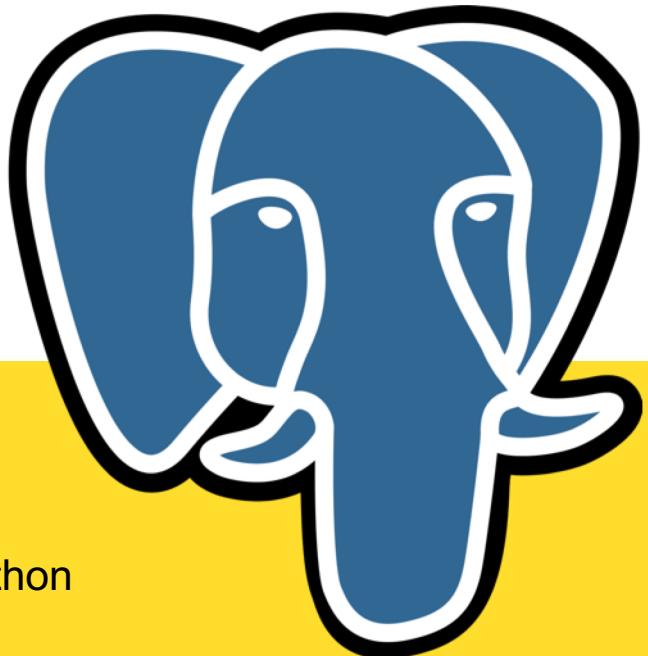


PostgreSQL

Что, почему, как поднять и подружить с python



Tinkoff.ru



Базы данных нужны всем

Не думайте, что это не про вас!

Что такое PostgreSQL?



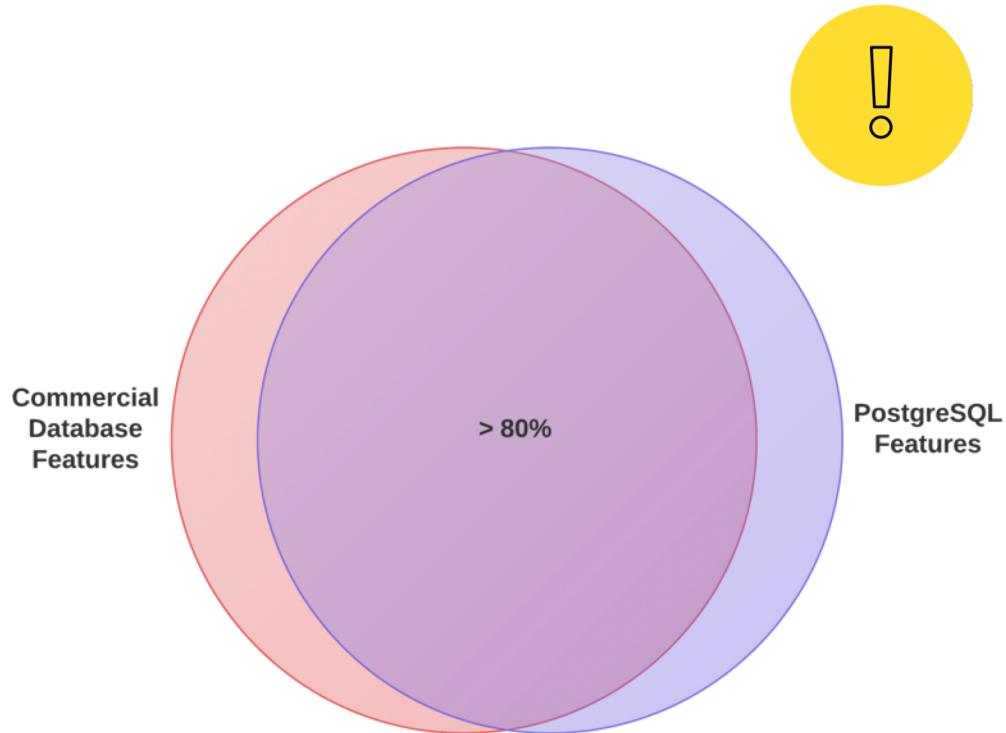
- PostgreSQL (произносится «Пост-Грэ-Эс-Кью-Эл») — свободная объектно-реляционная система управления базами данных (СУБД)
- высокопроизводительные и надёжные механизмы транзакций и репликации;
- расширяемая система встроенных языков программирования:
 - в стандартной поставке поддерживаются [PL/pgSQL](#), [PL/Perl](#), [PL/Python](#) и [PL/Tcl](#);
 - дополнительно можно использовать [PL/Java](#), [PL/PHP](#), [PL/Py](#), [PL/R](#), [PL/Ruby](#),
[PL/Scheme](#), [PL/sh](#) и [PL/V8](#)
 - а также имеется поддержка загрузки [C](#)-совместимых модулей;
- легкая расширяемость.



