E02 - SQL Review (Part 1)

E02, Business Intelligence

Oemer Furkan Coban

21.11.2025 19:04

Business Intelligence
Winter Term 2025/2026

University of Oldenburg

Accepted: 30 (100%)
Total: 30

1.1 *Select the names of all the products in the store.*

accepted

```sql
SELECT name FROM Products;
```

1.2 *Select the names and the prices of all the products in the store.*

accepted

```sql
SELECT name,price FROM Products;
```

1.3 *Select the name of the products with a price less than or equal to $200.*

accepted

```sql
SELECT name FROM Products WHERE price<= 200;
```

1.4 *Select all the products with a price between $60 and $120.*

accepted

```sql
SELECT * FROM Products WHERE price >= 60 AND price <= 120;
```

1.5 *Select the name and price in cents (i.e., the price must be multiplied by 100).*

accepted

```sql
SELECT name, price*100 FROM Products;
```

1.6 *Compute the average price of all the products.*

accepted

```sql
SELECT avg(price) FROM Products;
```

1.7 *Compute the average price of all products with manufacturer code equal to 2.*

accepted

```sql
SELECT avg(price) FROM Products WHERE manufacturer=2;
```

1.8 *Select the name and price of all products with a price larger than or equal to $180, and sort first by price (in descending order), and then by name (in ascending order).*

accepted

```sql
SELECT name, price FROM Products WHERE price >= 180 ORDER BY price DESC, name ASC;
```

1.9 *Select all the data from the products, including all the data for each product's manufacturer.*

accepted

```sql
SELECT *
FROM products JOIN manufacturers ON products.manufacturer = manufacturers.code;
```

1.10 *Select the product name, price, and manufacturer name of all the products.*

accepted

```sql
SELECT
  products.name AS name,
  products.price,
  manufacturers.name AS name
FROM products
JOIN manufacturers
  ON products.manufacturer = manufacturers.code;
```

1.11 *Select the average price of each manufacturer's products, showing only average price the manufacturer's code.*

accepted

```sql
SELECT
  AVG(products.price),
  manufacturers.code AS manufacturer
FROM products
JOIN manufacturers
  ON products.manufacturer = manufacturers.code
GROUP BY manufacturers.code;
```

1.12 *Select the average price of each manufacturer's products, showing only average price and the manufacturer's name.*

accepted

```sql
SELECT AVG(products.price) AS avg, manufacturers.name AS name
FROM products
JOIN manufacturers ON products.manufacturer = manufacturers.code
GROUP BY manufacturers.name;
```

1.13 *Select the names of manufacturer whose products have an average price larger than or equal to $150. Display average price and manufacturer's name.*

accepted

```sql
SELECT AVG(products.price) AS avg, manufacturers.name AS name
FROM products
JOIN manufacturers ON products.manufacturer = manufacturers.code
GROUP BY manufacturers.name
HAVING AVG(products.price) >= 150;
```

1.14 *Select the name and price of the cheapest product.*

accepted

```sql
SELECT name, price
FROM products
ORDER BY price
LIMIT 1;
```

1.15 *Select the name of each manufacturer along with the name and price of its most expensive product.*

accepted

```
SELECT products.name, products.price, manufacturers.name
FROM products
JOIN manufacturers ON products.manufacturer = manufacturers.code
WHERE (manufacturer, price) IN (
    SELECT manufacturer, MAX(price)
    FROM products
    GROUP BY manufacturer
);
```

2.1 *How many orders exist in total?*

accepted

```
SELECT COUNT(*) from orders;
```

2.2 *Find out how many items of each product were ordered (show product_code and counter only).*

accepted

```
SELECT product_code, COUNT(product_code) FROM orders GROUP BY product_code;
```

2.3 *Find out the product_code of the item that was sold only once.*

accepted

```
SELECT product_code FROM orders GROUP BY product_code HAVING COUNT(product_code)=1;
```

2.4 *Find out and print all the names and codes of the items, which were sold at least once.*

accepted

```
SELECT
    orders.product_code AS code, products.name
FROM products
JOIN orders
    ON products.code = orders.product_code;
```

2.5 *Find out and print the name of the item that was sold exactly one time.*

accepted

```
SELECT products.name from products JOIN orders ON products.code = orders.product_code GROUP
     BY products.name HAVING COUNT(orders.product_code) = 1;
```

2.6 *What is the total revenue generated by all orders?*

accepted

```
SELECT SUM(total_price) AS sum_ FROM orders;
```

2.7 *Find out the total revenue generated by each product.*

accepted

```sql
SELECT product_code, SUM(total_price) AS sum_ FROM orders GROUP BY product_code;
```

2.8 *Find out product_code of the items with the minimum revenue.*

accepted

```sql
SELECT product_code, SUM(total_price) AS sum_ FROM orders GROUP BY product_code ORDER BY sum_
    LIMIT 1;
```

2.9 *Find out product_code of the items with the maximum revenue*

accepted

```sql
SELECT product_code, SUM(total_price) AS sum_ FROM orders GROUP BY product_code ORDER BY sum_
    DESC LIMIT 1;
```

2.10 *Find out the most successful manufacturer (the one, which has the most revenue).*

accepted

```sql
SELECT
    manufacturers.name AS name
FROM orders
JOIN products
    ON orders.product_code = products.code
JOIN manufacturers
    ON products.manufacturer = manufacturers.code
GROUP BY manufacturers.name
ORDER BY SUM(orders.total_price) DESC
LIMIT 1;
```

2.11 *Sort the given orders by the time of the transaction.*

accepted

```sql
SELECT * FROM orders ORDER BY order_time ASC;
```

2.12 *When (on which date) the first order took place?*

accepted

```sql
SELECT order_time FROM orders ORDER BY order_time ASC LIMIT 1;
```

2.13 *When (on which date) the last order took place?*

accepted

```sql
SELECT order_time FROM orders ORDER BY order_time DESC LIMIT 1;
```

2.14 *What is the total amount of days between the first order and the last order (in time dimension). Amount of days should be casted to an integer.*

accepted

```sql
SELECT
    (SELECT order_time FROM orders ORDER BY order_time ASC LIMIT 1) AS start,
    (SELECT order_time FROM orders ORDER BY order_time DESC LIMIT 1) AS end,
    FLOOR(
        EXTRACT(EPOCH FROM (MAX(order_time) - MIN(order_time))) / 86400
    )::INTEGER AS "days between"
FROM orders;
```

2.15 *Find out the transaction (e.g. order) with the wrong price.*

accepted

```sql
 SELECT
    orders.order_id,
    orders.product_code,
    orders.order_time,
    orders.amount,
    orders.total_price,
    products.code,
    products.name,
    products.price,
    products.manufacturer
FROM orders
JOIN products
    ON orders.product_code = products.code
WHERE orders.total_price <> orders.amount * products.price;
```