

# SQL 作业解答

考虑关系模式 `product(product_no, name, price)`，完成下面的题目：

所有代码可见根目录 `code` 文件夹 [w6\\_sql.sql](#) [查看源码](#)

## 1 题目一

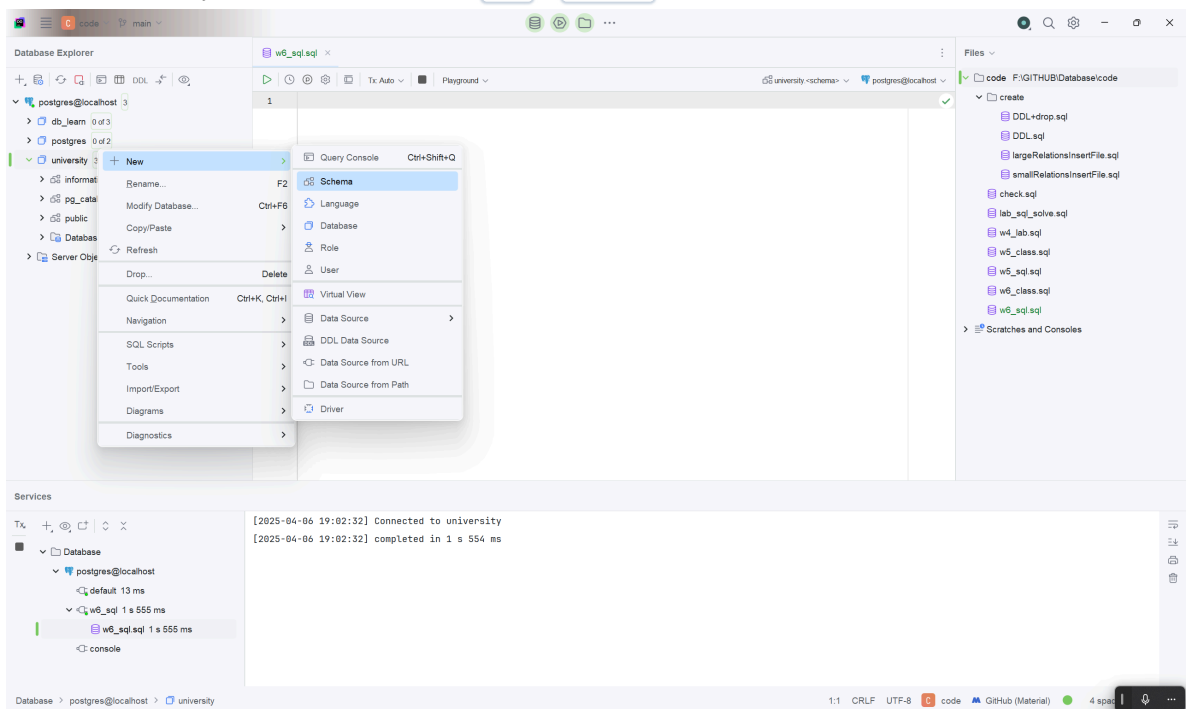
在数据库中创建该关系，并自建上面关系的 `txt` 数据文件，分别采用两种方式导入数据（要求截图重要步骤）：

1. 在 DataGrip 中可视化操作。
2. 使用 `COPY` 命令。

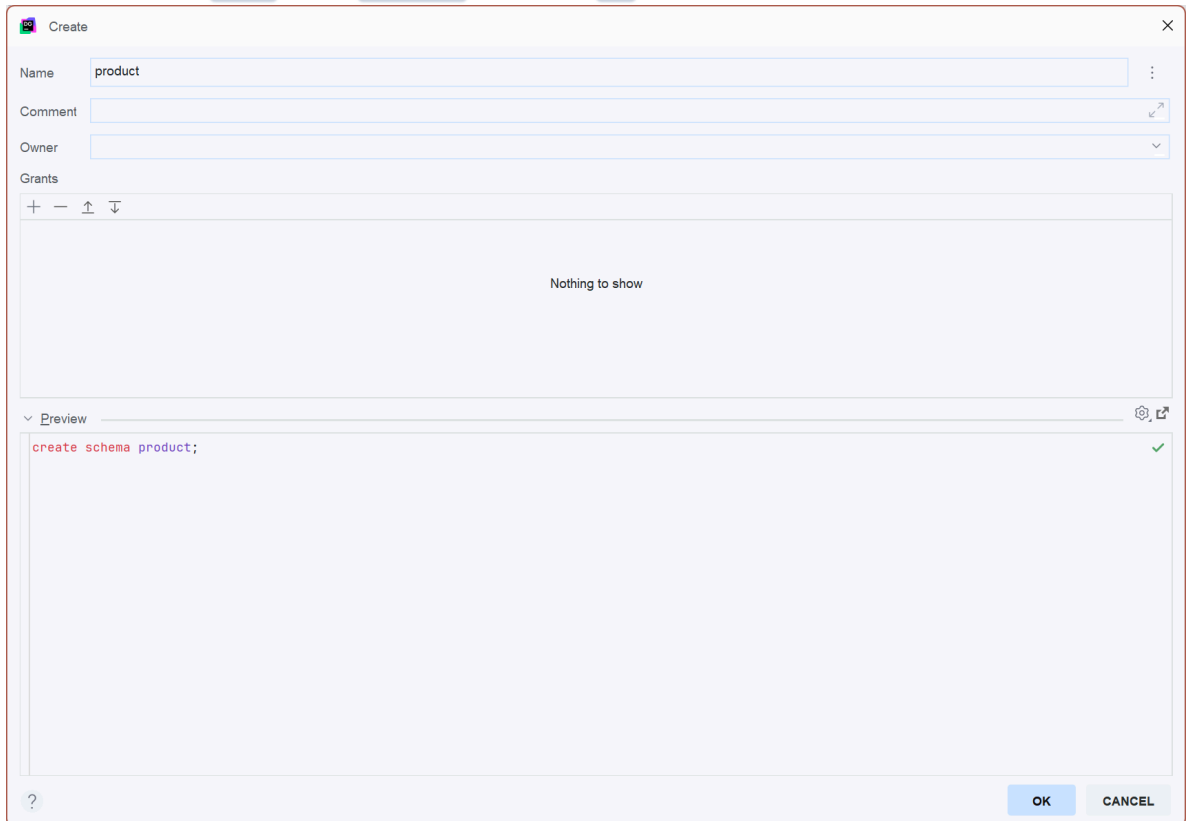
数据准备:本次数据来源于[CSDN](#), 通过事前删除无关列后经由[duplicate.py](#)脚本去重, 删去空值后生成数据文件为 [data.txt](#)

