ТЕХНОТРЕК

Урок второй

ORM, DRF репликации, шардинг

Антон Кухтичев

Посещаемость

Не забудьте отметиться!!!!

Программа на сегодня

1. ORM штучки;
2. Django Rest Framework (DRF);
3. Репликации. Виды;
4. Шардирование.

ТЕХНОТРЕК

ORM штучки

F()

Экземпляр F() рассматривается как ссылка на другое поле модели. Эти ссылки могут быть использованы для сравнения значений двух разных полей одного объекта модели.

Вместо такого кода:

```
article = Article.objects.get(pk=1)
article.count += 1
article.save()
```

Делаем так:

```
Article.objects.filter(pk=1).update(count=F('count')
+ 1)
```

Q()

Объект Q(), как и F, инкапсулирует SQL выражение в Python объекте, который может использоваться для указания операций над базой данных.

Объекты Q() позволяют определить условия и использовать их повторно. Это позволяет создавать сложные запросы к базе данных, используя операторы | (OR) и & (AND); в частности, это единственный способ использовать OR в QuerySets.

Копирование объектов

Нужно присвоить полю pk None и сохранить

```
blog = Blog(name='My blog', tagline='Blogging is easy')
blog.save() # blog.pk == 1

blog.pk = None
blog.save() # blog.pk == 2
```

bulk_create()

Этот метод позволяет сохранить в базе данных множество объектов одним запросом.

1. Метод модели save() не будет вызван, и сигналы pre_save и post_save не будут вызваны;
2. Не работает с дочерними моделями при multi-table наследовании;
3. Если первичный ключ модели это AutoField, его значение не будет получено и атрибут первичного ключа не будет установлен как это делает метод save() .

defer()

При сложной структуре данных модели могут содержать большое количество полей, некоторые из которых могут содержать большие объемы данных (например, TextField), или использовать ресурсоемкий процесс преобразования данных в объекты Python. Если вы точно знаете, что данные этих полей не будут использоваться при работе с результатами запроса, вы можете указать Django не выбирать эти поля из базы данных.

only()

Метод *only()* – противоположность метода defer(). Вызывайте его с полями, получение которых не должно быть отложено. Если у вас есть модель, почти все поля которой не должны выбираться из базы данных, используйте only(). Это сделает ваш код проще.

in_bulk()

Получает список первичных ключей и возвращает словарь ассоциирующий объекты с переданными ID.

При передаче в in_bulk() пустого списка будет получен пустой словарь.

```
>>> Blog.objects.in_bulk([1])
{1: <Blog: Beatles Blog>}
>>> Blog.objects.in_bulk([1, 2])
{1: <Blog: Beatles Blog>, 2: <Blog: Cheddar Talk>}
>>> Blog.objects.in_bulk([])
{}
```

When и Case

Для записи условного выражения применяется класс Case из модуля django.db.models. Его конструктор вызывается в формате:

```
Case(<условие_1>, <условие_2>, ..., <условие_n>[,
default=None][, output_field=None] )
```

Каждое условие записывается в виде экземпляра класса When, чей конструктор имеет вид

```
When(<условие>, then=None)
```

Union, intersection, difference

Uses SQL's UNION operator to combine the results of two or more QuerySets. For example:

```
>>> qs1.union(qs2, qs3)
```

The UNION operator selects only distinct values by default. To allow duplicate values, use the all=True argument.

union(), intersection(), and difference() return model instances of the type of the first QuerySet even if the arguments are QuerySets of other models. Passing different models works as long as the SELECT list is the same in all QuerySets (at least the types, the names don't matter as long as the types in the same order). In such cases, you must use the column names from the first QuerySet in QuerySet methods applied to the resulting QuerySet. For example:

Union, intersection, difference

Всё просто как апельсин:

- **union** -- объединение;
- **intersection** -- пересечение;
- **difference** --- разность.

```
union(*other_qs, all=False)
>>> qs1.union(qs2, qs3)
```

select_for_update()

Возвращает *queryset*, который лочит строки до окончания транзакции, генерируя SELECT ... FOR UPDATE SQL-выражение.

Например:

```
from django.db import transaction

entries =
Entry.objects.select_for_update().filter(author=requ
est.user)
with transaction.atomic():
    for entry in entries:
        ...
```

Вложенные запросы Subquery, Exists

Вложенные запросы могут использоваться в условиях фильтрации или для расчёта результатов у вычисляемых полей.

