工程作业报告

1.项目信息

①开发者信息：

学号 ：2313180　　姓名：欧广元　　专业：信息安全

②必备环境

[1]电脑装配有 python 环境，并且能够运行 Flask 框架

[2]数据库使用 Mysql 8.0

[3]前端 IDE 使用 VS code，电脑装配有 Javascript 的环境，并且 VS code 有直接打开运行的本地服务器的扩展

③系统主要功能简介：

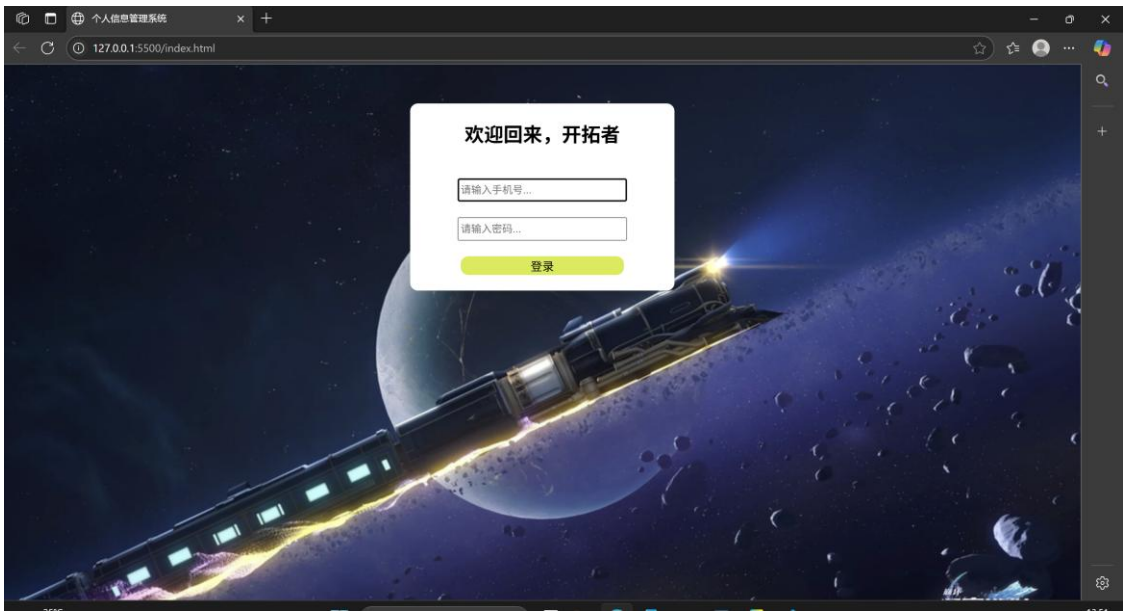
仿照崩坏星穹铁道游戏的玩家端操作界面，还原出一个搭建建立在玩家数据库上的网页，模拟实现部分游戏内玩家端的功能，如查询仓库，添加删除修改道具配置，角色信息等。

