Querying with SQLAlchemy - Lab

Introduction

In this lesson, we'll learn how to use SQLAlchemy to write queries about Microsoft's *Northwind Traders* database!

Objectives

You will be able to:

- Read and understand an ERD diagram
- Create queries with SQLAlchemy, including queries that involve many-to-many relationships

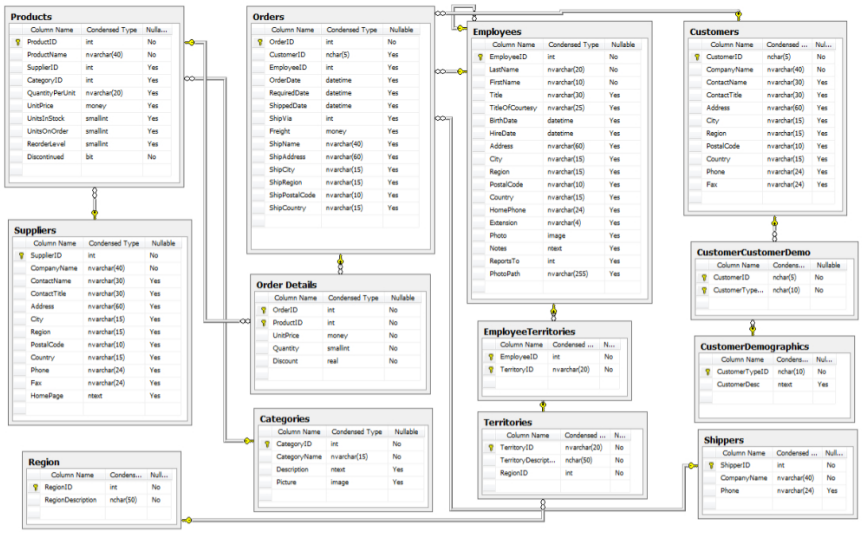
Getting Started

In order to complete this lab, we'll need to download a SQLite3-compatible version of the *Northwind Traders* database from Microsoft. Microsoft built this database back in the year 2000 to help showcase their SQL Server technology. Since then, it has been open-sourced and has become a great practice tool for every new generation of SQL learners.

Lucky for us, some generous programmers have already converted the Northwind database to a sqlite-compatible version and posted it on Github. We've already included the file the SQL database file that we'll be working with in the repo for this folder, along with the following ERD Diagram. However, if you would like to work with the larger version of this dataset at a future time, just clone this repo and follow their instructions to access it!

ERD Diagram For Northwind Traders

The following ERD Diagram describes the Northwind Traders Database:



If the text seems a bit hard to read inside this jupyter notebook, just go into the folder for this repo and open the `Northwind_ERD.png` file manually to see it full size.

Connecting to the Database

The first thing we'll need to do is connect to the Northwind Traders database, which can be found in the file `Northwind_small.sqlite`.

In the cell below, import the necessary tools from the `sqlalchemy` library, and take the necessary steps to connect to the database.

NOTE: We won't be modifying the information in the database at all, just querying it, so there's no need to import the various things you would need to create declarative base classes and the like.

In [21]:
```
import sqlalchemy
from sqlalchemy import create_engine
from sqlalchemy.orm import Session, sessionmaker

engine = create_engine("sqlite:///Northwind_small.sqlite", echo=True)
Session = sessionmaker(bind=engine)
session = Session()
```

Get Table Names and Table Information

One of the most useful things we can do when working with a new database is to inspect the tables until we have a solid idea of what we're looking at. As you work through this lab, you'll notice that there are some small discrepancies between the Table/Column names listed in the ERD and what they are actually are in the database. This may be annoying, but this is not an accident--sometimes, documentation is wrong! By learning how to inspect what tables exist in a database, as well as how which columns exist inside a table, we can save ourselves a lot of headaches by double checking.

In the cell below:

- Import `inspect` from sqlalchemy
- Create an inspector object by passing in `engine` to `inspect`
- Use the appropriate inspector function to get the names of all tables

In [58]:
```
from sqlalchemy import inspect

inspector = inspect(engine)

print(inspector.get_table_names())
```

```
2018-10-28 20:59:25,655 INFO sqlalchemy.engine.base.Engine SELECT name FROM sqlite_master WHERE type='table' ORDER BY name
INFO:sqlalchemy.engine.base.Engine:SELECT name FROM sqlite_master WHERE type='table' ORDER BY name
2018-10-28 20:59:25,658 INFO sqlalchemy.engine.base.Engine ()
INFO:sqlalchemy.engine.base.Engine:()
['Category', 'Customer', 'CustomerCustomerDemo', 'CustomerDemographic', 'Employee', 'EmployeeTerritory', 'Order', 'OrderDetail', 'Product', 'Region', 'Shipper', 'Supplier', 'Territory']
```

Great! We can now see exactly what each Table is named.

