

# 数据库工程作业

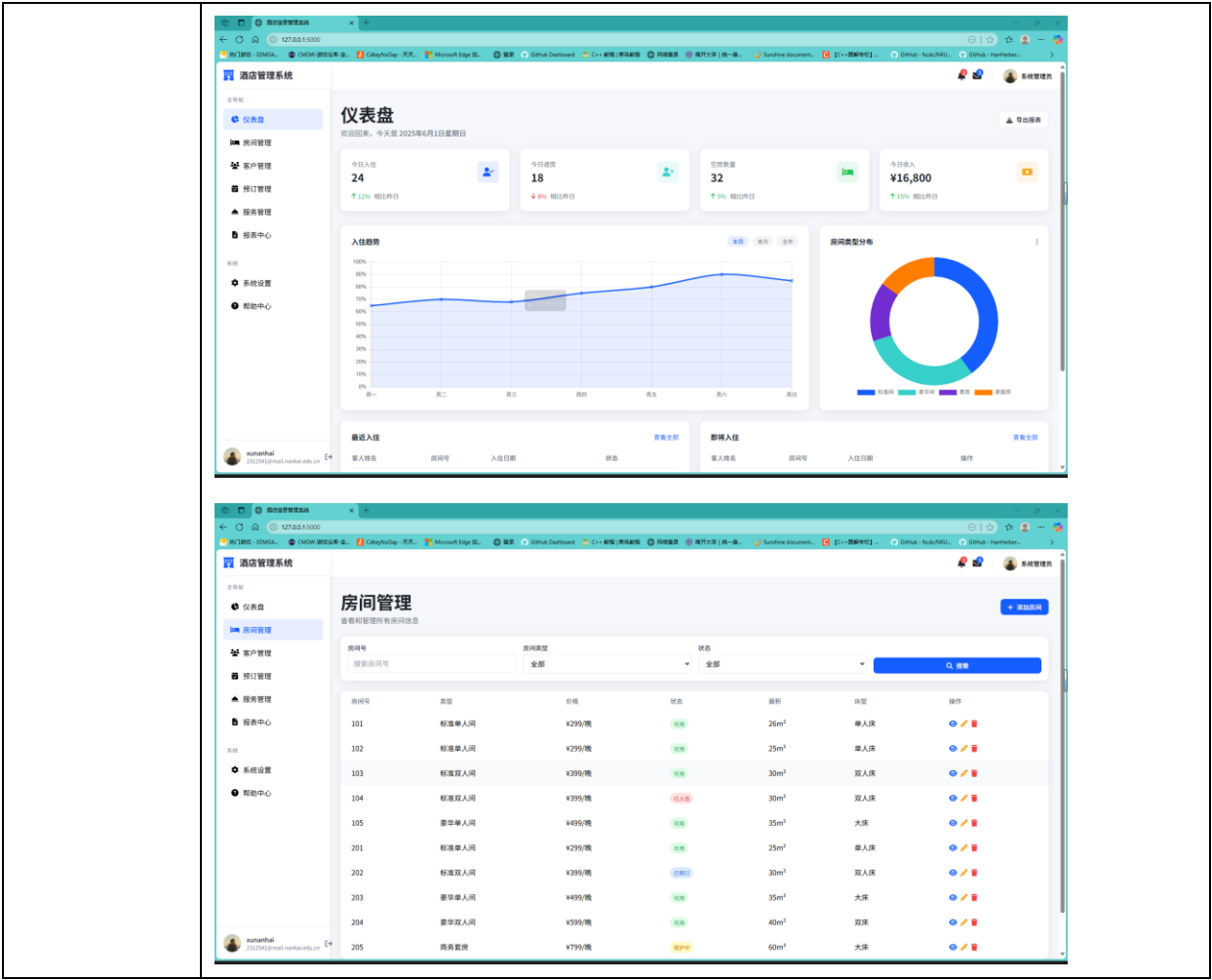
要求:

- 1. 完成一个小型的数据库信息管理系统（或部分功能），并填写工程作业报告；程序和报告请在规定时间之内上传。
- 2. 开发模式（B/S 或 C/S）、开发高级语言任选，后台数据库使用大型数据库管理系统（SQL Server、Oracle、MySQL 等），不要使用桌面数据库。
- 3. 报告中所列举的四种操作，每种操作举一个例子即可。
- 4. 作业成绩按照报告中的标准评分，程序只实现报告中涉及的部分即可。
- 5. 作业完成后，请将工程作业报告和程序打包提交给助教老师，并联系助教老师进行系统说明和演示，回答相关问题。

## 工程作业报告

1. 项目信息（10 分）

学号	2312541	姓名	徐南海	专业	物联网工程
项目名称	酒店服务管理系统				
必备环境	Python, MySQL				
系统主要功能简介（4 分）	这是一个基于 Flask 框架开发的酒店管理系统，包含客房管理、预订管理、客户管理和服务管理模块				
系统主要页面截图（6 分）					



