

JS Level 3

Node.js



ПРЕДИСЛОВИЕ



Предисловие

На прошлых лекциях мы организовали CRUD-сервис. При этом все данные мы хранили в памяти (т.е. при завершении работы нашего приложения все данные исчезали).

Конечно же, это не очень хорошо (и в реальных системах так не делают). Мы, конечно, можем всё хранить в файлах, но это тоже не очень удобно, поскольку нам буквально "вручную" потребуется доставать информацию оттуда и аккуратно складывать.



Предисловие

Поэтому для задачи постоянного и удобного хранения данных придумали специальное решение – системы управления базами данных (сокращённо СУБД).

Эти системы умеют эффективно хранить и обрабатывать большие объёмы данных, предоставляя нам удобный интерфейс для работы с ними.

С одним из подобных решений, а именно СУБД PostgreSQL мы и познакомимся.

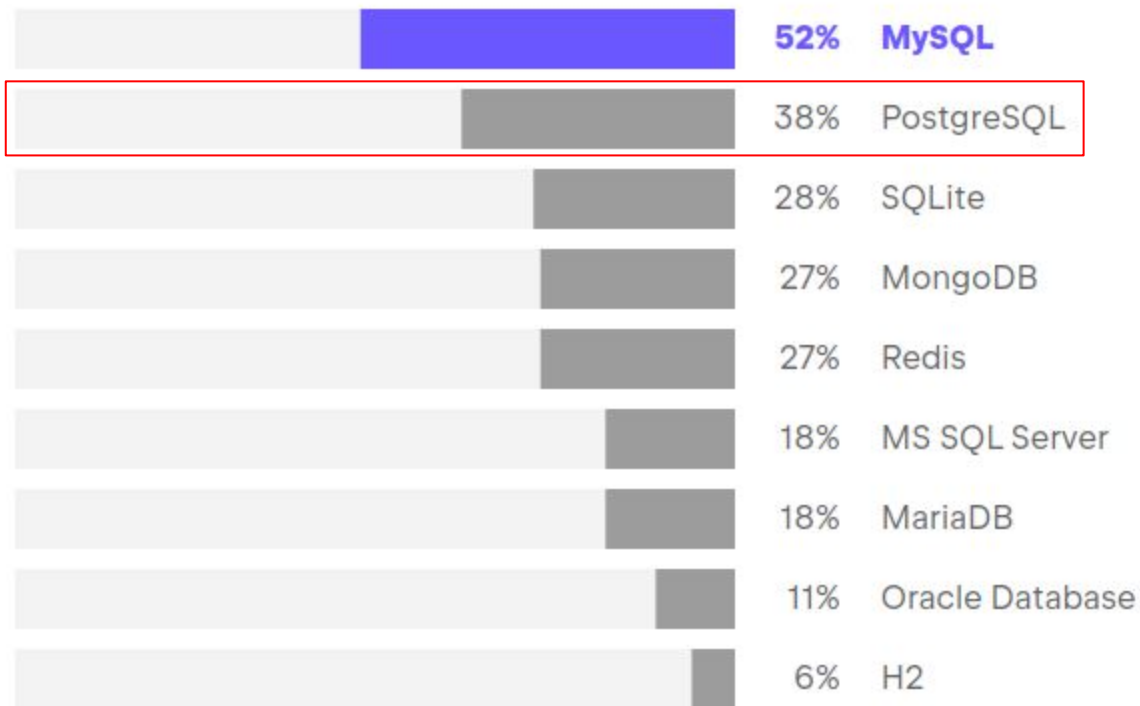


DATABASES



Databases

Базы данных (databases) – отдельные приложения, отвечающие за хранение данных.



<https://www.jetbrains.com/ru-ru/lp/devecosystem-2022/databases/>



Databases

Ряд СУБД используют SQL – это модель данных, которая позволяет представлять данные в виде набора связанных таблиц. Например:

↓ столбец (одно поле записи)

Таблица платежи (payments)			
id	senderId	recipientId	amount
1	1	2	1000
2	1	2	5000

← строка (одна запись)

Таблица пользователи (users)			
id	name	login	password
1	Василий	vasya	secret
2	Мария	masha	secret



Databases

Если проводить аналогию с JS, то таблица – это массив (в котором мы раньше хранили, например, посты), а каждая запись – это отдельный объект в этом массиве.

Важно: это именно аналогия.



Databases

СУБД предоставляет нам следующие ключевые механизмы:

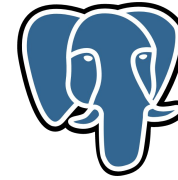
- запросов (найти все платежи Василия, найти 5 самых последних платежа и т. д.)
- целостности (нельзя вставить платёж с таким `senderId` или `recipientId`, которого нет в табличке `users`)
- транзакций (возможности выполнять последовательность операций как "единую логическую операцию", которая применяется либо целиком, либо не применяется вовсе – например, перевод средств между пользователями)
- и т.д.



POSTGRESQL



PostgreSQL



PostgreSQL – промышленная СУБД с открытым исходным, используемая в большом количестве проектов.

Предоставляет достаточно гибкие возможности для организации хранения данных и извлечения их.



PostgreSQL

Для установки PostgreSQL существует два способа:

1. Установка обычного приложения
2. Установка с помощью Docker

Второй способ является рекомендуемым (и современным), поэтому обязательно попробуйте сначала установить Docker.

И только если не получится, устанавливайте как обычное приложение.



Установка

Для установки в качестве обычного приложения перейдите по ссылке <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads> и скачайте установочный файл для вашей операционной системы (мы всё будем рассматривать на примере Windows):

PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
15.3	postgresql.org 	postgresql.org 			Not supported
14.8	postgresql.org 	postgresql.org 			Not supported
13.11	postgresql.org 	postgresql.org 			Not supported
12.15	postgresql.org 	postgresql.org 			Not supported
11.20	postgresql.org 	postgresql.org 			Not supported
10.23*					

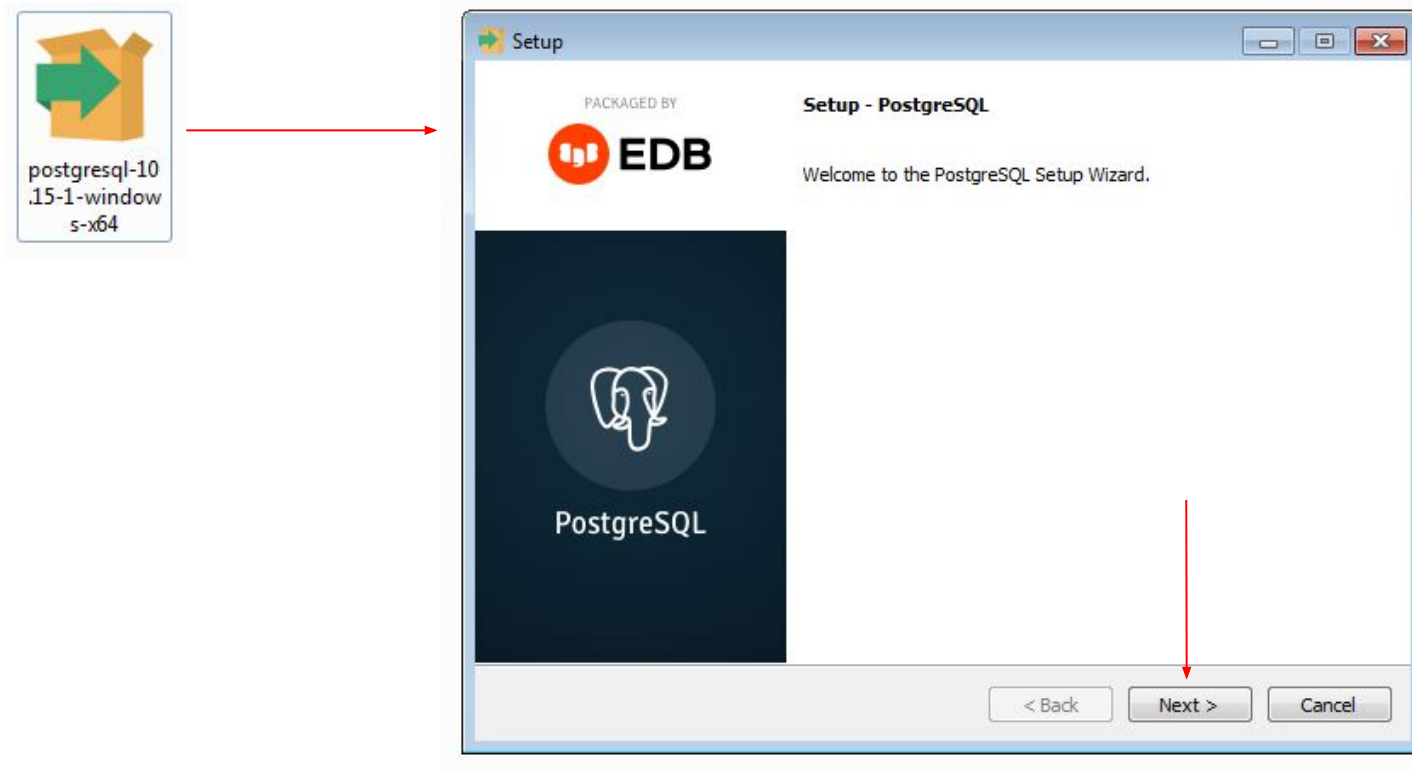
PostgreSQL

Мы будем использовать 10-ую версию (вы можете использовать любую не младше 10-ой)



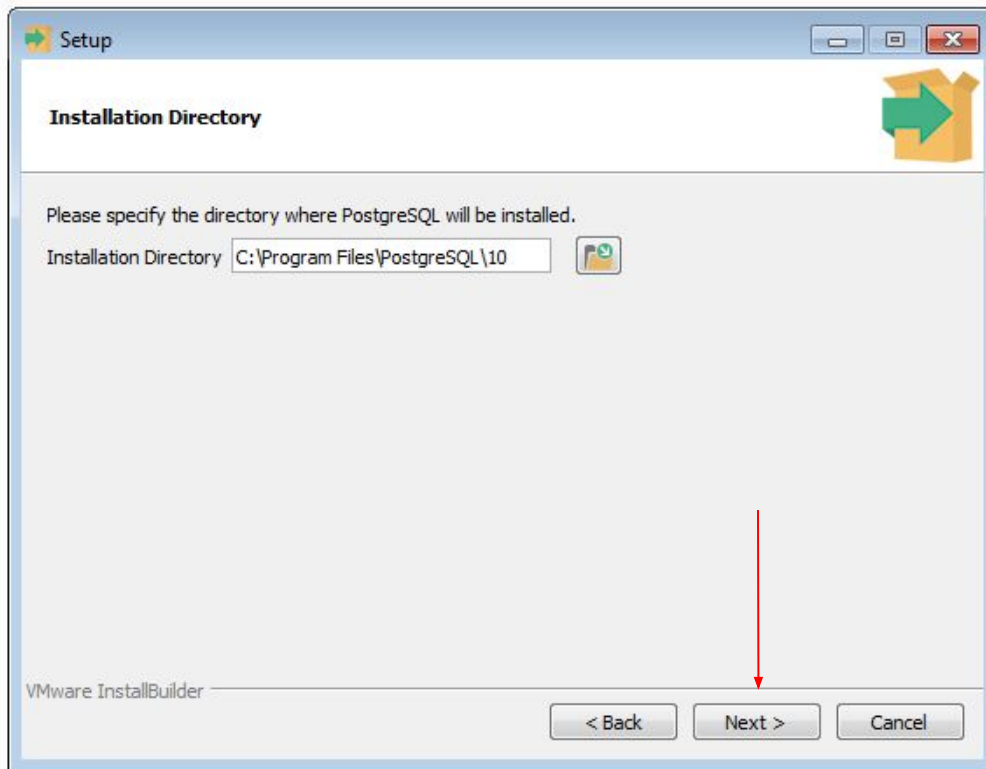
PostgreSQL

После скачивания нужной версии запустите установочный файл:



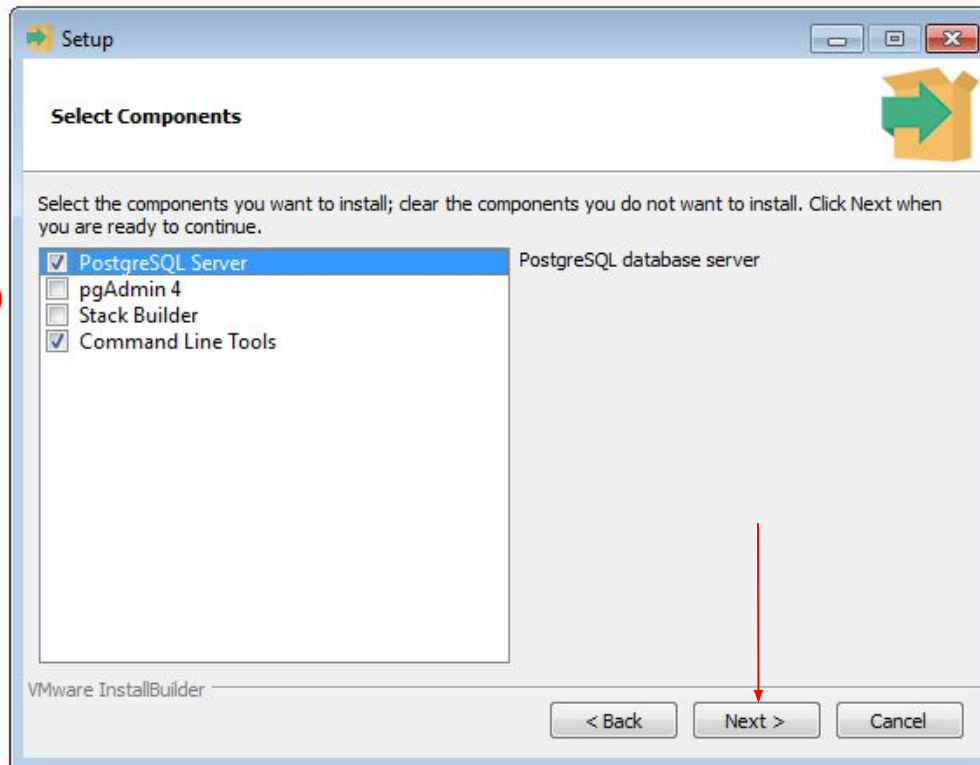
PostgreSQL

Оставьте установочный каталог по умолчанию:



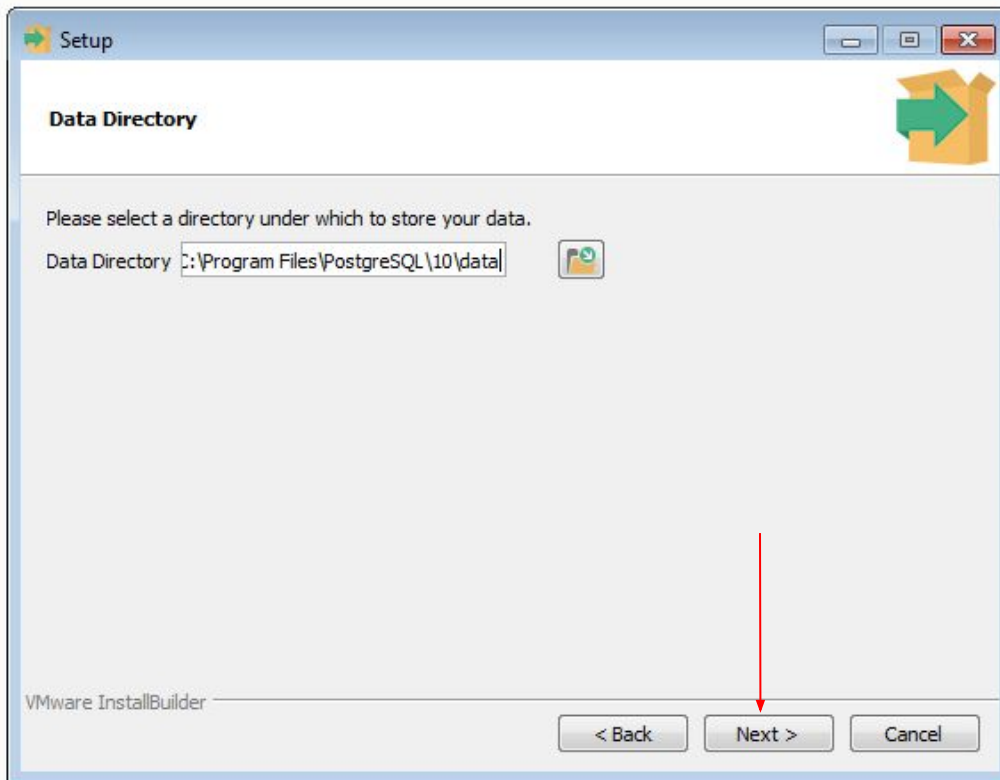
PostgreSQL

Оставьте флажки так, как указано на экране:



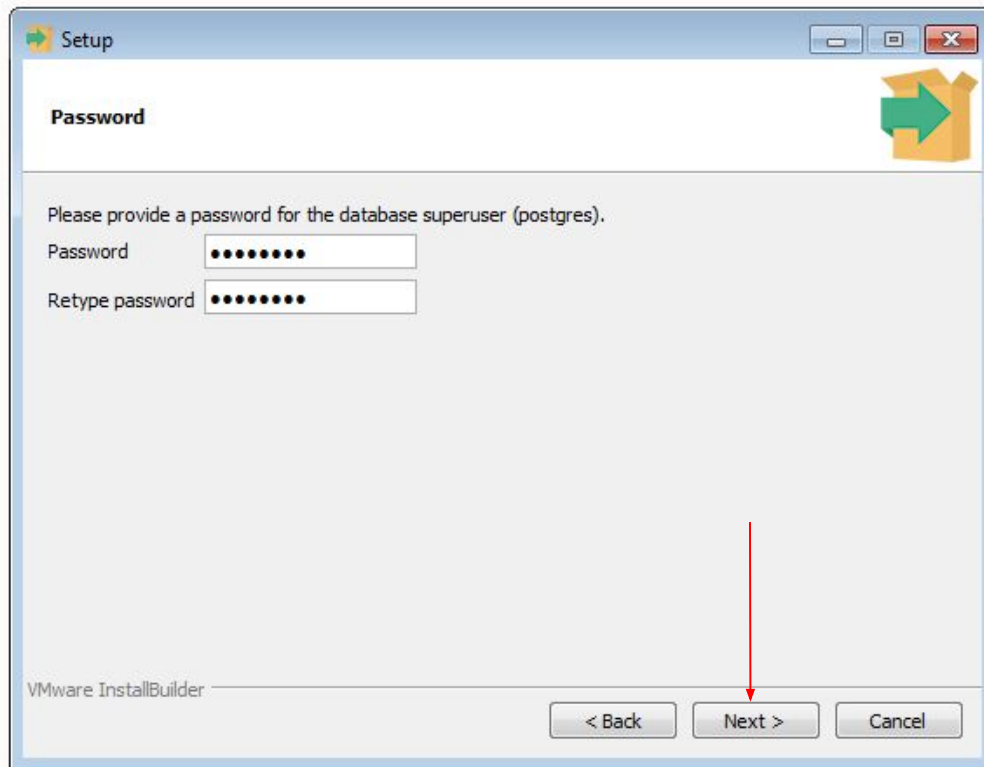
PostgreSQL

Оставьте каталог для данных по умолчанию:



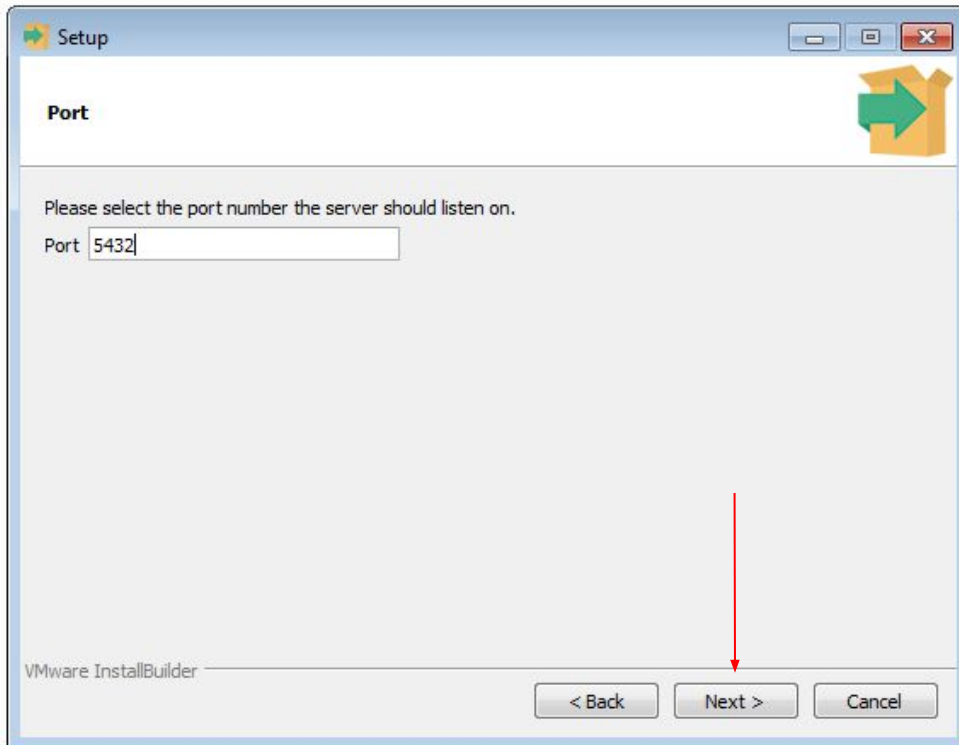
PostgreSQL

Введите логин и пароль суперпользователя (рекомендуем использовать для обоих значений **postgres**):



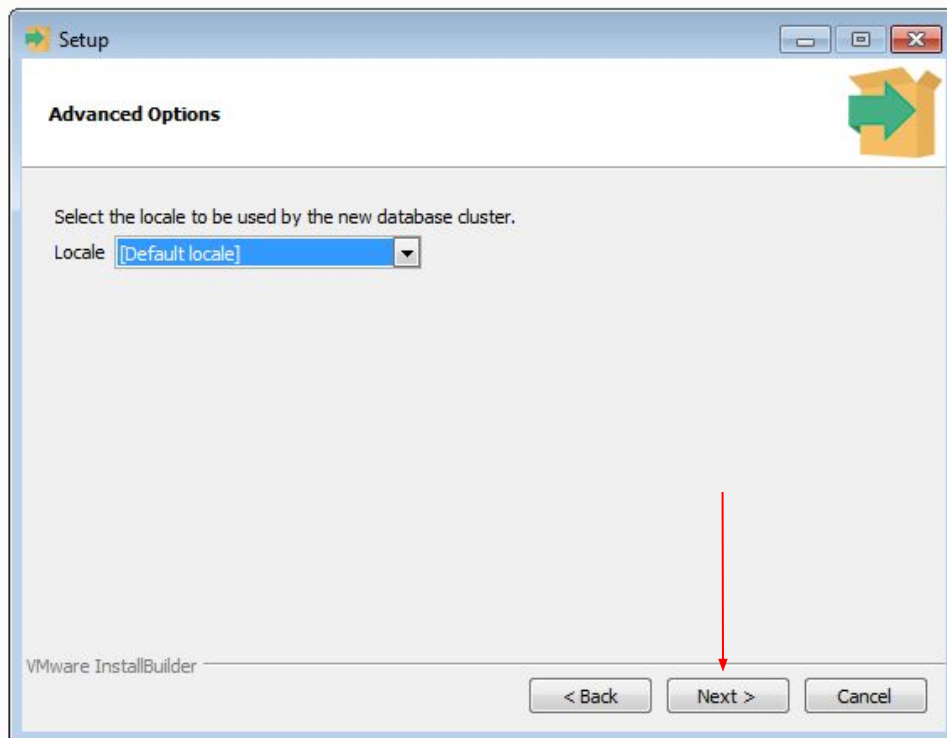
PostgreSQL

Оставьте порт 5432 (по умолчанию):