И что же в ней хорошего?



- Отличная серверная часть
- Поддержка кучи языков
- Hot Standby
- High Availability
- Онлайн бэкапы
- Point In Time Recovery
- Table Partitioning Spatial
- Functionality Full Text Search



И что же в ней хорошего?



- Object Level Privileges assigned to “Roles & User”
- Row Level Security
- Many Authentication mechanisms
 - Kerberos LDAP PAM GSSAPI
- Native SSL Support.
- Data Level Encryption (AES, 3DES, etc)



А еще?



- No Vendor Lock-in
- Отличная поддержка ANSI SQL standard
- Работает на любом утюге (есть сборки под много платформ, исходники открыты и доступны)
- “BSD-like” license – PostgreSQL License
- Крутое Open Source сообщество
 - Независимое (и отзывчивое) сообщество + есть митапы
 - 10+ committers and ~200 reviewers 1,500 contributors and 10,000+ members
- Миллионы скачиваний в год





Числовые типы

- integer
- bigint
- decimal
- numeric
- real
- double

Символьные типы

- varchar(n) variable-length with limit
- char(n) fixed-length, blank padded
- text variable unlimited length

Date/Time Types

- timestamp with timezone
- date
- time without timezone
- interval

Специальные типы

- smallserial
- serial
- bigserial cidr
- inet
- macaddr
- bytea

KV типы

- Xml
- hstore stores sets of key/value pairs
- json
- jsonb



Пользователи PostgreSQL



Коротко – Все



Тинькофф



Яндекс



Instagram





Немного теории



- Atomic – атомарность
Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично (Все или ничего)
- Consistent – консистентность
«Дай мне консистентный слепок данных»
- Isolated – изоляция
Во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат.
- Durable – Устойчивость
Если я говорю 'COMMIT;', то данные точно записались в БД и будут там пока я явно их не удалю





- Все или ничего
- У транзакции есть:
 - Начало (BEGIN;)
 - Работа (тонны кода SQL, в том числе INSERT / UPDATE / DELETE)
 - Окончание (END;) Вариантов ровно 2:
 - COMMIT; (сохранить все)
 - ROLLBACK; (откатить все изменения)
 - Когда транзакция окончена или ВСЕ изменения СОХРАНЯЮТСЯ или все изменения откатываются, третьего не дано. (если все работает ОК)



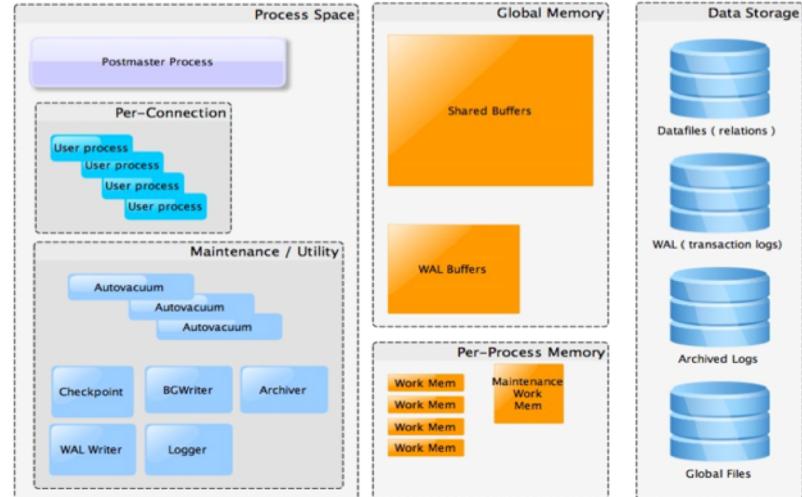
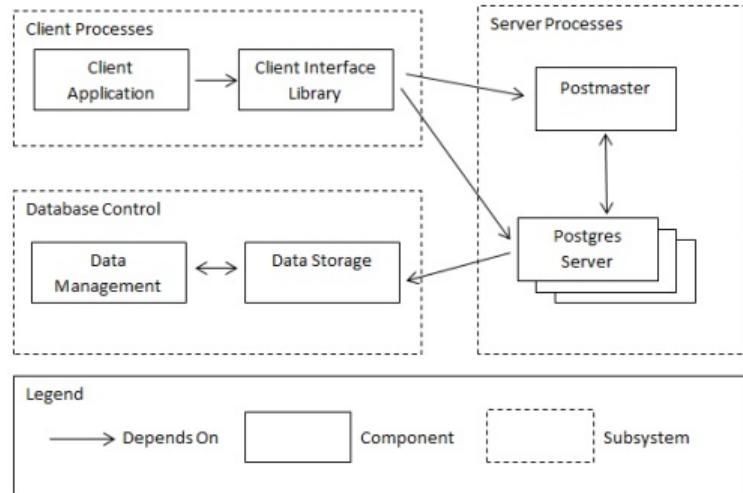


