# Занятие 24

SQL

# Что напечатает?

```
data_set = {tuple(): 1}
print(data_set)


for x in range(10):
    if x == 5:
        break
    else:
        print(x)
else:
    print('Python')
```

# Задача 23-1

Найдите длину наибольшей подстроки данной строки, которая является палиндромом.

Например, дана строка ‘aabbccddcc’ тогда длиной подстроки с наибольшим палиндромом является 6 (подстрока ‘ccddcc’)

# Задача 23-2

1. Напишите программу, которая считывает информацию из таблицы book и печатает ее в виде таблицы, соответствующей таблице из базы данных (заголовки и строки данных).

2. Более сложный вариант составить DataFrame на основании считанной информации, соответствующий таблице из базы данных.

# Задача 23-3

Создайте функцию, на вход которой подается список из целых положительных чисел, и которая в качестве результата возвращает самое большое число, которое можно составить из этих чисел.

Например, вход [1, 21, 3], результат 3211

Если вход [9, 81, 25], то результат 98125.

# 2 решения задачи 23-3

```
lst = [7, 71, 72, 73]
s = list(map(str, lst))
while True:
    print(s)
    for i in range(len(s) - 1):
        if s[i] + s[i + 1] < s[i + 1] + s[i]:
            s[i], s[i + 1] = s[i + 1], s[i]
            break
    else:
        break
print(int(''.join(map(str, s))))
```

```
import functools
def get_biggest(numbers):

    def compare(x, y):
        return (-1 if str(x) + str(y) > str(y) + str(x) else 1)

    nu = sorted(numbers, key = functools.cmp_to_key(compare))
    return int(''.join([str(i) for i in nu]))
```

# Найдите ошибку в данном решении задачи 23-3

```
join_to_biggest = lambda a: int(''.join(sorted(map(str, a), reverse=True)))

print(join_to_biggest([7, 71, 72]))
```

# Алгоритм взаимодействия с БД

1. Установка соединения с сервером БД функцией connect()

2. Выполнение запроса:

    2.1. Получить объект курсора методом cursor()

    2.2. Выполнить запрос методом execute()

    2.3. Если запрос на изменение данных или структуры БД, то нужно зафиксировать изменения методом commit()

    2.4. Если запрос на получение данных (SELECT):

        Обработать в программе полученный результат

3. Закрыть соединение с сервером БД методом close()

# Работа с Postgresql из Python программы

1. Install package psycopg2

2. Устанавливаем соединение:

```
import psycopg2
con = psycopg2.connect(
  database="postgres",
  user="postgres",
  password= “Здесь должен быть Ваш пароль",
  host="127.0.0.1",
  port="5432" # Может быть другой порт, например, 5433
)
```

# Создание таблицы

```
cur = con.cursor() # курсор
cur.execute('''CREATE TABLE STUDENT1
    (ADMISSION INT PRIMARY KEY NOT NULL,
    NAME TEXT NOT NULL,
    AGE INT NOT NULL,
    COURSE CHAR(50),
    DEPARTMENT CHAR(50));'‘’)
con.commit() # commit
con.close()  # закрыть соединение
```

# Загрузка данных

```
 cur = con.cursor()

 cur.execute( "INSERT INTO STUDENT (ADMISSION,NAME,AGE,COURSE,DEPARTMENT) VALUES (3419, 'Abel', 17, 'Computer Science', 'ICT')"

)

cur.execute("INSERT INTO STUDENT (ADMISSION,NAME,AGE,COURSE,DEPARTMENT) VALUES (3421, 'Joel', 17, 'Computer Science', 'ICT')"

)

cur.execute("INSERT INTO STUDENT (ADMISSION,NAME,AGE,COURSE,DEPARTMENT) VALUES (3422, 'Antony', 19, 'Electrical Engineering', 'Engineering')"

)

\cur.execute("INSERT INTO STUDENT (ADMISSION,NAME,AGE,COURSE,DEPARTMENT) VALUES (3423, 'Alice', 18, 'Information Technology', 'ICT')"

)

    con.commit()

    con.close()
```