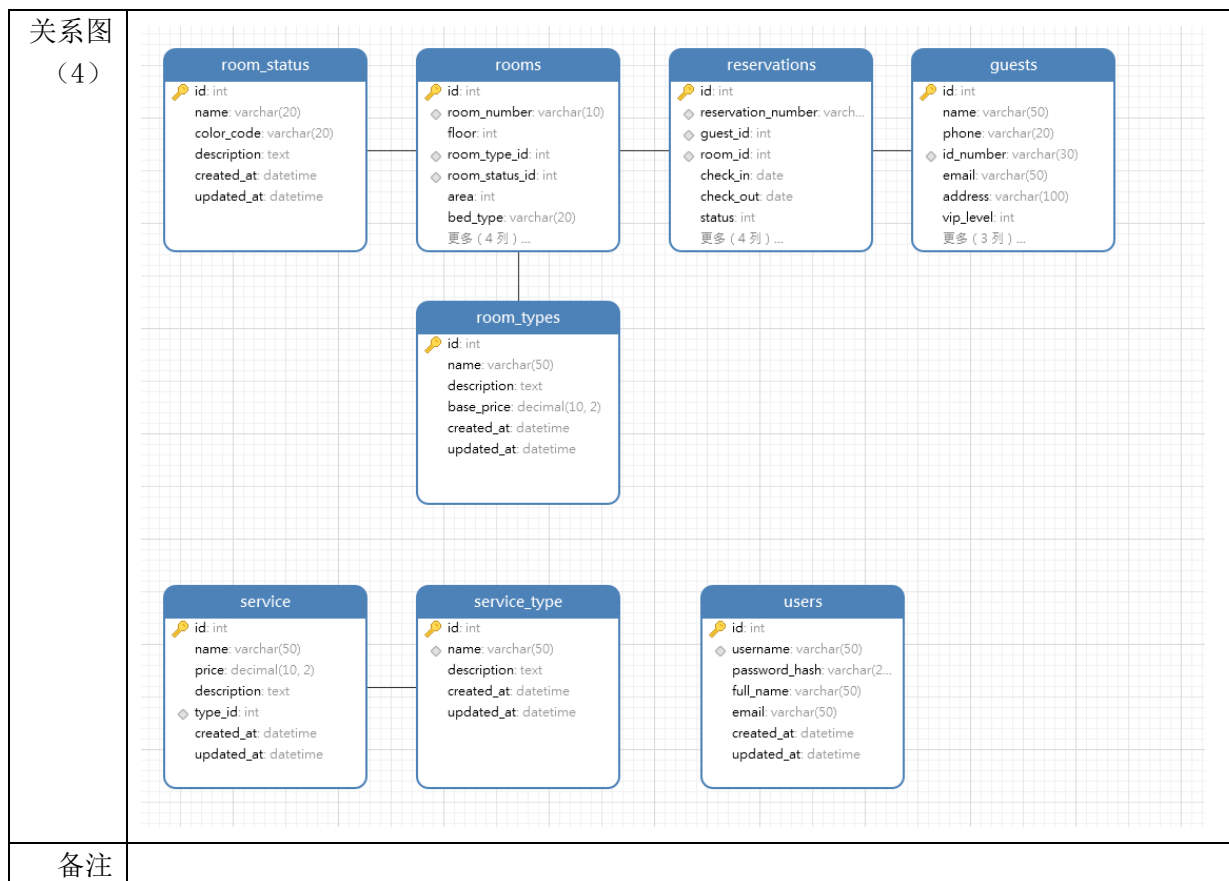
2. 系统配置（10分）

说明		(2分)请说明系统配置情况(后台数据库,高级语言); (8分)请使用连接串连接高级语言和数据库,并分析字符串的各个部分。			
配置 步骤 2分	DBMS	1. MySQL			
		2. ....			
	高级 语言	1. Python			
		2. ....			
连接串 分析 (6分)		序号	名称	功能说明	取值
		1	SQLALC HEMY_D ATABAS E_URI	在我们的 config.py 文件中与我们的 MySQL 进行连接,从而进行配置	
		2			
		...			
		...			

连接串代码 (截屏) (2 分)	<pre># 数据库配置 SQLALCHEMY_DATABASE_URI = os.environ.get(     'DATABASE_URL',     'mysql+pymysql://root:123456@localhost:3306/hotel_db?charset=utf8mb4' )</pre>
备注	连接过程中，我们使用了对应数据库的账号和密码，对应到项目的数据库 hotel_db，使用 utf-8 的格式

### 3. 数据库设计 (14 分)

说明	<p>(10 分) 按照数据表的创建顺序，依次给出所涉及数据表的信息，其中参照字段以“(字段 1, 字段 2, ……, 字段 n)”的形式给出，被参照字段以“表名(字段 1, 字段 2, ……, 字段 n)”的形式给出；</p> <p>(4 分) 一般 DBMS 都可以为数据库生成关系图，请将该图片截屏并粘贴到表格中。</p>				
数据表 (10)	创建顺序	数据表名称	主键	参照属性	被参照表及属性
	1	room_types	id	无参照属性	
	2	room_status	id	无参照属性	
	3	rooms	id	room_type_id	room_types(id)
				room_status_id	room_status(id)
	4	guests	id	无参照属性	
	5	service_type	id	无参照属性	
	6	service	id	type_id	service_type(id)
	7	reservations	id	guest_id	guests(id)
				room_id	rooms(id)
	8	user	id	无参照属性	



#### 4. 含有事务应用的删除操作（13分）

说明	(1分) 简要说明该操作所要完成的功能; (2分) 该操作会涉及的表(必须含有两张或两张以上的关系表,同时以“表名”的形式给出) (1分) 表连接涉及字段描述(描述方式为“表 1. 属性=表 2. 属性”) (1分) 删除条件涉及的字段描述(以“表名. 属性=? ”形式给出) (4分) 实现该操作的关键代码(高级语言、SQL),截图即可;(其中如果删除语句中不包含任何形式的事务应用将扣除3分) (4分) 如何执行该操作,按所述方法能够正常演示程序则给分。
功能描述 (1分)	该操作用于删除已取消的预订记录。系统会先检查预订状态,只有已取消的预订(status=4)才能被删除,以确保数据的一致性和业务逻辑的正确性。
涉及的表 (2分)	reservations (预订表) guests (客户表) rooms (房间表)
表连接涉及字段	reservations.guest_id = guests.id reservations.room_id = rooms.id

(1分)

删除条件字段描述(1分)

