object relational mappers

Daniel Jackson
objects & tables

a poor match
› SQL commands & Python objects

but an underlying similarity?
› the object model!

Review ➔ Text ➔ Rating

<table>
<thead>
<tr>
<th>reviews</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
object relational mappers

what they do
› let you access relational DB in OO style
› (mostly) hide SQL queries

examples
› SQLAlchemy (Python)
› ActiveRecord (Ruby)
› Hibernate (Java)
› Doctrine (PHP)

often built into frameworks
› Rails, Django
how a mapper works

app

`new`
`v = o.f`
`o.f = v`

`add`
`commit`
`rollback`

object pool

class.query...

new
`v = o.f`
`o.f = v`

ORM

database

select
insert
update
delete
mapper OM and key patterns

Class \(\xrightarrow{\text{class-table-map}}\) Table

\(\text{objects}\) \rightarrow \(\text{tuples}\)

Object \(\xrightarrow{\text{obj-tuple-map}}\) Tuple

Dirty Object

Unit of Work

Active Record

Identity Map
SQLAlchemy/Flask example
configuration incantations

```python
DATABASE = 'sqlite:///tipster.db'
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = DATABASE
db = SQLAlchemy(app)
```
class Review(db.Model):
    created = db.Column(db.Integer)
    content = db.Column(db.Text)
    rating = db.Column(db.Integer)
    id = db.Column(db.Integer, primary_key=True)
class Review(db.Model):
    created = db.Column(db.Integer)
    content = db.Column(db.Text)
    rating = db.Column(db.Integer)
    id = db.Column(db.Integer, primary_key=True)

    def __init__(self, dict):
        self.content = dict['content']
        self.rating = dict['rating']

can you guess?
› why isn’t created assigned in __init__?
declaring many-to-one

```python
class Review(db.Model):
    created = db.Column(db.Integer)
    ...

    about_id = db.Column(db.Integer, db.ForeignKey('subject.id'))
    about = db.relationship('Subject',
        backref=db.backref('reviews'))
```

what does this do?
› associates `review.about.id` with `review.about_id`
› synchronizes `subject.reviews` and `review.about`
C: creating a new review

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

key mapper operations
› add: register object in pool for tracking
› commit: make database update permanent
R: reading a review

```python
@app.route('/subjects/<sid>/reviews/<rid>', methods=['GET'])
def show_review (sid, rid):
    review = Review.query.get(rid)
    review.by = User.query.get(review.by.id)
    subject = Subject.query.get(sid)
    return render_template('review.html',
                           subject=subject, review=review)
```

our first query

```python
class.query.get(id)
```
U: updating a review

```python
@app.route('/subjects/<sid>/reviews/<rid>', methods=['POST'])
@requires_login
def update_review (sid, rid):
    review = Review.query.get(rid)
    review.created = time.time()
    review.rating = request.form['rating']
    review.content = request.form['content']
    db.session.commit()
    return redirect(url_for('show_subject', id=sid))
```

patterns in play here

› identity map: review returned by get is canonical
› active record: updates to review sync’d to database
› unit of work: commit makes permanent
D: deleting a review

```python
db.session.delete(review)
db.session.commit()
```
writing a query

```python
@staticmethod
def get_recents(max):
    """Return up to max most recently reviewed subjects""
    subjects = Subject.query.join(Review)
    .order_by(Review.created.desc())
    .limit(max).all()
    return subjects
```

```
@staticmethod
def get_recents(max):
    cur = g.db.execute("\n    select distinct subjects.*
    from subjects, reviews where reviews.about = subjects.id
    order by reviews.created desc limit %s\n    """ % max)
    return map(lambda row: Subject(row, from_db=True),
               cur.fetchall())
```

with SQLAlchemy

without