

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 | 2311983 | 姓名 | 余辰民 | 专业 | 信安法 |
| 项目名称 | 车辆信息管理系统 | | | | |
| 必备环境 | Python, MySQL | | | | |
| 系统主要功能简介（4 分） | <p>品牌信息管理：添加、查看、删除汽车品牌</p> <p>车型信息管理：区分燃油车和电动车，记录不同属性，添加车型、批量调价</p> <p>销售管理：处理车辆销售，记录客户信息</p> <p>综合查询：提供多种条件的灵活查询功能</p> <p>数据导出：支持将查询结果导出为 Excel</p> | | | | |
| 系统主要页面截图（6 分） |  | | | | |

车辆信息管理系统

品牌管理

车系管理

车型管理

销售管理

综合查询

+

车系信息录入

所属品牌:1:BMW

车系名称:

车系类型:Sedan

添加车系

清空表单

批量调价

| Series Id | Brand | Name | Category |
|-----------|-------|-------------|----------|
| 1 | BMW | 1 Series | Sedan |
| 2 | BMW | 2 Series | Sedan |
| 3 | BMW | 3 Series | Sedan |
| 4 | BMW | 4 Series | Coupe |
| 5 | BMW | 5 Series | Sedan |
| 6 | BMW | 6 Series GT | Sedan |
| 7 | BMW | 7 Series | Sedan |
| 8 | BMW | 8 Series | Coupe |
| 9 | BMW | M2 | Sedan |
| 10 | BMW | M3 | Sedan |
| 11 | BMW | M4 | Coupe |
| 12 | BMW | M5 | Sedan |
| 13 | BMW | X1 | SUV |
| 14 | BMW | X2 | SUV |
| 15 | BMW | X3 | SUV |

车辆信息管理系统

品牌管理

车系管理

车型管理

销售管理

综合查询

选择车型类型

燃油车型

电动车型

车型信息

所属品牌:1:BMW

所属车系:1:1 Series

车型名称:

发动机型号:

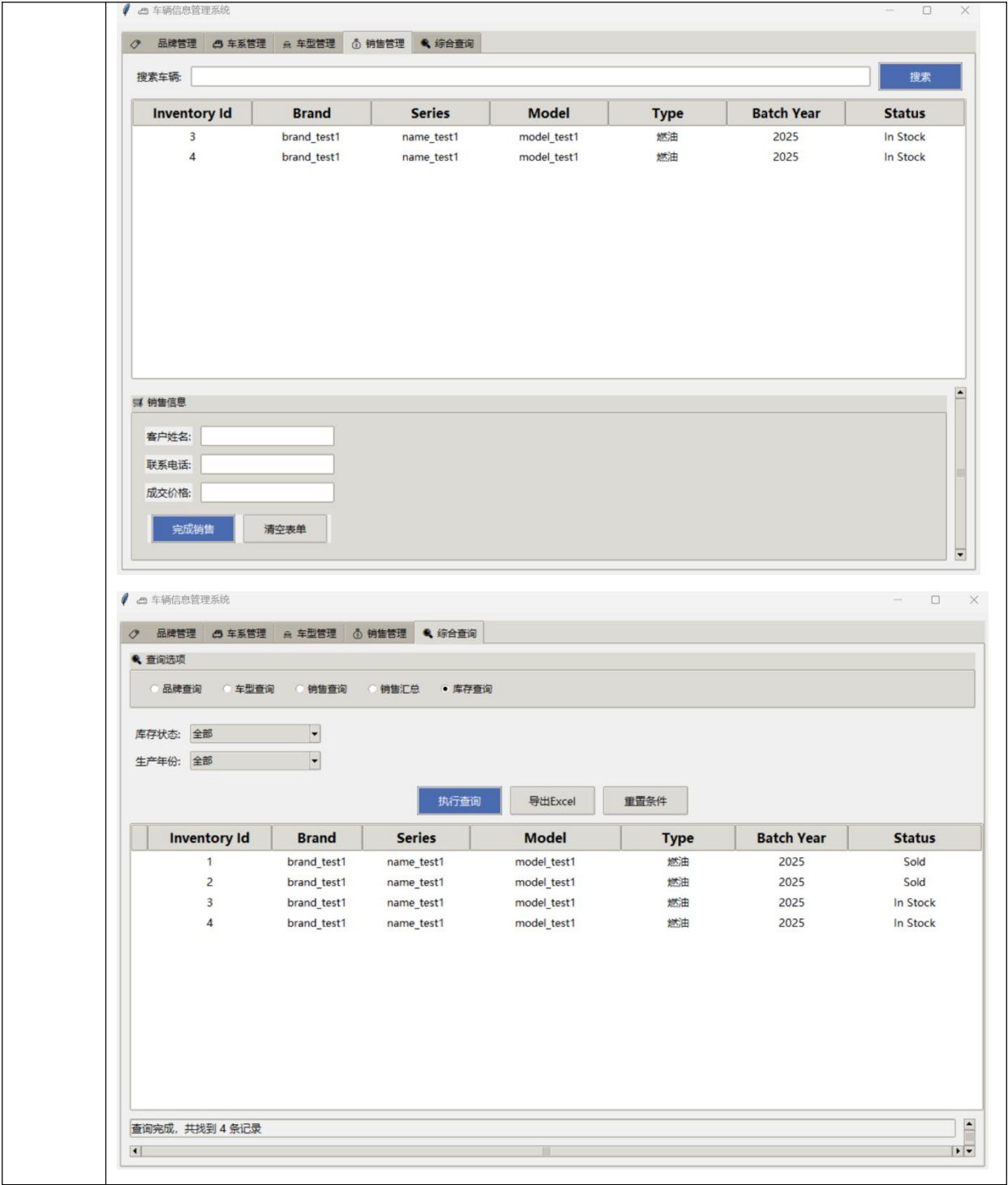
马力(HP):

燃油类型:Petrol

添加车型

清空表单

| Model Id | Brand | Series | Model Name | Type | Production Year | Spec |
|----------|-------------|------------|--------------------------|------|-----------------|-------------|
| 2043 | brand_test1 | name_test1 | model_test1 | 燃油 | 2025 | 001 (100HP) |
| 102 | Porsche | 718 | 718 Boxter Style Edition | 燃油 | 2023 | DPG (250HP) |
| 101 | Porsche | 718 | 718 Cayman Style Editic | 燃油 | 2023 | DPG (250HP) |
| 1021 | Porsche | 911 | 911 Carrera 3.0T | 燃油 | 2025 | DYH (394HP) |
| 1022 | Porsche | 911 | 911 Spirit 70 3.6T | 燃油 | 2025 | (485HP) |
| 1061 | Porsche | Cayenne | Cayenne 3.0T 逐梦版 | 燃油 | 2025 | DCB (353HP) |
| 1062 | Porsche | Cayenne | Caynne E-Hybrid 2.0T | 电动 | 2025 | None |



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. 创建 vehicle_information 数据库、执行提供的 SQL 脚本创建表结构, 输入数据 |
| | 高级 语言 | 1. Python 3.8+ |
| | | 2. 安装所需库: pymysql, tkinter |

| | 序号 | 名称 | 功能说明 | 取值 |
|-----------------------|--|-----------------|------------------|---|
| 连接串分析 (6分) | 1 | host | 数据库服务器地址 | localhost |
| | 2 | user | 数据库用户名 | root |
| | 3 | password | 数据库密码 | 123456 |
| | 4 | database | 数据库名称 | vehicle_information |
| | 5 | charset | 字符编码 | utf8 |
| | 6 | cursor class | 游标类, 影响游标返回数据的格式 | pymysql.cursors.DictCursor, 返回数据为字典形式, 方便通过键名访问数据 |
| 连接串代码 (截屏) (2分) |  <pre> self.connection = pymysql.connect(host='localhost', user='root', password='123456', database='vehicle_information', charset='utf8', cursorclass=pymysql.cursors.DictCursor) self.cursor = self.connection.cursor() </pre> | | | |
| 备注 | 连接过程中使用对应数据库的账号和密码, 对应到项目数据库, 并采用 utf-8 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 | brand | brand_id | 无 | 无 |
| | 2 | series | series_id | brand_id | brand(brand_id) |
| | 3 | model | model_id | series_id | series(series_id) |
| | 4 | combustionmodel | model_id | model_id | model(model_id) |
| | 5 | electricmodel | model_id | model_id | model(model_id) |
| | 6 | region | region_id | 无 | 无 |
| | 7 | customer | customer_id | region_id | region(region_id) |
| | 8 | productionbatch | batch_id | model_id | model(model_id) |
| | 9 | configuration | config_id | model_id | model(model_id) |
| | 10 | price | price_id | model_id | model(model_id) |
| | | | | region_id | region(region_id) |
| | 11 | inventory | inventory_id | model_id | model(model_id) |
| | | | | batch_id | productionbatch(batch_id) |
| | 12 | sale | sale_id | inventory_id | inventory(inventory_id) |
| | | | | customer_id | customer(customer_id) |