字段	规则
reservations.id	确保预订状态为已取消（status=4）才能删除
reservations.status	

代码(4分)

```
@api_bp.route('/reservations/<int:reservation_id>', methods=['DELETE'])
def delete_reservation(reservation_id):
    reservation = Reservation.query.get_or_404(reservation_id)

    # 限制只能删除已取消的预订
    if reservation.status != 4:
        return jsonify({'error': '只能删除已取消的预订'}), 400

    try:
        db.session.delete(reservation)
        db.session.commit()
        return jsonify({'message': '预订已删除', 'reservation_id': reservation_id})
    except Exception as e:
        db.session.rollback()
        return jsonify({'error': '删除预订失败', 'details': str(e)}), 500
```

程序演示(4分)

首先我们看到 reservations 表内 id=3 的 status 值为 4，代表顾客已经取消预定

id	reservation_number	guest_id	room_id	check_in	check_out	status	total_price	deposit	created_at
1	R20251001	1	1	2025-06-01	2025-06-03	2	598.00	500.00	2025-06-01 23:41:31
2	R20251002	2	3	2025-06-05	2025-06-08	2	1197.00	1000.00	2025-06-01 23:41:31
3	R20251003	3	6	2025-06-10	2025-06-11	4	299.00	300.00	2025-06-01 23:41:31
4	R20251004	4	14	2025-06-11	2025-06-13	1	3998.00	2000.00	2025-06-01 23:41:31
5	R20251005	5	8	2025-06-01	2025-06-02	2	499.00	500.00	2025-06-01 23:41:31

此时我们进入网页界面去删除王五的预定信息

### 预订管理

查看和管理所有预订信息

房间号

预订状态

入住日期

退房日期

搜索房间号

所有状态

yyyy/mm/日

yyyy/mm/日

搜索

预订编号	客户姓名	房间号	入住日期	退房日期	状态	操作
R20251001	张三	101	2025-06-01	2025-06-03	已确认	<a href="#">编辑</a> <a href="#">删除</a>
R20251002	李四	103	2025-06-05	2025-06-08	已确认	<a href="#">编辑</a> <a href="#">删除</a>
R20251003	王五	201	2025-06-10	2025-06-11	已取消	<a href="#">编辑</a> <a href="#">删除</a>
R20251004	赵六	304	2025-06-11	2025-06-13	待确认	<a href="#">编辑</a> <a href="#">删除</a>
R20251005	钱七	203	2025-06-01	2025-06-02	已确认	<a href="#">编辑</a> <a href="#">删除</a>

共 5 条记录

得到如下结果

系统管理员

### 预订管理

查看和管理所有预订信息









房间号

预订状态

入住日期

退房日期

搜索

预订编号	客户姓名	房间号	入住日期	退房日期	状态	操作
R20251001	张三	101	2025-06-01	2025-06-03	已确认	 
R20251002	李四	103	2025-06-05	2025-06-08	已确认	 
R20251004	赵六	304	2025-06-11	2025-06-13	待确认	 
R20251005	钱七	203	2025-06-01	2025-06-02	已确认	 

共 4 条记录

预订删除成功

再查看表内发现删除成功

id	reservation_number	guest_id	room_id	check_in	check_out	status	total_price	deposit	created_at
1	R20251001	1	1	2025-06-01	2025-06-03	2	598.00	500.00	2025-06-01 23:41:31
2	R20251002	2	3	2025-06-05	2025-06-08	2	1197.00	1000.00	2025-06-01 23:41:31
4	R20251004	4	14	2025-06-11	2025-06-13	1	3998.00	2000.00	2025-06-01 23:41:31
5	R20251005	5	8	2025-06-01	2025-06-02	2	499.00	500.00	2025-06-01 23:41:31

备注 实现了含有事务应用的删除操作。

### 5. 触发器控制下的添加操作（20 分）

说明	(1 分) 简要说明该操作所要完成的功能; (2 分) 简要说明该触发器所要完成的功能 (1 分) 该操作会涉及的表 (以 “表名” 的形式给出)。 (2 分) 该操作输入数据以及输入数据应该满足的条件, 如: 数值范围、是否为空; (6 分) 实现该操作的关键代码 (高级语言、SQL), 截图即可; (8 分) 如何执行该操作, 按所述方法能够正常演示程序则给分。	
功能描述 (1 分)	预订添加操作用于创建新的酒店预订记录, 包括客户信息、房间信息、入住时间、价格等关键信息。	
触发器描述 (2 分)	在添加预订时, 触发器会自动检查: 房间状态是否可用、预订时间是否与现有预订冲突、自动更新房间状态为“已预订”以及记录预订操作日志	
涉及的表 (1 分)	reservations (预订表) rooms (房间表) room_status (房间状态表) guests (客户表) reservation_logs (预订日志表)	
输入数据 (2 分)	字段	规则
	reservation_number	
	guest_id	
	room_id	
	check_in	通过检查入住时间是否匹配与状态逻辑是否合理来判断能否输入数据
	check_out	
	status	
	total_price	
	deposit	
	.....	.....
插入操作源码 (3 分)	(截屏)	