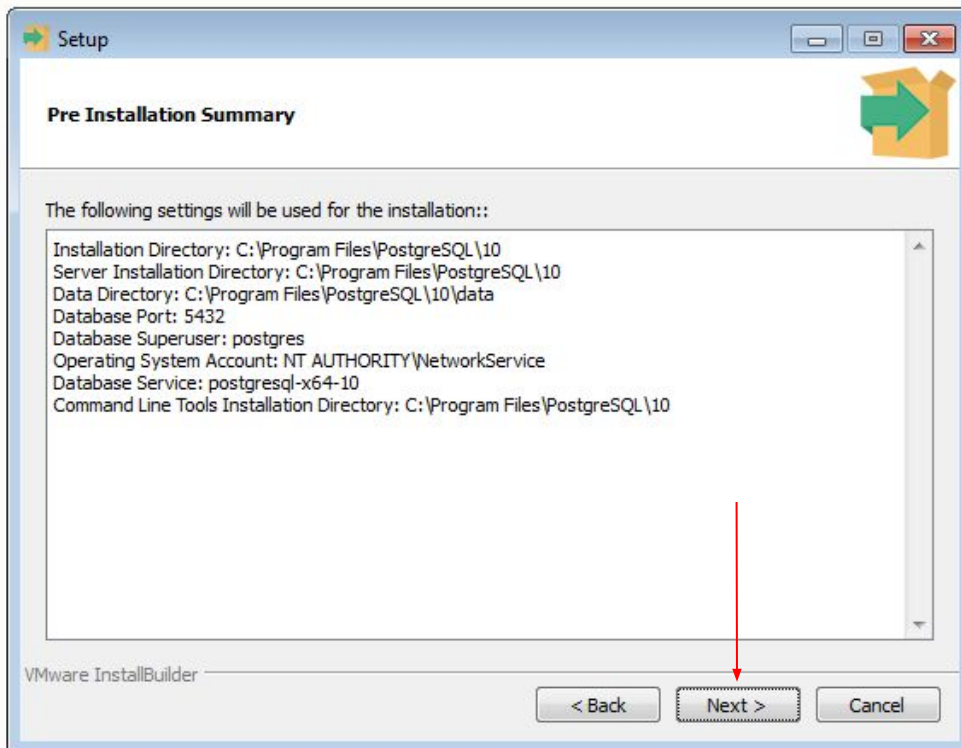
PostgreSQL

Оставьте локаль по умолчанию:



PostgreSQL

Согласитесь с выбранными настройками:



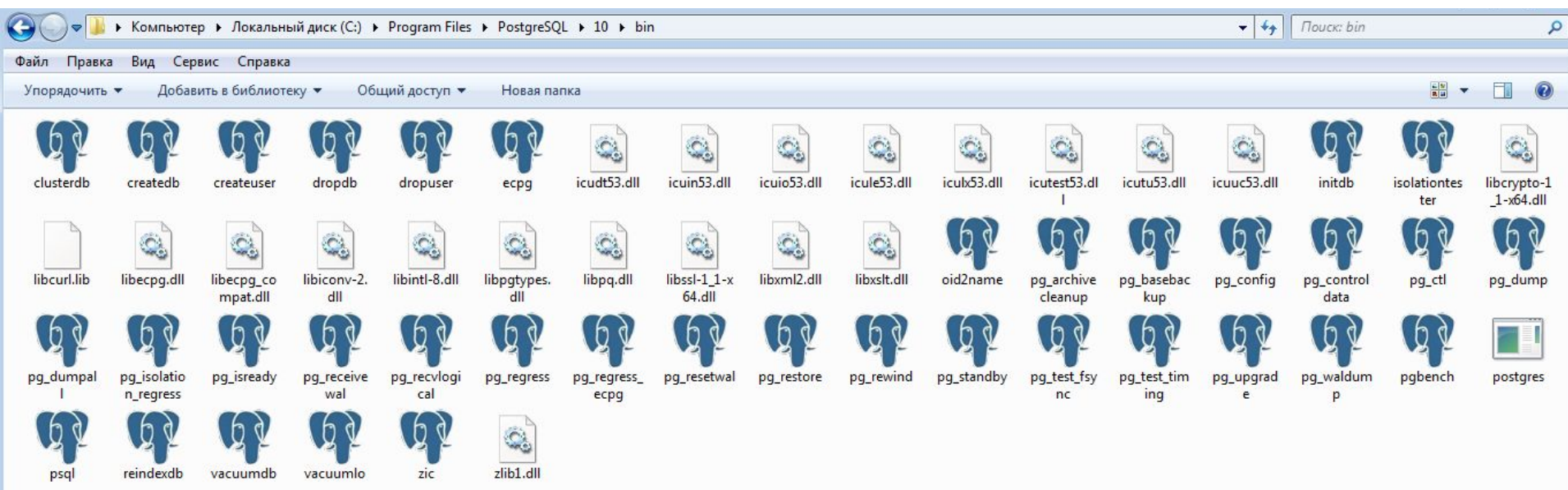
PostgreSQL

Дождитесь завершения установки и нажмите Finish:

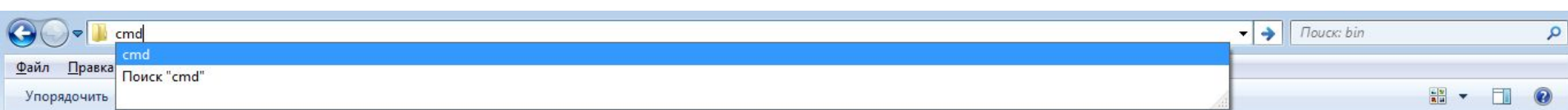


PostgreSQL

После установки перейдите в каталог **bin** (см. скриншот):

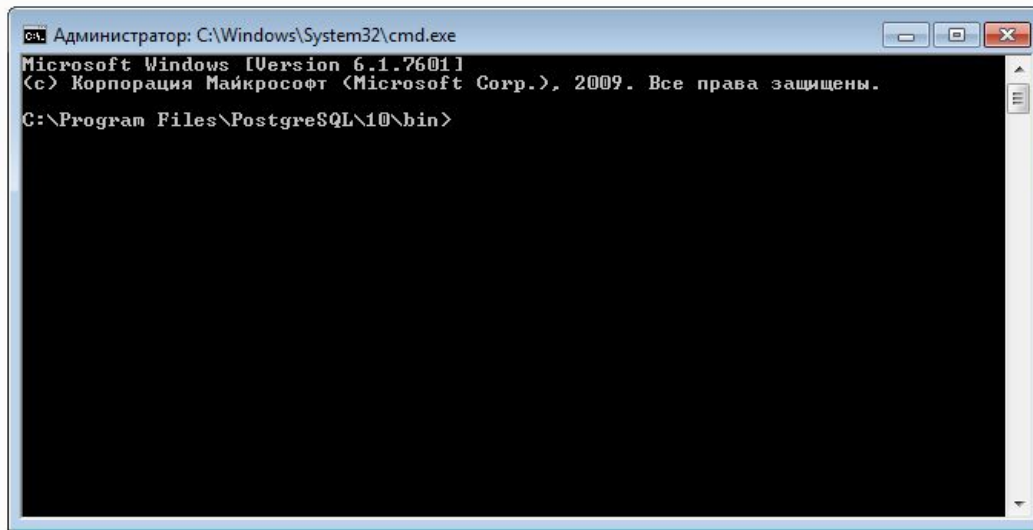


Введите в адресную строку **cmd** и нажмите **Enter**:



PostgreSQL

Откроется командная строка Windows именно в этом каталоге:



Нам необходимо:

1. Создать пользователя для подключения
2. Создать Базу Данных для работы
3. Передать пользователю права для работы с данной БД



PostgreSQL

Для настройки введите следующие команды:

```
psql -U postgres
```

*** вводите пароль пользователя postgres (не отображается) и жмёте **Enter** ***

```
create user app with login password 'pass';
```

```
create database db;
```

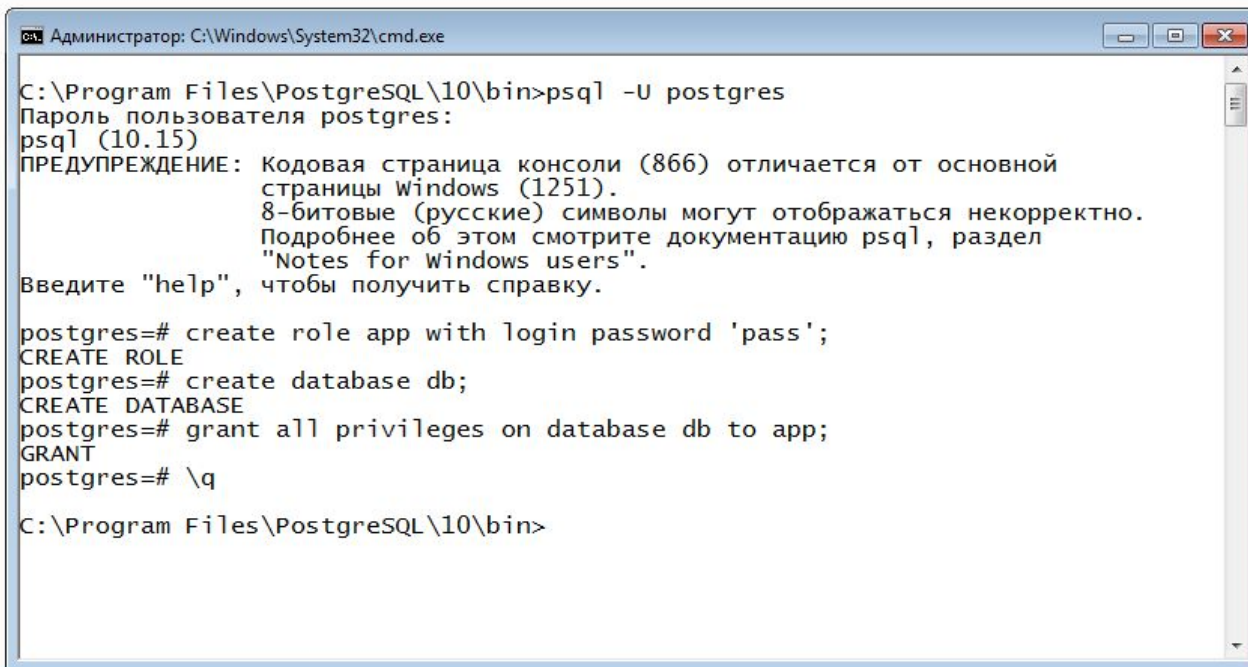
```
grant all privileges on database db to app;
```

```
\q
```

Каждая строка (кроме `\q`) заканчивается точкой с запятой и необходимо нажимать **Enter**.