Как оно работает?

Но без подробностей

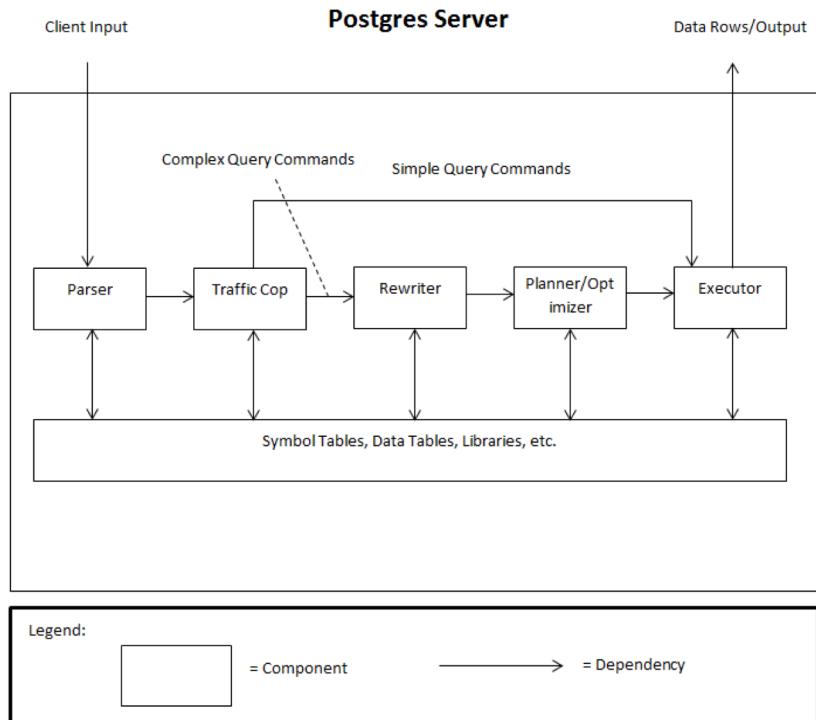


Взаимодействие с клиентом





Обработка запроса



Парсер

- Принимает SQL-запрос
- *lexer* раскладывает запрос на знакомые ему паттерны.
- *parser* строит из них дерево
- *parser* проверяет валидность SQL-запроса
- *Traffic cop* отправляет простые команды экзекьютору и сложные планировщику/оптимизатору

