- [Home](#)
- [About](#)
- [Buy the Book](#)
- [Contact the Author](#)
- [Follow the Blog](#)
- [License](#)

Sun 1 Jul 2012

A Simple SqlAlchemy 0.7 / 0.8 Tutorial

Posted by Mike under [Python](#), [SqlAlchemy](#)
[11 Comments](#)

A couple years ago I wrote a rather [flawed tutorial](#) about SQLAlchemy. I decided it was about time for me to re-do that tutorial from scratch and hopefully do a better job of it this time around. Since I'm a music nut, we'll be creating a simple database to store album information. A database isn't a database without some relationships, so we'll create two tables and connect them. Here are a few other things we'll be learning:

- Adding data to each table
- Modifying data
- Deleting data
- Basic queries

But first we need to actually make the database, so that's where we'll begin our journey. Note that SQLAlchemy is a 3rd party package, so you'll need to [install](#) it if you want to follow along.

How to Create a Database

Creating a database with SQLAlchemy is really easy. They have gone completely with their [Declarative](#) method of creating databases now, so we won't be covering the old school method. You can read the code here and then we'll explain it following the listing. If you want a way to view your SQLite database, I would recommend the SQLite Manager [plugin](#) for Firefox. Or you could use the [simple wxPython application](#) that I created a month ago.

```python
# table_def.py
from sqlalchemy import create_engine, ForeignKey
from sqlalchemy import Column, Date, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, backref

engine = create_engine('sqlite:///mymusic.db', echo=True)
Base = declarative_base()

########################################################################
class Artist(Base):
    """"""
    __tablename__ = "artists"
```

```python
    id = Column(Integer, primary_key=True)
    name = Column(String)

    #----------------------------------------------------------------------
    def __init__(self, name):
        """"""
        self.name = name

########################################################################
class Album(Base):
    """"""
    __tablename__ = "albums"

    id = Column(Integer, primary_key=True)
    title = Column(String)
    release_date = Column(Date)
    publisher = Column(String)
    media_type = Column(String)

    artist_id = Column(Integer, ForeignKey("artists.id"))
    artist = relationship("Artist", backref=backref("albums", order_by=id))

    #----------------------------------------------------------------------
    def __init__(self, title, release_date, publisher, media_type):
        """"""
        self.title = title
        self.release_date = release_date
        self.publisher = publisher
        self.media_type = media_type

# create tables
Base.metadata.create_all(engine)
```

If you run this code then you should see the following output sent to stdout:

```
2012-06-27 16:34:24,479 INFO sqlalchemy.engine.base.Engine PRAGMA
table_info("artists")
2012-06-27 16:34:24,479 INFO sqlalchemy.engine.base.Engine ()
2012-06-27 16:34:24,480 INFO sqlalchemy.engine.base.Engine PRAGMA table_info("albums")
2012-06-27 16:34:24,480 INFO sqlalchemy.engine.base.Engine ()
2012-06-27 16:34:24,480 INFO sqlalchemy.engine.base.Engine
CREATE TABLE artists (
id INTEGER NOT NULL,
name VARCHAR,
PRIMARY KEY (id)
)

2012-06-27 16:34:24,483 INFO sqlalchemy.engine.base.Engine ()
2012-06-27 16:34:24,558 INFO sqlalchemy.engine.base.Engine COMMIT
2012-06-27 16:34:24,559 INFO sqlalchemy.engine.base.Engine
CREATE TABLE albums (
id INTEGER NOT NULL,
title VARCHAR,
release_date DATE,
publisher VARCHAR,
media_type VARCHAR,
artist_id INTEGER,
PRIMARY KEY (id),
```

```
FOREIGN KEY(artist_id) REFERENCES artists (id)
)

2012-06-27 16:34:24,559 INFO sqlalchemy.engine.base.Engine ()
2012-06-27 16:34:24,615 INFO sqlalchemy.engine.base.Engine COMMIT
```

Why did this happen? Because when we created the engine object, we set its **echo** parameter to True. The [engine](#) is where the database connection information is and it has all the DBAPI stuff in it that makes communication with your database possible. You'll note that we're creating a SQLite database. Ever since Python 2.5, SQLite has been supported by the language. If you want to connect to some other database, then you'll need to edit the connection string. Just in case you're confused about what we're talking about, here is the code in question:

```
engine = create_engine('sqlite:///mymusic.db', echo=True)
```