# SELECT

```
cur = con.cursor()
cur.execute("SELECT admission, name, age, course, department from STUDENT")

rows = cur.fetchall() # возвращает список всех строк
for row in rows:
   print("ADMISSION =", row[0])
   print("NAME =", row[1])
   print("AGE =", row[2])
   print("COURSE =", row[3])
   print("DEPARTMENT =", row[4], "\n")
con.close()
```

# Cursor

cursor.fetchone() — возвращает 1 строку

cursor.fetchall() — возвращает список всех строк

cursor.fetchmany(size=5) — возвращает заданное количество строк

Также курсор является итерируемым объектом, поэтому можно так:

```
for row in cursor:
    print(row)
```

cursor.description

(Column(name='book_id', type_code=23), Column(name='title', type_code=1043), Column(name='author', type_code=1043), Column(name='price', type_code=23), Column(name='amount', type_code=23), Column(name='author_id', type_code=23))

# Задание

Напишите программу, которая считывает информацию из таблицы book и записывает всю эту информацию в эксельный файл.

Названия полей необходимо записать в первой строчке файла как названия колонок.

```
import openpyxl
xlfile = 'book2.xlsx'
wb = openpyxl.Workbook()
ws = wb.active
ws.append(['id','title','author','publisher_id','amount','price','author_id'])
for row in rows:
    ws.append(row)
wb.save(xlfile)
```

# Добавление данных из другой таблицы

- INSERT INTO book (title, author, price, amount) VALUES ('Война и мир','Толстой Л.Н.', 1070, 2), ('Анна Каренина', 'Толстой Л.Н.', 599, 3);

INSERT INTO book (title, author, price, amount)

SELECT title, author, price, amount FROM supply;

INSERT INTO book (title, author, price, amount) SELECT title, author, price, amount FROM supply

WHERE title NOT IN ( SELECT title FROM book );

# Соединяем book и author, чтобы имена писателей не дублировались

- ALTER TABLE book1 ADD COLUMN author_id int;
- UPDATE book1 SET author_id = 1 WHERE author like 'Б%’
- UPDATE book1 SET author_id = 2 WHERE author like 'Д%’
- UPDATE book1 SET author_id = 3 WHERE author like ‘Е%’
- ALTER TABLE book1 DROP author

- Добавим в таблицу book1 книгу “Лирика”, author_id = 4, цена 500, 2 штуки

# Соединение INNER JOIN

```
SELECT title, name_author
FROM author1 INNER JOIN book1
ON author1.author_id = book1.author_id
```

Результат запроса формируется так:

каждая строка одной таблицы сопоставляется с каждой строкой второй таблицы;

для полученной «соединённой» строки проверяется условие соединения;

если условие истинно, в таблицу результата добавляется соответствующая «соединённая» строка

# Внешнее соединение LEFT и RIGHT OUTER JOIN

```
SELECT title, author1.name_author
FROM book1 LEFT JOIN author1
ON author1.author_id = book1.author_id;
```

```
SELECT title, author1.name_author
FROM book1 RIGHT JOIN author1
ON author1.author_id = book1.author_id;
```

Результат запроса формируется так:

в результат включается внутреннее соединение (INNER JOIN) первой и второй таблицы в соответствии с условием;

затем в результат добавляются те записи первой таблицы, которые не вошли во внутреннее соединение на шаге 1, для таких записей соответствующие поля второй таблицы заполняются значениями NULL.

# FULL OUTER JOIN

Сначала выполняется внутреннее соединение.

Затем в результат добавляются все строки из T1, которым не соответствуют никакие строки в T2, а вместо значений столбцов T2 вставляются NULL.

И наконец, в результат включаются все строки из T2, которым не соответствуют никакие строки в T1, а вместо значений столбцов T1 вставляются NULL.

# Выполните этот запрос для таблиц book и supply

# Запрос в несколько этапов

Вывести авторов, общее количество книг которых на складе максимально.

Шаг 1.

