Введение в базы данных: PostgreSQL

Антон Кухтичев

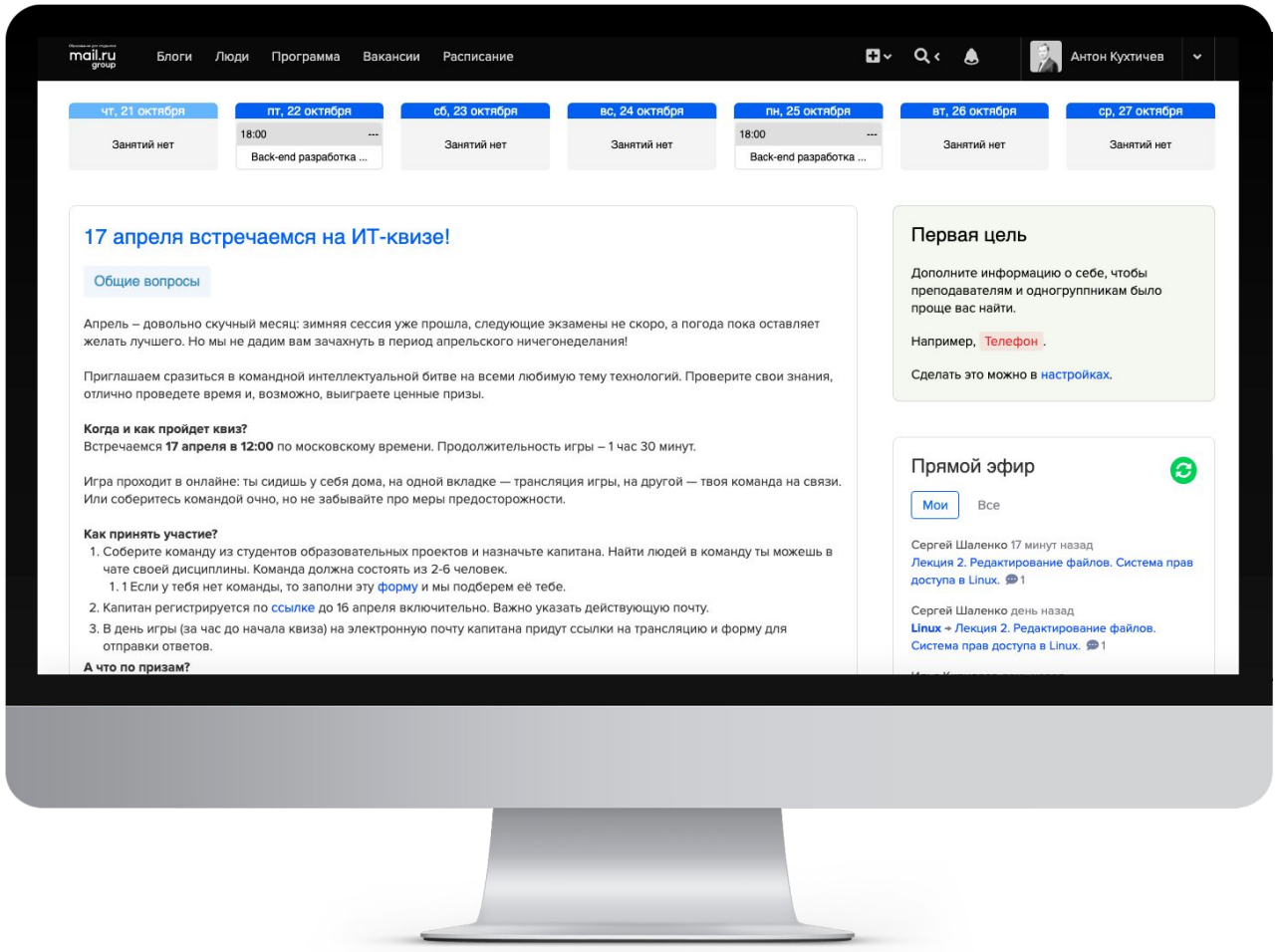
образование

Содержание занятия

- Квиз #4
- PostgreSQL
- Язык SQL
- Django и PostgreSQL
- Домашнее задание

Напоминание отметиться на портале

и оставить отзыв после лекции



Квиз #4

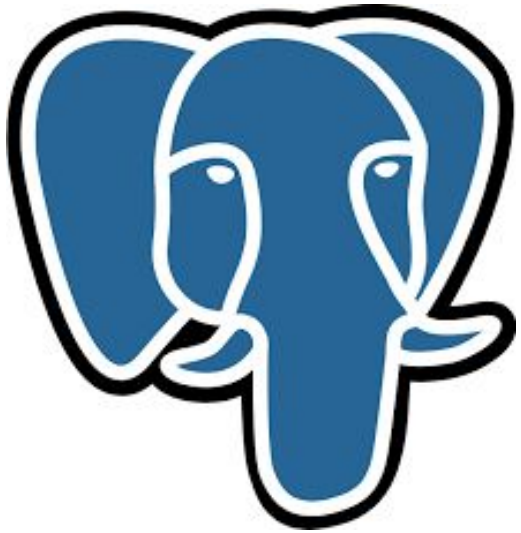
https://forms.gle/qw8GMQDBtqvFT4p28

PostgreSQL

Клиент psql: установка, запуск, создание БД и новых пользователей.