The string, **'sqlite:///mymusic.db'**, is our connection string. Next we create an instance of the declarative base, which is what we'll be basing our table classes on. Next we have two classes, **Artist** and **Album** that define what our database tables will look like. You'll notice that we have Columns, but no column names. SQLAlchemy actually used the variable names as the column names unless you specifically specify one in the Column definition. You'll note that we are using an "id" Integer field as our primary key in both classes. This field will auto-increment. The other columns are pretty self-explanatory until you get to the ForeignKey. Here you'll see that we're tying the **artist_id** to the id in the **Artist** table. The **relationship** directive tells SQLAlchemy to tie the Album class/table to the Artist table. Due to the way we set up the ForeignKey, the relationship directive tells SQLAlchemy that this is a **many-to-one** relationship, which is what we want. Many albums to one artist. You can read more about table relationships [here](#).

The last line of the script will create the tables in the database. If you run this script multiple times, it won't do anything new after the first time as the tables are already created. You could add another table though and then it would create the new one.

How to Insert / Add Data to Your Tables

A database isn't very useful unless it has some data in it. In this section we'll show you how to connect to your database and add some data to the two tables. It's much easier to take a look at some code and then explain it, so let's do that!

```python
import datetime
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from table_def import Album, Artist

engine = create_engine('sqlite:///mymusic.db', echo=True)

# create a Session
Session = sessionmaker(bind=engine)
session = Session()

# Create an artist
new_artist = Artist("Newsboys")
new_artist.albums = [Album("Read All About It",
                          datetime.date(1988,12,01),
                          "Refuge", "CD")]
```

```python
# add more albums
more_albums = [Album("Hell Is for Wimps",
                        datetime.date(1990,07,31),
                        "Star Song", "CD"),
                Album("Love Liberty Disco",
                        datetime.date(1999,11,16),
                        "Sparrow", "CD"),
                Album("Thrive",
                        datetime.date(2002,03,26),
                        "Sparrow", "CD")]
new_artist.albums.extend(more_albums)

# Add the record to the session object
session.add(new_artist)
# commit the record the database
session.commit()

# Add several artists
session.add_all([
    Artist("MXPX"),
    Artist("Kutless"),
    Artist("Thousand Foot Krutch")
    ])
session.commit()
```

First we need to import our table definitions from the previous script. Then we connect to the database with our engine and create something new, the Session object. The session is our handle to the database and let's us interact with it. We use it to create, modify, and delete records and we also use sessions to query the database. Next we create an Artist object and add an album. You'll note that to add an album, you just create a list of Album objects and set the artist object's "albums" property to that list or you can extend it, as you see in the second part of the example. At the end of the script, we add three additional Artists using the **add_all**. As you have probably noticed by now, you need to use the session object's **commit** method to write the data to the database. Now it's time to turn our attention to modifying the data.

A Note about __init__: As some of my astute readers pointed out, you actually do not need the **__init__** constructors for the table definitions. I left them in there because the official documentation was still using them and I didn't realize I could leave them out. Anyway, if you leave the __init__ out of your declarative table definition, then you'll need to use keyword arguments when creating a record. For example, you would do the following rather than what was shown in the previous example:

```python
new_artist = Artist(name="Newsboys")
new_artist.albums = [Album(title="Read All About It",
                            release_date=datetime.date(1988,12,01),
                            publisher="Refuge", media_type="CD")]
```

How to Modify Records with SQLAlchemy

What happens if you saved some bad data. For example, you typed your favorite album's title incorrectly or you got the release date wrong for that fan edition you own? Well you need to learn how to modify that record! This will actually be our jumping off point into learning SQLAlchemy queries as you need to find the record that you need to change and that means you need to write a query for it. Here's some code that shows us the way:

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from table_def import Album, Artist

engine = create_engine('sqlite:///mymusic.db', echo=True)

# create a Session
Session = sessionmaker(bind=engine)
session = Session()

# querying for a record in the Artist table
res = session.query(Artist).filter(Artist.name=="Kutless").first()
print res.name

