

# código.py

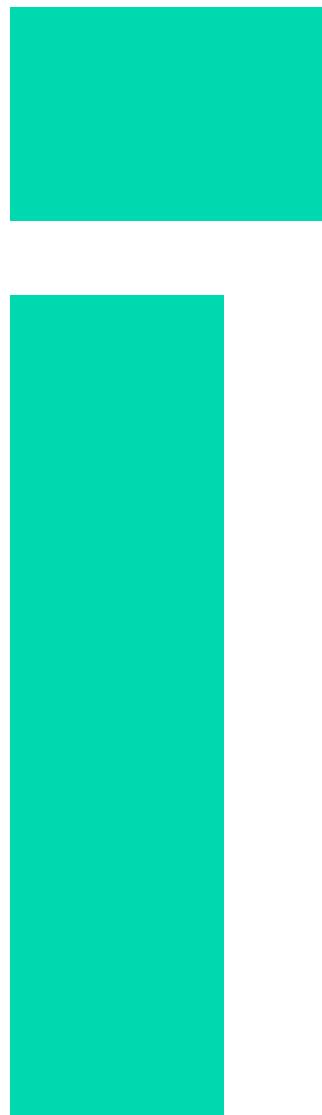
## GALÁXIA EXTRA

Banco de dados  
e SQL



# Introdução

Olá, seja bem-vindo à Galáxia Extra de banco de dados! Neste módulo abordaremos sobre a criação, edição e manipulação de banco de dados. Abordaremos também a linguagem SQL, utilizada na maioria dos bancos de dados para realizar operações. A parte boa é que se você chegou até aqui, sua lógica de programação está ficando cada vez mais afiada e aprender SQL será molezinha. Então vamos adiante!



# Mundo 1

## 1.1. O que é banco de dados?

Um banco de dados é um conjunto de informações relacionadas. Por exemplo, uma lista telefônica pode ser um banco de dados a partir do momento que nós buscamos um número de telefone através de uma classificação, que neste caso é o nome da pessoa, ou empresa.

Porém, existem alguns problemas na lista telefônica que um banco de dados SQL resolve. Um deles é que uma lista telefônica pode ter duas pessoas com o mesmo nome, coisa que em um banco de dados relacional isso não pode acontecer devido ao mecanismo de chaves.

Apesar de ter diversos formatos de banco de dados, o banco relacional é o mais usado entre todos. O conceito dele é simples: as informações entre tabelas podem se relacionar. No caso abaixo podemos ver um exemplo disso: (repare que mesmo que sejam tabelas diferentes, elas conseguem conversar).

**Cliente**

ID	nome	sobrenome
1	George	Blake
2	Sue	Smith

**Conta**

ID da Conta	Categoria_produto	ID_cliente	Saldo
103	CHM	1	R\$75
104	SAV	1	R\$250
105	CHM	2	R\$78
106	MM	2	R\$50

**Produto**

Categoria_produto	Nome
CHM	Emissão de cheques
SAV	Poupança
MM	Aplicações
LOC	Linha de crédito

