Join Statements ¶

Introduction

In this section, you will learn about several types of `JOIN` statements. Joins are the primary mechanism for combining data from multiple tables. In order to do this, you define the common attribute(s) between tables in order for them to be combined.

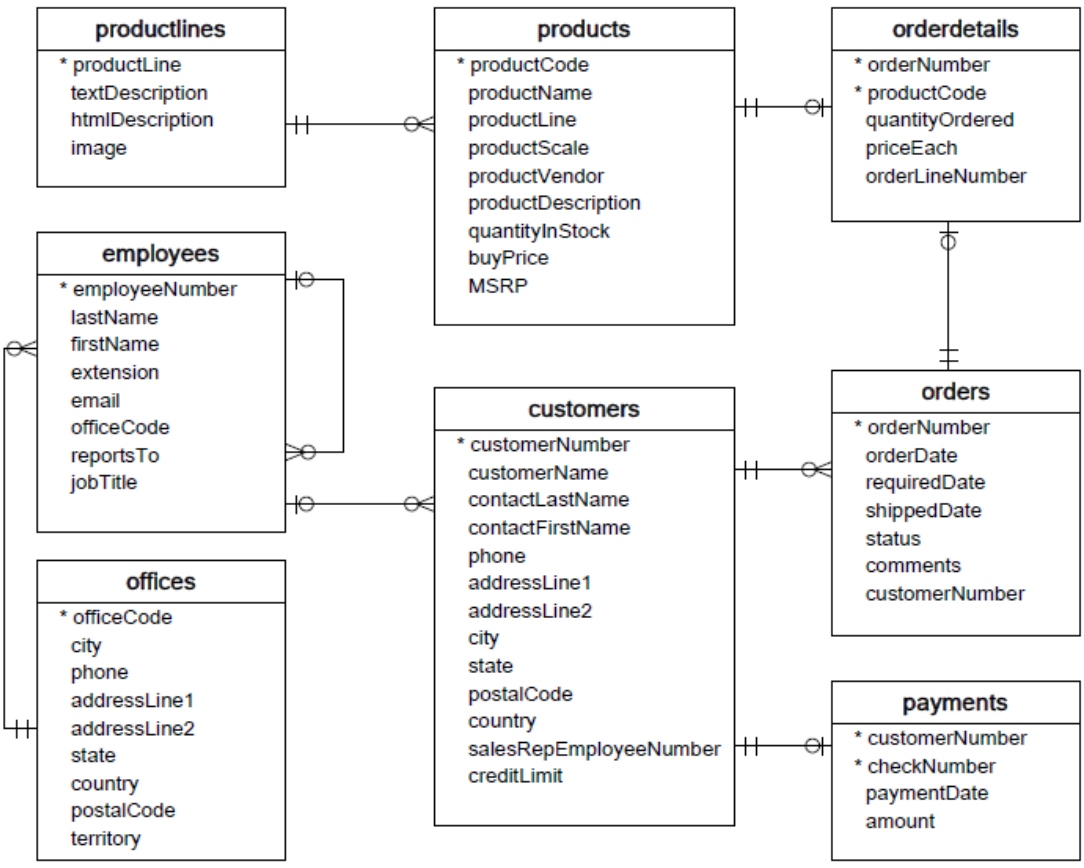
Objectives

You will be able to:

- Write SQL queries that make use of various types of joins
- Compare and contrast the various types of joins
- Discuss how primary and foreign keys are used in SQL
- Decide and perform whichever type of join is best for retrieving desired data

CRM ERD

In almost all industry cases, rather than just working with a single table you will generally need data from multiple tables. Doing this requires the use of **joins** using shared columns from the two tables. For example, here's a diagram of a mock customer relationship management (CRM) database.



Connecting to the Database

As usual, you'll start by connecting to the database.

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

In [7]:
```
<a href="http://www.flatironschool.com">Flatiron School</a>
```

```
  File "/tmp/ipykernel_66/1797433420.py", line 1
    <a href="http://www.flatironschool.com">Flatiron School</a>
    ^
SyntaxError: invalid syntax
```

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

Displaying Product Details Along with Order Details

Let's say you need to generate some report that includes details about products from orders. To do that, we would need to take data from multiple tables in a single statement.

