Introduction to Databases

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

A database consists of tables

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

State_Fact

name	abbreviation	type
New York	NY	state
Washington DC	DC	capitol
Washington	WA	state

Table consist of columns and rows

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

Table consist of columns and rows

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

Tables can be related

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

State_Fact

name	abbreviation	type
New York	NY	state
Washington DC	DC	capitol
Washington	WA	state

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Connecting to your database

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Meet SQLAlchemy

- Two main pieces
  - Core (Relational Model focused)
  - ORM (User Data Model focused)

There are many types of databases

- SQLite
- PostgreSQL
- MySQL
- Microsoft SQL Server
- Oracle SQL
- Many more

Connecting to a database

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///census_nyc.sqlite')
connection = engine.connect()
```

- Engine: common interface to the database from SQLAlchemy
- Connection string: All the details required to find the database (and login, if necessary)

A word on connection strings

```
'sqlite:///census_nyc.sqlite'
```

Driver + Dialect

A word on connection strings

```
'sqlite:///census_nyc.sqlite'
```

Filename

What's in your database?

Before querying your database, you'll want to know what is in it: what the tables are, for example:

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///census_nyc.sqlite')
print(engine.table_names())
```

```
['census', 'state_fact']
```

Reflection

Reflection reads database and builds SQLAlchemy `Table` objects

```
from sqlalchemy import MetaData, Table
metadata = MetaData()
census = Table('census', metadata, autoload=True,
  autoload_with=engine)
print(repr(census))
```

```
Table('census', MetaData(bind=None), Column('state', VARCHAR(
length=30), table=<census>), Column('sex', VARCHAR(length=1),
table=<census>), Column('age', INTEGER(), table=<census>),
Column('pop2000', INTEGER(), table=<census>), Column('pop2008',
INTEGER(), table=<census>), schema=None)
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Introduction to SQL queries

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

SQL Statements

- Select, insert, update, and delete data
- Create and alter data

Basic SQL querying

```
SELECT column_name FROM table_name
```

- `SELECT pop2008 FROM People`
- `SELECT * FROM People`

Basic SQL querying

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///census_nyc.sqlite')
connection = engine.connect()
stmt = 'SELECT * FROM people'
result_proxy = connection.execute(stmt)
results = result_proxy.fetchall()
```

ResultProxy vs ResultSet

```
result_proxy = connection.execute(stmt)


results = result_proxy.fetchall()
```

- `result_proxy` is a `ResultProxy`
- `results` is a `ResultSet`

Handling ResultSets

```
first_row = results[0]
print(first_row)
```

```
('Illinois', 'M', 0, 89600, 95012)
```

```
print(first_row.keys())
```

```
['state', 'sex', 'age', 'pop2000', 'pop2008']
```

```
print(first_row.state)
```

```
'Illinois'
```

SQLAlchemy to build queries

- Provides a Pythonic way to build SQL statements
- Hides differences between backend database types

SQLAlchemy querying

```
from sqlalchemy import Table, MetaData
metadata = MetaData()
census = Table('census', metadata, autoload=True,
  autoload_with=engine)
stmt = select([census])
results = connection.execute(stmt).fetchall()
```

SQLAlchemy select statement

- Requires a list of one or more Tables or Columns
- Using a table will select all the columns in it

```
stmt = select([census])
print(stmt)
```

```
'SELECT * from CENSUS'
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Congratulations!

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

You already...

- Know about the relational model
- Can make basic SQL queries

Coming up next...

- Beef up your SQL querying skills
- Learn how to extract all types of useful information from your databases using SQLAlchemy
- Learn how to create and write to relational databases
- Deep dive into the US census dataset!

See you in the next chapter!

INTRODUCTION TO DATABASES IN PYTHON

Filtering and targeting data

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Where clauses

```
stmt = select([census])
stmt = stmt.where(census.columns.state == 'California')
results = connection.execute(stmt).fetchall()
for result in results:
    print(result.state, result.age)
```