Question:

Are there any discrepancies between the table names and the ERD diagrams? if so, what are they?

Write your answer below this line:

Yes. Table names listed in the ERD are all plural (e.g. 'Employees'), while in reality, the table names are actually singular (e.g. 'Employee').

Let's inspect the column names on one of the tables as well. In the cell below, call the appropriate method to get all column names for the `'Employee'` table.

In [60]:
```
print(inspector.get_columns('Employee'))
```

```
[{'name': 'Id', 'type': INTEGER(), 'nullable': True, 'default': None, 'autoincrement': 'auto', 'primary_key': 1}, {'name':
'LastName', 'type': VARCHAR(length=8000), 'nullable': True, 'default': None, 'autoincrement': 'auto', 'primary_key': 0}, {'n
ame': 'FirstName', 'type': VARCHAR(length=8000), 'nullable': True, 'default': None, 'autoincrement': 'auto', 'primary_key':
0}, {'name': 'Title', 'type': VARCHAR(length=8000), 'nullable': True, 'default': None, 'autoincrement': 'auto', 'primary_ke
y': 0}, {'name': 'TitleOfCourtesy', 'type': VARCHAR(length=8000), 'nullable': True, 'default': None, 'autoincrement': 'aut
o', 'primary_key': 0}, {'name': 'BirthDate', 'type': VARCHAR(length=8000), 'nullable': True, 'default': None, 'autoincremen
t': 'auto', 'primary_key': 0}, {'name': 'HireDate', 'type': VARCHAR(length=8000), 'nullable': True, 'default': None, 'autoin
crement': 'auto', 'primary_key': 0}, {'name': 'Address', 'type': VARCHAR(length=8000), 'nullable': True, 'default': None, 'a
utoincrement': 'auto', 'primary_key': 0}, {'name': 'City', 'type': VARCHAR(length=8000), 'nullable': True, 'default': None,
'autoincrement': 'auto', 'primary_key': 0}, {'name': 'Region', 'type': VARCHAR(length=8000), 'nullable': True, 'default': No
ne, 'autoincrement': 'auto', 'primary_key': 0}, {'name': 'PostalCode', 'type': VARCHAR(length=8000), 'nullable': True, 'defa
ult': None, 'autoincrement': 'auto', 'primary_key': 0}, {'name': 'Country', 'type': VARCHAR(length=8000), 'nullable': True,
'default': None, 'autoincrement': 'auto', 'primary_key': 0}, {'name': 'HomePhone', 'type': VARCHAR(length=8000), 'nullable':
True, 'default': None, 'autoincrement': 'auto', 'primary_key': 0}, {'name': 'Extension', 'type': VARCHAR(length=8000), 'null
able': True, 'default': None, 'autoincrement': 'auto', 'primary_key': 0}, {'name': 'Photo', 'type': BLOB(), 'nullable': Tru
e, 'default': None, 'autoincrement': 'auto', 'primary_key': 0}, {'name': 'Notes', 'type': VARCHAR(length=8000), 'nullable':
True, 'default': None, 'autoincrement': 'auto', 'primary_key': 0}, {'name': 'ReportsTo', 'type': INTEGER(), 'nullable': Tru
e, 'default': None, 'autoincrement': 'auto', 'primary_key': 0}, {'name': 'PhotoPath', 'type': VARCHAR(length=8000), 'nullabl
e': True, 'default': None, 'autoincrement': 'auto', 'primary_key': 0}]
```

That output is good, but its a bit messy. Let's write a function that makes it a bit more readable, and only tells us what we need to know.

The current structure of the output is a list of dictionaries. Complete the function below so that when the function is called and passed a table name, it prints out the name and type of each column in a well-formatted way.

