Совместное использование хранимых процедур Postgresql и ORM
на примере Django

Камиль Исламов, АО «Троник»

План

- Сравнение некоторых особенностей SQL функций и ORM

- Возможности Django админ-панели

- Реализация бизнес-логики с «оглядкой» на Django

- Логика приложения через SQL функции

- Итоги

Некоторые преимущества ORM

- Можно работать с БД не зная SQL
- Структурирует программный код
- Унифицирует обращение к БД
- Предоставляет возможность «забывать» о БД
- Лёгкая миграция между разными СУБД
- Простота обучения и внедрения
- Надёжность, проверенная временем

Некоторые преимущества хранимых процедур

- Выигрыш производительности сложных запросов

- Минимизация обращений к БД
- Упрощение программного кода

- Использование спецефических преимуществ СУБД

- Удобное «выкатывание» обновлений
- Изменения в логике не требуют изменения программного кода

- Надёжность, проверенная временем ☺

Хранимые процедуры VS ORM

Особенности	Хранимые процедуры	ORM или аналоги
Оптимальное составление сложных запросов	●	-
Возможность разрабатывать без знания SQL	-	●
Обновления без перезапуска сервера	●	-
Простая смена СУБД	-	●
Использование преимуществ СУБД	●	-
Универсальность для любых задач	-	-

Примеры оптимального применения

Описание кейса	Хранимые процедуры	ORM или аналоги
Информационные системы с функциональной бизнес-логикой, использующей сложные взаимосвязи данных	●	-
Функциональные нетипичные проекты с множеством неочевидных взаимосвязей между сущностями	●	-
Контентные проекты, где много независимых друг от друга сущностей без сложных взаимосвязей	-	●
Проекты с множеством унифицированных справочников, с возможностью использовать единый интерфейс работы с данными	-	●
Универсальность для любых задач	-	-

Возможности Django админ-панели

- Больной выбор *шаблонов* с адаптивной вёрсткой

- Множество готовых *дэшбоардов*

- Встроенная система пользователей

- Вспомогательные плагины солидного сообщества (пикеры, экспорт, …)

- Фильтры, сортировки, инлайны, иерархия сущностей, меню и т.д.

- Журнал изменения данных и других действий пользователя

- В совокупности – это *мощная платформа* для быстрого запуска

Django Admin: шаблоны и пользователи

- Достойный набор *шаблонов* с адаптивной вёрсткой
  - Стандартный дизайн админ-панели
  - Django Suit – один из наиболее популярных, встроенные внешние и плагины
  - Django Jet – альтернативный, со встроенным дэшбоардом

