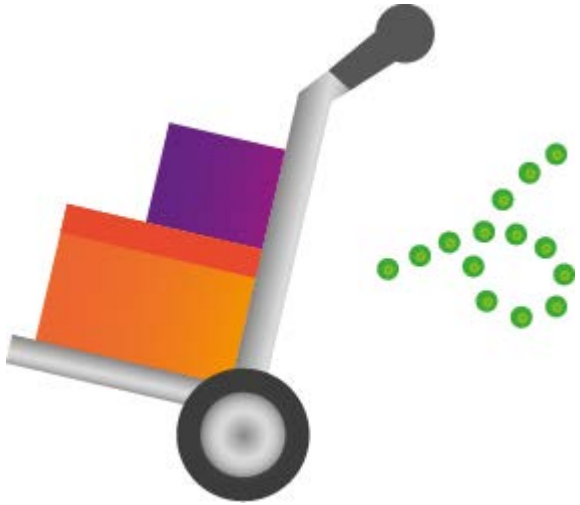
Persistence

Reindert-Jan Ekker
nl.linkedin.com/in/rjekker/
@rjekker

pluralsight
hardcore dev and IT training

In This Module



- **Data persistence**
  - Storing bookmarks
  - Storing user details
- **Models**
  - ORM: **Sqlalchemy**
  - Model classes and database tables
  - Simple relations and queries
  - Database administration with **Flask-script**
- **Along the way**
  - Breaking up the app in multiple Python files

Setting up Flask-SQLAlchemy

- `pip install flask-sqlalchemy`
- Import and configure

```
from flask_sqlalchemy import SQLAlchemy

app.config['SQLALCHEMY_DATABASE_URI'] = \
    'sqlite:///path/to/database'
db = SQLAlchemy(app)
```

- SQLalchemy also supports: MySQL, MS SQL, PostgreSQL, Oracle and more.

Model Classes

- **A model class represents a database table**
  - Every instance represents a row in that table
- **Inherit from db.Model**

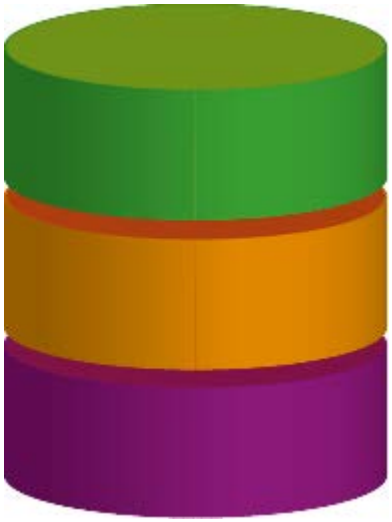
- **Flask-SQLAlchemy**
  - Takes care of some things that plain SQLAlchemy does not
  - Don't have to set `__tablename__` for models
  - `query` attribute on models used for querying

Columns

```
id = db.Column(db.Integer, primary_key=True)
url = db.Column(db.Text, nullable=False)
description = db.Column(db.String(300))
```

- **Defined as class attributes on the Model class**
  - Name of database column will be name of the attribute
- **Instance of `db.Column`**
  - Data type: `db.Integer, db.String, etc.`
  - Options: `primary_key, nullable, etc.`

Creating New Data



- **Simply create a new instance of the class**
  - `bm = new Bookmark(user=u, url=url, description=desc)`
- **Add it to the database session**
  - `db.session.add(bm)`
  - This does NOT add the data to the database
  - It registers the object with the session
- **Don't forget to commit**
  - `db.session.commit()`
  - Will run an SQL INSERT statement

Simple Queries

- Using the `query` attribute of a model class

- Get by primary key:
  - `Bookmark.query.get(1)`

- Retrieve all rows:
  - `Bookmark.query.all()`

- Select specific rows:
  - `Bookmark.query.filter_by(username="reindert").first()`

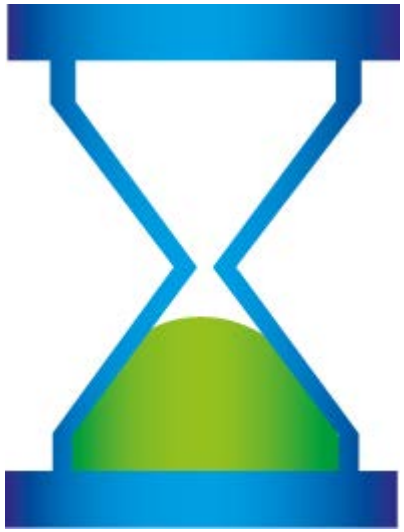
A One-To-Many Relation

```
class Bookmark(db.Model):
    user_id = db.Column(db.Integer,
            db.ForeignKey('user.id'), nullable=False)
```

```
class User(db.Model, UserMixin):
    bookmarks = db.relationship('Bookmark',
                backref='user', lazy='dynamic')
```

- **`db.relationship`**
  - Defines a one-to-many relation
  - First argument gives many side of the relation
- **`backref`**
  - Name of an attribute on the related object
- **`lazy`**
  - How the related rows should be loaded

Lazy Loading Options



- `lazy = 'select'` (default)
  - load the data lazily using a standard select statement
- `lazy = ’joined'`
  - load the data in the same query as the parent using a JOIN statement.
- `lazy = ’subquery'`
  - like 'joined’ but use a subquery.
- `lazy = 'dynamic’`
  - useful if you have many items
  - returns a query object which you can further refine before loading items.
  - usually what you want if you expect more than a handful of items

Flask-SQLAlchemy Helpers

```
@app.route('/user/<username>')
def user(username):
```

- `first_or_404()`
- `get_or_404()`

Simple Database Administration

- `db.create_all()`
- `db.drop_all()`
- Using Flask-Script to create an admin interface

Resources

- http://pythonhosted.org/Flask-SQLAlchemy/
  (http://goo.gl/nwxMzQ)

- http://docs.sqlalchemy.org/en/latest/
  (http://goo.gl/j48tsE)
- http://flask-script.readthedocs.org/en/latest/
  (http://goo.gl/yoO3mn)

Summary

- **Flask-Sqlalchemy**
  - Models
  - Columns
  - Create and drop
  - Insert data
  - Simple queries
  - One-to-many relations and lazy loading
  - first_or_404
- **Flask-Script**
- **Breaking up the application into multiple files**

Visuals made by Britt van de Pol