Introducing Asynchronous SQLAlchemy

Sebastiaan Zeeff — EuroPython 2021
The ingredients of this talk

• Synchronous vs Asynchronous Input/Output

• A comparison between synchronous and asynchronous SQLAlchemy

• Managing implicit I/O: Eager vs Lazy loading
Personal introduction

• Sebastiaan Zeeff (35), The Netherlands

• Codesmith and developer for Ordina Pythoneers

• Owner of Python Discord
Synchronous vs Asynchronous Input/Output
Synchronous vs Asynchronous Input/Output
Synchronous vs Asynchronous Input/Output
Synchronous vs Asynchronous Input/Output
Synchronous vs Asynchronous Input/Output
Synchronous vs Asynchronous Input/Output
Synchronous vs Asynchronous Input/Output
Synchronous vs Asynchronous Input/Output
Synchronous vs Asynchronous Input/Output

Event loop
Synchronous vs Asynchronous Input/Output

Current task

Event loop
Synchronous vs Asynchronous Input/Output

Current task

Scheduled tasks

Event loop
Synchronous vs Asynchronous Input/Output

Current task

Scheduled tasks

Event loop
Synchronous vs Asynchronous Input/Output

Current task

Scheduled tasks

Event loop
Synchronous vs Asynchronous Input/Output

Current task

Scheduled tasks

Event loop
Synchronous vs Asynchronous Input/Output

Current task

Scheduled tasks

Event loop
Synchronous vs Asynchronous Input/Output

Current task

Scheduled tasks

Event loop
• Asyncio is about how we schedule input/put asynchronously in our SQLAlchemy app
Asyncio is about how we schedule input/put asynchronously in our SQLAlchemy app.

- The database queries and responses you generate will mostly stay the same (although...).
• Asyncio is about how we schedule input/put asynchronously in our SQLAlchemy app

• The database queries and responses you generate will mostly stay the same (although...).

• This means that you can (mostly) build your queries in the way you're already used to!
Asynchronous SQLAlchemy in Action

• I'll compare the sync and async versions of the same action

• Versions used:
  • Python 3.9.6
  • SQLAlchemy 1.4.22
    • The asyncio extension is currently considered to be in a **beta** release
  • asyncpg 0.23.0 (async adapter); Psycopg2 2.9.1 (sync adapter)
  • PostgreSQL 13.3 in an alpine-based Docker container
Setting up the engine

Python REPL 3.9.6
>>> import sqlalchemy
>>> engine = sqlalchemy.create_engine(
...     "postgresql+psycopg2://europython:europython@127.0.0.1:9876/europython",
...     echo=True,
...     future=True,
... )

asyncio REPL 3.9.6
>>> from sqlalchemy.ext import asyncio as asyncio_ext
>>> engine = asyncio_ext.create_async_engine(
...     "postgresql+asyncpg://europython:europython@127.0.0.1:9876/europython",
...     echo=True,
...     future=True,
... )
Executing a simple SQL statement

Python REPL 3.9.6

```python
>>> from sqlalchemy import create_engine
>>> engine = create_engine('mysql+mysqlconnector://user:password@localhost:3306/dbname')
>>> statement = sqlalchemy.text("SELECT 'Hello, EuroPython 2021!'")
>>> with engine.connect() as conn:
...     result = conn.execute(statement)
...     print(result.scalar())
Hello, EuroPython 2021!
```

asyncio REPL 3.9.6

```python
>>> from sqlalchemy import create_engine
>>> engine = create_engine('mysql+mysqlconnector://user:password@localhost:3306/dbname')
>>> statement = sqlalchemy.text("SELECT 'Hello, EuroPython 2021!'")
>>> async with engine.connect() as conn:
...     result = await conn.execute(statement)
...     print(result.scalar())
Hello, EuroPython 2021!
```
Eager vs Lazy loading

• That looks simple, what's the catch?
Eager vs Lazy loading

• That looks simple, what's the catch?

• You can't just rely on implicit I/O!
A simple ORM model

class Traveler(Base):
    """A model representing a traveler."""
    __tablename__ = "traveler"