Найдем суммарное количество книг на складе по каждому автору. Поскольку фамилии автора в этой таблице нет, то группировку будем осуществлять по author_id.

SELECT author_id, SUM(amount) AS sum_amount FROM book GROUP BY author_id

# Шаг 2

В результирующей таблице предыдущего запроса необходимо найти максимальное значение. Для этого запросу, созданному на шаге 1, необходимо присвоить имя (например, query_in) и использовать его в качестве таблицы-источника после FROM. Затем уже находить максимум по столбцу sum_amount.

```
SELECT MAX(sum_amount) AS max_sum_amount
FROM
    (
     SELECT author_id, SUM(amount) AS sum_amount
     FROM book
     GROUP BY author_id
    ) query_in
```

# Шаг 3

Выведем фамилию автора и общее количество его книг

```
SELECT name_author, SUM(amount) as Количество
FROM
    author INNER JOIN book
    on author.author_id = book.author_id
GROUP BY name_author
```

# Шаг 4

Шаг 4.  Включим запрос с шага 2 в условие отбора запроса с шага 3. И получим всех авторов, общее количество книг которых максимально.

```
SELECT name_author, SUM(amount) as Количество
FROM author INNER JOIN book on author.author_id = book.author_id GROUP BY name_author
HAVING SUM(amount) =
(/* вычисляем максимальное из общего количества книг каждого автора */
        SELECT MAX(sum_amount) AS max_sum_amount
        FROM (/* считаем количество книг каждого автора */
        SELECT author_id, SUM(amount) AS sum_amount
        FROM book GROUP BY author_id )
        query_in );
```

# Задание

Найти автора (или авторов) с самой дорогой книгой

# Зачем нужен первичный ключ?

- **Первичным ключом (Primary key)** - называется атрибут или набор атрибутов, который уникальным образом идентифицирует сущность. Если первичный ключ состоит более чем из одного атрибута, то его называют составным первичным ключом.
- **Каждая сущность должна иметь первичный ключ,** в противном случае вы никогда не сможете однозначно ее идентифицировать. Требуются очень веские причины,почему ваша сущность не имеет первичного ключа, такое встречается, но крайне редко.

# Зачем нужен внешний ключ?

- **Внешние ключи** (Foreign key) - используются для организации связей между таблицами базы данных (родительскими и дочерними) и для поддержания ограничений ссылочной целостности данных.

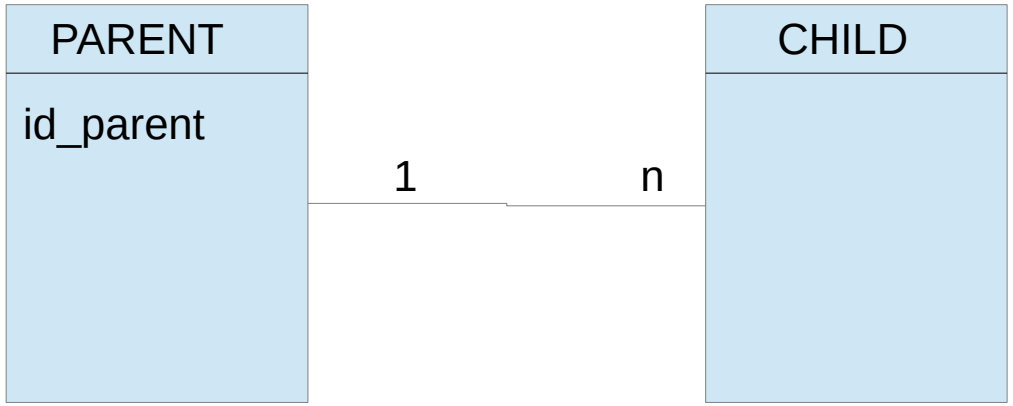
# FOREIGN KEY

```
CREATE TABLE book
(book_id INT PRIMARY KEY,
 title VARCHAR(50),
 author_id INT NOT NULL,
 genre_id INT,
 price DECIMAL(8, 2),
 amount int,
 FOREIGN KEY (author_id)  REFERENCES author (author_id),
 FOREIGN KEY (genre_id)  REFERENCES genre (genre_id) )
```