	<pre> def create_reservation():     data = request.get_json()      # 校验必填字段     required_fields = [         'reservation_number', 'guest_id', 'room_id',         'check_in', 'check_out', 'status',         'total_price', 'deposit'     ]     for field in required_fields:         if field not in data:             return jsonify({'error': f'缺少必填字段: {field}'}), 400      # 校验日期格式     try:         check_in = datetime.strptime(data['check_in'], '%Y-%m-%d')         check_out = datetime.strptime(data['check_out'], '%Y-%m-%d')     except ValueError:         return jsonify({'error': '日期格式错误, 正确格式: YYYY-MM-DD'}), 400      # 检查日期逻辑     if check_out &lt;= check_in:         return jsonify({'error': '退房时间必须晚于入住时间'}), 400      try:         # 创建新预订         new_reservation = Reservation(             reservation_number=data['reservation_number'],             guest_id=data['guest_id'],             room_id=data['room_id'],             check_in=check_in,             check_out=check_out,             status=data['status'],             total_price=data['total_price'],             deposit=data['deposit'],         )          db.session.add(new_reservation)         db.session.commit()          return jsonify({             'message': '预订创建成功',             'reservation_id': new_reservation.id,             'data': format_reservation(new_reservation)         }), 201      except Exception as e:         db.session.rollback()         return jsonify({'error': '创建预订失败', 'details': str(e)}), 500 </pre>
触发 器源 码 (3 分)	(截屏)



```

CREATE TRIGGER before_reservation_insert
BEFORE INSERT ON reservations
FOR EACH ROW
BEGIN
    DECLARE room_status INT;
    DECLARE existing_reservation INT;

    -- 检查房间状态
    SELECT room_status_id INTO room_status
    FROM rooms
    WHERE id = NEW.room_id;

    IF room_status != 1 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = '房间当前不可预订';
    END IF;

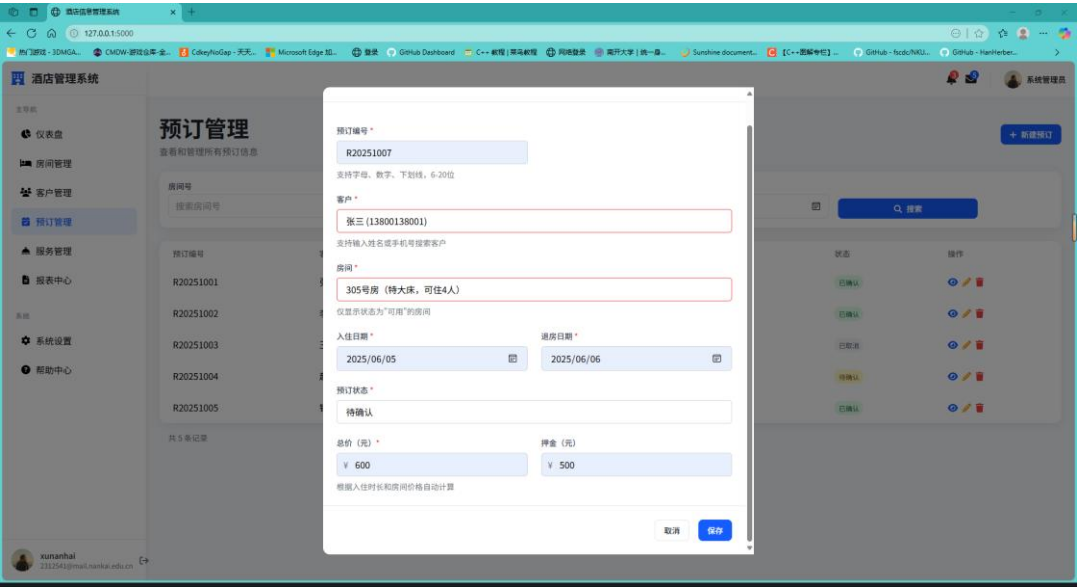
    -- 检查时间冲突
    SELECT COUNT(*) INTO existing_reservation
    FROM reservations
    WHERE room_id = NEW.room_id
    AND status != 4 -- 排除已取消的预订
    AND (
        (NEW.check_in < check_out AND NEW.check_out > check_in)
    );

    IF existing_reservation > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = '该时间段房间已被预订';
    END IF;

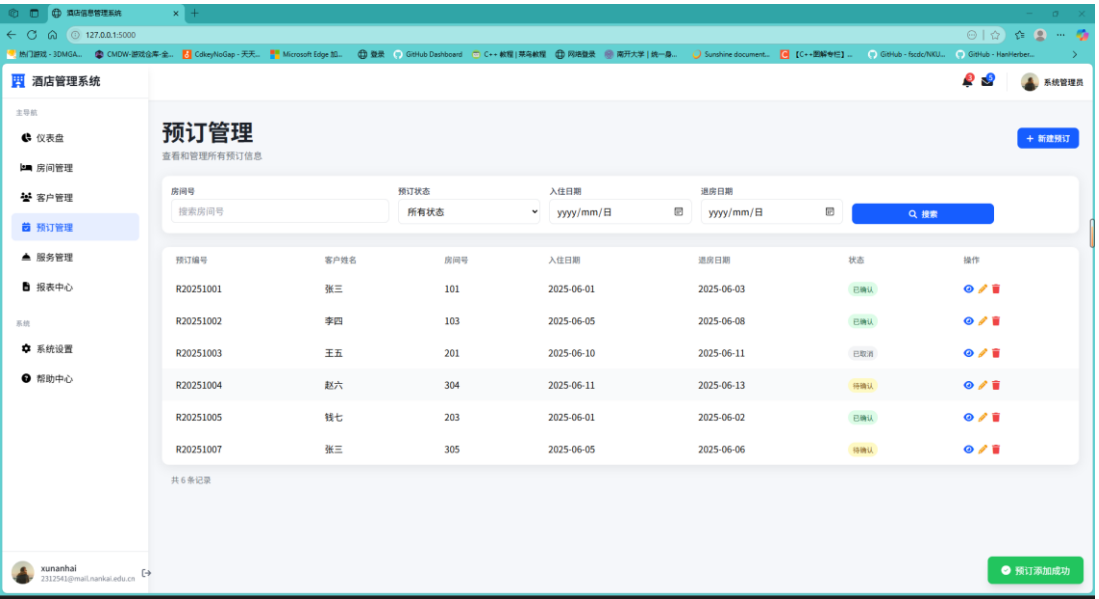
    -- 记录操作日志
    INSERT INTO reservation_logs (
        reservation_id,
        reservation_number,
        guest_id,
        room_id,
        action,
        action_time
    ) VALUES (
        NEW.id,
        NEW.reservation_number,
        NEW.guest_id,
        NEW.room_id,
        'INSERT',
        NOW()
    );
END //

