

数据库原理第 5 次作业

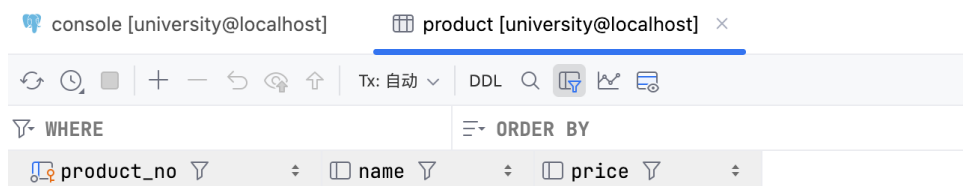
42233099 曾慧鑫

题目一：

1.创建表 product:

```
1 ✓ CREATE TABLE product (  
2     product_no INT PRIMARY KEY,  
3     name VARCHAR(100),  
4     price DECIMAL(10, 2)  
5 );
```

运行成功后，得到表格 product:

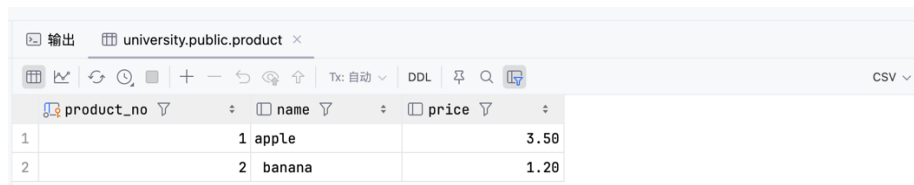


The screenshot shows a database console window with two tabs: 'console [university@localhost]' and 'product [university@localhost]'. The 'product' tab is active, displaying the table structure. The table has three columns: 'product_no' (INT), 'name' (VARCHAR), and 'price' (DECIMAL). The console interface includes a toolbar with various icons and a query input area.

自建 txt 文件，里面包含 apple 和 banana 的产品信息，在命令行使用 copy 导入数据库：

```
\copy product(product_no, name, price) FROM  
'/Users/zenghuixin/Desktop/product.txt' DELIMITER ';' CSV HEADER;
```

运行后，product 表格更新为：



The screenshot shows a database console window with a tab labeled '输出 university.public.product'. The table data is displayed in a grid format. The table has three columns: 'product_no', 'name', and 'price'. The data rows are as follows:

product_no	name	price
1	apple	3.50
2	banana	1.20

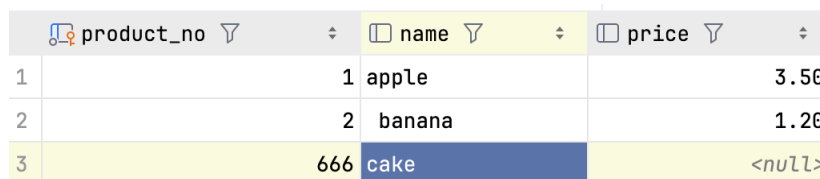
在命令行使用 copy 将 product 导出数据库为 product.csv:

```
\copy product TO '/Users/zenghuixin/Desktop/product.csv' DELIMITER ';' CSV  
HEADER;
```

题目二：

2.1 添加一个新的商品，编号为 666，名字为 cake，价格不详。

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



The screenshot shows a database console window displaying the product table data. The table has three columns: 'product_no', 'name', and 'price'. The data rows are as follows:

product_no	name	price
1	apple	3.50
2	banana	1.20
3	666 cake	<null>

2.2 使用一条 SQL 语句同时添加 3 个商品，内容自拟。

```
INSERT INTO product (product_no, name, price) VALUES
( product_no 667, name 'bread', price 50),
( product_no 668, name 'milk', price 20),
( product_no 669, name 'cheese', price 150);
```

	product_no	name	price
1	1	apple	3.50
2	2	banana	1.20
3	666	cake	<null>
4	667	bread	50.00
5	668	milk	20.00
6	669	cheese	150.00

2.3 将商品价格统一打 8 折。

```
UPDATE product
SET price = price * 0.8
WHERE price IS NOT NULL;
```

	product_no	name	price
1	669	cheese	120.00
2	668	milk	16.00
3	667	bread	40.00
4	666	cake	<null>
5	2	banana	0.96
6	1	apple	2.80

2.4 将价格大于 100 的商品上涨 2%，其余上涨 4%。

```
UPDATE product
SET price = price * 1.02
WHERE price > 100;
UPDATE product
SET price = price * 1.04
WHERE price <= 100 AND price IS NOT NULL;
```

	product_no	name	price
1	666	cake	<null>
2	669	cheese	122.40
3	1	apple	2.91
4	2	banana	1.00
5	667	bread	41.60
6	668	milk	16.64

2.5 将名字包含 cake 的商品删除。

```
DELETE FROM product
WHERE name ILIKE '%cake%';
```

	product_no	name	price
1	1	apple	2.91
2	2	banana	1.00
3	667	bread	41.60
4	668	milk	16.64
5	669	cheese	122.40

2.6 将价格高于平均价格的商品删除

```
DELETE FROM product
WHERE price > (SELECT AVG(price) FROM product WHERE price IS NOT NULL);
```

	product_no	name	price
1	1	apple	2.91
2	2	banana	1.00
3	668	milk	16.64

题目三：

3.1 针对 PostgreSQL

在 product 中插入 10000 个产品信息，分别使用 delete 和 truncate 进行清除：

```
INSERT INTO product (name, price)
SELECT
    name 'Product' || generate_series,
    price ROUND((random() * 1000)::numeric, 2)
FROM generate_series(1, 10000);
```

```
\timing
DELETE FROM product;
```

```
\timing
TRUNCATE TABLE product;
```

运行发现，delete 花费了 145ms，truncate 只使用了 20ms，这是因为 delete 是一行一行删，还要记录 WAL（写前日志），触发器也会执行；而 truncate 直接删除整个表的数据页，重置表，不记录逐行删除，触发器不会执行。

3.2 针对 MySQL

使用 python 语言生成数据文件 product_data.txt，分别使用 LOAD DATA 和 SELECT INTO 的插入：

```
import random

with open('product_data.txt', 'w') as f:
    for i in range(1, 100001):
        name = f'Product{i}'
        price = round(random.uniform(0, 1000), 2)
        f.write(f"{name},{price}\n")
```

```
LOAD DATA INFILE '/Users/zenghuixin/Desktop/product_data.txt'  
INTO TABLE product  
FIELDS TERMINATED BY ','  
LINES TERMINATED BY '\n'  
(name, price);  
  
INSERT INTO product (name, price)  
SELECT name, price FROM product;
```

运行发现 LOAD DATA INFILE 非常快而 INSERT INTO ... SELECT 较慢，这是因为 LOAD DATA INFILE 是专门为批量导入设计的工具。而 INSERT INTO ... SELECT 在数据量大时，需要读取已有数据再插入。