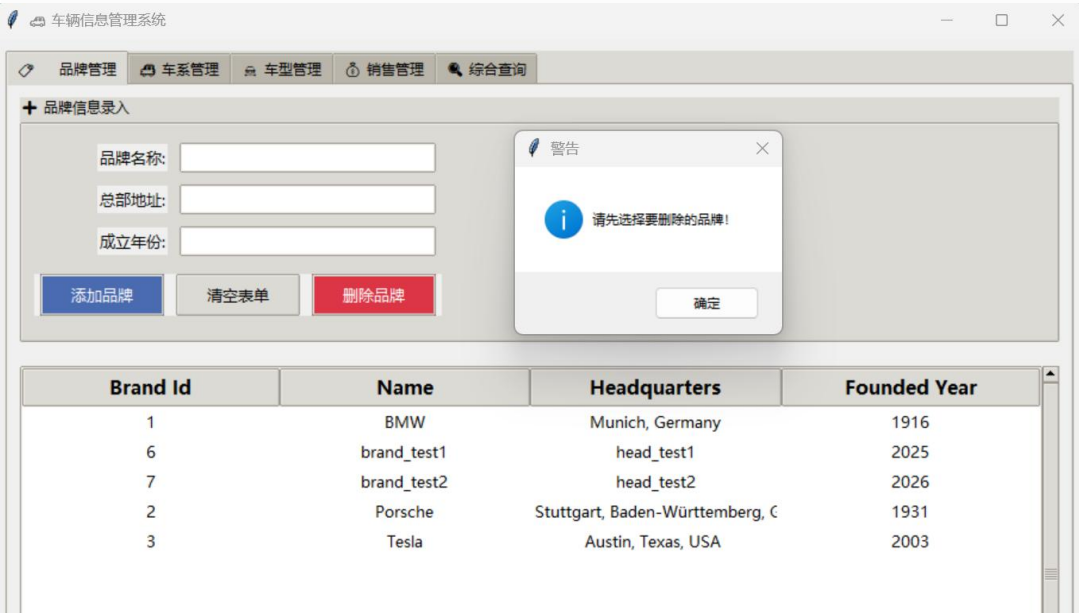
| | |
|------------|--|
| 关系图 (4) | <pre> erDiagram configuration --o{ model : "has" model --o{ combustionmodel : "has" model --o{ electricmodel : "has" model --o{ series : "has" series --o{ brand : "belongs to" price --o{ model : "has" price --o{ productionbatch : "has" price --o{ sale : "has" productionbatch --o{ model : "has" productionbatch --o{ inventory : "has" inventory --o{ sale : "has" sale --o{ region : "has" sale --o{ customer : "has" vw_sales_summary ..o{ brand : "uses" vw_sales_summary ..o{ series : "uses" vw_sales_summary ..o{ model : "uses" vw_sales_summary ..o{ sale : "uses" </pre> |
| 备注 | <p>图中的 brand、series、model、combustionmodel、electricmodel、region、customer、productionbatch、configuration、price、inventory、sale 是数据库中创建的表，vw_sales_summary 是数据库中创建的视图。所有外键关系已正确设置级联更新和删除规则，所有表都有自增主键。</p> |

4. 含有事务应用的删除操作（13 分）

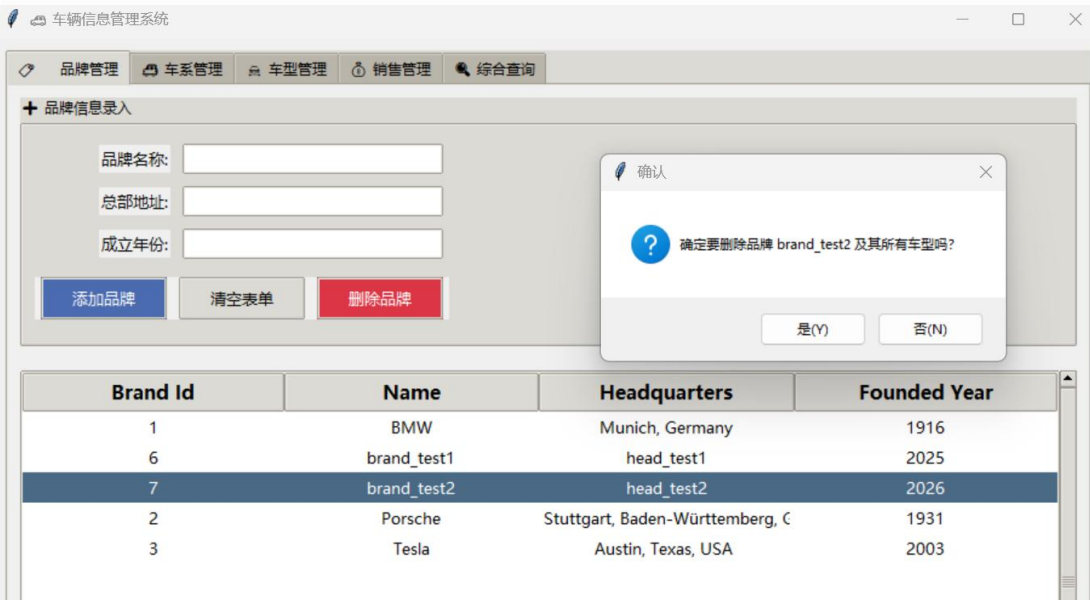
| | |
|------------------|---|
| 说明 | <p>(1 分) 简要说明该操作所要完成的功能；</p> <p>(2 分) 该操作会涉及的表（必须含有两张或两张以上的关系表，同时以“表名”的形式给出）</p> <p>(1 分) 表连接涉及字段描述（描述方式为“表 1. 属性=表 2. 属性”）</p> <p>(1 分) 删除条件涉及的字段描述（以“表名. 属性=? ”形式给出）</p> <p>(4 分) 实现该操作的关键代码（高级语言、SQL），截图即可；（其中如果删除语句中不包含任何形式的事务应用将扣除 3 分）</p> <p>(4 分) 如何执行该操作，按所述方法能够正常演示程序则给分。</p> |
| 功能描述 (1 分) | <p>该操作实现级联删除品牌及其所有关联数据（包括车系、车型、燃油车型/电动车型数据），通过事务确保在删除过程中若出现任何错误，所有操作将回滚，保持数据一致性。先删除子表数据（CombustionModel/ElectricModel），再删除 Model 表数据，然后删除 Series 表数据，最后删除 Brand 表数据。</p> |
| 涉及的表 (2 分) | <p>Brand（品牌表）、Series（车系表）、Model（车型表）、CombustionModel（燃油车型表）、ElectricModel（电动车型表）</p> |
| 表连接涉及字段 (1 分) | <p>Series.brand_id = Brand.brand_id Model.series_id = Series.series_id CombustionModel.model_id = Model.model_id ElectricModel.model_id = Model.model_id</p> |