### 1.1 关系创建

1. 首先选择 `university` 数据库,右键菜单中选择 `New` -> `Schema`

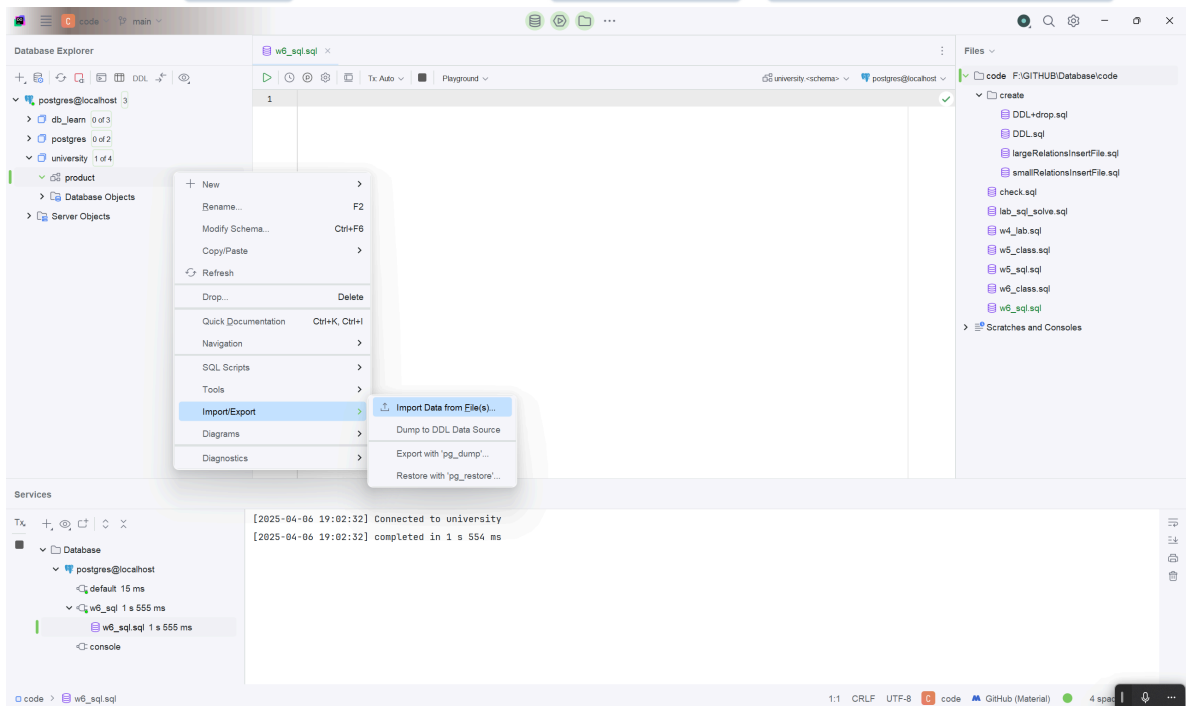


2. 在弹出的对话框中 **Name** 处输入 **product**，然后点击 **OK** 即创建成功

