 b 的要求（查询结果正反例各至少 3）。
 - b) 查找“留言”里留言为“(自己的学号) 喜欢看动作电影”的用户中，收藏的影视平均 score 大于（自己的学号倒数第一个非零位）的用户名字。思考如何实现使得执行效率高（内存消耗、执行速度等）。
 - c) 将 b) 中涉及的用户每个人的所有留言清空。

1.2 实验环境

下面是本次实验的电脑环境：

项目	内容
CPU	Intel(R) Core(TM)i5-1035G1 CPU @ 1.00GHz
内存	16GB
OS	Windows10
内核	4

此外，本次使用的语言是 mysql，版本是 8.0.36，使用的开发环境是使用 vscode 连

接 mysql 进行开发。

第二章 ER 图分析以及表的建立

首先从图中我们分析可以得到图中有三个实体，分别是用户，留言和影视，而三个实体的具体情况如下：

2.1 用户表的建立

字段名	中文描述	类型	长度	是否可以空	是否作为主键
id	用户编号	VARCHAR	20	否	是
name	姓名	VARCHAR	20	否	否
password	电话	VARCHAR	20	否	否

这里进行分析，由于用户实体具有三个属性那么就设置三个列名，而类型的话 name 和 password 无疑是 varchar 型，id 的话（这里理解为学生学号）考虑过 qq 号是以 int 型储存的，但是这里思考过后觉得不需要用 id 进行计算，所以 varchar 型更合适，而且有些编号如微信号是会出现字母和数字的组合的，所以字符型更加合适，长度的话这里设置为 20 就足够使用了。这里由于只有 id 不会重复所以把 id 设置为主键是最合适的。

相关的代码如下：

```
# 建立用户表
CREATE TABLE `user` (
  `id` VARCHAR(20) NOT NULL,
  `name` VARCHAR(20) NOT NULL,
  `password` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

2.2 影视表的建立

字段名	中文描述	类型	长度或备注	主键及外键情况
id	电影编号	VARCHAR	20	主键
name	电影名	VARCHAR	20	-
type	电影类型	VARCHAR	20	-
score	电影评分	DECIMAL	(10,1)	-
starring	电影主演	VARCHAR	30	-
date	电影上映日期	DATE	-	-
description	电影描述	TEXT	-	-
path	电影网址	VARCHAR	200	-

这里进行分析：电影类中具有八个属性，把他们作为列名比较合适，而电影 id 作为主键比较合适；而类型的话，description 由于很可能是大段文字，所以用 text 类型比较合适，上映日期用 DATE 比较合适，分数的话这里用 DECIMAL(10,1) 类型，指的是分数为 0-10 分，一位小数，这也和我们的实际电影评分的状况比较相近，而 path 的话有可能会很长，所以这里给出了 200 的长度，其他都是 VARCHAR 比较合适。

相关代码如下：

```
# 建立 video 表
CREATE TABLE `video` (
  `id` VARCHAR(20) NOT NULL,
  `name` VARCHAR(20) NOT NULL,
  `type` VARCHAR(20) NOT NULL,
  `score` DECIMAL(10,1) NOT NULL,
  `starring` VARCHAR(30) NOT NULL,
  `date` DATE NOT NULL,
  `description` TEXT NOT NULL,
  `path` VARCHAR(200) NOT NULL,
  PRIMARY KEY (`id`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

2.3 留言表的建立

字段名	中文描述	类型	长度或备注	主键及外键情况
id	留言的 id	VARCHAR	20	作为主键
user_id	用户 id	VARCHAR	20	外键，来自用户表格
description	留言的具体内容	TEXT	-	-
date	留言日期	DATE	-	-