Планировщик/Оптимизатор

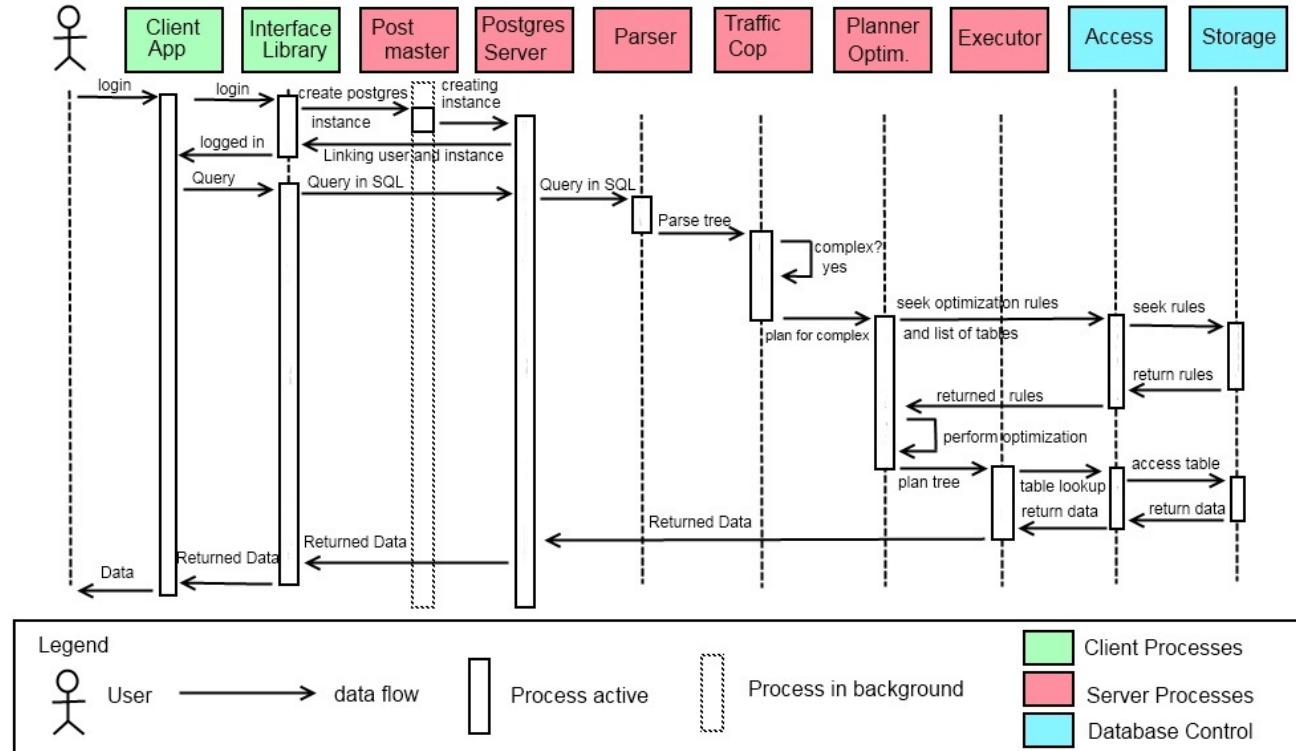
- SQL-запросы могут быть выполнены множеством способов, возвращая один и тот же результат
- Планировщик/Оптимизатор выберет лучший путь выполнения запроса из возможных (но это не точно)

Executor

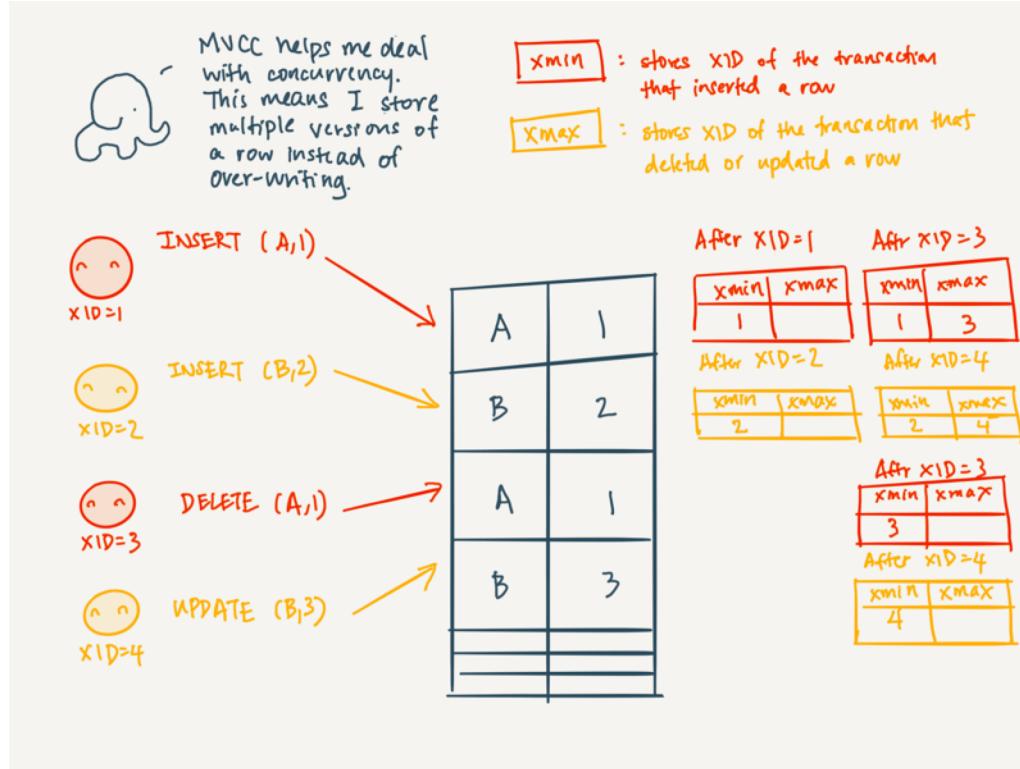
- Получает план выполнения от оптимизатора
- Получает данные из нужных таблиц
- Рекурсивно проходит по плану и выполняет необходимые действия на каждом узле дерева плана
- Возвращает данные клиенту