④系统主要页面截图：

[1] 玩家登录框：

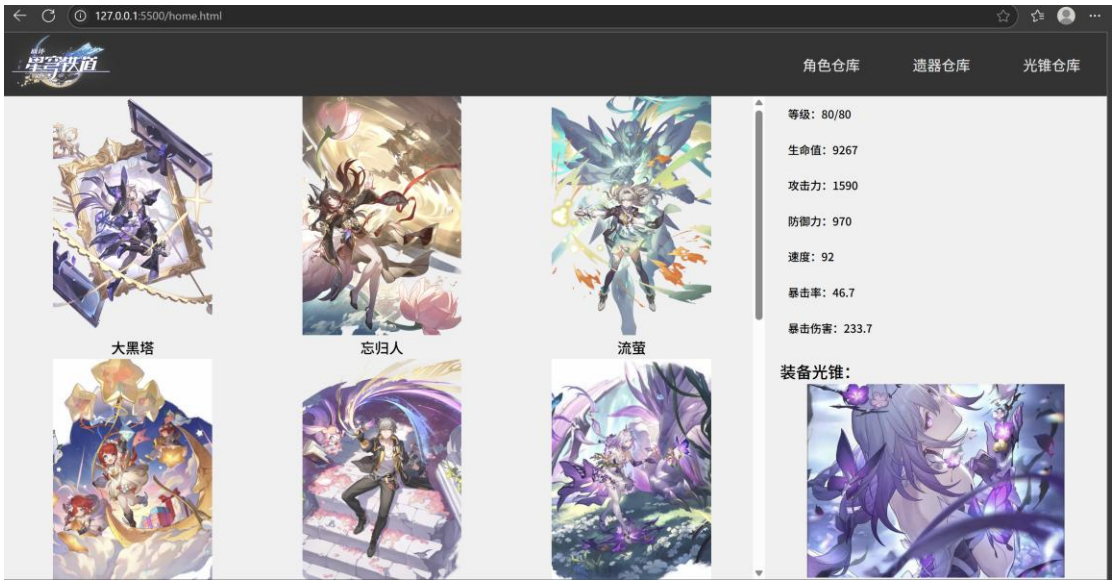
登陆后数据库才会调出属于该玩家的游戏数据，并显示

接下来的截图是使用我的账户信息登录后显示的内容，不同玩家会显示不同的信息。

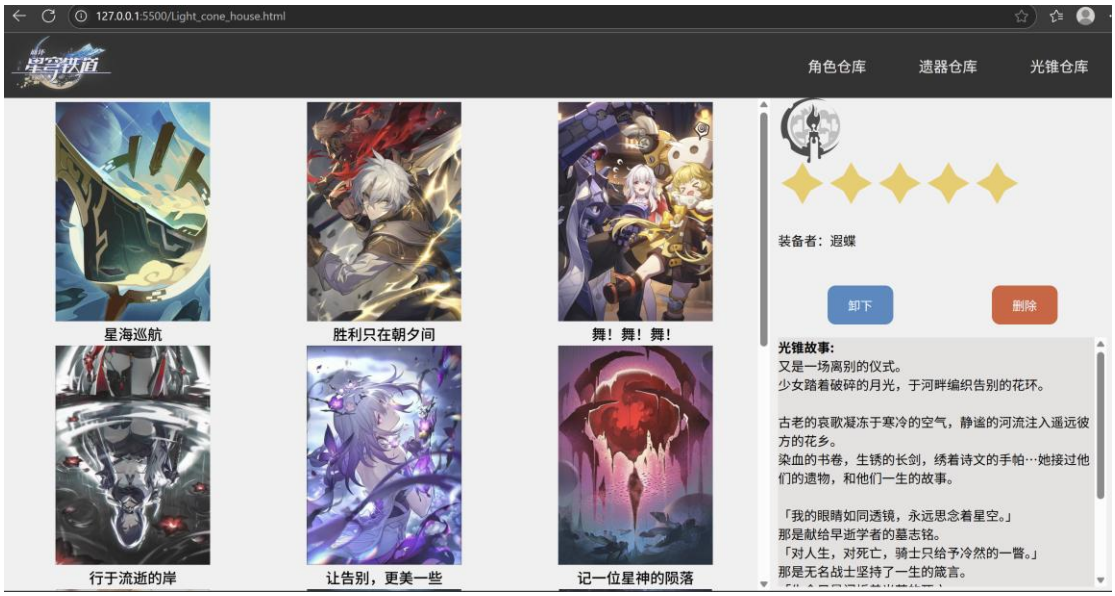


[2]角色仓库页面

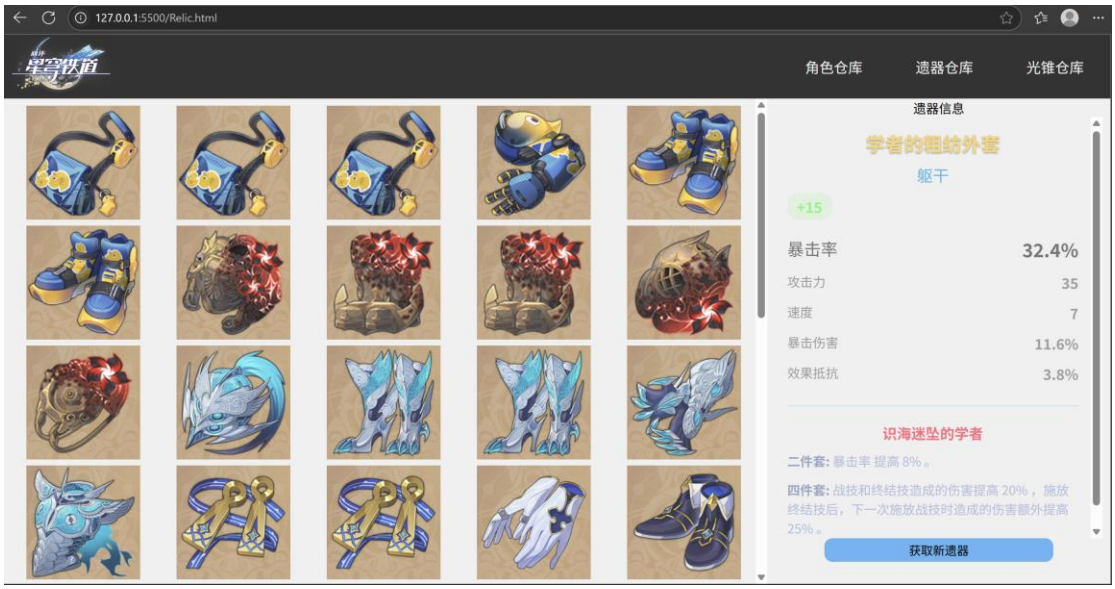
该页面会显示当前玩家所拥有的所有角色，点击角色图片会在右侧显示出该名角色的属性信息



[3]光锥仓库界面：
显示了该名玩家拥有的光锥，点击光锥图片可以在右侧显示出该光锥的信息。
同时玩家可以选择卸下或者删除该光锥。



[4]遗器仓库界面：
会显示出该玩家所拥有的所有遗器，点击某个遗器图片可在右侧显示出相关信息
同时玩家可以点击下方按钮获取新的遗器。



2.系统配置

①说明：
[1]后台数据库：Mysql 8.0
[2]高级语言：python 3.12

②配置步骤：
[1]DBMS
1.安装 Mysql 8.0 并安装开发软件 Mysql Workbench8.0 CE
2.创建数据库：star_rail

[2]高级语言：
1.安装 python 3.12
2.安装 Flask 库配置后端服务器

③连接串分析：

[1]数据库服务器地址：localhost:5000
[2]数据库登录用户名：root
[3]数据库登录密码：20050515ogy
[4]目标数据库名：star_rail

④连接串代码截屏

```python
# 数据库配置
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '20050515ogy',
    'database': 'star_rail'
}
```

```python
if __name__ == '__main__':
    app.run(host='localhost', port=5000, debug=True)
```

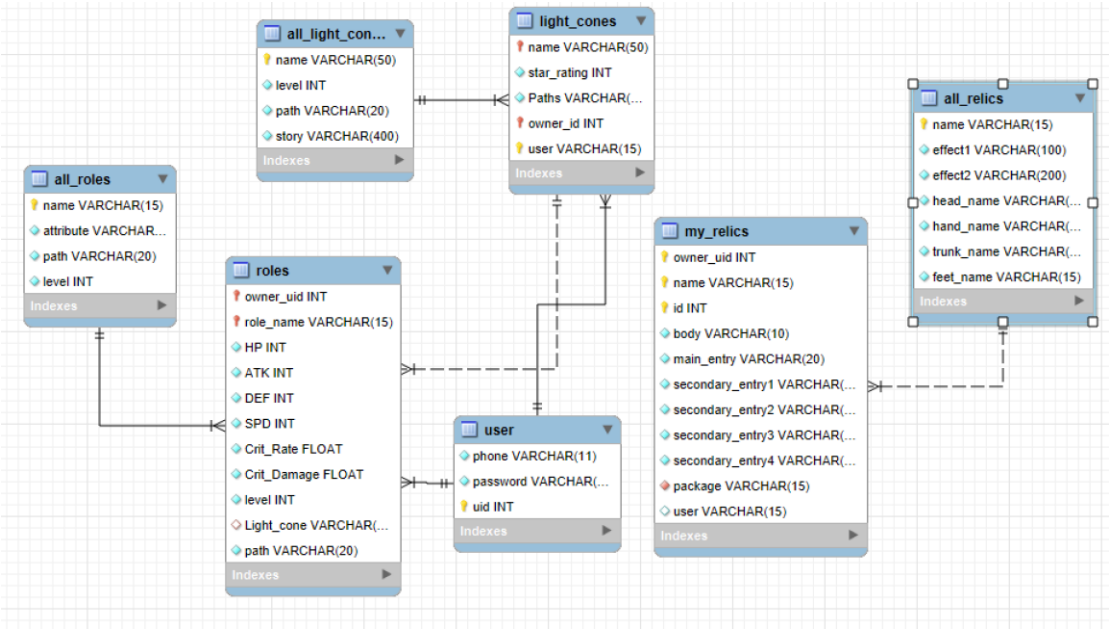
备注：配置过程使用的是 python 的 Flask 框架，若配置语言或框架不同，则需要对应调整驱动和语法。

3.数据库设计

①数据表：

创建顺序	数据表名称	主键	参照属性	被参照表及属性
1	user	uid	无	无
2	all_roles	name	无	无
3	all_light_cones	name	无	无
4	all_relics	name	无	无
5	light_cones	owner_uid,name,user	name, owner_uid	all_light_cones(name) user(uid)
6	my_relics	owner_uid,name,id	name, owner_uid	all_relics(name), user(uid)
7	roles	owner_uid,role_name	role_name, owner_uid Light_cone	all_roles(name), user(uid) my_relics(name)

②关系图：



备注：数据中的每个表设计到的属性只是原游戏的一小部分，原游戏中的每个实体集合字段应该会更多，本数据库只是出于模拟的目的复刻了其中的一部分

4.含有事务应用的删除操作

①功能描述：
在光锥仓库中，玩家能够删除目前已有的某个光锥，达到管理自己游戏道具的效果，与此同时，如果某个角色装备了这个光锥，那么这个角色的 Light_cone 字段也会被置空

②涉及的表
[1]light_cones
[2]roles

③表连接涉及字段：
light_cones.name = roles.light_cone

④删除条件字段描述
获取用户从前端传回的 uid 和 name 两个属性

字段	规则
light_cones.owner_uid	light_cones.owner_uid = uid
light_cones.name	light_cones.name = name

