Multi-database Patterns

Malcolm Tredinnick

Menu

- Three realistic patterns
- Increasing complexity
- Working code...

    github.com/malcolmt/django-multidb-patterns

The patterns

My examples versus The Real World(tm)

Hypothetical Project

- Product reviews
- A few (1000's?) product
- Bajillions of customer reviews

Hypothetical Project

- Product reviews
- ~~A few (1000's?)~~ Two products
- ~~Bajillions of~~ Some customer reviews
  - We would like to apologize for the hyperbole
  - Those responsible have been sacked
  - Mynd you, møøse bites Kan be pretty nasti...

Setup

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': ...,
        ...
        },
    "reviews": {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': ...,
        ...
        }
}
```

Database Routing

- Given a model (and maybe an instance)...
  - Which database to read from?
  - Which database to write to?
  - Should this model's table be sync'd to this db?
- Can say "don't care" and pass off responsibility

Database Routing

```
class ReviewRouter(object):
    """
    Sends all review-related operations to a database
    with the alias of "reviews". No other apps should use
    this db alias.
    """

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

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

    def allow_syncdb(self, db, model):
        ...
```

./manage.py syncdb --database=...

Admin

```
class ReviewAdmin(admin.ModelAdmin):
    ...
    _using = "reviews"

    def save_model(self, request, obj, form, change):
        obj.save(using=self._using)
```

Functional Separation

- Products table in one database (*default*)
- Reviews table in *reviews* database
- Router sends everything with app name of review to *reviews* alias.

Functional Separation

- Great for integrating existing databases with new work in second db
- Cannot have relations between databases
  - (see also “atomic”, “consistent”, etc)
- Easy to develop and debug

Access Separation

- Write / read
- Public / draft
- External / internal

Read / write

- db_for_read() returns “read” database
- db_for_write() returns “write” database

Synchronization lag

Arrange to stick reads to source

```
def add_review(request, product_id=None):

    if request.method == "POST":
        form = ReviewForm(request.POST)
        if form.is_valid():
            ...
            request.session[KEY] = time.time()
            ...
    ...
```

Make read decision in view

```
def show_review(request, review_id=None):
    ...

    review_mgr = reviews.models.Review.objects

    if (request.session.get(KEY, 0) > time.time() - 300):
        # Tie reads to the master reviews database.
        review_mgr = review_mgr.db_manager("reviews")
    ...
```

Sharding

- Distribute reviews amongst N (5, 10, 250...) database
- Pick a distribution facet (e.g. review id)
- Distribute evenly (hash)
- Need to know facet value to choose write db

Changing # of databases later is fiddly

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

    if model._meta.app_label != "reviews":
        return None

    try:
        obj = hints.pop("instance")
    except KeyError:
        raise Exception("Cannot get write db without instance.")

    return "reviews-%d" % obj.get_db_num()
```

```
class Review(models.Model):
    ...

    id = models.IntegerField(default=Ticket.objects.new,
            primary_key=True)

    ...

    def get_db_num(self):
            return 1 + (int(hashlib.md5(str(self.id)).hexdigest(), 16) %
                    settings.CLUSTER_SIZE)
```

```
def show_review(request, review_id=None):
    ...

    alias = ("reviews-%d" %
            reviews.models.get_db_for_id(review_id))
    try:
        review = reviews.models.Review.objects.using(alias). \
                get(id=review_id)
    except reviews.models.Review.DoesNotExist:
        ...
    ...
```

```
def product_reviews(request, product_id=None):
    ...
    for db_alias in router.alias_iter():
        current_qs = review_qs.using(db_alias)
        ...
```