Data Modeling with the SQLAlchemy ORM



Douglas Starnes
Author

linktr.ee/douglasstarnes

SQL: Structured Query Language

The defacto choice for interacting with relational databases

We need SQL, but do we really need to interact with it?

We can use a high-level language, such as Python, and abstract SQL implementation details

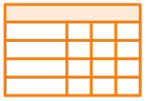
SQLAlchemy



SQLAlchemy offers an ORM, or Object Relational Mapper



Interact with relational databases using only Python code



Create model classes in Python that are mapped to database tables



Attributes of model classes are mapped to database table columns

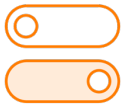
Leveling the Playing Field



Pythonistas commonly use SQLite for dev / PostgreSQL for prod



SQLite does not support a Boolean type – repurposes 0 and 1



PostgreSQL supports a Boolean type – values TRUE and FALSE



SQLAlchemy eliminates the need for two different codebases

SQLAlchemy still uses relational database concepts.



SQLAlchemy Core API



Core API is aligned with SQL, the ORM abstracts it



There is a select() function for SELECT statements, an insert() function for INSERT statements and so on



You get more control and, in some cases, better performance



This is at the cost of greater complexity in your code

Primary Key

Event Name

Event Date

Priority

Private

The CalendarEvent entity

Install SQLAlchemy

```
$ pip install sqlalchemy

Collecting sqlalchemy-2.0
Installing collected packages: sqlalchemy
Successfully installed sqlalchemy-2.0
```



Creating a Model Class

```
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
  pass
```



Creating a Model Class

```
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
```



Creating a Model Class

```
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
```



Creating a Model Class

```
from sqlalchemy.orm import DeclarativeBase, Mapped

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
```



Creating a Model Class

```
from sqlalchemy.orm import DeclarativeBase, Mapped

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int]
```



Creating a Model Class

```python
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column()
```



Creating a Model Class

```
from sqlalchemy import Integer
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer)
```



Creating a Model Class

```
from sqlalchemy import Integer
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True)
```



Creating a Model Class

```
from sqlalchemy import Integer
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
```



Creating a Model Class

```
from sqlalchemy import Integer
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str]
```



Creating a Model Class

```
from sqlalchemy import Integer, String
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str] = mapped_column(String(100))
```



Creating a Model Class

```
from sqlalchemy import Integer, String
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str] = mapped_column(String(100), nullable=False)
```



Creating a Model Class

```
from datetime import date
from sqlalchemy import Integer, String
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str] = mapped_column(String(100), nullable=False)
  event_date: Mapped[date]
```



Creating a Model Class

```
from datetime import date
from sqlalchemy import Integer, String, Date
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str] = mapped_column(String(100), nullable=False)
  event_date: Mapped[date] = mapped_column(Date)
```



Creating a Model Class

```
from datetime import date
from sqlalchemy import Integer, String, Date
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str] = mapped_column(String(100), nullable=False)
  event_date: Mapped[date] = mapped_column(Date, default=date.today)
```



Creating a Model Class

```
from datetime import date
from sqlalchemy import Integer, String, Date
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str] = mapped_column(String(100), nullable=False)
  event_date: Mapped[date] = mapped_column(Date, default=date.today)
  priority: Mapped[int]
```



Creating a Model Class

```
from datetime import date
from sqlalchemy import Integer, String, Date
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str] = mapped_column(String(100), nullable=False)
  event_date: Mapped[date] = mapped_column(Date, default=date.today)
  priority: Mapped[int] = mapped_column(Integer)
```



Creating a Model Class

```
from datetime import date
from sqlalchemy import Integer, String, Date, CheckConstraint
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str] = mapped_column(String(100), nullable=False)
  event_date: Mapped[date] = mapped_column(Date, default=date.today)
  priority: Mapped[int] = mapped_column(
    Integer,
    CheckConstraint(“priority >= 1 and priority <= 10”),
  )
```



Creating a Model Class

```
from datetime import date
from sqlalchemy import Integer, String, Date, CheckConstraint
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str] = mapped_column(String(100), nullable=False)
  event_date: Mapped[date] = mapped_column(Date, default=date.today)
  priority: Mapped[int] = mapped_column(
    Integer,
    CheckConstraint(“priority >= 1 and priority <= 10”),
    default=5,
  )
```



Creating a Model Class

```
from datetime import date
from sqlalchemy import Integer, String, Date, CheckConstraint, Boolean
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
  pass

class CalendarEvent(Base):
  __tablename__ = “calendar_event”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  event_name: Mapped[str] = mapped_column(String(100), nullable=False)
  event_date: Mapped[date] = mapped_column(Date, default=date.today)
  priority: Mapped[int] = mapped_column(
    Integer,
    CheckConstraint(“priority >= 1 and priority <= 10”),
    default=5,
  )
  is_private: Mapped[int] = mapped_column(Boolean, default=False)
```



One-to-many Relationships

```
class Calendar(Base):

class CalendarEvent(Base):
  # omitted for brevity
```



