SQLAlchemy

SQLAlchemy - це Python SQL інструментарій, який дозволяє розробникам отримувати доступ до та керувати SQL-базами даних, використовуючи мову програмування Python. Ви можете складати запити у вигляді рядка або ланцюга об'єктів Python для аналогічних запитів. Робота з об'єктами надає розробникам гнучкість та дозволяє створювати високопродуктивні додатки на основі SQL.

У простих словах, це дозволяє користувачам підключати бази даних, використовуючи мову Python, виконувати SQL-запити з використанням об'єктно-орієнтованого програмування та оптимізувати робочий процес.

Встановлення SQLAlchemy

Встановлення цього пакету та початок програмування досить просте.

Ви можете встановити SQLAlchemy за допомогою Python Package Manager (pip):

```
pip install sqlalchemy
```

У випадку використання дистрибутиву Anaconda Python, спробуйте ввести команду у терміналі conda:

```
conda install -c anaconda sqlalchemy
```

Давайте перевіримо, чи пакет успішно встановлено:

```
>>> import sqlalchemy
>>> sqlalchemy.__version__
'1.4.41'
```

Відмінно, ми успішно встановили SQLAlchemy версії 1.4.41.

Початок роботи

У цьому розділі ми навчимося підключати бази даних SQLite, створювати об'єкти таблиць та використовувати їх для запуску SQL-запиту.

Підключення до бази даних

Ми використовуватимемо базу даних SQLite про європейський футбол з Kaggle, яка має дві таблиці: "divisions" та "matchs".

Спочатку ми створимо об'єкти SQLite engine, вказавши розташування бази даних. Потім створимо об'єкт з'єднання, підключивши engine. Ми використовуватимемо об'єкт 'conn' для запуску всіх видів SQL-запитів.

```
from sqlalchemy import create_engine
engine = create_engine("sqlite:///european_database.sqlite")

conn = engine.connect()
```

Якщо ви хочете підключитися до баз даних PostgreSQL, MySQL, Oracle або Microsoft SQL Server, перевірте конфігурацію engine для плавного з'єднання з сервером.

Цей навчальний посібник SQLAlchemy передбачає, що ви розумієте основи Python та SQL. Якщо ні, це абсолютно нормально. Ви можете пройти курси з основ SQL та Python, щоб побудувати міцну основу.

Отримання доступу до таблиці

Щоб створити об'єкт таблиці, нам потрібно надати імена таблиць та метадані. Ви можете отримати метадані, використовуючи функцію `MetaData()` у SQLAlchemy.

```
metadata = db.MetaData() # витягуємо метадані
division= db.Table('divisions', metadata, autoload=True, autoload_with=engine) # об'єкт таблиці
```

Давайте виведемо метадані "divisions".

```
print(repr(metadata.tables['divisions']))
```

Метадані містять назву таблиці, назви стовпців з типами даних та схему.

```
Table('divisions', MetaData(), Column('division', TEXT(), table=<divisions>),
Column('name', TEXT(), table=<divisions>), Column('country', TEXT(),
table=<divisions>), schema=None)
```

Давайте використаємо об'єкт таблиці "division", щоб вивести назви стовпців.

```
print(division.columns.keys())
```

Таблиця містить стовпці division, name та country.

```
['division', 'name', 'country']
```

Простий SQL-запит

Тепер найцікавіше. Ми використаємо об'єкт таблиці, щоб запустити запит і отримати результат.

У наведеному нижче коді ми вибираємо всі стовпці для таблиці "division".

```
query = division.select() # SELECT * FROM divisions
print(query)
```

Зверніть увагу: ви також можете написати команду select як `db.select([division])`.

Щоб переглянути запит, виведіть об'єкт запиту, і він покаже команду SQL.

```
SELECT divisions.division, divisions.name, divisions.country
FROM divisions
```

Результат SQL-запиту

Тепер ми виконаємо запит за

допомогою об'єкта з'єднання та витягнемо перші п'ять рядків.

- `fetchone()`: витягує один рядок за один раз.
- `fetchmany(n)`: витягує n кількість рядків за один раз.
- `fetchall()`: витягує всі рядки.

```
exe = conn.execute(query) # виконуємо запит
result = exe.fetchmany(5) # витягуємо перші 5 результатів
print(result)
```

Результат показує перші п'ять рядків таблиці.

```
[('B1', 'Division 1A', 'Belgium'), ('D1', 'Bundesliga', 'Deutschland'),
('D2', '2. Bundesliga', 'Deutschland'), ('E0', 'Premier League', 'England'),
('E1', 'EFL Championship', 'England')]
```

Це дозволяє розпочати роботу з SQLAlchemy та надає базове розуміння його функцій.

Приклади використання SQLAlchemy в Python

У цьому розділі ми розглянемо різноманітні приклади використання SQLAlchemy для створення таблиць, вставки значень, виконання SQL-запитів, аналізу даних та керування таблицями.