**Transação**

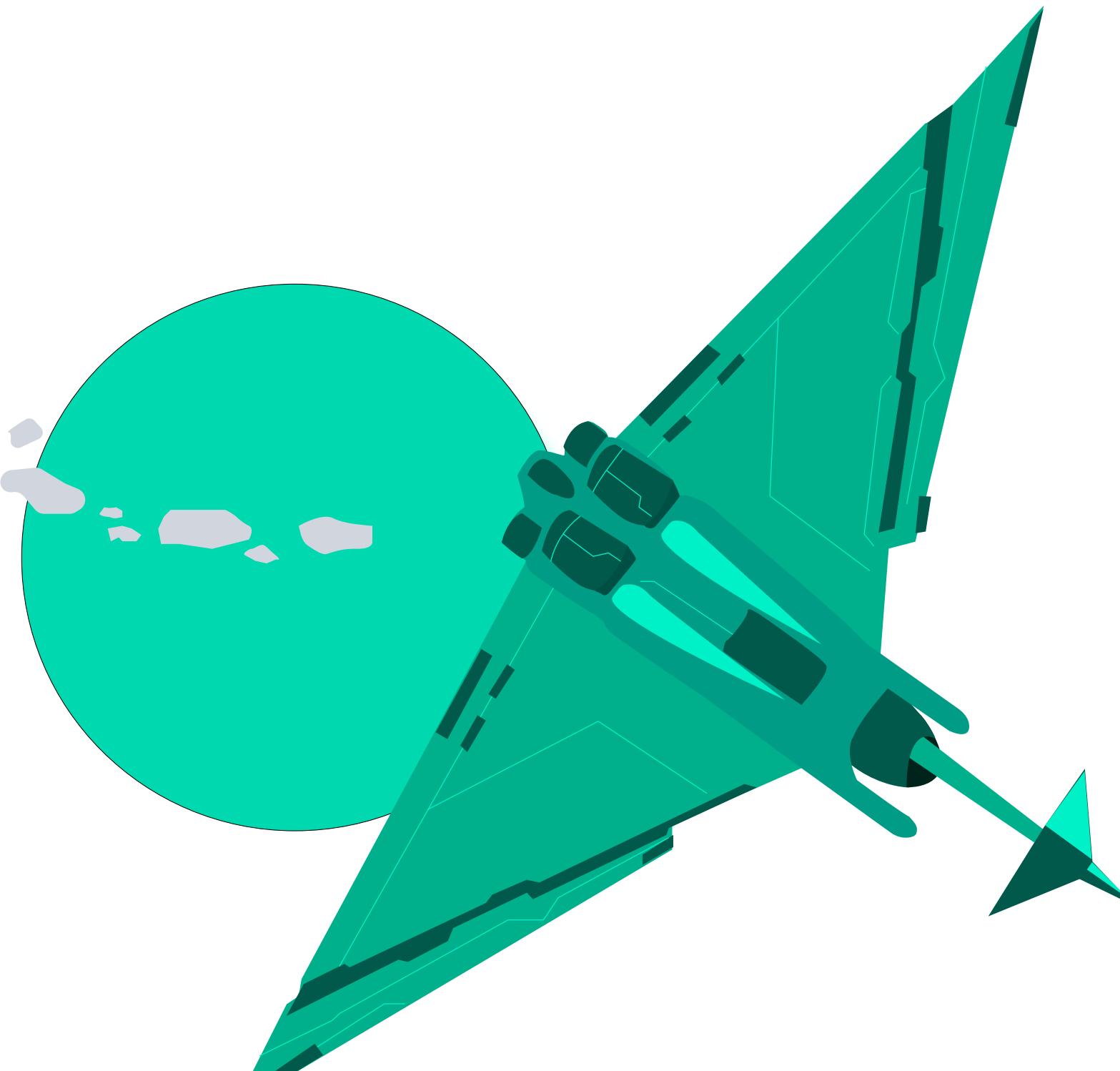
ID da transação	Tipo de transação	ID da conta	Valor	Data
987	DBT	103	R\$100	22/01/2014
979	CBT	103	R\$25	05/02/2014
981	DBT	104	R\$250	09/03/2014
982	CDT	105	R\$75	04/04/2014

Repare também que todas elas possuem um identificador único, conhecido como chave(key).

Uma dúvida que pode surgir: porque não criar uma só tabela com todas as informações?

A resposta é simples. É muito melhor você relacionar a tabela por identificadores e dividir suas tabelas em categorias.

Na imagem acima, caso usássemos uma única tabela, teríamos o sobrenome “blake” ao longo de todos os registros, repetindo ao menos 15 vezes. Separando em categorias, caso “George” se case e mude seu sobrenome para “Lima”, teremos que alterar apenas uma linha ao invés de 15.



## 1.2. Características fundamentais de um banco de dados

Um banco de dados possui características que podem ser resumidas por um acrônimo: ACID.