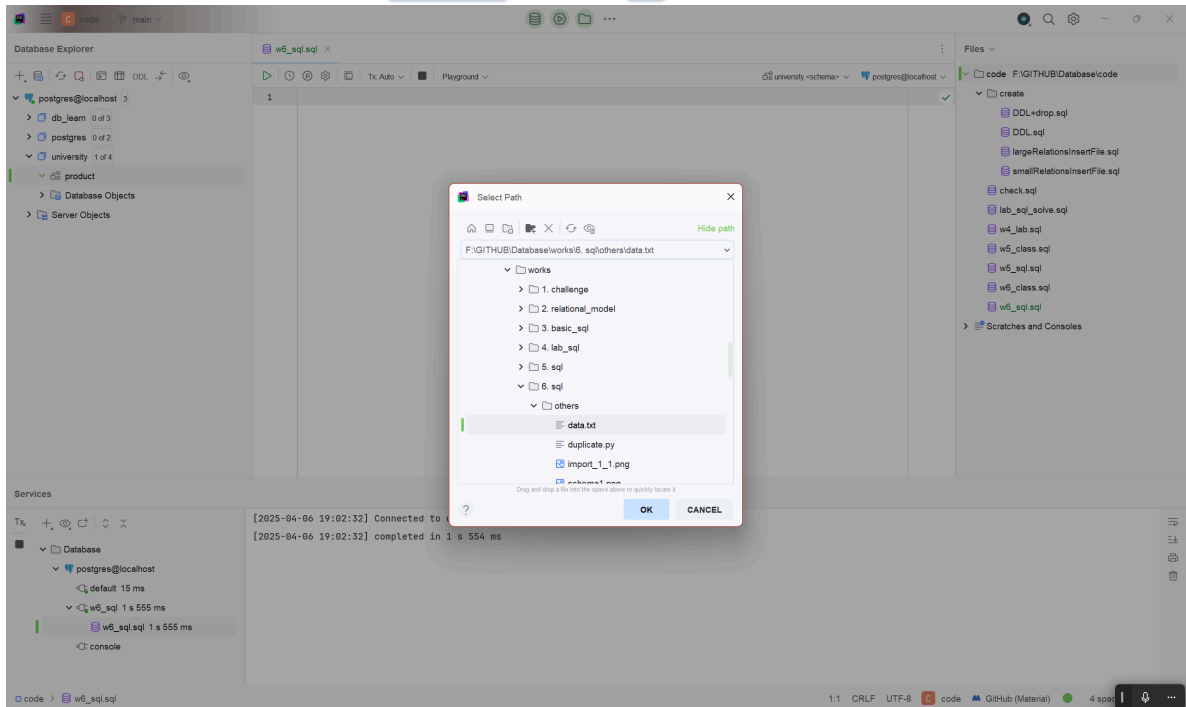


## 1.2 方法 1

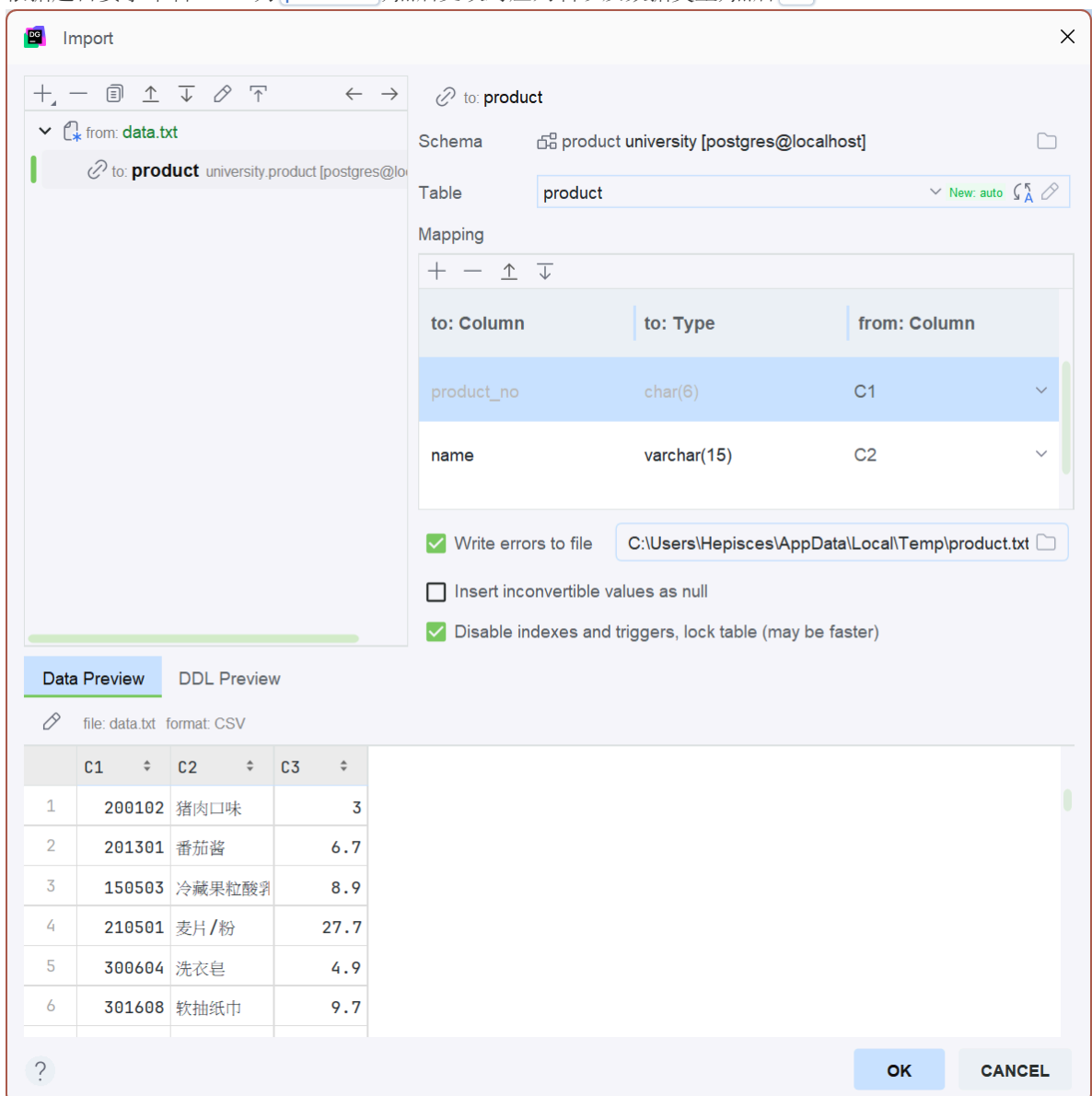
1. 在之前创建好的 **product** 关系上右键菜单中选择 **Import/Export** -> **Import Data from File(s)**



