Querying Data with the SQLAlchemy ORM



Douglas Starnes
Author

linktr.ee/douglasstarnes

Connecting to a Database

Connect to a database with an engine

Create an engine with the `create_engine` function

A connection string identifies the database

Connecting to a Database

```
from sqlalchemy import create_engine
```



Connecting to a Database

```
from sqlalchemy import create_engine

engine = create_engine(“sqlite:///countdown.db”)
```



Connecting to a Database

```
from sqlalchemy import create_engine

engine = create_engine(“sqlite:///countdown.db”)

# for PostgreSQL
psql_engine = create_engine(
  “postgresql://myuser:mypassword@localhost:5432/mydatabase”
)
```



So, You've Got an Engine... What Now?

```
from sqlalchemy import create_engine
from sqlalchemy.orm import DeclarativeBase

engine = create_engine("sqlite:///countdown.db")

class Base(DeclarativeBase):
  pass

# imagine model classes deriving from Base somewhere around here

Base.metadata.create_all(bind=engine)
```



But I've Got Existing Tables!



No problem!



The `create_all` method only creates tables that don't exist



You can call it multiple times with worrying



Easy to add new model classes later on

Starting Over

```
from sqlalchemy import create_engine
from sqlalchemy.orm import DeclarativeBase

engine = create_engine("sqlite:///countdown.db")

class Base(DeclarativeBase):
  pass

# imagine model classes deriving from Base somewhere around here

Base.metadata.create_all(bind=engine) # create tables that don't exist
Base.metadata.drop_all(bind=engine) # remove tables that do exist
```



Sessions

Sessions are how you interact with a database

Queries are sent and results returned through sessions

A session is bound to a SQLAlchemy engine

Using Sessions

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session
```



Using Sessions

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

calendar = Calendar(name=“My Calendar”)
```



Using Sessions

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

calendar = Calendar(name=“My Calendar”)

sess = Session(bind=engine)
```



Using Sessions

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

calendar = Calendar(name=“My Calendar”)

sess = Session(bind=engine)

sess.add(calendar)
```



Using Sessions

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

calendar = Calendar(name=“My Calendar”)

sess = Session(bind=engine)

sess.add(calendar)

sess.commit()
```



Using Sessions

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine("sqlite:///countdown.db")

calendar = Calendar(name="My Calendar")

sess = Session(bind=engine)

sess.add(calendar)

sess.commit()

sess.close()
```



Using Sessions – A Better Approach

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  calendar = Calendar(name=“My Calendar”)
  sess.add(calendar)
  sess.commit()
```



Retrieving Objects



The API reads like SQL



The select() function acts like a “SELECT *” SQL statement



Chain the where() method to the select() function to narrow results



The scalars() method returns the ORM objects

Retrieving Objects

```
from sqlalchemy import select
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = select(Calendar)
```



Retrieving Objects

```
from sqlalchemy import select
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = select(Calendar)
  calendars = sess.scalars(query).all()
```



Retrieving a Single Object

```
from sqlalchemy import select
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = select(Calendar).where(Calendar.id == calendar_id)
```



Retrieving a Single Object

```
from sqlalchemy import select
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = select(Calendar).where(Calendar.id == calendar_id)
  calendar = sess.scalars(query).first()
```



Searching by Name

```
from sqlalchemy import select
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

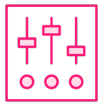
with Session(bind=engine) as sess:
  query = select(Calendar).where(Calendar.name.contains(name))
  calendar = sess.scalars(query).all()
```



Updating Objects



Use the `update()` function



Chain the `where()` method to get the specific object



Chain the `values()` method to update the attributes of the object



Since this changes the data, commit the session



Use the `rowcount` from the result to check if the update suceeded

Updates

```
from sqlalchemy import update
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = update(Calendar)
```



Updates

```
from sqlalchemy import update
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = update(Calendar).where(Calendar.id == calendar_id)
```



Updates