分析如下：留言主体具有四个属性，可以分别作为列名，而主键的话，显然是留言的 id 作为主键，而用户 id 来自于用户表，显然是作为外键。变量类型和影视表理解方式相近，description 用 TEXT 类型，date 用 DATE 类型相对来说合适，而 id 和 user_id 用 VARCAHR。代码如下：

```
# 建立 comment 表
CREATE TABLE `comment` (
  `id` VARCHAR(20) NOT NULL,
  `user_id` VARCHAR(20) NOT NULL,
  `description` TEXT NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`user_id`) REFERENCES `user` (`id`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

2.4 关系表——收藏表的建立

这里 E-R 图中除了三个实体之外，还有实体和实体之间的关系，从 E-R 图中总结出来的实体关系如下：

用户 < 留言 > 留言 (1: n)

用户 < 收藏 > 影视, 并有 user_id, video_id (m: n)

根据上面的两条实体关系, 用户留言这一个关系在留言表已经用外键进行连接, 所以没有必要进行增添; 用户收藏影视的话形成了 m:n 的关系, 同时还具有 user_id, video_id 的属性, 可以建立一个新的关系表, 具体情况如下:

字段名	中文描述	类型	长度	主键及外键情况
user_id	用户 id	VARCHAR	20	外键, 来自于用户表 同时也和 video_id 组合作为联合主键
video_id	影视 id	VARCHAR	20	外键, 来自于影视表 同时也和 user_id 组合作为联合主键

这里进行分析, 在该关系表中, 作为列名的显然是 user_id 和 video_id, 而这两个属性都来自于别的表, 显然都是作为外键, 其中 user_id 来自于用户表的 id, video_id 来自于影视表的 id, 而本表的外键的话, 选择 user_id 和 video_id 组合作为联合主键, 这样能保证收藏这一动作的唯一性。代码如下:

```
# 建立 favorite 表
CREATE TABLE `favorite` (
  `user_id` VARCHAR(100) NOT NULL,
  `video_id` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`user_id`, `video_id`),
  FOREIGN KEY (`user_id`) REFERENCES `user`(`id`),
  FOREIGN KEY (`video_id`) REFERENCES `video`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

第三章 主要任务

3.1 任务 a

任务要求如下:

在数据库各个表中插入多行数据, 其数量满足 b 的要求 (查询结果正反例各至少 3)。

这里我们的数据来源主要是两种, 第一种是自己构造, 以方便后面的要求, 另外的是通过爬虫获取一些较为合适的数据, 这里爬虫使用的是 python 语言, 不在这里赘述, 插入数据的情况如下:

1. 插入用户信息:

```
-- 插入用户数据
INSERT INTO `user` (`id`, `name`, `password`) VALUES
('u1', 'Jack', 'pw1'),
('u2', 'Nancy', 'pw2'),
('u3', 'George', 'pw3'),
('u4', 'Amy', 'pw4');
```

```
( 'u5', 'Martin', 'pw5'),
( 'u6', 'Adam', 'pw6'),
( 'u7', 'uzi', 'pw7'),
( 'u8', 'jiejie', 'pw8');
```

这里为了简洁，用户名都以 u+ 数字的形式表示，密码都以 pw+ 数字的形式表示，名字是一些常见的英文名，共插入 8 条数据。情况如下：

	Q	id varchar	name varchar	password varchar
	> 1	u1	Jack	pw1
	> 2	u2	Nancy	pw2
	> 3	u3	George	pw3
	> 4	u4	Amy	pw4
	> 5	u5	Martin	pw5
	> 6	u6	Adam	pw6
	> 7	u7	uzi	pw7
	> 8	u8	jiejie	pw8