А теперь тоже самое в одной картинке



MVCC или какой там еще vacuum?





Установка и настройка



- Готовые пакеты — предпочтительный способ
<https://www.postgresql.org/download/>
- входит в базовые репозитории дистрибутивов ОС GNU/Linux
(но там могут быть старые версии)
- пакеты RPM, DEB FreeBSD, OpenBSD пакеты из Ports and Packages Collection Mac OS X Windows



Необходимое ПО



- Обязательно tar, gzip/bzip2, GNU make, компилятор Си (C89)
Используются, но можно отключить библиотеки GNU
Readline, zlib
- Дополнительно языки программирования Perl, Python, Tcl
для использования PL/Perl, PL/Python, PL/Tcl Kerberos,
OpenSSL, OpenLDAP, PAM для аутентификации и
шифрования
- Отдельные инструменты при сборке из репозитория git





- sudo yum install postgresql-server
- Или скачать более свежий rpm и установить его



Конфигурация



- Postgresql.conf Основной файл конфигурации читается один раз при старте сервера если параметр указан несколько раз, применяется последнее значение
 - при сборке по умолчанию — в каталоге с данными (PGDATA)
 - При изменении файла надо перечитать: \$ pg_ctl reload или в psql => select pg_reload_conf();
 - изменения некоторых параметров требуют полного перезапуска сервера
- Pg_hba.conf – настройка доступа к серверу БД и отдельным базам
- Полный список параметров сервера БД искать тут:
<https://postgrespro.ru/docs/postgresql/10/runtime-config.html>



Postgresql.conf highlights



1. shared_buffers = 2GB
2. work_mem = 128MB
3. maintenance_work_mem = 1GB
4. effective_cache_size = 4GB # ~ 2/3 RAM
5. max_prepared_transactions = 0 # zero disables the feature
6. listen_addresses = '*' - слушать на всех интерфейсах
7. max_connections = 50 - число пользовательских соединений
8. superuser_reserved_connections = 3



Postgresql.conf highlights



- logging_collector = on
- log_directory = 'pg_log'
- log_filename = 'postgresql-%a.log'
- log_truncate_on_rotation = on
- log_rotation_age = 1d
- log_rotation_size = 0
- log_line_prefix = '%t db=%d user=%u '
- log_statement = 'all'
- autovacuum = on
- log_autovacuum_min_duration = 0 # -1 disables,
0 logs all actions and
- autovacuum_max_workers = 3
- autovacuum_naptime = 10min
- autovacuum_vacuum_threshold = 1800
- autovacuum_analyze_threshold = 900
- #autovacuum_vacuum_scale_factor = 0.2 #
fraction of table size before vacuum
- #autovacuum_analyze_scale_factor = 0.1 #
fraction of table size before analyze



Конфигурация





Pg_hba.conf



local	база пользователь метод-аутентификации [параметры-аутентификации]
host	база пользователь адрес метод-аутентификации [параметры-аутентификации]
hostssl	база пользователь адрес метод-аутентификации [параметры-аутентификации]
hostnossal	база пользователь адрес метод-аутентификации [параметры-аутентификации]
host	база пользователь IP-адрес IP-маска метод-аутентификации [параметры-аутентификации]
hostssl	база пользователь IP-адрес IP-маска метод-аутентификации [параметры-аутентификации]
hostnossal	база пользователь IP-адрес IP-маска метод-аутентификации [параметры-аутентификации]

TYPE DATABASE USER ADDRESS METHOD

local all all trust

То же, но для локальных замкнутых подключений по TCP/IP.