1) Subquery

```
from django.db.models import OuterRef, Subquery
newest =
Comment.objects.filter(post=OuterRef('pk')).order_by('-created_at')
Post.objects.annotate(newest_commenter_email=Subquery(newest.values('email'
)[:1]))
```

2) Exists

```
from django.db.models import Exists
Company.objects.filter(
            Exists(Employee.objects.filter(company=OuterRef('pk'),
salary__gt=10))
)
```

ТЕХНОТРЕК

Django Rest Framework (DRF)

Что такое Django Rest Framework?

Django Rest Framework (DRF) — это библиотека, которая работает со стандартными моделями Django для создания гибкого и мощного API для проекта.

Состоит из 3-х слоёв:

1. **Сериализатор**: преобразует информацию, хранящуюся в базе данных и определённую с помощью моделей Django, в формат, который легко и эффективно передается через API.
2. **Вид** (ViewSet): определяет функции (CRUD - чтение, создание, обновление, удаление), которые будут доступны через API.
3. **Маршрутизатор**: определяет URL-адреса, которые будут предоставлять доступ к каждому виду.

Что такое Django Rest Framework?

Django Rest Framework (DRF) — это библиотека, которая работает со стандартными моделями Django для создания гибкого и мощного API для проекта.

Устанавливаем библиотеку в виртуальном окружении.
```
pip install djangorestframework
```

Работа с DRF. Шаг 1

В settings.py в переменную INSTALLED_APPS добавляем строку вида:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    ...
    'rest_framework'
]
```

Работа с DRF. Шаг 2

Создаём класс вьюхи, унаследованный от APIView (views.py).

```
from rest_framework.response import Response
from rest_framework.views import APIView
from .models import Chat

class ChatView(APIView):
    def get(self, request):
        chats = Chat.objects.all()
        return Response({"chats": chats})
```

Работа с DRF. Шаг 3

Создаём класс сериалайзера в **отдельном** файле (chats/serializers.py).

```
from rest_framework import serializers

class ChatSerializer(serializers.Serializer):
    title = serializers.CharField(max_length=120)
```

Работа с DRF. Шаг 4

```
from rest_framework.response import Response
from rest_framework.views import APIView

from .models import Chat
from .serializers import ChatSerializer

class ChatView(APIView):
    def get(self, request):
        chats = Chat.objects.all()
        serializer = ChatSerializer(chats,
many=True)
        return Response({"chats": serializer.data})
```

Работа с DRF. Ясно. И что???

А ничего! На самом деле есть ещё класс **GenericAPIView**, который расширяет возможности APIView, добавляя в него часто используемые методы list и detail.

```
class ChatSerializer(serializers.ModelSerializer):
    class Meta:
        model = Chat
        fields = (‘title’, )

class ChatView(ListModelMixin, GenericAPIView):
    queryset = Chat.objects.all()
    serializer_class = ChatSerializer