In [62]:
```
def get_columns_info(col_name):
    cols_list = inspector.get_columns(col_name)

    print("Table Name: {}".format(col_name))
    print("")

    for column in cols_list:
        print("Name: {} \t Type: {}".format(column['name'], column['type']))

get_columns_info('Employee')
```

```
Table Name: Employee

Name: Id         Type: INTEGER
Name: LastName   Type: VARCHAR(8000)
Name: FirstName          Type: VARCHAR(8000)
Name: Title      Type: VARCHAR(8000)
Name: TitleOfCourtesy    Type: VARCHAR(8000)
Name: BirthDate          Type: VARCHAR(8000)
Name: HireDate   Type: VARCHAR(8000)
Name: Address    Type: VARCHAR(8000)
Name: City       Type: VARCHAR(8000)
Name: Region     Type: VARCHAR(8000)
Name: PostalCode         Type: VARCHAR(8000)
Name: Country    Type: VARCHAR(8000)
Name: HomePhone          Type: VARCHAR(8000)
Name: Extension          Type: VARCHAR(8000)
Name: Photo      Type: BLOB
Name: Notes      Type: VARCHAR(8000)
Name: ReportsTo          Type: INTEGER
Name: PhotoPath          Type: VARCHAR(8000)
```

```
# Expected Output:

Table Name: Employee

Name: Id     Type: INTEGER
Name: LastName   Type: VARCHAR(8000)
Name: FirstName      Type: VARCHAR(8000)
Name: Title      Type: VARCHAR(8000)
Name: TitleOfCourtesy    Type: VARCHAR(8000)
Name: BirthDate      Type: VARCHAR(8000)
Name: HireDate   Type: VARCHAR(8000)
Name: Address    Type: VARCHAR(8000)
Name: City   Type: VARCHAR(8000)
Name: Region     Type: VARCHAR(8000)
Name: PostalCode     Type: VARCHAR(8000)
Name: Country    Type: VARCHAR(8000)
Name: HomePhone      Type: VARCHAR(8000)
Name: Extension      Type: VARCHAR(8000)
Name: Photo      Type: BLOB
Name: Notes      Type: VARCHAR(8000)
Name: ReportsTo      Type: INTEGER
Name: PhotoPath      Type: VARCHAR(8000)
```

Connecting and Executing Raw SQL Statements

Sometimes, the easiest thing for us to do is to just execute a raw SQL statement. This is very easy with SQLAlchemy--we just need to establish a connection, and then use the appropriat methods to execute SQL statements!

In the cell below:

- Create a connection using the `engine` object's appropriate method and store it in the variable `con`
- Use the appropriate method from `con` to execute a raw SQL statement (in the form of a string) that gets everything from the `'Customer'` table with a LIMIT of 5. Store the results returned in the variable `rs`
- Use the `fetchall()` method to display all results from `rs`.

In [67]:
```
con = engine.connect()
rs = con.execute("SELECT * FROM Customer LIMIT 5")

print(rs.fetchall())
```

```
2018-10-28 21:14:50,541 INFO sqlalchemy.engine.base.Engine SELECT * FROM Customer LIMIT 5

INFO:sqlalchemy.engine.base.Engine:SELECT * FROM Customer LIMIT 5

2018-10-28 21:14:50,543 INFO sqlalchemy.engine.base.Engine ()

INFO:sqlalchemy.engine.base.Engine:()

[('ALFKI', 'Alfreds Futterkiste', 'Maria Anders', 'Sales Representative', 'Obere Str. 57', 'Berlin', 'Western Europe', '1220
9', 'Germany', '030-0074321', '030-0076545'), ('ANATR', 'Ana Trujillo Emparedados y helados', 'Ana Trujillo', 'Owner', 'Avd
a. de la Constitución 2222', 'México D.F.', 'Central America', '05021', 'Mexico', '(5) 555-4729', '(5) 555-3745'), ('ANTON',
'Antonio Moreno Taquería', 'Antonio Moreno', 'Owner', 'Mataderos  2312', 'México D.F.', 'Central America', '05023', 'Mexic
o', '(5) 555-3932', None), ('AROUT', 'Around the Horn', 'Thomas Hardy', 'Sales Representative', '120 Hanover Sq.', 'London',
'British Isles', 'WA1 1DP', 'UK', '(171) 555-7788', '(171) 555-6750'), ('BERGS', 'Berglunds snabbköp', 'Christina Berglund',
'Order Administrator', 'Berguvsvägen  8', 'Luleå', 'Northern Europe', 'S-958 22', 'Sweden', '0921-12 34 65', '0921-12 34 6
7')]
```

```
# Expected Output:

[('ALFKI', 'Alfreds Futterkiste', 'Maria Anders', 'Sales Representative', 'Obere Str. 57', 'Berlin', 'Western Europe', '12209',
'Germany', '030-0074321', '030-0076545'), ('ANATR', 'Ana Trujillo Emparedados y helados', 'Ana Trujillo', 'Owner', 'Avda. de la
Constitución 2222', 'México D.F.', 'Central America', '05021', 'Mexico', '(5) 555-4729', '(5) 555-3745'), ('ANTON', 'Antonio
Moreno Taquería', 'Antonio Moreno', 'Owner', 'Mataderos  2312', 'México D.F.', 'Central America', '05023', 'Mexico', '(5) 555-
3932', None), ('AROUT', 'Around the Horn', 'Thomas Hardy', 'Sales Representative', '120 Hanover Sq.', 'London', 'British
```

```
Isles', 'WA1 1DP', 'UK', '(171) 555-7788', '(171) 555-6750'), ('BERGS', 'Berglunds snabbköp', 'Christina Berglund', 'Order
Administrator', 'Berguvsvägen  8', 'Luleå', 'Northern Europe', 'S-958 22', 'Sweden', '0921-12 34 65', '0921-12 34 67')]
```

Incorporating Pandas DataFrames

So far we've been able to easily connect to a SQL database, inspect the tables, and execute queries. However, the results returned from queries haven't been in an easily readable format. We'll fix that by taking the results and storing it in a pandas DataFrame!

