SQL Exercises:
Orders Database

Example Database

The questions refer to the example database created by the following SQL commands:

```
Create table Customers(customerID int primary key, name text);
Create table Products(productID int primary key, price numeric);
Create table Orders(orderID int primary key,
     customer int, product int, quantity int,
     orderTime timestamp);
Alter table Orders add foreign key (customer)
     references Customers(customerID);
Alter table Orders add foreign key (product)
     references Products(productID);
```

This database stores information on products, customers, and their orders. The Customers table stores unique customer IDs, together with the customer's name. The Products table stores unique product IDs, together with the associated price. The Orders table stores for each order the ID of the customer who issued it, the ID of the ordered product (to simplify, we assume that each order only refers to one type of product), the quantity (meaning how many products were ordered), and the order timestamp.

Question 1 (Easy)

Write an SQL query that retrieves the IDs of the customers who issued the last five orders. The query result contains one single column with the customer ID.

Question 1 - Solution

```
Select customer from Orders order by orderTime desc limit 5;
```

The query sorts orders by the order time, starting with the highest timestamps (i.e., the most recent orders). Then, it retrieves the first five orders using a limit clause and selects the customer column, containing the customer ID (note: it is clear that the customer column refers to the ID due to the corresponding foreign key constraint in the database).

Question 2 (Easy)

Write an SQL query that retrieves the list of all orders, reporting for each order the order ID and the total cost of the order. The query result contains two columns: the order ID and the associated costs.

Question 2 - Solution

```
Select orderID, quantity * price from Orders O join Products P
on (O.product = P.productID);
```

To calculate the cost of an order, we need to multiply the number of items ordered with the associated price per item. This means we need data from the Orders and Products tables. The query joins both of them using the order ID. Finally, we select the order ID and the total costs (calculated using a product).

Question 3 (Easy)

Write an SQL query retrieving the number of orders issued after 1/1/2025 for the product with ID 5. The query result contains one single column with the count.

Question 3 - Solution

```
Select count(*) from Orders where product = 5 and orderTime >
date '1/1/2025';
```

The SQL query filters orders using a conjunction (AND) with two conditions, the restriction of the product ID and the restriction on the order date. It counts the resulting orders via a count aggregate.

Question 4 (Medium)

Write an SQL query that retrieves all customers having issued at least five orders. The query result contains one column with the customer ID.

Question 4 - Solution

```
Select customer from Orders group by customer having count(*) >=
5;
```

The query first groups orders by the customer ID. Then, it filters groups using a having clause to the ones containing at least five entries. For the remaining groups, it includes the customer ID in the query result.

Question 5 (Medium)

Write an SQL query retrieving customers who have not issued any orders yet. The query result contains one column with the customer name.

Question 5 - Solution

```
Select name from Customers C where not exists (select * from
Orders O where O.customer = C.customerID);
```

The query selects the name from the Customers table. It filters customers via a sub-query, testing for the absence of any orders associated with the ID of the currently considered customer.

Question 6 (Medium)

Write an SQL query that reports for each customer the timestamp of the last order. The query result contains two columns: the name of the customer and the order timestamp. The query result does not contain customers who did not issue any orders yet.

Question 6 - Solution

```
Select name, max(orderTime) from Customers C join Orders O on
(C.customerID = o.customer) group by customer, name;
```

The query joins the Customers and Orders tables and groups the resulting rows by the customer ID and name (the name can be omitted for many database systems as it is implied by the customer ID). For the resulting groups, the query selects the associated customer name (which is equal for all rows within the same group) and the latest order timestamp (using the maximum operator).

Question 7 (Hard)

Write an SQL query retrieving for each customer the total cost of all orders issued by the customer. The query result contains two columns: the customer name and the associated costs. The query result only contains customers who have issued at least one order.

Question 7 - Solution

```
Select name, sum(quantity * price) from Customers C, Orders O,
Products P where C.customerID = O.customer and O.product =
P.productID group by customer, name;
```

The query needs access to all three tables (to obtain the customer name, unique to the Customers table, the quantity, unique to the Orders table, and the price per product, unique to the Products table). Hence, the query joins all three tables and groups result rows by the customer ID and name. Finally, it calculates for each customer the total order volume by summing up over the products of quantity and price over all orders associated with that customer.

Question 8 (Hard)

Write an SQL query counting for each customer the number of associated orders. The query result contains two columns: the customer name and the order count. Note that the query must also handle customers who have not yet issued any orders.

Question 8 - Solution

```
Select name, count(product) from Customers C left outer join
Orders O on (C.customerID = o.customer) group by customer, name;
```

To represent customers who have not yet issued any orders, we need to use a left outer join, using Customers as the left table. This ensures that each customer in the Customers table appears in the join result, even if no matching orders are found. If no matching orders are found, the product field in the result rows is set to SQL NULL. By using this field as an argument to the count aggregate, we only count result rows where this value is not set to NULL. The query finally groups by the customer (name and ID).

Question 9 (Hard)

Write an SQL query retrieving all customers who ordered each product in the database at least two times. The query result contains one column with the customer name.

Question 9 - Solution

```
Select name from Customers C where not exists (select * from
Products P where not exists (select * from Orders O where
P.productID = O.product and C.customerID = O.customer group by
O.productID, O.customerID having sum(quantity) >= 2))
```

The query uses a type of double negation. For each customer, it verifies whether no product exists such that the customer did not order at least two instances. Note that those two instances could either appear in the same order or across two orders. To handle both cases, the innermost sub-query filters orders to the ones referring to the relevant product and customer, groups them by the product and customer (which results in at most one group due to the filter conditions), and finally filters out that group if the total quantity sum is below two (using the Having clause). As a result, the innermost sub-query returns an empty result if the customer ordered the product less than two times. In that case, the sub-query referring to the Products table returns a non-empty result which means that the condition in the outer-most query is not satisfied (meaning that the associated customer will not be returned).

Question 10 (Hard)

Write an SQL query that retrieves pairs of customers who ordered at least three of the same products. The query result contains two columns with customer IDs. Note that each combination of customers should only appear once in the query result.

Question 10 - Solution

```
Select O1.customer, O2.customer from Orders O1, Orders O2 where
O1.product = O2.product and O1.customer < O2.customer group by
O1.customer, O2.customer having count(*) >= 3;
```

The query retrieves pairs of orders with the same product and different customer IDs (with the first customer ID being the smaller one). It partitions those rows by the ID of the two customers and only retains groups with at least three elements. From the resulting rows, it selects the IDs of the two involved customers.