

# 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：

1. 使用 `COPY` 命令导入数据库（PostgreSQL）；或使用 `LOAD DATA` 命令导入数据库（MySQL）。
2. 将该关系导出为任意文件（如SQL、Txt、CSV、JSON等）。

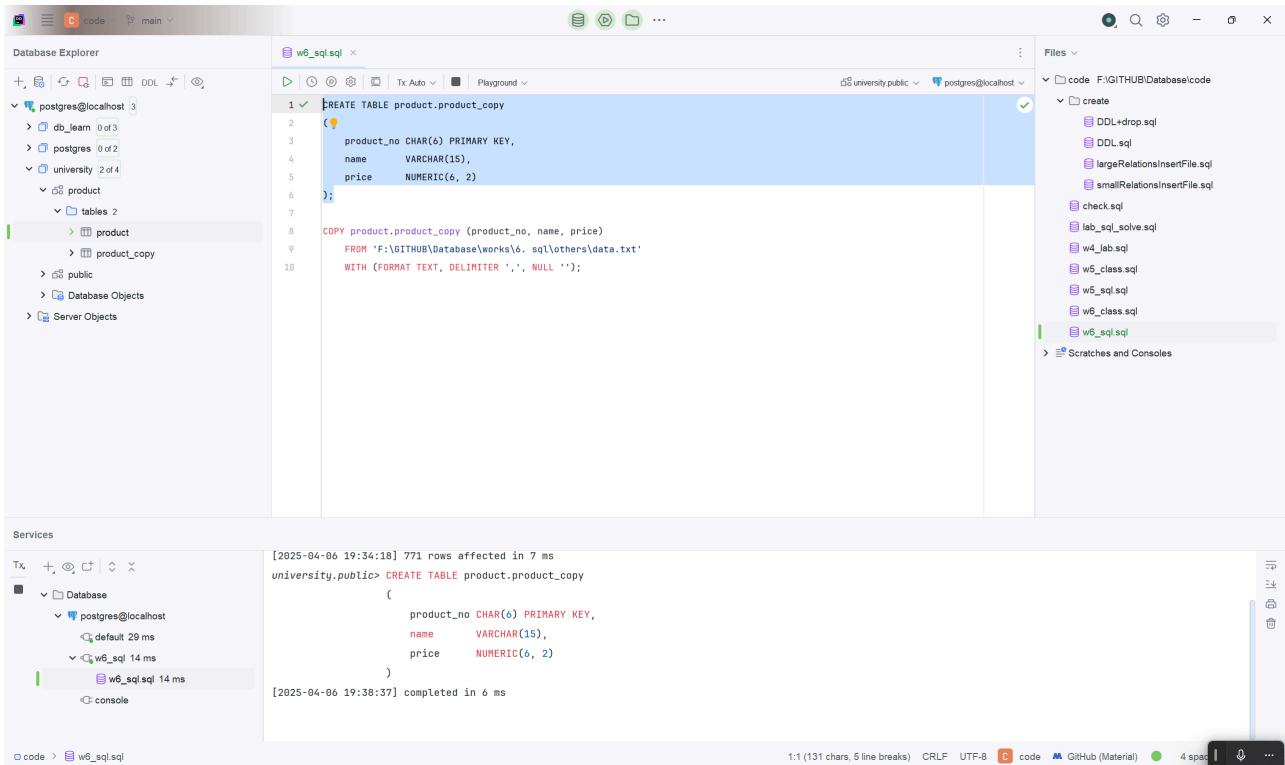
数据准备:本次数据来源于[CSDN](#)，通过事前删除无关列后经由[duplicate.py](#)脚本去重，删去空值后生成数据文件为[data.txt](#)

### 1.1 COPY命令

1. 通过代码先创建表, 命名为 `product`

```
1 CREATE TABLE product.product
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



2. 使用 `COPY` 命令导入数据, 设定分隔符为 `,`, 绝对路径导入

