Object Relational Mapping

Brian Dickinson

University of Rochester

Fall 2019
Reading from our SQLite Database with Python

Interact with SQLite databases in Python using `sqlite3`

```python
import sqlite3

# connect to database
conn = sqlite3.connect("path/to/database.db")
c = conn.cursor()

# construct basic query
query = ("SELECT customer_id, first_name, "
         "last_name, email_address, phone_number "
         "FROM Customers "
         "WHERE first_name=?")
params = ["Brian"]

# execute query and get row results
for row in c.execute(query, params):
    print(row)
```
import sqlite3

# connect to database
cconn = sqlite3.connect("path/to/database.db")
c = conn.cursor()

# construct basic query
query = ("INSERT INTO Customers "
  "(first_name, last_name,"
  " email_address, phone_number)"
  "VALUES (?,?,?,?)")
params = ["Brian", "Dickinson",
  "bdicken3@cs.rochester.edu", "8675309"]

# execute query and commit results to database
c.execute(query, params):
conn.commit()
SQL Injection

Hi, this is your son's school. We're having some computer trouble.

Oh, dear - did he break something? In a way-

Did you really name your son Robert?); DROP TABLE Students;--?

Oh, yes. Little Bobby Tables, we call him.

Well, we've lost this year's student records. I hope you're happy.

And I hope you've learned to sanitize your database inputs.
import sqlite3

class Customer(object):
    def __init__(self, first_name, last_name,
                 email_address, phone_number):
        self.customer_id = None
        self.first_name = first_name
        self.last_name = last_name
        self.email_address = email_address
        self.phone_number = phone_number
def insert(self, db_path):
    conn = sqlite3.connect(db_path)
    c = conn.cursor()
    # execute INSERT query
    c.execute("INSERT INTO Customers (first_name, last_name, email_address, phone_number) VALUES (?, ?, ?, ?)", (self.first_name, self.last_name, self.email_address, self.phone_number))
    # commit changes to database
    conn.commit()
    # set the customer_id assigned by database
    self.customer_id = c.lastrowid
    return self
def update(self, db_path):
    # initialize connection to database
    conn = sqlite3.connect(db_path)
    c = conn.cursor()
    # execute UPDATE query
    c.execute(
        ("UPDATE Customers SET \\
        "first_name=?, last_name=?, "
        "email_address=?, phone_number=?"
        "WHERE customer_id=?"),(
        (self.first_name, self.last_name,
        self.email_address,
        self.phone_number, self.customer_id))
    # commit changes to database
    conn.commit()
    return self
def delete(self, db_path):
    # initiate connection to database
    conn = sqlite3.connect(db_path)
    c = conn.cursor()
    # execute DELETE query
    c.execute(
        ("DELETE FROM Customers "
        "WHERE customer_id=?"),
        (self.customer_id,))
    # commit changes to database
    conn.commit()
    return self
@staticmethod
def select(db_path, customer_id=None, first_name=None,
    last_name=None, email_address=None,
    phone_number=None):
    # initialize database connection
    conn = sqlite3.connect(db_path)
    c = conn.cursor()

    # prepare the basic structure of the query
    query = ("SELECT customer_id, first_name, "
             "last_name, email_address, phone_number "
             "FROM Customers")

    # if necessary, add a WHERE clause
    if any(e is not None for e in (customer_id,
        first_name, last_name, email_address, email_address,
        phone_number)):
query += " WHERE "
params = []
if customer_id is not None:
    query += "customer_id=? "
    params.append(customer_id)
if first_name is not None:
    query += "first_name=? "
    params.append(first_name)
if last_name is not None:
    query += "last_name=? "
    params.append(last_name)
if email_address is not None:
    query += "email_address=? "
    params.append(email_address)
if phone_number is not None:
    query += "phone_number=? "
    params.append(phone_number)
# execute the query and construct Customer objects from the query results

```python
customers = []
for row in c.execute(query, params):
    customer = Customer(*row[1:])
    customer.customer_id = row[0]
    customers.append(customer)

return customers
```
What is Object Relational Mapping (ORM)?