```
California 0
California 1
California 2
California 3
California 4
California 5
...
```

Where clauses

- Restrict data returned by a query based on Boolean conditions
- Compare a column against a value or another column
- Often use comparisons `==` , `<=` , `>=` , or `!=`

Expressions

- Provide more complex conditions than simple operators
- E.g. `in_()`, `like()`, `between()`
- Many more in documentation
- Available as method on a `Column`

Expressions

```
stmt = select([census])
stmt = stmt.where(census.columns.state.startswith('New'))
for result in connection.execute(stmt):
    print(result.state, result.pop2000)
```

```
New Jersey 56983
New Jersey 56686
New Jersey 57011
...
```

Conjunctions

- Allow us to have multiple criteria in a where clause
- Eg. `and_()` , `or_()` , `not_()`

Conjunctions

```
from sqlalchemy import or_
stmt = select([census])
stmt = stmt.where(
    or_(census.columns.state == 'California',
        census.columns.state == 'New York'
    )
)
for result in connection.execute(stmt):
    print(result.state, result.sex)
```

```
New York M
...
California F
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Ordering query results

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Order by clauses

- Allows us to control the order in which records are returned in the query results
- Available as a method on statements `order_by()`

Order by ascending

```
print(results[:10])
```

```
[('Illinois',), ...]
```

```
stmt = select([census.columns.state])
stmt = stmt.order_by(census.columns.state)
results = connection.execute(stmt).fetchall()
print(results[:10])
```

```
[('Alabama',), ...]
```

Order by descending

- Wrap the column with `desc()` in the `order_by()` clause

Order by multiple

- Just separate multiple columns with a comma
- Orders completely by the first column
- Then if there are duplicates in the first column, orders by the second column
- Repeat until all columns are ordered

Order by multiple

```
print(results)
```

```
('Alabama', 'M')
```

```
stmt = select([census.columns.state, census.columns.sex])
stmt = stmt.order_by(census.columns.state, census.columns.sex)
results = connection.execute(stmt).first()
print(results)
```

```
('Alabama', 'F')
('Alabama', 'F')
...
('Alabama', 'M')
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Counting, summing, and grouping data

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

SQL functions

- E.g. `COUNT`, `SUM`
- `from sqlalchemy import func`
- More efficient than processing in Python
- Aggregate data

Sum example

```
from sqlalchemy import func
stmt = select([func.sum(census.columns.pop2008)])
results = connection.execute(stmt).scalar()
print(results)
```

```
302876613
```

Group by

- Allows us to group row by common values

Group by

```
stmt = select([census.columns.sex,
  func.sum(census.columns.pop2008)])
stmt = stmt.group_by(census.columns.sex)
results = connection.execute(stmt).fetchall()
print(results)
```

```
[('F', 153959198), ('M', 148917415)]
```

Group by

- Supports multiple columns to group by with a pattern similar to `order_by()`
- Requires all selected columns to be grouped or aggregated by a function

Group by multiple

```
stmt = select([census.columns.sex,
         census.columns.age,
         func.sum(census.columns.pop2008)
    ])
stmt = stmt.group_by(census.columns.sex,
         census.columns.age)
results = connection.execute(stmt).fetchall()
print(results)
```

```
[('F', 0, 2105442), ('F', 1, 2087705), ('F', 2, 2037280),
('F', 3, 2012742), ('F', 4, 2014825), ('F', 5, 1991082),
('F', 6, 1977923), ('F', 7, 2005470), ('F', 8, 1925725), ..
```

Handling ResultSets from functions

- SQLAlchemy auto generates "column names" for functions in the `ResultSet`
- The column names are often `func_#` such as `count_1`
- Replace them with the `label()` method

Using label()

```
print(results[0].keys())
```

```
['sex', u'sum_1']
```

```
stmt = select([census.columns.sex,
        func.sum(census.columns.pop2008).label('pop2008_sum')
    ])
stmt = stmt.group_by(census.columns.sex)
results = connection.execute(stmt).fetchall()
print(results[0].keys())
```