# Как добавить связь между таблицами

```
ALTER TABLE child_name
        ADD CONSTRAINT name_constraint
        FOREIGN KEY(id_parent)
        REFERENCES parent_name(id_parent)
```

# Пример




```
ALTER TABLE child ADD CONSTRAINT cnst_child_ref_parent
FOREIGN KEY (id_parent)
REFERENCES parent(id_parent)
ON DELETE RESTRICT
```

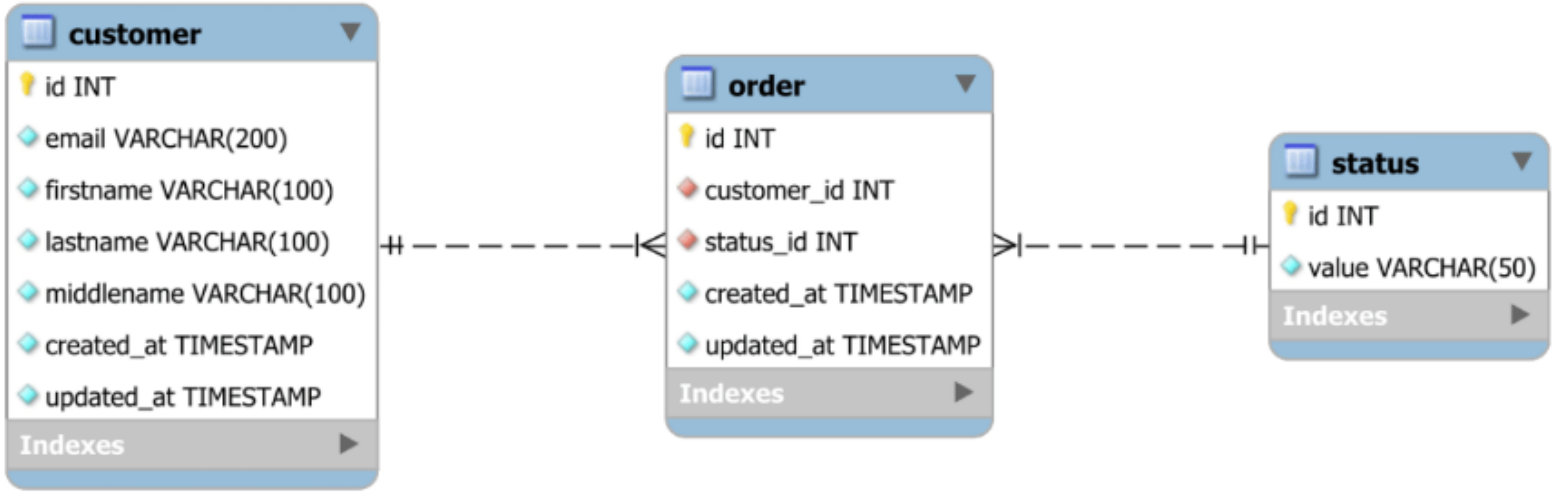
# Отношение один-ко-многим

Один экземпляр одной сущности может быть связан с множеством экземпляров другой сущности.

Например, покупатель может совершить множество заказов в интернет магазине, но один конкретный заказ может принадлежать только одному покупателю.

В таблицу order требуется добавить внешний ключ customer_id.

Значением внешнего ключа – будет значение первичного ключа из таблицы customer:



# Примеры

Один статус заказа можно использоваться во множестве заказов, но в один момент времени заказ может иметь только один статус;

У товара может быть множество отзывов, но конкретный отзыв может быть оставлен только для одного товара;

В одном отделе может работать множество сотрудников, но конкретный сотрудник работает только в одном отделе;

# Многие-ко-многим

Множество экземпляров одной сущности может быть связано с множеством экземпляров другой сущности.

Например, в одну категорию можно добавить множество товаров, однако для удобства пользователя товару можно назначить множество категорий.

В реляционной БД связь многие-ко-многим можно разрешить только через вспомогательную.