⑤代码：

```python
def Light_cones_delete():
    data = request.get_json()
    uid = data.get('uid')
    name = data.get('name')

    try:
        connect = get_db_connection()
        cursor = connect.cursor()
        connect.autocommit = False

        # 参数化查询删除操作
        delete_sql = """
            DELETE FROM light_cones
            WHERE owner_id = %s
            AND name = %s
        """
        cursor.execute(delete_sql, params: (uid, name))
        affected_rows = cursor.rowcount

        # 提交事务
        connect.commit()
```

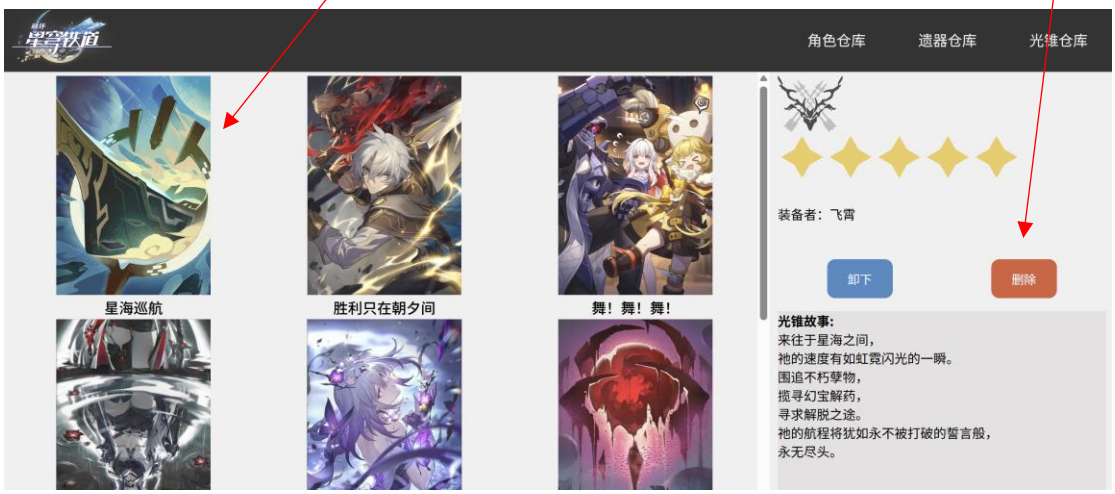
用户从前端传回 uid 和 name 数据之后，建立数据库连接，编写 mysql 的删除语句，然后执行，成功后要提交事务表示已完成。

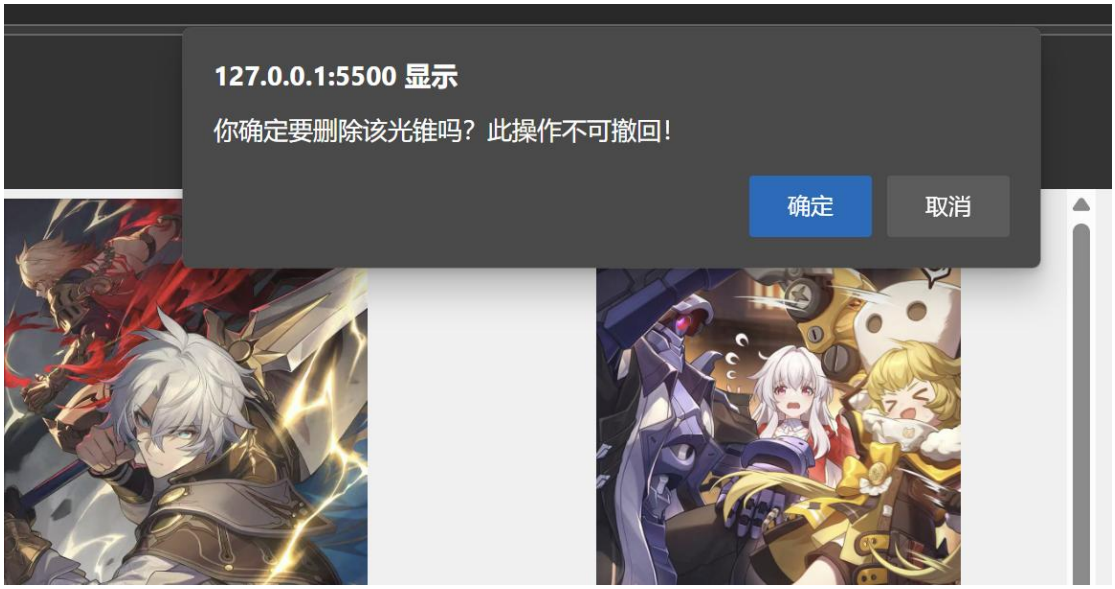
⑥程序演示：

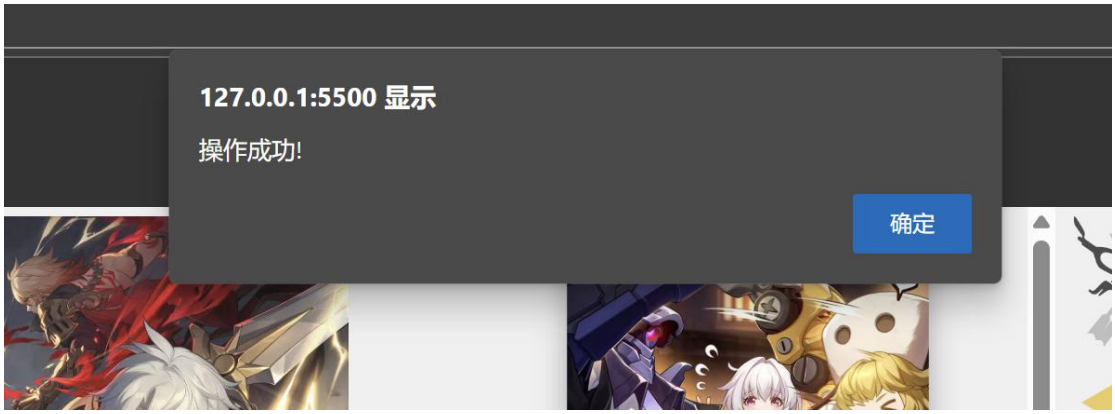
[1]

我登录我的手机号和密码来到我的游戏账户内，然后点击查看飞霄这个角色的相关信息，发现其目前装备了这个光锥

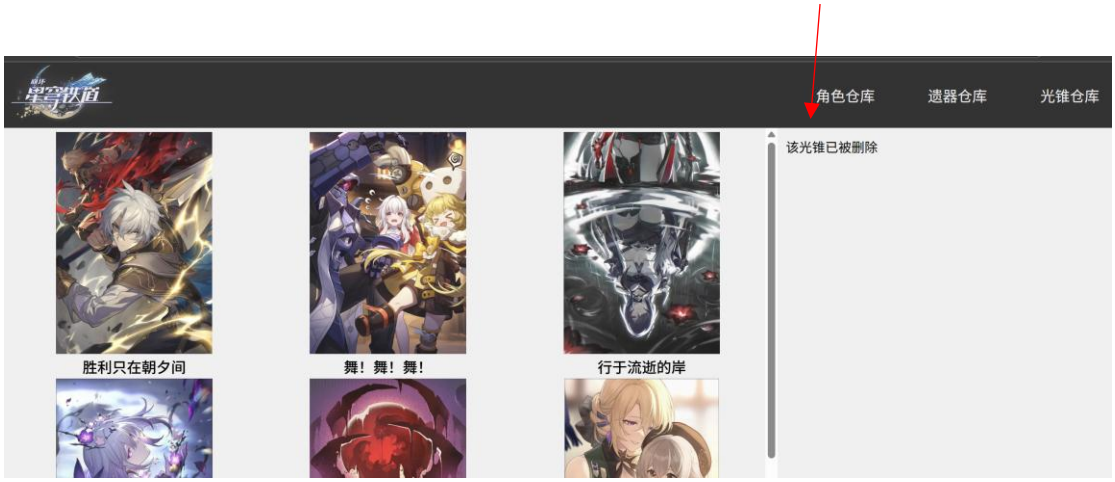
[2]
然后到光锥仓库界面下找到这个光锥，该光锥显示已经被飞霄装备，我们点击删除按钮：



