Pip install psycopg2

Итак для нашего удобства создадим два файла, первый с кфг второй для работы с бд.

В config пропишем нужные данные чтобы просто не захламлять основной скрипт

host = "127.0.0.1"

user = "nike1999"

password = "Atgiryta1321"

db\_name = "Test"

далее в майне импортнем модули

import psycopg2

from config import host, user, password, db\_name

далее всю работу подключения к бд как понял обязалово нужно производить под обработчиком исключений, если бдшка крашниться то все будет окэ

try:

    pass

except Exeption as ex:

    print(f"ОШИБКА ПРИ ПОДКЛЮЧЕНИИ К БД {ex}")

finally:

    ...

В трае будет подключение, в эксепт логично ошибку вернем.

А файнал закроем соединение с бд.

Создаем новую переменную и юзаем внутряковый метод, в котором передаем данные с кфг

try:

    connection = psycopg2.connect(

        host = host,

        user = user,

        password = password,

        database=db\_name

    )

Далее обрабатываем файнали

finally:

    if connection:

        connection.close()

        print("Успешно закрыли субд")

вот тут потупил что за строчка if connection крч как я понял когда мы подключились к бдшки в connection что-то передастся, и мы сможем ее закрыть в теории можно написать так if connection != None

Теперь чтобы непосредственно работать с бдшкой нам нужно создать курсор и в нем будем все прописывать, можно его просто запихнуть в перинную а потом закрывать в файнали а можно пропустить это конструкцией with.

    with connection.cursor() as cursor:

        pass

надо в кфг еще прописать кодировку на всякий пожарный

encode = "utf-8"

потом в трае

        client\_encoding=encode

запомни эти кфг

host = '127.0.0.1'

user = 'postgres'

password = 'Atgiryta1321'

db\_name = 'postgres'

import psycopg2 as pg

from config import host, user, password, db\_name

connection = None

try:

    connection = pg.connect(

        dbname=db\_name,

        host=host,

        user=user,

        password=password

    )

    connection.autocommit = True

    with connection.cursor() as cursor:

        cursor.execute(

            """

            SELECT \* FROM product;

            """

        )

        rows = cursor.fetchall()

    for row in rows:

        print(row)

except Exception as ex:

    print(f"ERROR {ex}")

finally:

    if connection is not None:

        connection.close()

        print("\*\*\*База данных успешно закрыта\*\*\*")

Подключение к бд через пайтон

Создать базу данных

**Create database name;**

Удалить бд

**Drop database name;**

Создание таблиц

**Create table name(**

**Colum\_name + type + огроничения (not null),**

**Colum\_name + type + огроничения (not null)**

**)**

Удалить таблицу

**Drop table name;**

**Not null –** говорит что значение должно быть обязательно заполнено

Вывести названия всех таблиц:

            SELECT

                tablename

            FROM

                pg\_catalog.pg\_tables

            WHERE

                schemaname = 'public';

заполнение таблиц данными:

**Insert into name(name\_colum\_1, name\_colum\_2, name\_colum\_2)**

**Values (“value\_1”, “value\_2”, “value\_3”)**

**Id –** таблицы можем не заполнять оно в теории автоматом заполняеться как праймари кей

ТЕПЕРЬ ИНТЕРЕСНАЯ ТЕМА.

Мы можем на сайте заполнить наши таблицы он сам введет данные, причем в огромных кол-вах.

[Mockaroo - Random Data Generator and API Mocking Tool | JSON / CSV / SQL / Excel](https://www.mockaroo.com/)

Написание запросов

**Select(запрос) \*(что конкректно) from(откуда) name;**

Выборка всего

**Select \* from name;**

Выборка конкретного столбца

**Select name\_colum from name\_table;**

Можем цеплять несколько строк:

select name\_product, price\_product from product;

**oreder by** сортировка после него указываем по какой колонки будем проводить сортировку, ASC – это сортировка по возрастанию, DESC – по убыванию.

select \* from product order by price\_product ASC;

**distinc** убирает повторения

select distinct name\_product from product order by name\_product ASC;

**where** - условие

select \* from product where price\_product > 200;

Условий может быть несколько.

            select \* from product where price\_product > 150 and description\_product = 'Green';

**limit** указывает ограничение выборки, сколько столбцов надо выбрать.

            select \* from product where price\_product > 150 and description\_product = 'Green' limit 1;

**offset** указывает с какой позиции перебирать.

select \* from product where price\_product > 150 offset 5 limit 3;

**in** уменьшает использование **or**

            select \* from supplier where information = 'Green' or information = 'Orange' or information = 'Blue';

            select \* from supplier where information in ('Green', 'Orange', 'Blue')

**between** между, берет промежуток от and до

            """

            select \* from product

            where price\_product

            between 100 and 250;

            """

**Like** ищет подстроку все входящие в подстроку, % - обозначает неограниченное кол-во эл-мен. После или до искомой подстроки.