    def get(self, request, *args, **kwargs):
        return self.list(request, *args, **kwargs)
```

ТЕХНОТРЕК

Репликации

Репликация

Репликация (replication) - хранение копий одних и тех же данных на нескольких машинах.

Репликация

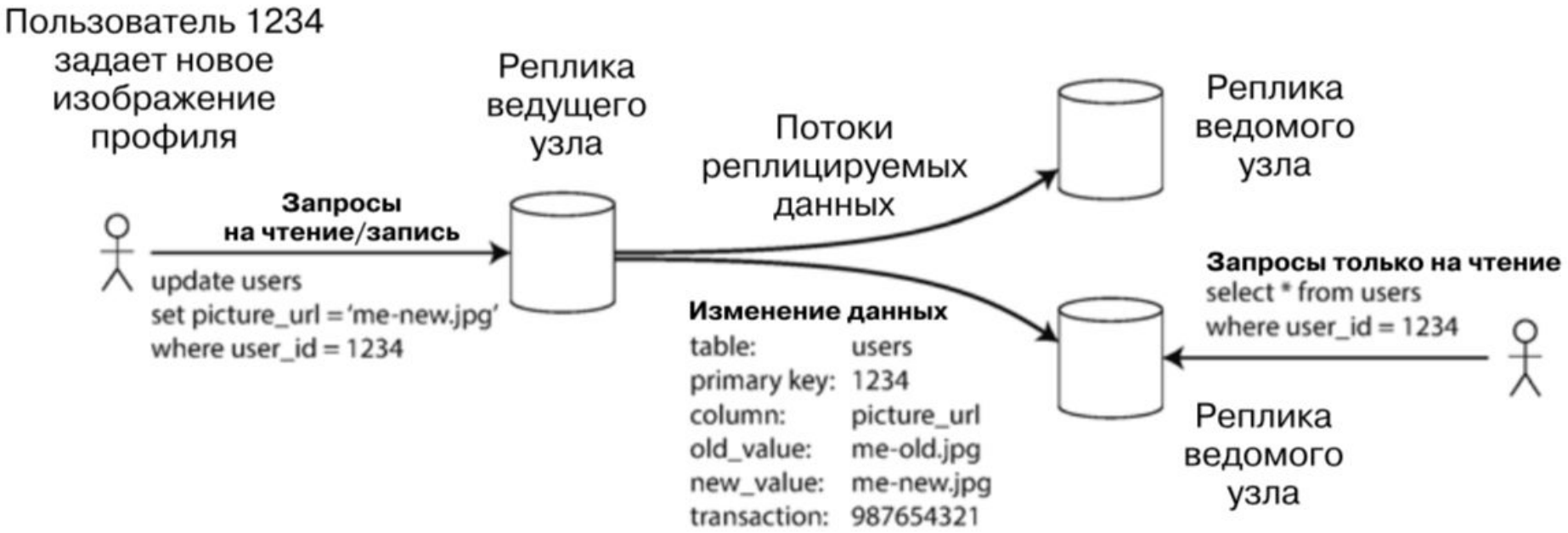
1. Ради хранения данных географически близко к пользователям (и сокращения, таким образом, задержек);
2. Чтобы система могла продолжать работать при отказе некоторых её частей (и повышения, таким образом, доступности);
3. Для горизонтального масштабирования количества машин, обслуживающих запросы на чтение (и повышения, таким образом, пропускной способности по чтению.

Виды репликации

Если реплицируемые данные не меняются с течением времени, то репликация не представляет сложности: просто нужно однократно скопировать их на каждый узел и всё. Основные сложности репликации заключаются в том, что делать с изменениями реплицированных данных.

1. С одним ведущим узлом (single-leader);
2. С несколькими ведущими узлами (multi-leader);
3. Без ведущего узла (leaderless).

С одним ведущим узлом (single-leader)



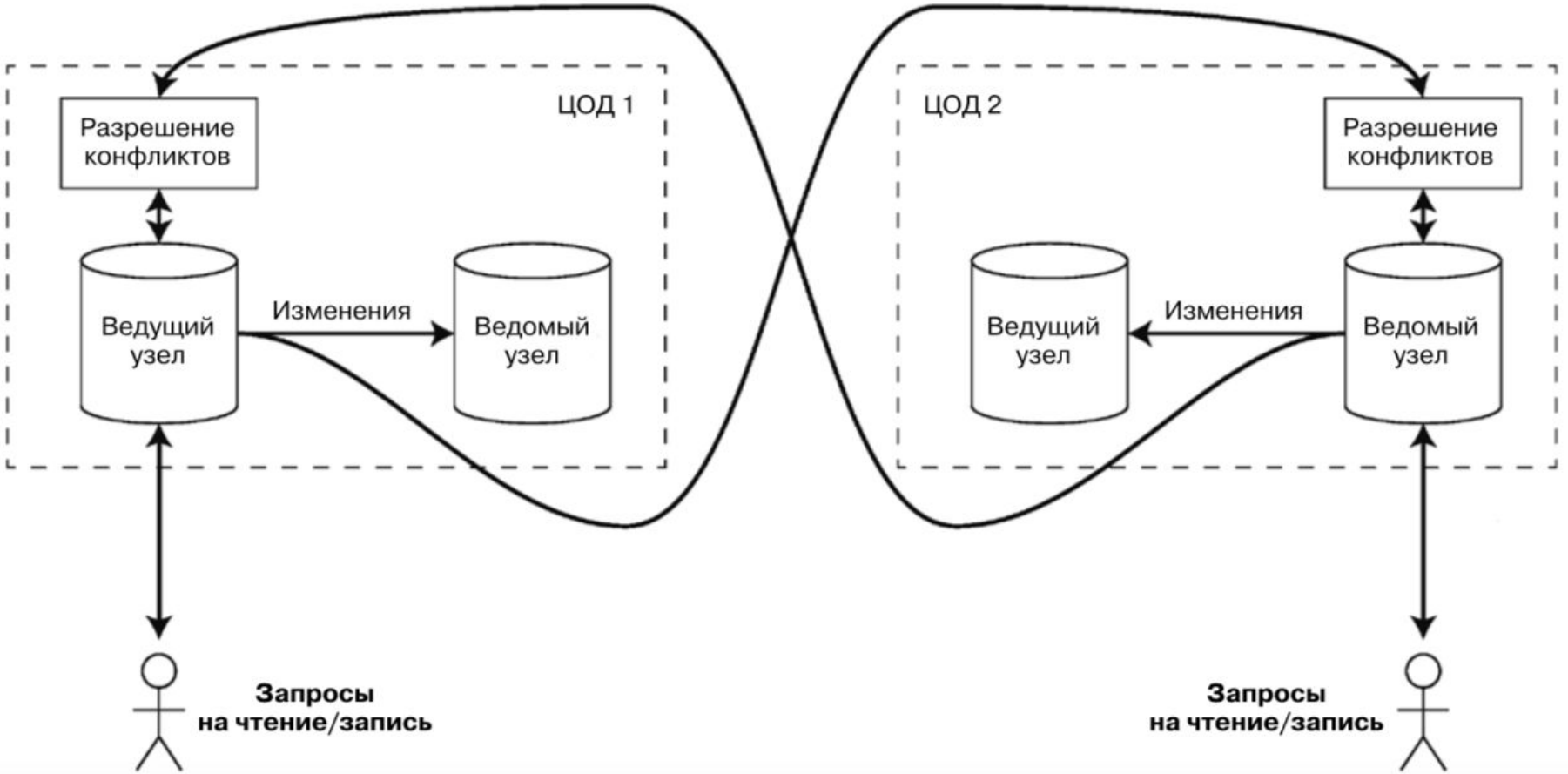
С одним ведущим узлом (single-leader)

1. Одна из реплик назначается ведущим (leader) узлом;
2. Другие реплики называются ведомыми (followers) узлами;