| 删除条件 字段描述 (1分) | 字段 | 规则 |
|----------------------|---|---------------|
| | Brand.brand_id | 删除指定品牌 ID 的记录 |
| | Series.brand_id | 删除该品牌下的所有车系 |
| | Model.series_id | 删除这些车系下的所有车型 |
| | CombustionModel.model_id | 删除燃油车型特殊属性 |
| | ElectricModel.model_id | 删除电动车型特殊属性 |
| 代码 (4分) | <pre>def delete_brand_with_models(self): selected = self.brand_tree.selection() if not selected: messagebox.showwarning(title="警告", message="请先选择要删除的品牌!") return brand_id = self.brand_tree.item(selected)['values'][0] brand_name = self.brand_tree.item(selected)['values'][1] if not messagebox.askyesno(title="确认", message=f"确定要删除品牌 {brand_name} 及其所有车型吗? "): return try: # 开始事务 self.cursor.execute("START TRANSACTION") # 1. 删除相关车型的特殊属性(燃油/电动) self.cursor.execute(query:""" DELETE cm FROM CombustionModel cm JOIN Model m ON cm.model_id = m.model_id JOIN Series s ON m.series_id = s.series_id WHERE s.brand_id = %s """, args: (brand_id,)) self.cursor.execute(query:""" DELETE em FROM ElectricModel em JOIN Model m ON em.model_id = m.model_id JOIN Series s ON m.series_id = s.series_id WHERE s.brand_id = %s """, args: (brand_id,)) # 2. 删除相关车型 self.cursor.execute(query:""" DELETE m FROM Model m JOIN Series s ON m.series_id = s.series_id WHERE s.brand_id = %s """, args: (brand_id,)) # 3. 删除相关车系 self.cursor.execute(query:"DELETE FROM Series WHERE brand_id = %s", args: (brand_id,)) # 4. 最后删除品牌 self.cursor.execute(query:"DELETE FROM Brand WHERE brand_id = %s", args: (brand_id,)) # 提交事务 self.db.commit() messagebox.showinfo(title="成功", message="品牌及其相关数据删除成功!") self.load_brand_data() self.load_model_data() except Exception as e: self.db.rollback() messagebox.showerror(title="错误", message=f"删除失败: {str(e)}")</pre> | |

首先通过 START TRANSACTION 开启事务，然后按照从属关系由下至上依次删除数据——先删除 CombustionModel 和 ElectricModel 表中的燃油/电动车型特殊属性（通过 JOIN 关联到目标品牌），接着删除 Model 表中的基础车型数据，然后删除 Series 表中的车系数据，最后删除 Brand 表中的品牌记录。每个 DELETE 语句都通过 brand_id 参数精确控制删除范围，JOIN 条件确保只删除关联数据。若全部操作成功，调用 connection.commit() 提交事务；若任何步骤出错，在 except 块中执行 connection.rollback() 回滚所有操作，并通过消息框提示用户失败原因。



在没选择要删除的品牌时出现提示“请先选择要删除的品牌”。



在品牌管理界面选择要删除的品牌，点击“删除品牌”按钮系统弹出确认对话框，确认后系统执行事务操作：先删除子表数据（CombustionModel/ElectricModel），再删除 Model 表数据，然后删除 Series 表数据，最后删除 Brand 表数据。若全部成功则提交事务，否则回滚。

程序
演示
（4
分）

| | <table><tr><th>Brand Id</th><th>Name</th><th>Headquarters</th><th>Founded Year</th></tr><tr><td>1</td><td>BMW</td><td>Munich, Germany</td><td>1916</td></tr><tr><td>6</td><td>brand_test1</td><td>head_test1</td><td>2025</td></tr><tr><td>2</td><td>Porsche</td><td>Stuttgart, Baden-Württemberg, C</td><td>1931</td></tr><tr><td>3</td><td>Tesla</td><td>Austin, Texas, USA</td><td>2003</td></tr></table> | Brand Id | Name | Headquarters | Founded Year | 1 | BMW | Munich, Germany | 1916 | 6 | brand_test1 | head_test1 | 2025 | 2 | Porsche | Stuttgart, Baden-Württemberg, C | 1931 | 3 | Tesla | Austin, Texas, USA | 2003 |
|----------|--|---------------------------------|--------------|--------------|--------------|---|-----|-----------------|------|---|-------------|------------|------|---|---------|---------------------------------|------|---|-------|--------------------|------|
| Brand Id | Name | Headquarters | Founded Year | | | | | | | | | | | | | | | | | | |
| 1 | BMW | Munich, Germany | 1916 | | | | | | | | | | | | | | | | | | |
| 6 | brand_test1 | head_test1 | 2025 | | | | | | | | | | | | | | | | | | |
| 2 | Porsche | Stuttgart, Baden-Württemberg, C | 1931 | | | | | | | | | | | | | | | | | | |
| 3 | Tesla | Austin, Texas, USA | 2003 | | | | | | | | | | | | | | | | | | |
| | 点击“是”后再刷新页面和数据库，发现确实删除了。 | | | | | | | | | | | | | | | | | | | | |
| 备注 | <div>1. 采用“从下至上”的删除顺序避免外键约束冲突</div> <div>2. 使用数据库事务确保操作的原子性</div> <div>3. 删除前显示将被影响的关联数据数量，提高用户体验</div> <div>4. 捕获所有异常并执行回滚操作</div> | | | | | | | | | | | | | | | | | | | | |

5. 触发器控制下的添加操作（20分）

| | | |
|---------------|--|--|
| 说明 | <p>(1分) 简要说明该操作所要完成的功能；</p> <p>(2分) 简要说明该触发器所要完成的功能</p> <p>(1分) 该操作会涉及的表（以“表名”的形式给出）。</p> <p>(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；</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>(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。</p> | |
| 功能描述 (1分) | 添加新车系，触发器确保同一品牌下车系名称唯一，该操作通过触发器实现对车系名称的唯一性约束，确保同一品牌下不能存在同名车系，维护数据完整性。 | |
| 触发器描述 (2分) | 检查同一品牌下车系名称是否已存在 插入新车系记录前自动触发 检查同一品牌 (brand_id) 下是否已存在相同名称 (name) 的车系 如果存在则抛出错误，阻止插入操作 错误信息明确提示“同一品牌下不能有同名车系” | |
| 涉及的表 (1分) | Series（车系表） | |
| 输入数据 (2分) | 字段 | 规则 |
| | name(必填，唯一) | 必填，同一 brand_id 下必须唯一（由触发器保证） |
| | brand_id(必填) | 必填，必须存在于 Brand 表的 brand_id |
| | category | 可选，值必须为['Sedan', 'SUV', 'Coupe', 'Electric']中的一种 |

插入操作
源码
(3 分)

```
def add_series(self): 1个用法
    """添加新车系(受触发器约束)"""
    if not self.series_name.get():
        messagebox.showwarning( title: "警告", message: "车系名称不能为空!")
        return

    selected_brand = self.brand_combo.get()
    if not selected_brand:
        messagebox.showwarning( title: "警告", message: "请选择所属品牌!")
        return

    try:
        brand_id = int(selected_brand.split(":")[0])
        series_name = self.series_name.get()
        category = self.series_category.get()

        self.cursor.execute(
            query: "INSERT INTO Series (brand_id, name, category) VALUES (%s, %s, %s)",
            args: (brand_id, series_name, category)
        )
        self.db.commit()
        messagebox.showinfo( title: "成功", message: "车系添加成功!")
        self.load_series_data()
        self.clear_series_form()
    except pymysql.Error as e:
        self.db.rollback()
        # 触发器错误会有特定的错误代码
        if e.args[0] == 1644: # MySQL自定义错误代码
            messagebox.showerror( title: "错误", message: f"添加失败: {e.args[1]}")
        else:
            messagebox.showerror( title: "错误", message: f"添加失败: {str(e)}")
```

触发器源码
(3 分)

| 名 | 触发 | 插入 | 更新 | 删除 |
|----------------------|--------|-------------------------------------|--------------------------|--------------------------|
| before_insert_series | BEFORE | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| before_series_inser | BEFORE | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

定义

```
1 BEGIN
2     DECLARE brand_count INT;
3
4     -- 检查品牌是否存在
5     SELECT COUNT(*) INTO brand_count FROM Brand WHERE brand_id = NEW.brand_id;
6
7     IF brand_count = 0 THEN
8         SIGNAL SQLSTATE '45000'
9         SET MESSAGE_TEXT = '添加失败: 指定的品牌不存在';
10    END IF;
11 END
```