In the cell below:

- Import pandas and set the standard alias.
- Create and execute a query that gets The firstname, lastname, and title of every person in the `'Employee'` table.
- Create a pandas DataFrame out of the results returned from `rs.fetchall()`
- Display the head of the new DataFRame

In [69]:
```
import pandas as pd
rs = con.execute("SELECT firstname, lastname, title from Employee")
df = pd.DataFrame(rs.fetchall())
df.head()
```

```
2018-10-28 21:19:27,831 INFO sqlalchemy.engine.base.Engine SELECT firstname, lastname, title from Employee
INFO:sqlalchemy.engine.base.Engine:SELECT firstname, lastname, title from Employee
2018-10-28 21:19:27,833 INFO sqlalchemy.engine.base.Engine ()
INFO:sqlalchemy.engine.base.Engine:()
```

Out[69]:

	0	1	2
0	Nancy	Davolio	Sales Representative
1	Andrew	Fuller	Vice President, Sales
2	Janet	Leverling	Sales Representative
3	Margaret	Peacock	Sales Representative
4	Steven	Buchanan	Sales Manager

Nice! We can now read our results. However, the columns of our DataFrame aren't labeled. Luckily, pandas plays nicely with the sqlalchemy library, and can actually execute sql queries!

Writing Queries with Pandas

In the cell below:

- Use the appropriate method from pandas to select all columns for every row in the `[Order]` table where the customer is `'VINET'`
- Be sure to pass in our `engine` as the 2nd parameter, otherwise it won't work!
- Display the head of the DataFrame created to ensure everything worked correctly.

In [70]:
```
df = pd.read_sql_query("SELECT * FROM [Order] WHERE CUSTOMERId = 'VINET'", engine)
df.head()
```

```
2018-10-28 21:22:58,764 INFO sqlalchemy.engine.base.Engine SELECT * FROM [Order] WHERE CUSTOMERId = 'VINET'
INFO:sqlalchemy.engine.base.Engine:SELECT * FROM [Order] WHERE CUSTOMERId = 'VINET'
2018-10-28 21:22:58,766 INFO sqlalchemy.engine.base.Engine ()
INFO:sqlalchemy.engine.base.Engine:()
```

Out[70]:

	Id	CustomerId	EmployeeId	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	ShipPostalCo
0	10248	VINET	5	2012-07-04	2012-08-01	2012-07-16	3	32.38	Vins et alcools Chevalier	59 rue de l'Abbaye	Reims	Western Europe	511
1	10274	VINET	6	2012-08-06	2012-09-03	2012-08-16	1	6.01	Vins et alcools Chevalier	59 rue de l'Abbaye	Reims	Western Europe	511
2	10295	VINET	2	2012-09-02	2012-09-30	2012-09-10	2	1.15	Vins et alcools Chevalier	59 rue de l'Abbaye	Reims	Western Europe	511
3	10737	VINET	2	2013-11-11	2013-12-09	2013-11-18	2	7.79	Vins et alcools Chevalier	59 rue de l'Abbaye	Reims	Western Europe	511
4	10739	VINET	3	2013-11-12	2013-12-10	2013-11-17	3	11.08	Vins et alcools Chevalier	59 rue de l'Abbaye	Reims	Western Europe	511

Great! As we can see from the output above, when we let pandas execute the SQL query for us, the DataFrame now contains columns with the correct labels. This is a great way to execute SQL while still making sure our results are easy to read and manipulate by using DataFrames!

Executing JOIN Statements

Let's try executing a JOIN statement inside `pd.read_sql_query`.

In the cell below:

- Write a query that gets the Order ID, Company Name, and the total count of orders made by each company (as num_orders).
- Group the results by Company Name
- Order the results by num_orders, descending
- Display the head of the DataFrame to ensure everything worked correctly.