```

说明：不违背触发器能够执行插入操作。  
首先我们插入数据



得到如下结果

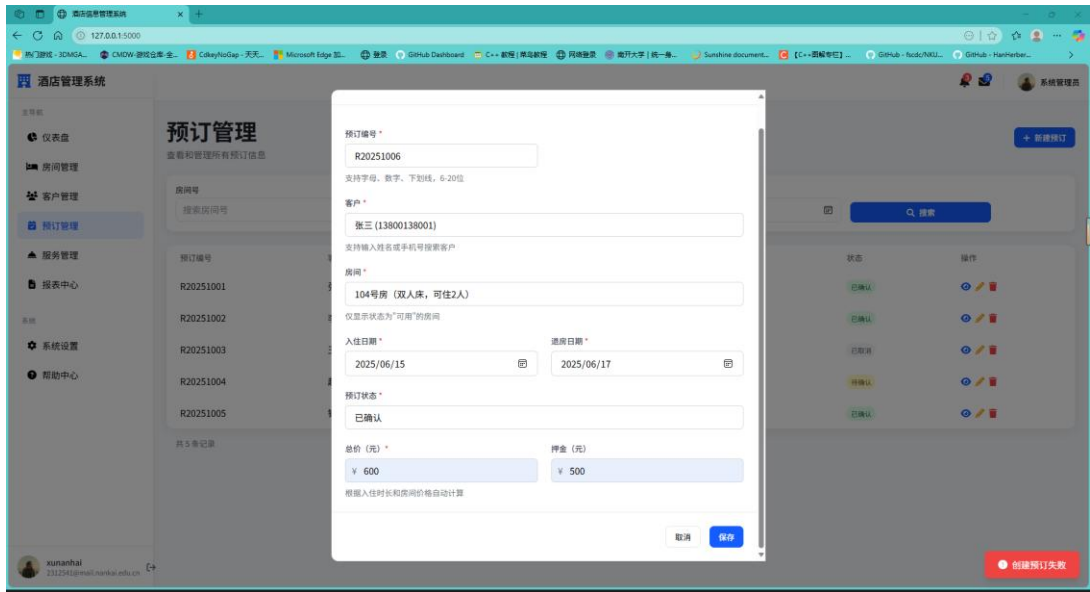


在表中发现成功插入

开始事务 文本 筛选 排序 列 导入 导出 数据生成 创建图表										
id	reservation_number	guest_id	room_id	check_in	check_out	status	total_price	deposit	created_at	updated_at
1	R20251001	1	1	2025-06-01	2025-06-03	2	598.00	500.00	2025-06-02 13:40:47	2025-06-02 13:40:47
2	R20251002	2	3	2025-06-05	2025-06-08	2	1197.00	1000.00	2025-06-02 13:40:47	2025-06-02 13:40:47
3	R20251003	3	6	2025-06-10	2025-06-11	4	299.00	300.00	2025-06-02 13:40:47	2025-06-02 13:40:47
4	R20251004	4	14	2025-06-11	2025-06-13	1	3998.00	2000.00	2025-06-02 13:40:47	2025-06-02 13:40:47
5	R20251005	5	8	2025-06-01	2025-06-02	2	499.00	500.00	2025-06-02 13:40:47	2025-06-02 13:40:47
7	R20251007	1	15	2025-06-05	2025-06-06	1	600.00	500.00	2025-06-02 12:06:39	2025-06-02 12:06:39

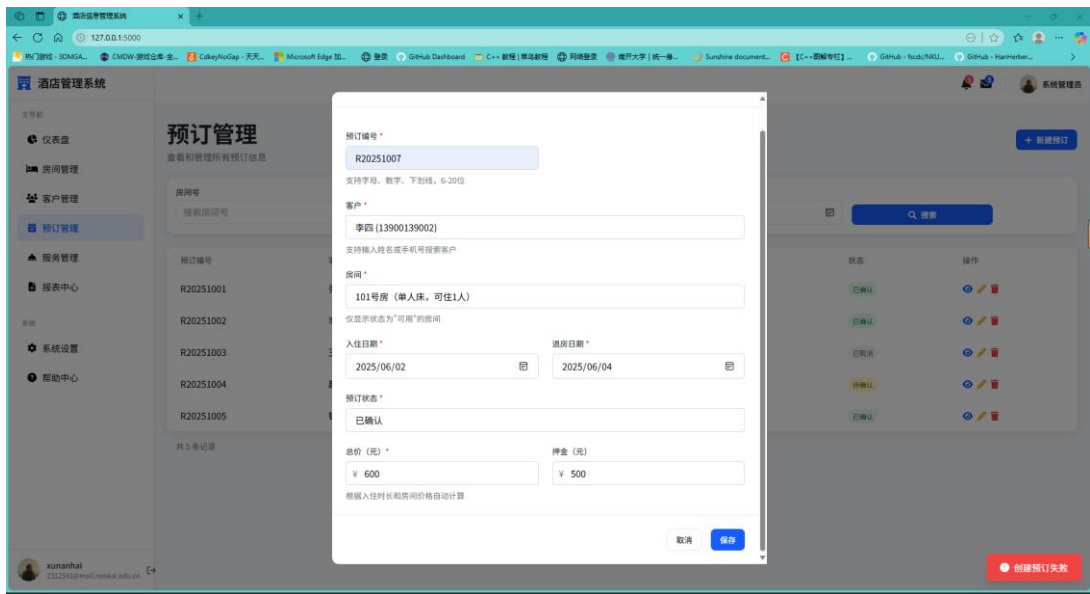
程序演示  
(4分)