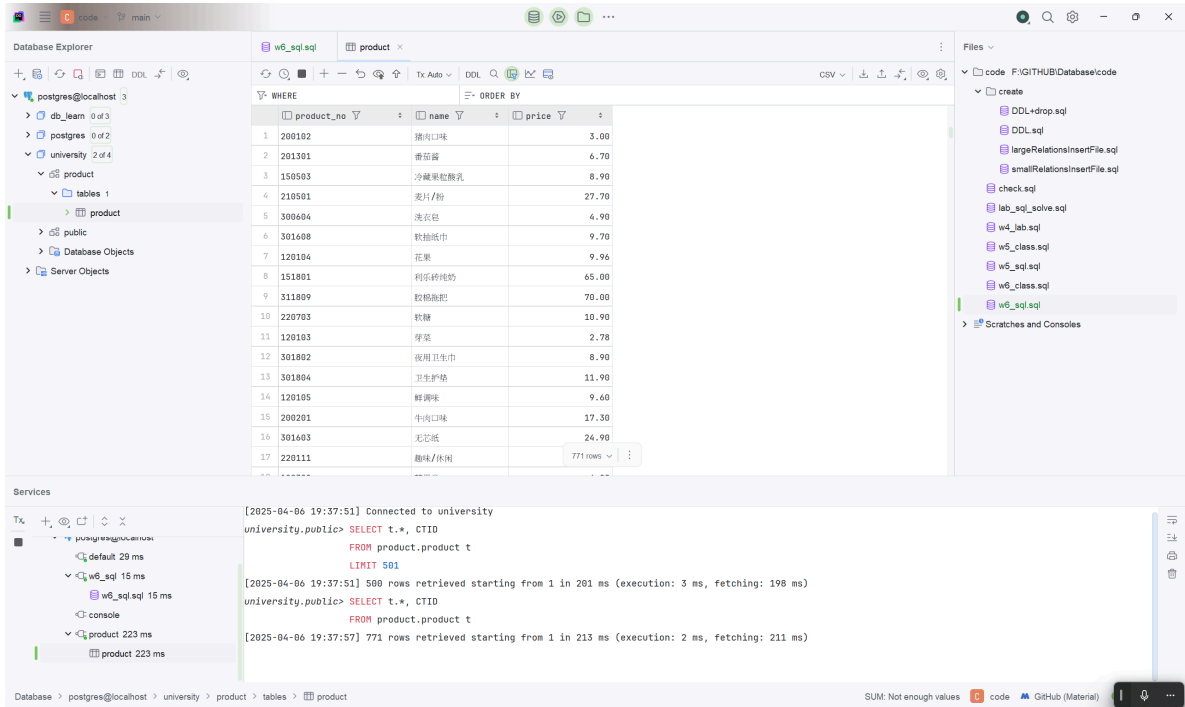
2. 在弹出的对话框找到之前创建好的 `data.txt` 文件, 点击 `OK`



3. 根据题目要求命名 Table 为 `product`, 然后更改对应列名以及数据类型, 然后 `OK`



4. 如图, 导入成功, 共 771 行, 与 txt 中相符

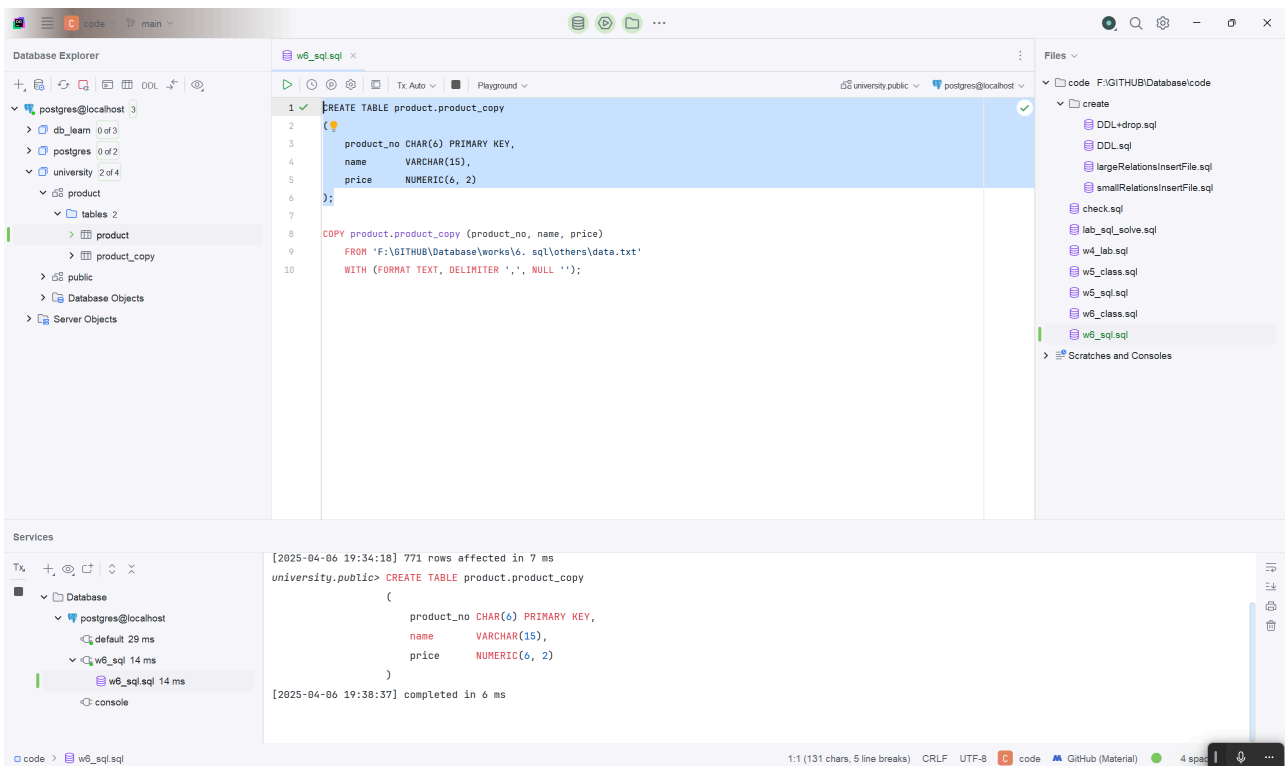


## 1.3 方法 2

1. 通过代码先创建表, 命名为 `product_copy`

```
1 CREATE TABLE product.product_copy
2 (
3     product_no CHAR(6) PRIMARY KEY,
4     name        VARCHAR(15),
5     price        NUMERIC(6, 2)
6 );
```