In [72]:
```
df = pd.read_sql_query("""SELECT o.ID, c.CompanyName, Count(*) num_orders FROM [Order] \
o INNER JOIN Customer c on o.CustomerID = c.ID GROUP BY c.CompanyName ORDER BY num_orders DESC""", engine)
df.head()
```

```
2018-10-28 21:26:44,542 INFO sqlalchemy.engine.base.Engine SELECT o.ID, c.CompanyName, Count(*) num_orders FROM [Order] o IN
NER JOIN Customer c on o.CustomerID = c.ID GROUP BY c.CompanyName ORDER BY num_orders DESC
INFO:sqlalchemy.engine.base.Engine:SELECT o.ID, c.CompanyName, Count(*) num_orders FROM [Order] o INNER JOIN Customer c on
o.CustomerID = c.ID GROUP BY c.CompanyName ORDER BY num_orders DESC
2018-10-28 21:26:44,544 INFO sqlalchemy.engine.base.Engine ()
INFO:sqlalchemy.engine.base.Engine:()
```

Out[72]:

	Id	CompanyName	num_orders
0	11064	Save-a-lot Markets	31
1	11072	Ernst Handel	30
2	11021	QUICK-Stop	28
3	11050	Folk och fä HB	19
4	11063	Hungry Owl All-Night Grocers	19

Expected Output

	Id	CompanyName	num_orders
0	11064	Save-a-lot Markets	31
1	11072	Ernst Handel	30
2	11021	QUICK-Stop	28
3	11050	Folk och fä HB	19
4	11063	Hungry Owl All-Night Grocers	19

Great job! Let's see if we can execute a join that includes entities with a many-to-many relationship.

JOINs with Many-To-Many Relationships

In the cell below:

- Write a query that selects the LastName, FirstName and number of territories assigned for every employee.
- Group the results by employee lastname
- Order by the total number of territories assigned to each employee, descending
- You'll need to make use a join table to solve this one--be sure to take a look at the ERD diagram again if needed!
- Store your results in a DataFrame and display the head to ensure that everything worked correctly.

NOTE: For long SQL statements, consider using the multiline string format in python, denoted by `"""three quotes"""` at the beginning and end. Note that if you hit enter to move to another line, be sure to add an `\` character at the end of the line to escape it--otherwise, your sql statements will contain `\n` newline characters wherever you hit enter to move to the next line.

In [73]:
```
q = """SELECT LastName, FirstName, COUNT(*) as TerritoriesAssigned from \
Employee \
JOIN EmployeeTerritory et on Employee.Id = et.employeeId \
GROUP BY Employee.lastname \
ORDER BY TerritoriesAssigned DESC"""

df2 = pd.read_sql_query(q, engine)
df2.head()
```

```
2018-10-28 21:31:57,925 INFO sqlalchemy.engine.base.Engine SELECT LastName, FirstName, COUNT(*) as TerritoriesAssigned from Employee JOIN EmployeeTerritory et on Employee.Id = et.employeeId GROUP BY Employee.lastname ORDER BY TerritoriesAssigned DESC

INFO:sqlalchemy.engine.base.Engine:SELECT LastName, FirstName, COUNT(*) as TerritoriesAssigned from Employee JOIN EmployeeTerritory et on Employee.Id = et.employeeId GROUP BY Employee.lastname ORDER BY TerritoriesAssigned DESC

2018-10-28 21:31:57,927 INFO sqlalchemy.engine.base.Engine ()

INFO:sqlalchemy.engine.base.Engine:()
```

Out[73]:

	LastName	FirstName	TerritoriesAssigned
0	King	Robert	10
1	Buchanan	Steven	7
2	Dodsworth	Anne	7
3	Fuller	Andrew	7
4	Suyama	Michael	5

Expected Output:

	LastName	FirstName	TerritoriesAssigned
0	King	Robert	10
1	Buchanan	Steven	7
2	Dodsworth	Anne	7
3	Fuller	Andrew	7
4	Suyama	Michael	5

Great job! You've demonstrated proficiency using raw sql with SQLAlchemy. However, we haven't yet touched all the fun declarative stuff. Let's get some practice working with SQLAlchemy `session` objects below!

Using SQLAlchemy Sessions

So far, we've just been using SQLAlchemy as a way to connect to a database and run SQL queries. However, SQLAlchemy is an ***Object-Relational Mapper***, and can map entities in our database to python objects! This can be incredibly helpful when we need to incorporate data from our database into an object-oriented program or model.

Let's start by getting some practice with `session` objects, because that's where all the magic happens.

Using `.query` Objects

Recall that we created a `session` object at the beginning of this lab by using SQLAlchemy's `sessionmaker` function and binding it to our `engine` object. We haven't used our `session` object too much thus far, but now we'll use it for queries!

The `session` object contains a `.query()` method which returns a query object containing the results of our query, with the results mapped to objects.

Before we can make use of Object-Relational Mappings, we need to make sure that we have mappings created that map the tables in our existing database to objects in python. We don't want to have to do this manually, so we'll make use the `automap` module inside of `sqlalchemy.ext`.

