Schema Migrations with South

Managing databases has never been better.

Werner Fortmann

Macquarie University
Department of Actuarial Studies
Department of Economics
e: werner.fortmann@gmail.com

August 23, 2012

What are migrations?

Migrations are a way of changing a database from one version to another. For example in Django, if you change one of your existing models you need to modify the underlying database table to reflect this change. Django can not do this itself, and that is where South steps in to save the day.
The example followed in this presentation is adapted from the one in the official South tutorial:

- `http://south.readthedocs.org/en/latest/tutorial/`

A Quick Example - Initial Model

Let's say that we have a simple user model. A dangerously simple one.

```python
class User(models.Model):
    username = models.CharField(max_length=255)
    password = models.CharField(max_length=60)
    name = models.TextField()
```

- If we sync this using django's syncdb, then django will automatically create a table *app_user*, with the fields *username*, *password*, and *name*.

A Quick Example - The Model Changes

- After we have synced the database we decide we also want to store the user's hair colour. Our model now becomes:

```
class User(models.Model):
    username = models.CharField(max_length=255)
    password = models.CharField(max_length=60)
    name = models.TextField()
    COLOURS=(('RED','red'),('BLUE','blue'),('OTHR','other'),)
    haircolour = models.CharField(max_length=4, choices=COLOURS, c
```

- If we run syncdb again the database doesn't change, because the User model's database table already exists.
- You could either create the column manually or drop the table and run syncdb again. **But there is a better way! South!**

A Quick Example - Enter South

- Installing and configuring south is easy:
  - pip install south
  - add 'south' to INSTALLED_APPS in your settings.py
- To use south you need to define the initial state of the database, so let's rollback our change to our User model so rolling back to our original model we have:

```
class User(models.Model):
    username = models.CharField(max_length=255)
    password = models.CharField(max_length=60)
    name = models.TextField()
```

- Our models and database are now matching, so we can initialise south.

Set Database Initial State

- The normal workflow to set the initial state is:
  1. ./manage.py schemamigration southintro - -initial
  2. ./manage.py migrate southintro
- But because our database tables already exist, the workflow is:
  1. ./manage.py schemamigration southintro - -initial
  2. ./manage.py migrate southintro - -fake
- Or you can do it in one go with:
  1. ./manage.py convert_to_south southintro

The Model Changes with South

- Let’s add the *haircolour* field again.

```
class User(models.Model):
    username = models.CharField(max_length=255)
    password = models.CharField(max_length=60)
    name = models.TextField()
    COLOURS=(('RED','red'),('BLUE','blue'),('OTHR','other'),)
    haircolour = models.CharField(max_length=4, choices=COLOURS,
```

- We can now use south to easily migrate our database schema:
  1. ./manage.py schemamigration southintro - -auto
  2. ./manage.py migrate southintro

Advanced Changes and Data Migration

- Our User model has a worrisome problem. The password is stored as plain text. Let's fix that by salting and hashing our password. Before we store it. We'll do that with the following steps:
  1. Add a password_salt and password_hash field
  2. Migrate our plain text passwords by creating salts and hashing the passwords.
  3. Remove the password field to delete the stored plain text passwords.
- Our model at step 1 looks like:

```
class User(models.Model):
    username = models.CharField(max_length=255)
    password = models.CharField(max_length=60)
    password_hash = models.CharField(max_length=100, null = True
    password_salt = models.CharField(max_length=8, null = True)
    name = models.TextField()
    COLOURS=(('RED','red'),('BLUE','blue'),('OTHR','other'),)
    haircolour = models.CharField(max_length=4, choices=COLOURS,
```

- Again we use South to migrate our schema.
  1. ./manage.py schemamigration southintro - -auto
  2. ./manage.py migrate southintro

Advanced Changes and Data Migration

- Now we need to migrate our data by creating a salt and hashing our plain text passwords.
- To do this we create a South data migration:
  - ./manage.py datamigration southintro hash_passwords
- This commands creates an empty shell of a data migration. We need to now go in and manually write the migration code.

```
def forwards(self, orm):
  "Write your forwards methods here."
  # Note: Remember to use orm['appname.ModelName'] rather than '
  import random, hashlib, string
  for user in orm.User.objects.all():
    user.password_salt = "".join([random.choice(string.letters)
    user.password_hash = hashlib.sha256(user.password_salt + use
    user.save()

def backwards(self, orm):
    "Write your backwards methods here."
    raise RuntimeError("Cannot reverse this migration.")
```

- Migrate with south: ./manage.py migrate southintro

Advanced Changes and Data Migration

- Finally we remove the plain text password field and make the salt and hash fields required.

```
class User(models.Model):
    username = models.CharField(max_length=255)
    password_hash = models.CharField(max_length=100, null = False)
    password_salt = models.CharField(max_length=8, null = False)
    name = models.TextField()
    COLOURS=(('RED','red'),('BLUE','blue'),('OTHR','other'),)
    haircolour = models.CharField(max_length=4, choices=COLOURS, c
```

- Migrate with south:
  1. ./manage.py schemamigration southintro - -auto
  2. ./manage.py migrate southintro