# changing the name
res.name = "Beach Boys"
session.commit()

# editing Album data
artist, album = session.query(Artist, Album).filter(Artist.id==Album.artist_id).filter
album.title = "Step Up to the Microphone"
session.commit()
```

Our first query goes out and looks up an Artist by name using the **filter** method. The ".first()" tells SQLAlchemy that we only want the first result. We could have used ".all()" if we thought there would be multiple results and we wanted all of them. Anyway, this query returns an Artist object that we can manipulate. As you can see, we changed the **name** from "Kutless" to "Beach Boys" and then committed out changes.

Querying a joined table is a little bit more complicated. This time we wrote a query that queries both our tables. It filters using the Artist id AND the Album title. It returns two objects: an artist and an album. Once we have those, we can easily change the title for the album. Wasn't that easy? At this point, we should probably note that if we add stuff to the session erroneously, we can rollback our changes/adds/deletes by using **session.rollback()**. Speaking of deleting, let's tackle that subject!

How to Delete Records in SQLAlchemy

Sometimes you just have to delete a record. Whether it's because you're involved in a cover-up or because you don't want people to know about your love of Britney Spears music, you just have to get rid of the evidence. In this section, we'll show you how to do just that! Fortunately for us, SQLAlchemy makes deleting records really easy. Just take a look at the following code!

```python
# deleting_data.py
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from table_def import Album, Artist

engine = create_engine('sqlite:///mymusic.db', echo=True)

# create a Session
Session = sessionmaker(bind=engine)
session = Session()

res = session.query(Artist).filter(Artist.name=="MXPX").first()
```

```
session.delete(res)
session.commit()
```

As you can see, all you had to do was create another SQL query to find the record you want to delete and then call **session.delete(res)**. In this case, we deleted our MXPX record. Some people think punk will never die, but they must not know any DBAs! We've already seen queries in action, but let's take a closer look and see if we can learn anything new.

The Basic SQL Queries of SQLAlchemy

SQLAlchemy provides all the queries you'll probably ever need. We'll be spending a little time just looking at a few of the basic ones though, such as a couple simple SELECTs, a JOINed SELECT and using the LIKE query. You'll also learn where to go for information on other types of queries. For now, let's look at some code:

```python
# queries.py
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from table_def import Album, Artist

engine = create_engine('sqlite:///mymusic.db', echo=True)

# create a Session
Session = sessionmaker(bind=engine)
session = Session()

# how to do a SELECT * (i.e. all)
res = session.query(Artist).all()
for artist in res:
    print artist.name

# how to SELECT the first result
res = session.query(Artist).filter(Artist.name=="Newsboys").first()

# how to sort the results (ORDER_BY)
res = session.query(Album).order_by(Album.title).all()
for album in res:
    print album.title

# how to do a JOINed query
qry = session.query(Artist, Album)
qry = qry.filter(Artist.id==Album.artist_id)
artist, album = qry.filter(Album.title=="Thrive").first()
print