- Множество готовых *дэшбоардов*

- Встроенная система пользователей
  - Безопасная, удобная и надёжная

Django Admin: готовый функционал

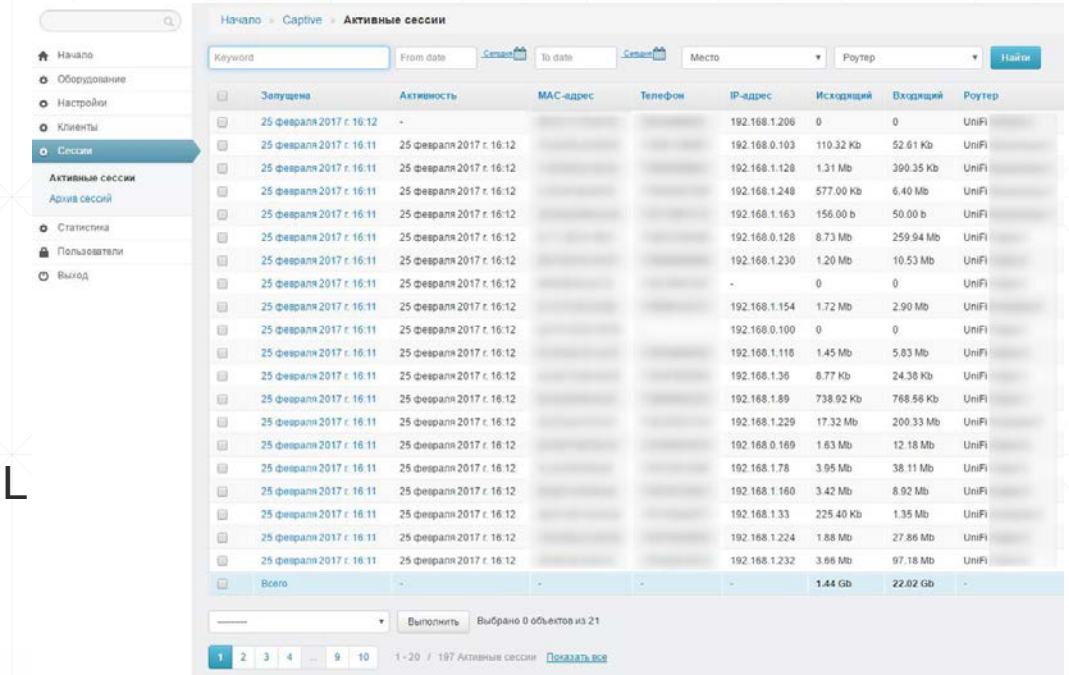
- Дополнительные плагины большого сообщества
  - Встроенные и кастомные комбобоксы
  - Собственные и модифицированные редакторы даты/времени
  - Экспорт в Excel, CSV и другие форматы

- Готовые инструменты работы с данными
  - Фильтры, быстрый поиск по множеству полей
  - Сортировки, дочерние объекты
  - Редактор многоуровневого меню, в т.ч. на основе сущностей

- Встроенная система журналирования
  - Протокол изменения данных и других действий пользователя

Web-приложение как Django Admin-панель

- Табличный интерфейс приложения
  - Таблица СУБД – это *Модель* Django
  - Каждая запись – это стандартная HTML форма
- Admin панель Django
  - Список моделей (таблиц) с кастомными наименованиями
  - Отображение таблицы (модели) в режиме пагинации с поиском и фильтрами
  - Редактирование записи с Foreign-комбобоксами, встроенной в *Модель* проверкой целостности данных



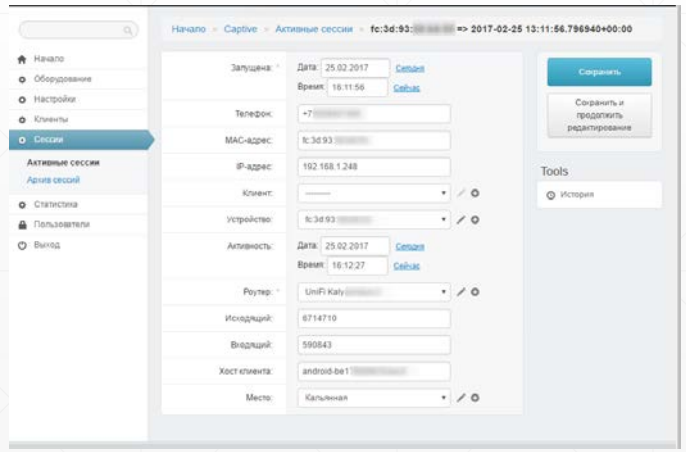


Схема данных Django в СУБД (по умолчанию)

- Таблица в БД → Django *Модель* (Python Class)
- Выборка из таблицы → Django *Set* (Python List)
- Строка таблицы → Django *Объект*
- Столбец строки → Django *Атрибут* объекта

Django Модель

ID	Name	Status_ID
1	Apple	111
2	Orange	222
3	Tomato	111

Django Объект

Атрибут Объекта

[{**id**: 1, **name**: Apple, **status**: Fresh},
{**id**: 2, **name**: Orange, **status**: Dried}]

– Django *Set*

Некоторые связи Django в СУБД

Тип Django	СУБД	Поле В СУБД	Атрибут объекта
Foreign Key	Внешний ключи	status_**id**	status
One-to-One	Внешний ключ + Уникальность	one2one_**id UNIQUE**	one2one
Many-to-Many	Таблица с двумя внешними ключами	many2many.dict1_id, many2many.dict2_id	many2many