执行后如图, 成功创建空表

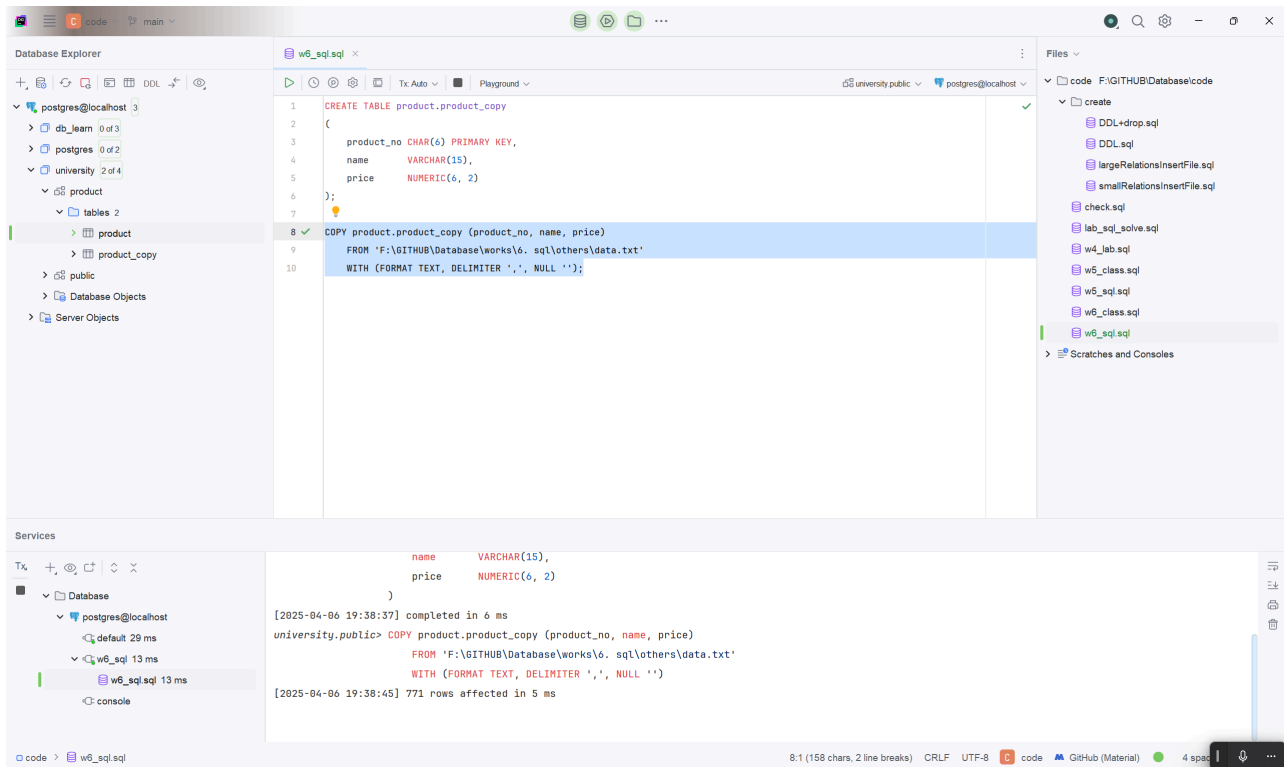


```

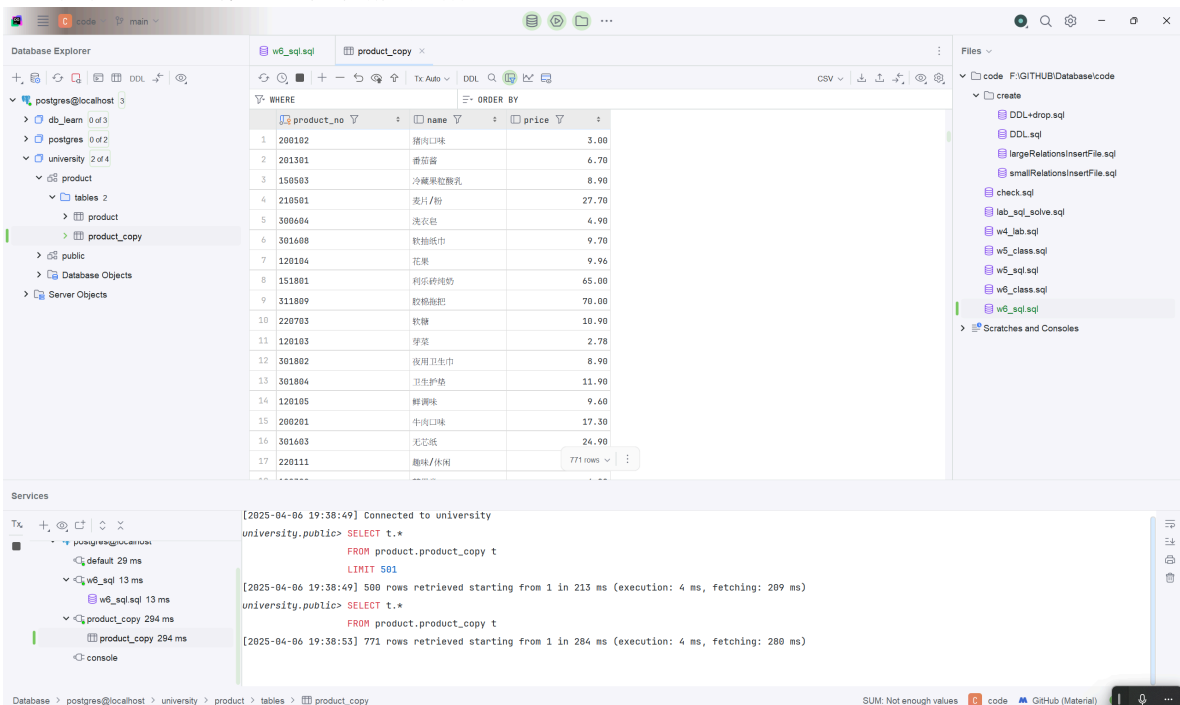
1 COPY product.product_copy (product_no, name, price)
2 FROM 'F:\GITHUB\Database\works\6. sql\others\data.txt'
3 WITH (FORMAT TEXT, DELIMITER ',', NULL '');