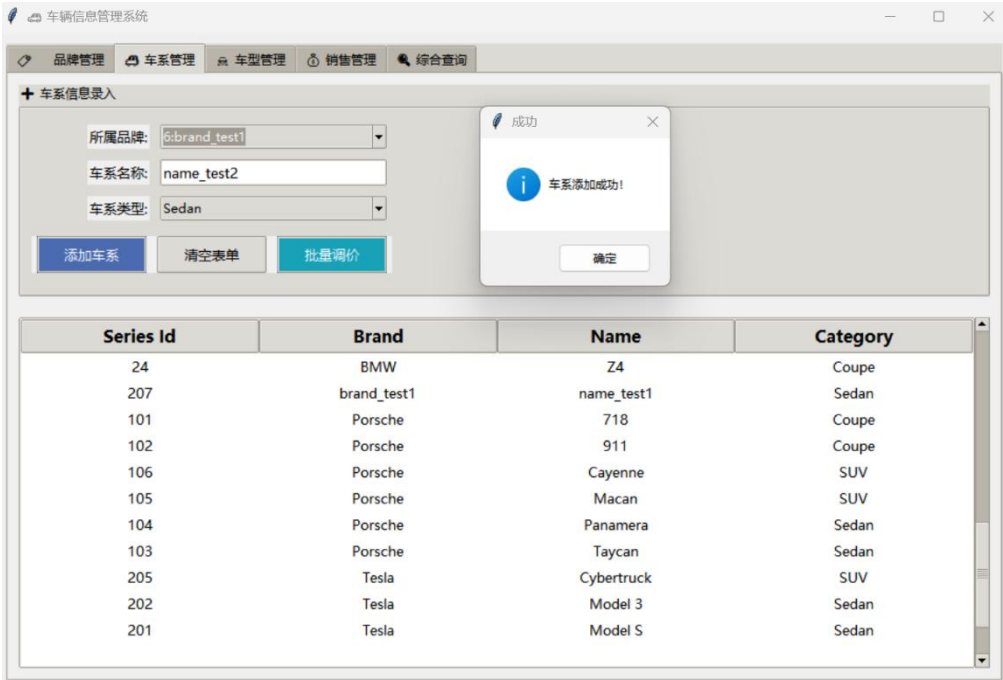
| 名 | 触发 | 插入 | 更新 | 删除 |
|---------------------|--------|-------------------------------------|--------------------------|--------------------------|
| before_insert_serie | BEFORE | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| before_series_inser | BEFORE | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |


```
1 BEGIN
2     DECLARE series_count INT;
3
4     SELECT COUNT(*) INTO series_count
5     FROM Series
6     WHERE brand_id = NEW.brand_id AND name = NEW.name;
7
8     IF series_count > 0 THEN
9         SIGNAL SQLSTATE '45000'
10        SET MESSAGE_TEXT = '同一品牌下不能有同名车系';
11    END IF;
12 END
```

当向 Series 表插入新记录前，触发器自动执行以下操作：首先声明 series_count 变量，然后通过 SELECT COUNT 查询统计当前品牌下同名车系的数量。如果查询结果大于 0，使用 SIGNAL SQLSTATE '45000' 抛出自定义错误，阻止 INSERT 操作执行；否则允许继续插入。其中 NEW.brand_id 和 NEW.name 表示待插入记录的品牌 ID 和名称。该触发器与应用程序中的表单验证形成双重保障，确保数据库层面的数据完整性，即使用户绕过前端验证也能阻止非法数据插入。

数据库中已有 brand_test1 品牌、name_test1 车系。

程序演示
(4 分)



如图所示，向数据库中选择 brand_test1 品牌，输入想添加的车系名称: name_test2，点击添加车系，由于没有发生任何冲突，是一个全新的车系名称，且品牌名称已存在，故现实车系添加成功。

对象 series @vehicle_information (localho...

开始事务 文本 筛选 排序 列 导入

| series_id | name | category | brand_id |
|-----------|------------|----------|----------|
| 19 | X7 | SUV | 1 |
| 20 | X3 M | SUV | 1 |
| 21 | X5 M | SUV | 1 |
| 22 | X6 M | SUV | 1 |
| 23 | XM | SUV | 1 |
| 24 | Z4 | Coupe | 1 |
| 101 | 718 | Coupe | 2 |
| 102 | 911 | Coupe | 2 |
| 103 | Taycan | Sedan | 2 |
| 104 | Panamera | Sedan | 2 |
| 105 | Macan | SUV | 2 |
| 106 | Cayenne | SUV | 2 |
| 201 | Model S | Sedan | 3 |
| 202 | Model 3 | Sedan | 3 |
| 203 | Model X | SUV | 3 |
| 204 | Model Y | SUV | 3 |
| 205 | Cybertruck | SUV | 3 |
| 206 | Roadster | Coupe | 3 |
| 207 | name_test1 | Sedan | 6 |
| 210 | name_test2 | Sedan | 6 |

在 navicat 中查看 series 表，发现的确添加成功。

程序演示
(4 分)

车辆信息管理系统

品牌管理 车系管理 车型管理 销售管理 综合查询

+ 车系信息录入

所属品牌: 6:brand_test1

车系名称: name_test1

车系类型: Sedan

添加车系 清空表单 批量调价

错误

添加车系失败: (1644, '同一品牌下不能有同名车系')


确定

| Series Id | Brand | Name | Category |
|-----------|-------------|------------|----------|
| 24 | BMW | Z4 | Coupe |
| 207 | brand_test1 | name_test1 | Sedan |
| 101 | Porsche | 718 | Coupe |
| 102 | Porsche | 911 | Coupe |
| 106 | Porsche | Cayenne | SUV |
| 105 | Porsche | Macan | SUV |
| 104 | Porsche | Panamera | Sedan |
| 103 | Porsche | Taycan | Sedan |
| 205 | Tesla | Cybertruck | SUV |
| 202 | Tesla | Model 3 | Sedan |
| 201 | Tesla | Model S | Sedan |