Таким образом связь многие-ко-многим превращается в две связи один-ко-многим.

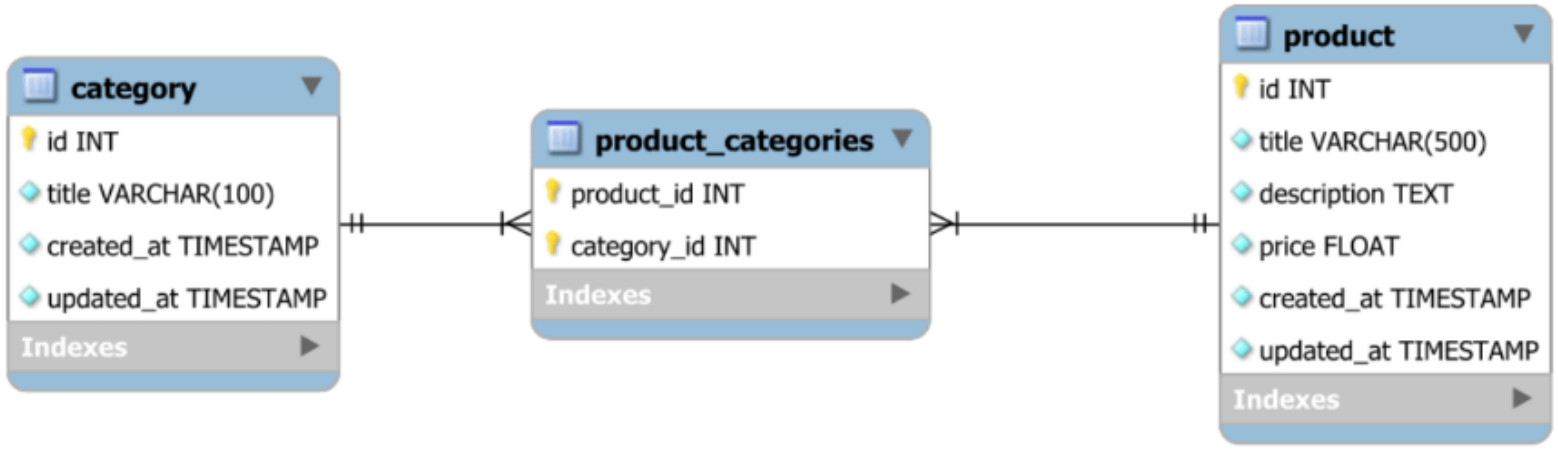
# Многие-ко-многим

В таблицу ассоциации product_categories требуется добавить два внешних ключа product_id и category_id.

Значением внешних ключей будут значения первичных ключей из таблиц product и category соответственно.

Чтобы избежать случайного добавления товара в одну категорию дважды, либо присвоения категории одному товару дважды, внешние ключи вместе образуют первичный ключ:

Например, PRIMARY KEY (product_id, category_id)



# Примеры многие-ко-многим

- Автор может написать множество книг, но одну книгу может написать множество авторов;
- На курсах дополнительного образования, студент может учиться на множестве курсов, но на одном курсе может учиться множество студентов;