```
['sex', 'pop2008_sum']
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

SQLAlchemy and pandas for visualization

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

SQLAlchemy and pandas

- DataFrame can take a SQLAlchemy `ResultSet`
- Make sure to set the DataFrame columns to the `ResultSet` keys

DataFrame example

```
import pandas as pd
df = pd.DataFrame(results)
df.columns = results[0].keys()
print(df)
```

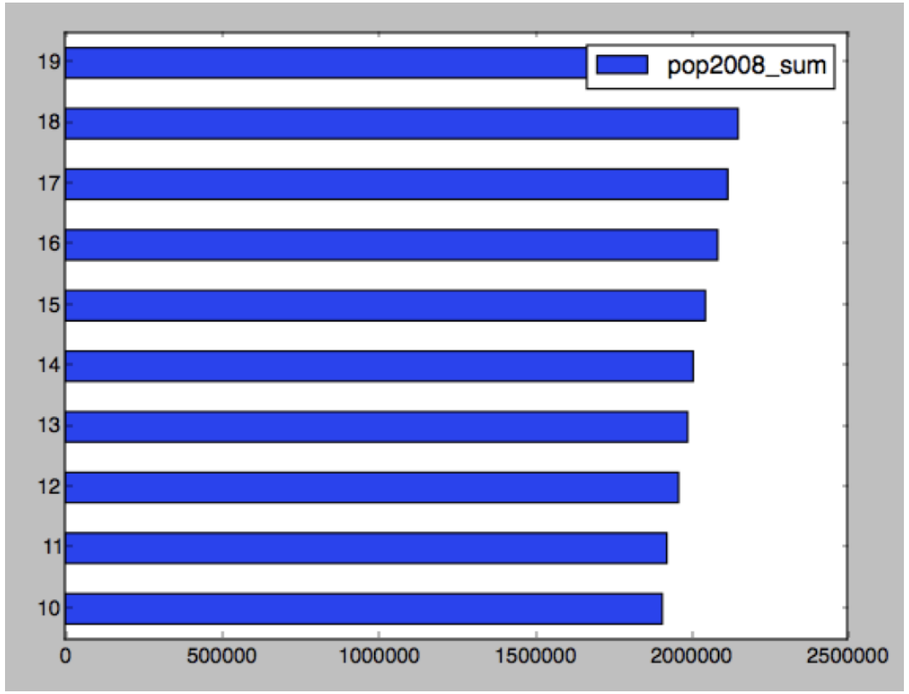
```
    sex  pop2008_sum
0     F      2105442
1     F      2087705
2     F      2037280
3     F      2012742
4     F      2014825
5     F      1991082
```

Graphing

- We can graph just like we would normally

Graphing example

```
import matplotlib.pyplot as plt
df[10:20].plot.barh()
plt.show()
```



Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Calculating values in a query

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Math operators

- addition `+`
- subtraction `-`
- multiplication `*`
- division `/`
- modulus `%`
- Work differently on different data types

Calculating difference

```
stmt = select([census.columns.age,
         (census.columns.pop2008 -
         census.columns.pop2000).label('pop_change')
    ])
stmt = stmt.group_by(census.columns.age)
stmt = stmt.order_by(desc('pop_change'))
stmt = stmt.limit(5)
results = connection.execute(stmt).fetchall()
print(results)
```

```
[(61, 52672), (85, 51901), (54, 50808), (58, 45575),
(60, 44915)]
```

Case statement

- Used to treat data differently based on a condition
- Accepts a list of conditions to match and a column to return if the condition matches
- The list of conditions ends with an else clause to determine what to do when a record doesn't match any prior conditions

Case example

```
from sqlalchemy import case
stmt = select([
        func.sum(
            case([
                (census.columns.state == 'New York',
                 census.columns.pop2008)
            ], else_=0))])
results = connection.execute(stmt).fetchall()
print(results)
```

```
[(19465159,)]
```

Cast statement

- Converts data to another type
- Useful for converting...
  - integers to floats for division
  - strings to dates and times
- Accepts a column or expression and the target Type