```

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

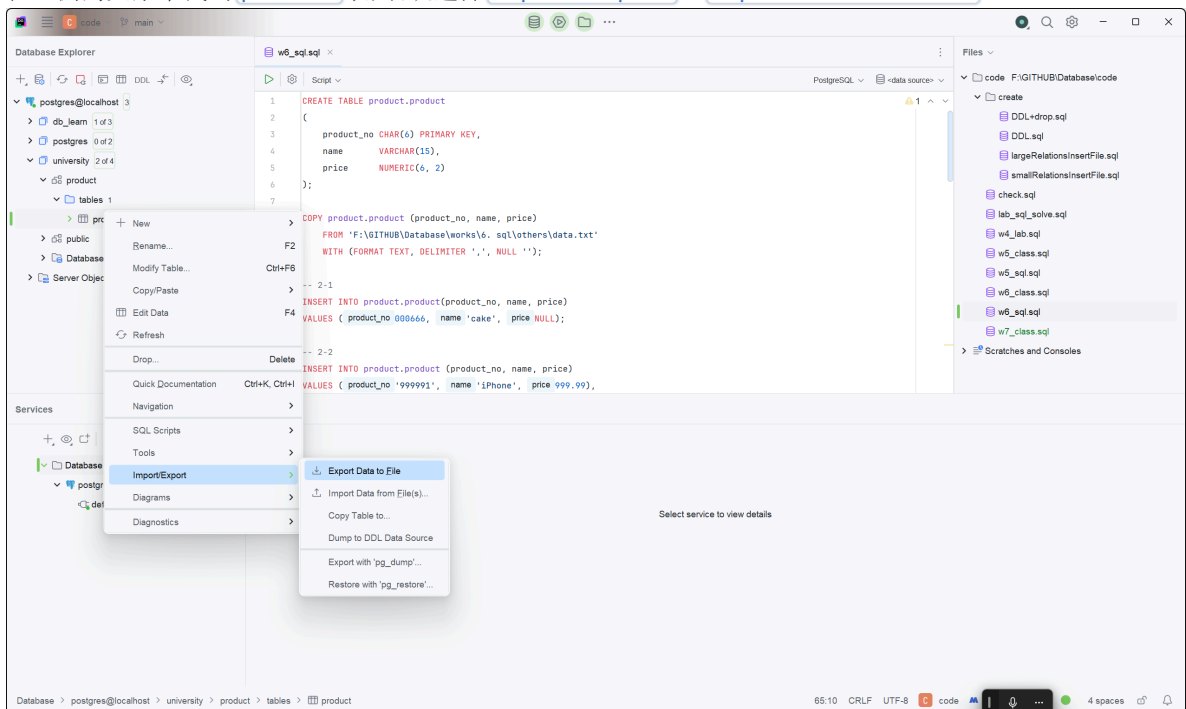
```

执行后如图, 成功导入数据

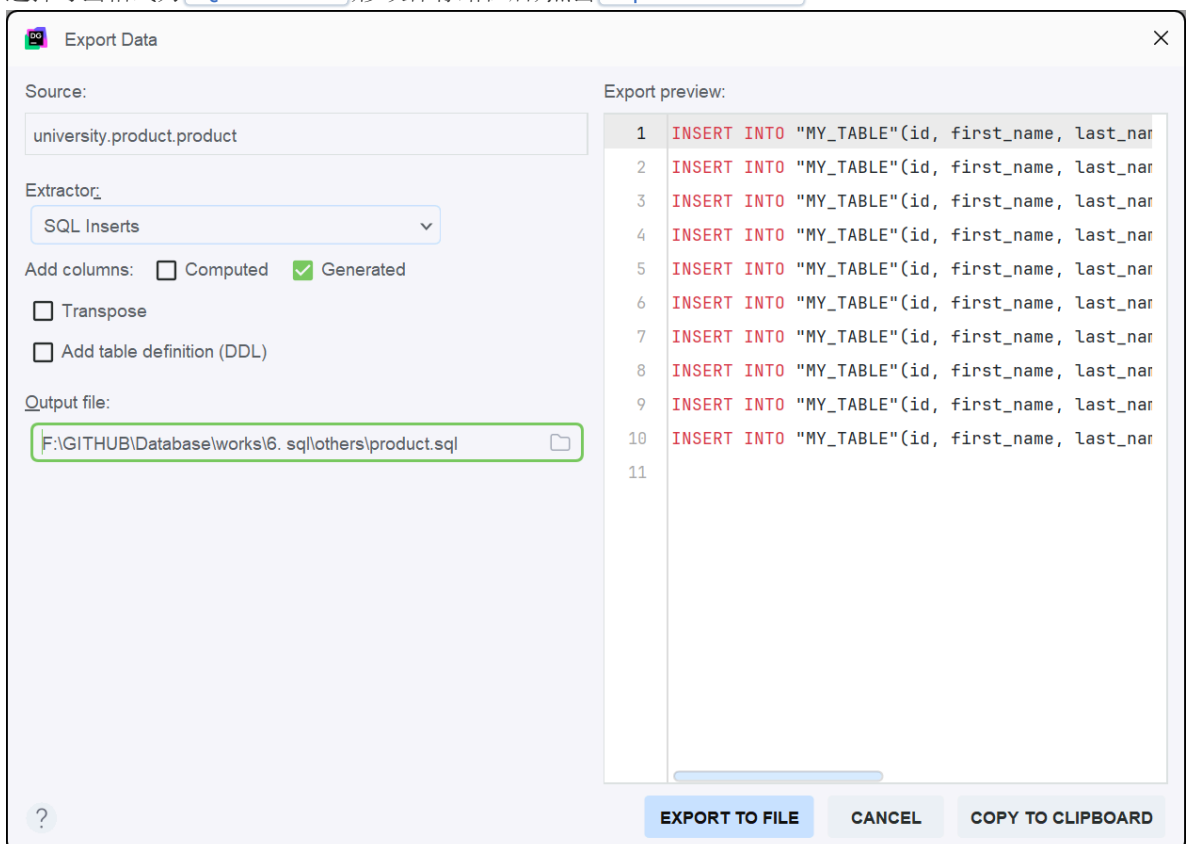
3. 查看表, 共 771 行, 与 txt 中相符

## 1.2 数据导出

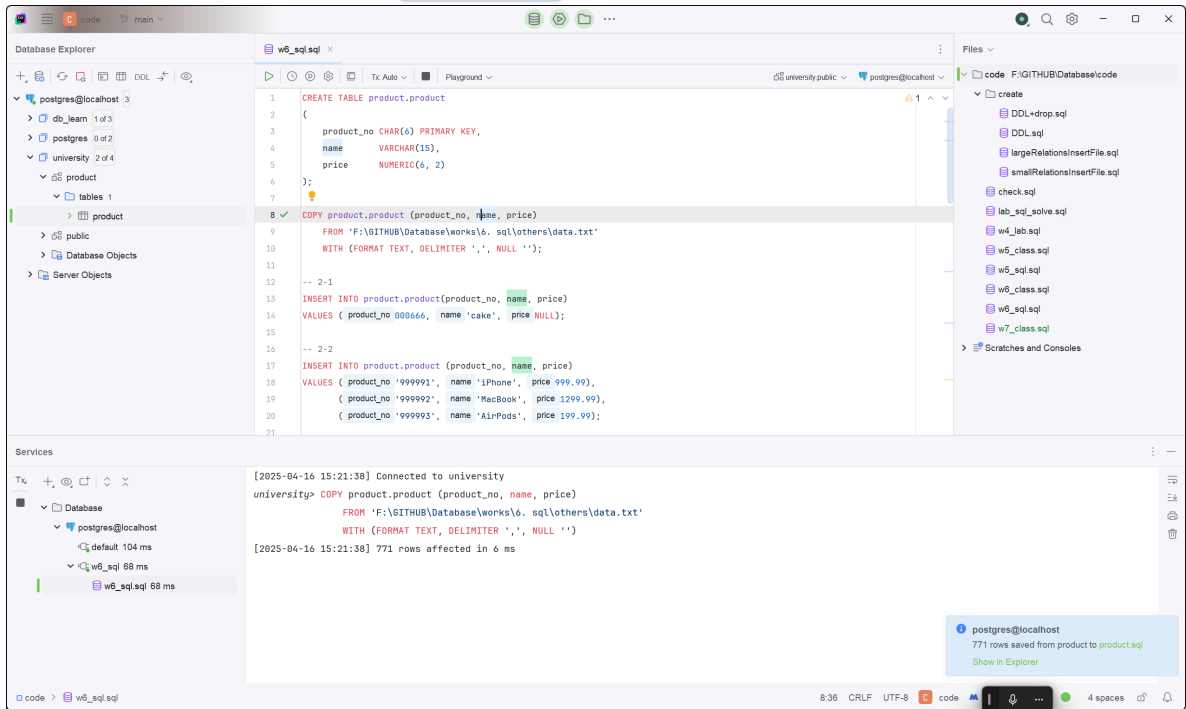
1. 在左侧的关系中找到 `product` 表, 右键选择 `import/export` -> `export data to file`



2. 选择导出格式为 `SQL Inserts`, 修改保存路径后, 点击 `export to file`



3. 右下角及终端提示成功, 导出文件为 `product.sql`, 可以查看数据: `product.sql`



## 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 1. 通过 `INSERT` 语句添加