图 3.1: 用户表

2. 插入影视数据：

```
-- 插入影视数据
INSERT INTO `video` (`id`, `name`, `type`, `score`, `starring`, `
    date`, `description`, `path`) VALUES
( 'v1', '我和我的祖国', '剧情', 8.2, '黄渤', '2019-09-30', '一部讲述
    中国历史和人民生活的影片.', '路径1'),
( 'v2', '火星救援', '科幻', 8.0, '马特·达蒙', '2015-11-25', '一部描绘
    人类逆境求生的电影.', '路径2'),
( 'v3', '你的名字', '动画', 8.4, '神木隆之介', '2016-12-02', '一部描
    绘时间穿越、命运交织的故事.', '路径3'),
( 'v4', '那些年，我们一起追过的女孩', '爱情', 7.4, '柯震东', '
    2011-08-19', '一部描绘青春的爱情故事.', '路径4'),
( 'v5', '波西米亚狂想曲', '传记', 8.2, '拉米·马雷克', '2019-03-22', '
    一部讲述摇滚乐团Queen主唱的人物传记.', '路径5'),
( 'v6', '致命魔术', '剧情', 3.5, '詹森·斯坦森', '2006-10-20', '一部被
    批评为过于夸张的剧情电影.', '路径6'),
( 'v7', '搏击之王', '动作', 2.8, '斯蒂芬·席格尔', '2002-06-06', '一部
    质感清淡、情节无新意的动作片.', '路径7'),
( 'v8', '大卫·贝肯之霸王别姬', '喜剧', 1.6, '大卫·贝肯', '2012-07-27'
    , '一部低俗，毫无逻辑的低端喜剧.', '路径8'),
( 'v9', '野蛮人巴巴', '奇幻', 2.3, '施瓦辛格', '1982-02-25', '一部被
    批评为太过商业化的奇幻电影.', '路径9'),
( 'v10', '母女大战', '家庭', 3.2, '琳赛·洛翰', '2006-05-01', '一部被
    批评为尴尬，令人难以忍受的家庭电影.', '路径10');
```

这里通过爬虫和生成工具搜索与生成了十部各种类型的电影，其中五部评分较高的电影和五部评分较低的电影，方便后面任务 b（由于自己的倒数第一位不为 0 的数是 6，相对比较靠近 5）；而电影的编号就采用较为简单的 v+ 数字系列，路径也进行了简写。

影视表的情况如下：

	Q	id varchar	name varchar	type varchar	score newdecim	starring varchar	date date	description blob	path varchar
	> 1	v1	我和我的祖国	剧情	8.2	黄渤	2019-09-30	一部讲述中国历史和人民	路径1
	> 2	v10	母女大战	家庭	3.2	琳赛·洛翰	2006-05-01	一部被批评为尴尬，令人	路径10
	> 3	v2	火星救援	科幻	8.0	马特·达蒙	2015-11-25	一部描绘人类逆境求生的	路径2
	> 4	v3	你的名字	动画	8.4	神木隆之介	2016-12-02	一部描绘时间穿越、命运	路径3
	> 5	v4	那些年，我们一	爱情	7.4	柯震东	2011-08-19	一部描绘青春的爱情故事	路径4
	> 6	v5	波西米亚狂想曲	传记	8.2	拉米·马雷克	2019-03-22	一部讲述摇滚乐团Queen	路径5
	> 7	v6	致命魔术	剧情	3.5	詹森·斯坦森	2006-10-20	一部被批评为过于夸张的	路径6
	> 8	v7	搏击之王	动作	2.8	斯蒂芬·席格	2002-06-06	一部质感清淡、情节无新	路径7
	> 9	v8	大卫·贝肯之霸王	喜剧	1.6	大卫·贝肯	2012-07-27	一部低俗，毫无逻辑的低	路径8
	> 10	v9	野蛮人巴巴	奇幻	2.3	施瓦辛格	1982-02-25	一部被批评为太过商业化	路径9

图 3.2: 影视表

3. 插入留言：

```
-- 插入留言数据
INSERT INTO comment (`id`, `user_id`, `description`, `date`) VALUES
('c1', 'u1', '2151406喜欢看动作电影.', '2024-03-30'),
('c2', 'u2', '无敌破坏王是我最喜欢的动画电影之一，很有创意.', '2024-03-29'),
('c3', 'u2', '星际穿越的剧情有点复杂，但还是喜欢它的科幻元素.', '2024-03-27'),
('c4', 'u3', '2151406喜欢看动作电影.', '2024-03-25'),
('c5', 'u2', '2151406喜欢看动作电影.', '2024-03-22'),
('c6', 'u1', '致命魔术实在是让人疑惑，难以理解其剧情.', '2024-03-20'),
('c7', 'u3', '搏击之王的动作场面需要再加工，特效也很一般.', '2024-03-18'),
('c8', 'u4', '大卫·贝肯之霸王别姬实在太低俗，理解不了为什么会有人喜欢.', '2024-03-15'),
('c9', 'u4', '期待野蛮人巴巴能更注重故事的构造，而不是只看重商业效益.', '2024-03-12'),
('c10', 'u5', '母女大战的剧情有些尴尬，对演员本身的浪费.', '2024-03-11'),
('c11', 'u6', '2151406喜欢看动作电影.', '2024-03-11'),
('c12', 'u7', '有操作的呀.', '2024-03-15'),
('c13', 'u7', '烂的一匹，洗澡去了.', '2024-03-12'),
('c14', 'u7', '2151406喜欢看动作电影.', '2024-03-11'),
('c15', 'u8', '2151406喜欢看动作电影.', '2024-03-11');
```