- Abstraction layer between objects and relations
- Simplifies using databases to store objects
- Usually provides portability between database engines
- Creates slight performance overhead, but is only a serious consideration in large enterprise projects
Flask-SQLAlchemy | Introduction

- Provides support for a large number of relational database engines
- Includes an object-relational mapper for Python objects
- Allows for low level access to database queries if needed for efficiency
- Integrates the SQLAlchemy package (which provides all of the above features) with Flask to make it easier to integrate with your web applications
import os
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

# get the absolute path of the directory containing
# this source code file
appdir = os.path.abspath(os.path.dirname(__file__))

# initialize flask application as usual
app = Flask(__name__)

# configure app’s database access
app.config['SQLALCHEMY_DATABASE_URI'] = \
    f'sqlite:///{os.path.join(appdir, 'data.sqlite')}"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# initialize the SQLAlchemy database adaptor
db = SQLAlchemy(app)
class Customer(db.Model):
    __tablename__ = "Customers"
    customer_id = db.Column(db.Integer(), primary_key=True,
                            autoincrement=True)
    first_name = db.Column(db.Unicode(64), nullable=False)
    last_name = db.Column(db.Unicode(64), nullable=False)
    email_address = db.Column(db.Unicode(256), nullable=False)
    phone_number = db.Column(db.String(16), nullable=False)
    orders = db.relationship("Order", backref="customer")
class Order(db.Model):
    __tablename__ = "Orders"
    order_id = db.Column(db.Integer(), primary_key=True,
                         autoincrement=True)
    order_date = db.Column(db.DateTime(), nullable=False)
    customer_id = db.Column(db.Integer(),
                             db.ForeignKey("Customers.customer_id"))
    items = db.relationship("Sale", backref="order")
class Sale(db.Model):
    __tablename__ = "Sales"
    sale_id = db.Column(db.Integer(), primary_key=True, autoincrement=True)
    item_id = db.Column(db.Integer(), nullable=False)
    order_id = db.Column(db.Integer(),
                        db.ForeignKey("Orders.order_id"))
# ... Flask app and db context already initialized

# define objects using the existing models
new_customer = Customer(first_name="Chris",
                        last_name="Robin", email_address="chris.robin@example.com",
                        phone_number="3849752")
new_order = Order(order_date=datetime(2019,6,11,13,44,38),
                  customer=new_customer)
new_sale1 = Sale(item_id="274985316", order=new_order)
new_sale2 = Sale(item_id="947514872", order=new_order)

# add all of these to the database session to track changes
db.session.add_all((new_customer, new_order, new_sale1, new_sale2))

# commit these changes to the database to save changes
db.session.commit()
# ... Flask app and db context already initialized
# ... data from previous slide already inserted

# update one of the objects you are mapping
new_customer.email_address = "chris.robin@gmail.com"

# add it to the session
db.session.add(new_customer)

# commit these changes
db.session.commit()
# ... Flask app and db context already initialized
# ... data from previous slides already inserted

# delete the second sale (that part of order cancelled)
db.session.delete(new_sale2)

# commit this change
db.session.commit()
# ... Flask app and db context already initialized
# ... data from previous slides already inserted

# SELECT * FROM Customers;
Customer.query.all()

# SELECT * FROM Customers WHERE first_name="Chris";
Customer.query.filter_by(first_name="Chris").all()
Summary

- Worked with SQLite databases directly with `sqlite3`
- Introduced **Flask-SQLAlchemy** and showed how to perform all core database functions using the SQLAlchemy ORM framework.
Bibliography I

Python Docs.
Db-api 2.0 interface for sqlite databases.

Miguel Grinberg.
The flask mega-tutorial part viii: Followers.

SQLAlchemy.
The python sql toolkit and object relational mapper.
https://www.sqlalchemy.org/.
Pallets Team.
Declaring models.

Pallets Team.
The python sql toolkit and object relational mapper.

Various.
Object-relational mapping.