# Отношения один-к-одному

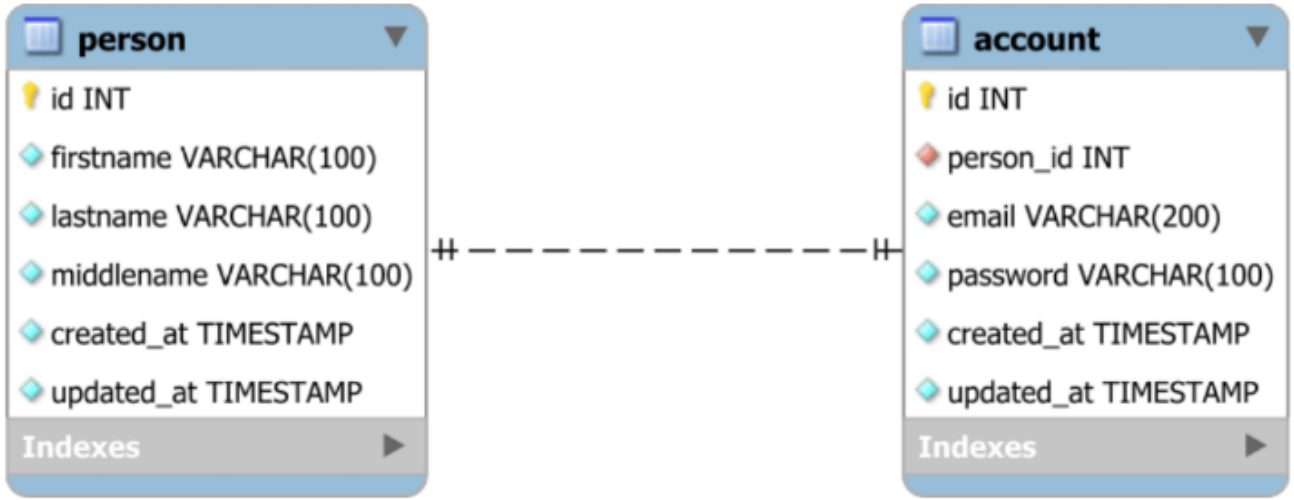
Один экземпляр одной сущности может быть связан не более чем с одним экземпляром другой сущности.

Используется, если необходимо отделить некоторый набор сведений, однозначно связанный с конкретным экземпляром.

Связь один-к-одному в БД описывается также, как связь один-ко-многим.

В таблицу account требуется добавить внешний ключ person_id. Значением внешнего ключа - будет значение первичного ключа из таблицы person .

Если вы хотите избежать случайного добавления второго аккаунта, то значение внешнего ключа person_id можно сделать уникальным индексом:



# Типы данных

INTEGER — целое число (smallint, bigint, int2, int4, int8)

NUMERIC — число с плавающей точкой (decimal, real, double precision, float)

VARCHAR() — текстовые данные (char)

TEXT — набор с максимальной длиной 65535

DATE — дата (SELECT current_time)

TIMESTAMP — дата и время (SELECT current_timestamp)

BOOLEAN — логический тип

JSON — текстовый json

# INNER JOIN

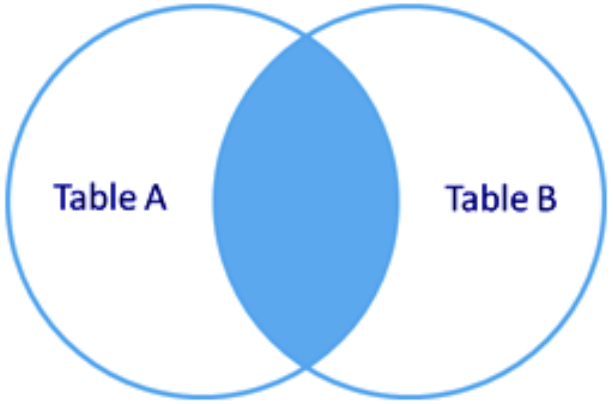



*рис. Inner join*

Шаблон запроса:

```
SELECT  a.name , b.value
  FROM  table1 a, table2 b
 WHERE  a.id = b.id
```

# Объединение, пересечение, «разность»

Союзы

```
SELECT n from numders1;
 UNION
SELECT n from mumbers2;
```

UNION — Объединяет в одну таблицу результаты 2-х и более запросов.

UNION ALL — Для получения списка со всеми дубликатами.

INTERSECT — Возвращает пересечение результатов нескольких запросов.

EXCEPT — Возвращает исключение результатов второго запроса из первого.

# Задание

Создайте два файла с одинаковым набором полей (например, id, name)

(можно использовать уже имеющийся файл author, например)

Введите несколько строк, совпадающих и нет.

