

Модуль 1.2 Введение в Data Science

План:

1. Базы данных и SQL.
2. Numpy - обзор библиотеки.
3. Pandas - обзор библиотеки.
4. Базовый эксплоративный анализ (EDA).

1. Базы данных и SQL.

Понятие Базы Данных

База Данных (БД) - это набор взаимосвязанных данных и правила хранения этих данных.



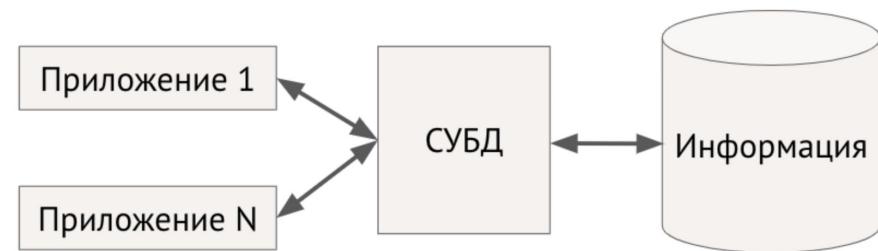
Система Управления Базами Данных

Система Управления Базами Данных

(СУБД) или Database Management

System (DMS) - это комплекс средств, которые позволяют управлять, создавать и извлекать информацию из базы данных.

Простым языком - это просто комплекс программных средств, которые позволяют взаимодействовать с базами данных.

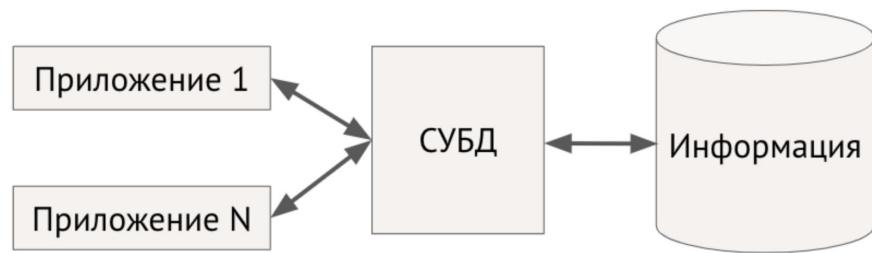


Аналогия для лучшего понимания

Архив с данными - вся та информация, которую мы хотим анализировать и хранить.

Архивариус - СУБД, через которого идут все манипуляции с базой данных (архивом) - создание, добавление, чтение и обновление информации.

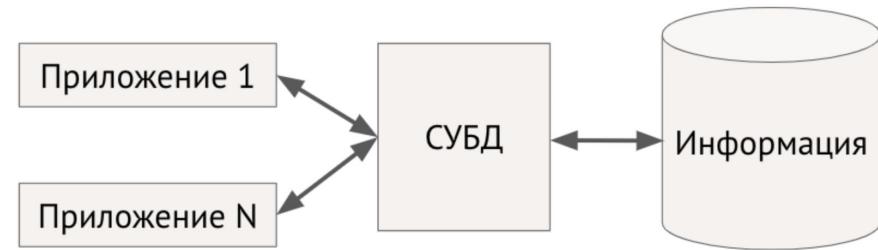
Посетитель - приложение, которому необходимо произвести набор операций с БД.



Типы СУБД

3 основных типа

- файл-серверные
- клиент-серверные
- встраиваемые



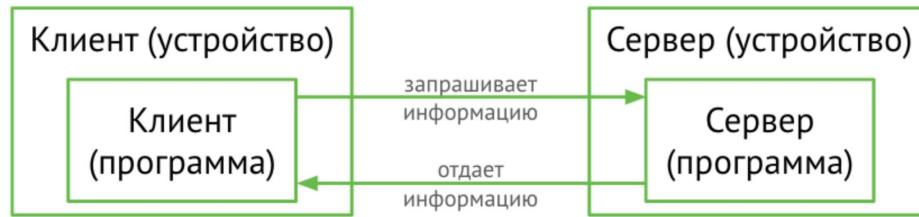
Клиент и Сервер

Клиент:

- программа, которая хочет получить информацию
- физическое устройство на котором работает приложение-клиент*

Сервер:

- специальная программа, которая отдает информацию
- физическое устройство на котором запущена программа-сервер*

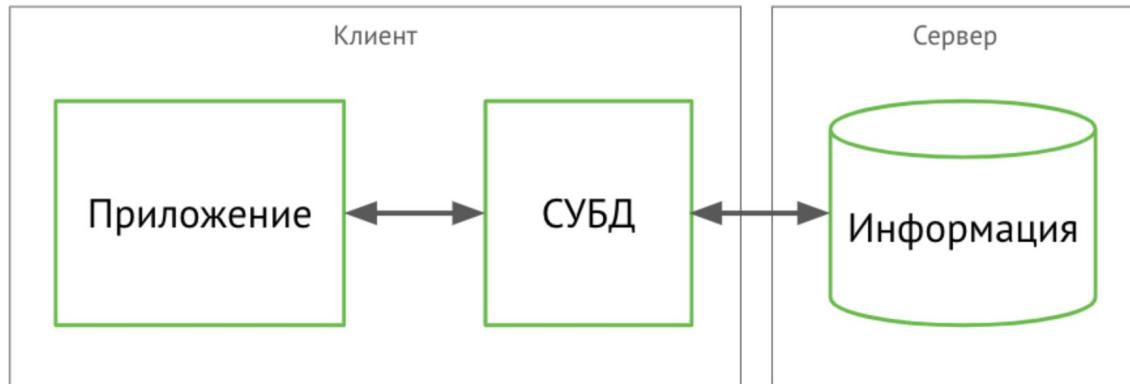


*Обычно данные программы расположены на разных вычислительных машинах и взаимодействуют между собой путем различных протоколов, но они могут быть расположены и на одной машине.

Файл-серверные СУБД

Файлы и информация хранятся на сервере, а СУБД на стороне клиента.

Например Microsoft Access

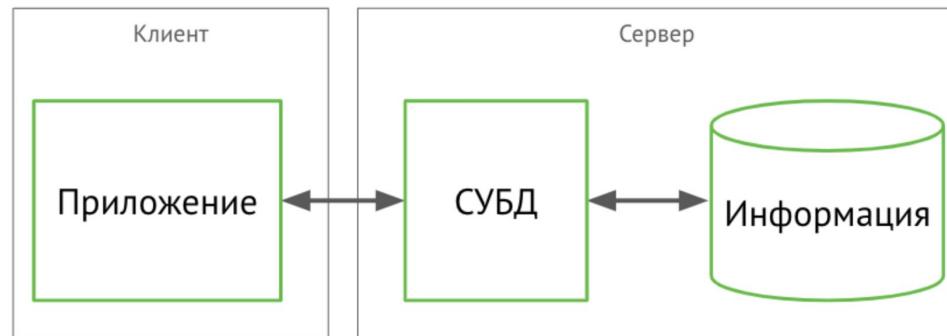


Клиент-серверные СУБД

И файлы с информацией и сама СУБД находятся на стороне сервера.

Клиент обращается за данными и информацией через легковесную **вспомогательную программу**, которая позволяет взаимодействовать с клиент-серверной СУБД.

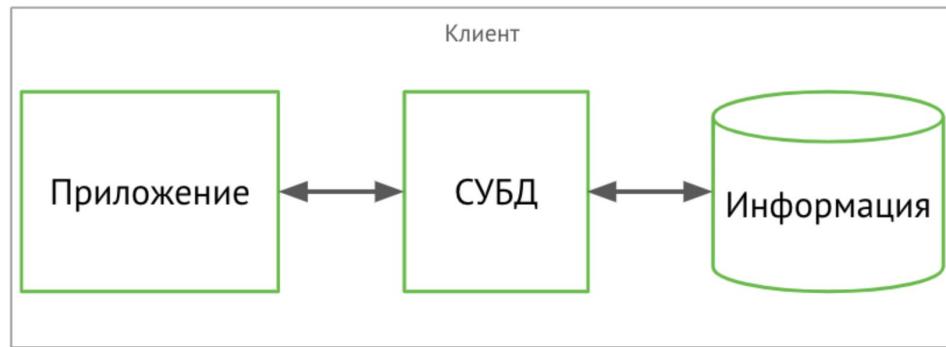
Программы: MySQL, PostgreSQL, Microsoft SQL, Oracle, MongoDB, Cassandra.



Встраиваемые СУБД

И файлы и СУБД хранятся строго на стороне клиента.

Программа: SQLite



Плюсы и минусы каждого типа СУБД

Тип СУБД	Плюсы	Минусы
Файл-серверные	<ul style="list-style-type: none">Сервер может быть обычным файловым хранилищемЛегко переносить базу	<ul style="list-style-type: none">Плохо параллелятся действия от разных клиентовТребуется установка СУБД на каждом клиенте
Клиент-серверные	<ul style="list-style-type: none">На клиенте не надо устанавливать СУБДХорошо параллелятся действия от разных клиентов	<ul style="list-style-type: none">Сервер должен быть достаточно производительным => дорого
Встраиваемые	<ul style="list-style-type: none">Не надо ничего устанавливать	<ul style="list-style-type: none">Подходит только для локального хранения

Типы Баз Данных

Реляционный тип (Relational Databases) – это базы данных , в которых информация строго структурирована и связана с другой информацией жесткими правилами.

