SQLAlchemy

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- The lecture will:
  - Provide a taste of the **SQLAlchemy** object-relational mapper
  - Give you enough information about **SQLAlchemy** to:
    - Help you decide if you want to use it in your project
    - Help you get started with it

Motivation

- **Problem**:
  - There is an ***impedence mismatch*** between the relational data model and the OOP data model
    - Awkward to map the relational data model (tables, rows, fields) to the OOP data model (objects, object references, object composition, class inheritance)

Motivation

- **Solution 1**: Cursors
  - As we've seen…
    - Cursor = array + indication of current element
  - DB driver maps each DB **table** to an array
    - Each element represents a row
    - Cursor keeps track of current row
  - DB driver maps each DB **row** to an array
    - Each element represents a field

Motivation

- **Solution 2**: ***Object-relational mapper (ORM)***
  - ORM maps each DB **table** to a **class**
    - E.g., books table => class Book
  - ORM maps each DB **row** to an **object**
    - E.g., row of books table => object of class Book
  - ORM maps each DB **field** to an **object field**
    - E.g. isbn field of some row of books table => isbn field of some object of class Book

Motivation

- **Solution 2**: **Object-relational mapper (ORM) (cont.)**
  - OO pgm fetches/stores data by sending messages to (calling methods of) objects, not by executing SQL statements

Motivation

- Some popular ORMs:
  - See https://en.wikipedia.org/wiki/List_of_object-relational_mapping_software
  - For Python the most popular is…

Motivation

- ***SQLAlchemy***
  - Who: Michael Bayer
  - When: 2006
  - Why: ORM for Python

Installing SQLAlchemy

- Linux, Mac, and MS Windows
  - At a bash shell or Command prompt:
    - Activate your cos333 virtual environment
    - `python -m pip install SQLAlchemy`

SQLAlchemy Pgmming in Python

- See **SQLAlchemy/database.py**
  - Informs SQLAlchemy of database schema
    - Names of tables
    - Names & data types of fields
    - Note:
      - Unusual use of class-level (static) fields
      - For each table, must specify which fields comprise the table's primary key

SQLAlchemy Pgmming in Python

To run the following programs:

Mac & Linux:

```
$ export DATABASE_URL=sqlite:///bookstore.sqlite
$ python create.py
$ python display.py
…
```

MS Windows

```
$ set DATABASE_URL=sqlite:///bookstore.sqlite
$ python create.py
$ python display.py
…
```

SQLAlchemy Pgmming in Python

Debugging trick:

```
engine = sqlalchemy.create_engine(DATABASE_URL)
```

```
engine = sqlalchemy.create_engine(DATABASE_URL,
    echo=True)
```

Commands SQLAlchemy to write to stdout each SQL statement that it issues

SQLAlchemy Pgmming in Python

- See **SQLAlchemy/create.py**
  - To create DB
    - Create engine
    - Create session
    - Create all tables
    - Create books, add to session, commit session
    - Same for authors, customers, zipcodes, orders

SQLAlchemy Pgmming in Python

- See **SQLAlchemy/display.py**
  - To display DB
    - Create engine
    - Create session
    - Send `query()` message to session
      - Specify table
    - Send `all()` message to result
      - Alternative: send `one()` message to result

SQLAlchemy Pgmming in Python

- See **SQLAlchemy/authorsearch.py**
  - To query DB
    - Create engine
    - Create session
    - Send `query()` message to session
    - Send `filter()` message to result
    - ...
    - Send `all()` or `one()` message to result

SQLAlchemy Pgmming in Python

- See **SQLAlchemy/order.py**
  - To update DB
    - Same as query, and then…
    - Update object fields
      - SQLAlchemy marks changed objects as “dirty”
    - Send `commit()` message to session
      - SQLAlchemy writes dirty objects to DB

SQLAlchemy Pgmming in Python

- See **SQLAlchemy/purchase.py**
  - SQLAlchemy supports transactions
    - Can send `commit()` or `rollback()` message to session

SQLAlchemy Pgmming in Python

- See **SQLAlchemy/recovery.py**
  - Transactions work!