这里满足 b 任务，在留言中有好几条 “2151406 喜欢看电影。”

留言表情况如下（这里截取一部分）：

Q	id varchar	user_id varchar	description blob	date date
> 1	12	u7	有操作的呀.	2024-03-15
> 2	c1	u1	2151406喜欢看动作电影.	2024-03-30
> 3	c10	u5	母女大战的剧情有些尴尬,	2024-03-11
> 4	c11	u6	2151406喜欢看动作电影.	2024-03-11
> 5	c13	u7	烂的一匹, 洗澡去了.	2024-03-12
> 6	c14	u7	2151406喜欢看动作电影.	2024-03-11
> 7	c15	u8	2151406喜欢看动作电影.	2024-03-11
> 8	c2	u2	无敌破坏王是我最喜欢的	2024-03-29
> 9	c3	u2	星际穿越的剧情有点复杂.	2024-03-27
> 10	c4	u3	2151406喜欢看动作电影.	2024-03-25
> 11	c5	u2	2151406喜欢看动作电影.	2024-03-22
> 12	c6	u1	致命魔术实在是让人疑惑.	2024-03-20

图 3.3: 留言表

4. 插入收藏信息

```
-- 插入收藏数据
INSERT INTO `favorite` (`user_id`, `video_id`) VALUES
('u1', 'v1'),
('u1', 'v3'),
('u1', 'v4'),
('u1', 'v10'),
('u2', 'v1'),
('u2', 'v4'),
('u2', 'v5'),
('u2', 'v7'),
('u2', 'v8'),
('u3', 'v4'),
('u3', 'v2'),
('u3', 'v1'),
('u4', 'v5'),
('u3', 'v3'),
('u5', 'v6'),
('u5', 'v5'),
('u6', 'v2'),
('u6', 'v6'),
('u6', 'v9'),
('u7', 'v6'),
('u7', 'v8'),
('u7', 'v5'),
('u7', 'v9'),
('u7', 'v10'),
('u8', 'v1'),
('u8', 'v2');
```

收藏表情况如下（展示部分）：

Q	user_id varchar	video_i varchar
> 1	u1	v1
> 2	u1	v10
> 3	u1	v3
> 4	u1	v4
> 5	u2	v1
> 6	u2	v4
> 7	u2	v5
> 8	u2	v7
> 9	u2	v8
> 10	u3	v1

图 3.4: 收藏表

这里我们先可以预先进行一个统计，方便后续检验代码是否正确，有“2151406 喜欢看电影。”留言的一共有 u1, u2, u3, u6, u7, u8 一共六位用户，其中 u1 用户收藏了共四部电影，平均分为 6.8 分，u2 用户收藏了五部电影，平均分为 5.64，u3 用户收藏了 4 部电影，平均分为 8 分，u6 收藏了三部电影，平均分为 4.6，u7 收藏了五部电影，平均分为 3.76，u8 收藏了两部电影，平均分为 8.1。经过分析我们可以发现在六个用户中满足条件的共有三位用户，分别为:u1,u3,u8, 他们对应的名字为 Jack, George,jiejie。

3.2 任务 b

任务要求如下：

查找“留言”里留言为“(自己的学号) 喜欢看动作电影”的用户中，收藏的影视平均 score 大于（自己的学号倒数第一个非零位）的用户名字。

这里我的学号是 2151406，所以任务就是查找留言为“2151406 喜欢看动作电影。”的用户中收藏的影视的平均 score 大于 6 的用户名字。

任务分析：这里需要的找到的用户有以下几个特征：具有“2151406 喜欢看动作电影。”的留言，这涉及到了留言表中的留言内容，同时通过留言表中的 user_id 与用户产生联系；同时收藏的影视平均 score > 6，这里涉及到的是收藏表以及影视表，收藏表中的用户 id 与用户建立起了联系，通过收藏表可以找到特定的用户收藏的电影；而影视表中的影视 id 和收藏表中的 video_id 建立起了联系，同时影视表中具有 score 属性，这里可以找到特定用户收藏的电影的分数，然后求平均分数和 6 比较即可。

