SQLAlchemy Cheat Sheet

Making Models

models.py

```
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy()

class Pet(db.Model):
    __tablename__ = "pets"

    # can specify multi-column unique or check constraints like:
    # __table_args__ = (
    #    db.UniqueConstraint("col1", "col2"),
    #    db.CheckConstraint("born <= died") )

    id = db.Column(
        db.Integer,
        primary_key=True,
        autoincrement=True)
    name = db.Column(
        db.String(50),
        db.CheckConstraint('len(name) >= 5'),
        nullable=False,
        unique=True)
    species = db.Column(
        db.String(30),
        nullable=True,
        default="cat")
    hunger = db.Column(
        db.Integer,
        nullable=False,
        default=20)
    created_at = db.Column(
        db.DateTime,
        nullable=False,
        default=db.func.now)
```

SQLAlchemy types:

- *Integer*, *String(len)*, *Text*, *Boolean*, *DateTime*, *Float*, *Numeric*

Field options *(all default to False)*:

- *primary_key*, *autoincrement*, *nullable*, *unique*, *default* (value or callback)

Creating/Dropping Tables:

- `db.create_all()`, `db.drop_all()`

Making and Deleting Instances

Making an instance and adding *(only need to do 1st time adding)*:

- `fluffy = Pet(name="Fluffy", species="cat")`
- `db.session.add(fluffy)` or `db.session.add_all([fluffy, bob])`

Deleting instance or deleting all matching data:

- `fluffy.query.delete()` or `Pet.query.filter(...).delete()`

Getting and Filtering

Getting record by primary key:

- `fluffy = Pet.query.get("fluffy")` or `Pet.query.get_or_404("fluffy")`

Simple Filtering: *(returns a "query", not the answer—see fetching below)*

- `Pet.query.filter_by(species="cat")`

Flexible filtering: *(returns "query")*

- `Pet.query.filter(Pet.species == "dog")`
- also: `Pet.hunger != 10`, `.filter(Pet.hunger < 10)`
- also: `Pet.name.like('%uff%')`, `Pet.name.like('%uff%')`, `Pet.hunger.in_([2, 7])`
- OR: `expr | expr`, AND: `expr & expr`, NOT: `~ expr`

Grouping, Ordering, Offsetting, Limiting:

- `.group_by('species', 'age')`
- `.group_by('species').having(db.func.count() > 2)`
- `.order_by('species', 'age')`, `.offset(10)`, `.limit(10)`

Getting lightweight tuples, not instances of model class:

- `db.session.query(Pet.name, Pet.hunger)` → `[("fluffy", 10), ("bob", 3)]`

Fetching:

- `query.get(pk)`
- `query.get_or_404(pk)` *(Flask-specific: get or raise 404)*
- `query.all()` *(get all as list)*
- `query.first()` *(get first record or None)*
- `query.one()` *(get first record, error if 0 or if > 1)*
- `query.one_or_none()` *(get first record, error if > 1, None if 0)*
- `query.count()` *(returns # of elements)*

Transactions

"Flushing" *(sending SQL to database, but doesn't commit transaction yet)*

- `db.session.flush()`

Committing or rolling back transactions:

- `db.session.commit()`, `db.session.rollback()`

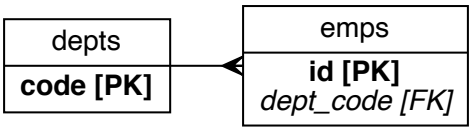
Handling Errors

Import SQLA exception classes from *sqlalchemy.exc*, like this:

```
from sqlalchemy import exc

try:
    User.query.delete()  # delete all users
except exc.IntegrityError:
    print("Cannot delete users because of ref integrity!")
```

Relationships




```
class Employee(db.Model):
    __tablename__ = "emps"
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    dept_code = db.Column(db.Text, db.ForeignKey('depts.dept_code'))
    # remember to make foreign keys `nullable=False` if required!

class Department(db.Model):
    __tablename__ = "depts"
    dept_code = db.Column(db.Text, primary_key=True)
    employees = db.relationship('Employee', backref='department')
```

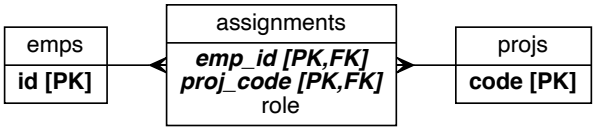
Can navigate like:

```
>>> jane.department     #  <Department finance>
>>> finance.employees   #  [<Employee jane>, <Employee bob>]
```

Can add/remove/clear foreign key data via relationships:

```
>>> finance.employees.append(bob)
>>> finance.employees.remove(bob)
>>> finance.employees.clear()
```

Many to Many Relationships




```
class Employee(db.Model):
    __tablename__ = "emps"
    id = db.Column(db.Integer, primary_key=True, auto_increment=True)
    # can nav from employee to projects, or project to employees
    projects = db.relationship(
      'Project', secondary='assignments', backref='employees')

class Project(db.Model):
    __tablename__ = "projs"
    code = db.Column(db.Text, primary_key=True)

class Assignment(db.Model):
    __tablename__ = "assignments"
    emp_id = db.Column(
        db.Integer,
        db.ForeignKey("emps.id"),
        primary_key=True)
    proj_code = db.Column(
        db.Text,
        db.ForeignKey("projs.code"),
        primary_key=True)
    role = db.Column(db.Text, nullable=False, default='')

    # if you want to nav emp<->assignment and project<->assignment
    project = db.relationship("Project", backref="assignments")
    project = db.relationship("Employee", backref="assignments")
```

Can navigate like:

```
>>> jane.projects        # [<Project A>, <Project B>]
>>> proj_a.employees     # [<Employee 1>, <Employee 2>]

>>> jane.assignments     # [<Assignment jane A>, <Assignment jane B>]
>>> proj_a.assignments   # [<Assignment jane A>, <Assignement bob A>]
>>> asn_jane_a.employee  # <Employee 1>
>>> asn_jane_a.project   # <Project A>
```

Can add/edit/remove foreign key data via relationships:

```
>>> jane.projects.add(project_a)
>>> jane.projects.remove(project_a)
>>> jane.projects.clear()

>>> jane.assignments.add(Assignment(proj_code='a', role='Chair'))
```

Written and maintained by Joel Burton <joel@joelburton.com> for Rithm School.