A => Atomicidade

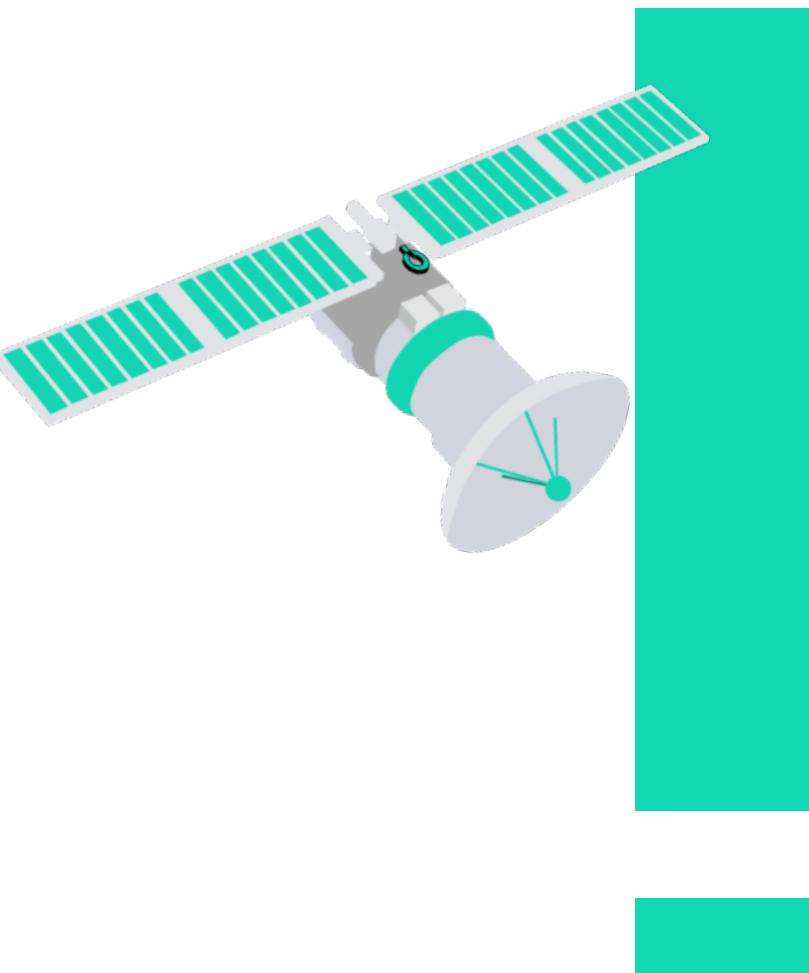
C => Consistência

I => Isolamento

D => Durabilidade

### Atomicidade:

Imagine que você deve modificar 1 milhão de linhas, porém durante o processo o banco de dados só conseguiu modificar 500 mil linhas. A característica da atomicidade define que essas mudanças só serão realizadas se toda a operação for realizada, como ela só foi metade realizada, as informações voltarão ao início da mesma forma.



### Consistência:

Garante que os bancos de dados vão ser encontrados com consistência nas seguintes características:

Todas as tabelas terão chave única

Restrições em uma tabela serão respeitadas

Os tipos de dados estabelecidos serão respeitados

Todas as relações entre chaves estarão garantidas

### Isolamento:

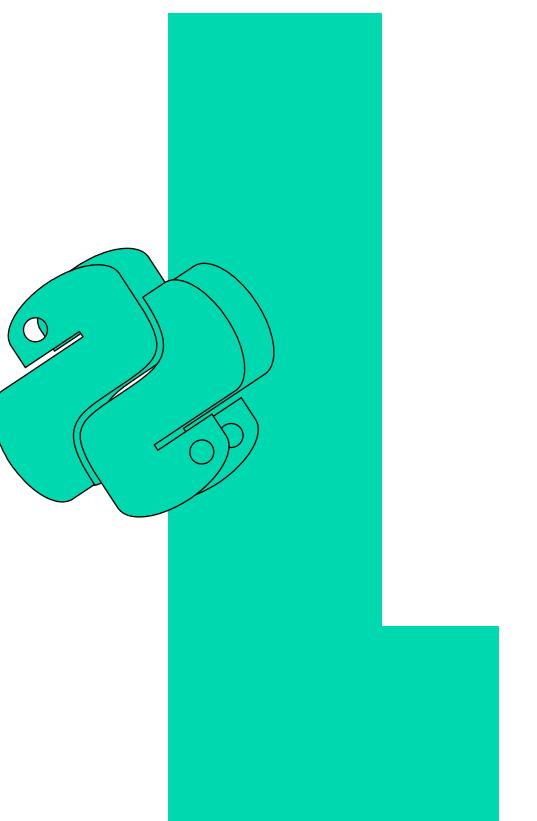
Garante que várias requisições, feitas ao mesmo tempo, e para o mesmo lugar, funcionem. Isolamento é um conjunto de técnicas que possibilitam que essas transações sejam feitas paralelamente sem que uma influencie na outra.