针对以上的操作，可以有几种实现的不同方式，分别如下（注：这里由于实现的效果是一致的，所以在最后展示运行截图）：

1. 方法 1:

代码如下：

```
--方法1，采用join函数的方法，把四个表直接连接起来进行查找
SELECT user.name as b
FROM user
JOIN comment ON user.id = comment.user_id
JOIN favorite ON user.id = favorite.user_id
JOIN video on favorite.video_id = video.id
WHERE comment.description = '2151406 喜欢看动作电影.'
GROUP BY user.id
HAVING AVG(video.`score`) > 6.0;
```

这里的方法是采用 join 函数，把几个表都连接在一起（加入需要的列名），首先通过 user.id 把用户和留言给连起来（加入了 comment），然后通过用户 id 再把用户表和收藏表连接起来（加入了 favorite），最后通过影视的 id 把收藏表和影视表连接起来（加入了 video）。然后限定条件是按照 where 指令，拥有‘2151406 喜欢看动作电影.’的被筛选出来，然后再通过 having 命令来筛选平均分 > 6 的用户。

运行结果如下：

2. 方法 2:

代码如下：

```
--方法2，
SELECT name
FROM user
WHERE id IN (
    SELECT user_id
    FROM comment
    WHERE description = '2151406 喜欢看动作电影.'
) AND id IN (
    SELECT user_id
    FROM favorite
    INNER JOIN video ON video_id = video.id
    GROUP BY user_id
    HAVING AVG(score) > 6
);
```

这里的方法是使用 where 与 and 两个部分连进行筛选。在 where 部分筛选的是留言的条件，输出的是用户的 id；然后 and 部分筛选的是平均分条件，这里使用 inner join 把 video 表和 favorite 表连接起来，输出的同样是用户的 id，同时满足这两个条件的用户 id 就是符合我们总条件的用户 id，然后根据用户表输出对应的姓名就可以。

3. 方法 3:

代码如下:

```
-- 方法3
SELECT name
FROM user
WHERE id IN (
    SELECT user_id
    FROM comment
    WHERE description = '2151406喜欢看动作电影.'
)
AND id IN (
    SELECT user_id
    FROM (
        SELECT user_id, AVG(score) AS avg_score
        FROM favorite
        INNER JOIN video ON video_id = video.id
        GROUP BY user_id
    ) AS avg_scores
    WHERE avg_score > 6
);
```

这里的方法和方法 2 比较相近, 通过是采用 where 和 and 两个条件来进行筛选, 第一个 where 的部分和方法 2 的一致, 而 and 部分和方法 2 不一样的是把 and 部分内部筛选的方式从 having——on 换成了又 where, 这里实现的效果是一样的, 只不过二者的区别是 having 是在分组后对数据进行过滤 where 是在分组前对数据进行过滤, 这里那种方法更加高效我认为还是需要考虑到数据的分布情况, 如果这里分组的类别特别多, 而相对来说需要查找的数据特别少的话, 用 where 会高效一些, where 可以更快地提前找到满足条件的数据。

4. 方法 4:

代码如下:

```
-- 方法4
CREATE TEMPORARY TABLE TempUsers
SELECT name
FROM user u
WHERE EXISTS (
    SELECT 1
    FROM comment c
    WHERE c.user_id = u.id AND description = '2151406喜欢看动作电影.'
)
AND EXISTS (
    SELECT 1
    FROM favorite f
    JOIN video v ON f.video_id = v.id
    WHERE f.user_id = u.id
    HAVING AVG(v.score) > 6
);
```

