

< Teach
Me
Skills />

Занятие 32.

SQL Alchemy

1

Преимущества

- **ORM (Object-Relational Mapping):** Одним из главных преимуществ SQLAlchemy является ORM-система. Она позволяет работать с базой данных, используя объекты и классы Python, что делает код более понятным, удобным для поддержки и масштабируемым. Разработчики могут думать о данных в терминах объектов, а не в терминах SQL.
- **Поддержка различных СУБД:** SQLAlchemy абстрагирует слой доступа к базам данных, что позволяет разработчикам работать с различными СУБД (например, MySQL, PostgreSQL, SQLite, Oracle) с использованием общего интерфейса. Это делает переход с одной базы данных на другую более простым и гибким.
- **Защита от SQL-инъекций:** SQLAlchemy предотвращает SQL-инъекции путем правильной обработки пользовательского ввода и параметризации запросов. Это повышает безопасность при работе с базами данных.

Преимущества

- **Поддержка транзакций:** SQLAlchemy позволяет управлять транзакциями в базе данных, обеспечивая целостность данных и возможность отката (rollback) в случае ошибки.
- **Гибкость и выразительность:** SQLAlchemy предоставляет богатый набор инструментов и возможностей для создания сложных запросов. Вы можете использовать как ORM-запросы, так и SQL-выражения, в зависимости от ваших потребностей.
- **Поддержка сессий:** SQLAlchemy предоставляет механизм сессий, который позволяет управлять состоянием объектов и их изменениями перед сохранением в базу данных. Это упрощает процесс отслеживания и сохранения изменений.

Преимущества

- **Миграции:** SQLAlchemy совместим с инструментами миграции, такими как Alembic, что упрощает управление изменениями схемы базы данных и обновление приложений.
- **Сообщество:** SQLAlchemy является популярным инструментом, и у него есть большое и активное сообщество разработчиков. Это означает, что вы можете легко найти поддержку, документацию и примеры кода в Интернете.
- **Использование в различных сферах:** SQLAlchemy может быть использован в различных типах проектов, от небольших веб-приложений до больших корпоративных систем.

Преимущества по сравнению с Django

- **Гибкость и независимость от фреймворка:** SQLAlchemy можно использовать в любом проекте на языке Python, независимо от того, используете вы Django или нет. Он предоставляет абстракцию для работы с различными СУБД, что позволяет вам легко переключаться между ними. В то время как Django ORM тесно связан с Django и предназначен для работы только внутри этого фреймворка.
- **Выразительность запросов:** SQLAlchemy предлагает более выразительный и мощный язык для создания запросов, особенно когда дело доходит до сложных запросов с использованием подзапросов и агрегаций. Это может быть особенно полезно в проектах с сложной структурой базы данных или большим объемом данных.

Преимущества по сравнению с Django

- **Поддержка сессий:** SQLAlchemy активно использует концепцию сессий, которая предоставляет контроль над состоянием объектов и изменениями перед сохранением в базу данных. Это может быть удобно для сложных транзакций или пакетной обработки данных.
- **Миграции:** SQLAlchemy имеет интеграцию с Alembic, что делает управление миграциями базы данных более гибким и независимым от фреймворка.
- **Абстракция от базы данных:** SQLAlchemy предоставляет мощные абстракции для работы с базами данных, что может быть полезно, если вы работаете с несколькими СУБД или хотите переключиться на другую базу данных в будущем.
- **Архитектурное разделение:** SQLAlchemy разделяет модель данных от способа выполнения запросов, что дает больше свободы и гибкости в проектировании архитектуры приложения.

2

Работа с моделями

Декларативный режим

это один из двух подходов к определению моделей данных в SQLAlchemy, который облегчает работу с базами данных, используя объектно-реляционное отображение (ORM). В SQLAlchemy существуют два режима определения моделей: классический и декларативный.

Декларативный режим позволяет определять модели данных с помощью обычных классов Python с использованием декларативных классов. Этот подход делает код более понятным и менее зависимым от сложных SQL-выражений. Основной идеей декларативного режима является создание классов, которые представляют таблицы в базе данных, и связь между классами и таблицами устанавливается автоматически на основе соглашений о именах.

Декларативный режим

```
class Student(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    first_name = db.Column(db.String(100), nullable=False)  
    last_name = db.Column(db.String(100), nullable=False)  
    birth_date = db.Column(db.Date)  
    male = db.Column(db.Boolean, nullable=False, default=True)  
    group_id = db.Column(db.Integer, db.ForeignKey('group.id'))  
  
    def __repr__(self):  
        return f"<Student {self.first_name} {self.last_name}>"
```

Классический режим

В классическом режиме определение моделей данных выполняется с помощью явного создания классов, представляющих таблицы в базе данных, и **определения маппинга между классами и таблицами**.

В классическом режиме создание классов и определение таблиц выполняются отдельно. Вы явно определяете классы Python для каждой таблицы, а затем связываете их с соответствующими таблицами в базе данных с помощью конструкций SQLAlchemy, таких как `Table` и `mapper`.