Но **like** очень воприимчев к регистру, но мы можем его игнорировать добавив **i**, т.е **ilike**

            select \* from product

            where name\_product

            like 'Bu%';

            """

**Count(\*)** считает кол-во допустим людей с таким же именем и какого то хуя обязательно нужно прописывать группировку причем по этому же столбцу **name.**

            select name, count(\*) from client

            group by name;

тут можем добавить условие, через **having**, которые выведет людей больше двух к примеру с таким же именем.

            select name, count(\*) from client

            group by name

            having count(\*) > 2;

с помощью оператора as можем давать псевдонимы столбца, как в питухоне крч:

**select \*, product\_name as name from product;**

если у нас есть не заполненная ячейка и их много, мы можем заменить пробел на что-то другое функцией **coalesce.**

            select coalesce(name\_product, 'тут пусто') from product;

**max** выводит максимальное значение

select max(price\_product) from product;

**min** минимальное

select min(price\_product) from product;

**avg** ср-знач

select avg(price\_product) from product;

**round** - округление

select round(avg(price\_product)) from product;

**sum** читает сумму.

select sum(price\_product) from product;

также работают все основные арифметические функции + - / ^ =

Если мы хотим, чтобы столбец имел только уникальные значения, то для него можно определить атрибут **UNIQUE**.

Добавить ограничения в имеющиюся таблицу вот так, alter table – это изменение, add constraint это добавление имени ограничения, потом unique и само ограничение на повторение, типо теперь у нас не может быть одинаковых почт.

alter table client add constraint unique\_adres UNIQUE (adres);

Так-же ограничение можно задавать **Check** указывать какие столбы конкретно могут существовать.

            alter table client add constraint unique\_adres Check (gender = 'Female' or gender 'Male');

**Delete** для удаления записей

**Delte from name\_table**

select \* from client

            where name ilike 'b%'

**update set используем для обновления данных в таблице**

            update client set name = 'Mike'

            where client.name ilike 'b%'

Удаление таблицы

drop table bysicle

Объединение таблиц **Join on** выводит только тех у кого логично есть связь по фк к пк

select \* from product

join

orders on orders.id\_prod = id\_product

Так же можно делать делать конкретную выборку на столбцы

            select Product.name\_product, Product.price\_product, orders.value, orders.date from product

            join

            orders on orders.id\_prod = id\_product

это все **inner join** он стандартный и может прописываться без **inner**

**left join –** все записи из левой таблицы и совпадения из правой таблицы

**right join –** все записи из правой таблицы и совпадения из левой таблицы

**Alter** работает с колонками.

**Update** со значением в колонках.

Для соединения нескольких таблиц

SELECT t1.column1, t2.column2, t3.column3

FROM table1 t1

JOIN table2 t2 ON t1.column1 = t2.column1

JOIN table3 t3 ON t2.column2 = t3.column2;

**Inner join** возвращает все одинаковые элементы из двух таблиц

**Left join** возвращает все элементы из левой таблицы и одинаковые из правой

**Right join** обратное left

**Full join** возвращает все строки если они имеют соответствие одной из таблиц.

**Cross join** каждая строка из первой таблицы объединяется со всеми строками из второй таблицы

Прикинь ситуацию ты заполняешь бд и тебе нужно чтобы, когда проходил одинаковый айдишник одной части из таблицы ты пытался заново кинуть запрос на заполнение и он бы не выкидывал ошибку что запись в бд уже существует. Это делается так.

        ON CONFLICT (id\_user) DO UPDATE

        SET

            photo = EXCLUDED.photo,

            purpose = EXCLUDED.purpose,

            description = EXCLUDED.description,

Мы указываем какой столбец возможен к повторению. Прописываем обновление DO UPDATE и SET выводим те же данные с EXCLUDED который говорит, какие данные в таблице мы обновим.