这里使用了 `create` 语句创建了一个临时的表，目的是把这一问中筛选出来的数据临时保存起来，方便问题 c 中对查询出来的用户的所有留言进行筛选；此外，方法 4 中使用了 `exists` 语句，它会对于符合条件的情况返回 `true`，对不符合条件的情况返回 `false`，而 `select 1` 的含义指的是查询返回至少一行数据，如果存在就会继续继续查询操作，不存在就结束，我认为这样的查询方式可能在大量数据筛选的时候会比 `in` 更加高效。

实现的筛选效果如下：

```
133  --方法2,
    ▶ Execute | JSON | Copy
✓ 134  SELECT name
135  FROM user
136  WHERE id IN (
137      SELECT user_id
138      FROM comment
139      WHERE description = '2151406喜欢看动作电影.'
140  ) AND id IN (
141      SELECT user_id
142      FROM favorite
143      INNER JOIN video ON video_id = video.id
144      GROUP BY user_id
145      HAVING AVG(score) > 6
146  ); 2ms
147
148  -- 方法3
```

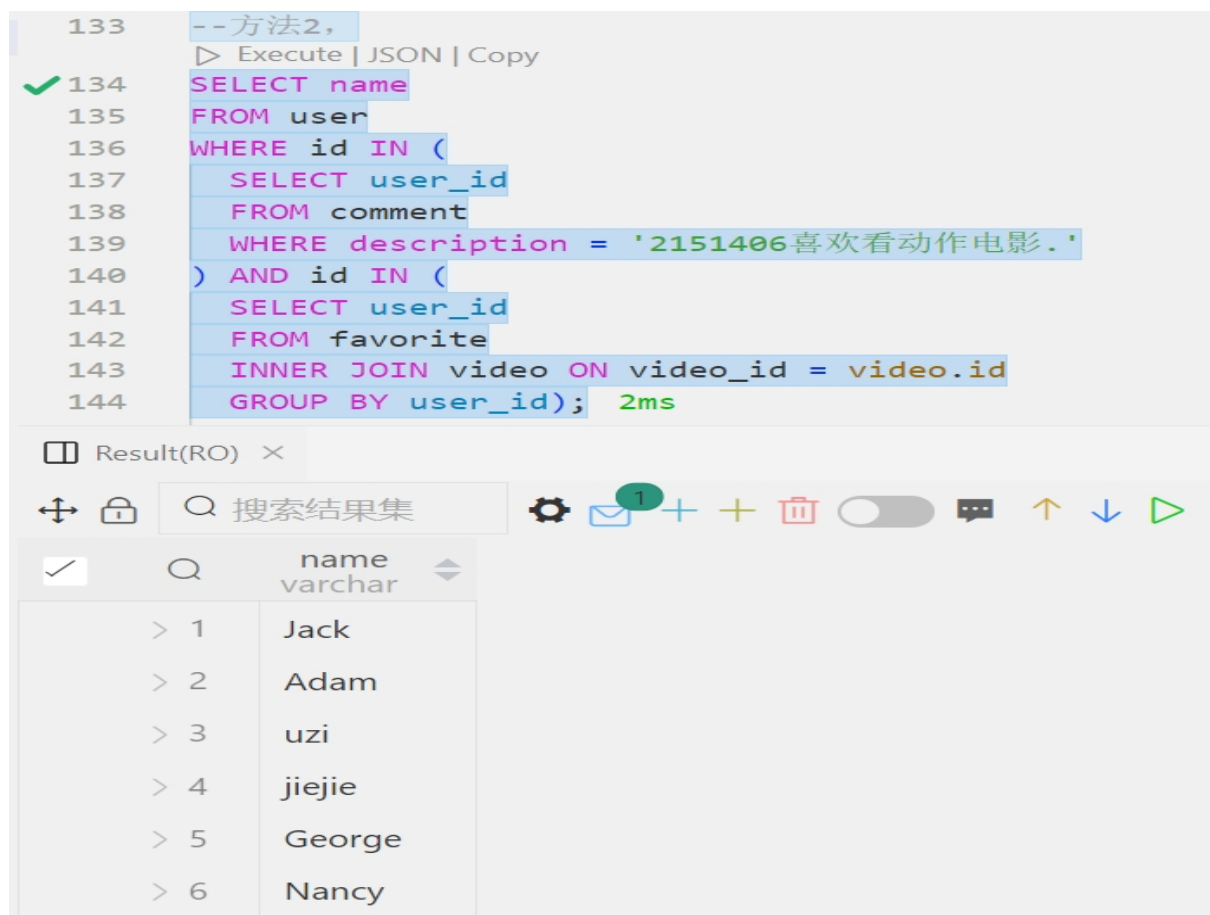
Result(RO) ×

🔍 搜索结果集

	name varchar
> 1	Jack
> 2	jiejie
> 3	George

图 3.5: 任务 b 结果

发现确实是之前分析的三个人。这里可以尝试把代码中的分数的约束去掉，只查找满足留言条件的人，结果如下：



The screenshot shows a SQL query execution interface. The query is as follows:

```

133  --方法2,
134  SELECT name
135  FROM user
136  WHERE id IN (
137      SELECT user_id
138      FROM comment
139      WHERE description = '2151406喜欢看动作电影.'
140  ) AND id IN (
141      SELECT user_id
142      FROM favorite
143      INNER JOIN video ON video_id = video.id
144      GROUP BY user_id); 2ms

```

The results are displayed in a table with the following data:

	name
> 1	Jack
> 2	Adam
> 3	uzi