    id = Column(Integer, primary_key=True)
    created_at = Column(DateTime, server_default=func.now())
    name = Column(String(128))
    age = Column(Integer)
Lazy loading vs Eager loading (server defaults)

```python
>>> sebastiaan = Traveler(name="Sebastiaan", age=35)
>>> with orm.Session(engine) as session:
...     with session.begin():
...         session.add(sebastiaan)
...     print("=
*=" * 64)
...     print("Sebastiaan was created at:", sebastiaan.created_at)
```
Lazy loading vs Eager loading (server defaults)

```python
>>> sebastiaan = Traveler(name="Sebastiaan", age=35)
>>> with orm.Session(engine) as session:
...     with session.begin():
...         session.add(sebastiaan)
...
...     print("=" * 64)
...     print("Sebastiaan was created at:", sebastiaan.created_at)
BEGIN (implicit)
INSERT INTO traveler (name, age) VALUES (%(name)s, %(age)s) RETURNING traveler.id
[generated in 0.00026s] {'name': 'Sebastiaan', 'age': 35}
COMMIT
================================================================
```
Lazy loading vs Eager loading (server defaults)

```python
>>> sebastiaan = Traveler(name="Sebastiaan", age=35)
>>> with orm.Session(engine) as session:
...     with session.begin():
...         session.add(sebastiaan)
...     print("=" * 64)
...     print("Sebastiaan was created at:", sebastiaan.created_at)
BEGIN (implicit)
INSERT INTO traveler (name, age) VALUES (%(name)s, %(age)s) RETURNING traveler.id
[generated in 0.00026s] {'name': 'Sebastiaan', 'age': 35}
COMMIT
================================================================
BEGIN (implicit)
SELECT traveler.id AS traveler_id, traveler.created_at AS traveler_created_at, traveler.name #...
WHERE traveler.id = %(pk_1)s
[generated in 0.00018s] {'pk_1': 1}
Sebastiaan was created at: 2021-07-28 09:42:06.723638
ROLLBACK
Lazy loading vs Eager loading (server defaults)

```python
>>> sebastiaan = Traveler(name="Sebastiaan", age=35)
>>> async_session = orm.sessionmaker(
...     engine,
...     expire_on_commit=False,
...     class_=asyncio_ext.AsyncSession
... )
>>> async with async_session() as session:
...     async with session.begin():
...         session.add(sebastiaan)
...     print("=")
...     print("Sebastiaan was created at:", sebastiaan.created_at)
```
Lazy loading vs Eager loading (server defaults)

```python
globals()['sebastiaan'] = Traveler(name='Sebastiaan', age=35)
globals()['async_session'] = orm.sessionmaker(
    ...   engine,
    ...   expire_on_commit=False,
    ...   class_=asyncio_ext.AsyncSession
    ...
)
async with async_session() as session:
    ...   async with session.begin():
    ...     session.add(sebastiaan)
    ...     print('= ' * 64)
    ...     print('Sebastiaan was created at:', sebastiaan.created_at)
BEGIN (implicit)
INSERT INTO traveler (name, age) VALUES (%s, %s) RETURNING traveler.id
[generated in 0.00049s] ('Sebastiaan', 35)
COMMIT

=================================================================
```
Lazy loading vs Eager loading (server defaults)

```python
>>> async with async_session() as session:
...     async with session.begin():
...         session.add(sebastiaan)
...     print("=" * 64)
...     print("Sebastiaan was created at:", sebastiaan.created_at)
...     
BEGIN (implicit)
INSERT INTO traveler (name, age) VALUES (%s, %s) RETURNING traveler.id
[generated in 0.00049s] ('Sebastiaan', 35)
COMMIT
================================================================
```
Lazy loading vs Eager loading (server defaults)

```python
>>> async with async_session() as session:
...     async with session.begin():
...         session.add(sebastiaan)
...     print("=" * 64)
...     print("Sebastiaan was created at:", sebastiaan.created_at)
BEGIN (implicit)
INSERT INTO traveler (name, age) VALUES (%s, %s) RETURNING traveler.id
[generated in 0.00049s] ('Sebastiaan', 35)
COMMIT
================================================================
BEGIN (implicit)
SELECT traveler.created_at AS traveler_created_at
FROM traveler
WHERE traveler.id = %s
[generated in 0.00061s] (2,)
ROLLBACK
Traceback (most recent call last):
... sqlalchemy.exc.MissingGreenlet: greenlet_spawn has not been called; can't call await_() here. Was IO attempted in an unexpected place? (Background on this error at: https://sqlalche.me/e/14/xd2s)```
class Traveler(Base):
    """A model representing a traveler."""

    __tablename__ = "traveler"

    id = Column(Integer, primary_key=True)
    created_at = Column(DateTime, server_default=func.now())
    name = Column(String(128))
    age = Column(Integer)
Working with ORM models

class Traveler(Base):
    """A model representing a traveler."""

    __tablename__ = "traveler"

    id = Column(Integer, primary_key=True)
    created_at = Column(DateTime, server_default=func.now())
    name = Column(String(128))
    age = Column(Integer)

    __mapper_args__ = {"eager_defaults": True}
Lazy loading vs Eager loading (server defaults)

```python
generate in 0.00040s] ('Sebastiaan', 35)
```

Sebastiaan was created at: 2021-07-28 10:15:32.675294
class Traveler(Base):
    """A model representing a traveler."""
    __tablename__ = "traveler"
    id = Column(Integer, primary_key=True)
    name = Column(String(128))
    age = Column(Integer)
    destination_id = Column(Integer, ForeignKey("country.id"))
    destination = orm.relationship("Country")

class Country(Base):
    """A model representing a country."""
    __tablename__ = "country"
    id = Column(Integer, primary_key=True)
    name = Column(String(128))
 Executing a simple SQL statement

```python
>>> sebastiaan = Traveler(name="Sebastiaan", age=35, destination=Country(name="Norway"))
>>> async with async_session() as session:
...    async with session.begin():
...        session.add(sebastiaan)
...```
Eager vs Lazy loading (relationships)