Створення таблиць

Спочатку ми створимо нову базу даних з назвою "datacamp.sqlite". Функція create_engine автоматично створить нову базу даних, якщо база з таким ім'ям ще не існує.

Після цього ми підключимося до бази даних та створимо об'єкт метаданих.

Ми використаємо функцію Table з SQLAlchemy, щоб створити таблицю під назвою "Student".

Вона складається з наступних стовпців:

- Id: Ціле число та первинний ключ
- Name: Рядок та не може бути пустим
- Major: Рядок із значенням за замовчуванням "Math"
- Pass: Логічне значення із значенням за замовчуванням True

Ми створили структуру таблиці. Давайте додамо її до бази даних за допомогою `metadata.create_all(engine)`.

```
engine = db.create_engine('sqlite:///datacamp.sqlite')
conn = engine.connect()
metadata = db.MetaData()

Student = db.Table('Student', metadata,
              db.Column('Id', db.Integer(),primary_key=True),
              db.Column('Name', db.String(255), nullable=False),
              db.Column('Major', db.String(255), default="Math"),
              db.Column('Pass', db.Boolean(), default=True)
              )

metadata.create_all(engine)
```

Вставка одного запису

Щоб додати один рядок, спочатку використовуємо `insert` та додаємо об'єкт таблиці. Після цього використовуємо `values` та додаємо значення в стовпці вручну. Це працює схоже до додавання аргументів до функцій Python.

Нарешті, ми виконаємо запит, використовуючи з'єднання для виклику функції.

```
query = db.insert(Student).values(Id=1, Name='Matthew', Major="English", Pass=True)
Result = conn.execute(query)
```

Давайте перевіримо, чи додали рядок до таблиці "Student", виконавши запит SELECT та витягнувши всі рядки.

```
output = conn.execute(Student.select()).fetchall()
print(output)
```

Ми успішно додали значення.

```
[(1, 'Matthew', 'English', True)]
```

Вставка багатьох записів

Додавання значень по одному - не практичний спосіб наповнення бази даних. Давайте додамо кілька значень, використовуючи списки.

Створіть запит на вставку для таблиці Student.
Створіть список кількох рядків з назвами стовпців і значеннями.
Виконайте запит, другий аргумент - values_list.

```
query = db.insert(Student)
values_list = [{'Id':'2', 'Name':'Nisha', 'Major':"Science", 'Pass':False},
               {'Id':'3', 'Name':'Natasha', 'Major':"Math", 'Pass':True},
               {'Id':'4', 'Name':'Ben', 'Major':"English", 'Pass':False}]
Result = conn.execute(query,values_list)
```

Щоб перевірити результати, виконайте простий запит SELECT.

```
output = conn.execute(db.select([Student])).fetchall()
print(output)
```

Тепер у таблиці є більше рядк

ів.

```
[(1, 'Matthew', 'English', True), (2, 'Nisha', 'Science', False), (3, 'Natasha', 'Math', True),
(4, 'Ben', 'English', False)]
```

Простий SQL-запит з SQLAlchemy

Замість використання об'єктів Python, ми також можемо виконувати SQL-запити, використовуючи рядок.

Просто додайте аргумент у вигляді рядка до функції `execute` та перегляньте результат за допомогою `fetchall`.

```
output = conn.execute("SELECT * FROM Student")
print(output.fetchall())
```

Ви можете навіть передавати складніші SQL-запити. У нашому випадку ми вибираємо стовпці Name і Major, де студенти успішно склали екзамен.

```
output = conn.execute("SELECT Name, Major FROM Student WHERE Pass = True")
print(output.fetchall())
```

Використання API SQLAlchemy

У попередніх розділах ми використовували прості об'єкти/API SQLAlchemy. Розгляньте більш складні та багатокрокові запити.

У прикладі нижче ми виберемо всі стовпці, де спеціальність студента - англійська.

```
query = Student.select().where(Student.columns.Major == 'English')
output = conn.execute(query)
print(output.fetchall())
```

Давайте застосуємо логіку AND до запиту WHERE. У нашому випадку ми шукаємо студентів, які мають спеціальність англійською, і вони не склали екзамен.

Зауважте: не рівне '!=' True - це False.

```
query = Student.select().where(db.and_(Student.columns.Major == 'English', Student.columns.Pass !=
True))
output = conn.execute(query)
print(output.fetchall())
```

Лише Бен не склав екзамен з англійської мови.

```
[(4, 'Ben', 'English', False)]
```

Використовуючи подібну таблицю, ми можемо виконувати всі види команд, як показано у таблиці нижче.

Ви можете копіювати та вставити ці команди, щоб перевірити результати самостійно. Перевірте робоче середовище DataCamp, якщо у вас виникли проблеми з будь-якими з наданих команд.