```

执行后如图, 成功导入数据



3. 查看表, 共 771 行, 与方法一和原始文件相符



## 2 题目二

1. 添加一个新的商品, 编号为 **666**, 名字为 **cake**, 价格不详。
2. 使用一条 SQL 语句同时添加 3 个商品, 内容自拟。
3. 将商品价格统一打 8 折。
4. 将价格大于 100 的商品上涨 2%, 其余上涨 4%。

5. 将名字包含 `cake` 的商品删除。
6. 将价格高于平均价格的商品删除。

## 2.1 通过 `INSERT` 语句添加(这里由于定义原因, 补全编号为 6 位)

```
1 INSERT INTO product.product (product_no, name, price)
2 VALUES (000666, 'cake', NULL);
```

执行后如图, 成功添加

The screenshot shows a database IDE interface with the following components:

- Database Explorer:** Shows the database structure. The 'product' table is highlighted under the 'university' database.
- SQL Editor:** Contains the following SQL script:

```
1 CREATE TABLE product.product_copy
2 (
3     product_no CHAR(6) PRIMARY KEY,
4     name VARCHAR(15),
5     price NUMERIC(6, 2)
6 );
7
8 COPY product.product_copy (product_no, name, price)
9 FROM 'F:\GITHUB\Database\works\6. sql\others\data.txt'
10 WITH (FORMAT TEXT, DELIMITER ',', NULL '');
11
12 -- 2-1
13 INSERT INTO product.product (product_no, name, price)
14 VALUES (product_no 000666, name 'cake', price NULL);
```
- Table Viewer:** Displays the 'product' table with columns 'product\_no', 'name', and 'price'. The new record (666, cake, NULL) is visible at the bottom.
- Services:** Shows the execution history. The last query executed was the INSERT statement, which completed successfully in 2 ms.

## 2.2 仍然使用 `INSERT` 语句添加

```
1 INSERT INTO product.product (product_no, name, price)
2 VALUES ('999991', 'iPhone', 999.99),
3         ('999992', 'MacBook', 1299.99),
4         ('999993', 'AirPods', 199.99);
```

执行后如图, 成功添加

The screenshot shows a database IDE with the following components:

- Database Explorer:** Shows a PostgreSQL database with a schema named 'product' containing a table 'product'.
- SQL Editor:** Contains the following SQL script:

```
1 CREATE TABLE product.product_copy
2 (
3     product_no CHAR(6) PRIMARY KEY,
4     name VARCHAR(15),
5     price NUMERIC(6, 2)
6 );
7
8 COPY product.product_copy (product_no, name, price)
9 FROM 'F:\GITHUB\Database\works\6. sql\others\data.txt'
10 WITH (FORMAT TEXT, DELIMITER ',', NULL '');
11
12 -- 2-1
13 INSERT INTO product.product (product_no, name, price)
14 VALUES (product_no 000666, name 'cake', price NULL);
15
16 -- 2-2
17 INSERT INTO product.product (product_no, name, price)
18 VALUES
19 (product_no '999991', name 'iPhone', price 999.99),
20 (product_no '999992', name 'MacBook', price 1299.99),
21 (product_no '999993', name 'AirPods', price 199.99);
```

The execution results show that 1 row was affected in 5 ms for the first insert and 3 rows were affected in 6 ms for the second insert. The 'product' table now contains the following data:

product_no	name	price
151603	奶杯	4.00
110304	干货软足类	1.00
341204	晴纶毯	30.00
201007	菜籽油	89.00
330102	蓝球	23.00
130805	规划画点	5.00
300504	电蚊香液	29.90
311819	香烟拍	2.00
666	cake	<null>
999991	iPhone	999.99
999992	MacBook	1299.99
999993	AirPods	199.99
- Services:** Shows the execution timeline for the SQL script, including the time taken for each statement.