In [3]:
```
q = """
SELECT *
FROM orderdetails
JOIN products
    ON orderdetails.productCode = products.productCode
LIMIT 10
;
"""
pd.read_sql(q, conn)
```

Out[3]:

	orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber	productCode	product
0	10100	S18_1749	30	136.00	3	S18_1749	1917 Touring
1	10100	S18_2248	50	55.09	2	S18_2248	191 Tou
2	10100	S18_4409	22	75.46	4	S18_4409	19 F 8 Spide
3	10100	S24_3969	49	35.29	1	S24_3969	Me Ben Rc
4	10101	S18_2325	25	108.06	4	S18_2325	1932 M Ford J-
5	10101	S18_2795	26	167.06	1	S18_2795	Mer Ben
6	10101	S24_1937	45	32.53	3	S24_1937	Ch Deluxe
7	10101	S24_2022	46	44.35	2	S24_2022	1938 C Presi Lim
8	10102	S18_1342	39	95.55	2	S18_1342	1937 L I
9	10102	S18_1367	41	43.13	1	S18_1367	Mer Benz S Rc

Compared to the Individual Tables:

`orderdetails` Table:

In [4]:
```
pd.read_sql("""SELECT * FROM orderdetails LIMIT 10;""", conn)
```

Out[4]:

	orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber
0	10100	S18_1749	30	136.00	3
1	10100	S18_2248	50	55.09	2
2	10100	S18_4409	22	75.46	4
3	10100	S24_3969	49	35.29	1
4	10101	S18_2325	25	108.06	4
5	10101	S18_2795	26	167.06	1
6	10101	S24_1937	45	32.53	3
7	10101	S24_2022	46	44.35	2
8	10102	S18_1342	39	95.55	2
9	10102	S18_1367	41	43.13	1

`products` Table:

In [5]:
```
pd.read_sql("""SELECT * FROM products LIMIT 10;""", conn)
```

Out[5]:

	productCode	productName	productLine	productScale	productVendor	productDescription	quar
0	S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10	Min Lin Diecast	This replica features working kickstand, front...	
1	S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10	Classic Metal Creations	Turnable front wheels; steering function; deta...	
2	S10_2016	1996 Moto Guzzi 1100i	Motorcycles	1:10	Highway 66 Mini Classics	Official Moto Guzzi logos and insignias, saddl...	
3	S10_4698	2003 Harley-Davidson Eagle Drag Bike	Motorcycles	1:10	Red Start Diecast	Model features, official Harley Davidson logos...	
4	S10_4757	1972 Alfa Romeo GTA	Classic Cars	1:10	Motor City Art Classics	Features include: Turnable front wheels; steer...	
5	S10_4962	1962 LanciaA Delta 16V	Classic Cars	1:10	Second Gear Diecast	Features include: Turnable front wheels; steer...	
6	S12_1099	1968 Ford Mustang	Classic Cars	1:12	Autoart Studio Design	Hood, doors and trunk all open to reveal highl...	
7	S12_1108	2001 Ferrari Enzo	Classic Cars	1:12	Second Gear Diecast	Turnable front wheels; steering function; deta...	
8	S12_1666	1958 Setra Bus	Trucks and Buses	1:12	Welly Diecast Productions	Model features 30 windows, skylights & glare r...	
9	S12_2823	2002 Suzuki XREO	Motorcycles	1:12	Unimax Art Galleries	Official logos and insignias, saddle bags loca...	

The `USING` clause

A more concise way to join the tables, if the column name is identical, is the `USING` clause. Rather then saying on `tableA.column = tableB.column` we can simply say `USING(column)`. Again, this only works if the column is **identically named** for both tables.

In [6]:
```
q = """
SELECT *
FROM orderdetails
JOIN products
    USING(productCode)
LIMIT 10
;
"""
pd.read_sql(q, conn)
```

Out[6]:

	orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber	productName	produc
0	10100	S18_1749	30	136.00	3	1917 Grand Touring Sedan	Vintage
1	10100	S18_2248	50	55.09	2	1911 Ford Town Car	Vintage
2	10100	S18_4409	22	75.46	4	1932 Alfa Romeo 8C2300 Spider Sport	Vintage
3	10100	S24_3969	49	35.29	1	1936 Mercedes Benz 500k Roadster	Vintage
4	10101	S18_2325	25	108.06	4	1932 Model A Ford J-Coupe	Vintage
5	10101	S18_2795	26	167.06	1	1928 Mercedes-Benz SSK	Vintage
6	10101	S24_1937	45	32.53	3	1939 Chevrolet Deluxe Coupe	Vintage
7	10101	S24_2022	46	44.35	2	1938 Cadillac V-16 Presidential Limousine	Vintage
8	10102	S18_1342	39	95.55	2	1937 Lincoln Berline	Vintage
9	10102	S18_1367	41	43.13	1	1936 Mercedes-Benz 500K Special Roadster	Vintage

More Aliasing

You can also assign tables an **alias** by entering an alternative shorthand name. This is slightly different than the previous lesson where we introduced aliases for column names, since now we are aliasing *tables*.