```
from sqlalchemy import update
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = update(Calendar).where(Calendar.id == calendar_id).values(name=name)
```



Updates

```
from sqlalchemy import update
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = update(Calendar).where(Calendar.id == calendar_id).values(name=name)
  result = sess.execute(query)
```



Updates

```
from sqlalchemy import update
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = update(Calendar).where(Calendar.id == calendar_id).values(name=name)
  result = sess.execute(query)
  sess.commit()
```



Updates

```
from sqlalchemy import update
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = update(Calendar).where(Calendar.id == calendar_id).values(name=name)
  result = sess.execute(query)
  sess.commit()

  if result.rowcount > 0:
    # update succeeded!
```



Deletes

```
from sqlalchemy import delete
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///countdown.db”)

with Session(bind=engine) as sess:
  query = delete(Calendar).where(Calendar.id == calendar_id)
  result = sess.execute(query)
  sess.commit()

  if result.rowcount > 0:
    # delete succeeded!
```



One-to-many Relationship Inserts

Reference the parent – the “one side” – by id

Reference the parent directly

Append to the parent’s child collection

One-to-many Data Model

```
from typing import List
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship

class Calendar(Base):
  # omitted for brevity
  events: Mapped[List["CalendarEvent"]] = relationship(
    back_populates="calendar",
    cascade="all",
  )

class CalendarEvent(Base):
  # omitted for brevity
  calendar_id: Mapped[int] = mapped_column(
    ForeignKey("calendar.id"), ondelete="CASCADE"
  )
  calendar: Mapped["Calendar"] = relationship(back_populates="events")
```



Reference Parent by Id

```
with Session(engine) as sess:
  new_event = CalendarEvent(
    name=event_name,
    # other attributes
    calendar_id=calendar_id
  )
  sess.add(new_event)
  sess.commit()
```



Reference Parent Directly

```
with Session(engine) as sess:
  query = select(Calendar).where(Calendar.id == calendar_id)
  calendar = sess.scalars(query).first()
  new_event = CalendarEvent(
    name=event_name,
    # other attributes
    calendar=calendar
  )
  sess.add(new_event)
  sess.commit()
```



Append to Parent Collection

```
with Session(engine) as sess:
  query = select(Calendar).where(Calendar.id == calendar_id)
  calendar = sess.scalars(query).first()
  new_event = CalendarEvent(
    name=event_name,
    # other attributes
  )
  calendar.events.append(new_event)
  sess.add(new_event)
  sess.commit()
```



Listing Child Objects

Iterate over the collection of child objects in the parent object

Query the child objects directly

Iterate Over the Child Collection

```
with Session(engine) as sess:
  query = select(Calendar).where(Calendar.id == calendar_id)
  calendar = sess.scalars(query).first()
  for event in calendar.events:
    print(f"{event.name} - {event.priority}")
```



Accessing Child Objects Directly

```
with Session(engine) as sess:
  query = select(CalendarEvent).where(CalendarEvent.calendar_id == calendar_id)
  if not private:
    query = query.where(CalendarEvent.private == False)
  query = query.where(CalendarEvent.priority >= min_priority)
  events = sess.scalars(query).all()
  for event in events:
    print(f"{event.name} - {event.priority}")
```



Deleting the One-to-many Way



The code is the same



Deleting a child object directly reacts the same way



Deleting a parent object also deletes the children (cascading delete)



SQLite does not, by default, enable foreign keys

Enable Foreign Keys in SQLite

```python
from sqlalchemy import event
from sqlalchemy.engine import Engine

@event.listens_for(Engine, "connect")
def set_sqlite_pragma(conn, _):
    cursor = conn.cursor()
    cursor.execute("PRAGMA foreign_keys=ON")
    cursor.close()
```



Updating Child Objects

Directly update the foreign key of the parent

Set the attribute referencing the parent

Updating Child Objects

