Using the Python Django ORM For SQL Data Analysis

Ian Stokes-Rees **@ijstokes**
(http://about.me/ijstokes), **(http://twitter.com/ijstokes)** **(http://www.linkedin.com/in/ijstokes)**

PyData Boston , July 2013
(http://pydata.org/bos2013)

Continuum Analytics
(http://continuum.io)

(http://continuum.io)

Setup

- Clone my tutorial notes from github: **https://github.com/ijstokes/sql-analysis-with-django-orm.git (https://github.com/ijstokes/sql-analysis-with-django-orm.git)**
- Using MySQL **Employees Test DB (https://launchpad.net/test-db/)** for trial data, with modified **`employees.sql`** **(employees.sql)** to work with SQLite3.
- Don't download this now -- 25 MB compressed, 250 MB uncompressed
- Unzip the Employees test database into the GitHub directory

Data Transformation

- Some work to massage SQL from MySQL syntax to something SQLite3 will be happy with -- repo contains updated schema in `employees.sql`, but some manual search-and-replace is still required.
- Hint if you use vim, adapt for other language search/replace:

```
%s/),/);^M INSERT INTO `replace_with_table_name` VALUES/g
```

- Need to change all foreign key references to department by string name to department by integer.
- Need to remove composite primary keys and add an `id` field to all tables except `employees`.
- Need to run `add_idx.py load_TABLENAME.dump` on all dump tables except `load_employees.dump`

Create SQLite3 DB from SQL Dump

```
sqlite3 -init employees.sql employees.db
```

This will create 6 tables. The main one is:

- Employees
- `emp_no`
- `birth_date`
- `first_name`
- `last_name`
- `gender`
- `hire_date`

The other 5 describe:

- Departments
- Managers (DeptManager)
- Salaries
- Titles
- Department Assignments (DeptEmp)

Slurping this data in takes a little while (about 30 minutes for me).

Initialize Django Environment

You need to have Django installed. A few of these options will work if you don't have it already:

```
<span class="kw">pip</span> install django
<span class="kw">conda</span> install django
```

Now initialize the Django project:

```
<span class="kw">django-admin.py</span> startproject datasnoop
<span class="kw">mv</span> datasnoop/datasnoop/* datasnoop
<span class="kw">mv</span> datasnoop/manage.py .
<span class="kw">rm</span> -Rf datasnoop/datasnoop
```

This sets up the basic Django pieces. Now we need to add the "app" that will be the specific container for the Employee data:

```
<span class="kw">pushd</span> datasnoop
<span class="kw">./manage.py</span> startapp employees
<span class="kw">popd</span>
```

Grab Models from Existing DB

Edit `datasnoop/settings.py` to point to `employee.db`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, '../employees_db/employees.db'),
    }
}
```

And also modify `datasnoop/settings.py` to include `datasnoop.employees` in `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'datasnoop.employees'
)
```

Now we can use `inspectdb` to extract models from the DB:

```
./manage.py inspectdb > datasnoop/employees/models.py
```

Let's take a look at what this has given us in `datasnoop/employees/models.py`:

```
<span class="kw">class</span> Employees(models.Model):
    emp_no     = models.IntegerField(primary_key=True)
    birth_date = models.DateField()
    first_name = models.CharField(max_length=<span class="dv">14</span>)
    last_name  = models.CharField(max_length=<span class="dv">16</span>)
    gender     = models.CharField(max_length=<span class="dv">1</span>)
    hire_date  = models.DateField()
    <span class="kw">class</span> Meta:
        <span class="kw">managed</span>  = <span class="kw">False</span>
        db_table = <span class="st">'employees'</span>
```

Nice! We now have an ORM definition for interacting with the DB.

Create Model Admin Definitions

For our whizzy Django Admin interface to work, we need to register the auto-generated models using Django's admin class format.