Классический режим

```
from sqlalchemy import Column, Integer, String, create_engine, MetaData, Table
from sqlalchemy.orm import mapper

# Создаем экземпляр класса MetaData
metadata = MetaData()

# Определяем класс Python для таблицы "users"
class User(object):
    def __init__(self, id, username, email):
        self.id = id
        self.username = username
        self.email = email
```

Классический режим

```
users_table = Table('users', metadata,
    Column('id', Integer, primary_key=True),
    Column('username', String(50)),
    Column('email', String(100))
)

# Связываем класс User с таблицей "users"
mapper(User, users_table)

# Создаем соединение с базой данных
engine = create_engine('sqlite:///example.db')

# Создаем таблицу в базе данных (если еще не существует)
metadata.create_all(engine)
```

Классический режим - преимущества

- **Отображение данных на объекты:** Классический режим позволяет создавать классы Python для таблиц в базе данных. Это делает возможным обращаться к данным как к объектам Python, что делает код более понятным и позволяет использовать привычные объектно-ориентированные методы и свойства.
- **Упрощение взаимодействия с базой данных:** Классический ORM скрывает сложности взаимодействия с базой данных и предоставляет более простой и понятный интерфейс для работы с данными. Разработчикам не нужно писать сложные SQL-запросы, так как ORM берет на себя задачу преобразования объектов в SQL и обратно.

Классический режим - преимущества

- **Увеличение безопасности:** Использование классического режима помогает избежать SQL-инъекций, поскольку ORM автоматически обрабатывает пользовательский ввод и параметризует SQL-запросы.
- **Улучшение сопровождаемости и расширяемости:** Использование классического режима делает код более организованным и легким для сопровождения. Модели данных могут быть определены один раз и могут быть использованы в разных частях приложения, что способствует повторному использованию кода.
- **Отделение бизнес-логики от базы данных:** Отделение моделей данных от кода взаимодействия с базой данных позволяет легко изменять структуру базы данных или заменять СУБД без значительных изменений в приложении.

Классический режим - когда нужен

- **Сложная структура базы данных:** Если ваша база данных имеет сложную структуру с нестандартными связями между таблицами, классический режим позволяет более точно контролировать маппинг между классами и таблицами. Вы можете определить каждую таблицу и ее отношения явно, что делает код более гибким при работе с комплексными базами данных.
- **Использование существующей базы данных:** Если у вас уже есть существующая база данных, которую вы хотите использовать в вашем Python-приложении, классический режим позволяет более гибко настроить отображение таблиц и сделать его соответствующим вашим требованиям.

Классический режим - когда нужен

- **Потребность в более точном контроле:** Классический режим предоставляет разработчикам более точный контроль над отображением данных и запросами к базе данных. Это может быть полезно, если вам требуется оптимизировать запросы или работать с особенностями конкретной СУБД.
- **Приемущества низкоуровневого доступа:** Классический режим позволяет использовать объекты SQLAlchemy напрямую для выполнения низкоуровневых операций, таких как вставка, обновление или удаление записей без обязательного использования ORM-запросов.

Классический режим - когда нужен

Хотя классический режим предоставляет больше гибкости и контроля, он также может потребовать больше кода в сравнении с декларативным режимом. Поэтому, при выборе между классическим и декларативным режимом, важно учитывать сложность вашей базы данных, требования к производительности и предпочтения разработчиков. В некоторых случаях, декларативный режим может быть более удобным и быстрым способом работы с базой данных.

3

Alembic

Alembic - преимущества

- **Миграции базы данных:** Основное преимущество Alembic - это его способность управлять миграциями баз данных. Миграции позволяют изменять структуру базы данных, добавлять новые таблицы, изменять или удалять существующие столбцы и т.д., не теряя данные или сталкиваясь с проблемами целостности. Миграции делают процесс обновления базы данных более гибким, безопасным и удобным.
- **Версионирование схемы:** Alembic позволяет версионировать схему базы данных. Это означает, что каждая миграция получает свой уникальный идентификатор, и система отслеживает, какие миграции были применены к базе данных. Это упрощает процесс обновления схемы на различных средах и управление разными версиями базы данных.

Alembic - преимущества

- **Автоматическое обнаружение изменений:** Alembic может автоматически обнаруживать изменения в модели данных SQLAlchemy и генерировать соответствующие миграции. Это упрощает процесс создания миграций и уменьшает вероятность ошибок вручную написанных миграций.
- **Интеграция с SQLAlchemy:** Alembic тесно интегрирован с SQLAlchemy, что позволяет легко использовать существующие модели данных для определения миграций. Он предоставляет мощный инструментарий для работы с базами данных, которые используют SQLAlchemy.

Alembic - преимущества

- **Поддержка различных СУБД:** Alembic поддерживает различные СУБД, такие как PostgreSQL, MySQL, SQLite и другие. Это позволяет использовать Alembic в разнообразных проектах, работающих с различными СУБД.
- **Понятный и простой синтаксис:** Alembic имеет простой и понятный синтаксис для создания миграций. Это упрощает процесс создания и применения миграций даже для разработчиков, не имеющих большого опыта работы с базами данных.
- **Откат миграций:** Alembic позволяет откатывать миграции, что позволяет возвращаться к предыдущим версиям схемы базы данных в случае необходимости.