Пример:

- SQLite
- MySQL
- PostgreSQL
- Microsoft SQL

Нереляционный тип (NoSQL) – это базы данных, в которых нет жестких ограничений ни по структуре, ни по связям между информацией

Пример:

- Redis
- MongoDB
- Cassandra
- BigQuery

Реляционные Базы Данных

Сущность - отдельный описываемый объект. Например, рассмотрим успеваемость студента на курсе.

Отношение - таблица.



id	name	gpa
1	Егор	4.82
2	Егор	4.11
3	Егор	3.88

Пример отношения «Успеваемость студентов»

Реляционные Базы Данных

Атрибут (или поле) - столбец в таблице.

Запись (или кортеж) - строка в таблице.

The diagram shows a table with three rows and four columns. The columns are labeled 'id', 'name', and 'gpa'. The first row has values 1, Егор, and 4.82. The second row, which is highlighted with a blue border, has values 2, Егор, and 4.11. The third row has values 3, Егор, and 3.88. A red arrow points from the text 'Атрибут' to the 'name' column header. A blue arrow points from the text 'Кортеж' to the second row of the table.

	id	name	gpa
1		Егор	4.82
2		Егор	4.11
3		Егор	3.88

Схема Базы Данных. Таблицы и Данные

Таблица 1

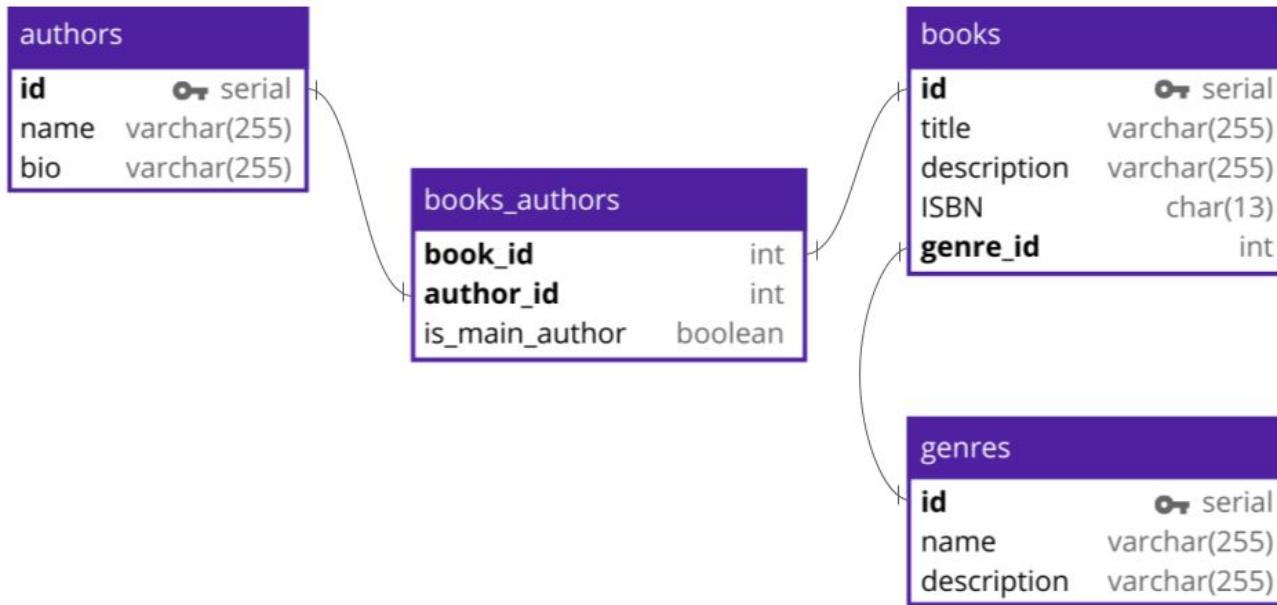
Атрибут 1	Атрибут 2	Атрибут 3
1	Егор	4.25
2	Дима	3.82
3	Миша	4.15

Таким образом описываются конкретные данные в таблице.

Таблица 1
Атрибут 1
Атрибут 2
Атрибут 3

Таким способом описываются таблицы и их атрибуты: информацию какого вида и типа таблица содержит.

Пример Схемы БД



Практика

Есть категории интернет-магазина и есть товары.

Каждый товар принадлежит строго одной категории.

К товарам могут написать отзывы (к одному товару можно написать множество отзывов). Необходимо хранить информацию о категориях, товарах и отзывах.

Визуализировать решение можно в любом удобном графическом редакторе.

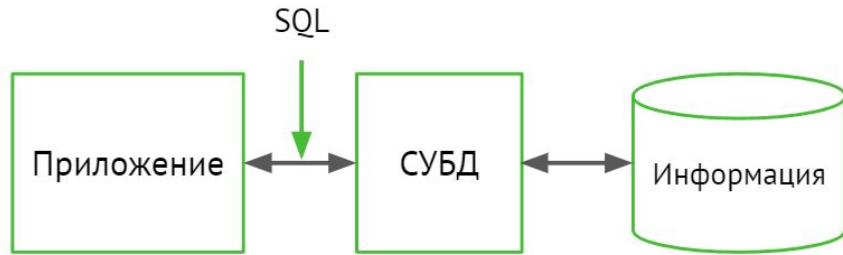
Предлагаю онлайн-платформу <https://draw.io>

Structured Query Language (SQL)

Structured Query Language (SQL) -

язык для извлечения, изменения, удаления и добавления данных.

Данный язык понимает СУБД, которая и производит соответствующие операции с базой данных.



Structured Query Language (SQL)

Пример запроса

```
SELECT * FROM student;
```

Результат выполнения

	id	name	gpa	birth
1	12	Карина	4.7	2000-09-12 00:00:00
2	13	Игорь	3.8	2000-01-26 00:00:00
3	15	Илья	4.2	1999-05-08 00:00:00
4	17	Вова	[NULL]	1999-04-14 00:00:00

Типы запросов в SQL

- **DDL** (Data Definition Language) - CREATE, ALTER, DROP
- **DML** (Data Manipulation Language) - SELECT, INSERT, UPDATE, DELETE
- **TCL** (Transaction Control Language) - COMMIT, ROLLBACK, SAVEPOINT
- **DCL** (Data Control Language) - GRANT, REVOKE, DENY

Нереляционные Базы Данных (NoSQL)

NoSQL (not only SQL) - это нереляционная база данных.

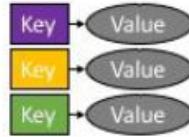
Структура данных, используемая в NoSQL отличается от структуры реляционной базы данных.



Типы Нереляционных Баз Данных

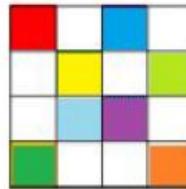
«Ключ-значение»

Redis
Berkley DB
MemcacheDB



Колоночные

Cassandra
HBase



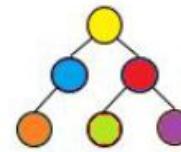
Графовые

Neo4j
OrientDB



Документоориентированные

MongoDB
CouchDB



<https://aws.amazon.com/ru/nosql/>

Инструкция по установке PostgreSQL и DBeaver

1. PostgreSQL

- **Windows:**

<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

- **Linux:**

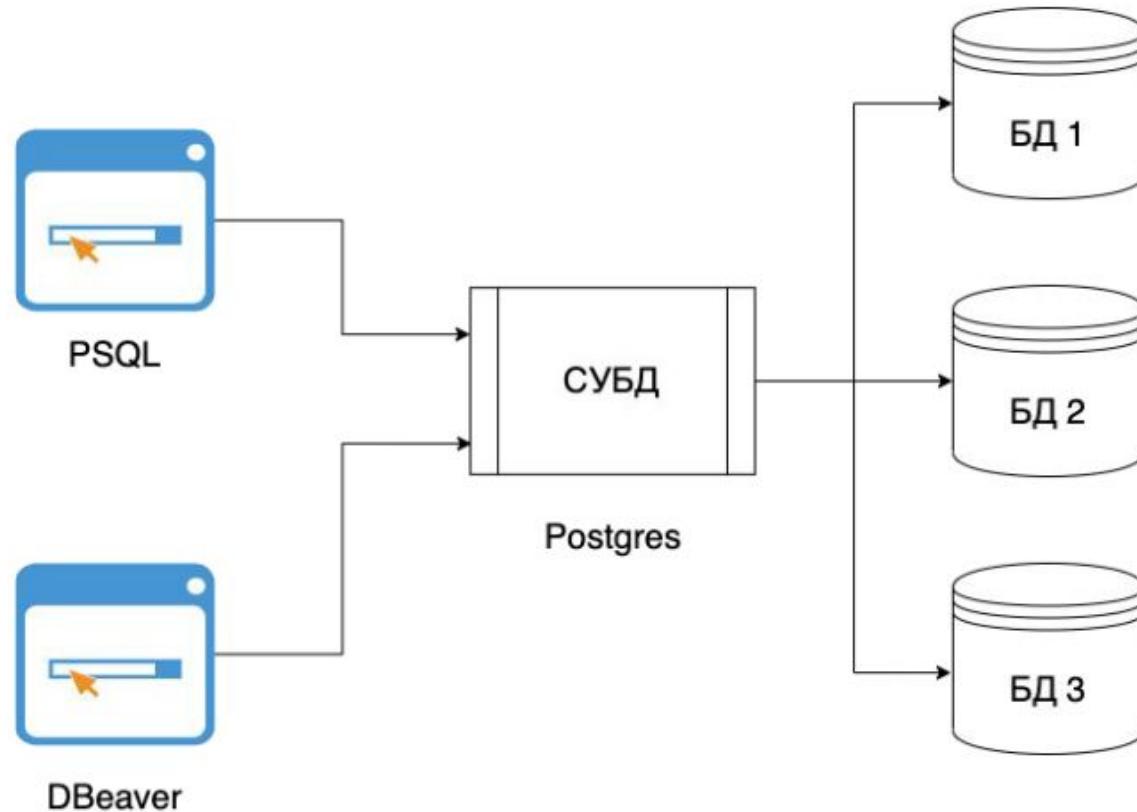
```
# PostgreSQL
sudo apt update && sudo apt install postgresql-12
# pgAdmin4
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add - echo "deb
http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg main" | sudo tee /etc/apt/sources.list.d/pgdg.list
sudo apt update && sudo apt install pgadmin4
```

