Creating databases and tables

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

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=False
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

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

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