TYPE DATABASE USER ADDRESS METHOD

host all all 127.0.0.1/32 password

local all @admins,+support md5

host all +group 0.0.0.0/0 ldap ldapserver="ldapsrv.company.name" ldapport="389" ldapprefix="EVIL_CORP\" ldapsuffix=""



Полезное – pg_stat_statements

```
psql> CREATE EXTENSION pg_stat_statements
```

```
$ psql dbname
dbname=# \x Расширенный вывод включен.
dbname = select * from pg_stat_statements;
userid | 10
dbid | 16388
query | SELECT "words".* FROM "words" WHERE "words".
"id" = ? LIMIT ?
calls | 27
total_time | 0.277
rows | 27
shared_blk_hit | 76
shared_blk_read | 6
shared_blk_dirtied | 0
shared_blk_written | 0
local_blk_hit | 0
local_blk_read | 0
```

```
local_blk_dirtied | 0
local_blk_written | 0
temp_blk_read | 0
temp_blk_written | 0
blk_read_time | 0.051
blk_write_time | 0
```



- Терминальный клиент для работы с PostgreSQL
- Поставляется вместе с СУБД
- Используется для интерактивной работы и выполнения скриптов
- Запуск

```
$ psql -d база -U роль -h узел -р порт
```



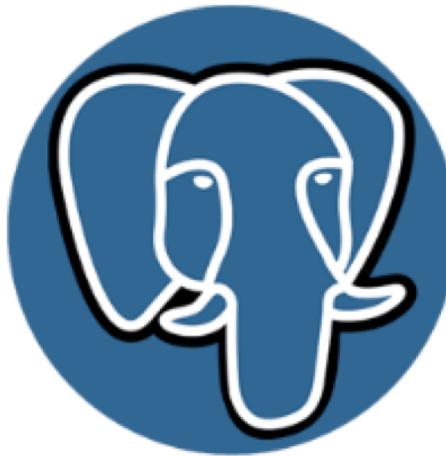
- Новое подключение в psql => \c[onnect] база роль узел порт
- Информация о текущем подключении => \conninfo



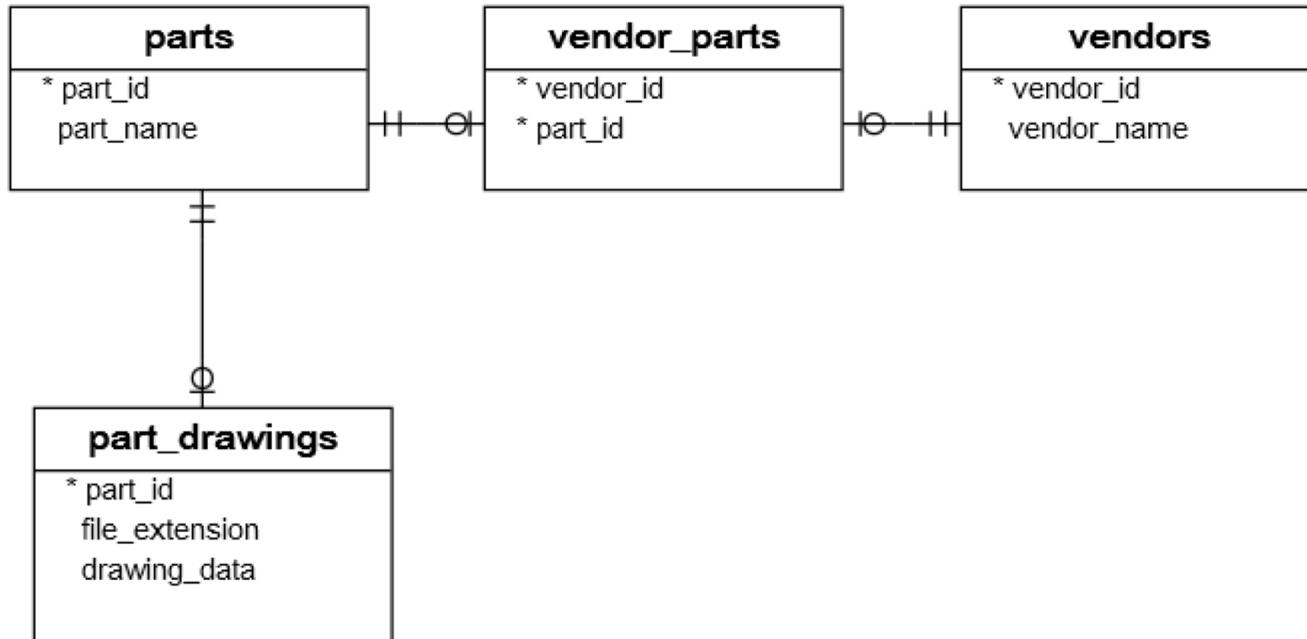
- В командной строке ОС \$ psql --help \$ man psql
- В psql
 - => \?
 - список команд psql => \? variables переменные
 - psql => \h[elp] список команд SQL
 - => \h команда синтаксис команды SQL
 - => \q выход
- Полезное:
 - ~/.psqlrc
 - ~/.psql_history поиск по истории с помощью readline количество хранимых команд можно изменить