- **MacOS X:**

```
brew install postgres
postgres -V
pg_ctl -D /usr/local/var/postgres start
createuser -P -s postgres
```

2. DBeaver: <https://dbeaver.io/>

Клиент - СУБД - БД



Работа через консоль

```
# запустить консольное приложение для управления БД от текущего пользователя  
psql
```

```
# запустить консольное приложение для управления БД от пользователя <user> например, postgres  
psql -U <user>
```

```
# запустить консольное приложение для управления конкретной БД - <database>  
psql -d <database>
```

```
# запустить консольное приложение для управления конкретной БД, <database>, от пользователя  
<user>  
psql -U <user> -d <database>
```

Создание ролей и БД. Консоль

```
# входим в режим управления от пользователя postgres (БД тоже postgres)
psql -U postgres
```

```
# Создаем БД с именем <name>
create database <basename>;
```

```
# Удаляем БД с именем <name>
drop database <basename>;
```

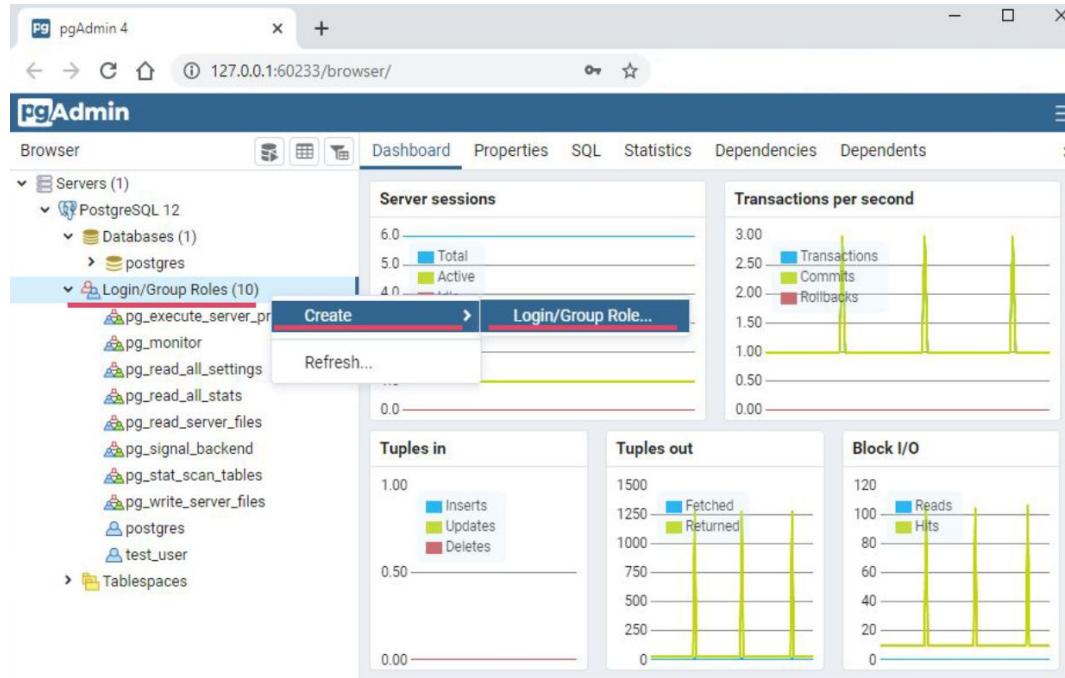
```
# создаем пользователя с именем <name> и паролем <pass>
create user <name> with password '<password>';
```

```
# удаляем пользователя с именем <name>
drop user <name>;
```

```
# указываем, что владельцем БД <db_name> является пользователь <user_name>
alter database <db_name> owner to <user_name>;
```

Создание ролей и БД. pgAdmin4

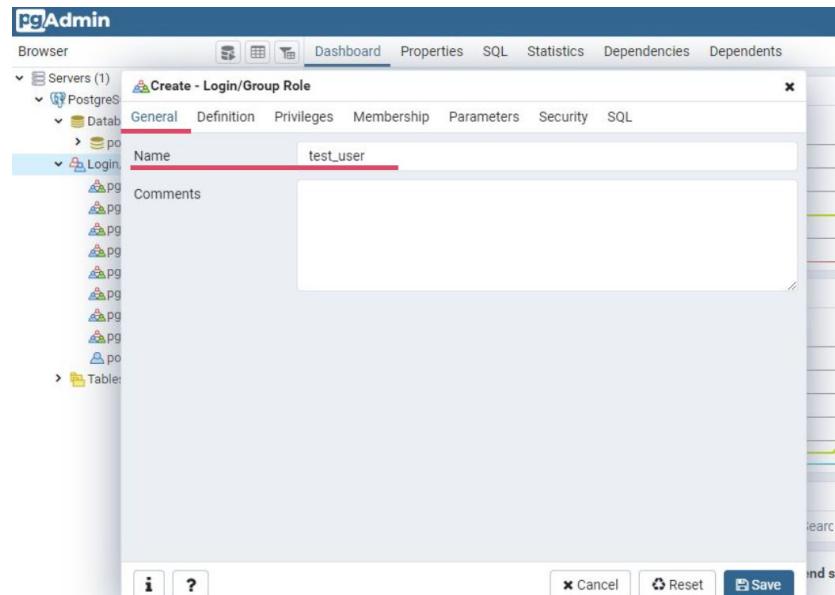
1. Правой клавишей мыши по разделу **Login/Group Roles**
2. Create
3. Login / Group Role



Создание ролей и БД. pgAdmin4

1. Правой клавишей мыши по разделу **Login/Group Roles**
2. Create
3. Login / Group Role

На вкладке General заполняем поле Name - это имя пользователя.

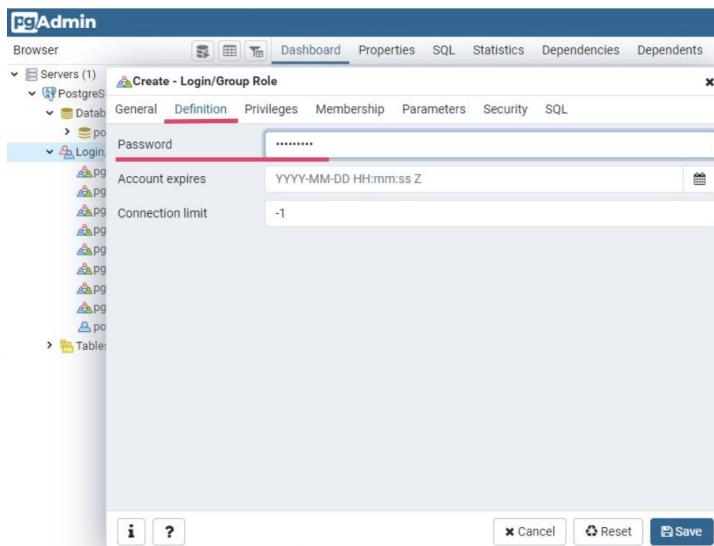


Создание ролей и БД. pgAdmin4

1. Правой клавишей мыши по разделу **Login/Group Roles**
2. Create
3. Login / Group Role

На вкладке **General** заполняем поле **Name** - это имя пользователя.

На вкладке **Definition** заполняем поле **Password** - это пароль для пользователя.