3. Когда клиенту требуется прочитать данные из базы, он может выполнить запрос или к ведущему узлу, или к любому из ведомых. Однако запросы на запись раз- решено отправлять только ведущему;

Где применяется

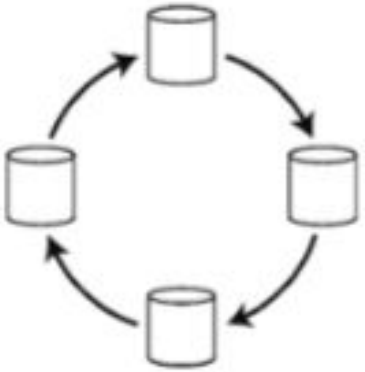
1. MySQL, PostgreSQL (начиная с версии 9.0), Oracle Data Guard;
2. MongoDB, RethinkDB и Espresso;
3. Kafka, RabbitMQ.

С несколькими ведущими узлами (multi-leader)

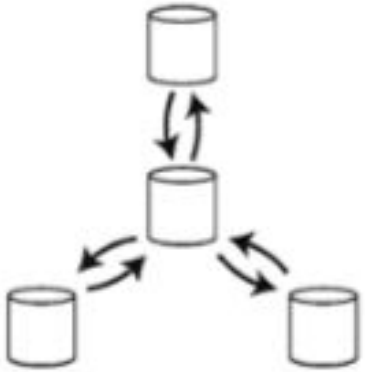


Топологии репликации с несколькими ведущими узлами

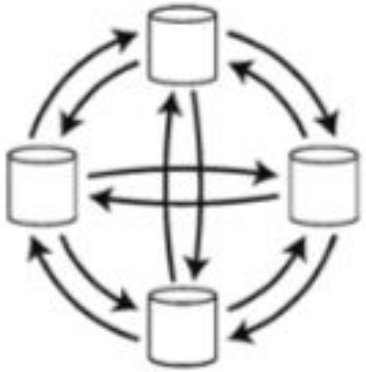
Топология репликации (replication topology) описывает пути, по которым операции записи распространяются с одного узла на другой.



А. Топология типа «кольцо»



Б. Топология типа «звезда»



В. Топология типа «каждый с каждым»

Без ведущего узла (leaderless)

Клиенты отправляют информацию о каждой из операций записи одному из нескольких узлов и читают из нескольких узлов параллельно, чтобы обнаружить узлы с устаревшими данными и внести поправки.

ТЕХНОТРЕК

Шардирование

Что это такое???

В случае очень больших наборов данных или объёмов обрабатываемой информации репликаций недостаточно: необходимо разбить данные на секции (partitions), иначе говоря, выполнить **шардинг** (sharding) данных. **Секционирование** (partitioning), представляет собой способ умышленного разбиения большого набора данных на меньшие.

Когда нужно применять шардирование?

1. Когда функциональное разбиение и репликация не помогают;