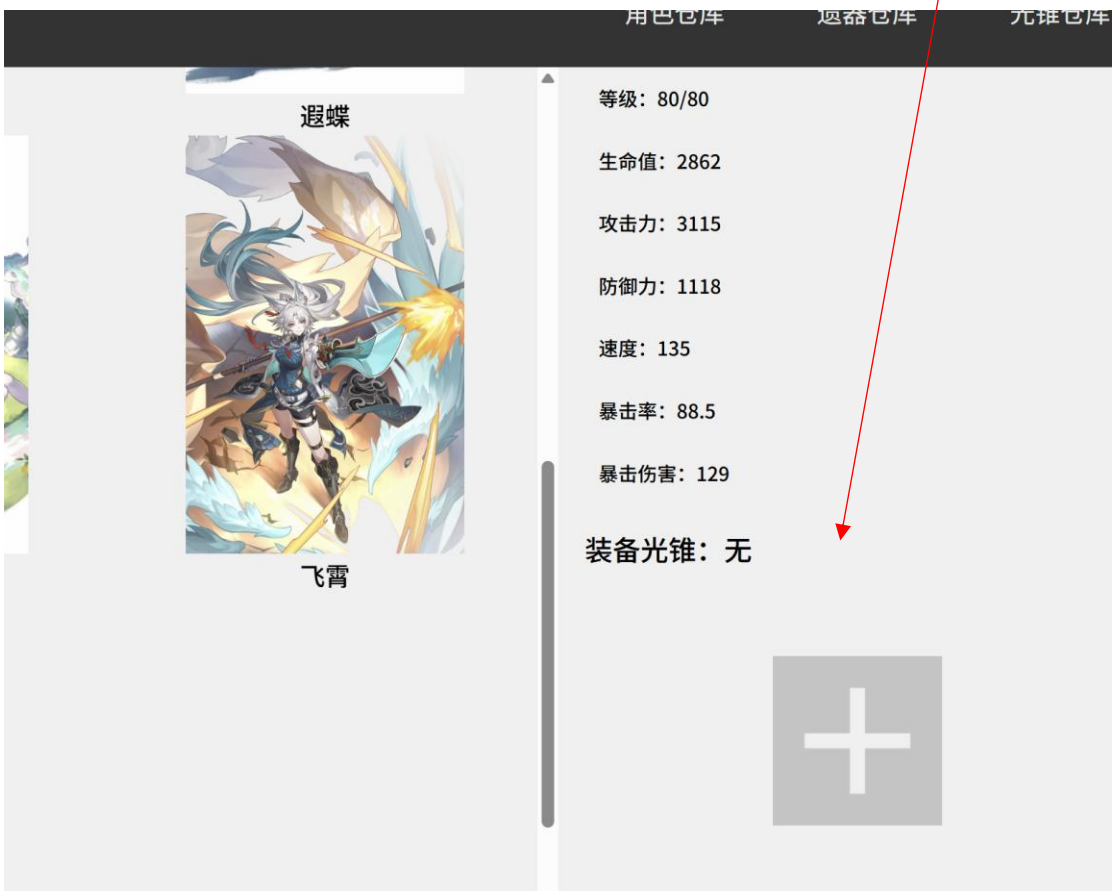




[3]操作成功之后可以查看，发现刚才那个光锥已经不存在了：

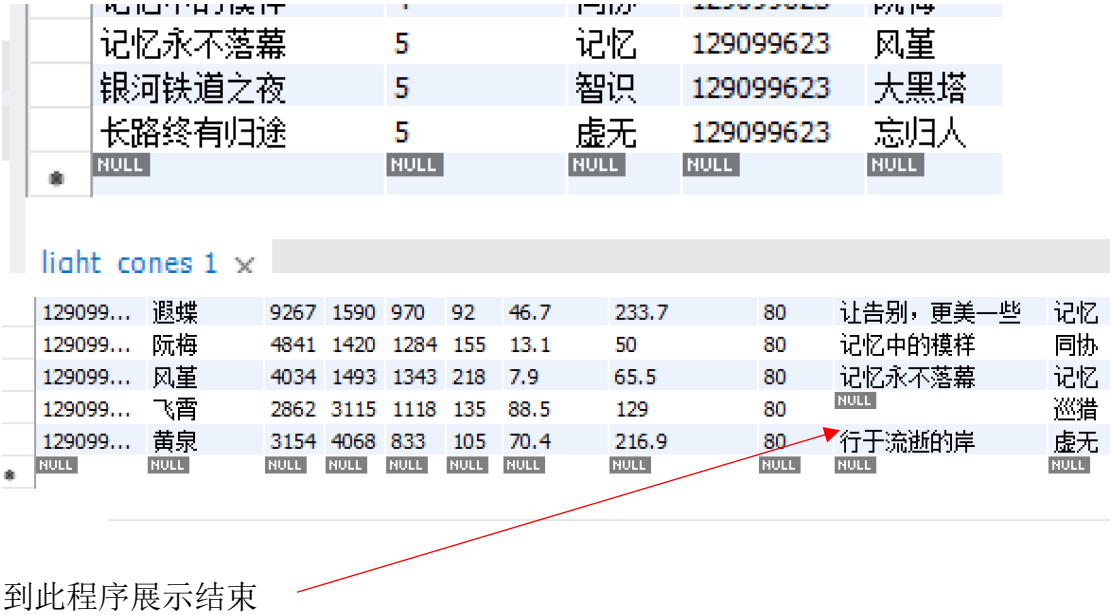


[4]再回到角色页面查看，点击飞霄的主页，我们发现这个光锥的确已经被删除，角色目标装备的光锥为空：



同时显示一个加号表示可以装备别的光锥。

[5]看到数据库内部，会发现相关信息的确已经被删除：



到此程序展示结束

5.触发器控制下的添加操作

①功能描述
玩家可以在遗器仓库里自主获取新的遗器，在数据库里为在表中添加进新的一行，需要触发器进行控制

②触发器描述
在玩家执行插入操作之前：
[1]检查所有字段是否为 null，或者有空字符串””
[2]检查 main_entry 和 secondary_entry1，secondary_entry2，secondary_entry3，secondary_entry4 这五个字段中属性值有没有负值的

一旦违背[1][2]，就拒绝插入

③涉及的表：
my_relics

④输入数据：

字段	规则
owner_uid	受外键约束，不能为 null 且不能为””
name	受外键约束，不能为 null 且不能为””
id	不能为 null 且不能为””
body	不能为 null 且不能为””
main_entry	不能为 null 且不能为””，同时不能出现负值
secondary_entry1	不能为 null 且不能为””，同时不能出现负值
secondary_entry2	不能为 null 且不能为””，同时不能出现负值
secondary_entry3	不能为 null 且不能为””，同时不能出现负值

secondary_entry4	不能为 null 且不能为””，同时不能出现负值
package	不能为 null 且不能为””