In the cell below:

- Import `MetaData` from `sqlalchemy`
- Import `automap_base` from `sqlalchemy.ext.automap`
- Create a `MetaData` object
- Use the metadata object's `reflect` method on our `engine`
- Call `automap_base` and set the `metadata` parameter to our `metadata`. Store the results returned inside of the variable `Base`
- Call `base.prepare()`
- Map `Employee` and `Customer` to the `Employee` and `Customer` classes, which can be found inside of `Base.classes`

In [90]:
```
from sqlalchemy import MetaData
from sqlalchemy.ext.automap import automap_base

metadata = MetaData()

metadata.reflect(engine)

Base = automap_base(metadata=metadata)

Base.prepare()

Employee, Customer = Base.classes.Employee, Base.classes.Customer
```

```
2018-10-28 22:19:08,162 INFO sqlalchemy.engine.base.Engine SELECT name FROM sqlite_master WHERE type='table' ORDER BY name

INFO:sqlalchemy.engine.base.Engine:SELECT name FROM sqlite_master WHERE type='table' ORDER BY name

2018-10-28 22:19:08,163 INFO sqlalchemy.engine.base.Engine ()

INFO:sqlalchemy.engine.base.Engine:()

2018-10-28 22:19:08,176 INFO sqlalchemy.engine.base.Engine PRAGMA table_info("Category")

INFO:sqlalchemy.engine.base.Engine:PRAGMA table_info("Category")

2018-10-28 22:19:08,177 INFO sqlalchemy.engine.base.Engine ()

INFO:sqlalchemy.engine.base.Engine:()

2018-10-28 22:19:08,180 INFO sqlalchemy.engine.base.Engine SELECT sql FROM  (SELECT * FROM sqlite_master UNION ALL   SELECT * FROM sqlite_temp_master) WHERE name = 'Category' AND type = 'table'

INFO:sqlalchemy.engine.base.Engine:SELECT sql FROM  (SELECT * FROM sqlite_master UNION ALL   SELECT * FROM sqlite_temp_master) WHERE name = 'Category' AND type = 'table'
```

Now that we have some mappings set up, we can make use of `session.query()` help us query our database!

Writing Basic Queries

Let's use the `query()` object to get all the employees from the `'Employee'` table.

In the cell below:

- Create a for loop that iterates through the results returned by a `session.query()` of the Employee table (pass this as a variable, not a string).
- Order the results by the Employee's `.HireDate` attribute.
- Print the last name, first name, and hire date of each employee.

In [95]:
```
for instance in session.query(Employee).order_by(Employee.HireDate):
    print("Name: {}, {}  Hired: {}".format(instance.LastName, instance.FirstName, instance.HireDate))
```

```
2018-10-28 22:28:23,210 INFO sqlalchemy.engine.base.Engine SELECT "Employee"."Id" AS "Employee_Id", "Employee"."LastName" AS "Employee_LastName", "Employee"."FirstName" AS "Employee_FirstName", "Employee"."Title" AS "Employee_Title", "Employee"."TitleOfCourtesy" AS "Employee_TitleOfCourtesy", "Employee"."BirthDate" AS "Employee_BirthDate", "Employee"."HireDate" AS "Employee_HireDate", "Employee"."Address" AS "Employee_Address", "Employee"."City" AS "Employee_City", "Employee"."Region" AS "Employee_Region", "Employee"."PostalCode" AS "Employee_PostalCode", "Employee"."Country" AS "Employee_Country", "Employee"."HomePhone" AS "Employee_HomePhone", "Employee"."Extension" AS "Employee_Extension", "Employee"."Photo" AS "Employee_Photo", "Employee"."Notes" AS "Employee_Notes", "Employee"."ReportsTo" AS "Employee_ReportsTo", "Employee"."PhotoPath" AS "Employee_PhotoPath"
FROM "Employee" ORDER BY "Employee"."HireDate"
```

```
INFO:sqlalchemy.engine.base.Engine:SELECT "Employee"."Id" AS "Employee_Id", "Employee"."LastName" AS "Employee_LastName", "Employee"."FirstName" AS "Employee_FirstName", "Employee"."Title" AS "Employee_Title", "Employee"."TitleOfCourtesy" AS "Employee_TitleOfCourtesy", "Employee"."BirthDate" AS "Employee_BirthDate", "Employee"."HireDate" AS "Employee_HireDate", "Employee"."Address" AS "Employee_Address", "Employee"."City" AS "Employee_City", "Employee"."Region" AS "Employee_Region", "Employee"."PostalCode" AS "Employee_PostalCode", "Employee"."Country" AS "Employee_Country", "Employee"."HomePhone" AS "Employee_HomePhone", "Employee"."Extension" AS "Employee_Extension", "Employee"."Photo" AS "Employee_Photo", "Employee"."Notes" AS "Employee_Notes", "Employee"."ReportsTo" AS "Employee_ReportsTo", "Employee"."PhotoPath" AS "Employee_PhotoPath"
FROM "Employee" ORDER BY "Employee"."HireDate"
```

```
2018-10-28 22:28:23,212 INFO sqlalchemy.engine.base.Engine ()
```

```
INFO:sqlalchemy.engine.base.Engine:()
```

```
Name: Leverling, Janet  Hired: 2024-04-01
Name: Davolio, Nancy  Hired: 2024-05-01
Name: Fuller, Andrew  Hired: 2024-08-14
Name: Peacock, Margaret  Hired: 2025-05-03
Name: Buchanan, Steven  Hired: 2025-10-17
Name: Suyama, Michael  Hired: 2025-10-17
Name: King, Robert  Hired: 2026-01-02
Name: Callahan, Laura  Hired: 2026-03-05
Name: Dodsworth, Anne  Hired: 2026-11-15
```