Команди

- `in`: `Student.select().where(Student.columns.Major.in_(['English','Math']))`
- `and`, `or`, `not`: `Student.select().where(db.or_(Student.columns.Major == 'English', Student.columns.Pass = True))`
- `order by`: `Student.select().order_by(db.desc(Student.columns.Name))`
- `limit`: `Student.select().limit(3)`
- `sum`, `avg`, `count`, `min`, `max`: `db.select([db.func.sum(Student.columns.Id)])`
- `group by`: `db.select([db.func.sum(Student.columns.Id),Student.columns.Major]).group_by(Student.columns.Pass)`
- `distinct`: `db.select([Student.columns.Major.distinct()])`

Щоб дізнатися про інші функції та команди, перегляньте офіційну документацію SQL Statements and Expressions API.

Виведення результатів до Pandas DataFrame

Аналітики та вчені цінують роботу з pandas DataFrame. У цій частині ми навчимося, як перетворити результат запиту SQLAlchemy в pandas DataFrame.

Спочатку виконайте запит і збережіть результати.

```
query = Student.select().where(Student.columns.Major.in_(['English','Math']))
output = conn.execute(query)
results = output.fetchall()
```

Потім використовуйте функцію DataFrame і надайте результати SQL як аргумент. Нарешті, додайте назви стовпців за допомогою першого рядка результату `results[0]` та `.keys()`.

Зауваження: ви можете надати будь-який дійсний рядок для вилучення назв стовпців за допомогою `keys()`.

```
data = pd.DataFrame(results)
data.columns = results[0].keys()
data
```

Виведення до Pandas DataFrame

Аналіз даних з SQLAlchemy

У цій частині ми підключимо базу даних європейського футболу та виконаємо складні запити та візуалізуємо результати.

Підключення двох таблиць

Як завжди, ми підключимо базу даних за допомогою функцій `create_engine` та `connect`.

У нашому випадку ми будемо об'єднувати дві таблиці, тому нам потрібно створити два об'єкти таблиць: `division` та `match`.

```
engine = create_engine("sqlite:///european_database.sqlite")
conn = engine.connect()
metadata = db.MetaData()
division = db.Table('divisions', metadata, autoload=True, autoload_with=engine)
match = db.Table('matchs', metadata, autoload=True, autoload_with=engine)
```

Виконання складного запиту

Ми виберемо стовпці як з `division`, так і з `match`.
Приєднаємо їх за допомогою загального стовпця: `division.division` та `match.Div`.
Виберемо всі стовпці, де `division` - E1, а сезон - 2009.
Відсортуємо результат за стовпцем `HomeTeam`.

```
query = db.select([division,match]).\\
select_from(division.join(match,division.columns.division == match.columns.Div)).\\
where(db.and_(division.columns.division == "E1", match.columns.season == 2009 )).\\
order_by(match.columns.HomeTeam)
output = conn.execute(query)
results = output.fetchall()

data = pd.DataFrame(results)
data.columns = results[0].keys()
data
```

Аналітика даних з SQLAlchemy

Після виконання запиту ми перетворили результат у pandas DataFrame.

Обидві таблиці об'єднані, і результати показують тільки дивізію E1 для сезону 2009, відсортовану за стовпцем HomeTeam.

Візуалізація даних

Тепер, коли у нас є DataFrame, ми можемо візуалізувати результати у вигляді стовпчикової діаграми за допомогою Seaborn.

Ми:

- Встановимо тему "whitegrid".
- Змінимо розмір візуалізації на 15X6.
- Повернемо позначки осі X на 90 градусів.
- Встановимо палітру кольорів "pastels".
- Побудуємо стовпчикову діаграму "HomeTeam" проти "FTHG" з синім кольором.
- Побудуємо стовпчикову діаграму "HomeTeam" проти "FTAG" з червоним кольором.
- Відобразимо легенду у верхньому лівому куті.
- Приберемо позначки осей X та

Y.

- Приберемо зліва та знизу.

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="whitegrid")

f, ax = plt.subplots(figsize=(15, 6))
plt.xticks(rotation=90)
sns.set_color_codes("pastel")
sns.barplot(x="HomeTeam", y="FTHG", data=data,
            label="Home Team Goals", color="b")

sns.barplot(x="HomeTeam", y="FTAG", data=data,
            label="Away Team Goals", color="r")
ax.legend(ncol=2, loc="upper left", frameon=True)
ax.set(ylabel="", xlabel="")
sns.despine(left=True, bottom=True)
```

Візуалізація даних з SQLAlchemy

Збереження результатів у CSV

Після конвертації результату запиту в pandas DataFrame, ви можете просто використати функцію `.to_csv` з іменем файлу.