- ```
1 INSERT INTO product.product(product_no, name, price)
2 VALUES (666, 'cake', NULL);
```

执行后如图, 成功添加

The screenshot shows a database IDE with the following components:

- Database Explorer:** Shows the database structure, including the 'product' table and its 'product\_copy' table.
- 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 View:** Displays the 'product' table with columns 'product\_no', 'name', and 'price'. The data includes rows for various products, with the last row being '666', 'cake', and '<null>'.
- Services:** Shows the execution of the SQL script, indicating that 500 rows were retrieved starting from 1 in 126 ms (execution: 2 ms, fetching: 124 ms).

## 2.2 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 the database structure, including the 'product' table and its 'product\_copy' table.
- 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);
```
- Table View:** Displays the 'product' table with columns 'product\_no', 'name', and 'price'. The data includes rows for various products, with the last row being '999993', 'AirPods', and '199.99'.
- Services:** Shows the execution of the SQL script, indicating that 1 row was affected in 5 ms and 3 rows were affected in 6 ms.

## 2.3 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 displays a PostgreSQL database interface with the following components:

- Database Explorer:** Shows the database structure, including the 'product' table and its 'product\_copy' table.
- SQL Editor:** Contains the SQL script for creating a copy table and updating the original table.

```
1 CREATE TABLE product.product_copy
2 (
3     name VARCHAR(15),
4     price NUMERIC(6, 2)
5 );
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 (product_no '999991', name 'iPhone', price 999.99),
19         (product_no '999992', name 'MacBook', price 1299.99),
20         (product_no '999993', name 'AirPods', price 199.99);
21
22 -- 2-3
23 UPDATE product.product
24 SET price=price*0.8;
```
- Query Results:** Displays the result of the UPDATE operation, showing the updated prices for the first three rows of the 'product' table.

| 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  |
- Services:** Shows the execution time for the SQL script, which is 16 ms.

## 2.4 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 management tool interface. The left pane displays the 'Database Explorer' with a tree view showing the 'product' table. The main editor shows a SQL script with the following content:

```
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 (product_no '999991', name 'iPhone', price 999.99),
19 (product_no '999992', name 'MacBook', price 1299.99),
20 (product_no '999993', name 'AirPods', price 199.99);
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;
```

The right pane shows a table view of the 'product' table with columns 'product\_no', 'name', and 'price'. The table contains 775 rows, with the last row being 'AirPods' with a price of 163.19. The bottom pane shows the 'Services' section with a query execution log indicating that 500 rows were retrieved starting from 1 in 76 ms.