```
class CalendarEvent(Base):
  # omitted for brevity
  calendar_id: Mapped[int] = mapped_column(
    ForeignKey("calendar.id"), ondelete="CASCADE"
  )
  calendar: Mapped["Calendar"] = relationship(back_populates="events")

with Session(engine) as sess:
  query = select(CalendarEvent).where(CalendarEvent.id == event_id)
  event = sess.scalars(query).first()
  event.calendar_id = new_calendar_id
  sess.commit()
```



Updating Child Objects

```
class CalendarEvent(Base):
  # omitted for brevity
  calendar_id: Mapped[int] = mapped_column(
    ForeignKey("calendar.id"), ondelete="CASCADE"
  )
  calendar: Mapped["Calendar"] = relationship(back_populates="events")

with Session(engine) as sess:
  query = select(CalendarEvent).where(CalendarEvent.id == event_id)
  event = sess.scalars(query).first()

  query = select(Calendar).where(Calendar.id == new_calendar_id)
  new_calendar = sess.scalars(query).first()

  event.calendar = new_calendar
  sess.commit()
```



Many-to-many Relationship Inserts

Insert an instance of the association object

Reference the foreign keys by id

Reference the related ORM objects directly

Attending an Event by Id

```
with Session(engine) as sess:
  attendee = EventAttendee(
    event_id=event_id,
    person_id=person_id,
    response_status=ResponseStatus.PENDING
  )
  session.add(attendee)
  session.commit()
```



Attending an Event by Reference

```
with Session(engine) as sess:
  event_query = select(CalendarEvent).where(CalendarEvent.id == event_id)
  event = session.scalars(event_query).first()
  # same for person
  attendee = EventAttendee(
    event=event,
    person=person,
    response_status=ResponseStatus.PENDING
  )
  session.add(attendee)
  session.commit()
```



Listing a Calendar's Events

```
with Session(engine) as sess:
  calendar_query = select(Calendar).where(Calendar.id == calendar_id)
  calendar = session.scalars(calendar_query).first()

  for event in calendar.events:
    # display calendar information
```



Listing a Calendar's Events (and the Event's Attendees)

```
with Session(engine) as sess:
  calendar_query = select(Calendar).where(Calendar.id == calendar_id)
  calendar = session.scalars(calendar_query).first()