同样，在上面的条件下，尝试向 brand_test1 品牌下添加已有的车系 name_test1，结果如图所示，报错 1644，‘同一品牌下不能有同名车系’。

| | <div><div>对象</div><div>series @vehicle_information (localho...</div></div> <div><div>开始事务</div><div>文本</div><div>筛选</div><div>排序</div><div>列</div><div>导入</div></div> <table><tr><th>series_id</th><th>name</th><th>category</th><th>brand_id</th></tr><tr><td>19</td><td>X7</td><td>SUV</td><td>1</td></tr><tr><td>20</td><td>X3 M</td><td>SUV</td><td>1</td></tr><tr><td>21</td><td>X5 M</td><td>SUV</td><td>1</td></tr><tr><td>22</td><td>X6 M</td><td>SUV</td><td>1</td></tr><tr><td>23</td><td>XM</td><td>SUV</td><td>1</td></tr><tr><td>24</td><td>Z4</td><td>Coupe</td><td>1</td></tr><tr><td>101</td><td>718</td><td>Coupe</td><td>2</td></tr><tr><td>102</td><td>911</td><td>Coupe</td><td>2</td></tr><tr><td>103</td><td>Taycan</td><td>Sedan</td><td>2</td></tr><tr><td>104</td><td>Panamera</td><td>Sedan</td><td>2</td></tr><tr><td>105</td><td>Macan</td><td>SUV</td><td>2</td></tr><tr><td>106</td><td>Cayenne</td><td>SUV</td><td>2</td></tr><tr><td>201</td><td>Model S</td><td>Sedan</td><td>3</td></tr><tr><td>202</td><td>Model 3</td><td>Sedan</td><td>3</td></tr><tr><td>203</td><td>Model X</td><td>SUV</td><td>3</td></tr><tr><td>204</td><td>Model Y</td><td>SUV</td><td>3</td></tr><tr><td>205</td><td>Cybertruck</td><td>SUV</td><td>3</td></tr><tr><td>206</td><td>Roadster</td><td>Coupe</td><td>3</td></tr><tr><td>207</td><td>name_test1</td><td>Sedan</td><td>6</td></tr><tr><td>210</td><td>name_test2</td><td>Sedan</td><td>6</td></tr></table> <div>在 navicat 中查看 series 表，发现并没有重复添加。</div> | series_id | name | category | brand_id | 19 | X7 | SUV | 1 | 20 | X3 M | SUV | 1 | 21 | X5 M | SUV | 1 | 22 | X6 M | SUV | 1 | 23 | XM | SUV | 1 | 24 | Z4 | Coupe | 1 | 101 | 718 | Coupe | 2 | 102 | 911 | Coupe | 2 | 103 | Taycan | Sedan | 2 | 104 | Panamera | Sedan | 2 | 105 | Macan | SUV | 2 | 106 | Cayenne | SUV | 2 | 201 | Model S | Sedan | 3 | 202 | Model 3 | Sedan | 3 | 203 | Model X | SUV | 3 | 204 | Model Y | SUV | 3 | 205 | Cybertruck | SUV | 3 | 206 | Roadster | Coupe | 3 | 207 | name_test1 | Sedan | 6 | 210 | name_test2 | Sedan | 6 |
|-----------|--|-----------|----------|----------|----------|----|----|-----|---|----|------|-----|---|----|------|-----|---|----|------|-----|---|----|----|-----|---|----|----|-------|---|-----|-----|-------|---|-----|-----|-------|---|-----|--------|-------|---|-----|----------|-------|---|-----|-------|-----|---|-----|---------|-----|---|-----|---------|-------|---|-----|---------|-------|---|-----|---------|-----|---|-----|---------|-----|---|-----|------------|-----|---|-----|----------|-------|---|-----|------------|-------|---|-----|------------|-------|---|
| series_id | name | category | brand_id | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | X7 | SUV | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | X3 M | SUV | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 21 | X5 M | SUV | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | X6 M | SUV | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | XM | SUV | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | Z4 | Coupe | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 101 | 718 | Coupe | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 102 | 911 | Coupe | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 103 | Taycan | Sedan | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 104 | Panamera | Sedan | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 105 | Macan | SUV | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 106 | Cayenne | SUV | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 201 | Model S | Sedan | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 202 | Model 3 | Sedan | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 203 | Model X | SUV | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 204 | Model Y | SUV | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 205 | Cybertruck | SUV | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 206 | Roadster | Coupe | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 207 | name_test1 | Sedan | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 210 | name_test2 | Sedan | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 备注 | <div>触发器需要在数据库初始化时创建；</div> <div>错误代码 1644 需要与触发器中的 SIGNAL SQLSTATE '45000' 对应；</div> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

6. 存储过程控制下的更新操作（18 分）

| | | |
|-------------------|---|--|
| 说明 | <p>（1 分）简要说明该操作所要完成的功能；</p> <p>（1 分）简要说明该存储过程所要完成的功能；</p> <p>（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）</p> <p>（1 分）表连接涉及字段描述（描述方式为“表 1. 属性=表 2. 属性”）</p> <p>（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；</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>（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。</p> | |
| 功能描述 （1 分） | <p>批量更新车型价格，支持按品牌或车系筛选，并按指定涨幅比例调整价格，可设置最大涨幅金额限制。</p> | |
| 存储过程功能描述 （1 分） | <p>存储过程 <code>update_model_prices</code> 实现以下功能：</p> <p>根据输入参数（品牌 ID、车系 ID）筛选需要更新的车型；</p> <p>按指定百分比调整车型基础价格；</p> <p>应用最大涨幅金额限制（如设置）；</p> <p>返回实际更新的记录数；</p> <p>确保价格调整在事务中完成，保证数据一致性。</p> | |
| 涉及的关系表 （2 分） | <p>Brand（品牌表）、Series（车系表）、Model（车型表）、CombustionModel（燃油车型表）、ElectricModel（电动车型表）、Price（车型价格表）</p> | |
| 表连接涉及字段(1) | <p><code>Model.series_id = Series.series_id</code></p> <p><code>Series.brand_id = Brand.brand_id</code></p> <p><code>CombustionModel.model_id = Model.model_id</code>（左连接）</p> <p><code>ElectricModel.model_id = Model.model_id</code>（左连接）</p> <p><code>Price.model_id = Model.model_id</code></p> | |
| 更改字段 （2 分） | 字段 | 规则 |
| | <p><code>Price.msrp</code></p> <p><code>Price.dealer_price</code></p> | <p>新价格 = 原价格 × (1 + 涨幅百分比/100)</p> <p>如果设置了最大涨幅金额，则：</p> <p>新价格 = MIN(原价格 × (1 + 涨幅百分比/100), 原价格 + 最大涨幅金额)</p> |
| | | |
| 更新代码 （3 分） |  <pre>self.series_price_btn = ttk.Button(btn_frame, text="批量调价", style='Info.TButton', command=lambda: print("批量调价按钮点击") or self.controller.show_price_update_dialog()) self.series_price_btn.pack(side='left', padx=5) btn_frame.grid(row=5, columnspan=2, pady=10)</pre> <p>存储过程封装了复杂的价格调整逻辑，接收四个参数：品牌 ID、车系 ID、涨价百分比和最大涨幅金额。</p> | |


```

def show_price_update_dialog(self): 2 个用法 (1 个动态)
    dialog = tk.Toplevel(self.view.root)
    dialog.title("批量更新车型价格")
    dialog.geometry("400x300")
    # 品牌选择
    ttk.Label(dialog, text="品牌:").grid(row=0, column=0, padx=5, pady=5, sticky='e')
    brand_combo = ttk.Combobox(dialog, state='readonly')
    brand_combo.grid(row=0, column=1, padx=5, pady=5, sticky='ew')
    # 车系选择
    ttk.Label(dialog, text="车系:").grid(row=1, column=0, padx=5, pady=5, sticky='e')
    series_combo = ttk.Combobox(dialog, state='readonly')
    series_combo.grid(row=1, column=1, padx=5, pady=5, sticky='ew')
    # 价格涨幅
    ttk.Label(dialog, text="价格涨幅(%):").grid(row=2, column=0, padx=5, pady=5, sticky='e')
    price_increase = ttk.Entry(dialog)
    price_increase.grid(row=2, column=1, padx=5, pady=5, sticky='ew')
    # 最大涨幅限制
    ttk.Label(dialog, text="最大涨幅金额:").grid(row=3, column=0, padx=5, pady=5, sticky='e')
    max_increase = ttk.Entry(dialog)
    max_increase.grid(row=3, column=1, padx=5, pady=5, sticky='ew')
    # 加载品牌数据
    brands = self.db.get_all_brands()
    brand_combo['values'] = ["全部品牌"] + [f"{b['brand_id']}:{b['name']}" for b in brands]
    brand_combo.current(0)

```

```

class MainController: 2 用法
    def show_price_update_dialog(self): 2 个用法 (1 个动态)

    def execute_update():
        try:
            # 解析品牌ID
            brand_id = None
            selected_brand = brand_combo.get()
            if selected_brand != "全部品牌":
                brand_id = int(selected_brand.split(":")[0])
            # 解析车系ID
            series_id = None
            selected_series = series_combo.get()
            if selected_series != "全部车系":
                series_id = int(selected_series.split(":")[0])
            # 获取价格参数
            increase = float(price_increase.get())
            max_inc = float(max_increase.get()) if max_increase.get() else 0
            # 调用更新
            self.db.update_model_prices(brand_id, series_id, increase, max_inc)
            self.show_message(title="成功", message=f"成功更新了车型的价格!")
            dialog.destroy()
            self.load_initial_data()
        except ValueError:
            self.show_message(title="错误", message="请输入有效的数字!", is_error=True)
        except Exception as e:
            self.show_message(title="错误", message=f"价格更新失败: {str(e)}", is_error=True)

```

过程内部首先声明 affected_rows 变量记录影响行数，然后开启事务。核心更新语句通过多表 JOIN (Price-Model-Series) 定位目标记录，使用 LEAST 函数实现两种价格调整逻辑：基础计算是原价 × (1+涨幅百分比)，同时判断如果设置了最大涨幅(p_max_increase>0)，则取“计算价格”和“原价+最大涨幅”中的较小值。WHERE 子句动态过滤条件，当参数为 NULL