Implicit JOINs using `.filter()`

One great benefit of using `session.query()` to query our data is that we can easily execute ***implicit joins*** by making use of the `.filter()` method.

So far we've only explicitly specified mappings for the `Employee` and `Customer` classes. We'll need to do this now for the `Product` and `Category` classes before we can use them with `session.query()`.

In the cell below, set the mappings for `Product` and `Category`.

HINT: This will look just like the last line of the code cell where we declared our mappings previously. No need to repeat all the steps for getting metadata and creating a `Base` class. Just set the mappings!

In [98]:
```
Product, Category = Base.classes.Product, Base.classes.Category
```

Great!

Now, in the cell below:

- Create a for loop that iterates through all results returned from a query of Products and Categories
- Use the `.filter()` method to only include cases where the Product's `.CategoryID` matches the Category's `.Id` attribute.
- Print out the name of each product, followed by the name of the category that it belongs to.

In [99]:
```
for p, c in session.query(Product, Category).filter(Product.CategoryId==Category.Id).all():
    print("Product Name: {}  Category Name: {}".format(p.ProductName, c.CategoryName))
```

```
2018-10-28 22:34:35,617 INFO sqlalchemy.engine.base.Engine SELECT "Product"."Id" AS "Product_Id", "Product"."ProductName" AS "Product_ProductName", "Product"."SupplierId" AS "Product_SupplierId", "Product"."CategoryId" AS "Product_CategoryId", "Product"."QuantityPerUnit" AS "Product_QuantityPerUnit", "Product"."UnitPrice" AS "Product_UnitPrice", "Product"."UnitsInStock" AS "Product_UnitsInStock", "Product"."UnitsOnOrder" AS "Product_UnitsOnOrder", "Product"."ReorderLevel" AS "Product_ReorderLevel", "Product"."Discontinued" AS "Product_Discontinued", "Category"."Id" AS "Category_Id", "Category"."CategoryName" AS "Category_CategoryName", "Category"."Description" AS "Category_Description"
FROM "Product", "Category"
WHERE "Product"."CategoryId" = "Category"."Id"
```

```
INFO:sqlalchemy.engine.base.Engine:SELECT "Product"."Id" AS "Product_Id", "Product"."ProductName" AS "Product_ProductName", "Product"."SupplierId" AS "Product_SupplierId", "Product"."CategoryId" AS "Product_CategoryId", "Product"."QuantityPerUnit" AS "Product_QuantityPerUnit", "Product"."UnitPrice" AS "Product_UnitPrice", "Product"."UnitsInStock" AS "Product_UnitsInStock", "Product"."UnitsOnOrder" AS "Product_UnitsOnOrder", "Product"."ReorderLevel" AS "Product_ReorderLevel", "Product"."Discontinued" AS "Product_Discontinued", "Category"."Id" AS "Category_Id", "Category"."CategoryName" AS "Category_CategoryName", "Category"."Description" AS "Category_Description"
FROM "Product", "Category"
WHERE "Product"."CategoryId" = "Category"."Id"
```

```
2018-10-28 22:34:35,618 INFO sqlalchemy.engine.base.Engine ()
```

```
INFO:sqlalchemy.engine.base.Engine:()
```

```
Product Name: Chai  Category Name: Beverages
Product Name: Chang  Category Name: Beverages
Product Name: Aniseed Syrup  Category Name: Condiments
Product Name: Chef Anton's Cajun Seasoning  Category Name: Condiments
Product Name: Chef Anton's Gumbo Mix  Category Name: Condiments
Product Name: Grandma's Boysenberry Spread  Category Name: Condiments
Product Name: Uncle Bob's Organic Dried Pears  Category Name: Produce
Product Name: Northwoods Cranberry Sauce  Category Name: Condiments
Product Name: Mishi Kobe Niku  Category Name: Meat/Poultry
Product Name: Ikura  Category Name: Seafood
Product Name: Queso Cabrales  Category Name: Dairy Products
Product Name: Queso Manchego La Pastora  Category Name: Dairy Products
Product Name: Konbu  Category Name: Seafood
Product Name: Tofu  Category Name: Produce
Product Name: Genen Shouyu  Category Name: Condiments
Product Name: Pavlova  Category Name: Confections
Product Name: Alice Mutton  Category Name: Meat/Poultry