说明：违背触发器要求，不能够执行插入操作，系统报错。  
一、 尝试预订一个状态不是“可用”的房间  
按照如下数据尝试预定房间



发现预定失败，说明 104 房间在已入住的状态下，我们无法预定

二、尝试预订一个时间冲突的房间  
按照如下数据尝试预定房间



发现预定失败，说明预定时间段内 101 房间已经有客人，我们无法预定

备注

这些错误信息是由触发器抛出的，触发器会阻止这些不符合业务规则的预订操作。这证明了触发器在数据库层面有效地保护了数据的一致性和业务规则的执行。

## 6. 存储过程控制下的更新操作（18 分）

说明	(1 分) 简要说明该操作所要完成的功能; (1 分) 简要说明该存储过程所要完成的功能; (2 分) 说明该操作涉及操作的表 (必须包含两张或两张以上的关系表, 以“表名形式”描述) (1 分) 表连接涉及字段描述 (描述方式为“表 1. 属性=表 2. 属性”) (2 分) 该操作会修改字段 (以“表名. 字段名”的形式给出), 以及修改规则, 如新数值的计算方法、在何种条件下予以修改等; (6 分) 实现该操作的关键代码 (高级语言、SQL), 截图即可; (5 分) 如何执行该操作, 按所述方法能够正常演示程序则给分。	
功能描述 (1 分)	更新预订状态操作用于修改预订记录的状态 (待确认、已确认、已入住、已取消), 并同步更新相关房间的状态。	
存储过程功能描述 (1 分)	存储过程 update_reservation_status 用于: 验证预订状态变更的合法性、更新预订状态、同步更新房间状态、记录状态变更日志以及确保数据一致性	
涉及的关系表 (2 分)	reservations (预订表) rooms (房间表) room_status (房间状态表) reservation_logs (预订日志表)	
表连接涉及字段 (1)	reservations.room_id = rooms.id rooms.room_status_id = room_status.id	
更改字段 (2 分)	字段	规则
	reservations.status	1=待确认 2=已确认 3=已入住 4=已取消
	rooms.room_status_id	- 当预订状态变为“已确认”时, 房间状态更新为“已预订”(3) - 当预订状态变为“已入住”时, 房间状态更新为“已入住”(2) - 当预订状态变为“已取消”时, 房间状态更新为“可用”(1)
	.....	.....
更新代码 (3 分)	(截屏)	















	<pre>@api_bp.route('/reservations/&lt;int:reservation_id&gt;/status', methods=['PUT']) def update_reservation_status(reservation_id):     data = request.get_json()     new_status = data.get('status')      if new_status not in [1, 2, 3, 4]:         return jsonify({'error': '无效的预订状态'}), 400      try:         # 调用存储过程         db.session.execute(             'CALL update_reservation_status(:reservation_id, :new_status)',             {'reservation_id': reservation_id, 'new_status': new_status}         )         db.session.commit()          # 获取更新后的预订信息         reservation = Reservation.query.get(reservation_id)         return jsonify({             'message': '预订状态更新成功',             'reservation_id': reservation_id,             'new_status': new_status,             'data': format_reservation(reservation)         })      except Exception as e:         db.session.rollback()         return jsonify({'error': '更新预订状态失败', 'details': str(e)}), 500</pre>
创建 存储 过程 源码 (3 分)	<div><div><pre>-- 创建存储过程 DELIMITER // CREATE PROCEDURE update_reservation_status(     IN p_reservation_id INT,     IN p_new_status INT ) BEGIN     DECLARE current_status INT;     DECLARE room_id INT;     DECLARE EXIT HANDLER FOR SQLEXCEPTION     BEGIN         ROLLBACK;         SIGNAL SQLSTATE '45000'         SET MESSAGE_TEXT = '更新预订状态失败';     END;      START TRANSACTION;      -- 获取当前预订状态和房间ID     SELECT status, room_id INTO current_status, room_id     FROM reservations     WHERE id = p_reservation_id;      -- 验证状态变更的合法性     IF current_status = 4 THEN         SIGNAL SQLSTATE '45000'         SET MESSAGE_TEXT = '已取消的预订不能更改状态';     END IF;      -- 更新预订状态     UPDATE reservations     SET status = p_new_status,         updated_at = NOW()     WHERE id = p_reservation_id;      -- 根据新状态更新房间状态     CASE p_new_status         WHEN 2 THEN -- 已确认             UPDATE rooms             SET room_status_id = 3, -- 已预订                 updated_at = NOW()             WHERE id = room_id;         WHEN 3 THEN -- 已入住             UPDATE rooms             SET room_status_id = 2, -- 已入住                 updated_at = NOW()             WHERE id = room_id;         WHEN 4 THEN -- 已取消</pre></div><div><pre>        updated_at = NOW()         WHERE id = room_id;     END CASE;      -- 记录状态变更日志     INSERT INTO reservation_logs (         reservation_id,         reservation_number,         guest_id,         room_id,         action,         action_time     )     SELECT         id,         reservation_number,         guest_id,         room_id,         CONCAT('STATUS_CHANGE_', p_new_status),         NOW()     FROM reservations     WHERE id = p_reservation_id;      COMMIT; END // DELIMITER ;</pre></div></div>
存储 过程	(截屏)

执行源码  
(1分)
















```
# 调用存储过程
db.session.execute(
    'CALL update_reservation_status(:reservation_id, :new_status)',
    {'reservation_id': reservation_id, 'new_status': new_status}
)
db.session.commit()
```

说明：不违背存储过程，能够执行更新操作

首先我们看到预定记录表中赵六所对应的 304 房间预定处于待确认状态

预订编号	客户姓名	房间号	入住日期	退房日期	状态	操作
R20251001	张三	101	2025-06-01	2025-06-03	已确认	  
R20251002	李四	103	2025-06-05	2025-06-08	已确认	  
R20251003	王五	201	2025-06-10	2025-06-11	已取消	  
R20251004	赵六	304	2025-06-11	2025-06-13	待确认	  
R20251005	钱七	203	2025-06-01	2025-06-02	已确认	  