时忽略该条件。更新后通过 ROW_COUNT() 获取影响行数，提交事务后返回该数值。这种设计既支持全局调价也支持精确范围调整，且通过事务保证大批量更新时的数据安全。

创建
存储
过程
源码
(3
分)

```
1 CREATE DEFINER=`root`@`localhost` PROCEDURE `update_model_prices_new`(  
2     IN p_brand_id INT,  
3     IN p_series_id INT,  
4     IN p_price_increase DECIMAL(5,2),  
5     IN p_max_increase DECIMAL(12,2)  
6 )  
7 BEGIN  
8     DECLARE done INT DEFAULT FALSE;  
9     DECLARE model_id_val INT;  
10    DECLARE current_price DECIMAL(12,2);  
11    DECLARE new_price DECIMAL(12,2);  
12    -- 声明游标，获取符合条件的车型  
13    DECLARE model_cursor CURSOR FOR  
14        SELECT m.model_id, p.msrp  
15        FROM Model m  
16        JOIN Price p ON m.model_id = p.model_id  
17        JOIN Series s ON m.series_id = s.series_id  
18        WHERE (p_brand_id IS NULL OR s.brand_id = p_brand_id)  
19        AND (p_series_id IS NULL OR m.series_id = p_series_id);  
20    -- 声明异常处理  
21    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;  
22    -- 验证价格涨幅百分比  
23    IF p_price_increase < -100 THEN  
24        SIGNAL SQLSTATE '45000'  
25        SET MESSAGE_TEXT = '价格降幅不能超过100%';  
26    END IF;  
27    -- 开始事务  
28    START TRANSACTION;  
29    -- 打开游标  
30    OPEN model_cursor;  
  
31    -- 循环处理每个车型  
32    model_loop: LOOP  
33        FETCH model_cursor INTO model_id_val, current_price;  
34        IF done THEN  
35            LEAVE model_loop;  
36        END IF;  
37        -- 计算新价格  
38        SET new_price = current_price * (1 + p_price_increase/100);  
39        -- 检查最大涨幅限制  
40        IF p_max_increase > 0 AND (new_price - current_price) > p_max_increase THEN  
41            SET new_price = current_price + p_max_increase;  
42        END IF;  
43        -- 确保价格不会变为负数  
44        IF new_price <= 0 THEN  
45            SET new_price = current_price * 0.1; -- 最低设置为原价的10%  
46        END IF;  
47        -- 更新车型价格  
48        UPDATE Price  
49        SET msrp = new_price,  
50        dealer_price = new_price * 0.9 -- 经销商价格为MSRP的90%  
51        WHERE model_id = model_id_val;  
52    END LOOP;  
53    -- 关闭游标  
54    CLOSE model_cursor;  
55    -- 提交事务  
56    COMMIT;  
57    -- 返回更新的车型数量  
58    SELECT ROW_COUNT() AS models_updated;  
59 END
```

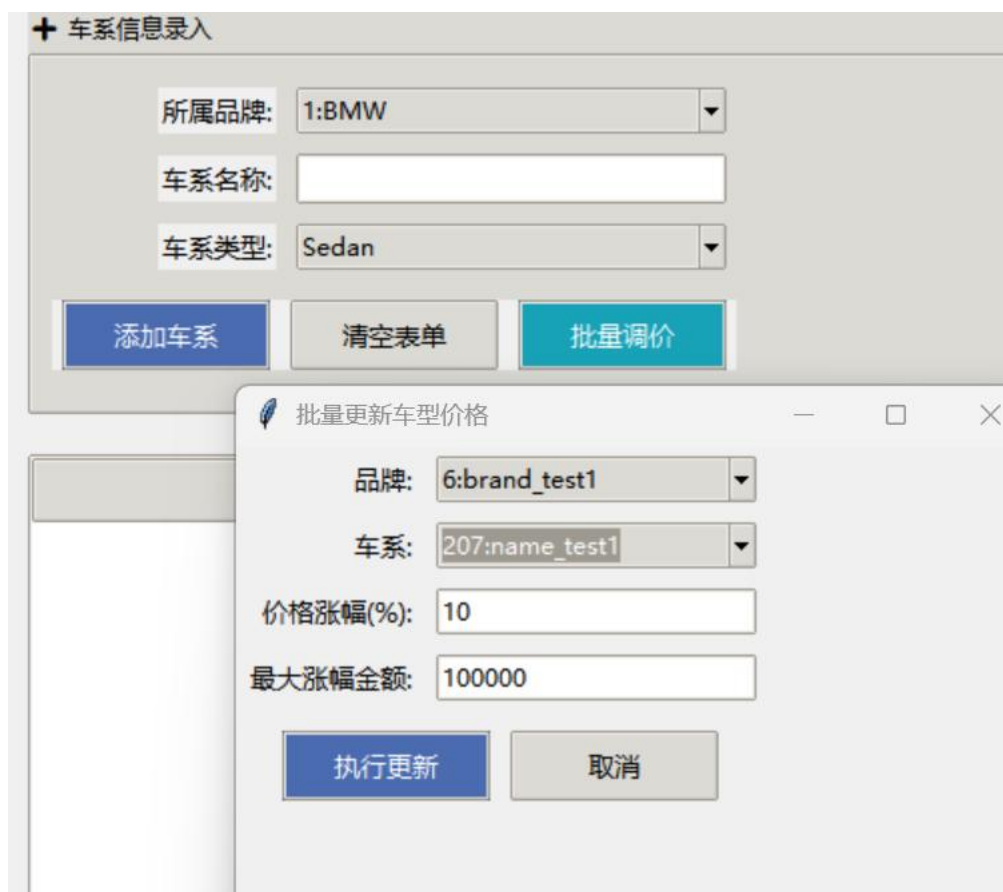
存储
过程
执行
源码
(1
分)

```
def update_model_prices(self, brand_id=None, series_id=None, price_increase=0, max_increase=0):
    try:
        self.cursor.callproc(
            procname: "update_model_prices_new",
            args: (brand_id, series_id, price_increase, max_increase)
        )
        result = self.cursor.fetchone()
        self.connection.commit()
        return result['models_updated'] if result else 0
    except pymysql.Error as e:
        self.connection.rollback()
        raise e
```

程序
演示
(2
分)

| | | | | |
|------|------|---|-----------|-----------|
| 2043 | 2043 | 1 | 150000.00 | 100000.00 |
|------|------|---|-----------|-----------|

数据库中已有品牌为 brand_test1, 车系为 name_test1 的价格, 左边为 msrp (车企报价), 右边为 dealer_price (经销商报价), 分别为 15 万元和 10 万元。



在“车系管理”界面点击“批量调价”按钮, 在弹出的对话框中选择品牌 brand_test1 选择车系 name_test1, 输入价格涨幅百分比 (如 10 表示涨价 10%), 可选输入最大涨幅金额 (如 100000 表示最高涨价 100000 元, 此处 100000 即为不设限制)。

所属品牌: 1:BMW

车系名称:

车系类型: Sedan

车系

清空表单

批量调价

成功

成功更新了车型的价格!

