Programming for AI

Abdul Haseeb

BS(AI)-IV

SQL Alchemy

- **SQLAlchemy** simplifies the connection between Python and SQL databases by automatically converting Python class calls into SQL statements.

Works with Almost all databases

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

Engine

- To connect to a database, we need some way to talk to it.
- An Engine provides that common interface between the code and database

SQL Alchemy code to connect to a database using connection string and print table names

```
#Library
from sqlalchemy import create_engine, inspect

# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite')

# Use SQLAlchemy's inspector to get table names
inspector = inspect(engine)

# Get and print all table names
table_names = inspector.get_table_names()
```

Reflection

- Reads database and builds SQLAlchemy Table objects.

#Continued with previous code:

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

Column names

- print(census.columns.keys())

Datatypes and column names

```
#Continue with previous code
table = metadata.tables['census']

# Print column names and their data types
for column in table.columns:
    print(f"{column.name} - {column.type}")
```

SQL Query Language

- Select, insert, update and delete the data
- Create, alter and delete tables and columns

Selecting all rows and columns

```
from sqlalchemy import create_engine, text
engine = create_engine('sqlite:///census.sqlite')

with engine.connect() as connection:
    stmnt = text('SELECT * FROM census')
    result = connection.execute(stmnt)
    print(result.fetchall())
```

Explanation for the previous code

- Using text() for Raw SQL: SQLAlchemy's modern approach for executing raw SQL strings is to wrap them in the text() construct. This explicitly tells SQLAlchemy that the string is a SQL statement to be executed.

- Context Manager for Connection: Using a with engine.connect() as connection: block is the recommended way to manage connections. This ensures that the connection is automatically closed when you exit the with block, even if errors occur.

Printing first row and it’s keys

```
with engine.connect() as connection:
    stmnt = text('SELECT * FROM census')
    result = connection.execute(stmnt)
    results=result.fetchall()
    print(results[0])
    print(results[0].keys())
```

SQLAlchemy to build queries

- Provides a pythonic way to build sql statements

SQLAlchemy to build queries

```
#Library
from sqlalchemy import create_engine, MetaData, Table, select

# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite')
with engine.connect() as connection:
    metadata=MetaData()
    census=Table('census',metadata, autoload_with=engine)
    stmnt = select(census)
    result = connection.execute(stmnt)
    print(result.fetchall())
```

SQLAlchemy to build queries

```
# Library
from sqlalchemy import create_engine, inspect, text, MetaData, Table, select, Column

# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite')
with engine.connect() as connection:
    metadata = MetaData()
    census = Table('census', metadata, autoload_with=engine)

    # Select the 'state' and 'sex' columns
    stmnt = select(census.c.state, census.c.sex)
    result = connection.execute(stmnt)
    print(result.fetchall())
```

Filtering with Where

```
from sqlalchemy import create_engine, inspect, text, MetaData, Table, select, Column


# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite')
with engine.connect() as connection:
    metadata = MetaData()
    census = Table('census', metadata, autoload_with=engine)

    stmnt = select(census)
    stmnt =stmnt.where(census.columns.state=='California’)
    results=connection.execute(stmnt).fetchall()
    print(results)
```

Printing the state and age for all the filtered rows

- for result in results:
  - print(result.state, result.age)

Expressions

- Provide more complex conditions than simple operators
- E.g: in_(), like(), between()
- Exp:
  - stmnt=stmnt.where(census.columns.pop2000. between(2000,4500))

Conjuctions

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

Code to get all the records for California or New York

```
from sqlalchemy import create_engine, inspect, text, MetaData, Table, select, Column, or_

# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite')
with engine.connect() as connection:
    metadata = MetaData()
    census = Table('census', metadata, autoload_with=engine)

    stmnt=select(census)
    stmnt=stmnt.where(
        or_(census.columns.state==‘California’,
                census.columns.state==‘New York’)
        )
    for result in connection.execute(stmnt):
        print(result.state, result.sex)
```

Order By

```
from sqlalchemy import create_engine, inspect, text, MetaData, Table, select, Column, or_

# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite')
with engine.connect() as connection:
    metadata = MetaData()
    census = Table('census', metadata, autoload_with=engine)
    print(census.columns)
    stmnt=select(census.columns.state, census.columns.pop2008)
    stmnt= stmnt.where(census.columns.state=='California')
    stmnt = stmnt.order_by(census.columns.pop2008)
    for result in connection.execute(stmnt):
        print(result)
```

Descending

```
from sqlalchemy import create_engine, inspect, text, MetaData, Table, select, Column, or_,desc

# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite')
with engine.connect() as connection:
    metadata = MetaData()
    census = Table('census', metadata, autoload_with=engine)
    print(census.columns)
    stmnt=select(census.columns.state, census.columns.pop2008)
    stmnt= stmnt.where(census.columns.state=='California')
    stmnt = stmnt.order_by(desc(census.columns.pop2008))
    for result in connection.execute(stmnt):
        print(result)
```

Counting, Summing and Grouping Data

- **Aggregate functions:**
  - Collapse multiple records into one.

Summing

```
from sqlalchemy import create_engine, inspect, text, MetaData, Table, select, Column, or_,desc,func

# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite’)

with engine.connect() as connection:
    metadata = MetaData()
    census = Table('census', metadata, autoload_with=engine)
    print(census.columns)
    stmnt=select(func.sum(census.columns.pop2008))
    result=connection.execute(stmnt).scalar()
    print(result)
```

Group By

- Let’s get the population count by gender.

```
from sqlalchemy import create_engine, inspect, text, MetaData, Table, select, Column, or_,desc,func

# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite’)

with engine.connect() as connection:
    metadata = MetaData()
    census = Table('census', metadata, autoload_with=engine)
    print(census.columns)
    stmnt=select(census.columns.sex, func.sum(census.columns.pop2008))
    stmnt=stmnt.group_by(census.columns.sex)
    results=connection.execute(stmnt).fetchall()
    print(results)
```

Group By Multiple

- Let’s get the population count by gender.

```
from sqlalchemy import create_engine, inspect, text, MetaData, Table, select, Column, or_,desc,func

# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite’)

with engine.connect() as connection:
    metadata = MetaData()
    census = Table('census', metadata, autoload_with=engine)
    print(census.columns)
    stmnt=select(census.columns.sex, census.columns.age, func.sum(census.columns.pop2008))
    stmnt=stmnt.group_by(census.columns.sex, census.columns.age)
    results=connection.execute(stmnt).fetchall()
    print(results)
```

Converting the Result Set into pandas Dataframe

```
from sqlalchemy import create_engine, inspect, text, MetaData, Table, select, Column, or_,desc,func
import pandas as pd
# Create engine to connect to SQLite database
engine = create_engine('sqlite:///census.sqlite')

with engine.connect() as connection:
    metadata = MetaData()
    census = Table('census', metadata, autoload_with=engine)

    stmnt=select(census)
    results=connection.execute(stmnt).fetchall()

    df=pd.DataFrame(results)
    df.columns=census.columns
    print(df.head())
```