⑤插入操作源码：

```
@app.route( rule: '/insert_new_relic', methods=['POST'])
def Insert_new_relic():
    # 获取到前端发送来的玩家uid
    data = request.get_json()
    owner_uid = int(data.get('owner_uid'))
    name = data.get('name')
    id = data.get('id')
    body = data.get('body')
    main_entry = data.get('main_entry')
    secondary_entry1 = data.get('secondary_entry1')
    secondary_entry2 = data.get('secondary_entry2')
    secondary_entry3 = data.get('secondary_entry3')
    secondary_entry4 = data.get('secondary_entry4')
    package = data.get('package')
    user = ""
    try:
        #尝试建立连接
        connect = get_db_connection()
        cursor = connect.cursor()

        insert_query = """
                INSERT INTO my_relics (
                    owner_uid,
                    name,
```

```
def Insert_new_relic():
                    secondary_entry2,
                    secondary_entry3,
                    secondary_entry4,
                    package
                ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
                """
        values = (
            owner_uid,
            name,
            id,
            body,
            main_entry,
            secondary_entry1,
            secondary_entry2,
            secondary_entry3,
            secondary_entry4,
            package
        )
        print(values)
        cursor.execute(insert_query, values)
        connect.commit()
```

⑥触发器源码：

```
CREATE DEFINER=`root`@`localhost` TRIGGER `my_relics_BEFORE_INSERT` BEFORE INSERT ON `my_
    -- 检查所有字段是否为空
    IF (
        NEW.owner_uid = '' OR
        NEW.name = '' OR
        NEW.id = '' OR
        NEW.body = '' OR
        NEW.main_entry = '' OR
        NEW.secondary_entry1 = '' OR
        NEW.secondary_entry2 = '' OR
        NEW.secondary_entry3 = '' OR
        NEW.secondary_entry4 = '' OR
        NEW.package = ''
    ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '存在一个或多个字段为违规空值';
    END IF;


    -- 检查冒号后的负号
    IF (
        NEW.main_entry REGEXP '：[^：]*[-—][0-9]' OR   -- 中文冒号「：」
        NEW.secondary_entry1 REGEXP '：[^：]*[-—][0-9]' OR
        NEW.secondary_entry2 REGEXP '：[^：]*[-—][0-9]' OR
        NEW.secondary_entry3 REGEXP '：[^：]*[-—][0-9]' OR
        NEW.secondary_entry4 REGEXP '：[^：]*[-—][0-9]'
    ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '属性值不能包含负值';
    END IF;
END
```

⑦程序演示——不违背

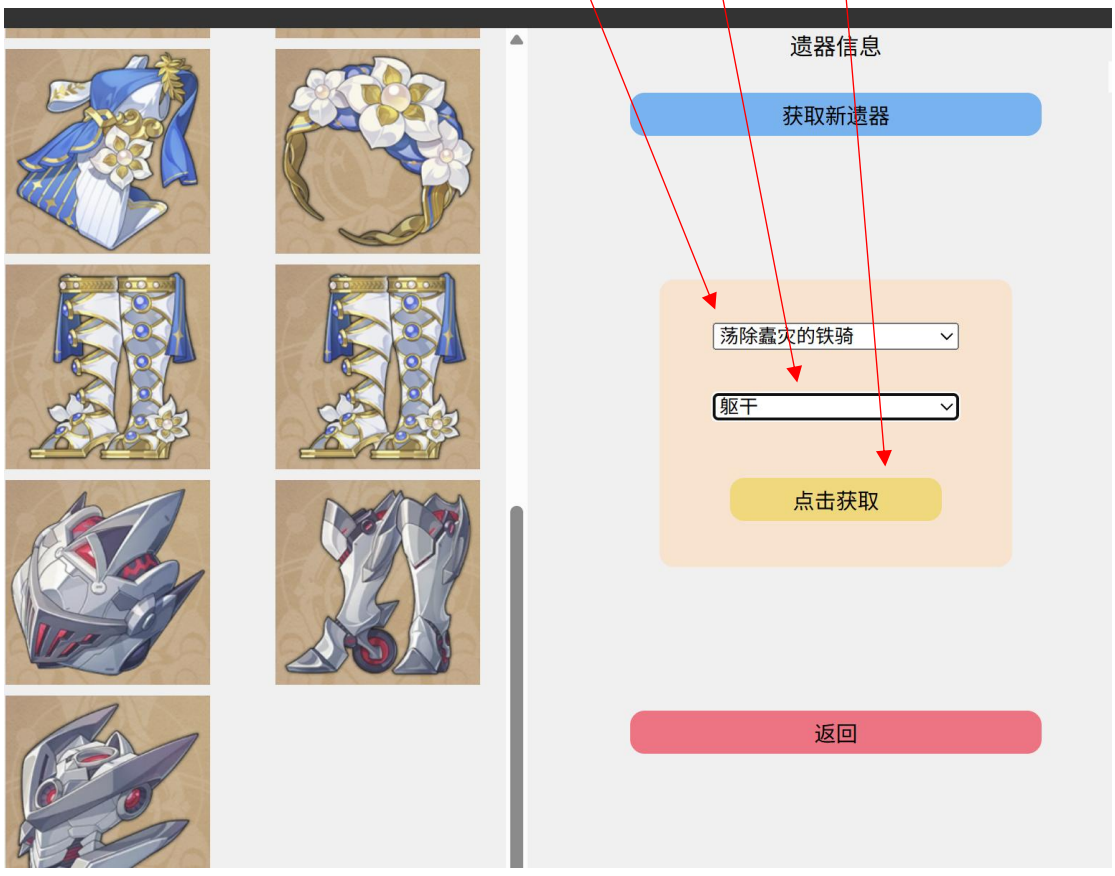
当插入的各个字段都符合规则和触发器的过滤规则时，能够插入

```
INSERT INTO my_relics (
                    owner_uid,
                    name,
                    id,
                    body,
                    main_entry,
                    secondary_entry1,
                    secondary_entry2,
                    secondary_entry3,
                    secondary_entry4,
                    package
) VALUES (129099623, "铁骑的银影装甲", 3,"手部", "攻击力：  43.2%", "攻击力：  35", "攻击力：  11.2%", "暴击伤害：  11.6%", "效果
```

22	16:34:44	INSERT INTO my_relics (	owner_uid,	name,	id,	...	1 row(s) affected

表现在前端就是：
[1]打开遗器界面，点击获取新遗器按钮

[2]在遗器选择界面选择你想要获取的遗器套装和部位，点击获取



[3]获取成功后，可以看到多出了一件我们刚刚选择的遗器，并且可以看到它的各种信息，注意，这些信息还原了原游戏，是通过随机分配的形式获取主词条和 4 个副词条的属性和值的。

到这里添加就成功了。

⑧程序演示——违背

由于前端代码本身在随机生成，用户输入的功能上已经做了充足的过滤，通过前端不可能能输入不合法的数据到数据库。所以直接在 Mysql workben 中演示违背的情况：

[1]空字符串违背：

可以看到主词条那个字段的值是一个空字符串""，这种数据是我们不允许的，所以当执行该命令的时候，触发器报错，并打印存在一个或多个字段为违规空值。

```
                    owner_uid,
                    name,
                    id,
                    body,
                    main_entry,
                    secondary_entry1,
                    secondary_entry2,
                    secondary_entry3,
                    secondary_entry4,
                    package
) VALUES (129099623, "铁骑的银影装甲", 3,"手部", "", "攻击力：  35", "攻击力：  11.2%", "暴击
```

```
24  16:45:56  INSERT INTO my_relics (  owner_uid,  name,  id,  ...  Error Code: 1644. 存在一个或多个字段为违规空值
```