One-to-many Relationships

```
class Calendar(Base):
  __tablename__ = “calendar”

class CalendarEvent(Base):
  # omitted for brevity
```



One-to-many Relationships

```
class Calendar(Base):
  __tablename__ = “calendar”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)

class CalendarEvent(Base):
  # omitted for brevity
```



One-to-many Relationships

```
class Calendar(Base):
  __tablename__ = “calendar”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  calendar_name: Mapped[str] = mapped_column(String(100), nullable=False)

class CalendarEvent(Base):
  # omitted for brevity
```



One-to-many Relationships

```
from typing import List

class Calendar(Base):
  __tablename__ = “calendar”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  calendar_name: Mapped[str] = mapped_column(String(100), nullable=False)

  events: Mapped[List[]]

class CalendarEvent(Base):
  # omitted for brevity
```



One-to-many Relationships

```
from typing import List

class Calendar(Base):
  __tablename__ = “calendar”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  calendar_name: Mapped[str] = mapped_column(String(100), nullable=False)

  events: Mapped[List[“CalendarEvent”]]

class CalendarEvent(Base):
  # omitted for brevity
```



One-to-many Relationships

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

class Calendar(Base):
  __tablename__ = “calendar”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  calendar_name: Mapped[str] = mapped_column(String(100), nullable=False)

  events: Mapped[List[“CalendarEvent”]] = relationship()

class CalendarEvent(Base):
  # omitted for brevity
```



One-to-many Relationships

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

class Calendar(Base):
  __tablename__ = “calendar”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  calendar_name: Mapped[str] = mapped_column(String(100), nullable=False)

  events: Mapped[List[“CalendarEvent”]] = relationship(
    back_populates=“calendar”,
  )

class CalendarEvent(Base):
  # omitted for brevity
```



One-to-many Relationships

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

class Calendar(Base):
  __tablename__ = “calendar”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  calendar_name: Mapped[str] = mapped_column(String(100), nullable=False)

  events: Mapped[List[“CalendarEvent”]] = relationship(
    back_populates=“calendar”,
    cascade=“all”,
  )

class CalendarEvent(Base):
  # omitted for brevity
```



One-to-many Relationships

```
from typing import List
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]
```



One-to-many Relationships

```
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”),
  )
```



One-to-many Relationships

```
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”
  )
```



One-to-many Relationships

```
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"]
```



One-to-many Relationships

```
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")
```



Many-to-many Relationships



Many-to-many Relationships

```
class Person(Base):
```



Many-to-many Relationships

```
class Person(Base):
  __tablename__ = “person”
```



Many-to-many Relationships

```
class Person(Base):
  __tablename__ = “person”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
```



Many-to-many Relationships

```
class Person(Base):
  __tablename__ = “person”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  name: Mapped[str] = mapped_column(String(100), nullable=False)
```



Many-to-many Relationships

```
class Person(Base):
  __tablename__ = “person”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  name: Mapped[str] = mapped_column(String(100), nullable=False)
  email: Mapped[str] = mapped_column(String(100), nullable=False)
```



Many-to-many Relationships

```
class Person(Base):
  __tablename__ = “person”
  id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
  name: Mapped[str] = mapped_column(String(100), nullable=False)
  email: Mapped[str] = mapped_column(String(100), nullable=False, unique=True)
```



Many-to-many Relationships

```
class EventAttendee(Base):  # an association table
```



Many-to-many Relationships

```
class EventAttendee(Base):  # an association table
  __tablename__ = “event_attendee”
```



Many-to-many Relationships

```
class EventAttendee(Base):  # an association table
  __tablename__ = “event_attendee”
  event_id: Mapped[int] = mapped_column(
    ForeignKey(“calendar_event.id”, ondelete=“CASCADE”), primary_key=True
  )
  person_id: Mapped[int] = mapped_column(
    ForeignKey(“person.id”, ondelete=“CASCADE”), primary_key=True
  )
```



Many-to-many Relationships

```
class EventAttendee(Base):  # an association table
  __tablename__ = “event_attendee”
  event_id: Mapped[int] = mapped_column(
    ForeignKey(“calendar_event.id”, ondelete=“CASCADE”), primary_key=True
  )
  person_id: Mapped[int] = mapped_column(
    ForeignKey(“person.id”, ondelete=“CASCADE”), primary_key=True
  )
  event: Mapped[“CalendarEvent”]
```



Many-to-many Relationships

```
class EventAttendee(Base):  # an association table
  __tablename__ = “event_attendee”
  event_id: Mapped[int] = mapped_column(
    ForeignKey(“calendar_event.id”, ondelete=“CASCADE”), primary_key=True
  )
  person_id: Mapped[int] = mapped_column(
    ForeignKey(“person.id”, ondelete=“CASCADE”), primary_key=True
  )
  event: Mapped[“CalendarEvent”] = relationship(back_populates=“attendees”)
```



Many-to-many Relationships

