software
studio

a do-it-yourself relational mapper

Daniel Jackson

For now, all rights reserved. Daniel Jackson, 2011.
starting point: the data cycle

update request

show request

response

request.form

<table>
<thead>
<tr>
<th>id</th>
<th>rating</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>yummy!</td>
</tr>
</tbody>
</table>

database table

<table>
<thead>
<tr>
<th>id</th>
<th>by</th>
<th>content</th>
<th>rating</th>
<th>about</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>yummy!</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>neat</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
ReST & CRUD

most requests are just
› create, read, update, delete

web app is just moving data
› request to table
› table to response

are databases a good match?
› old reporting apps: report = table
› new web apps: request = response = tuple

<table>
<thead>
<tr>
<th>operation</th>
<th>HTTP</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>POST</td>
<td>INSERT</td>
</tr>
<tr>
<td>read</td>
<td>GET</td>
<td>SELECT</td>
</tr>
<tr>
<td>update</td>
<td>PUT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>delete</td>
<td>DELETE</td>
<td>DELETE</td>
</tr>
</tbody>
</table>
@app.route('/subjects/<sid>/reviews/<rid>', methods=['GET'])
def show_review(sid, rid):
    review = Review.get_by_id(rid)
    review.user = User.get_by_id(review.by)
    subject = Subject.get_by_id(sid)
    return render_template('review.html', subject=subject, review=review)

class Review:
    def __init__(self, created, rating, content):
        self.created = created
        self.rating = rating
        self.content = content

    @staticmethod
def get_review_by_id(id):
        cur = g.db.execute(''select created, rating, content from reviews where reviews.id = ?''', [id])
        row = cur.fetchone()
        return Review(row[0], row[1], row[2])

class Subject:
    ...
idea: a generalized get

```python
review = Review.get_by_id(rid)

class DatabaseObject(object):
    
    def __init__(self, d):
        for k in d.keys():
            self.__dict__[k] = d[k]

    @classmethod
def get_by_id(_class, id):
        cur = g.db.execute('select * from %s where id = ?' % _class.TABLE_NAME, [id])
        row = cur.fetchone()
        return _class(row)

class Review(DatabaseObject):
    
    TABLE_NAME = 'reviews'

exploiting two things
  › can use * to project onto all columns
  › returned row object can be accessed by column names
what about update?

```python
@app.route('/subjects/<sid>/reviews/<rid>', methods=['POST'])
def update_review (sid, rid):
    review = Review.get_by_id(rid)
    review.created = time.time()
    review.update(request.form)
    return redirect(url_for('show_subject', id=sid))
```

what does review.update do?
- copy form to database table

```sql
cur.execute(
    'update reviews set content = ?, rating = ? where id = ?',
    ['yummy!', 5, 3]
)```
update method

SQL command to be executed

```python
cur.execute(
    'update reviews set content = ?, rating = ? where id = ?',
    ['yummy!', 5, 3]
)
```

class DatabaseObject(object):
    @classmethod
def update_string(_class, d):
        return ('update ' + _class.TABLE_NAME
            + ' set ' + ', '.join(map(lambda colname: (colname + ' = ?'), d.keys()))
            + ' where id = ?')

def update(self, d):
    cur = g.db.cursor()
    values = [d[k] for k in d.keys()]
    values.append(self.id)
cur.execute(self.__class__.update_string(d), values)
g.db.commit()
return

class Review(DatabaseObject):
    TABLE_NAME = 'reviews'
```

• relies on subtle property of d.keys() -- what is it?
what about create?

```python
@app.route('/subjects/<id>/reviews', methods=['POST'])
@requires_login
def create_review (id):
    r = Review(request.form)
    r.created = time.time()
    r.about = id
    r.by = session['user_id']
    r.save()
    return redirect(url_for('show_subject', id=id))
```

for update
› columns determined by keys of dictionary given

for insert
› want every column to be defined
making a save command

class DatabaseObject(object):
    @classmethod
    def insert_string(_class):
        return ('insert into ' + _class.TABLE_NAME
               + '(' + ', '.join(_class.COLUMNS)
               + ') values (' + ', '.join(len(_class.COLUMNS) * ['?']) + ')')

def save(self):
    cur = g.db.cursor()
    cur.execute(self.__class__.insert_string(),
                [self.__dict__[k] for k in self.__class__.COLUMNS])
    id = cur.lastrowid
    g.db.commit()
    self.id = id
    return id

class Review(DatabaseObject):
    TABLE_NAME = 'reviews'
    COLUMNS = ['created', 'by', 'about', 'content', 'rating']
not everything fits

still need to handcode some queries
  › eg, getting most recent reviews for home page

class Subject(DatabaseObject):
  @classmethod
def get_recents(_class, max):
    '''Return up to max most recently reviewed subjects'''
    cur = g.db.execute('''select distinct subjects.* from subjects, reviews
                  where reviews.about = subjects.id
                  order by reviews.created desc
                  limit %s''' % max)
    return map(lambda row: Subject(row, from_db=True), cur.fetchall())