### Durabilidade:

Garante que os efeitos das requisições feitas persistam, mesmo em caso de queda de luz, queda de internet, travamento, etc.

### 1.3. Por que usar banco de dados?

O banco de dados é apenas um lugar onde podemos guardar nossas informações, e sua integração com o Python é rápida e segura. Imagine que sempre que você precisasse de uma informação da CVM você tivesse que acessar o site, baixar informações, etc. Não parece, mas isso demandaria muito tempo. Para isso foi criado o banco de dados: você só precisa recolher essas informações uma vez.



## Mundo 2

### 1.1. O que é banco de dados?

SQL é a linguagem utilizada na maioria dos bancos de dados. Ela é uma linguagem de consulta e, dessa forma, não é aconselhado fazer operações, por mais que seja possível. O objetivo principal é a consulta, criação e exclusão de tabelas.

Ela é considerada uma linguagem fácil, pois não precisa de conhecimentos profundos em programação e não possui pacotes externos. O Python, por exemplo, tem o pandas, numpy, dentre outros, enquanto o SQL possui apenas a sintaxe base.

Por mais que os bancos relacionais(MySQL, PostgreSQL, Oracle, MariaDB) utilizem a linguagem SQL, ela difere em alguns detalhes de um para outro, mas nada que interfira na compreensão.

## Mundo 3

### 3.1. Instalando o MySQL no Windows

Acesse a página do instalador do SQL e selecione o seu sistema operacional:

<https://dev.mysql.com/downloads/installer/>

#### MySQL Community Downloads

MySQL Installer

General Availability (GA) Releases Archives

### MySQL Installer 8.0.31

Select Operating System: Microsoft Windows

Platform	Version	File Size	Action
Windows (x86, 32-bit), MSI Installer	8.0.31	5.5M	<a href="#">Download</a>
Windows (x86, 32-bit), MSI Installer	8.0.31	431.7M	<a href="#">Download</a>

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

E clique no botão selecionando o instalador de menor tamanho:

#### MySQL Community Downloads

MySQL Installer

General Availability (GA) Releases Archives

### MySQL Installer 8.0.31

Select Operating System: Microsoft Windows

Looking for previous GA versions?

Platform	Version	File Size	Action
Windows (x86, 32-bit), MSI Installer	8.0.31	5.5M	<a href="#">Download</a>
Windows (x86, 32-bit), MSI Installer	8.0.31	431.7M	<a href="#">Download</a>

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

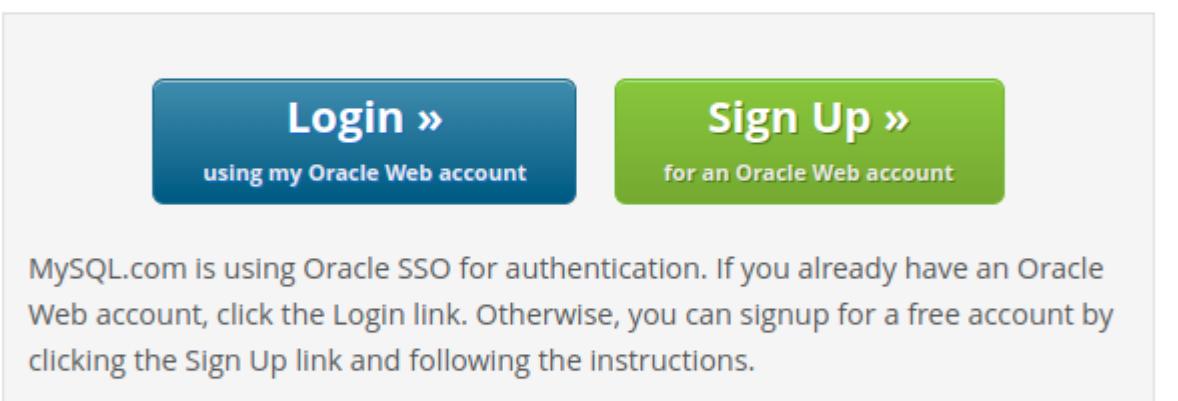
Você será redirecionado para outra página Clique no link abaixo para começar o download automaticamente.