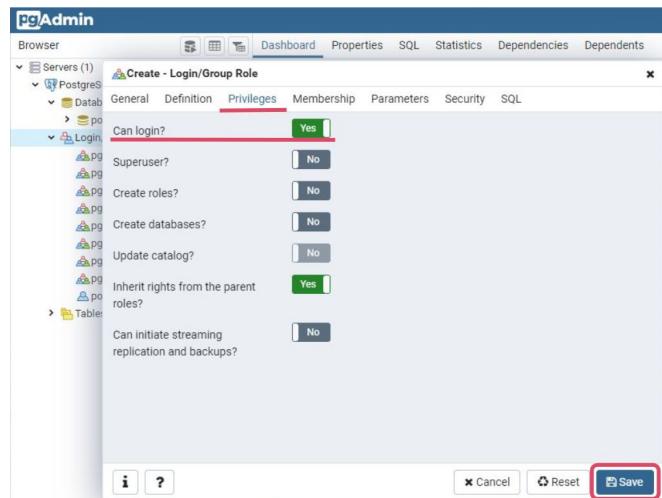
Создание ролей и БД. pgAdmin4

1. Правой клавишей мыши по разделу **Login/Group Roles**
2. Create
3. Login / Group Role

На вкладке **General** заполняем поле **Name** - это имя пользователя.

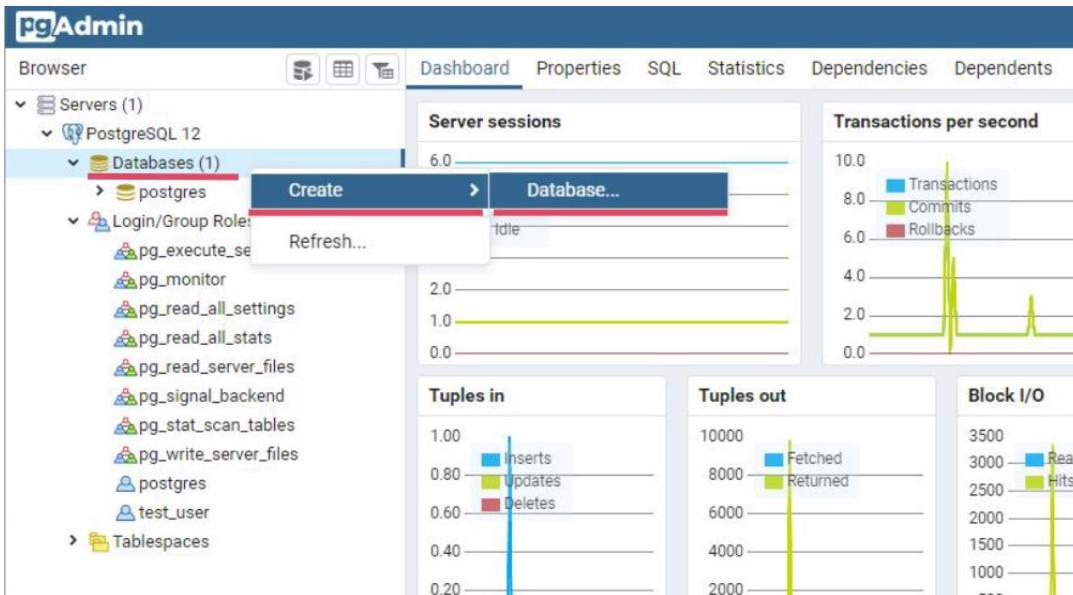
На вкладке **Definition** заполняем поле **Password** - это пароль для пользователя.

На вкладке **Privileges** отмечаем пункт **Can login?** и нажимаем **Save**.



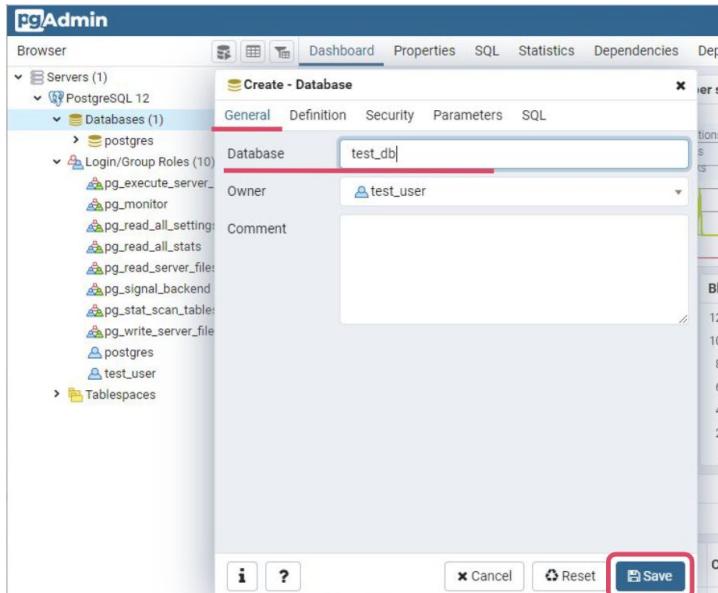
Создание ролей и Бд. pgAdmin4

1. Правой клавишей мыши по разделу **Databases**
2. Create
3. Database



Создание ролей и БД. pgAdmin4

1. На вкладке General заполняем поле Database - это название БД
2. В поле Owner пользователя, которого только что создали - это будет владелец БД
3. Нажимаем кнопку Save



DDL запросы

Какие DDL запросы запомнили?

DDL (Data Definition Language). CREATE

Синтаксис:

```
CREATE table [if not exists] <name> (
    <col_name_1> <col_type_1> [constraints],
    ...,
    <col_name_N> <col_type_N> [constraints],
    [constraints]
);
```

Пример:

```
create table if not exists Student (
    Id serial primary key,
    Name varchar(40) not null,
    GPA real,
    check (GPA > 0)
);
```

Создание таблиц: <https://www.postgresql.org/docs/12/sql-createtable.html>

DDL (Data Definition Language). Типы полей

Столбцы в реляционной базе данных могут хранить записи только одного типа. Основные типы:

Тип	Пояснение	Пример
integer	целые числа	id integer
serial	целые числа с автоинкрементом	id serial
numeric	десятичные числа	gpa numeric(3, 2)
character varying	строки ограниченной длины	name varchar(40)
text	строки произвольной длины	message text
date	дата (без времени)	birthday date
timestamp	дата + время	created_at timestamp
boolean	булевые значений	active boolean
jsonb	JSON-поля	data jsonb

Типы полей: <https://www.postgresql.org/docs/12/datatype.html>

DDL (Data Definition Language). CONSTRAINTS

Ограничение	Описание	Пример
primary key	первичный ключ, обязывает поле быть уникальным и не пустым	<code>id serial primary key</code>
not null	значение не может быть пустым (не может отсутствовать)	<code>name varchar(40) not null</code>
unique	все значения в этом поле должны быть уникальным	<code>tag varchar(80) unique</code>
check	добавить проверку значения на описанное условие	<code>price numeric check(price > 0)</code>
foreign key	внешний ключ, обязывает значение соответствовать значению из другой таблицы	<code>product_id integer references products(id)</code>

Primary Key

Primary Key (первичный ключ) - это отдельное поле или комбинация полей, которые однозначно определяют запись - кортеж.

```
create table if not exists Student (
    id serial primary key,
    name varchar(40) not null,
    gpa numeric(3, 2) check (gpa >= 0 and gpa <= 5)
);

create table if not exists Student (
    name varchar(40) primary key,
    gpa numeric(3, 2) check (gpa >= 0 and gpa <= 5)
); # какой недостаток?

create table if not exists Student (
    name varchar(40),
    gpa numeric(3, 2) check (gpa >= 0 and gpa <= 5),
    constraint student_pk primary key (name, gpa)
); # здесь все ок?
```

Связи между отношениями

Связи между отношениями

Модель ситуации:

Есть система онлайн обучения.