We need to create a file `datasnoop/employees/admin.py` that contains:

```
from django.contrib import admin
from datasnoop.employees.models import Departments, DeptEmp, DeptManager
from datasnoop.employees.models import Employees, Salaries, Titles

for cls in (Departments, DeptEmp, DeptManager, Employees, Salaries, Titles):
    admin.site.register(cls)
```

We'll see shortly that a bit more work will be needed but this is a good start.

Update DB with Django Doodads

Take a look at our current tables:

```
$ sqlite3 employees_db/employees.db .tables
    departments   dept_manager  salaries
    dept_emp      employees     titles
```

Django has some of its own administrative tables it needs, so we'll create these:

```
./manage.py syncdb
```

When prompted, add an admin user account with a valid email address and password, then look at the tables that now exist:

```
$ sqlite3 employees_db/employees.db .tables
    auth_group                  dept_manager
    auth_group_permissions      django_admin_log
    auth_permission             django_content_type
    auth_user                   django_session
    auth_user_groups            employees
    auth_user_user_permissions  salaries
    departments                 titles
    dept_emp
```

Let's Take A Look!

We'll startup a simple webserver to connect to the admin interface of our Django website:

```
./manage.py runserver
```

Now connect to the admin interface via **`http://localhost:8000/admin/`** **(http://localhost:8000/admin/)** with the username and password you set earlier.

Well, not bad, but the aggregated lists of objects aren't very informative. We'll fix that next.

Augment ModelAdmins

Take a look at `datasnoop/employees/admin_basic.py`, which is what we started with:

```
from datasnoop.employees.models import Departments, DeptEmp, DeptManager, Employees, Salar
ies, Titles

for cls in (Departments, DeptEmp, DeptManager, Employees, Salaries, Titles):
    admin.site.register(cls)
```

We're just taking advantage of the automatic admin interface. Instead, we need to specify exactly which fields we want to display in our list view, which we do in `datasnoop/employees/admin_list.py`:

```
class EmployeesAdmin(admin.ModelAdmin):
    list_display = ('emp_no', 'last_name', 'first_name', 'gender', 'birth_date', 'hire_dat
e')
admin.site.register(Employees, EmployeesAdmin)
```

Now let's see what this give us through our Admin web interface: **`http://localhost:8000/admin/`** **(http://localhost:8000/admin/)**

Introduce Search and Filter

If we click on the column headers, we can see it will sort the results by that column. Django Admin also provides mechanisms to facilitate search and filter. Let's add those in, by looking at `datasnoop/employees/admin_filter.py`:

```
class EmployeesAdmin(admin.ModelAdmin):
    list_display = ('emp_no', 'last_name', 'first_name', 'gender', 'birth_date', 'hire_dat
e')
    list_filter  = ('gender', 'birth_date', 'hire_date')
    search_fields = ['last_name']
    date_hierarchy = 'birth_date'
admin.site.register(Employees, EmployeesAdmin)
```

Foreign Keys

If you didn't perform the SQL data transformations described earlier, then the auto-generated models from `inspectdb` won't identify foreign keys, so unfortunately we can't reference through.

As a challenge, see if you can modify `datasnoop/employees/models.py` to reference ForeignKeys correctly -- there are some catches because of how Django wants to auto-index all keys.

Without `model.ForeignKey('Foo')` references we can't grab data from referenced fields.

Django CLI to the ORM

We get into the Django CLI by doing:

```
./manage.py shell
```

From here we can do:

```
from datasnoop.employees.models import Employees, Departments, Salaries
men   = Employees.objects.filter(gender='M')
parto = Employees.objects.get(emp_no=10003)
```

Further References

- **Django ORM (https://docs.djangoproject.com/en/dev/topics/db/)**
- **Django Legacy DB Integration (https://docs.djangoproject.com/en/dev/howto/legacy-databases/)**
- **Django Admin Interface (https://docs.djangoproject.com/en/dev/ref/contrib/admin/)**