```python
>>> sebastiaan = Traveler(name="Sebastiaan", age=35, destination=Country(name="Norway"))
>>> async with async_session() as session:
...     async with session.begin():
...         session.add(sebastiaan)
...
>>> statement = sqlalchemy.select(Traveler).where(Traveler.name == "Sebastiaan")
```
Eager vs Lazy loading (relationships)

```python
>>> sebastiaan = Traveler(name="Sebastiaan", age=35, destination=Country(name="Norway"))
>>> async with async_session() as session:
...     async with session.begin():
...         session.add(sebastiaan)
...
>>> statement = sqlalchemy.select(Traveler).where(Traveler.name == "Sebastiaan")
>>> async with async_session() as session:
...     result = await session.execute(statement)
...     sebastiaan = result.scalar()
...```
Eager vs Lazy loading (relationships)

```python
>>> sebastiaan = Traveler(name="Sebastiaan", age=35, destination=Country(name="Norway"))
>>> async with async_session() as session:
...     async with session.begin():
...         session.add(sebastiaan)
...
>>> statement = sqlalchemy.select(Traveler).where(Traveler.name == "Sebastiaan")
>>> async with async_session() as session:
...     result = await session.execute(statement)
...     sebastiaan = result.scalar()
...
>>> print(sebastiaan)
Traveler(destination=NOT_LOADED, id=1, name='Sebastiaan', age=35, destination_id=1)
```
Eager vs Lazy loading (relationships)

```python
>>> sebastiaan = Traveler(name="Sebastiaan", age=35, destination=Country(name="Norway"))
>>> async with async_session() as session:
...     async with session.begin():
...         session.add(sebastiaan)
...
>>> statement = sqlalchemy.select(Traveler).where(Traveler.name == "Sebastiaan")
>>> async with async_session() as session:
...     result = await session.execute(statement)
...     sebastiaan = result.scalar()
...
>>> print(sebastiaan)
Traveler(destination=NOT_LOADED, id=1, name='Sebastiaan', age=35, destination_id=1)
>>> print(sebastiaan.destination)  # ERROR! Requires I/O to lazily load destination!
```
Eager vs Lazy loading (relationships)

```python
>>> statement = (...
... sqlalchemy.select(Traveler)...
... .where(Traveler.name == "Sebastiaan")...
... .options(orm.joinedload(Traveler.destination))
...)
```
Eager vs Lazy loading (relationships)

```python
>>> statement = 
...     sqlalchemy.select(Traveler)  
...     .where(Traveler.name == "Sebastiaan")  
...     .options(orm.joinedload(Traveler.destination))  
...
>>> async with async_session() as session:
...     result = await session.execute(statement)
...     sebastiaan = result.scalar()
...```
Eager vs Lazy loading (relationships)

```python
>>> statement = (
...     sqlalchemy.select(Traveler)
...     .where(Traveler.name == "Sebastiaan")
...     .options(orm.joinedload(Traveler.destination))
... )
```
Eager vs Lazy loading (relationships)

```python
>>> statement = (
...     sqlalchemy.select(Traveler)
...     .where(Traveler.name == "Sebastiaan")
...     .options(orm.joinedload(Traveler.destination))
... )

```async with async_session() as session:
...
   result = await session.execute(statement)
...
sebastiaan = result.scalar()
...
``` >>> print(sebastiaan)
Traveler(
    destination=Country(id=1, name='Norway'),
    id=1, name='Sebastiaan',
    age=35,
    destination_id=1
)
``` >>> print("Sebastiaan is traveling to:", sebastiaan.destination.name)
Sebastiaan is traveling to Norway
Using `run_sync`

- What if I want to run something that uses synchronous I/O functions?
Using `run_sync`

• Wat if I want to run something that uses synchronous I/O functions?

• You can use AsyncSession.run_sync!

```python
async with asyn_session() as session:
    ...  await session.run_sync(Base.metadata.create_all)
```
Using `run_sync`

- What if I want to run something that uses synchronous I/O functions?
- You can use AsyncSession.run_sync!
- For example, you can use the `MetaData.create_all` function like this:

```python
>>> async with async_session() as session:
...     await session.run_sync(Base.metadata.create_all)
```
Using `run_sync`

• Wat if I want to run something that uses synchronous I/O functions?

• You can use AsyncSession.run_sync!

• For example, you can use the `MetaData.create_all` function like this:

```python
>>> async with async_session() as session:
...     await session.run_sync(Base.metadata.create_all)
```

• This only works for asynchronous adapter functions!
Summary

• Most of your knowledge of SQLAlchemy is directly transferable

• You need to think carefully about operations that perform I/O

• You can still run synchronous database I/O functions with run_sync
Introducing Asychronous SQLAlchemy

Questions?

Sebastiaan Zeeff — EuroPython 2021