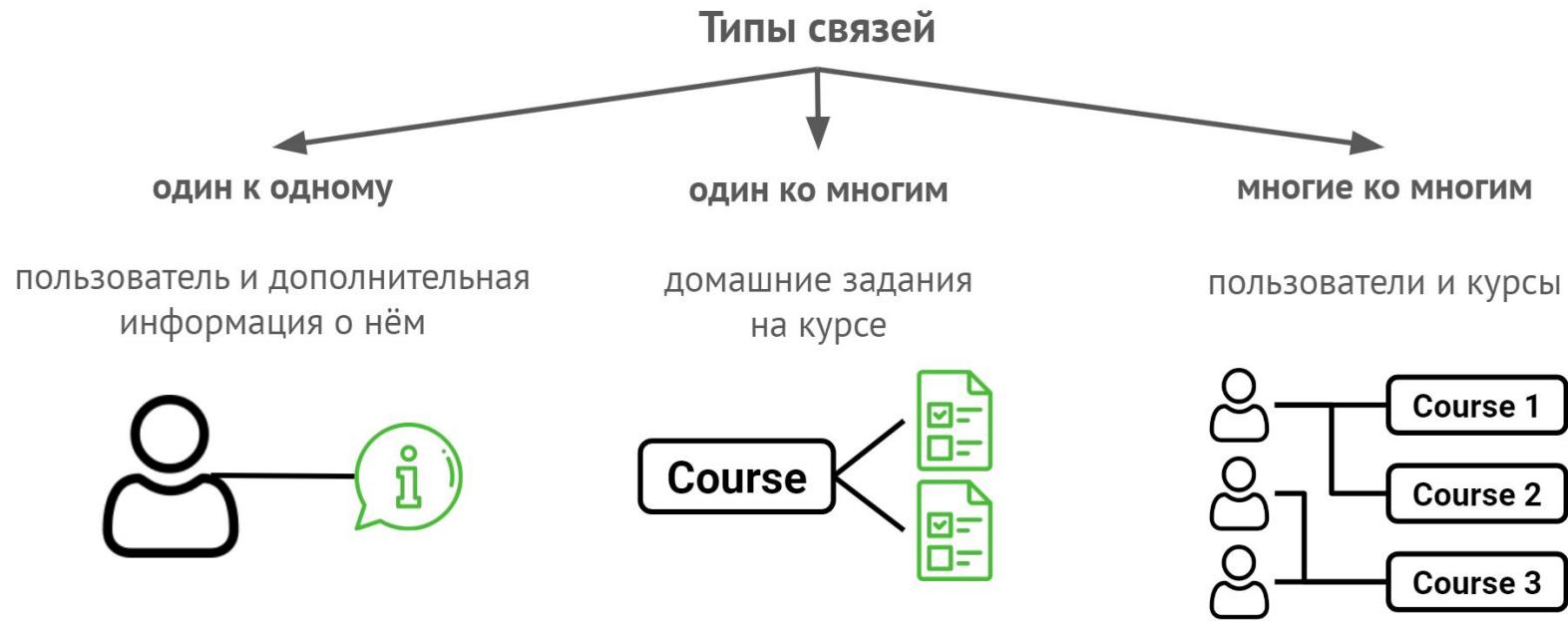
В ней существуют пользователи - студенты. У каждого пользователя есть почта (она же выступает логином), пароль и имя.

Также у пользователя есть возможность указать дополнительную информацию: дату рождения, город, интересы.

Пользователи могут записываться на курсы.

В рамках курса пользователи должны выполнять домашние задания и загружать их в систему.

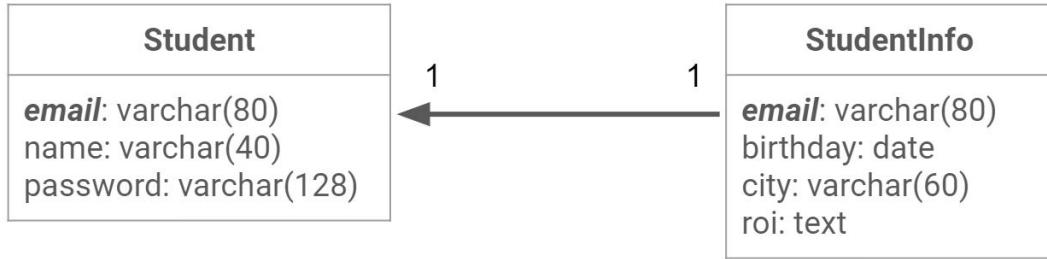
Связи между отношениями



Связи, как и первичный ключ, можно попросить контролировать СУБД.
Для этого используется ограничение **foreign key**.

ОДИН К ОДНОМУ

Связываем студента и дополнительную информацию о нём.

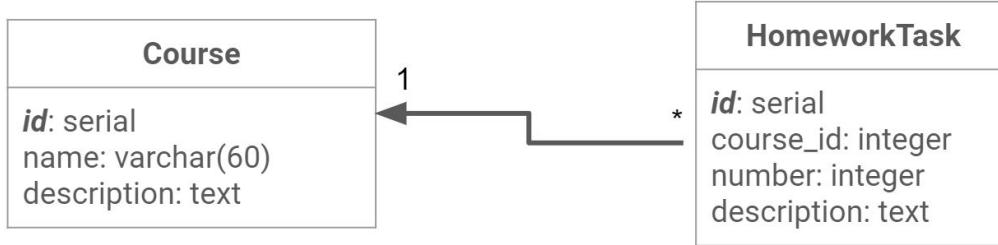


```
create table if not exists Student (
    email varchar(80) primary key,
    name varchar(40) not null,
    password varchar(128) not null
);

create table if not exists StudentInfo (
    email varchar(80) primary key references Student(email),
    birthday date,
    city varchar(60),
    roi text
);
```

ОДИН КО МНОГИМ

Связываем описание домашних заданий с курсами, к которым они относятся.

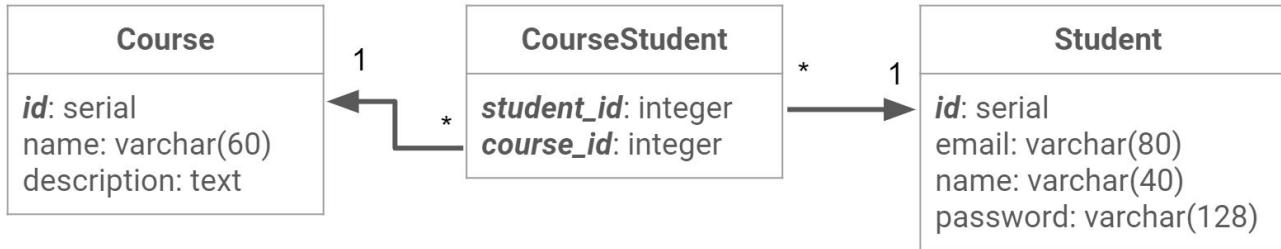


```
create table if not exists Course (
    id serial primary key,
    name varchar(60) not null,
    description text
);

create table if not exists HomeworkTask (
    id serial primary key,
    course_id integer not null references Course(id),
    number integer not null,
    description text not null
);
```

Многие ко многим

Связываем студентов и курсы, на которые они записаны.

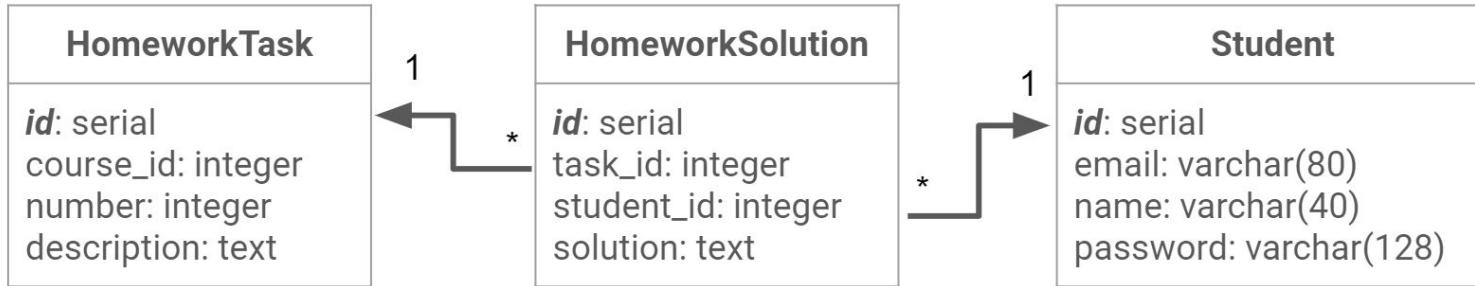


```
create table if not exists CourseStudent (
    course_id integer references Course(id),
    student_id integer references Student(id),
    constraint pk primary key (course_id, student_id)
);
```

student_id	course_id
1 (Вова)	1 (Python)
1 (Вова)	2 (Java)