> 4	jiejie
> 5	George
> 6	Nancy

图 3.6: 任务 b 结果对照

发现确实是有指定发言的六个人，也说明了用于筛选的函数是正确的。然后思考关于几种方式执行效率高（内存消耗、执行速度）在实验报告的最后一部分。

3.3 任务 c

任务要求：将 b) 中涉及的用户每个人的所有留言清空。

代码如下：

```

-- 清空所选用户的留言内容
UPDATE comment
SET description = ''
WHERE user_id IN (
    SELECT u.id
    FROM user u
    INNER JOIN TempUsers t ON u.name = t.name
);

```

这里是使用了任务 b 中保存的临时表，而临时表中保存的是用户的姓名，我们只需要通过用户表就能通过用户的姓名得到用户的 id，然后再用 update 操作将 comment 表中对应用户的 description 设置为空就可以。这里由于在设置变量的时候，将 description 设置为了非 null（因为考虑到留言没有任何内容没有意义），所以这里采用的是将留言

内容换成 ° 的方式，如果设置为了允许 null，那么可以 set 为 null。

留言内容被清空之后情况如下：

	id	user_id	description	date
> 1	c1	u1		2024-03-30
> 2	c6	u1		2024-03-20
> 3	c2	u2	无敌破坏王是我最喜欢的	2024-03-29
> 4	c3	u2	星际穿越的剧情有点复杂	2024-03-27
> 5	c5	u2	2151406喜欢看动作电影	2024-03-22
> 6	c4	u3		2024-03-25
> 7	c7	u3		2024-03-18
> 8	c8	u4	大卫·贝肯之霸王别姬实在	2024-03-15
> 9	c9	u4	期待野蛮人巴巴能更注重	2024-03-12
> 10	c10	u5	母女大战的剧情有些尴尬	2024-03-11
> 11	c11	u6	2151406喜欢看动作电影	2024-03-11
> 12	12	u7	有操作的呀	2024-03-15
> 13	c13	u7	烂的一匹，洗澡去了	2024-03-12
> 14	c14	u7	2151406喜欢看动作电影	2024-03-11
> 15	c15	u8		2024-03-11

图 3.7: 任务 c 结果

我们可以看到三名用户的留言已经全空了；同时这里考虑的是把留言的内容清空，而不删除该条留言的其他内容，如果考虑掉要把整条留言删除掉，那可以使用 delete 语句，相关代码如下：

```
--情况所选用户的整条留言
DELETE comment
FROM comment
JOIN user ON comment.user_id = user.id
WHERE user.name IN (
    SELECT name
    FROM TempUsers
);
```