PostgreSQL



Администратор: C:\Windows\System32\cmd.exe

```
C:\Program Files\PostgreSQL\10\bin>psql -U postgres
Пароль пользователя postgres:
psql (10.15)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
                  страницы Windows (1251).
                  8-битовые (русские) символы могут отображаться некорректно.
                  Подробнее об этом смотрите документацию psql, раздел
                  "Notes for Windows users".
Введите "help", чтобы получить справку.

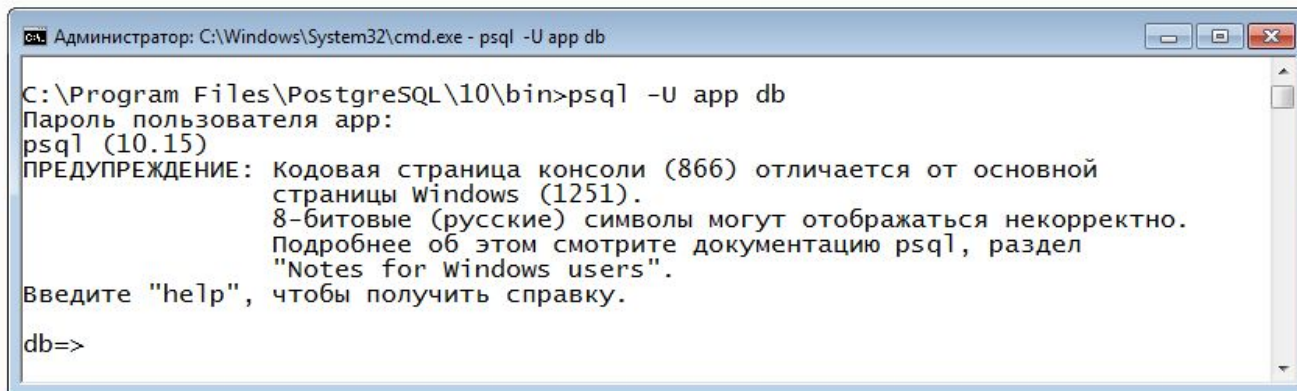
postgres=# create role app with login password 'pass';
CREATE ROLE
postgres=# create database db;
CREATE DATABASE
postgres=# grant all privileges on database db to app;
GRANT
postgres=# \q

C:\Program Files\PostgreSQL\10\bin>
```



PostgreSQL

Проверяете подключение с помощью команды:



```
Администратор: C:\Windows\System32\cmd.exe - psql -U app db

C:\Program Files\PostgreSQL\10\bin>psql -U app db
Пароль пользователя app:
psql (10.15)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
                  страницы windows (1251).
                  8-битовые (русские) символы могут отображаться некорректно.
                  Подробнее об этом смотрите документацию psql, раздел
                  "Notes for windows users".
Введите "help", чтобы получить справку.

db=>
```

Если видите **db=>**, значит вы всё сделали правильно.



DOCKER



Docker

[Docker](#) – самая популярная на сегодняшний день система контейнеризации.

Контейнер – это приложение, упакованное в специальный формат для удобства разработки, развёртывания, распространения и использования приложений.

В чём суть? Вот мы с вами хотим использовать PostgreSQL – для этого нам придётся её скачать, установить, разобраться с тем, как запускать и т.д. А если нам потребуется использовать другую систему, например, Apache Kafka? Нам придётся повторить этот процесс заново, причём для Kafka процесс установки, настройки и запуска будет совершенно другим.



Docker

А что если сделать так, что все эти приложения (PostgreSQL и другие) распространялись бы одинаково, запускать их можно было бы одинаково и настраивать? Представьте, что PostgreSQL можно скачать, установить и запустить одной командой?

Тогда бы мы не тратили время (особенно на этапе программирования) на то, чтобы разбираться с особенностями установки и запуска, а смогли бы быстрее приступить к разработке. В этом и заключается основная идея Docker'a – нам предоставляют готовые, упакованные в единый формат приложения:



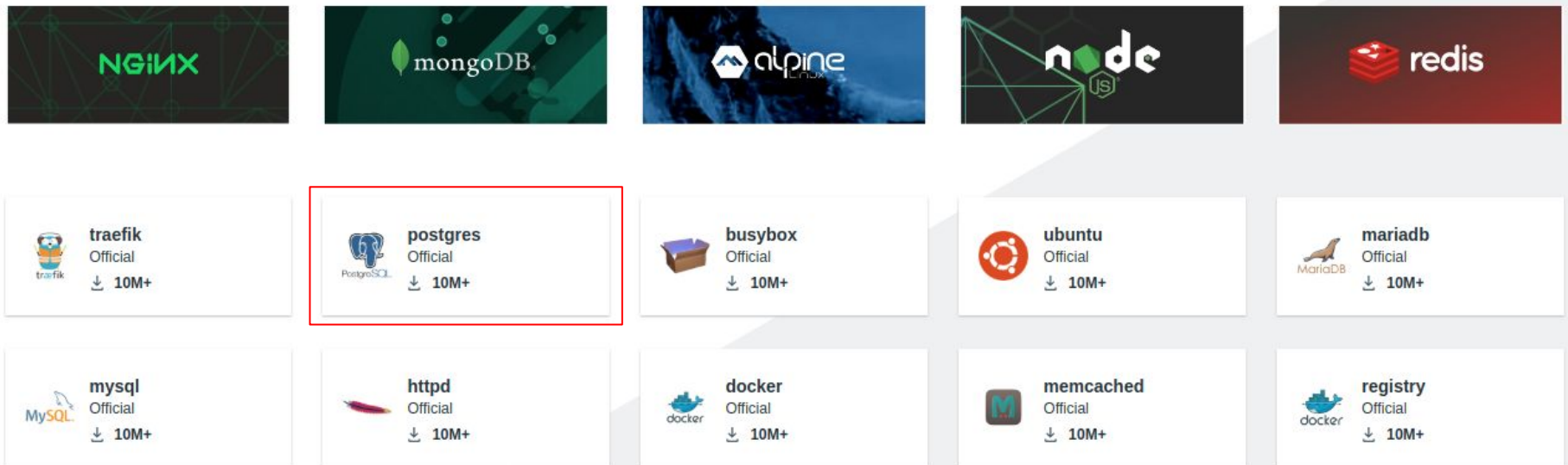
← упакованные в единый формат приложения

Docker



Docker Hub

Docker предоставляет [Docker Hub](#) – специальный веб-сервис, на котором и размещены эти упакованные приложения (далее будем говорить образы)



[See all Official Images >](#)



Docker

Наша задача – установить сам Docker и научиться запускать приложения.

Важно: Docker будет "съедать" очень много трафика (порядка 500 Мб на данную лекцию) поэтому будьте готовы.



УСТАНОВКА



Установка

Перед установкой убедитесь, что на вашем компьютере:

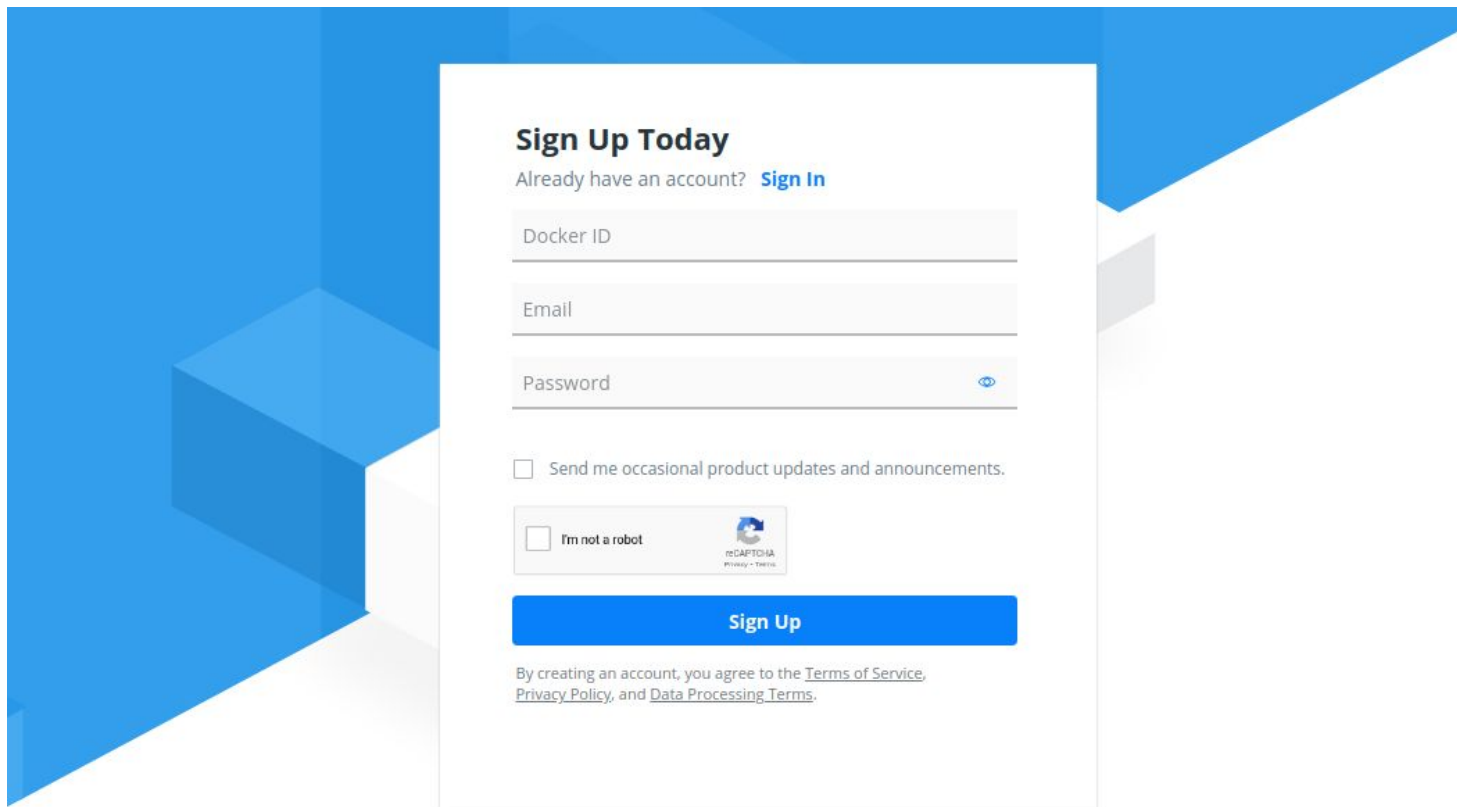
1. Есть хотя бы 4 гигабайта ОЗУ
2. Включена виртуализация в настройках BIOS (Intel VT-X или AMD-V) – для Mac OS и Windows
3. У вас хороший Интернет (Docker и образы очень много весят)

Если хотя бы одно из требований не выполняется, то установить Docker не получится, но вы можете воспользоваться [облачной версией](#) (для тренировки).



Docker ID

Для работы с Docker и сервисами Docker необходимо зарегистрироваться на Docker Hub (получить Docker ID). Перейдите на сайт <https://hub.docker.com> и зарегистрируйтесь:



The image shows a registration form on the Docker Hub website. The form is titled 'Sign Up Today' and includes a link for users who already have an account. It contains input fields for 'Docker ID', 'Email', and 'Password'. There is a checkbox for receiving product updates and a reCAPTCHA verification step. A prominent blue 'Sign Up' button is at the bottom of the form. Below the button, there is a disclaimer about agreeing to the Terms of Service, Privacy Policy, and Data Processing Terms.


Sign Up Today
Already have an account? [Sign In](#)

Docker ID

Email

Password

☐ Send me occasional product updates and announcements.

☐ I'm not a robot 

Sign Up

By creating an account, you agree to the [Terms of Service](#), [Privacy Policy](#), and [Data Processing Terms](#).