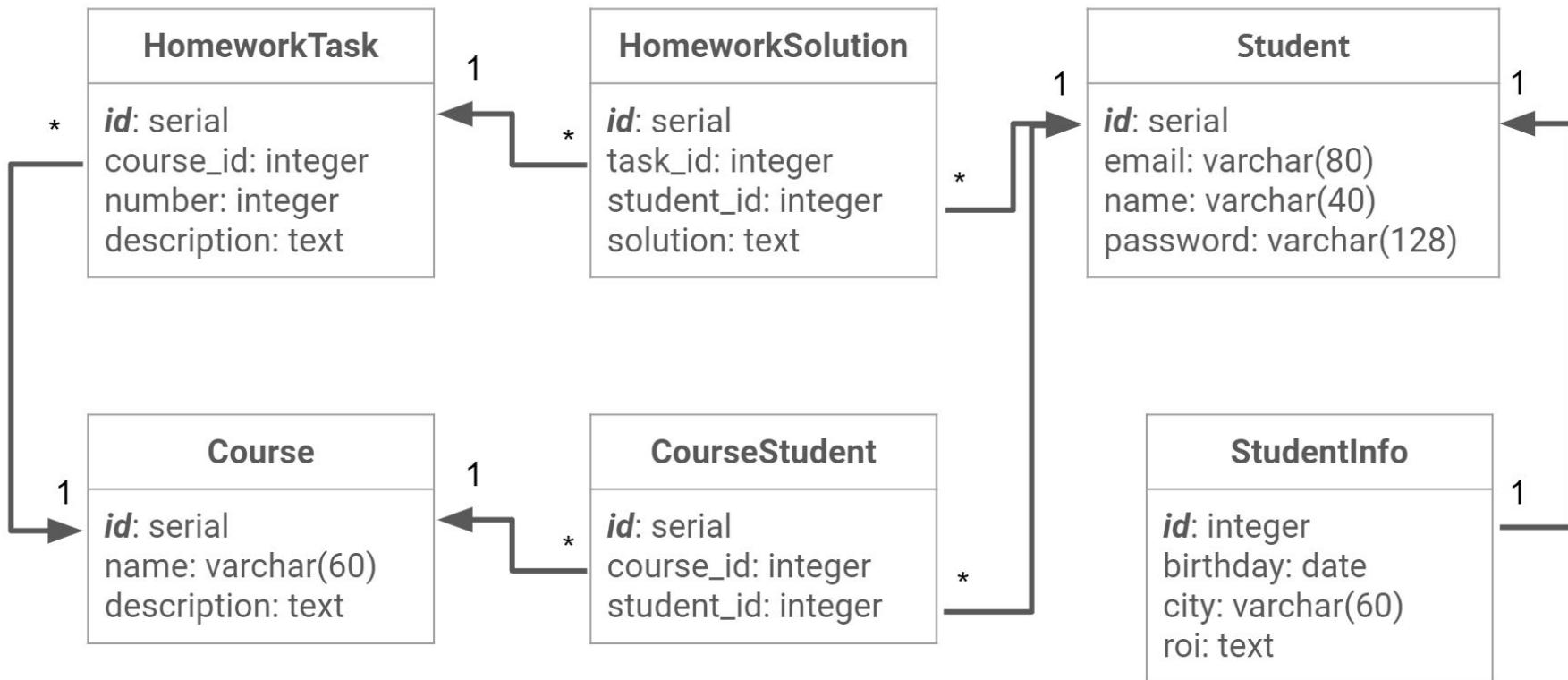
Многие ко многим

Связываем студента и домашние работы, которые он отправил в систему.



```
create table if not exists HomeworkSolution (
    id serial primary key,
    task_id integer not null references HomeworkTask(id),
    student_id integer not null references Student(id),
    solution text not null
);
```

Примерная схема



Нормальные формы (НФ)

Нормализация – процесс постепенного преобразования отношений (таблиц) для того, чтобы убрать дублирование данных. Это помогает уменьшить потенциальную противоречивость в БД.

HomeworkTask

<u>id</u>	course	number	description
1	Python	1	...
2	Python	2	...
3	JavaScript	1	...
4	Python	3	...

Первая нормальная форма (1НФ)

Сохраняемые данные на пересечении строк и столбцов должны представлять скалярное значение, а таблицы не должны содержать повторяющихся строк.

<u>Сотрудник</u>	<u>Номер телефона</u>
Иванов И. И.	283-56-82, 390-57-34
Петров П. П.	708-62-34

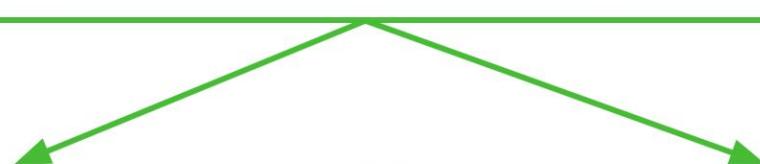


<u>Сотрудник</u>	<u>Номер телефона</u>
Иванов И. И.	283-56-82
Иванов И. И.	390-57-34
Петров П. П.	708-62-34

Вторая нормальная форма (2НФ)

1НФ + каждый столбец, который не является ключом, должен зависеть от первичного ключа.

<u>Филиал компании</u>	<u>Должность</u>	Зарплата	Наличие графического планшета
Филиал в Варшаве	Дизайнер	40000	Есть
Филиал в Минске	Программист	60000	Нет
Филиал в Минске	Программист	40000	Нет



<u>Филиал компании</u>	<u>Должность</u>	Зарплата	<u>Должность</u>	Наличие графического планшета
Филиал в Варшаве	Дизайнер	40000	Дизайнер	Есть
Филиал в Минске	Программист	60000	Программист	Нет
Филиал в Минске	Программист	40000		

Третья нормальная форма (ЗНФ)

2НФ + каждый столбец, который не является ключом, должен зависеть от первичного ключа **напрямую**.

<u>Сотрудник</u>	<u>Отдел</u>	<u>Телефон</u>
Гришин	Бухгалтерия	11-22-33
Васильев	Бухгалтерия	11-22-33
Петров	Снабжение	44-55-66



<u>Сотрудник</u>	<u>Отдел</u>
Гришин	Бухгалтерия
Васильев	Бухгалтерия
Петров	Снабжение

<u>Отдел</u>	<u>Телефон</u>
Бухгалтерия	11-22-33
Снабжение	44-55-66

Прочие нормальные формы

Всего в реальность 8 нормальных форм, но достаточно знаний 3-х

Какие еще есть нормальные формы:

- Нормальная форма Бойса-Кодда
- Четвертая нормальная форма
- Пятая нормальная форма
- Шестая нормальная форма

Статья с примерами <https://habr.com/ru/post/254773/>

Плюсы и минусы нормализации

Плюсы	Минусы
Декомпозиция информации	Время на приведение к нормальным формам
Строгое хранение данных без возможности хранить дублированную и противоречивую информацию	Накладные расходы при извлечении информации на объединение таблиц

Практика

Создайте таблицы с прошлой практики и установите связи между ними.

Напоминание:

Есть категории интернет-магазина и есть товары. Каждый товар принадлежит строго одной категории. К товарам могут написать отзывы (к одному товару можно написать множество отзывов). Необходимо хранить информацию о категориях, товарах и отзывах.



DDL (Data Definition Language). ALTER и DROP

```
alter table <name> ...
# добавить атрибут
add column <col_name> <col_type> [constraints];

# переименовать таблицу
rename to <new_table_name>;

# переименовать атрибут
rename <col_name> to <new_col_name>;

# изменить тип атрибута
alter column <col_name> set data type <col_type>;

# добавить ограничение
add constraint <constraint_name> <constraint>;

# удалить ограничение
drop constraint <constraint_name>;

# удалить атрибут
drop column <col_name>;
```



```
# удалить таблицу
drop table <name>;
```

SELECT запросы

База данных Pagila

Огромный набор данных по фильмам, который позволит вам попрактиковаться.

Скачивание:

<https://www.postgresqltutorial.com/postgresql-sample-database/>

https://dataedo.com/samples/html/Pagila/doc/Pagila_10/modules/Pagila_database_diagram_103/module.html

SELECT запросы

SELECT-запросы предназначены для отбора необходимых строк или столбцов из одной или нескольких таблиц.

Общий вид структуры select-запроса:

```
SELECT [DISTINCT | ALL] table_columns  
FROM table  
[WHERE condition]  
[GROUP BY table_columns]  
[HAVING condition]  
[ORDER BY table_columns [ASC | DESC]]  
[LIMIT number]
```

Обязательные элементы запроса

Необязательные элементы запроса
в строгой последовательности

SELECT запросы и **DISTINCT**

```
SELECT * FROM table;
```

* – выбор всех столбцов таблицы;

table – имя нужной таблицы.

Через запятую можно перечислить имена необходимых столбцов:

```
SELECT title, release_year FROM film;
```

Оператор **DISTINCT** позволяет выбирать только уникальные значения из базы данных – он отсеивает дубли:

```
SELECT DISTINCT rating FROM film;
```

Арифметические Операции

Можно использовать различные арифметические операторы для данных, хранящихся в таблицах:

- $+$ – сложение;
- $-$ – вычитание;
- $/$ – деление;
- $*$ – умножение;
- $\%$ – взятие остатка от деления.

Оператор WHERE и фильтрация по условиям

Оператор **WHERE** нужен для фильтрации таблиц по условиям.

В качестве условий можно использовать:

- сравнения (`=, >, <, >=, <=, !=`);
- оператор **IN**;
- оператор **BETWEEN**;
- оператор **LIKE**.

С условиями можно применять логические операторы **and**, **or** и **not**:

- Оператор **AND** отображает запись, если оба операнда истинны;
- Оператор **OR** отображает запись, если хотя бы один operand истинен;
- Оператор **NOT** инвертирует исходный operand.

Примеры запросов с фильтрацией

```
SELECT title, release_year FROM film  
WHERE release_year >= 2000;
```

```
SELECT first_name, last_name, active FROM staff  
WHERE NOT active = true;
```

```
SELECT title, rental_rate, replacement_cost FROM film  
WHERE rental_rate <= 0.99 AND replacement_cost <= 9.99;
```

Примеры запросов с фильтрацией с оператором IN

Оператор **IN** отбирает строки, в которых есть перечисленные значения. Если значений много – они перечисляются через запятую. Например:

```
SELECT title, description, rating FROM film  
WHERE rating IN ('R', 'NC-17');
```

```
SELECT title, description, rating FROM film  
WHERE rating NOT IN ('G', 'PG');
```

Примеры запросов с фильтрацией с оператором BETWEEN

Оператор **BETWEEN** позволяет проверить, находится ли выражение в диапазоне значений. Крайние значения включаются в диапазон.

```
SELECT title, rental_rate FROM film  
WHERE rental_rate BETWEEN 0.99 AND 1.99;
```



```
SELECT title, rental_rate FROM film  
WHERE rental_rate NOT BETWEEN 0.99 AND 1.99;
```



Примеры запросов с фильтрацией с оператором LIKE

Оператор **LIKE** позволяет проверить, находится ли в значении заданный текстовый шаблон.