  for event in calendar.events:
    # display calendar information
    for attendee in event.attendees:
      print(f"{attendee.person.name}")
```



Association Object Issues



To get the attendees for a `CalendarEvent`, must drill down into `EventAttendee` objects



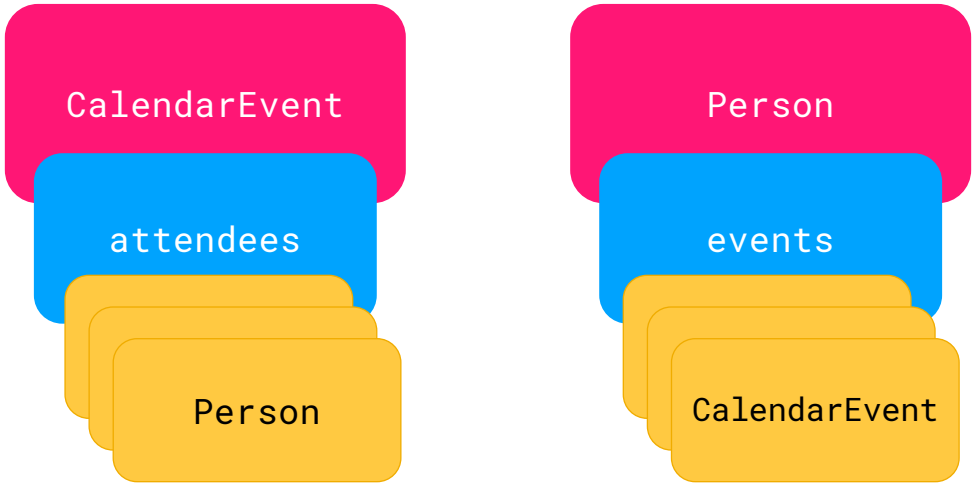
Same issue retrieving the events a `Person` is attending

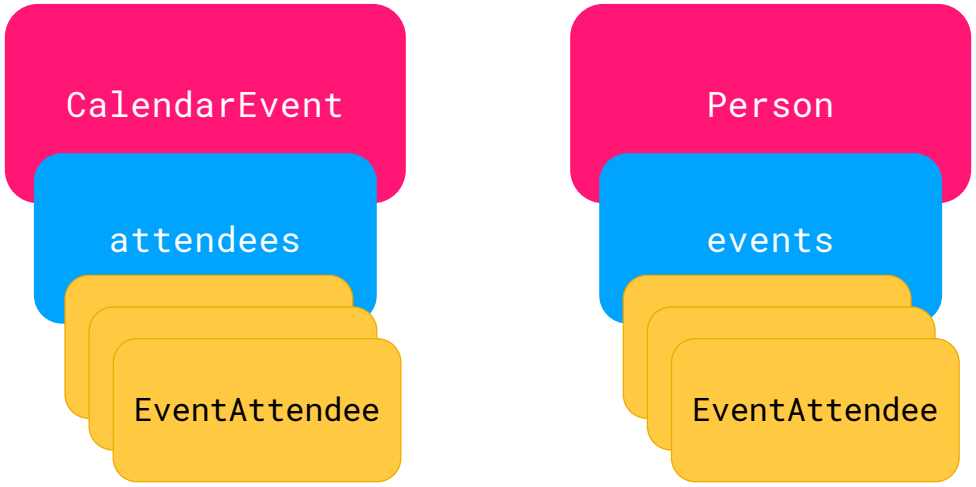


The additional layer is not a lot of extra code to type



It could create some cognitive dissonance


CalendarEvent
attendees
Person
Person
events
CalendarEvent


CalendarEvent
attendees
EventAttendee
Person
events
EventAttendee

Association Proxy

Provides direct access to the related objects in a many-to-many relationship

No need to drill down into the association object

Lose access to extra attributes on the association object (ie. `ResponseStatus`)

Creating an Association Proxy

```python
from sqlalchemy.ext.associationproxy import association_proxy, AssociationProxy

class CalendarEvent(Base):
  # ...
  attendees: Mapped[List["EventAttendee"]] = relationship(
      back_populates="event",
      cascade="all",
  )

  attendees_proxy: AssociationProxy = association_proxy(
      "attendees",
      "person",
      creator=lambda person: EventAttendee(person=person),
  )
```



Creating an Association Proxy

```python
from sqlalchemy.ext.associationproxy import association_proxy, AssociationProxy

class Person(Base):
  # ...
  events: Mapped[List["EventAttendee"]] = relationship(
      back_populates="person",
      cascade="all",
  )

  events_proxy: AssociationProxy = association_proxy(
      "events",
      "event",
      creator=lambda person: EventAttendee(event=event),
  )
```



Listing Events and Attendees (with Association Proxy)

```
with Session(engine) as sess:
  calendar_query = select(Calendar).where(Calendar.id == calendar_id)
  calendar = session.scalars(calendar_query).first()

  for event in calendar.events:
    # display calendar information
    for attendee in event.attendees_proxy:
      print(f"{attendee.name}")
```



Naming Conventions Count!

```
with Session(engine) as sess:
  calendar_query = select(Calendar).where(Calendar.id == calendar_id)
  calendar = session.scalars(calendar_query).first()

  for event in calendar.events:
    # display calendar information
    for person in event.people:
      print(f"{person.name}")
```



Updates and Deletes

Not supported with association proxies

Must access the association objects instead

Similar to performing updates and deletes with other relationships