确定

点击“执行更新”按钮，系统显示成功更新。

| | | | | | |
|--|------|------|---|-----------|-----------|
| | 2043 | 2043 | 1 | 165000.00 | 148500.00 |
|--|------|------|---|-----------|-----------|

来到数据库观察 price 表，发现的确涨价 10%，且按照设定的——经销商报价为车企报价的 90%。

品牌: 6:brand_test1

车系: 207:name_test1

价格涨幅(%): 50

最大涨幅金额: 5000

执行更新

取消

同样，在“车系管理”界面点击“批量调价”按钮，在弹出的对话框中选择品牌 brand_test1 选择车系 name_test1,输入价格涨幅百分比为 50,输入最大涨幅金额为 5000，测试是否只涨价 5000 元（上调 50%明显大于 5000 元）。

| | | | | | |
|--|------|------|---|-----------|-----------|
| | 2043 | 2043 | 1 | 170000.00 | 153000.00 |
|--|------|------|---|-----------|-----------|

点击“执行更新”按钮，系统显示成功更新。来到数据库观察 price 表，发现的确只涨价了 5000 元，且按照设定的——经销商报价为车企报价的 90%。

品牌: 6:brand_test1

车系: 207:name_test1

价格涨幅(%): -99

最大涨幅金额: 99999

执行更新

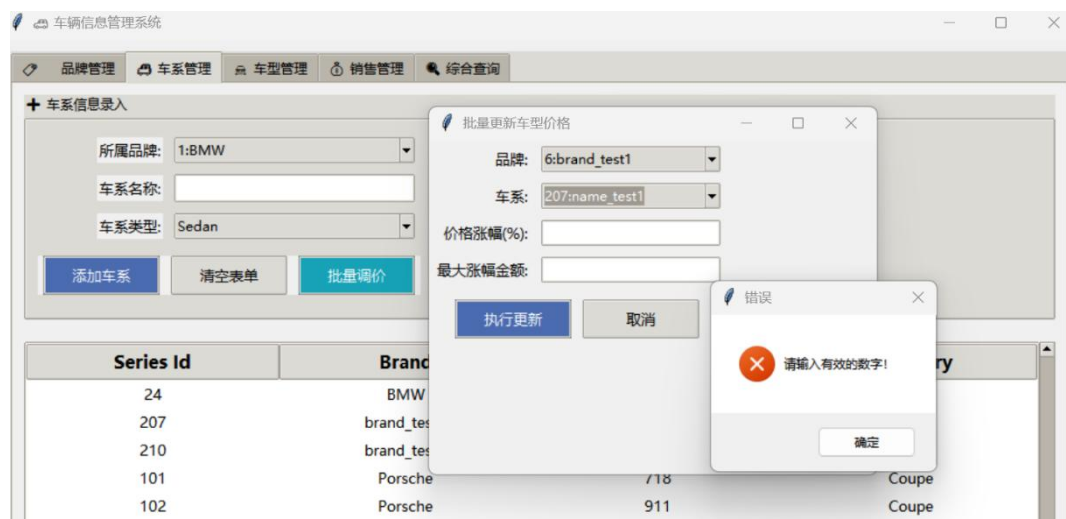
取消

同样，在“车系管理”界面点击“批量调价”按钮，在弹出的对话框中选择品牌 brand_test1 选择车系 name_test1,输入价格涨幅百分比为-99,测试是否确保价格最低为原价的 10%，且

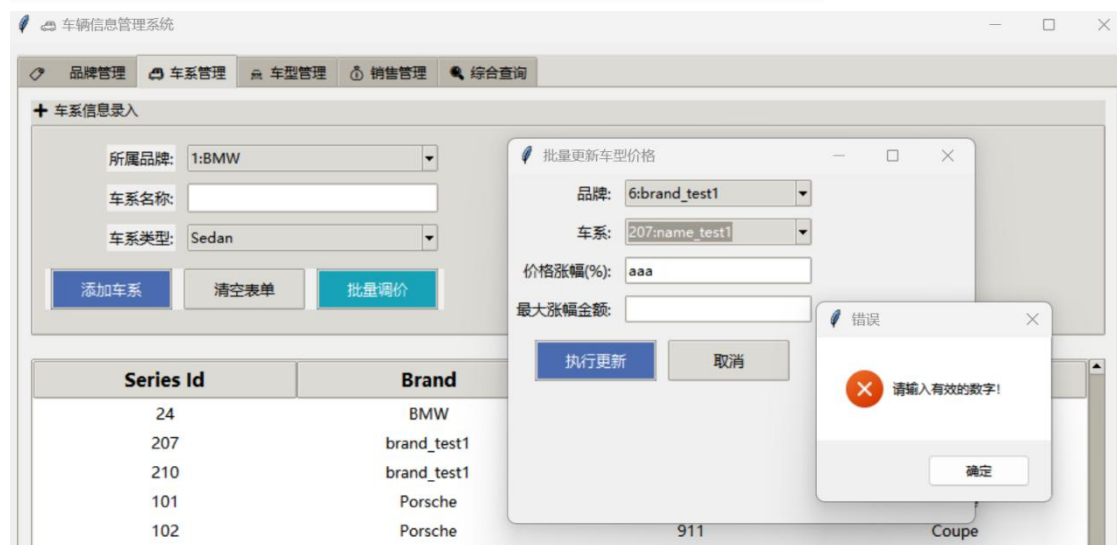
不为负数。

| | | | | |
|------|------|---|---------|---------|
| 2043 | 2043 | 1 | 1700.00 | 1530.00 |
|------|------|---|---------|---------|

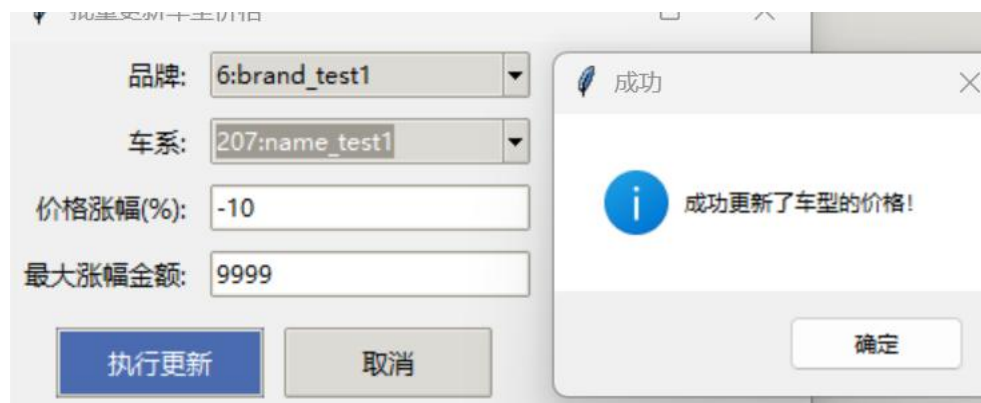
点击“执行更新”按钮，系统显示成功更新。来到数据库观察 price 表，发现的确只降价为原来的 10%，且按照设定的——经销商报价为车企报价的 90%。



不输入涨幅百分比直接执行 → 系统提示“请输入有效的数字！”



输入非数字字符 → 系统提示“请输入有效的数字！”



程序
演示
(2
分)

| | | | | | | | | | | | |
|------|---|------|---------|---------|---------|---------|------|------|---|---------|---------|
| | <div><table><tr><td>2043</td><td>2043</td><td>1</td><td>1700.00</td><td>1530.00</td></tr></table><table><tr><td>2043</td><td>2043</td><td>1</td><td>1530.00</td><td>1377.00</td></tr></table><p>输入负的涨幅百分比 → 价格会被相应调低（符合业务逻辑）</p><div><div><div>品牌: 6:brand_test1</div><div>车系: 207:name_test1</div><div>价格涨幅(%): -110</div><div>最大涨幅金额: 9999999</div><div>执行更新</div><div>取消</div></div><div><div>错误</div><div>价格更新失败: (1644, '价格降幅不能超过100%')</div><div>确定</div></div></div></div> | 2043 | 2043 | 1 | 1700.00 | 1530.00 | 2043 | 2043 | 1 | 1530.00 | 1377.00 |
| 2043 | 2043 | 1 | 1700.00 | 1530.00 | | | | | | | |
| 2043 | 2043 | 1 | 1530.00 | 1377.00 | | | | | | | |
| 备注 | <p>存储过程的优势：</p> <p>将复杂业务逻辑封装在数据库层，减少网络往返次数，保证操作的原子性</p> <p>提高安全性（避免 SQL 注入），便于统一维护和优化</p> | | | | | | | | | | |