# how to use LIKE in a query
res = session.query(Album).filter(Album.publisher.like("S%a%")).all()
for item in res:
    print item.publisher
```

The first query we run will grab all the artists in the database (a SELECT *) and print out each of their name fields. Next you'll see how to just do a query for a specific artist and return just the first result. The third query shows how do a SELECT * on the Album table and order the results by album title. The fourth query is the same query (a query on a JOIN) we used in our editing section except that we've

broken it down to better fit [PEP8](#) standards regarding line length. Another reason to break down long queries is that they become more readable and easier to fix later on if you messed something up. The last query uses LIKE, which allows us to pattern match or look for something that's "like" a specified string. In this case, we wanted to find any records that had a Publisher that started with a capital "S", some character, an "a" and then anything else. So this will match the publishers Sparrow and Star, for example.

SQLAlchemy also supports IN, IS NULL, NOT, AND, OR and all the other filtering keywords that most DBAs use. SQLAlchemy also supports literal SQL, scalars, etc, etc.

Wrapping Up

At this point you should know SQLAlchemy well enough to get started using it confidently. The project also has excellent documentation that you should be able to use to answer just about anything you need to know. If you get stuck, the SQLAlchemy users group / mailing list is very responsive to new users and even the main developers are there to help you figure things out.

Source Code

- [sqlalchemy0708.zip](#)

Further Reading

- SQLAlchemy official [documentation](#)
- wxPython and SQLAlchemy: [Loading Random SQLite Databases for Viewing](#)
- SqlAlchemy: [Connecting to pre-existing databases](#)
- wxPython and SqlAlchemy: [An Intro to MVC and CRUD](#)
- Another Step-by-Step [SqlAlchemy Tutorial](#) (part 1 of 2)

« [Reportlab: Mixing Fixed Content and Flowables](#) | [wxPython by Example Repository](#) »

---

11 Comments   **Mouse Vs. the Python**   ① **Login** ⌄

♥ **Recommend** 7   ↱ **Share**   Sort by Best ⌄

> Join the discussion…

Florian Kraus · 3 years ago
You've got a little mistake. For getting the first result you have to use the .first()-method on a query. Using the one()-method will also return only one result, but if there is no entry (or more than one) in the database which matches with your query SQLAlchemy will throw an error. See also http://docs.sqlalchemy.org/en/...

Greetings from Germany :)

2 ⌃ | ⌄ • Reply • Share ›

Jon · 2 years ago

Useful stuff - thanks a lot!

One thing that caught me - the datetime.date format doesn't work when a single-digit month or day has leading zeroes. I fiddled around 'til I found this out. Seems a bit obsessive of Python but I'm new around here.

∧ | ∨ · Reply · Share ›

geek42 · 2 years ago

i really want to know how to insert data and return the auto increment id after succeseful for the sqlite engine, in postgresql, you just use returing, but it doesnt work in sqlite case

∧ | ∨ · Reply · Share ›

alfaha · 2 years ago

there might be a "accidental" advantage to not having __init__
you may insert only those variables inside the table with the var=value lists ?

∧ | ∨ · Reply · Share ›

Michel C · 2 years ago

Thank for your blog. I have been following a few posts and found them all well explained and very efficient. Keep up with the good work.

∧ | ∨ · Reply · Share ›

Michael K · 2 years ago

Thanks for the introduction! This is really well-written and helpful.

∧ | ∨ · Reply · Share ›

Beslin · 3 years ago

Nicely explained. Keep posting.

∧ | ∨ · Reply · Share ›

Frank · 3 years ago

I m not a technical person so don't know anything about the about post. But may be later I will understand. :)