Percentage example

```
from sqlalchemy import case, cast, Float
stmt = select([
        (func.sum(
            case([
                (census.columns.state == 'New York',
                 census.columns.pop2008)
            ], else_=0)) /
         cast(func.sum(census.columns.pop2008),
              Float) * 100).label('ny_percent')])
results = connection.execute(stmt).fetchall()
print(results)
```

```
[(Decimal('6.4267619765'),)]
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

SQL relationships

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Relationships

- Allow us to avoid duplicate data
- Make it easy to change things in one place
- Useful to break out information from a table we don't need very often

Relationships

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

State_Fact

name	abbreviation	type
New York	NY	state
Washington DC	DC	capitol
Washington	WA	state

Automatic joins

```
stmt = select([census.columns.pop2008,
               state_fact.columns.abbreviation])
results = connection.execute(stmt).fetchall()
print(results)
```

```
[(95012, u'IL'),
 (95012, u'NJ'),
 (95012, u'ND'),
 (95012, u'OR'),
 (95012, u'DC'),
 (95012, u'WI'),
 ...
```

Join

- Accepts a Table and an optional expression that explains how the two tables are related
- The expression is not needed if the relationship is predefined and available via reflection
- Comes immediately after the `select()` clause and prior to any `where()`, `order_by()` or `group_by()` clauses

select_from()

- Used to replace the default, derived FROM clause with a join
- Wraps the `join()` clause

select_from() example

```
stmt = select([func.sum(census.columns.pop2000)])
stmt = stmt.select_from(census.join(state_fact))
stmt = stmt.where(state_fact.columns.circuit_court == '10')
result = connection.execute(stmt).scalar()
print(result)
```

```
14945252
```

Joining tables without predefined relationship

- Join accepts a Table and an optional expression that explains how the two tables are related
- Will only join on data that match between the two columns
- Avoid joining on columns of different types

select_from() example

```
stmt = select([func.sum(census.columns.pop2000)])
stmt = stmt.select_from(
        census.join(state_fact, census.columns.state
        == state_fact.columns.name))
stmt = stmt.where(
        state_fact.columns.census_division_name ==
        'East South Central')
result = connection.execute(stmt).scalar()
print(result)
```

```
16982311
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Working with hierarchical tables

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Hierarchical tables

- Contain a relationship with themselves
- Commonly found in:
  - Organizational
  - Geographic
  - Network
  - Graph

Hierarchical tables - example

Employees

id	name	job	manager
1	Johnson	Admin	6
2	Harding	Manager	9
3	Taft	Sales I	2
4	Hoover	Sales I	2

Hierarchical tables - alias()

- Requires a way to view the table via multiple names
- Creates a unique reference that we can use

Querying hierarchical data

```
managers = employees.alias()
stmt = select(
        [managers.columns.name.label('manager'),
         employees.columns.name.label('employee')])
stmt = stmt.select_from(employees.join(
        managers, managers.columns.id ==
        employees.columns.manager)
stmt = stmt.order_by(managers.columns.name)
print(connection.execute(stmt).fetchall())
```

```
[(u'FILLMORE', u'GRANT'),
 (u'FILLMORE', u'ADAMS'),
 (u'HARDING', u'TAFT'), ...
```

group_by and func

- It's important to target `group_by()` at the right alias
- Be careful with what you perform functions on
- If you don't find yourself using both the alias and the table name for a query, don't create the alias at all

Querying hierarchical data

```
managers = employees.alias()
stmt = select([managers.columns.name,
        func.sum(employees.columns.sal)])
stmt = stmt.select_from(employees.join(
        managers, managers.columns.id ==
        employees.columns.manager)
stmt = stmt.group_by(managers.columns.name)
print(connection.execute(stmt).fetchall())
```

```
[(u'FILLMORE', Decimal('96000.00')),
 (u'GARFIELD', Decimal('83500.00')),
 (u'HARDING', Decimal('52000.00')),
 (u'JACKSON', Decimal('197000.00'))]
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Handling large ResultSets

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Dealing with large ResultSets

- `fetchmany()` lets us specify how many rows we want to act upon
- We can loop over `fetchmany()`
- It returns an empty list when there are no more records
- We have to close the `ResultProxy` afterwards

Fetching many rows