SQLAlchemy Bonus

- SQLAlchemy does ***connection pooling***
  - Pattern:
    - Create one engine for app
      - Maintains pool of open connections
      - Pool is shared across multiple threads
    - As needed:
      - Ask engine for Session
        - » Engine returns pooled Session
      - Use Session
      - Close Session
        - » Pools (does not close) the Session
  - Minimizes creation of DB connections
  - Described later in the course

SQLAlchemy Assessment

- SQLAlchemy assessment
  - (pro) Eliminates impedence mismatch
  - (pro) Efficient
  - (pro) Insulates programmer from SQL
  - (con) Insulates programmer from SQL!
    - Bad if your intention is to learn SQL

SQLAlchemy Assessment

- Assignments
  - You **may not** use SQLAlchemy
- Project
  - If you use Python and a relational DB, then you almost certainly **should** use SQLAlchemy

Summary

- There is much more to SQLAlchemy
  - Database creation
  - Automatic management of related tables
  - …
- For more info:
  - https://www.tutorialspoint.com/sqlalchemy
    - Don't read the *SQLAlchemy Core* section
    - Read the *SQLAlchemy ORM* section

Summary

- The lecture has:
  - Provided a taste of the **SQLAlchemy** object-relational mapper
  - Given you enough information about **SQLAlchemy** to:
    - Help you decide if you want to use it in your project
    - Help you get started with it
- See also:
  - **Appendix**: Examples of SQL Statements and SQLAlchemy Code

Appendix: Examples of SQL Statements and SQLAlchemy Code

SQL & SQLAlchemy

```
SELECT * from books;
```

```
query = session.query(database.Book)
table = query.all()
for row in table:
    print(row.isbn, row.title, row.quantity)
```

SQL & SQLAlchemy

```
SELECT isbn, title from books;
```

```
query = session.query(database.Book.isbn,
   database.Book.title)
table = query.all()
for row in table:
    print(row.isbn, row.title)
```

SQL & SQLAlchemy

```
SELECT * from books ORDER BY quantity DESC;
```

```
query = session.query(database.Book).order_by(
    database.Book.quantity.desc())
table = query.all()
for row in table:
    print(row.isbn, row.title, row.quantity)
```

SQL & SQLAlchemy

```
SELECT * from books WHERE quantity=650;
```

```
query = session.query(database.Book).filter(
    database.Book.quantity==650)
table = query.all()
for row in table:
    print(row.isbn, row.title, row.quantity)
```

SQL & SQLAlchemy

```
SELECT * from orders WHERE isbn=123 AND
custid=222;
```

```
query = session.query(database.Order).filter(
    database.Order.isbn==123,
    database.Order.custid==222)
table = query.all()
for row in table:
    print(row.custid, row.isbn, row.quantity)
```

SQL & SQLAlchemy

```
SELECT * from books WHERE title LIKE 'The%';
```

```
query = session.query(database.Book).filter(
    database.Book.title.like('The%'))
table = query.all()
for row in table:
    print(row.isbn, row.title, row.quantity)
```

SQL & SQLAlchemy

```
SELECT * from books, authors WHERE
books.isbn=authors.isbn;
```

```
query = session.query(database.Book,
    database.Author).filter(
    database.Book.isbn == database.Author.isbn)
table = query.all()
for b, a in table:
    print(b.isbn, b.title, b.quantity, a.isbn, a.author)
```

SQL & SQLAlchemy

```
SELECT custname, title, orders.quantity
FROM books, customers, orders WHERE
books.isbn=orders.isbn AND
orders.custid=customers.custid;
```

```
query = session.query(
    database.Customer.custname,
    database.Book.title,
    database.Order.quantity).filter(
    database.Book.isbn == database.Order.isbn,
    database.Order.custid == database.Customer.custid)
table = query.all()
for row in table:
    print(row.custname, row.title, row.quantity)
```

SQL & SQLAlchemy

```
UPDATE books SET quantity=60 WHERE isbn=123;
```

```
query = session.query(database.Book).filter(
    database.Book.isbn == 123)
table = query.all()
for row in table:
    row.quantity = 60
session.commit()
```

SQL & SQLAlchemy

```
INSERT INTO books (isbn, title, quantity) VALUES
('456', 'Core Java', 120);
```

```
new_book = database.Book(isbn='456',
    title='Core Java', quantity='120')
session.add(new_book)
session.commit()
```

SQL & SQLAlchemy

```
DELETE FROM books WHERE isbn=456;
```

```
query = session.query(database.Book).filter(
    database.Book.isbn == 456)
table = query.all()
for row in table:
    session.delete(row)
session.commit()
```