[2]存在负值违背

当输入的一个词条的属性值为负值的时候，触发器报错，显示属性值不能包含赋值。

```
INSERT INTO my_relics (
                    owner_uid,
                    name,
                    id,
                    body,
                    main_entry,
                    secondary_entry1,
                    secondary_entry2,
                    secondary_entry3,
                    secondary_entry4,
                    package
) VALUES (129099623, "铁骑的银影装甲", 3,"手部", "攻击力：  -43.2%", "攻击力：  35", "攻击力：
```

```
25  16:47:32  INSERT INTO my_relics (  owner_uid,  name,  id,  ...  Error Code: 1644. 属性值不能包含负值
```

6.存储过程控制下的更新操作

①功能表述：

在角色页面中，玩家和实现为某一个角色装备光锥仓库中的某一个光锥的操作，这个过程由存储过程控制。

②存储过程功能描述：

更新玩家角色表的装备光锥字段和玩家光锥表的使用者字段前，要先检查前者五否为 null，且后者是否为”无”。如果不是，则报错；如果是，那就执行更新操作，修改这两个表的内容。

③涉及的关系表
[1]light_cones
[2]roles

④表连接涉及字段
[1]light_cones.user = roles.name
[2]roles.light_cone = light_cones.name

⑤更改字段

字段	规则
light_cones.user	更新前，该字段必须为”无”
roles.light_cone	更新前，该字段必须为 null

⑥更新代码：

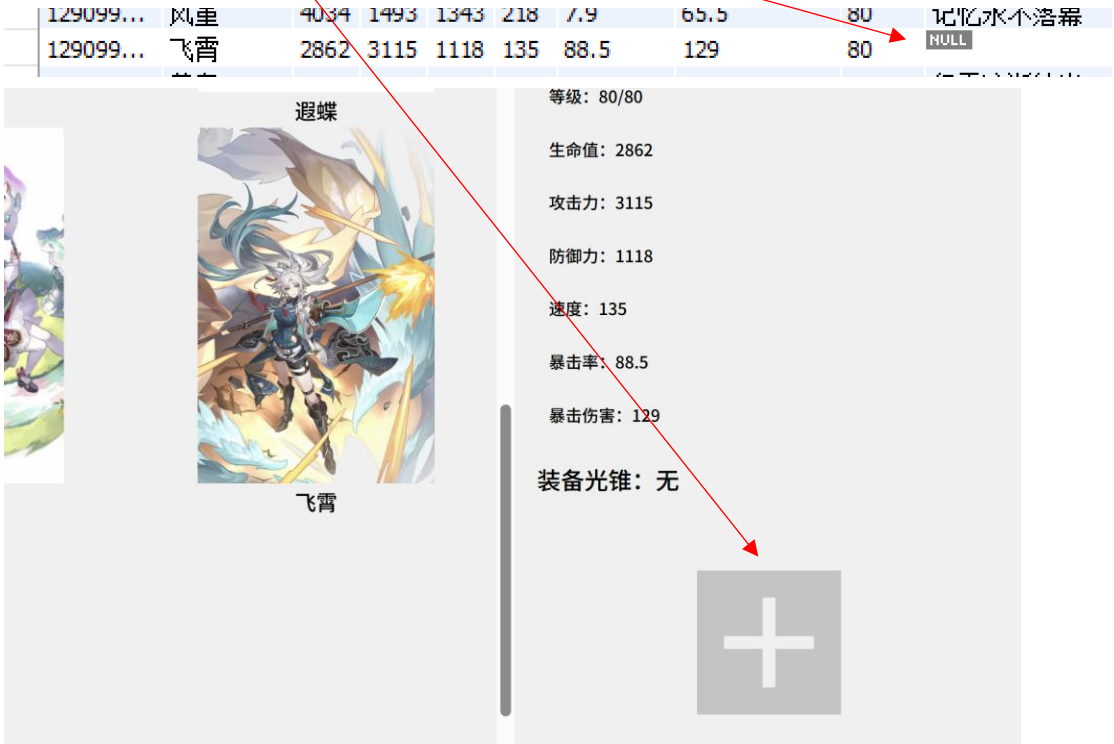
```python
@app.route( rule: '/Light_cones_equip', methods=['POST'])
def Light_cones_equip():
    data = request.get_json()
    uid = data.get('uid')
    user = data.get('user')
    name = data.get('name')

    try:
        connect = get_db_connection()
        cursor = connect.cursor()

        # 调用存储过程执行安全装备操作
        cursor.callproc( procname: "EquipLightCone",  args: (uid, name, user
        connect.commit()

        return jsonify({
            'success': True,
            'message': '光锥装备成功'
        })

    except mysql.connector.Error as err:
        # 提取存储过程的定制错误消息
        error_message = f'数据库错误: {err.msg}'
```

⑦存储过程执行源码：

```
DELIMITER $$EquipLightCone

CREATE PROCEDURE `EquipLightCone`(
    IN p_uid INT,
    IN p_name VARCHAR(50),
    IN p_user VARCHAR(15)
)
BEGIN
    DECLARE v_lightcone_user VARCHAR(15);
    DECLARE v_role_lightcone VARCHAR(50);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    START TRANSACTION;

    -- 检查光锥表：当前user值必须为"无"
    SELECT `user` INTO v_lightcone_user
    FROM `light_cones`
    WHERE `owner_id` = p_uid
    AND `name` = p_name
    FOR UPDATE;

    IF v_lightcone_user != '无' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = '光锥已被其他角色装备';
    END IF;

    -- 检查角色表：当前Light_cone值必须为NULL
    SELECT `Light_cone` INTO v_role_lightcone
    FROM `roles`
    WHERE `owner_uid` = p_uid
    AND `role_name` = p_user
    FOR UPDATE;
```

```
    IF v_role_lightcone IS NOT NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = '角色已装备其他光锥';
    END IF;

    -- 执行光锥更新操作
    UPDATE `light_cones`
    SET `user` = p_user
    WHERE `owner_id` = p_uid
    AND `name` = p_name;

    -- 执行角色装备更新
    UPDATE `roles`
    SET `Light_cone` = p_name
    WHERE `owner_uid` = p_uid
    AND `role_name` = p_user;

    COMMIT;
END
```