```
while more_results:
    partial_results = results_proxy.fetchmany(50)
    if partial_results == []:
        more_results = False
    for row in partial_results:
        state_count[row.state] += 1
results_proxy.close()
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Creating databases and tables

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Creating databases

- Varies by the database type
- Databases like PostgreSQL and MySQL have command-line tools to initialize the database
- With SQLite, the `create_engine()` statement will create the database and file is they do not already exist

Building a table

```
from sqlalchemy import (Table, Column, String,
       Integer, Decimal, Boolean)
employees = Table('employees', metadata,
       Column('id', Integer()),
       Column('name', String(255)),
       Column('salary', Decimal()),
       Column('active', Boolean()))
metadata.create_all(engine)
engine.table_names()
```

```
[u'employees']
```

Creating tables

- Still uses the Table object like we did for reflection
- Replaces the `autoload` keyword arguments with Column objects
- Creates the tables in the actual database by using the `create_all()` method on the MetaData instance
- You need to use other tools to handle database table updates, such as Alembic or raw SQL

Creating tables - additional column options

- `unique` forces all values for the data in a column to be unique
- `nullable` determines if a column can be empty in a row
- `default` sets a default value if one isn't supplied.

Building a table with additional options

```
employees = Table('employees', metadata,
       Column('id', Integer()),
       Column('name', String(255), unique=True, nullable=False),
       Column('salary', Float(), default=100.00),
       Column('active', Boolean(), default=True))
employees.constraints
```

```
{CheckConstraint(...
Column('name', String(length=255), table=<employees>, nullable=Fals
Column('salary', Float(), table=<employees>,
       default=ColumnDefault(100.0)),
Column('active', Boolean(), table=<employees>,
       default=ColumnDefault(True)), ...
UniqueConstraint(Column('name', String(length=255),
                 table=<employees>, nullable=False))}
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Inserting data into a table

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Adding data to a table

- Done with the `insert()` statement
- `insert()` takes the table we are loading data into as the argument
- We add all the values we want to insert in with the `values` clause as `column=value` pairs
- Doesn't return any rows, so no need for a fetch method

Inserting one row

```
from sqlalchemy import insert
```

```
stmt = insert(employees).values(id=1,name='Jason',
           salary=1.00, active=True)
result_proxy = connection.execute(stmt)
print(result_proxy.rowcount)
```

```
1
```

Inserting multiple rows

- Build an insert statement without any values
- Build a list of dictionaries that represent all the values clauses for the rows you want to insert
- Pass both the statement and the values list to the execute method on connection

Inserting multiple rows

```
stmt = insert(employees)
values_list = [{'id': 2, 'name': 'Rebecca',
                'salary': 2.00, 'active': True},
               {'id': 3, 'name': 'Bob',
                'salary': 0.00, 'active': False}]
result_proxy = connection.execute(stmt, values_list)
print(result_proxy.rowcount)
```

```
2
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Updating data in a table

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Updating data in a table

- Done with the `update()` statement
- Similar to the `insert()` statement but includes a `where` clause to determine what record will be updated
- We add all the values we want to update with the `values()` clause as `column=value` pairs

Updating one row

```
from sqlalchemy import update
stmt = update(employees)
stmt = stmt.where(employees.columns.id == 3)
stmt = stmt.values(active=True)
result_proxy = connection.execute(stmt)
print(result_proxy.rowcount)
```

```
1
```

Updating multiple rows

- Build a `where` clause that will select all the records you want to update

Inserting multiple rows

```
stmt = update(employees)
stmt = stmt.where(employees.columns.active == True)
stmt = stmt.values(active=False, salary=0.00)
result_proxy = connection.execute(stmt)
print(result_proxy.rowcount)
```

```
3
```

Correlated updates

```
new_salary = select([employees.columns.salary])
new_salary = new_salary.order_by(
    desc(employees.columns.salary))
new_salary = new_salary.limit(1)
stmt = update(employees)
stmt = stmt.values(salary=new_salary)
result_proxy = connection.execute(stmt)
print(result_proxy.rowcount)
```

```
3
```

Correlated updates

- Uses a `select()` statement to find the value for the column we are updating
- Commonly used to update records to a maximum value or change a string to match an abbreviation from another table

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Deleting data from a database

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

Deleting data from a table

- Done with the `delete()` statement
- `delete()` takes the table we are loading data into as the argument
- A `where()` clause is used to choose which rows to delete
- Hard to undo so be careful!