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) | <p>查询销售汇总数据，提供按品牌分类的销售统计分析，包括：</p> <p>各品牌销售总量，销售总收入，平均销售价格</p> <p>支持按品牌类别筛选，结果可导出为 Excel。</p> |
| 视图功能描述(1分) | <p>创建视图 sales_summary_view 实现：</p> <p>多表关联聚合查询，按品牌分组统计关键销售指标</p> <p>计算衍生字段：</p> <p>销售总量（COUNT），总收入（SUM），均价（AVG）</p> <p>结果按总收入降序排列</p> |
| 涉及的关系表(2分) | <p>Sale（销售记录表），Inventory（库存车辆表），Model（车型表）</p> <p>Series（车系表），Brand（品牌表）</p> |
| 表连接字段（1 分） | <p>Sale.inventory_id = Inventory.inventory_id</p> <p>Inventory.model_id = Model.model_id</p> <p>Model.series_id = Series.series_id</p> <p>Series.brand_id = Brand.brand_id</p> |
| 创建视图代码(3分) | <pre> 1 CREATE VIEW vw_sales_summary AS 2 SELECT 3 b.name AS brand, 4 COUNT(s.sale_id) AS total_sales, 5 SUM(s.final_price) AS total_revenue, 6 AVG(s.final_price) AS avg_price 7 FROM Sale s 8 JOIN Inventory i ON s.inventory_id = i.inventory_id 9 JOIN Model m ON i.model_id = m.model_id 10 JOIN Series sr ON m.series_id = sr.series_id 11 JOIN Brand b ON sr.brand_id = b.brand_id 12 GROUP BY b.name; </pre> |
| 查询代码（3 分） | <pre> class MainController: 2 用法 def execute_query(self): 4 个用法 (3 个动态) elif query_type == "sale": self.query_sales() </pre> |


```

class MainController: 2 用法

def query_sales(self): 1 个用法
    start_date = self.view.start_date.get()
    end_date = self.view.end_date.get()
    customer = self.view.customer_query.get()
    min_price = self.view.min_price.get()
    max_price = self.view.max_price.get()
    conditions = []
    params = []
    if start_date:
        try:
            datetime.strptime(start_date, format: '%Y-%m-%d')
            conditions.append("s.sale_date >= %s")
            params.append(start_date)
        except ValueError:
            self.show_message(title: "错误", message: "开始日期格式错误, 请使用 YYYY-MM-DD 格式!", is_error=True)
            return
    if end_date:
        try:
            datetime.strptime(end_date, format: '%Y-%m-%d')
            conditions.append("s.sale_date <= %s")
            params.append(end_date)
        except ValueError:
            self.show_message(title: "错误", message: "结束日期格式错误, 请使用 YYYY-MM-DD 格式!", is_error=True)
            return
    if customer:
        conditions.append("c.name LIKE %s")
        params.append(f"%{customer}%")

```

```

class MainController: 2 用法

def query_sales(self): 1 个用法

    if min_price and max_price:
        try:
            min_val = float(min_price)
            max_val = float(max_price)
            if min_val > max_val:
                self.show_message(title: "错误", message: "最低价格不能大于最高价格!", is_error=True)
                return
            conditions.append("s.final_price BETWEEN %s AND %s")
            params.extend([min_val, max_val])
        except ValueError:
            self.show_message(title: "错误", message: "请输入有效的价格数字!", is_error=True)
            return
    elif min_price:
        try:
            conditions.append("s.final_price >= %s")
            params.append(float(min_price))
        except ValueError:
            self.show_message(title: "错误", message: "请输入有效的价格数字!", is_error=True)
            return
    elif max_price:
        try:
            conditions.append("s.final_price <= %s")
            params.append(float(max_price))
        except ValueError:
            self.show_message(title: "错误", message: "请输入有效的价格数字!", is_error=True)
            return

```

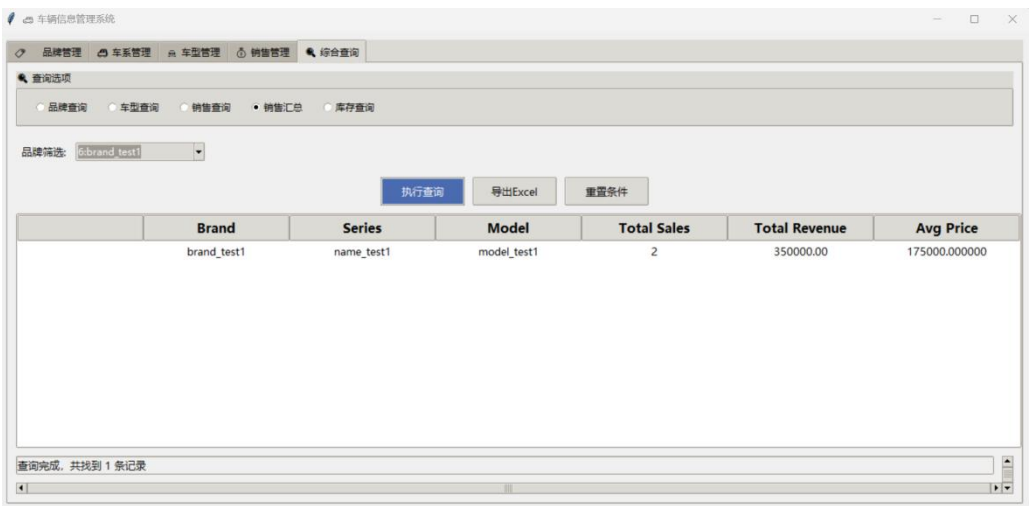
```
where = "WHERE " + " AND ".join(conditions) if conditions else ""
query = f"""
    SELECT s.sale_id, c.name as customer, i.brand, i.series, i.model,
           s.sale_date, s.final_price
    FROM Sale s
    JOIN Customer c ON s.customer_id = c.customer_id
    JOIN (
        SELECT i.inventory_id, b.name AS brand, s.name AS series, m.name AS model
        FROM Inventory i
        JOIN Model m ON i.model_id = m.model_id
        JOIN Series s ON m.series_id = s.series_id
        JOIN Brand b ON s.brand_id = b.brand_id
    ) i ON s.inventory_id = i.inventory_id
    {where}
    ORDER BY s.sale_date
"""

results = self.db.execute_query(query, params)
self.view.display_query_results(results)
```

该视图通过五表关联构建销售分析报表。主要计算三个关键指标：COUNT(s.sale_id)统计每个品牌的销售总量，SUM(s.final_price)计算总销售收入，AVG(s.final_price)得出平均售价。视图使用 GROUP BY 按品牌名称分组，结果集包含 brand、total_sales、total_revenue 和 avg_price 四个字段。应用程序查询该视图时，只需执行简单 SELECT 而无需处理复杂 JOIN，且数据总是实时计算的最新结果。

```
# 查询视图
def query_sales_summary(self):
    self.cursor.execute("SELECT * FROM vw_sales_summary ORDER BY total_revenue DESC")
    self.display_query_results()
```

程序演示
(4 分)



点击主界面“综合查询”选项卡，系统显示查询条件表单和结果表格区域。在查询类型选择区域点击“销售汇总”单选按钮。在品牌筛选下拉框选择 brand_test1，点击“执行查询”按钮，系统自动调用 query_sales_summary() 方法。表格显示各品牌销售数据，包含：brand（品牌名称）total_sales（销售总量）total_revenue（总收入）avg_price（平均价格）状态栏显示“查询完成，共找到 1 条记录”，数据默认按 total_revenue 降序排列。

| | <div><div>vw_sales_summary @vehicle_information (localhost_3306) - 视图</div><div><div>文件 编辑 查看 窗口 帮助</div><div><div>开始事务 文本 筛选 排序 列 导出 创建图表</div><table><tr><th>brand</th><th>series</th><th>model</th><th>total_sales</th><th>total_revenue</th><th>avg_price</th></tr><tr><td>brand_test1</td><td>name_test1</td><td>model_test1</td><td>2</td><td>350000.00</td><td>175000.000000</td></tr></table></div></div></div> <div>在数据库视图中查看，发现确实如此。</div> | brand | series | model | total_sales | total_revenue | avg_price | brand_test1 | name_test1 | model_test1 | 2 | 350000.00 | 175000.000000 |
|-------------|--|-------------|-------------|---------------|---------------|---------------|-----------|-------------|------------|-------------|---|-----------|---------------|
| brand | series | model | total_sales | total_revenue | avg_price | | | | | | | | |
| brand_test1 | name_test1 | model_test1 | 2 | 350000.00 | 175000.000000 | | | | | | | | |
| 备注 | 简化复杂查询逻辑，保证数据一致性（实时计算），提高查询性能（可视化） | | | | | | | | | | | | |