```
output = conn.execute("SELECT * FROM matchs WHERE HomeTeam LIKE 'Norwich'")
results = output.fetchall()

data = pd.DataFrame(results)
data.columns = results[0].keys()

data.to_csv("SQl_result.csv",index=False)
```

CSV-файл у SQL-таблицю

У цій частині ми конвертуємо CSV-файл з даними біржі в SQL-таблицю.

Спочатку підключіться до бази даних SQLite.

```
engine = create_engine("sqlite:///datacamp.sqlite")
```

Далі, імпортуйте CSV-файл, використовуючи функцію `read_csv`. У кінці використовуйте функцію `to_sql`, щоб зберегти pandas DataFrame як SQL-таблицю.

Зокрема, функція `to_sql` вимагає підключення та ім'я таблиці в якості аргументу. Ви також можете використовувати `if_exists` для заміни існуючої таблиці з такою самою назвою та `index` для скидання стовпця індексу.

```
df = pd.read_csv('Stock Exchange Data.csv')
df.to_sql(con=engine, name="Stock_price", if_exists='replace', index=False)
>>> 2222
```

Для перевірки результатів нам потрібно підключитися до бази даних і створити об'єкт таблиці.

```
conn = engine.connect()
metadata = db.MetaData()
stock = db.Table('Stock_price', metadata, autoload=True, autoload_with=engine)
```

Потім виконайте запит і відобразіть результати.

```
query = stock.select()
exe = conn.execute(query)
result = exe.fetchmany(5)
```

```
for r in result:
    print(r)
```

Як бачите, нам вдалося успішно перенести всі значення з CSV-файлу до SQL-таблиці.

Управління SQL-таблицею

Оновлення значень у таблиці

Оновити значення досить просто. Ми використовуватимемо функції `update`, `values` та `where`, щоб оновити конкретне значення в таблиці.

```
table.update().values(column_1=1, column_2=4,...).where(table.columns.column_5 >= 5)
```

У нашому випадку ми змінили значення "Pass" з False на True там, де ім'я студента - "Nisha".

```
Student = db.Table('Student', metadata, autoload=True, autoload_with=engine)
query = Student.update().values(Pass = True).where(Student.columns.Name == "Nisha")
results = conn.execute(query)
```

Для перевірки результатів виконаємо простий запит та відобразимо результати у вигляді pandas DataFrame.

```
output = conn.execute(Student.select()).fetchall()
data = pd.DataFrame(output)
data.columns = output[0].keys()
data
```

Ми успішно змінили значення "Pass" на True для імені студента "Nisha".

Оновлення значень в SQL

Видалення записів

Видалення рядків схоже на оновлення. Воно потребує функцій `delete` та `where`.

```
table.delete().where(table.columns.column_1 == 6)
```

У нашому випадку ми видаляємо запис студента з іменем "Ben".

```
Student = db.Table('Student', metadata, autoload=True, autoload_with=engine)
query = Student.delete().where(Student.columns.Name == "Ben")
```

```
results = conn.execute(query)
```

Для перевірки результатів виконаємо швидкий запит та відобразимо результати у вигляді dataframe. Як бачите, ми видалили рядок з ім'ям студента "Ben".

```
output = conn.execute(Student.select()).fetchall()
data = pd.DataFrame(output)
data.columns = output[0].keys()
data
```

Видалення значень

Видалення таблиць

Якщо ви використовуєте SQLite, видалення таблиці призведе до помилки "база даних заблокована". Чому? Тому що SQLite - це дуже легка версія, яка може виконувати тільки одну функцію одночасно. Зараз вона виконує запит SELECT. Нам потрібно закрити всі виконання перед видаленням таблиці.

```
results.close()
exe.close()
```

Після цього використовуйте функцію drop_all метаданих та виберіть об'єкт таблиці для видалення однієї таблиці. Ви також можете використовувати команду `Student.drop(engine)` для видалення однієї таблиці.

```
metadata.drop_all(engine, [Student], checkfirst=True)
```

Якщо ви не вказуєте жодну таблицю для функції drop_all, вона видалить всі таблиці в базі даних.

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

Видалення таблиць

Висновок

Урок з SQLAlchemy охоплює різні функції SQLAlchemy, від підключення бази даних до модифікації таблиць. Якщо вас цікавить навчання більш складних функцій SQLAlchemy,

спробуйте пройти інтерактивний курс "Введення в роботу з базами даних у Python". Ви дізнаєтеся основи реляційних баз даних, фільтрації, сортування та групування. Крім того, ви дізнаєтесь про більш складні функції SQLAlchemy для маніпулювання даними.

Якщо ви зіткнетесь з будь-якими проблемами при виконанні цього уроку, ви можете запустити вихідний код за допомогою робочого простору та порівняти його зі своїм кодом. Ви навіть можете скопіювати та вставити ці команди для тестування результатів на своєму комп'ютері.