## 2.3 通过 UPDATE 语句更新

- 1 UPDATE product.product
- 2 SET price=price \* 0.8;

执行后如图, 成功打折

The screenshot shows the same database IDE after executing the UPDATE statement. The SQL Editor now contains the following script:

```
1 CREATE TABLE product.product_copy
2 (
3     product_no CHAR(6) PRIMARY KEY,
4     name VARCHAR(15),
5     price NUMERIC(6, 2)
6 );
7
8 COPY product.product_copy (product_no, name, price)
9 FROM 'F:\GITHUB\Database\works\6. sql\others\data.txt'
10 WITH (FORMAT TEXT, DELIMITER ',', NULL '');
11
12 -- 2-1
13 INSERT INTO product.product (product_no, name, price)
14 VALUES (product_no 000666, name 'cake', price NULL);
15
16 -- 2-2
17 INSERT INTO product.product (product_no, name, price)
18 VALUES
19 (product_no '999991', name 'iPhone', price 999.99),
20 (product_no '999992', name 'MacBook', price 1299.99),
21 (product_no '999993', name 'AirPods', price 199.99);
22
23 -- 2-3
24 UPDATE product.product
25 SET price=price*0.8;
```

The execution results show that 1 row was affected in 5 ms for the first insert and 3 rows were affected in 6 ms for the second insert. The 'product' table now contains the following data:

product_no	name	price
200102	猪肉口味	2.40
201301	番茄酱	5.36
150503	冷藏果粒酸乳	7.12
210501	麦片/粉	22.16
300604	洗衣皂	3.92
301608	软抽纸巾	7.76
7120104	花果	7.97
151801	柯乐牌纯奶	52.00
311809	酸粉脆肥	56.00
220703	软糖	8.72
120103	芽菜	2.22
301802	夜用卫生巾	7.12
301804	卫生护垫	9.52
120105	鲜调味	7.68
200201	牛肉口味	13.84
301603	无芯纸	19.92
220111		5.52

## 2.4 通过 UPDATE 语句更新

```
1 UPDATE product.product
2 SET price = CASE
3     WHEN price > 100 THEN price * 1.02
4     ELSE price * 1.04
5 END;
```

执行后如图, 成功更新

The screenshot shows a database IDE interface. On the left, the 'Database Explorer' pane shows a PostgreSQL database named 'university' with a table 'product'. The main editor displays a SQL script with several statements, including an UPDATE statement that is highlighted. The script includes INSERT statements for 'cake', 'iPhone', 'MacBook', and 'AirPods', followed by an UPDATE statement that increases prices by 2% for items over 100 and by 4% for others. A SELECT statement is also present. The right pane shows a table view of 'product' with columns 'product\_no', 'name', and 'price'. The table contains 14 rows of data, including the 'cake' entry which has a price of 0.00. The bottom pane shows the execution log, indicating that the UPDATE statement was completed successfully and that 500 rows were retrieved by the subsequent SELECT statement.

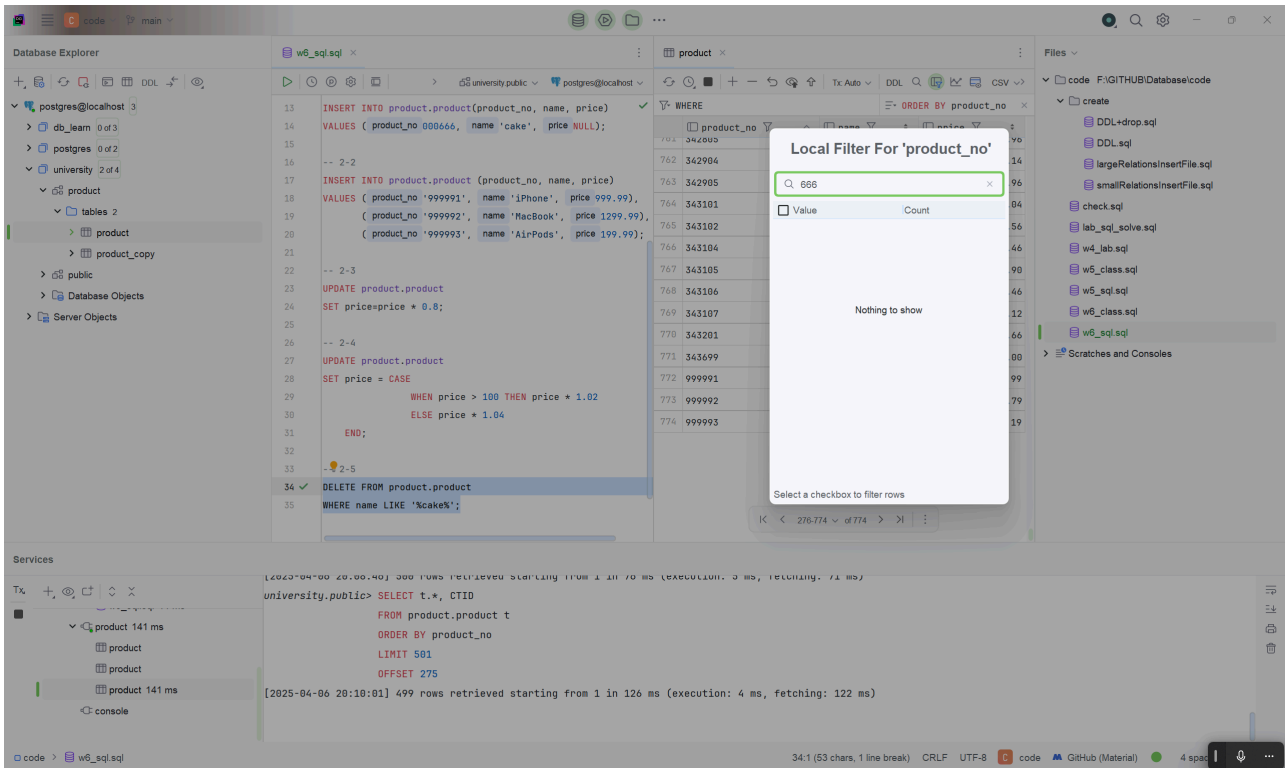
product_no	name	price
342904	文饰运动鞋	19.14
342905	女运动鞋	24.96
343101	男塑料拖鞋	24.04
343102	女塑料拖鞋	16.56
343104	男布/绒面拖鞋	22.46
343105	女布/绒面拖鞋	9.90
343106	童布/绒面拖鞋	27.46
343107	鞋套	29.12
343201	鞋垫	6.66
343699	其他时装包袋	0.00
666	cake	<null>
999991	iPhone	815.99
999992	MacBook	1060.79
999993	AirPods	163.19

