

# SQL Subqueries ¶

## Introduction

SQL queries can get complex. For example, you might have been a little thrown off by the many to many join in the last lab. There, you had to join four tables. This is just the tip of the iceberg. Depending on how your database is set up, you might have to join subset views of multiple tables. When queries get complex like this, it is often useful to use the concept of subqueries to help break the problem into smaller, more digestible tasks.

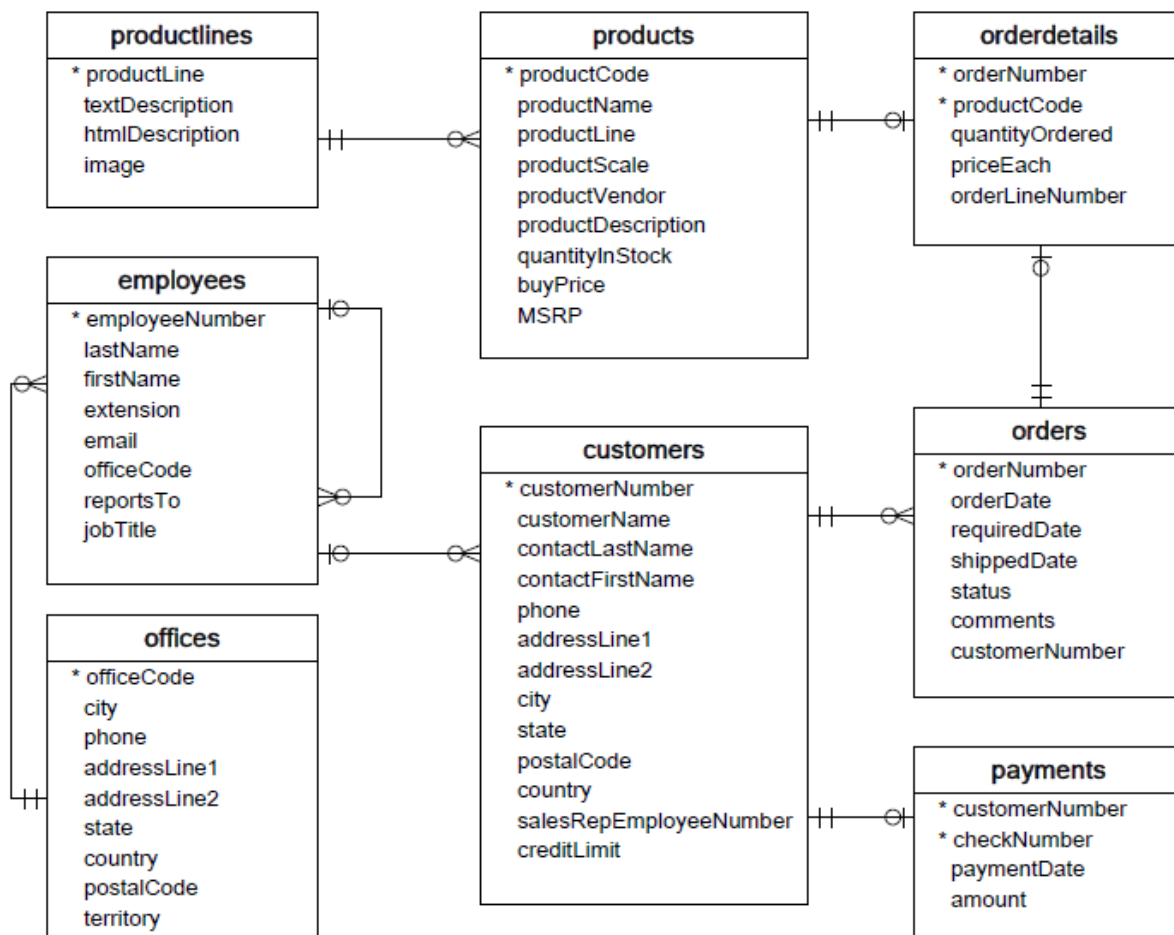
## Objectives

You will be able to:

- Write subqueries to decompose complex queries

## Our Customer Relationship Management ERD

As a handy reference, here's the schema for the CRM database you'll continue to practice with.



```
In [1]: import sqlite3
import pandas as pd
```

```
In [2]: conn = sqlite3.Connection('data.sqlite')
```

## Substituting JOIN with Subqueries

Let's start with a query of employees from the United States. Using your current knowledge, you could solve this using a join.

```
In [3]: q = """
SELECT lastName, firstName, officeCode
FROM employees
JOIN offices
    USING(officeCode)
WHERE country = "USA"
;"""
pd.read_sql(q, conn)
```

```
Out[3]:
```

	lastName	firstName	officeCode
0	Bow	Anthony	1
1	Firrelli	Jeff	1
2	Jennings	Leslie	1
3	Murphy	Diane	1
4	Patterson	Mary	1
5	Thompson	Leslie	1
6	Firrelli	Julie	2
7	Patterson	Steve	2
8	Tseng	Foon Yue	3
9	Vanauf	George	3

Another approach would be to use a subquery. Here's what it would look like:

```
In [4]: q = """
SELECT lastName, firstName, officeCode
FROM employees
WHERE officeCode IN (SELECT officeCode
                      FROM offices
                      WHERE country = "USA")
;
"""
pd.read_sql(q, conn)
```

```
Out[4]:
```

	lastName	firstName	officeCode
0	Murphy	Diane	1
1	Patterson	Mary	1
2	Firrelli	Jeff	1
3	Bow	Anthony	1
4	Jennings	Leslie	1
5	Thompson	Leslie	1
6	Firrelli	Julie	2
7	Patterson	Steve	2
8	Tseng	Foon Yue	3
9	Vanauf	George	3

There it is, a query within a query! This can be very helpful and also allow you to break down problems into constituent parts. Often queries can be formulated in multiple ways as with the above example. Other times, using a subquery might be essential. For example, what if you wanted to find all of the employees from offices with at least 5 employees?

## Subqueries for Filtering Based on an Aggregation

Think for a minute about how you might write such a query.

Now that you've had a minute to think it over, you might see some of the challenges with this query. On the one hand, we are looking to filter based on an aggregate condition: the number of employees per office. You know how to do this using the `GROUP BY` and `HAVING` clauses, but the data we wish to retrieve is not aggregate data. We only wish to **filter** based on the aggregate, not retrieve aggregate data. As such, this is a natural place to use a subquery.

```
In [5]: q = """
SELECT lastName, firstName, officeCode
FROM employees
WHERE officeCode IN (
    SELECT officeCode
    FROM offices
    JOIN employees
    USING(officeCode)
    GROUP BY 1
    HAVING COUNT(employeeNumber) >= 5
)
;
"""
pd.read_sql(q, conn)
```

```
Out[5]:
```

	lastName	firstName	officeCode
0	Murphy	Diane	1
1	Patterson	Mary	1
2	Firrelli	Jeff	1
3	Bondur	Gerard	4
4	Bow	Anthony	1
5	Jennings	Leslie	1
6	Thompson	Leslie	1
7	Bondur	Loui	4
8	Hernandez	Gerard	4
9	Castillo	Pamela	4
10	Gerard	Martin	4

You can chain queries like this in many fashions. For example, maybe you want to find the average of individual customers' average payments:

(It might be more interesting to investigate the standard deviation of customer's average payments, but standard deviation is not natively supported in SQLite as it is in other SQL versions like PostgreSQL.)

```
In [6]: q = """
SELECT AVG(customerAvgPayment) AS averagePayment
FROM (
    SELECT AVG(amount) AS customerAvgPayment
    FROM payments
    JOIN customers
        USING(customerNumber)
    GROUP BY customerNumber
)
;"""
pd.read_sql(q, conn)
```

```
Out[6]:
```

	<b>averagePayment</b>
<b>0</b>	31489.754582

## Summary

In this lesson, you were briefly introduced to the powerful concept of subqueries and how you can use them to write more complex queries. In the upcoming lab, you'll really start to strengthen your SQL and data wrangling skills by using all of the SQL techniques introduced thus far.