Варианты интеграции с Django

- Задействование системы пользователей

- Классические справочники

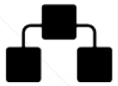
- Частичное изменение данных

- Только для чтения: таблицы

- Только для чтения: представления

Взаимодействие с Django-auth

- Добавление собственных полей в *django_users*
  - Информация, фотографии, и т.д.
  - Foreign Key на сущности бизнес-логики



- Использование Foreign Key на *django_users.id*
  - Принадлежность сущностей к пользователям
  - Дополнительный функционал привилегий

- Самостоятельные сущности с одинаковым *id*
  - Отдельная структура с собственной иерархией
  - Множество различной информации о пользователе

Модели Django как справочники

- Таблица полностью управляется Django
- В каких случаях удобно использовать:

  - Вписывается в понятие «то, что выпадает в комбобоксе»
  - Изменение данных не критично для бизнес-логики

  - Требуется «ручная» работа с данными
- Преимущества:

  - Удобный встроенный инструментарий редактирования
  - Целостность данных можно обеспечить в описании Модели

  - Наглядность и простота обработки данных

Частичное применение Модели Django

- Django управляет некоторыми полями Таблицы
- В каких случаях удобно использовать:

  - Элементы бизнес-логики, с которыми взаимодействуют (клиенты, и т.д.)
  - Изменение некоторых полей не критично для бизнес-логики

  - Требуется возможность «ручной» работы с данными
- Преимущества:

  - Привычный инструментарий редактирования
  - Встроенный поиск, фильтры, вложенные дочерние объекты

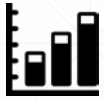
  - Критичные поля можно скрыть или запретить изменять

Модели Django как таблица для чтения

- Модель Django как Таблица с запретом редактирования всех полей

- Встроенный удобный поиск, фильтрация, сортировка

- Просмотр элементов бизнес-логики в привычном инструментарии
- Наглядный доступ к спискам дочерних объектов



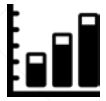
- Встроенные библиотеки визуализации

- Готовые выгрузки в Excel, CSV и другие популярные форматы

Модели Django как Представление

- Модель Django ассоциируется с любой View и её полями

- Эффективный аналитический инструментарий
  - статистика, «онлайн», архивы, группировка и т.д.

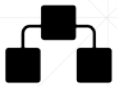


- Встроенные библиотеки визуализации

- Готовые выгрузки в Excel, CSV и другие популярные форматы

Логика приложения вне Django-Admin

- Сервер приложений обеспечивает запуск функций
  - Встроенная система маршрутов



  - Запускается соответствующий маршруту Django-метод
  - В каждом Методе выполняется одна транзакция с одной функцией

- Один POST/GET запрос – один вызов соответствующей функции

- Реализация самой функции
  - Обработка данных таблиц, не являющихся Django-моделями
  - Обновление полей, скрытых или неизменяемых в Django-моделях

Логика приложения внутри Django Admin-панели

- Интерфейс редактирования:
  - стандартная HTML форма
  - ограничения модели в виде CONSTRAINT’ов
- Изменение объектов:
  - Таблицы типа Справочник – ORM операции INSERT, UPDATE, DELETE
  - Частичное редактирование – ORM или функция
    - ORM – если допустимо «ручное» изменение некритичных полей
    - возможные параметры к вызову функции – это атрибуты объекта
  - Таблицы для чтения или Представление – замена функцией или запрет на изменение всех полей

Преимущества концепции “*Django+SQL функции*”

- Сохранение преимуществ реализации бизнес-логики через функции

- Полноценное использование возможностей Django Admin-панели

- Дополнительные возможности Админ-панели при:
  - использовании *представлений* как Django-модели

  - использовании *SQL-функций* вместо ORM

- Экономия ресурсов на разработку при оптимальной эффективности

Вопросы

- kam@kambox.ru
- ikamil
- kamil.islamov

Совместное использование хранимых процедур Postgresql и ORM на примере Django