Product Name: Carnarvon Tigers  Category Name: Seafood
Product Name: Teatime Chocolate Biscuits  Category Name: Confections
Product Name: Sir Rodney's Marmalade  Category Name: Confections
Product Name: Sir Rodney's Scones  Category Name: Confections
Product Name: Gustaf's Knäckebröd  Category Name: Grains/Cereals
Product Name: Tunnbröd  Category Name: Grains/Cereals
Product Name: Guaraná Fantástica  Category Name: Beverages
Product Name: NuNuCa Nuß-Nougat-Creme  Category Name: Confections
Product Name: Gumbär Gummibärchen  Category Name: Confections
Product Name: Schoggi Schokolade  Category Name: Confections
Product Name: Rössle Sauerkraut  Category Name: Produce
Product Name: Thüringer Rostbratwurst  Category Name: Meat/Poultry
Product Name: Nord-Ost Matjeshering  Category Name: Seafood
Product Name: Gorgonzola Telino  Category Name: Dairy Products
Product Name: Mascarpone Fabioli  Category Name: Dairy Products
Product Name: Geitost  Category Name: Dairy Products
Product Name: Sasquatch Ale  Category Name: Beverages
Product Name: Steeleye Stout  Category Name: Beverages
Product Name: Inlagd Sill  Category Name: Seafood
Product Name: Gravad lax  Category Name: Seafood
Product Name: Côte de Blaye  Category Name: Beverages
Product Name: Chartreuse verte  Category Name: Beverages
Product Name: Boston Crab Meat  Category Name: Seafood
Product Name: Jack's New England Clam Chowder  Category Name: Seafood
Product Name: Singaporean Hokkien Fried Mee  Category Name: Grains/Cereals
Product Name: Ipoh Coffee  Category Name: Beverages
```

```
Product Name: Gula Malacca  Category Name: Condiments
Product Name: Rogede sild  Category Name: Seafood
Product Name: Spegesild  Category Name: Seafood
Product Name: Zaanse koeken  Category Name: Confections
Product Name: Chocolade  Category Name: Confections
Product Name: Maxilaku  Category Name: Confections
Product Name: Valkoinen suklaa  Category Name: Confections
Product Name: Manjimup Dried Apples  Category Name: Produce
Product Name: Filo Mix  Category Name: Grains/Cereals
Product Name: Perth Pasties  Category Name: Meat/Poultry
Product Name: Tourtière  Category Name: Meat/Poultry
Product Name: Pâté chinois  Category Name: Meat/Poultry
Product Name: Gnocchi di nonna Alice  Category Name: Grains/Cereals
Product Name: Ravioli Angelo  Category Name: Grains/Cereals
Product Name: Escargots de Bourgogne  Category Name: Seafood
Product Name: Raclette Courdavault  Category Name: Dairy Products
Product Name: Camembert Pierrot  Category Name: Dairy Products
Product Name: Sirop d'érable  Category Name: Condiments
Product Name: Tarte au sucre  Category Name: Confections
Product Name: Vegie-spread  Category Name: Condiments
Product Name: Wimmers gute Semmelknödel  Category Name: Grains/Cereals
Product Name: Louisiana Fiery Hot Pepper Sauce  Category Name: Condiments
Product Name: Louisiana Hot Spiced Okra  Category Name: Condiments
Product Name: Laughing Lumberjack Lager  Category Name: Beverages
Product Name: Scottish Longbreads  Category Name: Confections
Product Name: Gudbrandsdalsost  Category Name: Dairy Products
Product Name: Outback Lager  Category Name: Beverages
Product Name: Flotemysost  Category Name: Dairy Products
Product Name: Mozzarella di Giovanni  Category Name: Dairy Products
Product Name: Röd Kaviar  Category Name: Seafood
Product Name: Longlife Tofu  Category Name: Produce
Product Name: Rhönbräu Klosterbier  Category Name: Beverages
Product Name: Lakkalikööri  Category Name: Beverages
Product Name: Original Frankfurter grüne Soße  Category Name: Condiments
```

Summary

Great job! You've just used SQLAlchemy to work with a sample production database. Note that there are many, many more awesome things that SQLAlchemy can do, but they're outside the scope of this lesson. However, if you're interested in learning more, don't be afraid to take a look at the SQLAlchemy documentation and work through some tutorials in your spare time!