## 2.5 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

The screenshot shows the same database management tool interface as before, but with the SQL script updated to delete rows. The script now includes a DELETE statement:

```
34 DELETE FROM product.product
35 WHERE name LIKE '%cake%';
```

The right pane shows the 'product' table view. A 'Local Filter For 'product\_no'' dialog box is open, showing a search for '666' and a 'Count' of 0. The table view shows that the row with 'cake' has been deleted. The bottom pane shows the 'Services' section with a query execution log indicating that 499 rows were retrieved starting from 1 in 126 ms.

## 2.6 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

The screenshot shows a PostgreSQL IDE interface. The left pane displays the 'Database Explorer' with a tree view of the database structure, including 'product' and 'product\_copy' tables. The main editor pane contains a SQL script with the following queries:

```
17 INSERT INTO product.product (product_no, name, price)
18 VALUES (product_no '999991', name 'iPhone', price 999.99),
19 (product_no '999992', name 'MacBook', price 1299.99),
20 (product_no '999993', name 'AirPods', price 199.99);
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;
```

The right pane shows the 'product' table with columns 'product\_no', 'name', and 'price'. The table contains 14 rows of data, including products like 'iPhone', 'MacBook', 'AirPods', and various shoes.

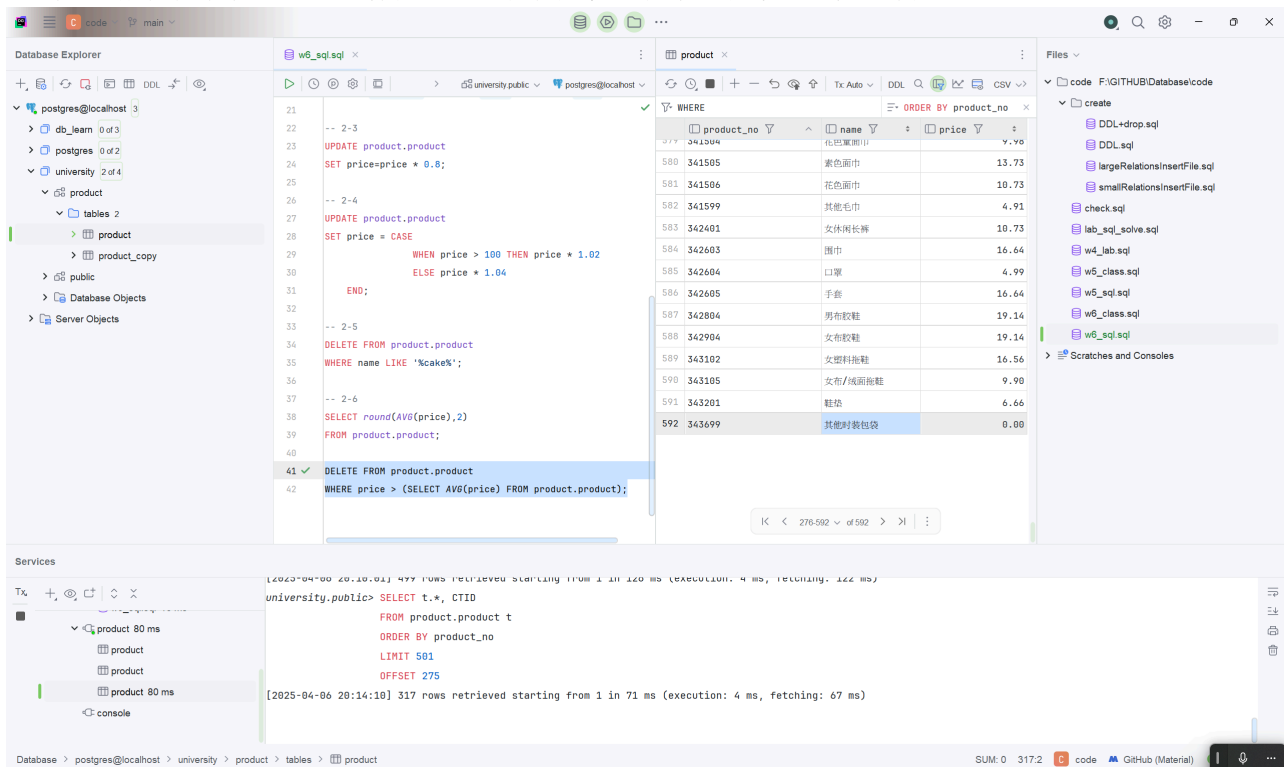
The bottom pane shows the 'Output' window with the result of the final query: a single row with the value '19.21'.