房间管理中的 304 房间也处于可用状态

房间号	类型	价格	状态	面积	床型	操作
301	豪华单人房	¥499/晚	可用	35m²	大床	  
302	豪华双人房	¥599/晚	可用	40m²	双床	  
303	商务套房	¥799/晚	可用	60m²	大床	  
304	总统套房	¥1999/晚	可用	120m²	特大床	  
305	总统套房	¥1999/晚	可用	120m²	特大床	  

我们更新状态，改为已入住状态，发现表中 304 房间的状态值已改为 2（已入住）

id	room_number	floor	room_type_id	room_status_id	area	bed_type	max_guests	description	created_at	updated_at
1	101	1	1	1	25	单人床	1	圆雕花园, 安静舒适	2025-06-02 13:40:47	2025-06-02 13:40:47
2	102	1	1	1	25	单人床	1	采光良好, 视野开阔	2025-06-02 13:40:47	2025-06-02 13:40:47
3	103	1	2	1	30	双人床	2	空间宽敞, 设施齐全	2025-06-02 13:40:47	2025-06-02 13:40:47
4	104	1	2	2	30	双人床	2	安静角落, 适合家庭	2025-06-02 13:40:47	2025-06-02 13:40:47
5	105	1	3	1	35	大床	1	行政楼层, 专属服务	2025-06-02 13:40:47	2025-06-02 13:40:47
6	201	2	1	1	25	单人床	1	视野开阔, 安静舒适	2025-06-02 13:40:47	2025-06-02 13:40:47
7	202	2	2	3	30	双人床	2	新装修, 设施现代	2025-06-02 13:40:47	2025-06-02 13:40:47
8	203	2	3	1	35	大床	1	配备迷你吧台	2025-06-02 13:40:47	2025-06-02 13:07:41
9	204	2	4	1	40	双床	2	行政待遇, 免费早餐	2025-06-02 13:40:47	2025-06-02 13:40:47
10	205	2	5	4	60	大床	2	独立客厅, 商务办公	2025-06-02 13:40:47	2025-06-02 13:40:47
11	301	3	3	1	35	大床	1	豪华装修, 视野极佳	2025-06-02 13:40:47	2025-06-02 13:40:47
12	302	3	4	1	40	双床	2	行政楼层, 安静私密	2025-06-02 13:40:47	2025-06-02 13:40:47
13	303	3	5	1	60	大床	2	商务设施齐全	2025-06-02 13:40:47	2025-06-02 13:40:47
14	304	3	6	2	120	特大床	4	总统级享受, 全套设施	2025-06-02 13:40:47	2025-06-02 13:08:23
15	305	3	6	1	120	特大床	4	专属管家服务	2025-06-02 13:40:47	2025-06-02 13:40:47

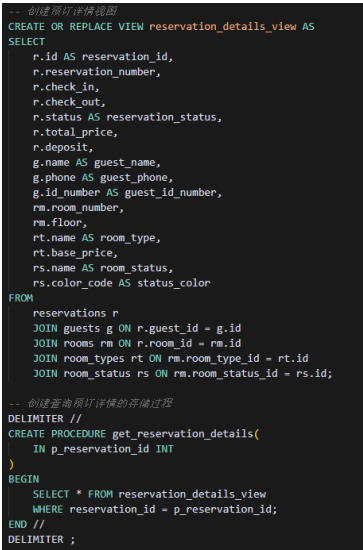
程序演示  
(2分)

说明：违背存储过程，系统报错；

我们编写如下脚本来测试更新报错结果

	<div><pre>test_reservation_errors.py &gt; ... 1 import requests 2 import json 3 4 BASE_URL = 'http://localhost:5000' 5 6 def test_invalid_status_update(): 7     """测试无效的状态更新""" 8     # 尝试将已取消的预订(状态4)更新为其他状态 9     reservation_id = 3 # ID为3的预订是已取消状态 10    data = { 11        'status': 2 # 尝试更新为已确认状态 12    } 13 14    response = requests.put( 15        f'{BASE_URL}/reservations/{reservation_id}/status', 16        json=data 17    ) 18 19    print("测试无效状态更新:") 20    print(f"状态码: {response.status_code}") 21    print(f"响应内容: {response.json()}\n") 22 23    def test_invalid_status_code(): 24        """测试无效的状态码""" 25        reservation_id = 1 26        data = { 27            'status': 5 # 无效的状态码 28        } 29 30        response = requests.put( 31            f'{BASE_URL}/reservations/{reservation_id}/status', 32            json=data 33        ) 34 35        print("测试无效状态码:") 36        print(f"状态码: {response.status_code}") 37        print(f"响应内容: {response.json()}\n") 38 39    if __name__ == '__main__': 40        print("开始测试存储过程错误处理...\n") 41        test_invalid_status_update() 42        test_invalid_status_code()</pre></div> <div>得到如下结果</div> <div><pre>开始测试存储过程错误处理...  测试无效状态更新: 状态码: 500 响应内容: {'details': 'Textual SQL expression 'CALL update_reservation_s...'' should be explicitly declared as text('CALL update_reservation_s...')', 'error':  测试无效状态码: 状态码: 400 响应内容: {'error': '无效的预订状态'}</pre></div> <div>发现报错我们无法更新已取消的预定状态以及使用无效的状态码去赋值</div>
备注	<div>这些错误处理机制确保了：</div> <div><div>1、数据一致性：防止无效的状态转换</div><div>2、业务规则遵守：确保预订状态变更符合业务逻辑</div><div>3、用户友好：提供清晰的错误信息</div><div>4、事务安全：所有错误都会触发事务回滚</div></div>