∧ | ∨ · Reply · Share ›

Anon · 3 years ago

Aren't the __init__ methods in your declarative classes unnecessary? The Base class created by declarative_base has a default __init__ that assigns keyword args to instance attributes. True, you can't pass positional args using the default __init__, but in most cases I'm not sure why you'd want to.

I know in your example it doesn't matter much, but IRL I have seen unaware users create massive __init__ methods just to assign positional args to instance attributes, which was A) a waste of time up front and B) a pain in the ass because you're forced to provide *all* args even if some columns have

defaults.

ᐱ | ᐯ • Reply • Share ›

Pablo Marti ➜ Anon • 3 years ago

They are indeed unnecessary

ᐱ | ᐯ • Reply • Share ›

kracekumar • 3 years ago

There is no need to create `__init__` for sqlalchemy clasees, It is added automatically.
http://docs.sqlalchemy.org/en/...

ᐱ | ᐯ • Reply • Share ›

ALSO ON MOUSE VS. THE PYTHON

WHAT'S THIS?

PyDev of the Week: Daniel Greenfeld

1 comment • 4 months ago

Avat **chillaranand** — please interview david beazley!!!

Plotting Data Online via Plotly and Python

4 comments • 6 months ago

Avat **Anonymouse** — It's free!! At least as for now (and I hope they will never charge for its use).

50% off on Python 101 Until 2015

2 comments • 5 months ago

Avat **Mike Driscoll** — I hope he likes it!

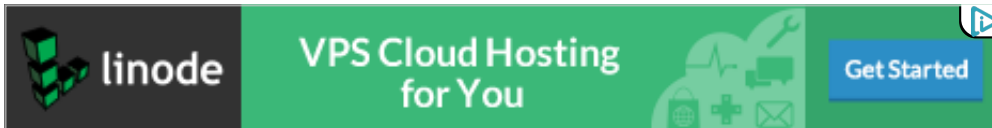### Python 101 Screencast Kickstarter Update

7 comments • a month ago

Avat **Mike Driscoll** — Yeah, this mic had high reviews, but appears to be crap. I'll have to upgrade my …

✉ **Subscribe**   Ⓓ **Add Disqus to your site**   ▷ **Privacy**

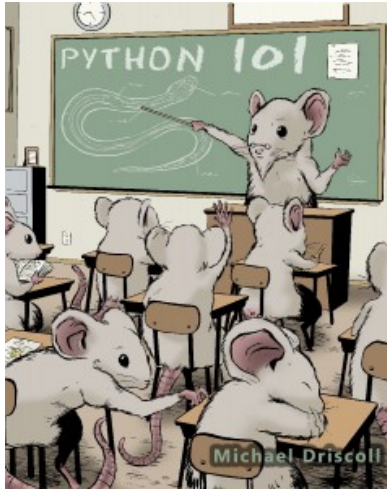- # Archived Entry

  - **Post Date :**
  - Sunday, Jul 1st, 2012 at 6:51 pm

- Category :
- [Python](#) and [SqlAlchemy](#)
- Tags :
- [SQL](#), [SqlAlchemy](#), [sqlite](#)
- Do More :
- You can [leave a response](#), or [trackback](#) from your own site.

- # Buy Python 101

  ## [Order the eBook](#)

  ## [or buy the Paperback!](#)

- # Recent Posts

  - [PyDev of the Week: Adina Howe](#)
  - [47 Hours to End of Kickstarter + New Video](#)
  - [PyDev of the Week: Noah Gift](#)
  - [Python 101 Screencast Update – Another Demo Video](#)
  - [Using pyGal Graphs in Flask](#)

- # Search:

  Search

- # Links

  - [@MouseVsPython](#)
  - [Buy the Shirt!](#)
  - [Python](#)
  - [wxPython](#)

- # Archives

  - Archives Select Month ▼

- # Recent Comments

  - - Eugene T. on [Tkinter: How to Show / Hide a Window](#)
    - Secret Admirer on [47 Hours to End of Kickstarter + New Video](#)
    - [Graphing packet details with PyShark, pygal, and Flask | thePacketGeek](#) on [Using pyGal Graphs in Flask](#)
    - [Mike Driscoll](#) on [Using pyGal Graphs in Flask](#)
    - tray on [Using pyGal Graphs in Flask](#)
  -
    July 2012

S	M	T	W	T	F	S
[1](#)	2	3	4	5	[6](#)	[7](#)
[8](#)	[9](#)	[10](#)	[11](#)	[12](#)	[13](#)	14
15	[16](#)	[17](#)	[18](#)	[19](#)	[20](#)	[21](#)
22	[23](#)	[24](#)	25	[26](#)	27	[28](#)
29	30	[31](#)				

    [« Jun](#)   [Aug »](#)

- # Tags

  - [binaries](#) [Book Preview](#) [Book Review](#) [Books](#) [ConfigObj](#) [Cross-Platform](#) [Distribution](#) [Django](#) [eBooks](#) [Education](#) [GUI](#) [kivy](#) [logging](#) [Packaging](#) [PSF](#) [PyCon](#) [PyCon 2010](#) [PyCon 2011](#) [PyCon 2012](#) [Pyowa](#) [PyPDF](#) [PyQT](#) [PySide](#) [Python](#) [Python 101](#) [Python Advocacy](#) [Python PDF Series](#) [Python Web Frameworks](#) [PyWin32](#) [Reportlab](#) [sizer](#) [SQL](#) [SqlAlchemy](#) [sqlite](#) [System Admin](#) [Testing](#) [threading](#) [Threads](#) [Tkinter](#) [TurboGears](#) [Tutorial](#) [Windows](#) [wxPython](#) [XML](#) [XML Parsing Series](#)

- # Members

  - - [Log in](#)
    - [Entries RSS](#)
    - [Comments RSS](#)
    - [WordPress.org](#)