整条留言都被清空之后如下：

comment

🔍 搜索结果集

⚙️

📧

1

+

+

🗑️

🔍

🗨️

⬆️

⬆️

🎯

👁️

耗时: 3ms

		id varchar	user_id varchar	description blob	date date
<input type="checkbox"/>	> 1	12	u7	有操作的呀.	2024-03-15
<input type="checkbox"/>	> 2	c10	u5	母女大战的剧情有些尴尬,	2024-03-11
<input type="checkbox"/>	> 3	c11	u6	2151406喜欢看动作电影.	2024-03-11
<input type="checkbox"/>	> 4	c13	u7	烂的一匹, 洗澡去了.	2024-03-12
<input type="checkbox"/>	> 5	c14	u7	2151406喜欢看动作电影.	2024-03-11
<input type="checkbox"/>	> 6	c2	u2	无敌破坏王是我最喜欢的,	2024-03-29
<input type="checkbox"/>	> 7	c3	u2	星际穿越的剧情有点复杂,	2024-03-27
<input type="checkbox"/>	> 8	c5	u2	2151406喜欢看动作电影.	2024-03-22
<input type="checkbox"/>	> 9	c8	u4	大卫·贝肯之霸王别姬实在	2024-03-15
<input type="checkbox"/>	> 10	c9	u4	期待野蛮人巴巴能更注重	2024-03-12

图 3.8: 任务 c 结果

我们这里可以看到整条留言被清空后，comment 只剩下了 10 条，说明三名用户留的 5 条留言已经被删除了。

第四章 总结与思考

4.1 实验总结

本次实验主要是进行了数据库的搭建及应用，首先通过提供的 ER 图分析了实体和实体之间的关系，然后建立了三个实体表与关系表；然后分别向四个表导入了部分数据，导入数据后进行筛选操作；这里主要进行了两个任务，一个是查找“留言”里留言为“(自己的学号) 喜欢看动作电影”的用户中，收藏的影视平均 score 大于（自己的学号倒数第一个非零位）的用户名字，另一个是将 b) 中涉及的用户每个人的所有留言清空。其中在第一个任务中用到了多种查找与筛选的方法。

4.2 个人思考与心得

通过本次实验，我有以下几点收获：

1. 第一点是对任务 b 中查找相关的数据的方式的思考，由于本例中数据量较少，所以几种方式并没有明显的差别；而当实际应用的时候一般面对的都是大批量的数据，所以必须要考虑内存消耗与执行速度。

在本次实验所用的四种方法中显然第一种方法是最简洁的，但是显然它用了三个 join 把这几张表都连接起来了，在实际应用的时候肯定会占用相当大的内存，查找效率也会非常慢。

而第二种和第三种方法相近，他们都使用了 where 或者是 having on 的方式进行筛选，不同的地方在于他们使用时机的不同：

WHERE 子句在 GROUP BY 分组和聚合函数之前对数据行进行过滤；HAVING 子句对 GROUP BY 分组和聚合函数之后的数据行进行过滤。

而这里我也通过一些小实验总结出了几种字句在查找时候的执行顺序：from>where>group (含聚合)>having>order>select。这里的聚合函数指的是 SUM, COUNT, MAX, AVG 等函数，而我认为这几种查找字句孰优孰略并没有一个定论，需要根据实际情况进行判定，数据的整体分布情况以及需要查找的数据的分布情况以及查找中的子查询的数据的分布情况都会影响效率，所以需要根据具体的情况和它们各自的用法来进行考量。

而第四种方式是使用了 exists 语句，我认为在数据量非常大的时候 exist 的工作方式使得在查找速度上优于 IN，但是可能要根据实际的数据情况进行分析。

2. 第二点是在任务 c 的时候考虑到的，在清楚了留言之后想再看原表的内容发现看不到了，只能 drop 掉所有的表重新建立表与导入数据，然后我认为在实际工程中应用增删改等操作时候一定要把原数据进行备份，可以通过对同一个表 copy 多个表来实现，也可以通过导出到 csv 等文件中实现，否则一旦进行了一些不可逆的操作后会对整个工程造成巨大的影响。
3. 第三是对建立表的思考，在实验中设计数据库表结构和字段时，需要考虑数据类型、约束以及表之间的关联。如何合理的建立表的列名，主键，外键，数据类型，长度等的情况以及表的个数都是非常重要的，对于整个数据工程来说是关键的起始步骤，良好的数据库设计对于后续数据操作和查询至关重要。