When aliasing columns the goal is usually to improve readability by giving something a more specific or easier-to-read name. For example, `name AS employee_name`, `AVG(AVG) AS average_batting_average`, or `COUNT(*) AS num_products`.

When aliasing tables the goal is usually to shorten the name, in order to shorten the overall query. So typically you'll see examples that alias a longer table name to a one-character or two-character shorthand. For example, `orderdetails AS od` or `products AS p`. (It is also possible to use aliases to clarify what exactly is in a table, like how aliases are used for columns, just less common.)

The following query produces the same result as the previous ones, using aliases `od` and `p` for `orderdetails` and `products`, respectively:

In [7]:
```
q = """
SELECT *
FROM orderdetails AS od
JOIN products AS p
    ON od.productCode = p.productCode
LIMIT 10;
"""
pd.read_sql(q, conn)
```

Out[7]:

	orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber	productCode	product
0	10100	S18_1749	30	136.00	3	S18_1749	1917 Touring
1	10100	S18_2248	50	55.09	2	S18_2248	191 Tov
2	10100	S18_4409	22	75.46	4	S18_4409	19 F 8 Spide
3	10100	S24_3969	49	35.29	1	S24_3969	Me Ben Rc
4	10101	S18_2325	25	108.06	4	S18_2325	1932 M Ford J-
5	10101	S18_2795	26	167.06	1	S18_2795	Mer Ben
6	10101	S24_1937	45	32.53	3	S24_1937	Ch Deluxe
7	10101	S24_2022	46	44.35	2	S24_2022	1938 C Presi Lim
8	10102	S18_1342	39	95.55	2	S18_1342	1937 I I
9	10102	S18_1367	41	43.13	1	S18_1367	Mer Benz S Rc

Note that just like with column aliases, the `AS` keyword is optional in SQLite. So, instead of `FROM orderdetails AS od` you could write `FROM orderdetails od` with the same outcome.

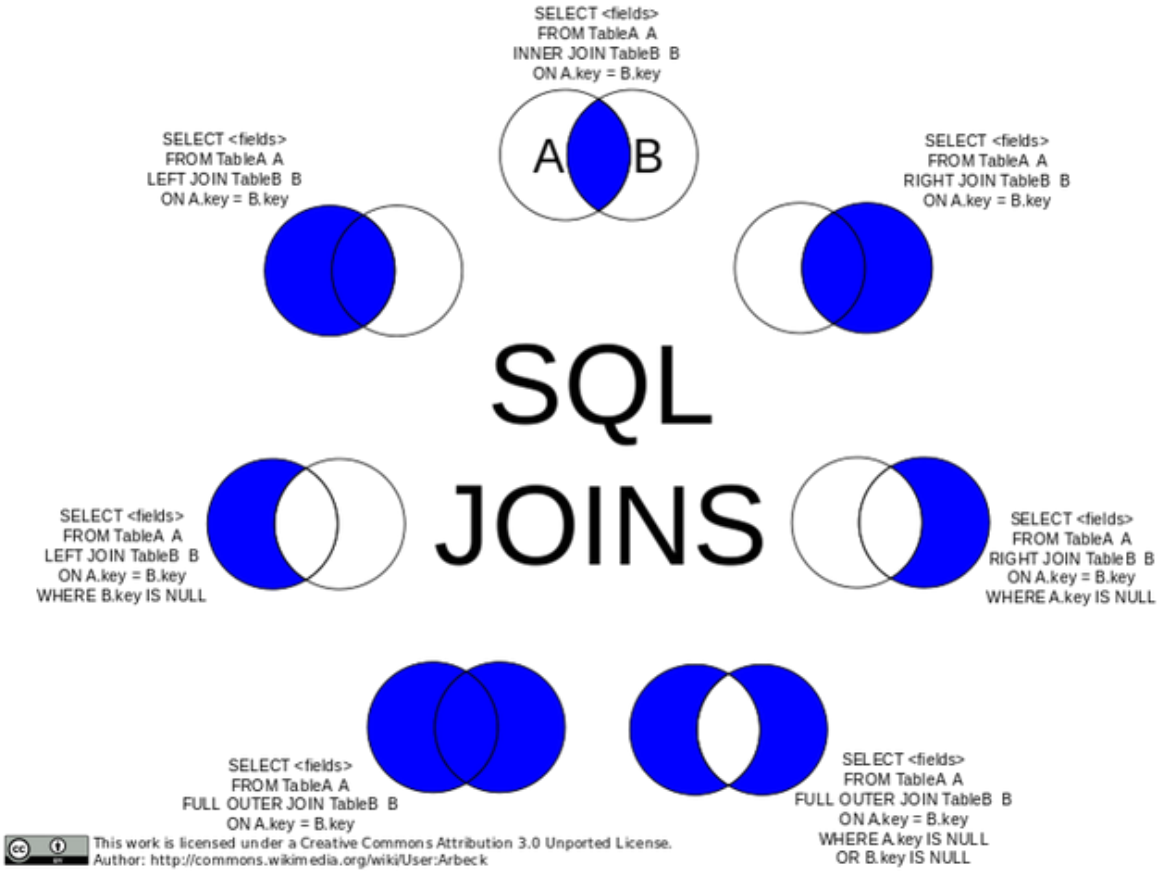
It is somewhat more common to see `AS` used with column aliases and skipped with table aliases, but again, you'll want to check the syntax rules of your particular type of SQL as well as style guidelines from your employer to know which syntax to use in a professional setting.