Установщики

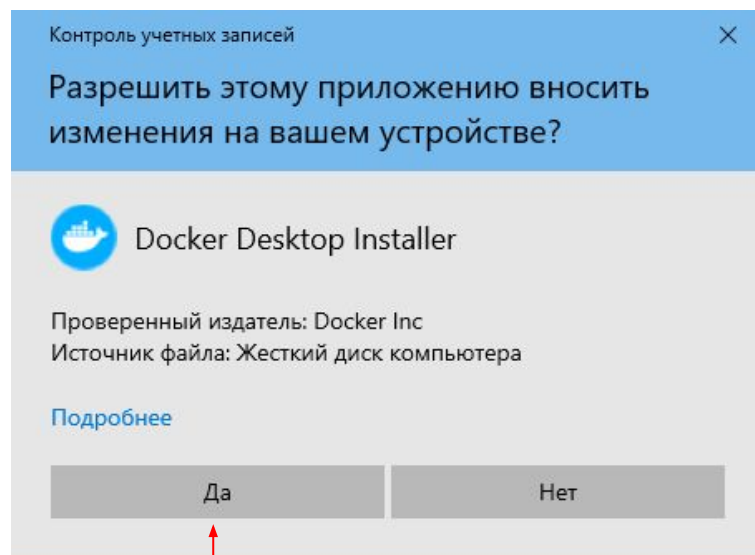
В зависимости от того, какая у вас версия операционной системы, вам нужны разные установщики:

- Windows 11, Mac OS: <https://www.docker.com/products/docker-desktop>
- Windows 7 – не поддерживается, используйте установку PostgreSQL в виде обычного приложения
- Linux: <https://docs.docker.com/engine/install/ubuntu/> (и другие)



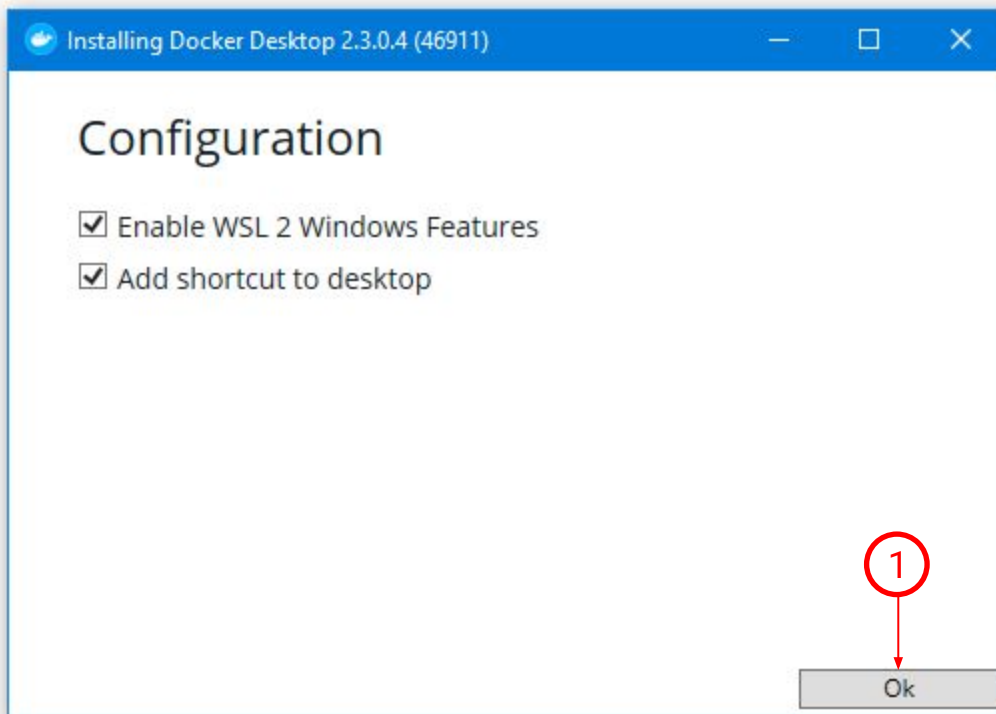
Windows 11

Запустите скачанный файл:



Windows 11

Оставьте выбранными опции по умолчанию:



Windows 11

Дождитесь завершения установки, после чего запустите Docker:



Windows 11

Запуск не будет быстрым, вам нужно дождаться, пока в системном трее перестанет анимироваться иконка Docker:



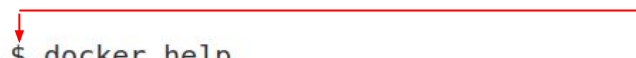
ИСПОЛЬЗОВАНИЕ



Help

Работа с Docker, как и с Git, подразумевает введение команд в консоли. Поэтому создайте пустой проект в VS Code, откройте консоль и начнём.

Первая команда, с которой нужно ознакомиться: **help** – показывает вам справку по командам Docker:

 \$ docker help \$ вводить не нужно

```
Usage:  docker [OPTIONS] COMMAND
```

```
A self-sufficient runtime for containers
```

```
Management Commands:
```

```
builder      Manage builds
config       Manage Docker configs
container     Manage containers
```

```
Run 'docker COMMAND --help' for more information on a command.
```



System

Команда **system** позволяет получить общую информацию о системе:

```
$ docker system --help
```

```
Usage:  docker system COMMAND
```

Manage Docker

Commands:

df	Show docker disk usage
events	Get real time events from the server
info	Display system-wide information
prune	Remove unused data

```
$ docker system info
```

Client:

```
  Debug Mode: false
```

Server:

```
Containers: 144
```

```
  Running: 1
```

```
  Paused: 0
```

```
  Stopped: 143
```

```
Images: 1078
```

```
Server Version: 19.03.6
```



Образы и контейнеры

Чтобы работать дальше – нужно понять следующие два термина:

- Образ (image или container image) – это упакованное приложение (все файлы + команда, с помощью которой можно запустить)
- Контейнер (container) – запущенное из образа приложение

Т.е. представьте, что вы берёте, запаковываете в один архив ваше приложение, все необходимые файлы (например, изображения, HTML и т.д.) и команду, с помощью которой это всё можно запустить (`./app`) – это и будет образ.

Контейнер – это когда вы берёте этот архив и "запускаете" (представьте, что при двойном клике на архив запускается ваше приложение из архива). Т.е. может быть один образ – но много контейнеров, запущенных из него.



Образы и контейнеры

Ещё более яркая аналогия – exe-файлы. На вашем компьютере всего один файл `notepad.exe` (блокнот). Но вы можете этих блокнотов запустить хоть 100 штук.

`notepad.exe` – образ, а контейнеры – запущенные экземпляры.



Общий процесс работы

Общий процесс работы выглядит следующим образом:

1. Вы скачиваете образ (по умолчанию они скачиваются из Docker Hub)
2. Создаёте из образа контейнер
3. Запускаете контейнер
4. Работаете с контейнером
5. Останавливаете контейнер
6. Удаляете контейнер



Скачивание образа

Вы просто пишете [имя образа](#) с Docker Hub:



↓ 1B+

[hello-world](#) – это простейшее приложение, которое при запуске печатает Hello

```
$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:7f0a9f93b4aa3022c3a4c147a449bf11e0941a1fd0bf4a8e6c9408b2600777c5
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
```



Скачивание образа

Увидеть список скачанных образов можно с помощью команды `docker image ls`:

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	bf756fb1ae65	8 months ago	13.3kB

Т.е. Docker скачивает образ на ваш компьютер и использует его в дальнейшем (не скачивает заново, пока образ есть на вашем компьютере).

Чтобы удалить образ, введите команду `docker image rm <имя образа>`, например, `docker image rm hello-world` (пока выполнять не нужно).



Создание контейнера

Создаётся контейнер с помощью команды `docker container create`:

```
$ docker container create hello-world  
bd5227683f09287ef31a61c58c1cd7de57768bbc2d6ca5dcdfc8070615123ef5
```

То, что напечаталось – это `id` контейнера (у вас будет другой). Он нужен для дальнейшего взаимодействия с этим контейнером.

Целиком весь `id` писать не нужно, достаточно первых 5-6 символов.

Обратите внимание: Docker достаточно умный и если вы ещё не скачали образ, а сразу ввели `docker container create` – он заодно за вас его скачает.



Запуск контейнера

Запускается контейнер с помощью команды `docker container start`:

```
$ docker container start -a bd5227
```

```
Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.
```

```
To generate this message, Docker took the following steps:
```

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

```
To try something more ambitious, you can run an Ubuntu container with:
```

```
$ docker run -it ubuntu bash
```

```
Share images, automate workflows, and more with a free Docker ID:
```

```
https://hub.docker.com/
```

```
For more examples and ideas, visit:
```

```
https://docs.docker.com/get-started/
```



Всё вместе

На самом деле, так редко кто делает, потому что в большинстве случаев вам просто нужно скачать образ (если его нет) и запустить **новый** контейнер.

Для этого есть команда `docker container run`, которая сделает всё сама:

```
$ docker container run hello-world
```

```
Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.
```

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

```
https://hub.docker.com/
```

For more examples and ideas, visit:

```
https://docs.docker.com/get-started/
```



run vs container run

Вы можете встретить написание `docker run` вместо `docker container run`. Дело в том, что раньше команда действительно была `docker run`, но со временем авторы Docker решили навести порядок в командах и теперь команды по управлению контейнерами пишутся как `docker container`, а образами – `docker image`.

Но старые команды по прежнему работают и для краткости вы можете писать `docker run` вместо `docker container run`.



Остановка контейнера

По умолчанию, когда программа, запущенная внутри контейнера заканчивает свою работу, сам контейнер останавливается (т.е. вручную его останавливать не надо, хотя вы можете выполнить `docker container stop <id контейнера>`, чтобы принудительно остановить его).

Посмотреть какие у вас сейчас есть контейнеры и в каком состоянии можно с помощью команды `docker container ls -a` (флаг `-a` означает все, т.к. по умолчанию выводятся только работающие в данный момент):

```
$ docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e4359eb27985	hello-world	"/hello"	4 minutes ago	Exited (0) 4 minutes ago		kind_lamarr
bd5227683f09	hello-world	"/hello"	12 minutes ago	Exited (0) 9 minutes ago		festive_elgama1



Имена контейнеров

Вместо идентификаторов контейнеров можно использовать имена:

```
$ docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
e4359eb27985	hello-world	"/hello"	4 minutes ago	Exited (0) 4 minutes ago	
bd5227683f09	hello-world	"/hello"	12 minutes ago	Exited (0) 9 minutes ago	