Установка PostgreSQL

```
# Установка на Ubuntu
sudo apt install postgresql-10
# Установка на MacOS
brew install postgresql@10
# Проверка подключения
sudo -u <USER_NAME> psql
```



Сильные стороны PostgreSQL

- надежность и устойчивость;
- превосходная поддержка;
- кроссплатформенность;
- доступность;
- расширяемость.

Пользователи и базы из коробки

Пользователи

- `postgres` — супер-пользователь внутри Postgres, может все;

Базы данных

- `template1` — шаблонная база данных;
- `template0` — шаблонная база данных на всякий случай;
- `postgres` — база данных по-умолчанию, копия `template1`.

Пользователи и базы из коробки

Создать пользователя и базу:

```
postgres=# CREATE USER quack_db WITH password 's3cr3t';
CREATE ROLE
postgres=# CREATE DATABASE quack_db OWNER quack;
CREATE DATABASE
```

Проверить подключение

```
$ psql --host=localhost --user=quack quack
Password for user quack: *****
```

Использование psql

Подключение через UNIX сокет, имя пользователя совпадает с пользователем Linux:

```
$ psql db_name
```

Подключение через TCP сокет:

```
$ psql --host=127.0.0.1 --user=db_user db_name
```

Если вы хотите подключаться к своей базе без ввода пароля:

```
postgres=# CREATE USER your_linux_user;
postgres=# GRANT ALL ON DATABASE db_name TO your_linux_user;
```

Команды psql

- `\?` — показать список команд;
- `\du` — показать список пользователей с привилегиями;
- `\l` — показать список баз данных;
- `\c db_name2` — подключиться к другой базе данных;
- `\dt` — показать список таблиц;
- `\d table_name` — показать колонки таблицы
- `\x` — переключить режим вывода
- `\q` — выход из psql

Язык SQL

Типы данных, синтаксис, основные виды операции.

Типы данных

- `smallint`, `integer`, `bigint` — целое, 2/4/8 байт;
- `smallserial`, `serial`, `bigserial` — целое, 2/4/8 байт, автоувеличение;
- `timestamp` — дата и время, с точностью до микросекунд;
- `text` — строка произвольной длины;
- `varchar` — строка ограниченное длины;
- `char` — строка ограниченное длины, дополненная пробелами;
- `uuid` — UUID;
- `jsonb` — JSON документ;

Виды операций

- `SELECT` — выборка данных;
- `UPDATE` — обновление значений столбцов;
- `DROP` — удаление;
- `ALTER` — изменение;
- `INSERT` — вставка;
- `CREATE` — создание;
- `JOIN` — объединение;
  - `INNER JOIN` (или просто JOIN);
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `FULL OUTER JOIN`
  - `CROSS JOIN`

Создание таблиц

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    nick TEXT NOT NULL UNIQUE CHECK (length(nick) < 32),
    name TEXT NOT NULL CHECK (length(name) < 32)
)
```

Создание таблиц (2)

```
CREATE TABLE messages (
    message_id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(user_id),
    content TEXT NOT NULL CHECK (length(content) < 65536),
    added_at TIMESTAMP NOT NULL DEFAULT NOW()
)
```

Изменение и удаление таблиц

```
ALTER TABLE users
    ADD COLUMN avatar TEXT DEFAULT NULL,
    DROP CONSTRAINT users_name_check,
    ADD CONSTRAINT users_name_check
        CHECK (length(name) < 64);
DROP TABLE users;
DROP TABLE users CASCADE; -- не повторять в production :)
```

Добавление данных

```
INSERT INTO users (nick, name)
            VALUES  ('mort.rainey', 'Морт Рейни'),
                    ('john.shooter', 'Кокни Шутер');

INSERT INTO users VALUES (5, 'todd.downey', 'Тодд Дауни');
```

Обновление данных

```sql
UPDATE users SET name = 'Не такой как все'
WHERE user_id = 2;

DELETE FROM users WHERE user_id % 2 = 0;
```

Выборка данных

```
SELECT message_id, content, added_at::DATE
FROM messages
WHERE user_id = 3
  AND message_id < 100500
ORDER BY added_at DESC
LIMIT 10
```

Выборка из нескольких таблиц

