Configuring Multiple Databases in Django

Ryan Scott

Introduction

- Software Engineer @ Percipient Networks
- Former Node.js and Salesforce developer
- Several years using Python and Django for fun
- 7 months using Python and Django professionally

Topics:

- Adding a second database to Django
- Using the second database
- Migrating the second database
- Testing the second database

Why do we need a second database?

- Integrating with another application or service
- Separation of concerns
- Different types of data => different types of databases

Real-world scenario

- Strongarm service receives ~5 million DNS requests per day
- Goal is to save all requests and provide analytics for users about their network
- Current database contains over 1 billion DNS requests (and growing!)
- Saved in a second database to keep separate from mission-critical data (user accounts, infections, etc.)

Django's database settings

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'defaultdb',
        'USER': 'username',
        'PASSWORD': 'password',
        ...
    }
}
```

Adding a second database

```
DATABASES = {
    'default': {...},
    'example_db': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'exampledb',
        'USER': 'username2',
        'PASSWORD': 'password2',
        ...
    }
}
```

Manually selecting a database

```
ExampleModel.objects.using('example_db').all()
```

or

```
x = ExampleModel(name='Fred')
x.save(using='example_db')
```

Django Database Router

- Programmatically decide which database to use
- Powerful (and complex) conditional logic
- Installed by adding the class path to the `DATABASE_ROUTERS` setting:

```
DATABASE_ROUTERS = ['path.to.ExampleDatabaseRouter']
```

- Works with multiple databases and multiple router classes

Django Database Router

```
class ExampleDatabaseRouter(object):
    def db_for_read(self, model, **hints):
        ...

    def db_for_write(self, model, **hints):
        ...

    def allow_relation(self, obj1, obj2, **hints):
        ...

    def allow_migrate(self, db, app_label, model_name=None, **hints):
        ...
```

Django Database Router

Reading

```
def db_for_read(self, model, **hints):
    """
    Send all read operations on Example app models to `example_db`.
    """

    if model._meta.app_label == 'example':
        return 'example_db'
    return None
```

Django Database Router

Writing

```
def db_for_write(self, model, **hints):
    """
    Send all write operations on Example app models to `example_db`.
    """

    if model._meta.app_label == 'example':
        return 'example_db'
    return None
```

Django Database Router

Relationships

```
def allow_relation(self, obj1, obj2, **hints):
    """Determine if relationship is allowed between two objects."""

    # Allow any relation between two models that are both in the Example app.
    if obj1._meta.app_label == 'example' and obj2._meta.app_label == 'example':
        return True
    # No opinion if neither object is in the Example app (defer to default or other routers).
    elif 'example' not in [obj1._meta.app_label, obj2._meta.app_label]:
        return None

    # Block relationship if one object is in the Example app and the other isn't.
        return False
```

Django Database Router

Migrations

```
def allow_migrate(self, db, app_label, model_name=None, **hints):
    """Ensure that the Example app's models get created on the right database."""

    if app_label == 'example':
        # The Example app should be migrated only on the example_db database.
        return db == 'example_db'
    elif db == 'example_db':
        # Ensure that all other apps don't get migrated on the example_db database.
        return False

    # No opinion for all other scenarios
    return None
```

Migrating multiple databases

- The `migrate` command operates on 1 database at a time.
- You must run `migrate` for each database!
- Django keeps track of migrations that were run against a database even if the migration wasn't actually applied to that database.
- This can quickly get repetitive. Checkout Fabric for simplifying this into one command.

Unit tests with multiple databases

- Fixtures can be loaded similar to the default database
- Enable fixtures with the `multi_db` parameter
- Fixture name must follow the pattern:
  `fixture_name.db_name.json`
- This feels a little like "magic"

Unit test example

```
class TestMyViews(TestCase):
    # Load fixture data from all databases
    multi_db = True

    # Load `test.json` into the default database
    # Load `test.example_db.json` into the `example_db` database
    fixtures = ['test']

    def test_index_page_view(self):
        call_some_test_code()
```

See my recent blog post for more details

strongarm.io/blog/multiple-databases-in-django/

Thank you

github.com/percipient/talks

ryan@strongarm.io