NAMES
kind_lamarr
festive_elgama1

Они генерируются случайным образом, но можно задать собственные:

```
$ docker container run --name hello-go hello-world
```

```
Hello from Docker!  
This message shows that your installation appears to be working correctly.
```

```
$ docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
ffa2692d071b	hello-world	"/hello"	37 seconds ago	Exited (0) 34 seconds ago	

NAMES
hello-go



Удаление контейнера

Удалить остановленный контейнер можно с помощью команды `docker container rm <id>`.

Но обычно удаляют все остановленные (ненужные контейнеры) с помощью команды `docker system prune`.



Важно

К контейнерам следует относиться как к одноразовой посуде – т.е. задача контейнера выполнить свою работу и потом его можно выбросить (`docker system prune` удалит его).



Docker

Мы рассмотрели лишь минимальный перечень команд, которые вам будут нужны в повседневном использовании.

Теперь же рассмотрим решение нашей задачи – установку PostgreSQL.



POSTGRESQL DOCKER IMAGE



Docker Hub

Первое, что нужно сделать – это перейти на Docker Hub и поискать там PostgreSQL:

1 - 25 of 14,243 results for **postgres**. [Clear search](#)

Most Popular



postgres

Updated 12 hours ago

The PostgreSQL object-relational database system provides reliability and data integrity.

Container

Linux

PowerPC 64 LE

mips64le

IBM Z

ARM 64

x86-64

386

ARM

Databases

OFFICIAL IMAGE



10M+ 8.6K

Downloads Stars

OFFICIAL IMAGE



Выбирать нужно только образы с пометкой



Docker Hub

Заходим внутрь и читаем документацию. Там будет написано достаточно много, но мы приведём основные моменты:

1. У каждого образа есть теги (это "аналог" версий)
2. У некоторых образов есть переменные окружения – т.е. переменные, задав которые можно настроить запускаемый контейнер (например, задать логин и пароль пользователя БД и саму БД)
3. У большинства образов есть открываемые порты – т.е. мы можем попросить Docker "пробросить" порты из контейнера в операционную систему (например, PostgreSQL внутри контейнера работает на порту 5432, мы можем сделать так, чтобы все запросы в ОС на порт 5432 отправлялись на порт 5432 внутри контейнера)



latest

По умолчанию, когда мы пишем `docker image pull` или `docker container run`, мы не указываем версию образа. Она автоматически выставляется в `latest`, т.е. `docker image pull hello-world` равнозначно `docker image pull hello-world:latest`.

Но если мы хотим конкретную версию, как в случае PostgreSQL лучше указывать:

- `13.1` , `13` , `latest`
- `13.1-alpine` , `13-alpine` , `alpine`
- `12.5` , `12`
- `12.5-alpine` , `12-alpine`
- `11.10` , `11`
- `11.10-alpine` , `11-alpine`
- `10.15` , `10`
- `10.15-alpine` , `10-alpine`

Т.е. будет `docker container run postgres:10` (пока не запускайте эту команду).



Переменные окружения

Переменные окружения обычно перечислены в описании контейнера:

- `POSTGRES_USER` – имя пользователя (владельца) создаваемой БД (то, что мы делали через `create user + grant privileges`)
- `POSTGRES_PASSWORD` – пароль пользователя
- `POSTGRES_DB` – название создаваемой БД (то, что мы делали через `create database`)
- и т.д.

Они указываются с помощью флага `-e`:

`docker container run -e POSTGRES_USER=app -e POSTGRES_PASSWORD=pass -e POSTGRES_DB=db postgres:10` (не запускайте эту команду). Уже выглядит страшновато?



Порты

Проброс портов осуществляется с помощью флага `-p`:

```
docker container run -p 5432:5432 -e POSTGRES_USER=app -e  
POSTGRES_PASSWORD=pass -e POSTGRES_DB=db postgres:10
```

 (не запускайте эту команду).

Где первым указан порт в вашей операционной системе, а вторым – порт внутри контейнера.

Теперь точно страшно. А представьте, вам придётся не копировать эту команду из презентации, а вводить руками. Так точно не пойдёт.



DOCKER COMPOSE



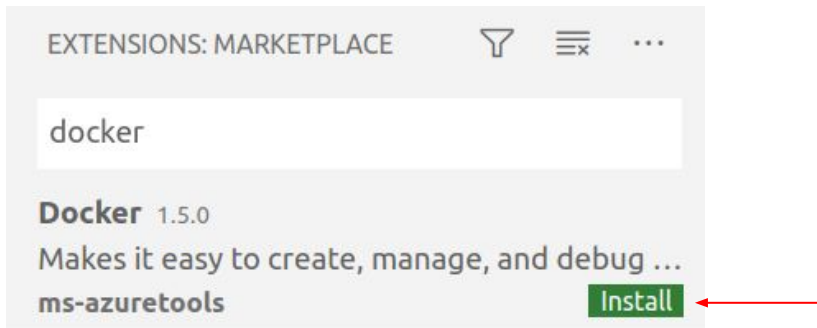
Docker Compose

Чтобы решить эту проблему (а также ряд других) придумали специальный инструмент Docker Compose. Мы можем просто записать всю конфигурацию в файл и потом запускать из файла одной командой.



VS Code

Чтобы удобно работать с файлами Docker Compose установите расширение:



Как и в случае с SQL Tools у вас появится боковая панелька Docker, где можно визуально работать с Docker (запускать из образов контейнеры и т.д.).



Docker Compose

После чего создайте файл `docker-compose.yml` (называться должен именно так):

 `docker-compose.yml`

```
1  services:
2    db:
3      image: postgres:10
4      ports:
5        - "5432:5432"
6      environment:
7        - POSTGRES_USER=app
8        - POSTGRES_PASSWORD=pass
9        - POSTGRES_DB=db
10
```

Внимательно следите за всеми отступами и тем, что пишете.



Docker Compose

Теперь откройте терминал и выполните команду `docker compose up` (Docker Compose прочитает ваш файл создаст новый контейнер).

Дождитесь следующих строк:

```
PostgreSQL init process complete; ready for start up.
```

```
2020-11-18 12:48:28.052 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2020-11-18 12:48:28.052 UTC [1] LOG:  listening on IPv6 address ":::", port 5432
2020-11-18 12:48:28.131 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2020-11-18 12:48:28.384 UTC [64] LOG:  database system was shut down at 2020-11-18 12:48:26 UTC
2020-11-18 12:48:28.478 UTC [1] LOG:  database system is ready to accept connections
```



Docker Compose

После этого можете подключаться (не закрывайте вкладку, где запустили Docker Compose) с помощью SqlTools (читайте дальше).



Docker Compose

Для того, чтобы остановить контейнер (а с ним и сервер PostgreSQL) нужно нажать в консоли, где вы запустили Docker Compose **Ctrl + C** и подождать. Контейнер остановится и при следующем запуске (**docker compose up**) снова запустится. Поскольку при такой процедуре контейнер не удаляется, все ваши данные, которые вы сохранили в PostgreSQL – сохранятся.

Если же вы выполните команду **docker compose down**, то удалите остановленный контейнер, а значит, при следующем запуске **docker compose up** контейнер будет создан заново (и никаких данных в нём не будет).

Мы решим эту проблему тем, что саму структуру БД будем создавать из кода (т.е. ваше приложение при запуске само будет делать **CREATE TABLE**).



Docker Compose

Обратите внимание: вы можете просто скопировать файл `docker-compose.yml` в другой проект и снова там запустить `docker compose up` (но это будет уже другой контейнер с другими данными)



PSQL



psql

Для работы с БД не из кода используется специальный инструмент – psql ([документация на русском](#)).

Если вы устанавливали PostgreSQL как обычное приложение, то подключаетесь командой:

```
psql -U app db
```

Если Docker Compose:

```
docker compose exec db psql -U app -d db
```

Все дальнейшие команды мы будем выполнять с помощью этого инструмента.



SQL



SQL

SQL (раньше переводилось как структурированный язык запросов) – модель данных (т.е. вот эти самые таблички и связи между ними) и язык, который позволяет работать с БД. Т.е. под одним термином понимают две вещи, которые также следует определять в зависимости от контекста. Мы, в первую очередь, будем говорить о языке запросов.

Вы можете встретить термин "реляционные БД", "реляционная модель" – мы будем считать это синонимом SQL-модели данных (хотя там и есть некоторые тонкие различия).



SQL

Сам язык SQL состоит из нескольких подязыков:

- Data Definition Language (DDL) – определяет структуру хранимых данных
- Data Manipulation Language (DML) – определяет операции добавления, изменения и удаления данных
- Data Retrieval Language (DRL) – определяет операции извлечения (выбора) данных
- Access Control – управление доступом (то, что мы с вами выдавали доступ на базу **db** пользователю **app**)



Диалекты SQL

Нужно отметить, что каждая СУБД в целом использует похожий синтаксис (правила написания и ключевые слова) SQL, но есть специфические отличия.

Например, в одной СУБД ключевое слово (что оно значит мы разберём чуть позже) – `AUTOINCREMENT`, в другой – `AUTO_INCREMENT`, а в третьей – его вообще нет, вместо этого используется отдельный тип `SERIAL`.

Это называется диалектом – как в реальном языке, когда в одной области страны разговаривают немного по-другому, чем в другой.



SQL

В SQL базах данных (в отличие от JS) мы не можем просто взять и начать записывать данные:

1. Мы должны сначала описать, какие таблицы у нас будут, какие в них будут столбцы и какого типа (тип фиксирован)
2. Затем уже можно в эти таблицы добавлять строки, удалять, обновлять и осуществлять выборку (CRUD)

Ключевая идея: зафиксировать набор столбцов (они определяют структуру) и произвольно менять набор строк (они определяют сами записи). Это как с классами – мы использовали классы для того, чтобы у всех объектов был одинаковый набор полей.



DDL

В принципе, мы уже проделывали операцию моделирования данных: мы просто должны определить, какие у объектов есть свойства и как это соответствует тем типам данных, которые есть в конкретном диалекте SQL.

Начнём с простого: создадим табличку для хранения наших платежей (пока без всяких пользователей).

Для этого нам нужно понять:

1. Как создавать таблицы
2. Какие типы данных бывают



Типы данных

Типов данных достаточно много. Но нас будут интересовать следующие:

- целые числа (`INT`, `BIGINT`)
- автоматически увеличивающиеся целые числа (`SERIAL`, `BIGSERIAL`)
- текстовые поля (`TEXT`)
- логические (`BOOL`)
- timestamp (`TIMESTAMP`)
- JSON (`JSON` и `JSONB`)



SQL & Shell Commands

`psql` – командная оболочка, которую мы запускаем для выполнения команд PostgreSQL позволяет вводить как SQL-запросы (вроде `create database`, `create user`), так и собственные команды (вроде `\q`, `\dt`).

Все SQL-запросы должны завершаться точкой с запятой, в то время как остальные – не обязаны.

Чтобы отличать одни от других, мы будем SQL-запросы писать в ВЕРХНЕМ РЕГИСТРЕ (хотя это не обязательно), а команды оболочки – в нижнем.



Shell Commands

Справку по командам
можно получить с

помощью команды

`help`:

```
$ docker-compose exec bankdb psql -U app -d db
psql (10.15 (Debian 10.15-1.pgdg90+1))
Type "help" for help.

db=# help
You are using psql, the command-line interface to PostgreSQL.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit
```

Соответственно, для справки по SQL-командам:

`\h <команда>`, например: `\h CREATE USER`

Для справки по встроенным командам:

`\?`



DDL

В рамках DDL определяется три ключевых действия:

- **CREATE** – создать
- **ALTER** – изменить
- **DROP** – удалить



DDL

Общий синтаксис достаточно сложный, поэтому мы начнём с самой простой формы:

```
CREATE TABLE <имя таблицы> (  
    <имя столбца> <тип столбца> <ограничения>,  
    <имя столбца> <тип столбца> <ограничения>,  
    <имя столбца> <тип столбца> <ограничения>,  
);
```



DDL

Есть разные схемы именования таблиц. Мы будем придерживаться следующей: название таблицы пишется во множественном числе с маленькой буквы и соответствует названию тех записей, что мы собираемся хранить.

Т.е. собираемся хранить пользователей – `users`. Собираемся хранить платежи – `payments`.

Пробуем – вводим `CREATE TABLE payments (` и нажимаем на `Enter`:

```
db=# CREATE TABLE payments (  
db(##
```

Как вы видите – символ приглашения изменился на `->` – это значит, что вы можете продолжать вводить следующую строку (пока не завершите всё `;`).



DDL

```
db=# CREATE TABLE payments (  
db(# id BIGSERIAL,  
db(# amount INT,  
db(# created TIMESTAMPTZ  
db(# );  
CREATE TABLE
```

Посмотрим, что таблица была создана:

```
db=# \dt  
          List of relations  
Schema |   Name   | Type  | Owner  
-----+-----+-----+-----  
public | payments | table | app  
(1 row)
```

Дальше мы не будем ходить по справке, а просто будем показывать вам самые распространённые команды.



DDL

Например, для просмотра описания таблицы:

```
db=# \d payments;
```

Table "public.payments"				
Column	Type	Collation	Nullable	Default
id	bigint		not null	nextval('payments_id_seq'::regclass)
amount	integer			
created	timestamp with time zone			

Чуть позже мы рассмотрим Nullable и Default, сейчас же обсудим типичные ошибки.



DDL

Нельзя одну таблицу создать два раза – т.е. если вы ошиблись и попытаетесь ввести команду создания таблицы заново, то получите:

```
db=# create table payments(  
db(# id bigserial  
db(# );
```

```
ERROR:  relation "payments" already exists
```

← мы специально не стали вводить целиком



SQL TOOLS



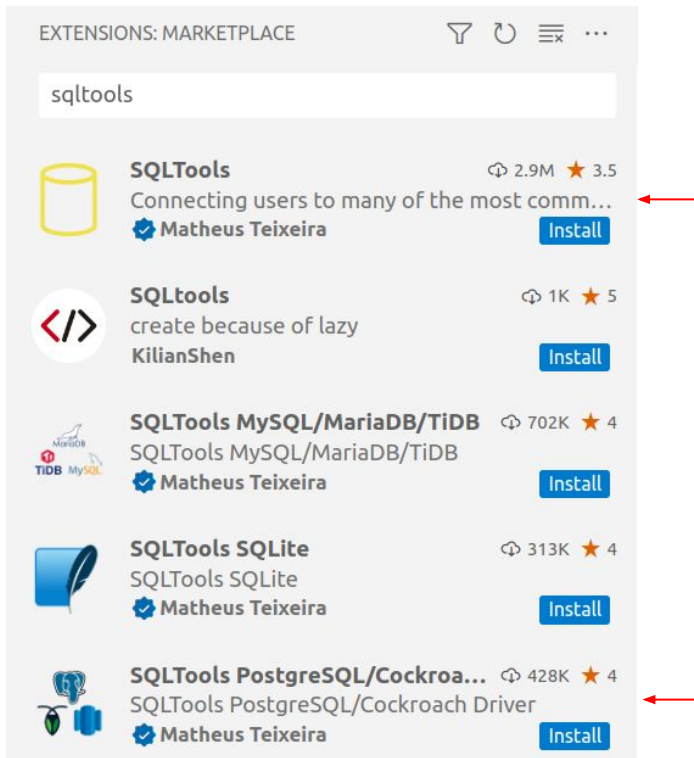
VS Code

Как вы уже заметили, работать в таком режиме не особо удобно, поэтому мы установим несколько расширений для VS Code, которые позволят нам работать удобнее.



VS Code

Откройте панельку расширений и в поиске введите SQLTools:

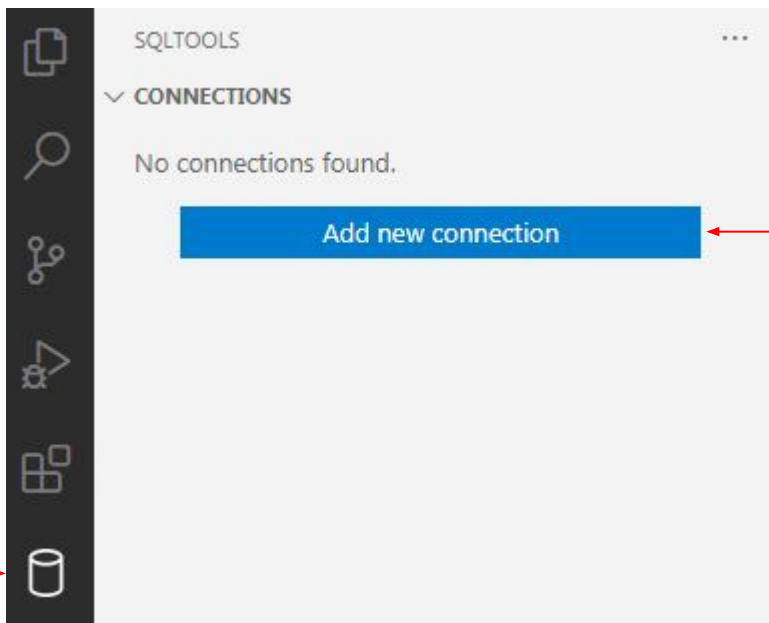


Нам нужно установить 1 и 5. Теперь, не закрывая VS Code, в котором у вас запущен Docker Compose, создайте в каталоге `C:\projects` каталог `sql` и откройте его в VS Code.



VS Code

В боковой панельке у вас должна появиться иконка базы данных, необходимо нажать на неё и затем нажать на кнопку Add new connection:



VS Code

Выберите PostgreSQL:



VS Code

Заполните поля, как на скриншоте ниже:

Connection name*	db
Connection group	
Connect using*	Server and Port
Server Address*	localhost
Port*	5432
Database*	db
Username*	app
Use password	Save as plaintext in settings
Password*

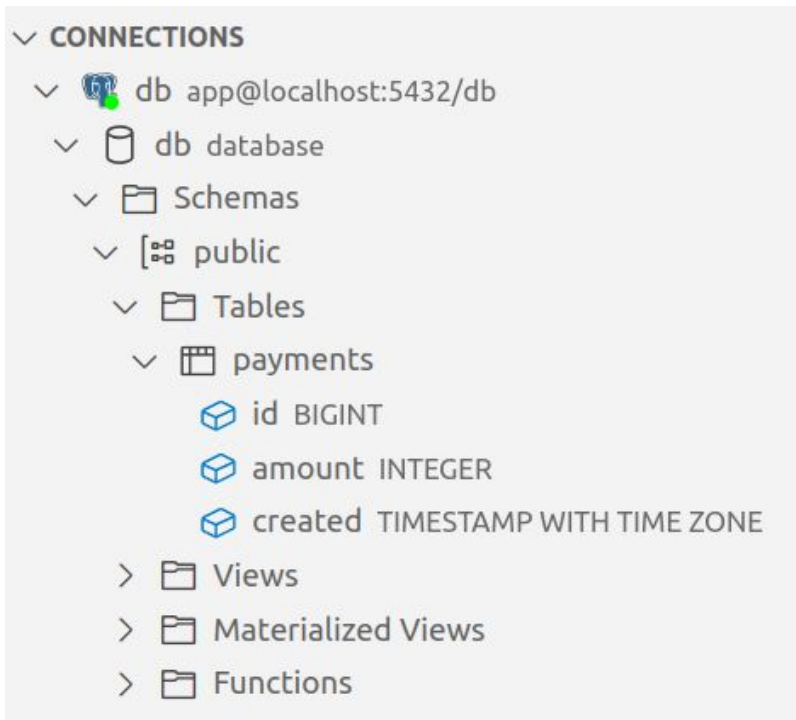


VS Code

После чего нажмите Save Connection:

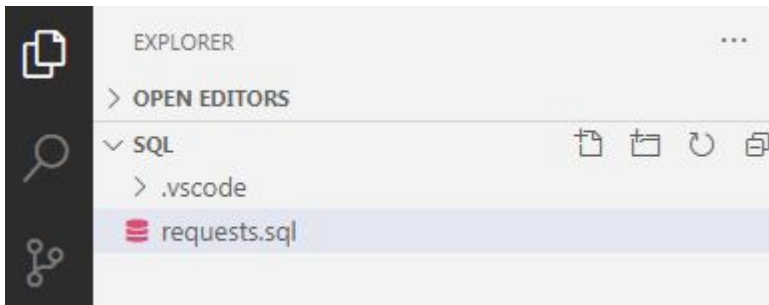
SAVE CONNECTION

И нажмите Connect Now (в боковой панелике появится подключение):



VS Code

После чего создайте файл `requests.sql` (в нём мы и будем писать запросы):



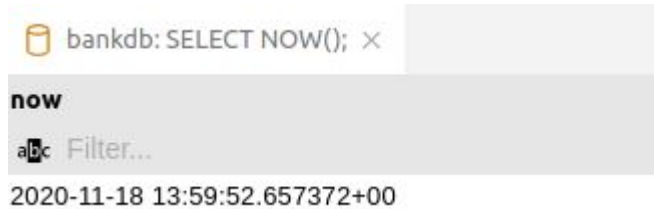
VS Code

Наберите `SELECT NOW();` и нажмите Run on active connection:



VS Code

Результат запроса будет отображён в боковой панели (справа):



SqlTools

Настройки подключения (файл внутри .vscode):

.vscode > {} settings.json > ...

```
1  {  
2    "sqltools.connections": [  
3      {  
4        "previewLimit": 50,  
5        "server": "localhost",  
6        "port": 5432,  
7        "driver": "PostgreSQL",  
8        "name": "db",  
9        "database": "db",  
10       "username": "app",  
11       "password": "pass"  
12     }  
13   ]  
14 }  
15
```



SqlTools

Итак, теперь, когда у нас есть удобный инструмент для выполнения SQL запросов, давайте вернёмся к изучению SQL.



DDL



DDL

Итак, мы остановились на том, что нельзя дважды создать таблицу с одним именем. У нас есть варианты **ALTER** (изменить) или **DROP** (удалить). **ALTER** мы рассматривать не будем, а вот с **DROP** поэкспериментируем:


```
DROP TABLE payments;
```

Этот запрос удалит всю таблицу **payments** (вместе с данными, если они там были).

Важно: будьте очень осторожны с этим запросом! Это очень частый случай, когда программист случайно удалял таблицу с данными. Восстановить их потом будет нельзя.



DDL



The screenshot shows a SQL editor window titled 'requests.sql'. A context menu is open over the first line of code, which is '1 DROP TABLE payments;'. The menu options are 'Run on active connection' and 'Select block'. The text 'DROP TABLE payments;' is highlighted in blue.

```
requests.sql  
1 DROP TABLE payments;
```

Чтобы выполнить запрос, достаточно выделить строку с запросом и нажать **Ctrl + E** два раза (или правой кнопкой мыши Run query).



DDL

Иногда в боковой панели вы будете видеть ошибку:



Query with errors. Please, check the error below.

[CONSOLE](#)[RE-RUN QUERY](#)[EXPORT](#)[OPEN](#)

А снизу будет пояснение проблемы:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE: MESSAGES

Cannot destructure property 'name' of 'undefined' as it is undefined. 09:12:15

В большинстве случаев это значит, что проблема в вашем запросе. Присылайте в чат запрос и сообщение – мы вам поможем.



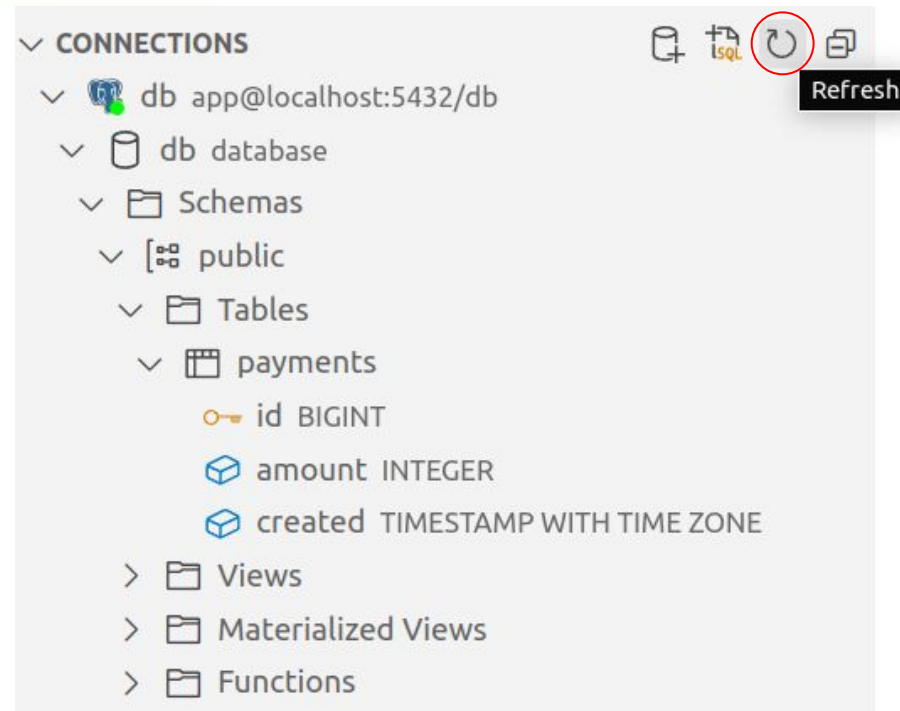
CREATE

Теперь создадим таблицу "правильно" и по кусочкам разберём, что значат те или иные вещи:

▶ Run on active connection | ≡ Select block