2. Разбиваем данные на маленькие кусочки и храним на многих серверах;
3. “Единственное” решение для крупного масштаба;
4. Нужно аккуратное планирование.

Подходы к секционированию

1. Секционирование по диапазонам значений ключа (ключи сортируются и секция содержит все ключи, начиная с определенного минимума до определенного максимума);
2. Хеш-секционирование (вычисляется хеш-функция каждого ключа и к каждой секции относится определенный диапазон хешей);

Домашнее задание №2

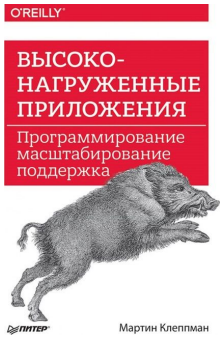
1. Написать (**НЕ переписать! Это означает, что старые views нужно оставить**) новые view, работающие с DRF. Использовать нужно ModelViewSet, ModelSerializer, DefaultRouter.
2. Написать скрипт, делающий бекап базы данных, с ротацией этих бекапов за *n* последних вызовов; *n* задаётся через конфиг.

Срок сдачи

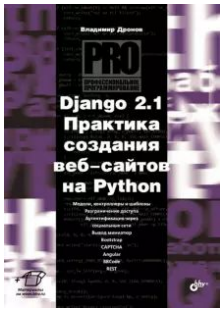
Сроков нет, но вы держитесь

Полезные ссылки

Высоконагруженные приложения. Программирование, масштабирование, поддержка | Клеппман Мартин



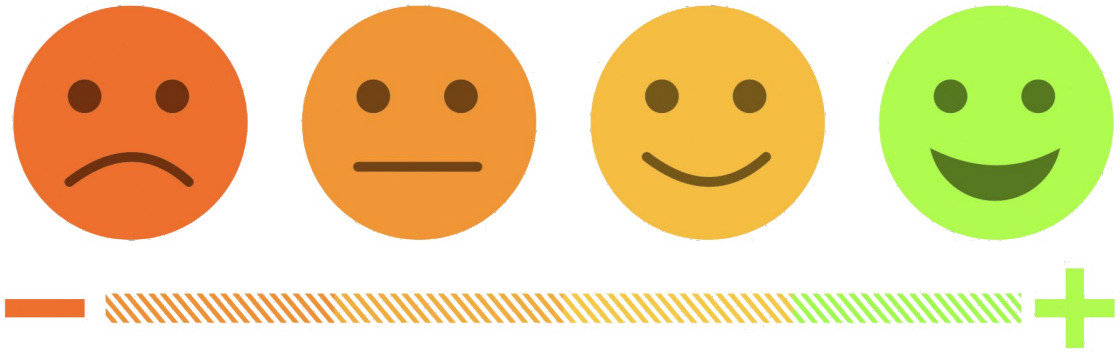
Django 2.1. Практика создания веб-сайтов на Python | Дронов Владимир Александрович



Для саморазвития (опционально)
Чтобы не набирать двумя пальчиками

Обратная связь

Не забудьте поставить оценки!!!!



ТЕХНОТРЕК

Спасибо за внимание!

Антон Кухтичев

a.kukhtichev@corp.mail.ru