В шаблонах кроме обычных символов могут использоваться два служебных:

- **%** – ноль, один или несколько любых символов,
- **_** – один любой символ.

```
SELECT title, description FROM film  
WHERE description LIKE '%Scientist%';
```

Важно. В python знак **%** зарезервирован, поэтому необходимо использовать **%%**.

Сортировка при помощи ORDER BY

ORDER BY используется для сортировки таблицы по указанным столбцам.

```
SELECT title, rental_rate FROM film  
    ORDER BY rental_rate;
```

```
SELECT title, rental_rate FROM film  
    ORDER BY rental_rate DESC;
```

```
SELECT title, length, rental_rate FROM film  
    ORDER BY length DESC, rental_rate;
```

Оператор LIMIT

LIMIT используется, чтобы ограничивать количество возвращаемых записей из одной или нескольких таблиц.

```
SELECT title, length, rental_rate FROM film
  WHERE rental_rate > 2.99
  ORDER BY length DESC, rental_rate
  LIMIT 15;
```

Aggregation Functions

Агрегирующие функции выполняют вычисления над значениями в столбце.

Существует 5 таких функций:

- **SUM()** – вычисляет сумму значений;
- **MIN()** – вычисляет минимальное значение;
- **MAX()** – вычисляет максимальное значение;
- **AVG()** – вычисляет среднее значение;
- **COUNT()** – вычисляет количество строк в столбце.

Вложенные запросы

Вложенный запрос – запрос, который используется внутри другого запроса. Каждый вложенный запрос так же может содержать один или несколько вложенных запросов. Количество вложенных запросов неограничено.

Вложенные подзапросы обрабатываются «снизу вверх». Сначала обрабатывается вложенный запрос самого нижнего уровня. Далее значения, полученные по результату его выполнения, передаются и используются при исполнении подзапроса более высокого уровня и т.д.

Например:

```
SELECT title, length FROM film -- данный запрос будет выполнен вторым
    WHERE length >= (
        SELECT AVG(length) FROM film); -- сначала будет выполнен этот запрос
```

GROUP BY

GROUP BY – оператор, с помощью которого можно задать агрегацию по нужным столбцам. Применяется в связке с агрегирующими функциями.

```
SELECT rating, COUNT(title) FROM film  
GROUP BY rating;
```

```
SELECT rating, AVG(length) FROM film  
WHERE release_year = 2006  
GROUP BY rating;
```

Группировочные столбцы должны обязательно присутствовать в **SELECT**.

HAVING

HAVING – оператор, позволяющий отфильтровать данные **после** группировки (является аналогом WHERE).

```
SELECT last_name FROM actor  
GROUP BY last_name  
HAVING COUNT(last_name) = 1;
```

Не путайте: WHERE фильтрует данные до группировки, а HAVING после.

ALIAS

Alias (псевдонимы) используются для временного переименования таблиц и столбцов. Это необязательный функционал, но может быть очень полезным и удобным, когда имена длинные и сложные или они повторяются много раз в рамках запроса (пригодится при объединении!). По этой причине псевдонимы делают максимально краткими.

```
SELECT column_name [AS] column_aliases  
FROM table_name [AS] table_aliases
```

Оператор **AS** является необязательным.

При использовании псевдонимов важно знать и помнить о техническом порядке исполнения запросов. **WHERE** и **HAVING** исполняются **до SELECT**, поэтому в них псевдонимы использовать не получится.

Объединение таблиц

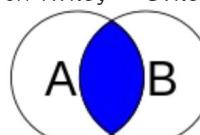
JOIN – оператор, предназначенный для объединения таблиц по определенному столбцу или связке столбцов (как правило, по первичному ключу).

JOIN записывается после **FROM** и до **WHERE**. Через оператор **ON** необходимо явно указывать столбцы, по которым происходит объединение. В общем виде структура запросы с **JOIN** выглядит так:

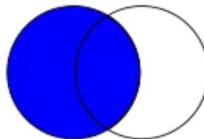
```
SELECT tables_columns FROM table_1  
JOIN table_2 ON table_1.column = table_2.column;
```

Виды JOIN'ов

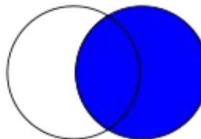
```
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key
```



```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
```

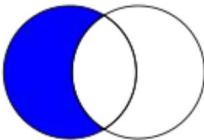


```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
```

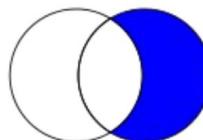


SQL JOINS

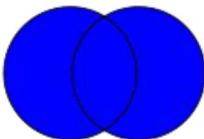
```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL
```



```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
```



```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
```



```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```



Виды JOIN'ов

[INNER] JOIN – в выборку попадут только те данные, по которым выполняются условия соединения;

LEFT [OUTER] JOIN – в выборку попадут все данные из таблицы **A** и только те данные из таблицы **B**, по которым выполнится условие соединения. Недостающие данные вместо строк таблицы **B** будут представлены **NULL**.

RIGHT [OUTER] JOIN – в выборку попадут все данные из таблицы **B** и только те данные из таблицы **A**, по которым выполнится условие соединения. Недостающие данные вместо строк таблицы **A** будут представлены **NULL**.

FULL [OUTER] JOIN – в выборку попадут все строки таблицы **A** и таблицы **B**. Если для строк таблицы **A** и таблицы **B** выполняются условия соединения, то они объединяются в одну строку. Если данных в какой-то таблице не имеется, то на соответствующие места вставляются **NULL**.

CROSS JOIN – объединение каждой строки таблицы **A** с каждой строкой таблицы **B**.

INNER JOIN

[INNER] JOIN – в выборку попадут только те данные, по которым выполняются условия соединения.

Левая таблица (к ней присоединяем)

employees		
<u>id</u>	<u>name</u>	<u>office_id</u>
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленных

Правая таблица

offices	
<u>id</u>	<u>office_name</u>
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем только тех сотрудников, которые сидят в кабинетах и только те кабинеты, в которых сидят сотрудники.



<u>employees.id</u>	<u>name</u>	<u>offices.id</u>	<u>office_name</u>
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126

LEFT JOIN

LEFT JOIN – в выборку попадут все данные из левой таблицы и только те данные из правой, по которым выполняется условие соединения.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленных

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем всех сотрудников и только те кабинеты, в которых кто-то сидит.

employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126
4	Mary	NULL	NULL

RIGHT JOIN

RIGHT JOIN – в выборку попадут все данные из правой таблицы и только те данные из левой, по которым выполняется условие соединения.

Левая таблица (к ней присоединянем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленщиков

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем все кабинеты и только тех сотрудников, которые сидят в них (не удаленщиков).



employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126
NULL	NULL	4	Кабинет 22

LEFT JOIN с фильтрацией по полю

LEFT JOIN с фильтрацией по полю – в выборку попадут только те данные из левой таблицы, по которым не выполняется условия соединения.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленных

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем только удаленных.

employees.id	name	offices.id	office_name
4	Mary	NULL	NULL

RIGHT JOIN с фильтрацией по полю

RIGHT JOIN с фильтрацией по полю – в выборку попадут только те данные из правой таблицы, по которым не выполняется условия соединения.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленных

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем только пустые
кабинеты.

employees.id	name	offices.id	office_name
NULL	NULL	4	Кабинет 22

FULL OUTER JOIN

FULL OUTER JOIN – выборку попадут все строки исходных таблиц. Если по данным не выполняется условие соединения, то на соответствующие места вставляются **NULL**.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленных

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем абсолютно все кабинеты и абсолютно всех сотрудников.

employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126
4	Mary	NULL	NULL
NULL	NULL	4	Кабинет 22

INDICES (Индексы)

Индексы – это специальные объекты баз данных, которые нужны для ускорения получения данных из них.

Для добавления индекса используется команда **CREATE INDEX**.