`LEFT JOIN`s

By default a `JOIN` is an `INNER JOIN`, or the intersection between two tables. In other words, the `JOIN` between orders and products is only for `productCodes` that are in both the `orderdetails` and `products` tables. If a product had yet to be ordered (and wasn't in the `orderdetails` table) then it would also not be in the result of the `JOIN`.

The `LEFT JOIN` keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side if there is no match.

There are many other types of joins, displayed below. Of these, SQLite does not support outer joins, but it is good to be aware of as more powerful versions of SQL such as PostgreSQL support these additional functions.



For example, the statement

```
SELECT * FROM products LEFT JOIN orderdetails
```

would return all products, even those that hadn't been ordered. You can imagine that all products in inventory should have a description in the product table, but perhaps not every product is represented in the orderdetails table.

In [8]:
```
q = """
SELECT *
FROM products
LEFT JOIN orderdetails
    USING(productCode)
;
"""
df = pd.read_sql(q, conn)

print("Number of records returned:", len(df))
print("Number of records where order details are null:", len(df[df.orderNumber.is
```

```
Number of records returned: 2997
Number of records where order details are null: 1
```

In [9]:
```
df[df.orderNumber.isnull()]
```

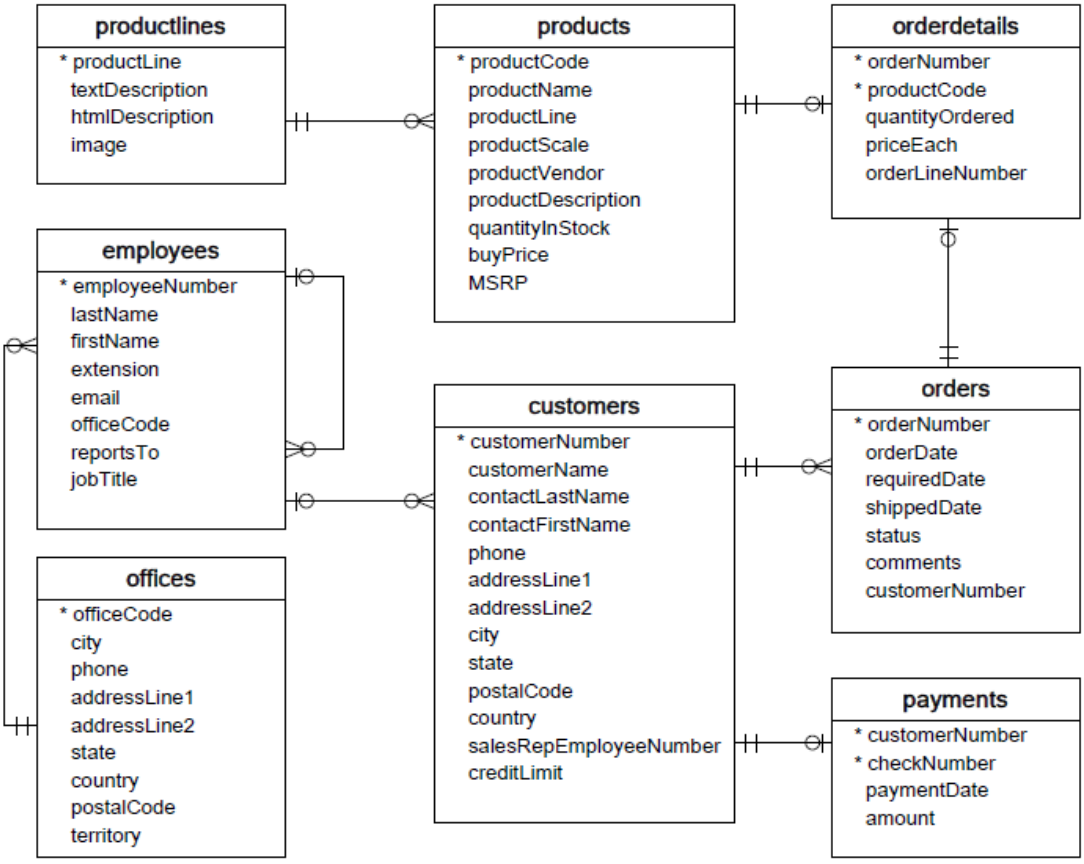
Out[9]:

	productCode	productName	productLine	productScale	productVendor	productDescription	c
1122	S18_3233	1985 Toyota Supra	Classic Cars	1:18	Highway 66 Mini Classics	This model features soft rubber tires, working...	

As you can see, it's a rare occurrence, but there is one product that has yet to be ordered.

Primary Versus Foreign Keys

Another important consideration when performing joins is to think more about the key or column you are joining on. As you'll see in upcoming lessons, this can lead to interesting behavior if the join value is not unique in one or both of the tables. In all of the above examples, you joined two tables using the **primary key**. The primary key(s) of a table are those column(s) which uniquely identify a row. You'll also see this designated in our schema diagram with the asterisk (*).



You can also join tables using **foreign keys** which are not the primary key for that particular table, but rather another table. For example, `employeeNumber` is the primary key for the employees table and corresponds to the `salesRepEmployeeNumber` of the customers table. In the customers table, `salesRepEmployeeNumber` is only a foreign key, and is unlikely to be a unique identifier, as it is likely that an employee serves multiple customers. As such, in the resulting view `employeeNumber` would no longer be a unique field.

In [10]:
```
q = """
SELECT *
FROM customers AS c
JOIN employees AS e
    ON c.salesRepEmployeeNumber = e.employeeNumber
ORDER By employeeNumber
;
"""
pd.read_sql(q, conn)
```

Out[10]:

	customerNumber	customerName	contactLastName	contactFirstName	phone	addressLin
0	124	Mini Gifts Distributors Ltd.	Nelson	Susan	4155551450	5677 Strong
1	129	Mini Wheels Co.	Murphy	Julie	6505555787	5557 No Pendale Stre
2	161	Technics Stores Inc.	Hashimoto	Juri	6505556809	9408 Fu Cir
3	321	Corporate Gift Ideas Co.	Brown	Julie	6505551386	7734 Strong
4	450	The Sharp Gifts Warehouse	Frick	Sue	4085553659	3086 Ingle l
...	...	...	...	...	...	
95	298	Vida Sport, Ltd	Holz	Mihael	0897-034555	Grenzacherw 2
96	344	CAF Imports	Fernandez	Jesus	+34 913 728 555	Merchar Hou
97	376	Precious Collectables	Urs	Braun	0452-076555	Hauptstr.
98	458	Corrida Auto Replicas, Ltd	Sommer	Martín	(91) 555 22 82	C/ Araquil,
99	484	Iberia Gift Imports, Corp.	Roel	José Pedro	(95) 555 82 82	C/ Romero,

100 rows × 21 columns

Notice that this also returned both columns: `salesRepEmployeeNumber` and `employeeNumber`. These columns contain identical values so you would probably actually only want to select one or the other.

Summary

In this lesson, you investigated joins. This included implementing the `ON` and `USING` clauses, aliasing table names, implementing `LEFT JOIN`, and using primary vs. foreign keys.