## 7. 含有视图的查询操作（15 分）

说明	<p>（1 分）简要说明该操作所要完成的功能；</p> <p>（1 分）简要说明建立的该视图的功能；</p> <p>（2 分）简要说明该操作涉及的关系数据表（以“表名”的形式给出）</p> <p>（1 分）简要说明表连接涉及的字段（以“表 1. 属性=表 2. 属性”）</p> <p>（6 分）实现该操作的关键代码（高级语言、SQL），截图即可；</p> <p>（4 分）如何执行该操作，按所述方法能够正常演示程序则给分。</p>
操作功能描述(1分)	创建一个视图来展示预订详情，包含房间信息、客户信息和预订状态，方便前台人员快速查询预订情况。
视图功能描述(1分)	创建名为 reservation_details_view 的视图，整合预订信息、房间信息、客户信息和状态信息，提供完整的预订详情查询功能
涉及的关系表(2分)	<p>reservations（预订表）</p> <p>rooms（房间表）</p> <p>guests（客户表）</p> <p>room_status（房间状态表）</p> <p>room_types（房间类型表）</p>
表连接字段（1 分）	<p>reservations.room_id = rooms.id</p> <p>reservations.guest_id = guests.id</p> <p>rooms.room_status_id = room_status.id</p> <p>rooms.room_type_id = room_types.id</p>
创建视图代码(3分)	<p>（截屏）</p>  <pre> -- 创建预订详情视图 CREATE OR REPLACE VIEW reservation_details_view AS SELECT     r.id AS reservation_id,     r.reservation_number,     r.check_in,     r.check_out,     r.status AS reservation_status,     r.total_price,     r.deposit,     g.name AS guest_name,     g.phone AS guest_phone,     g.id_number AS guest_id_number,     rm.room_number,     rm.floor,     rt.name AS room_type,     rt.base_price,     rs.name AS room_status,     rs.color_code AS status_color FROM     reservations r     JOIN guests g ON r.guest_id = g.id     JOIN rooms rm ON r.room_id = rm.id     JOIN room_types rt ON rm.room_type_id = rt.id     JOIN room_status rs ON rm.room_status_id = rs.id;  -- 创建查询预订详情的存储过程 DELIMITER // CREATE PROCEDURE get_reservation_details(     IN p_reservation_id INT ) BEGIN     SELECT * FROM reservation_details_view     WHERE reservation_id = p_reservation_id; END // DELIMITER ; </pre>
查询代码（3 分）	（截屏）



	<pre>import mysql.connector from datetime import datetime import json from decimal import Decimal  class DecimalEncoder(json.JSONEncoder):     """处理Decimal类型的JSON编码器"""     def default(self, obj):         if isinstance(obj, Decimal):             return float(obj)         return super(DecimalEncoder, self).default(obj)  def connect_to_database():     """连接到数据库"""     return mysql.connector.connect(         host="localhost",         user="root",         password="123456",         database="hotel_db"     )  def create_view(cursor):     """创建预订详情视图"""     view_sql = """         CREATE OR REPLACE VIEW reservation_details_view AS         SELECT r.id AS reservation_id, \                r.reservation_number, \                r.check_in, \                r.check_out, \                r.status AS reservation_status, \                r.total_price, \                r.deposit, \                g.name AS guest_name, \                g.phone AS guest_phone, \                g.id_number AS guest_id_number, \                r.room_number, \                r.floor, \                r.room_type, \                r.base_price, \                r.room AS room_status, \                r.color_code AS status_color         FROM reservations r \              JOIN guests g ON r.guest_id = g.id \              JOIN rooms rm ON r.room_id = rm.id \     """     cursor.execute(view_sql)     print("视图创建成功!")  def query_reservation_details(cursor, reservation_id):     """查询预订详情"""     query = """         SELECT * \         FROM reservation_details_view \         WHERE reservation_id = %s \     """     cursor.execute(query, (reservation_id,))     result = cursor.fetchone()      if result:         # 提取列名         columns = [desc[0] for desc in cursor.description]         # 将结果转换为字典         details = dict(zip(columns, result))          # 格式化日期时间         if 'check_in' in details:             details['check_in'] = details['check_in'].strftime("%Y-%m-%d")         if 'check_out' in details:             details['check_out'] = details['check_out'].strftime("%Y-%m-%d")          return details     return None  def main():     try:         # 连接数据库         conn = connect_to_database()         cursor = conn.cursor()          # 创建视图         create_view(cursor)          # 查询预订详情         reservation_id = 1 # 示例预订ID         details = query_reservation_details(cursor, reservation_id)          if details:             print("\n预订详情:")             print(json.dumps(details, indent=2, ensure_ascii=False, cls=DecimalEncoder))         else:             print(f"\n未找到预订ID为 {reservation_id} 的详情")      except mysql.connector.Error as err:         print(f"数据库错误: {err}")     except Exception as e:         print(f"发生错误: {e}")     finally:         if 'cursor' in locals():             cursor.close()         if 'conn' in locals():             conn.close()  if __name__ == "__main__":     main()</pre>
程序演示 (4分)	<p>在 view_query_demo.py 脚本文件的运行下我们得到如下结果</p> <pre>C:\Users\Administrator\AppData\Local\Programs\Python\Python313\python.exe "D:\Database Project\view_query_demo.py" 视图创建成功!  预订详情: {   "reservation_id": 1,   "reservation_number": "R20251001",   "check_in": "2025-06-01",   "check_out": "2025-06-03",   "reservation_status": 2,   "total_price": 598.0,   "deposit": 500.0,   "guest_name": "张三",   "guest_phone": "13800138001",   "guest_id_number": "110101199001011234",   "room_number": "101",   "floor": 1,   "room_type": "标准单人房",   "base_price": 299.0,   "room_status": "可用",   "status_color": "#22c55e" }  进程已结束，退出代码为 0</pre>
备注	