```
CREATE INDEX index_name  
ON table_name (columns_names);
```

Удалить индекс можно так:

```
DROP INDEX index_name;
```

Когда использовать индексы

Индексы ускоряют запросы на получение данных (**SELECT**), но замедляют на изменение (**UPDATE/INSERT**). Выбор столбца/столбцов для создания индексов зависит от того, какие столбец/столбцы чаще используются для фильтрации в **WHERE**.

Индексы использовать не рекомендуется, когда:

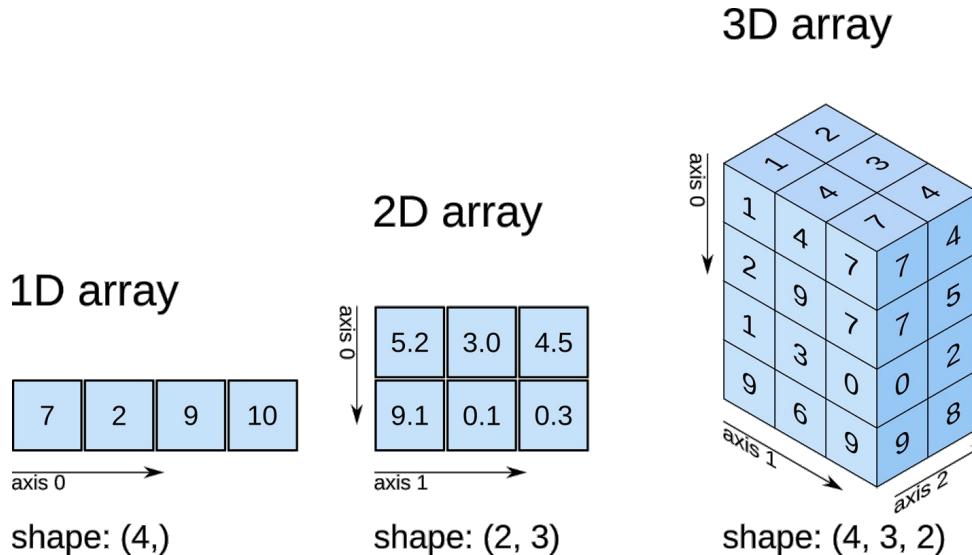
- таблицы в БД небольшие;
- данные в таблицах/столбцах часто меняются;
- в столбце много NULL-значений.

<https://habr.com/ru/post/247373/>

2. Numpy. Введение в библиотеку.

NumPy.

NumPy (**N**umerical **P**ython) - пакет для научных вычислений в Python. Библиотека предоставляет огромное количество методов обработки многомерных массивов, различных производных объектов (например матриц), а также позволяет производить математические, логические, I/O операции, операции связанные с базовой линейной алгеброй, базовыми статистическими вычислениями, статистическими симуляциями и много чего ещё.



Почему NumPy.

- Скорость вычислений в NumPy намного выше
- Все операции выполняются в скомпилированном **коде (C++)**

Где применяется библиотека NumPy.

- Полноценная **замена вычислениям в MatLab**
- Математические вычисления
- Визуализация математических вычислений
- **Backend** (быстрая обработка данных, обработка изображений)
- Машинное обучение

**Погнали в
Jupyter Notebook**

3. Pandas. Введение в EDA.

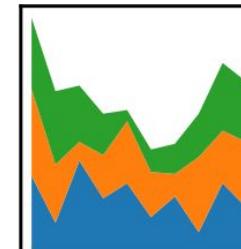
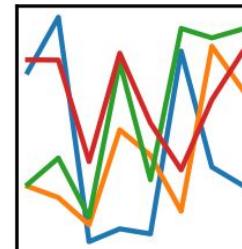
Pandas.

Pandas — это библиотека, написанная для языка программирования Python для обработки и анализа табличных данных. Специалисты по данным часто работают с данными, хранящимися в табличных форматах, таких как .csv, .tsv или .xlsx.

В Pandas очень удобно загружать, обрабатывать и анализировать такие табличные данные с помощью SQL-подобных запросов. В связке с **Matplotlib** и **Seaborn**, Pandas предоставляет широкие возможности для визуального анализа табличных данных.

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

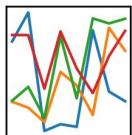
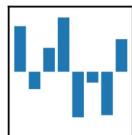


Pandas. Основные возможности библиотеки

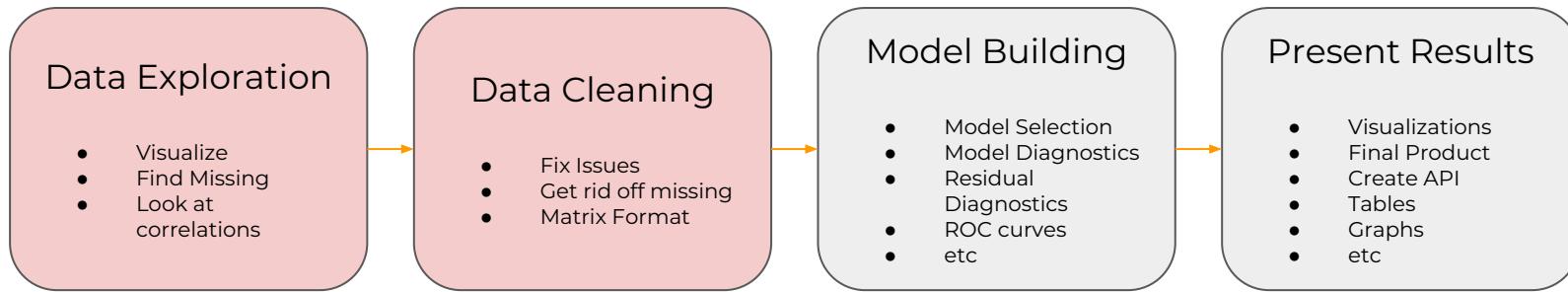
1. Под капотом реализован объект **DataFrame** для манипулирования индексированными массивами двумерных данных
2. Реализован при помощи Cython.
3. Реализованы инструменты для обмена данными между структурами в памяти и файлами разных форматов
4. Присутствует огромное количество методов для обработки данных
5. Существует возможность переформатирования наборов данных, в том числе для создания сводных таблиц
6. Есть возможность среза данных по значениям индекса, расширенные возможности индексирования датасета, осуществление выборки из больших наборов данных
7. Используется в качестве главного инструмента для анализа датасетов малого и среднего объема.

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Pandas. EDA.



Pandas. Структура данных Series и DataFrame.

Ядром pandas являются две структуры данных, в которых происходят все операции:

- Series
- DataFrame

Series — это структура, используемая для работы с последовательностью одномерных данных.

Dataframe — более сложная и подходит для нескольких измерений.

Pandas. Структура данных Series.

Series — это объект библиотеки pandas, спроектированный для представления одномерных структур данных, похожих на массивы, но с дополнительными возможностями. Такая структура состоит из двух связанных между собой массивов. Основной столбец (колонка) содержит данные ([типа NumPy](#)), а в дополнительном (index) хранятся метки (индексы).

SERIES

index	value
0	2
1	12
2	72
3	432
...	...

Pandas. Структура данных DataFrame.

Dataframe — это табличная структура данных, напоминающая таблицы из Microsoft Excel.

Ее главная задача — позволить использовать многомерные Series. Dataframe состоит из упорядоченной коллекции колонок, каждая из которых содержит значение разных типов (числовое, строковое, булевое и так далее).

DataFrame Object					
index	columns				
	Age	Gender	Location	...	
0	25	M	London	...	
1	20	F	Warsaw	...	
2	27	F	Amsterdam	...	
3	30	M	Minsk	...	
...

Pandas. Структура данных DataFrame.

В отличие от **Series** у которого есть массив индексов с метками, ассоциированных с каждым из элементов, **Dataframe** имеет сразу два таких.

Первый ассоциирован **со строками (рядами)**. Каждая метка ассоциирована со всеми значениями в ряду. **Второй** содержит **метки для каждой из колонок**.

Dataframe можно воспринимать как dict, состоящий из Series, где:

ключи — названия колонок

значения — объекты Series, которые формируют колонки самого объекта Dataframe. Наконец, все элементы в каждом объекте Series связаны в соответствии с массивом меток, называемым index.

**Погнали в
Jupyter Notebook**

Q&A

Дополнительные материалы

Теория:

Книга [Python Data Science Handbook](#) Прочтайте главы:

- [Introduction to NumPy](#)
- [Data Manipulation with Pandas](#)

Чтобы быстро освежить в памяти нужную конструкцию, вот несколько шпаргалок:

- [NumPy](#)
- [Pandas](#)

Практика:

Набивайте руку на задачах:

1. Тренируйте навык. Создайте Jupyter Notebook и повторите все действия из [10 Minutes to pandas](#) - официального введения в библиотеку.
2. Большой ноутбук [100-pandas-puzzles](#). Порешайте задачи.
3. Не только копируйте код, но и понимайте зачем вы все это делаете и для чего. Советую поработать немного с ноутбуками из [репозитория](#). А именно закрепите навыки:
 - [Getting and Knowing Your Data](#)
 - [Filtering and Sorting Data](#)
 - [Grouping](#)
 - [Apply](#)
 - [Merge](#)
 - [Stats](#)
 - [Creating Series and DataFrames](#)
 - [TimeSeries](#)

Дополнительные материалы

SQL:

- Небольшой [тutorиал](#) по работе с запросами SQL
- [Шпаргалка по SQL](#) для того, чтобы быстро вспомнить основные команды и операторы
- Выполняйте упражнения на HackerRank в [разделе SQL](#)

Продвинутый уровень SQL:

- [Тutorиал от Kaggle](#) по продвинутым возможностям SQL (объединения, аналитические функции, вложенные запросы и циклические расчеты, оптимизация запросов)
- Дополнительные [продвинутые техники](#) для работы с запросами
- [Большой тьюториал по SQL](#) для целей аналитики и анализа данных