Учим дружить postgres и python



Что будем делать



Создадим БД и подключимся из python



```
psql>CREATE DATABASE suppliers;
```

Библиотека для python: [psycopg2](#)

```
import psycopg2
conn = psycopg2.connect(host="localhost",database="suppliers", user="postgres",
password="postgres")
```

Лучше убрать настройки подключения в конфиг

```
[postgresql]
host=localhost
database=suppliers
user=postgres
password=postgres
```



Выполнение запросов из python



```
cur = conn.cursor()
print('PostgreSQL database version:')
try:
    cur.execute('SELECT version()')
    # display the PostgreSQL database server version
    db_version = cur.fetchone()
    print(db_version)
    # close the communication with the PostgreSQL
    cur.close()
except (Exception, psycopg2.DatabaseError) as error:
    print(error)
finally:
    if conn is not None:
        conn.close()
        print('Database connection closed.')
```



Как создаются таблицы (немного магии SQL)



```
commands = (
    """
CREATE TABLE vendors (
    vendor_id SERIAL PRIMARY KEY,
    vendor_name VARCHAR(255) NOT NULL
)
""",
    """
CREATE TABLE parts (
    part_id SERIAL PRIMARY KEY,
    part_name VARCHAR(255) NOT NULL
)
""",
    """
CREATE TABLE part_drawings (
    part_id INTEGER PRIMARY KEY,
    file_extension VARCHAR(5) NOT NULL,
    drawing_data BYTEA NOT NULL,
    FOREIGN KEY (part_id)
        REFERENCES parts (part_id)
    ON UPDATE CASCADE ON DELETE CASCADE
)
"""
,
```

```
    """
CREATE TABLE vendor_parts (
    vendor_id INTEGER NOT NULL,
    part_id INTEGER NOT NULL,
    PRIMARY KEY (vendor_id , part_id),
    FOREIGN KEY (vendor_id)
        REFERENCES vendors (vendor_id)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (part_id)
        REFERENCES parts (part_id)
    ON UPDATE CASCADE ON DELETE CASCADE
)
"""
)
```



А теперь в python



```
try:  
    params = config()  
    conn = psycopg2.connect(**params)  
    cur = conn.cursor()  
    for command in commands:  
        cur.execute(command)  
    cur.close()  
    conn.commit()  
except (Exception, psycopg2.DatabaseError) as error:  
    print(error)  
finally:  
    if conn is not None:  
        conn.close()
```



Вставка данных



```
def insert_vendor_list(vendor_list):
    """ insert multiple vendors into the vendors table """
    sql = "INSERT INTO vendors(vendor_name) VALUES(%s)"
    conn = None
    try:
        params = config()
        conn = psycopg2.connect(**params)
        cur = conn.cursor()
        cur.executemany(sql,vendor_list)
        conn.commit()
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()
```



Изменение данных



```
sql = """ UPDATE vendors
    SET vendor_name = %s
    WHERE vendor_id = %s"""

conn = None
updated_rows = 0
try:
    params = config()
    conn = psycopg2.connect(**params)
    cur = conn.cursor()
    cur.execute(sql, (vendor_name, vendor_id))
    updated_rows = cur.rowcount
    conn.commit()
    cur.close()
except (Exception, psycopg2.DatabaseError) as error:
    print(error)
finally:
    if conn is not None:
        conn.close()
```



Проверяем в psql



###Было

```
suppliers=# SELECT * FROM vendors WHERE vendor_id = 1;
vendor_id | vendor_name
-----+-----
 1 | 3M Co.
(1 row)
```

###Стало

```
suppliers=# SELECT * FROM vendors WHERE vendor_id = 1;
vendor_id | vendor_name
-----+-----
 1 | 3M Corp
(1 row)
```