Alembic

```
id_column = (
    sa.Column("id", sa.BigInteger().with_variant(sa.Integer, "sqlite"), primary_key=True)
        if engine == "sqlite" else sa.Column('id', sa.BigInteger(), nullable=False)
)

if engine != "oracle":
    op.create_table(
        'transaction',
        id_column,
        *liveness_transaction_columns,
        sa.PrimaryKeyConstraint('id')
    )

else:
    op.create_table(
        'transaction',
        *liveness_transaction_columns
    )
    conn.execute(text("""alter table transaction ADD (id NUMBER GENERATED BY DEFAULT AS IDENTITY)"""))
    conn.execute(text("""alter table transaction ADD CONSTRAINT pk_transaction PRIMARY KEY (id)"""))
```

4

Особенности работы

Что такое session

- В SQLAlchemy, **сессия (Session)** - это механизм, который обеспечивает контекст для работы с базой данных и управления состоянием объектов. Она представляет собой интерфейс для выполнения операций CRUD (Create, Read, Update, Delete) над данными в базе данных с использованием моделей данных, определенных с помощью SQLAlchemy.
- Когда вы создаете сессию, SQLAlchemy отслеживает объекты, с которыми вы работаете в этой сессии. Важно понимать, что объекты Python, представляющие данные в сессии, отражают состояние данных в базе данных. SQLAlchemy следит за изменениями, внесенными в объекты, и автоматически применяет изменения в базу данных в рамках транзакции, когда вы вызываете метод `commit()`.

Что такое session

Основные операции, которые вы можете выполнять с использованием сессии в SQLAlchemy:

- **Добавление объектов (CREATE):** Вы можете создавать новые объекты моделей, добавлять их в сессию с помощью метода add(), и при вызове commit(), объекты будут добавлены в базу данных.
- **Чтение объектов (READ):** С помощью методов query() и filter() вы можете создавать запросы к базе данных и извлекать объекты из базы данных.
- **Обновление объектов (UPDATE):** Если вы изменили свойства объектов в сессии, они будут автоматически обновлены в базе данных при вызове commit().
- **Удаление объектов (DELETE):** Вы можете удалить объекты из сессии с помощью метода delete(), и они будут удалены из базы данных при вызове commit()

pgbouncer

5

Что такое PgBouncer

это легковесный прокси-сервер для PostgreSQL, который предоставляет пул подключений к базе данных. Он служит промежуточным слоем между клиентскими приложениями и сервером PostgreSQL, оптимизируя и управляя соединениями к базе данных.

Зачем нужен PgBouncer и какие преимущества он предоставляет:

- **Эффективное использование ресурсов:** PgBouncer создает пул подключений к базе данных и переиспользует соединения между клиентскими запросами. Это позволяет снизить накладные расходы на установление новых соединений и улучшить производительность базы данных, особенно при большом количестве клиентов и запросов.

Что такое PgBouncer

- **Управление количеством соединений:** PgBouncer позволяет управлять количеством активных соединений к базе данных, что помогает предотвратить перегрузку сервера при большом числе клиентских запросов.
- **Разгрузка сервера базы данных:** Поскольку PgBouncer управляет пулом соединений, он может снизить нагрузку на сервер базы данных, особенно в условиях высокой нагрузки от множества клиентов.
- **Обработка временных отказов:** Если база данных временно недоступна или перегружена, PgBouncer может отказать новым клиентским подключениям и сохранить их в ожидании до тех пор, пока база данных снова не станет доступной.

Что такое PgBouncer

- **Улучшение безопасности:** PgBouncer позволяет настроить параметры аутентификации и авторизации для клиентских подключений, что улучшает безопасность системы.
- **Изоляция подключений:** Каждое клиентское приложение работает с собственным соединением из пула PgBouncer, что обеспечивает изоляцию запросов между приложениями и предотвращает взаимное влияние друг на друга.
- **Балансировка нагрузки:** PgBouncer может выполнять балансировку нагрузки между несколькими серверами PostgreSQL, что позволяет распределить нагрузку и обеспечить высокую доступность.

Что такое PgBouncer

В целом, PgBouncer - это важный инструмент для оптимизации работы с PostgreSQL базами данных, обеспечивая эффективное использование ресурсов, улучшение производительности, обработку временных отказов и повышение безопасности системы. Он часто используется в высоконагруженных средах, где необходимо обрабатывать большое количество запросов от множества клиентов.

6

Домашнее задание

Домашнее задание

Написать запросы:

- вычислить сколько у каждого пользователя идей
- у какого пользователя максимальное количество идей
- отфильтровать идеи, в которых есть слово "подставить свое"
- получить пользователей, у которых количество идей больше 5
- обновить имя пользователя с id=1
- удалить все идеи, в которых есть слово "подставить свое"

Спасибо за внимание!