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



### 3 题目三

使用参考下面的语句添加10万条商品,

```
1 -- PostgreSQL Only
2 INSERT INTO product (name, price)
3 SELECT
4   'Product' || generate_series, -- 生成名称 Product1, Product2, ...
5   ROUND((random() * 1000)::numeric, 2) -- 生成0到1000之间的随机价格, 保留2位小数
6 FROM generate_series(1, 100000);
```

比较 `DELETE` 和 `TRUNCATE` 的性能差异。

首先在每次执行删除前都运行如下命令保证数据表相同:

```
1 COPY product.product (product_no, name, price)
2   FROM 'F:\GITHUB\Database\works\6. sql\others\data.txt'
3   WITH (FORMAT TEXT, DELIMITER ',', NULL '');
4
5 INSERT INTO product.product (product_no, name, price)
6 SELECT 't' || generate_series,
7        'Product' || generate_series, -- 生成名称 Product1, Product2, ...
8        ROUND((RANDOM() * 1000)::numeric, 2) -- 生成0到1000之间的随机价格, 保留2位小数
9 FROM GENERATE_SERIES(0, 99999);
```

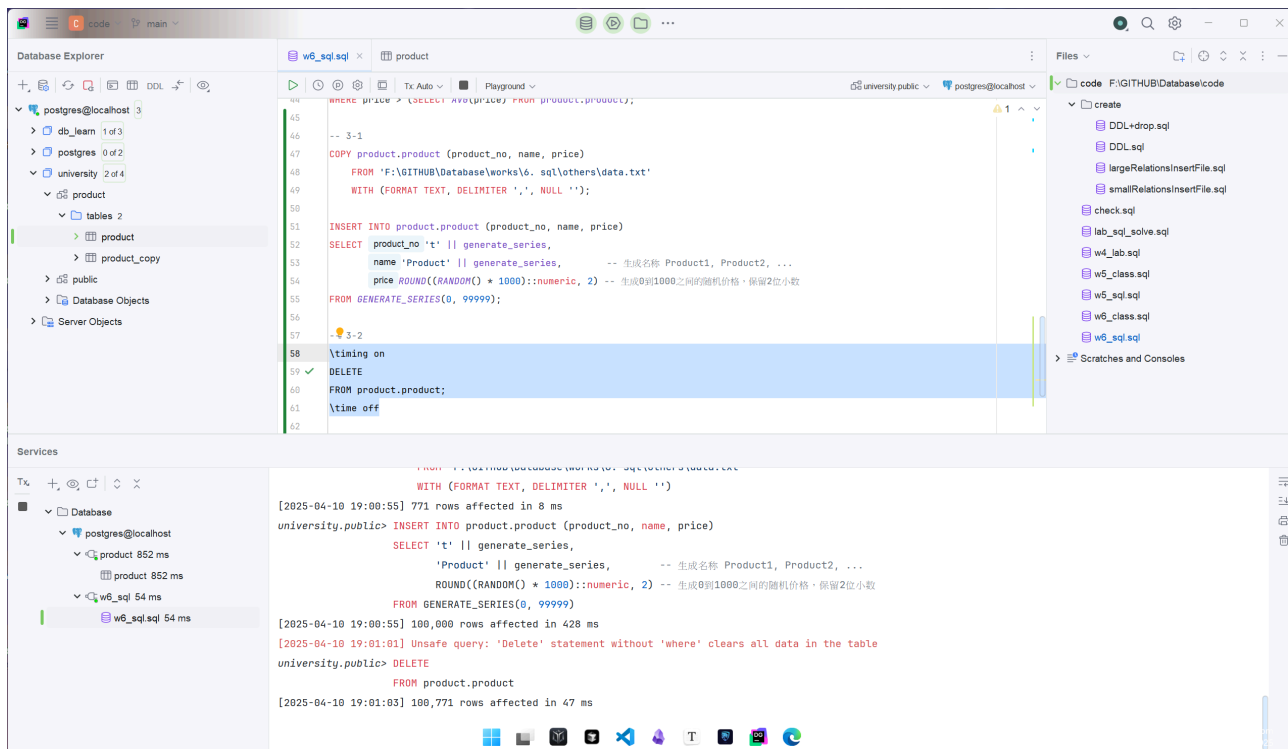
然后分别对 `delete` 和 `truncate` 计时:

- `delete`

执行

```
1 \timing on
2 DELETE
3 FROM product.product;
4 \time off
```

如图:



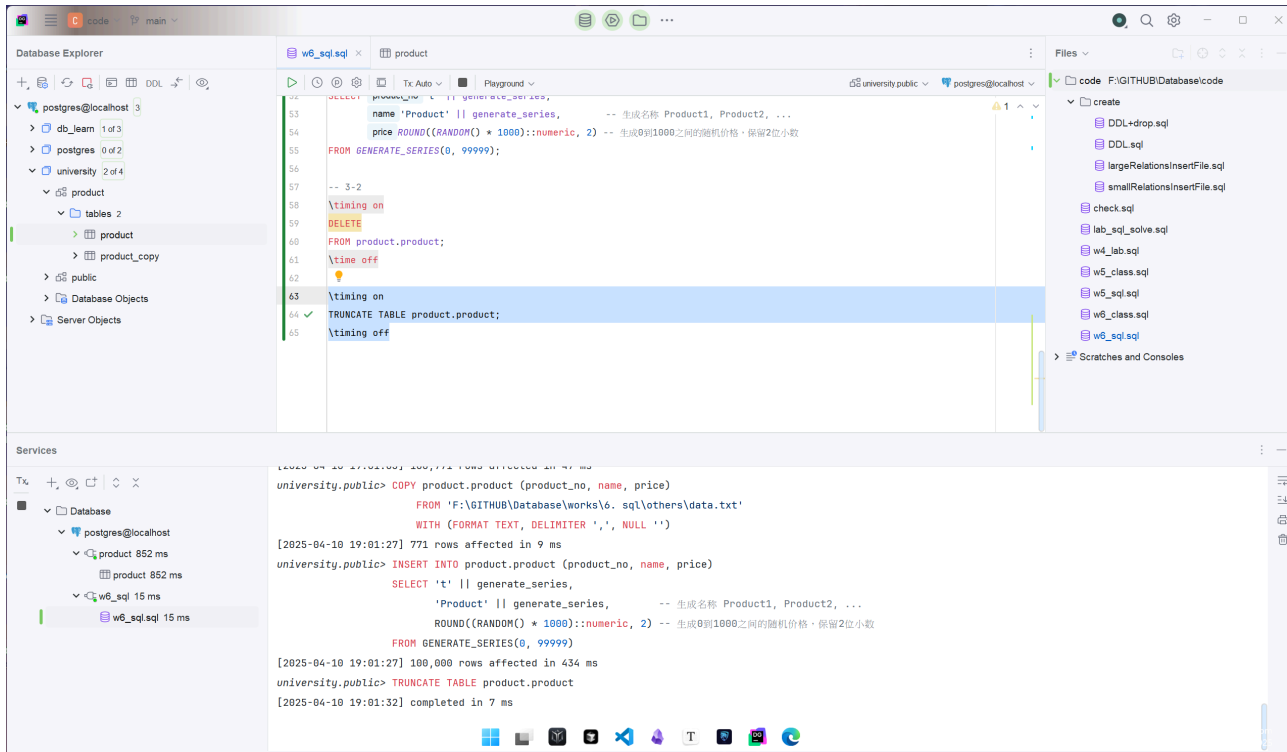
运行时间为 47ms

- truncate

执行

```
1 \timing on
2 TRUNCATE TABLE product.product;
3 \time off
```

如图:



运行时间为**7ms**

结论:在删除表内容上, **truncate** 比 **delete** 性能显著优异