Deleting all data from a table

```
from sqlalchemy import delete
stmt = select([func.count(extra_employees.columns.id)])
connection.execute(stmt).scalar()
```

```
3
```

```
delete_stmt = delete(extra_employees)
result_proxy = connection.execute(delete_stmt)
result_proxy.rowcount
```

```
3
```

Deleting specific rows

- Build a `where()` clause that will select all the records you want to delete

Deleting specific rows

```
stmt = delete(employees).where(employees.columns.id == 3)
result_proxy = connection.execute(stmt)
result_proxy.rowcount
```

```
1
```

Dropping a table completely

- Uses the `drop()` method on the table
- Accepts the engine as an argument so it knows where to remove the table from
- Won't remove it from metadata until the Python process is restarted

Dropping a table

```
extra_employees.drop(engine)
print(extra_employees.exists(engine))
```

```
False
```

Dropping all the tables

- Uses the `drop_all()` method on MetaData

Dropping all the tables

```
metadata.drop_all(engine)
engine.table_names()
```

```
[]
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Census case study

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

Census case study

- Preparing SQLAlchemy and the database
- Loading data into the database
- Solving data science problems with queries

Part 1: preparing SQLAlchemy and the database

- Create an engine and `MetaData` object

```
from sqlalchemy import create_engine, MetaData
engine = create_engine('sqlite:///census_nyc.sqlite')
metadata = MetaData()
```

Part 1: preparing SQLAlchemy and the database

- Create and save the census table

```
from sqlalchemy import (Table, Column, String,
        Integer, Decimal, Boolean)
employees = Table('employees', metadata,
        Column('id', Integer()),
        Column('name', String(255)),
        Column('salary', Decimal()),
        Column('active', Boolean()))
metadata.create_all(engine)
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Populating the database

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Part 2: populating the database

- Load a CSV file into a values list

```
values_list = []
for row in csv_reader:
    data = {'state': row[0], 'sex': row[1], 'age': row[2],
            'pop2000': row[3], 'pop2008': row[4]}
    values_list.append(data)
```

Part 2: Populating the Database

- Insert the values list into the census table

```
from sqlalchemy import insert
stmt = insert(employees)
result_proxy = connection.execute(stmt, values_list)
print(result_proxy.rowcount)
```

```
2
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Querying the database

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

datacamp

Part 3: answering data science questions with queries

- Determine average age for males and females

```
from sqlalchemy import select
stmt = select([census.columns.sex,
          (func.sum(census.columns.pop2008 *
           census.columns.age) /
           func.sum(census.columns.pop2008)
         ).label('average_age')])
stmt = stmt.group_by('census.columns.sex')
results = connection.execute(stmt).fetchall()
```

Part 3: answering data science questions with queries

- Determine the percentage of Females for each state

```
from sqlalchemy import case, cast, Float
stmt = select([
          (func.sum(
              case([
                  (census.columns.state == 'New York',
                   census.columns.pop2008)
              ], else_=0)) /
           cast(func.sum(census.columns.pop2008),
                Float) * 100).label('ny_percent')])
```

Part 3: answering data science questions with queries

- Determine the top 5 states by population change from 2000 to 2008

```
stmt = select([census.columns.age,
          (census.columns.pop2008-
          census.columns.pop2000).label('pop_change')
      ])
stmt = stmt.order_by('pop_change')
stmt = stmt.limit(5)
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Congratulations!

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

Congratulations!

INTRODUCTION TO DATABASES IN PYTHON