```
class EventAttendee(Base):  # an association table
  __tablename__ = “event_attendee”
  event_id: Mapped[int] = mapped_column(
    ForeignKey(“calendar_event.id”, ondelete=“CASCADE”), primary_key=True
  )
  person_id: Mapped[int] = mapped_column(
    ForeignKey(“person.id”, ondelete=“CASCADE”), primary_key=True
  )
  event: Mapped[“CalendarEvent”] = relationship(back_populates=“attendees”)
  person: Mapped[“Person”] = relationship(back_populates=“events”)
```



Many-to-many Relationships

```
class EventAttendee(Base):
  # omitted for brevity
  event: Mapped[“CalendarEvent”] = relationship(back_populates=“attendees”)
  person: Mapped[“Person”] = relationship(back_populates=“events”)

class CalendarEvent(Base):
  # omitted for brevity
  attendees: Mapped[List[“EventAttendee”]] = relationship(
    back_populates=“event”,
    cascade=“all”,
  )
```



Many-to-many Relationships

```python
class EventAttendee(Base):
  # omitted for brevity
  event: Mapped[“CalendarEvent”] = relationship(back_populates=“attendees”)
  person: Mapped[“Person”] = relationship(back_populates=“events”)

class CalendarEvent(Base):
  # omitted for brevity
  attendees: Mapped[List[“EventAttendee”]] = relationship(
    back_populates=“event”,
    cascade=“all”,
  )

class Person(Base):
  # omitted for brevity
  attendees: Mapped[List[“EventAttendee”]] = relationship(
    back_populates=“event”,
    cascade=“all”,
  )
```



Explicitly Creating the Association Table

```
from sqlalchemy import Table, Column, Integer, ForeignKey

event_attendee = Table(
  “event_attendee”,
  Base.metadata,
  Column(
    “event_id”,
    Integer,
    ForeignKey(“calendar_event.id”, ondelete=“CASCADE”),
    primary_key=True,
  ),
  Column(
    “person_id”,
    Integer,
    ForeignKey(“person.id”, ondelete=“CASCADE”),
    primary_key=True,
  ),
)
```



Adding Attendee Status

I might be there, or I might not ...



Adding Attendee Status

```
from enum import Enum
```



Adding Attendee Status

```
from enum import Enum

class ResponseStatus(Enum):
  ACCEPTED = “accepted”
  DECLINED = “declined”
  PENDING = “pending”
  TENTATIVE = “tentative”
```



Adding Attendee Status

```
from enum import Enum

class ResponseStatus(Enum):
  ACCEPTED = “accepted”
  DECLINED = “declined”
  PENDING = “pending”
  TENTATIVE = “tentative”

class EventAttendee(Base):
  # omitted for brevity
```



Adding Attendee Status

```
from enum import Enum
from sqlalchemy import Enum

class ResponseStatus(Enum):
  ACCEPTED = “accepted”
  DECLINED = “declined”
  PENDING = “pending”
  TENTATIVE = “tentative”

class EventAttendee(Base):
  # omitted for brevity
  response_status: Mapped[ResponseStatus] = mapped_column(
    Enum(ResponseStatus)
  )
```



Adding Attendee Status

```
from enum import Enum
from sqlalchemy import Enum as SAEnum

class ResponseStatus(Enum):
  ACCEPTED = “accepted”
  DECLINED = “declined”
  PENDING = “pending”
  TENTATIVE = “tentative”

class EventAttendee(Base):
  # omitted for brevity
  response_status: Mapped[ResponseStatus] = mapped_column(
    SAEnum(ResponseStatus)
  )
```



Adding Attendee Status

```
from enum import Enum
from sqlalchemy import Enum as SAEnum

class ResponseStatus(Enum):
  ACCEPTED = “accepted”
  DECLINED = “declined”
  PENDING = “pending”
  TENTATIVE = “tentative”

class EventAttendee(Base):
  # omitted for brevity
  response_status: Mapped[ResponseStatus] = mapped_column(
    SAEnum(ResponseStatus),
    default=ResponseStatus.PENDING,
    nullable=False,
  )
```

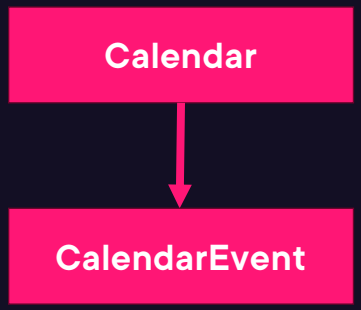


Countdown Application Data Model

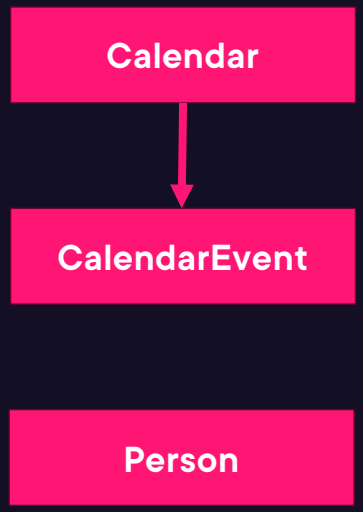
Calendar



Countdown Application Data Model



Countdown Application Data Model



Countdown Application Data Model