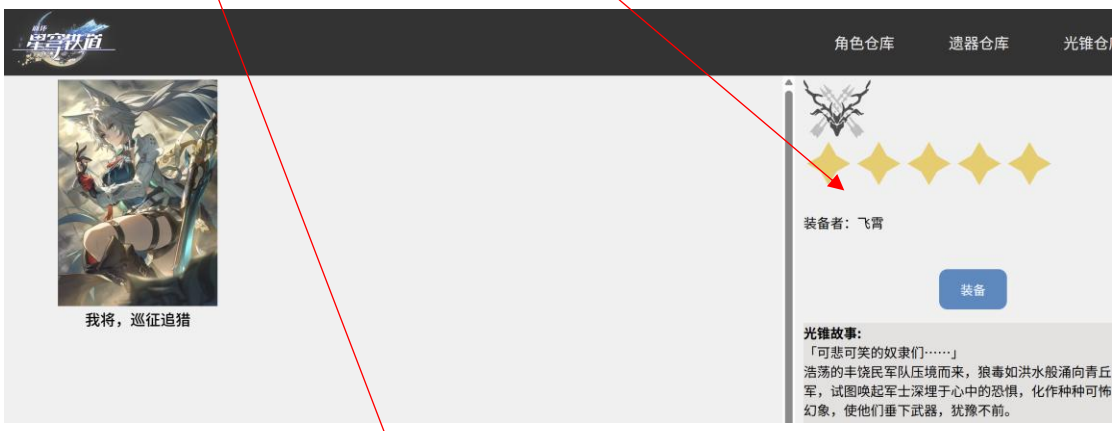
⑧程序演示——不违背存储过程：
前文我们删除了角色飞霄原本装备的光锥，所以现在该角色没有装备光锥，所以我们以该角色为例子进行演示
[1]在数据库中确认飞霄没有装备任何光锥，并且打开角色界面，看到飞霄没有装备任何光锥，点击“+”号进行装备页面

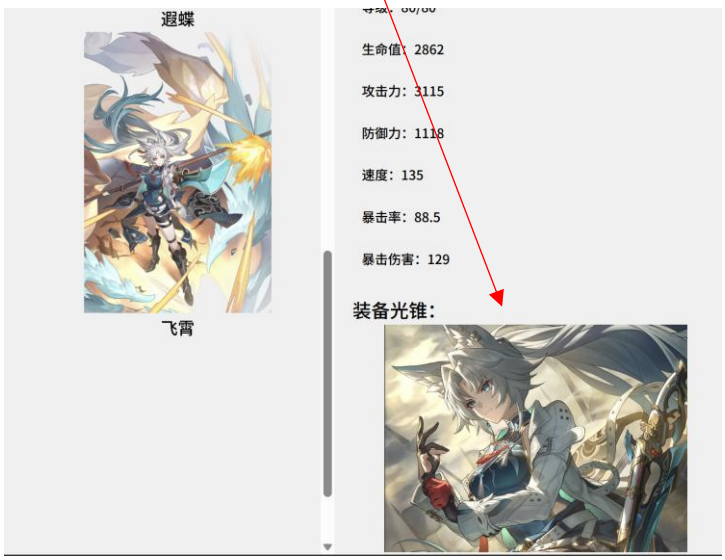


[2]进入装备页面后，看到有一个光锥是当前可装备的，在数据库中确认该光锥的确没有其他人装备，然后点击装备

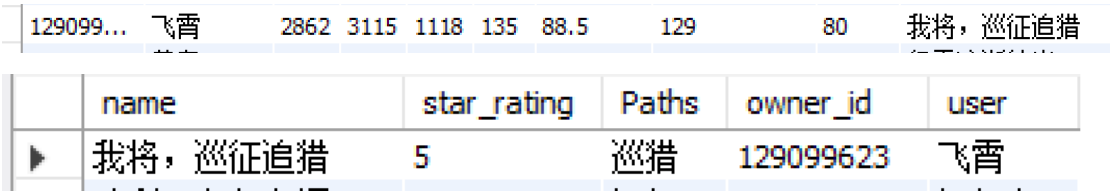
	name	star_rating	Paths	owner_id	user
▶	我将，巡征追猎	5	巡猎	129099623	无

[3]点击装备点击确定后，可以看到装备者的确变成了飞霄，并且在角色页面中，也能看到该光锥被装备了：





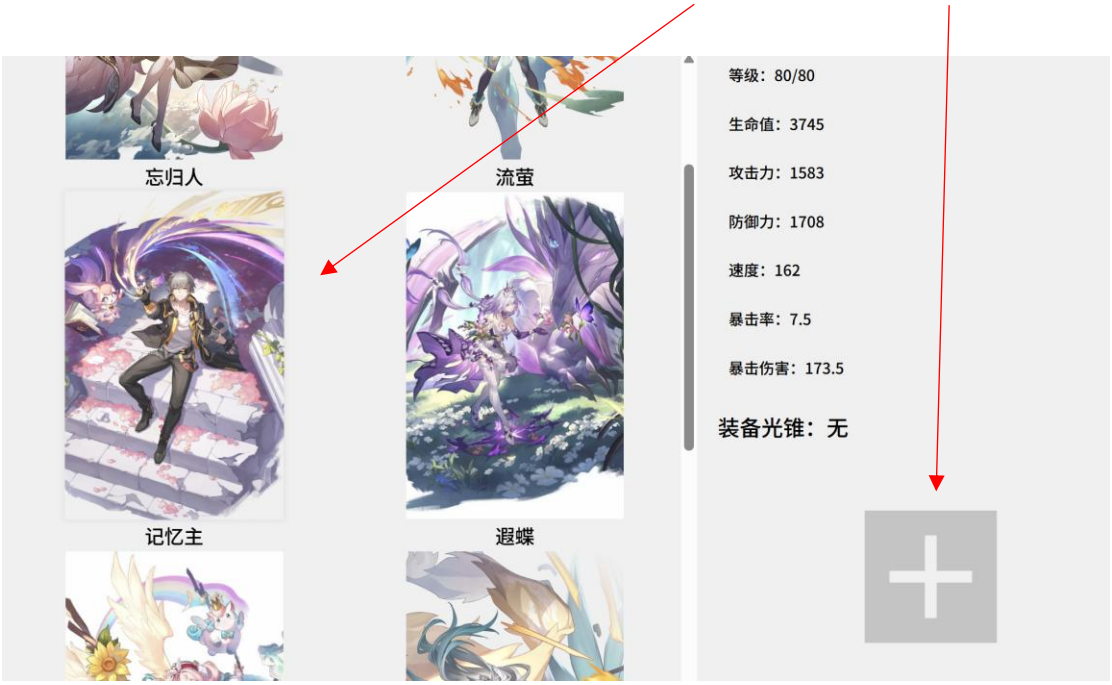
然后数据库中的数据也对应的改变了：



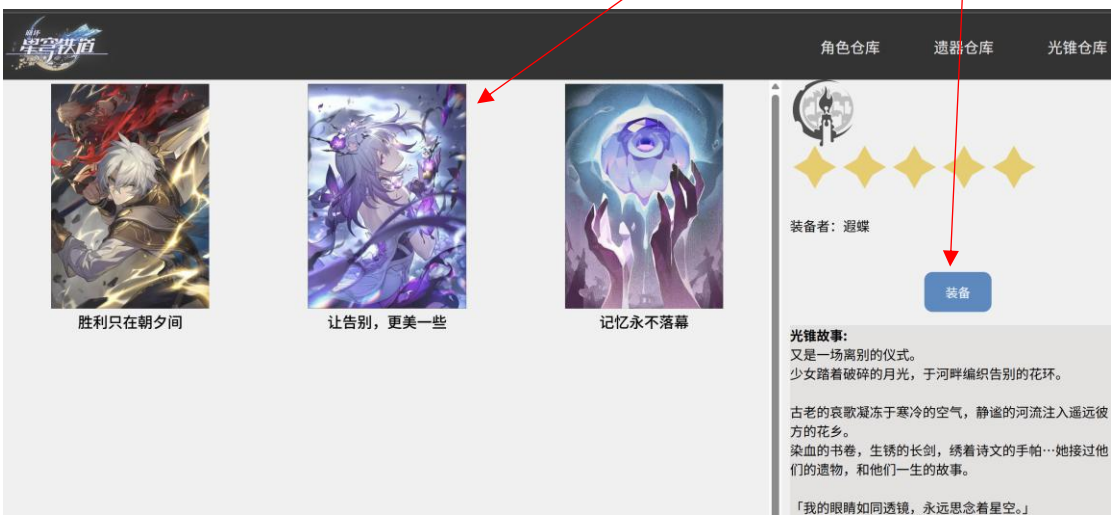
	name	star_rating	Paths	owner_id	user
▶	我将，巡征追猎	5	巡猎	129099623	飞霄