```
1 CREATE TABLE payments (  
2   id BIGSERIAL PRIMARY KEY,  
3   amount INT NOT NULL CHECK (amount > 0),  
4   created TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP  
5 );
```

Чтобы обновить данные в боковой панели,
нажмите на кнопку обновить:



PRIMARY KEY

PRIMARY KEY (первичный ключ) – это значит, что для всех строк в этой таблице (которые мы будем вставлять позже), значение этого столбца должно быть уникально (т.е. id не может повторяться) и не может быть **NULL**.

Вполне логично для **id**.



BIGSERIAL

Тип **BIGSERIAL** означает автоматически увеличивающийся **BIGINT**, что мы хотим, чтобы база данных сама делала +1, при вставке новых записей. Когда мы хранили всё в памяти, мы сами это теперь за нас это будет делать база данных.

В данном случае мы сделали это лишь для примера, поскольку использовать целые числа в качестве идентификаторов транзакций – не лучшая идея (достаточно быстро исчерпаются + неудобно распределять по различным базам данных).



NOT NULL

NOT NULL означает, что в это поле нельзя писать **NULL**. Для **PRIMARY KEY** (далее – PK) это ограничение в PostgreSQL действует автоматически, поэтому писать его вручную не нужно.



DEFAULT

DEFAULT означает, что если при вставке записи мы не укажем значение для данного поля будет использовано то, которое задано здесь по умолчанию.

В частности, **CURRENT_TIMESTAMP** означает, что будет взято текущее время.



UNIQUE & CHECK

Ещё существуют два ограничения:

- **UNIQUE** – значение столбца может быть уникально (при этом туда можно записать **NULL**, если не написано **NOT NULL**)
- **CHECK** – позволяет перед записью проверить определённые условия

Т.е. фактически, база данных делает примерно всё то же самое, что мы делали с вами, когда делали базовую реализацию сервисов.



NULL

NULL – это особое значение в базах данных. **NULL** не равен ничему, включая самого себя (фактически, это маркер, указывающий на отсутствие значения у поля).

И поскольку **NULL** не равен самому себе, то в столбцы с ограничением **UNIQUE** спокойно можно записывать **NULL**.



DML & DRL



DML & DRL

Создавать таблицы мы научились, самое время научиться в них подставлять данные (добавлять, обновлять и удалять) и извлекать их оттуда.

DML:

- **INSERT** (вставка данных)
- **UPDATE** (обновление данных)
- **DELETE** (удаление данных)

DRL (иногда – DQL):

- **SELECT** (выборка данных)



Упрощение

Запросы в SQL имеют множество вариаций, поэтому мы пойдём по стандартной схеме – научимся работать в первую очередь с теми, что имеют наиболее частое применение.



INSERT

Запрос INSERT выглядит следующим образом:

`INSERT INTO <имя таблицы>(<столбец1>, ..., <столбецN>) VALUES (<значение1>, ..., <значениеN>);`

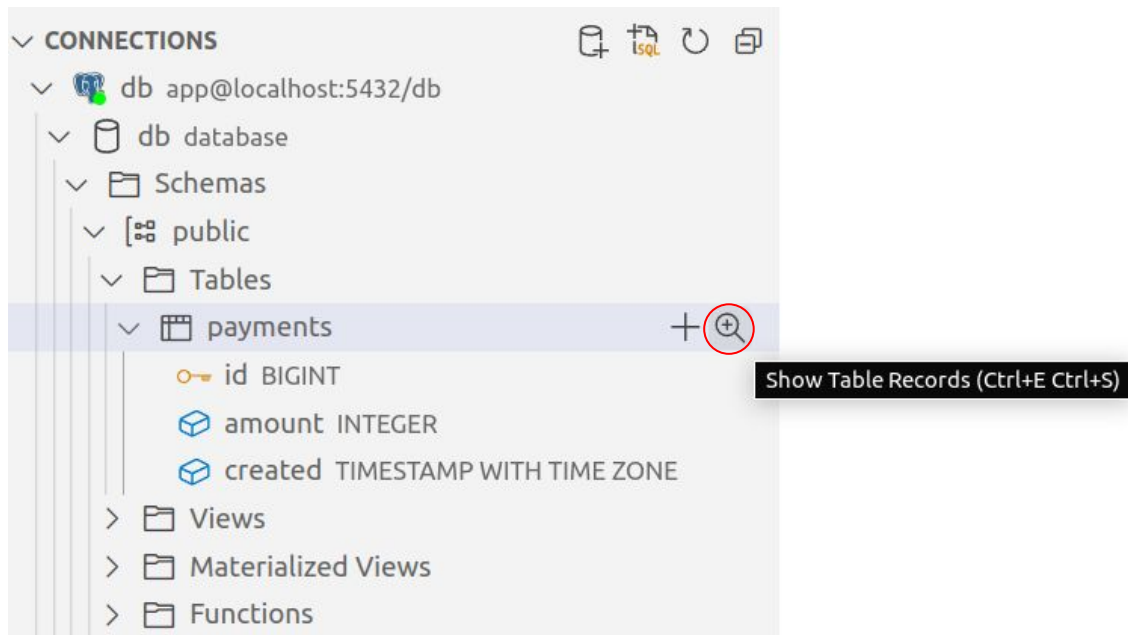
```
INSERT INTO payments (amount) VALUES (1000);
```

Для тех столбцов, что не указаны в списке, берутся значения по умолчанию (`DEFAULT`) либо `NULL`.



INSERT

Чтобы проверить, что действительно вставилось, нужно нажать на иконку лупы:



Или выполнить запрос **SELECT** (о нём чуть позже).



INSERT

Подставим ещё несколько записей:

```
INSERT INTO payments (amount) VALUES (2000);  
INSERT INTO payments (amount) VALUES (5000);
```

id	amount	created
abc Filter...	abc Filter...	abc Filter...
1	1000	2020-11-18 14:20:51.557326
2	2000	2020-11-18 14:31:27.29661
3	5000	2020-11-18 14:31:35.911605



UPDATE

UPDATE позволяет обновлять существующие записи. Выглядит вот так:

UPDATE <имя таблицы> **SET** <столбец1> = <значение1> **WHERE** <условие>;

Например:

```
UPDATE payments SET amount = 2000 WHERE id = 1;
```

Важный момент: и в этом запросе, и в запросе **DELETE** (который мы изучим следующим) часть с **WHERE** не обязательна. Но если вы её не укажете, тогда обновятся все записи (у всех **amount** будет **2000**), что не очень хорошо.

Если по условию в **WHERE** ничего найдено не будет, то никаких действий (включая генерирование ошибок) произведено не будет.



WHERE

Базовые операторы для **WHERE**:

- **=** – равно (обратите внимание, не **==**, а один символ)
- **!=** – не равно (также можно использовать **<>**)
- **>** – больше
- **<** – меньше
- **>=** – больше или равно
- **<=** – меньше или равно
- **IS NULL** – значение столбца **NULL**
- **IS NOT NULL** – значение столбца не **NULL**

с **NULL** используются только эти



DELETE

DELETE позволяет удалять существующие записи. Выглядит вот так:

```
DELETE FROM <имя таблицы> WHERE <условие>;
```

Например:

Старайтесь по-минимуму использовать **DELETE**. Вместо этого обычно добавляет **BOOL** поле **removed** (признак того, что запись удалена) и используют **UPDATE**.



SELECT

SELECT позволяет выбирать существующие записи. Выглядит вот так:

SELECT <столбец1>, ..., <столбецN> **FROM** <имя таблицы> **WHERE** <условие>;

Например:

```
SELECT id, amount FROM payments;
```

Опять же, блок **WHERE** не обязателен, если не указать его, то будут выбраны все строки.

Примечание: в некоторых руководствах вы можете увидеть **SELECT * FROM posts**, что означает выбрать все столбцы. Мы настоятельно не рекомендуем вам привыкать использовать данную конструкцию, поскольку она часто является одной из причин, по которым впоследствии ваш сервис может начать "тормозить".

Привыкайте указывать именно те столбцы, которые вам нужны.



SELECT

В **WHERE** можно объединять несколько условий с помощью **AND** (тогда должны выполняться оба условия), **OR** (тогда одно из двух) или **NOT** (отрицание):

```
WHERE amount >= 1000 AND amount <= 10000;
```



ИТОГИ



ИТОГИ

Сегодня мы познакомились с базовыми SQL запросами. Вы достаточно активно будете работать с SQL (и PostgreSQL в частности), поэтому уделите языку SQL максимум внимания.



ДОМАШНЕЕ ЗАДАНИЕ



banners

Что нужно сделать – это разработать схему базы данных (`CREATE TABLE`) и базовые запросы для всех методов (`getAll`, `getById`, `create`, `update`, `remove`).



banners

Как это сделать: в файле `schema.sql` опишите запрос на создание таблицы (`CREATE TABLE`), который создаёт таблицу `banners`, где у каждого баннера есть следующие столбцы:

- a. `id` (идентификатор) – целое число, первичный ключ, автоматически увеличивается на 1
- b. `title` (название баннера) – текст, не NULL
- c. `content` (содержимое баннера) – текст, не NULL
- d. `button` (содержимое кнопки) – текст, не NULL
- e. `link` (ссылка) – текст, не NULL
- f. `image` (путь до картинки) – текст, не NULL
- g. `created` (дата создания) – TIMESTAMPTZ (по умолчанию – текущая)

Этот файл нужно включать во все архивы с ДЗ.



Задания

Далее выполните следующие задания (каждое задание – отдельное ДЗ):

1. В файле `all.sql` опишите запрос `SELECT`, который выбирает столбцы `id`, `title`, `content`, `button`, `link`, `image`, `created`
2. В файле `byId.sql` опишите запрос `SELECT`, который выбирает поля `id`, `title`, `content`, `button`, `link`, `image`, `created` для баннера с `id = 2`
3. В файле `add.sql` опишите запрос на создание баннера со значениями:
 - a. `id` – автоматически
 - b. `title` – 'Осуществите свои мечты' (обратите внимание, в PostgreSQL строки пишутся в одинарных кавычках)
 - c. `content` – 'Вкладывайте депозит в Алифе и получайте больше половины от дохода'
 - d. `button` – 'Вкладывать'
 - e. `link` – '<https://alif.tj/deposit#CountYourIncome>'
 - f. `image` – 'granat.svg'
 - g. `created` (дата создания) – по умолчанию



Задания

4. В файле `edit.sql` опишите запрос `UPDATE`, который обновляет баннер с `id = 4` выставляя `title` равным 'Машина мечты ждёт вас'
5. В файле `remove.sql` опишите запрос `DELETE`, который удаляет баннер с `id = 5`

Все файлы должны располагаться в каталоге `sql` (т.е. вы упаковываете именно каталог `sql` и отправляете на проверку).



Спасибо за внимание

alif skills

2023г.