Выполните команды SELECT с командами UNION, UNION ALL, INTERSECT, EXCEPT

# View

CREATE VIEW view_name AS

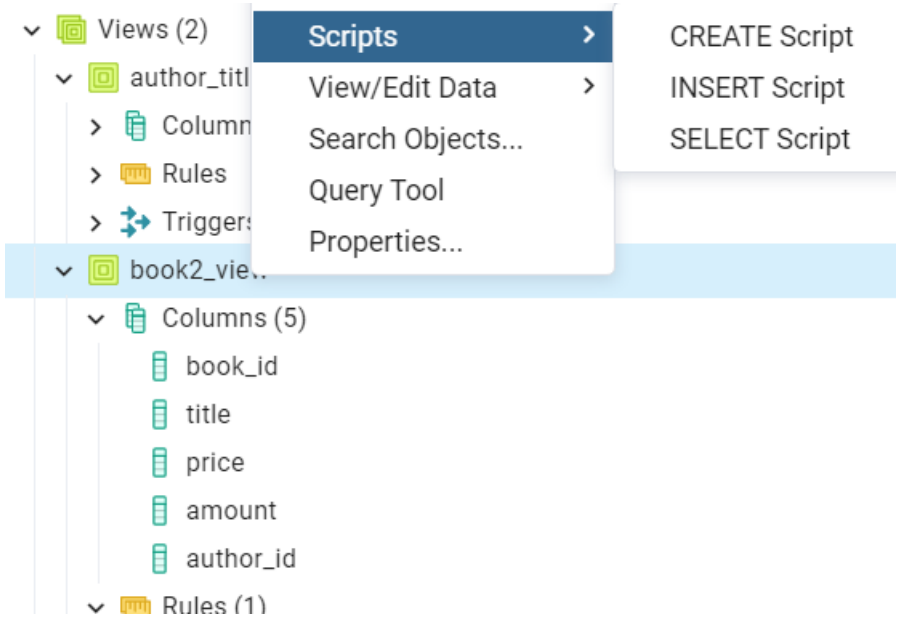
SELECT select_statement

SELECT * FROM view_name – можем использовать в запросах

CREATE OR REPLACE VIEW – есть ограничения

Давайте перепишем сложный запрос с помощью создания view

# Как посмотреть View – через Create script



# Index

Индекс – это дополнительная структура данных для ускорения работы запросов.

Результат запроса с индексом и без должны быть одинаковыми.

Индексы важны для поиска, для сортировки, группировки, соединения таблиц.

Индексы не всегда полезны!!!

Затраты на содержание, могут замедлять работу при больших объемах при вводе и коррекции данных.

# CREATE INDEX

CREATE INDEX имя-индекса

ON имя-таблицы (имя-столбца, …)

CREATE INDEX book2_author_id

ON book2 (author_id)


```
1  -- Index: book2_author_id
2  Loading...
3  -- DROP INDEX IF EXISTS public.book2_author_id;
4
5  CREATE INDEX IF NOT EXISTS book2_author_id
6      ON public.book2 USING btree
7      (author_id ASC NULLS LAST)
8      TABLESPACE pg_default;
9
```

# Задание

Создайте и напечатайте заказ на закупку книг, чтобы каждой книги на складе стало не меньше 100 экземпляров.

Заказ должен включать порядковый номер позиции (от 1 до …), автора, название книги, сколько надо докупить экземпляров до 100 (если уже не меньше 100, то можно не покупать), цену, стоимость.

Последняя строчка должна состоять из слова ИТОГО и общей суммы.

# Задание 24-0

Инсталлируйте модуль PyQt6

# Задача 24-1

Напишите функцию, которая сортирует числовой список, не используя никаких функций, вроде sort,  sorted, min, max и т.д.

# Задача 24-2

Создайте запрос, который находит авторов, у которых только минимальное количество книг на складе (в таблице book). Используйте для этого View.

# Задача 24-3

Создайте функцию, которая принимает на входе строку из круглых скобок, открывающих и закрывающих, и возвращает True или False, если строка является «правильной», т.е. никогда количество закрывающих не больше, чем количество открывающих, если двигаться по строке слева направо.

Примеры:

"()" => true

")(()))" => false

"(" => false

"(()) (( () () ) () )" => true