```
SELECT nick AS author, name, messages.*
FROM messages
JOIN users USING (user_id)
WHERE user_id = 3
  AND message_id < 100500
ORDER BY added_at DESC
LIMIT 10
```

I KNOW SQL

http://www.sql-ex.ru

Django и PostgreSQL

ORM, как подключить БД к Django-приложению, создание и миграция моделей.

Типы полей (1)

- `IntegerField`
- `AutoField`
- `BooleanField`
- `CharField`
- `EmailField`
- `DateField`
- `DateTimeField`
- `FloatField`
- `TextField`

Типы полей (2)

- Один-к-одному → `OneToOneField`
- Один-ко-многим → `ForeignKey`
- Многим ко многим → `ManyToManyField`

ForeignKey

```
class Movie(models.Model):
    title = models.CharField()
    genre = models.ForeignKey(null=True, on_delete=models.CASCADE)
```

- `models.CASCADE` — при удалении жанра удаляются и фильмы с таким жанром;
- `models.PROTECT` — запрещает удалять жанры, пока есть фильмы с таким жанром;
- `models.SET_NULL` — фильмы останутся в БД даже при удалении жанра, но значение в поле genre у фильмов изменится на None.
- `models.SET_DEFAULT` — работает как SET_NULL, но вместо None выставляет значение по-умолчанию.

Как сделать своё поле с первичным ключём

Если по каким-либо причинам Вы не хотите, чтобы создавалось поле id, то надо создать своё с `primary_key=True`!

```
class Blog(models.Model):
    my_super_pk = models.AutoField(primary_key=True)
    ...
```

Параметры у поля в модели

`verbose_name` — человекочитаемое название поля;

`null` — может ли быть поле NULL в БД, по умолчанию — `False`

`blank` — может ли быть поле пустым в форме, по умолчанию — False

`choices` — выбор значения из списка;

`default` — значение по умолчанию;

`db_index` — нужно ли создать индекс в БД;

`help_text` — подробное описание поля;

`primary_key` — первичный ключ;

`unique` — уникальное ли поле;

`editable` — можно ли поле изменять;

Добавить базу данных в settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'quack_db',
        'USER': 'quack',
        'PASSWORD': 's3cr3t',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

Добавить базу данных в settings.py

```
# Делаем миграцию
./manage.py migrate

# Создаем суперпользователя
./manage.py createsuperuser

# и запускаем сервер
./manage runserver
```

Некоторые полезные опции

- `./manage.py dbshell` — запустить клиент базы данных;
- `./manage.py showmigrations` — показать историю миграций;
- `./manage.py makemigrations` — создать миграции;
- `./manage.py migrate` — применить миграции;
- `./manage.py sqlmigrate <app> <migration id>` — вывести SQL-код для миграции;
- `./manage.py shell` — запустить python shell;
- `./manage.py validate` — проверть структуру моделей.

Откат до предыдущей миграции

```
# В общем случае команда выглядит так:
$ ./manage.py migrate <ваше приложение> <название предыдущей миграции>
# Откат всех миграций!
$ ./manage.py migrate <ваше приложение> zero
# Примерчик:
$ ./manage.py showmigrations movies
movies
 [X] 0001_initial
 [X] 0002_movie_genre
 [X] 0003_auto_20201005_1456
$ ./manage.py migrate movies 0002_movie_genre
```

Создание пустой миграции

```
$ ./manage.py makemigrations --empty <ваше приложение>
```

```
from django.db import migrations


class Migration(migrations.Migration):
    dependencies = [
        ('yourappname', '0001_initial'),
]
    operations = [
        migrations.RunPython(forwards_func, reverse_func),
]
```

Еще немного про миграции

```
def forwards_func(apps, schema_editor):
    SomeModel = apps.get_model('some_app', 'SomeModel')
    SomeModel.objects.all().update(field=True)

def reverse_func(apps, schema_editor):
    SomeModel = apps.get_model('some_app', 'SomeModel')
    SomeModel.objects.all().update(field=None)
```

SQL-запросы напрямую из Django (1)

```
from django.db import connection

def my_custom_sql(self):
    with connection.cursor() as cursor:
        cursor.execute("SELECT foo FROM bar WHERE baz = %s", [self.baz])
        row = cursor.fetchone()

    return row
```

SQL-запросы напрямую из Django (2)

```
>>> people = Person.objects.raw('SELECT *, age(birth_date) AS
age FROM myapp_person')
>>> for p in people:
...     print(f"{p.first_name} is {p.age}.")
```

Домашнее задание

Домашнее задание #5

- Установить Postgres, создать нового пользователя и БД и настроить доступ;
- Спроектировать базу данных проекта, подготовить модели и мигрировать их в БД.
- Необходимо реализовать одно из отношений:
  - Один-ко-многим;
  - Многие-ко-многим;
  - Один-к-одному.

Смотри подробнее файл `homework.md` в репозитории с лекциями!

Полезная литература

- Документация Django
- Документация PostgreSQL

- Для саморазвития (опционально)

  Чтобы не набирать двумя пальчиками

Спасибо за внимание!

Вопросы?