## 2.5 通过 DELETE 语句删除

```
1 DELETE
2 FROM product.product
3 WHERE name LIKE '%cake%';
```



执行后如图, 成功删除

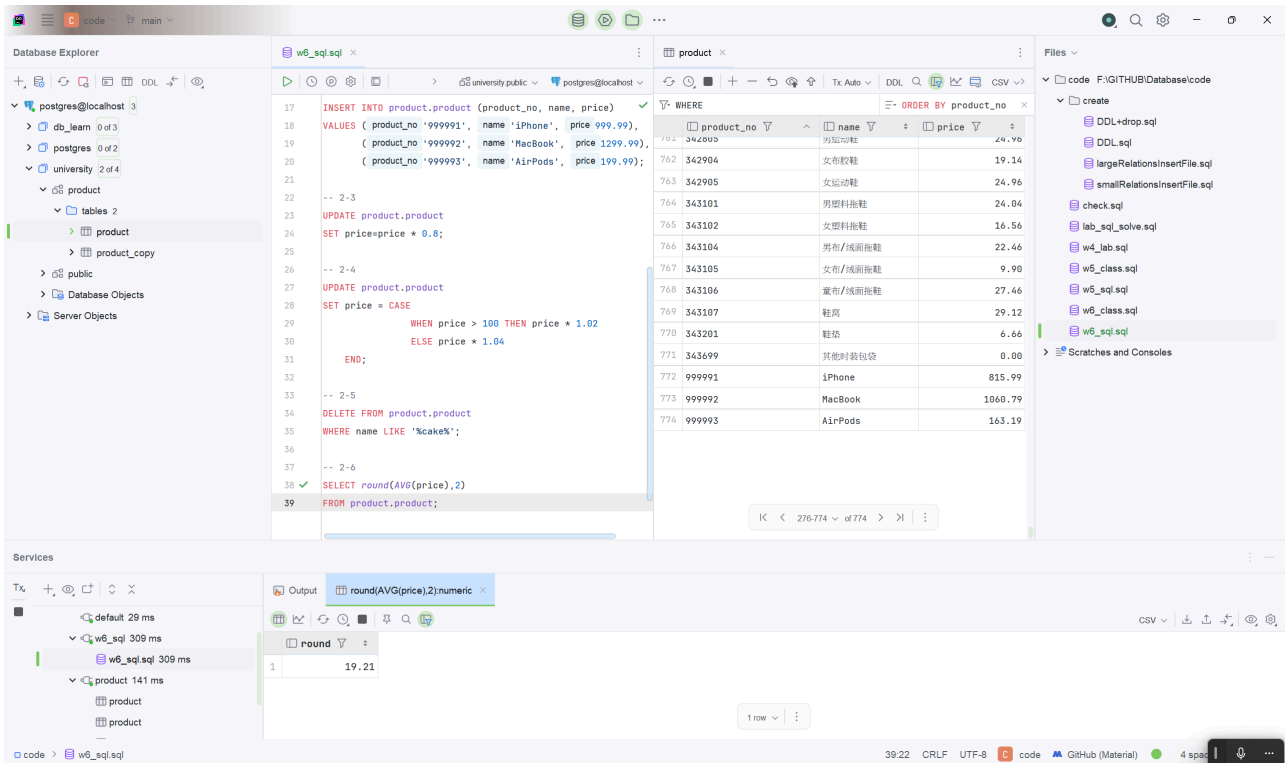


## 2.6 通过DELETE语句删除

首先查看平均价格

- 1 SELECT AVG(price)
- 2 FROM product.product;

执行后如图, 平均价格为 19.21



然后通过DELETE语句删除

```

1 DELETE
2 FROM product.product
3 WHERE price > (SELECT AVG(price) FROM product.product);

```

如图, 成功删除, 大于 19.21 的商品已经不在表格中, 包括第二小问添加的三个商品

The screenshot displays a PostgreSQL database management interface. The left sidebar shows the 'Database Explorer' with the 'product' table selected. The main editor shows a SQL script with the following content:

```

21
22 -- 2-3
23 UPDATE product.product
24 SET price=price * 0.8;
25
26 -- 2-4
27 UPDATE product.product
28 SET price = CASE
29     WHEN price > 100 THEN price * 1.02
30     ELSE price * 1.04
31 END;
32
33 -- 2-5
34 DELETE FROM product.product
35 WHERE name LIKE '%cake%';
36
37 -- 2-6
38 SELECT round(AVG(price),2)
39 FROM product.product;
40
41 DELETE FROM product.product
42 WHERE price > (SELECT AVG(price) FROM product.product);

```

The right sidebar shows the 'product' table view with columns: product\_no, name, and price. The table contains 592 rows, with the last row (592) having a price of 0.00.

The bottom section shows the 'Services' tab with a query execution log. The log indicates that the query was executed successfully, returning 317 rows in 71 ms (execution: 4 ms, fetching: 67 ms).