⑨程序演示——违背存储过程

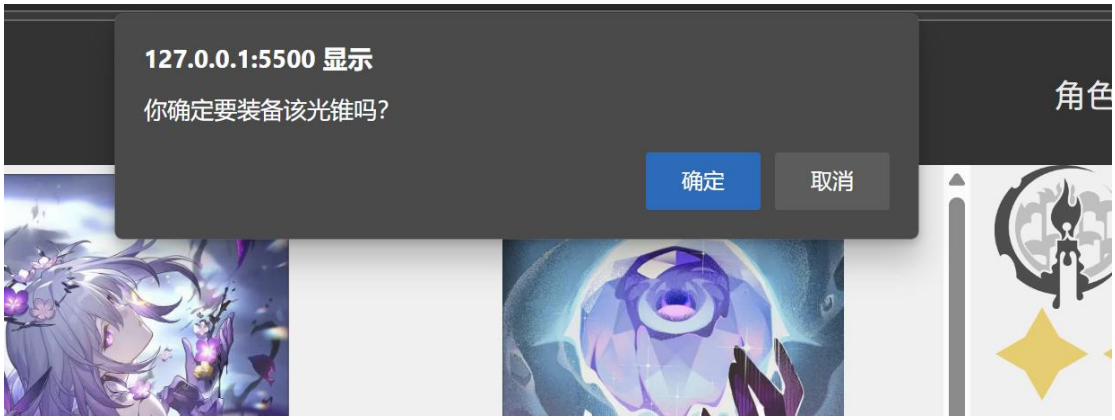
[1]我们在角色页面找到另外一个还没有装备光锥的角色，点击进入装备页面：

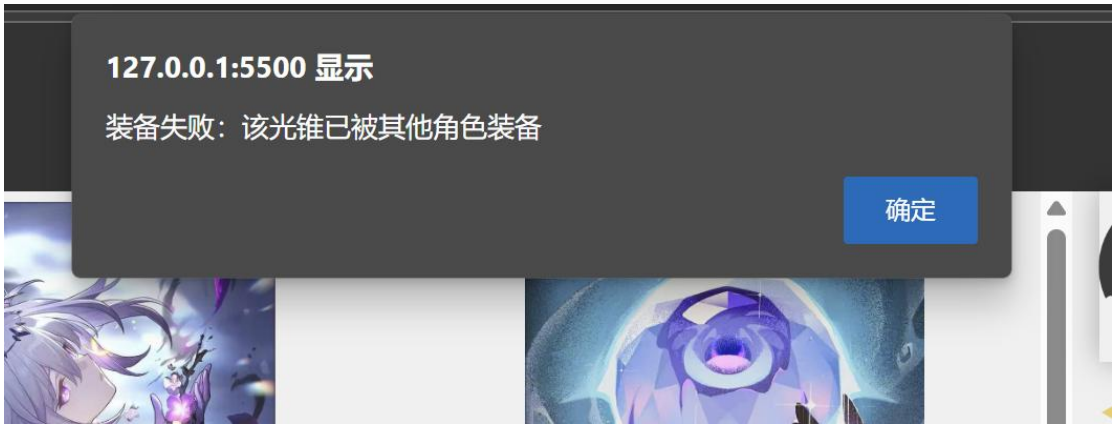


[2]然后在这里，我们故意找到一个已经有了装备者的光锥，并且点击装备



[3]然后能看到装备失败的结果，点 F12 能看到数据库传来的报错，因为这违背了存储过程控制下的更新操作，所以报错并拒绝执行。





7.含有视图的查询操作

①操作功能描述：
因为在玩家个人拥有的光锥表 light_cones 中，是没有光锥的背景故事的，
该字段 story 只在所有光锥这个表 all_light_cones 中有，为避免在一次点击下的二次查询，我们创建视图来简化流程

②视图功能描述：
创建视图 lightcones_details,将 light_cones 表中的基础信息和 all_light_cones 中的 story 信息连接，以展示每个光锥的详细信息。

③涉及的关系表：
[1] all_light_cones
[2] light_cones

④表连接字段：
light_cones.name = all_light_cones.name

⑤创建视图代码：

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `light_cones_view` AS
    SELECT
        `lc`.`owner_id` AS `owner_id`,
        `lc`.`name` AS `name`,
        `lc`.`star_rating` AS `star_rating`,
        `lc`.`Paths` AS `Paths`,
        `lc`.`user` AS `user`,
        `alc`.`story` AS `light_cone_story`
    FROM
        (`light_cones` `lc`
        JOIN `all_light_cones` `alc` ON ((`lc`.`name` = `alc`.`name`)))
```

⑥查询代码

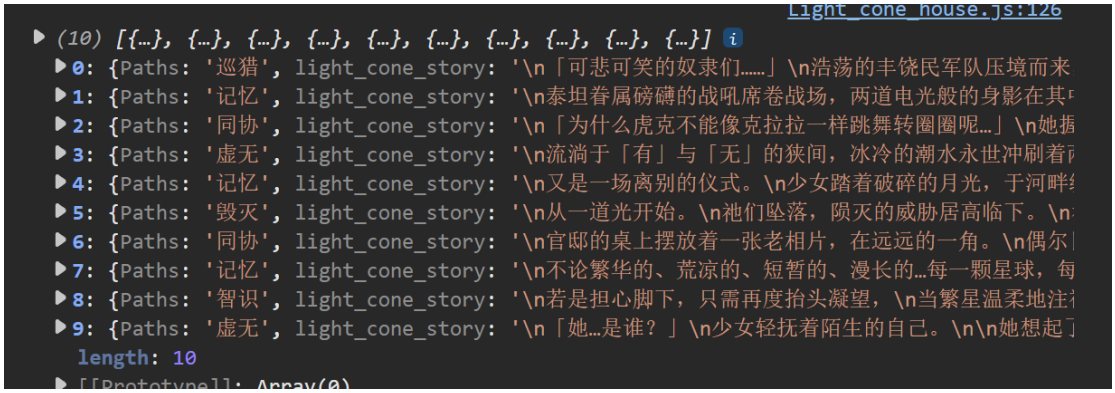
```python
@app.route( rule: '/get_user_light_cones', methods=['POST'])
def get_user_light_cones():
    data = request.get_json()
    uid = int(data.get('uid'))

    try:
        # 获取数据库连接
        connect = get_db_connection()
        cursor = connect.cursor(dictionary=True)  # 使用字典格式返回结果

        # 查询视图数据
        query = """
            SELECT
                name,
                star_rating,
                Paths,
                user,
                light_cone_story
            FROM light_cones_view
            WHERE owner_id = %s
        """
        cursor.execute(query,  params: (int(uid),))

        light_cones = cursor.fetchall()
```

⑦程序演示：

我们事先在前端 javascript 代码中添加一行打印的代码，方便我们进行查看：

```
console.log(data.result);
```

当我们打开光锥界面时，点开 F12 时，可以看到每一个光锥的详细信息都被打印出来了：



演示完毕。