Хранимые процедуры и их вызов из кода



```
CREATE OR REPLACE FUNCTION get_parts_by_vendor(id integer)
RETURNS TABLE(part_id INTEGER, part_name VARCHAR) AS
$$
BEGIN
RETURN QUERY

SELECT parts.part_id, parts.part_name
FROM parts
INNER JOIN vendor_parts ON vendor_parts.part_id = parts.part_id
WHERE vendor_id = id;

END; $$

LANGUAGE plpgsql;
```



Хранимые процедуры и их вызов из кода



```
""" get parts provided by a vendor specified by the vendor_id """
conn = None
try:
    # read database configuration
    params = config()
    conn = psycopg2.connect(**params)
    cur = conn.cursor()

    # another way to call a stored procedure
    # cur.execute("SELECT * FROM get_parts_by_vendor( %s); ",(vendor_id,))
    cur.callproc('get_parts_by_vendor', (vendor_id,))
    row = cur.fetchone()
    while row is not None:
        print(row)
        row = cur.fetchone()
    cur.close()
except (Exception, psycopg2.DatabaseError) as error:
    print(error)
finally:
    if conn is not None:
        conn.close()
```



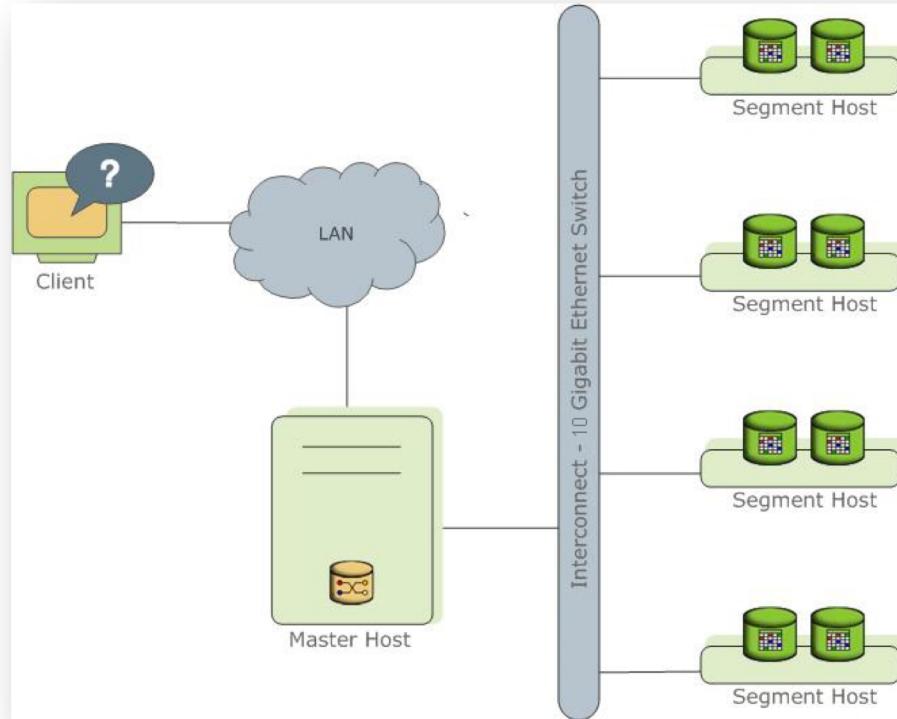
А что если данные – не текст?



```
def write_blob(part_id, path_to_file, file_extension):
    """ insert a BLOB into a table """
    conn = None
    try:
        drawing = open(path_to_file, 'rb').read()
        params = config()
        conn = psycopg2.connect(**params)
        cur = conn.cursor()
        cur.execute("INSERT INTO part_drawings(part_id,file_extension,drawing_data) " +
                   "VALUES(%s,%s,%s)",
                   (part_id, file_extension, psycopg2.Binary(drawing)))
        conn.commit()
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()
```



Архитектура Greenplum





- Соберите (установите) PostgreSQL + pg_stat_statements (пригодится потом)
- Создайте кластер баз данных, запустите сервер
- Убедитесь, что сервер работает =)
- Создайте БД
- Создайте 3 таблицы (пользователи, заказы, товары)
- Наполните их данными
- Заберите их python'ом из БД в CSV-файл



Контакты



- Telegram: @chetvegr
- Gmail: 4etvegr@gmail.com