## MySQL Community Downloads

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system



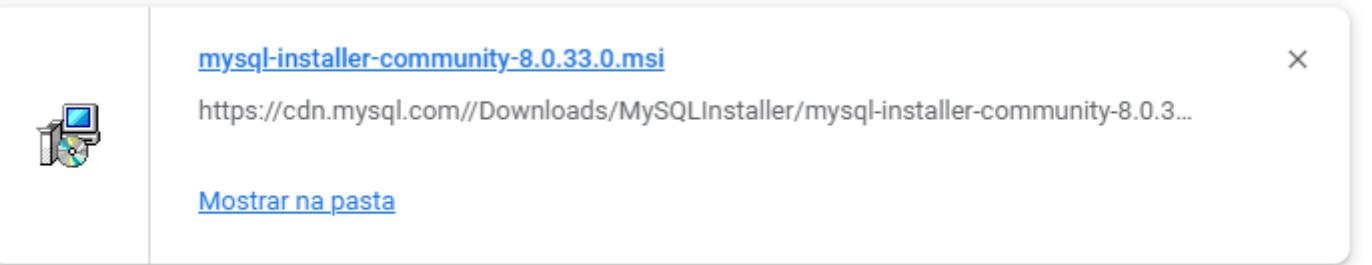
No thanks, just start my download.

ORACLE © 2023 Oracle

[Privacy / Do Not Sell My Info](#) | [Terms of Use](#) | [Trademark Policy](#) |

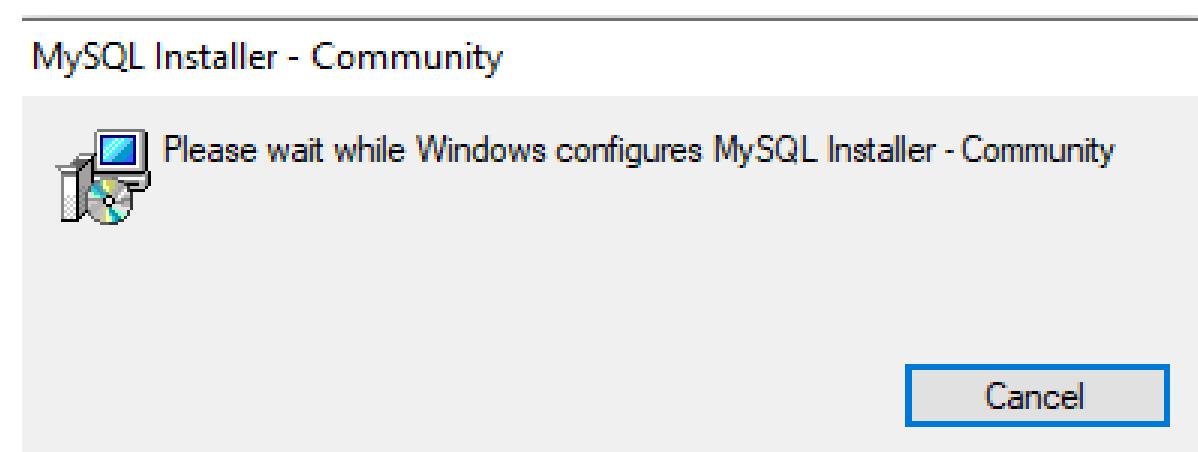


Aperte Ctrl + j e verifique se o arquivo de instalação está concluído, caso sim, clique para começar a instalação.



Dependendo do seu computador, talvez seja necessário instalar algumas dependências para que o MySQL funcione, só colocar o nome da dependência necessária no Google e baixar.

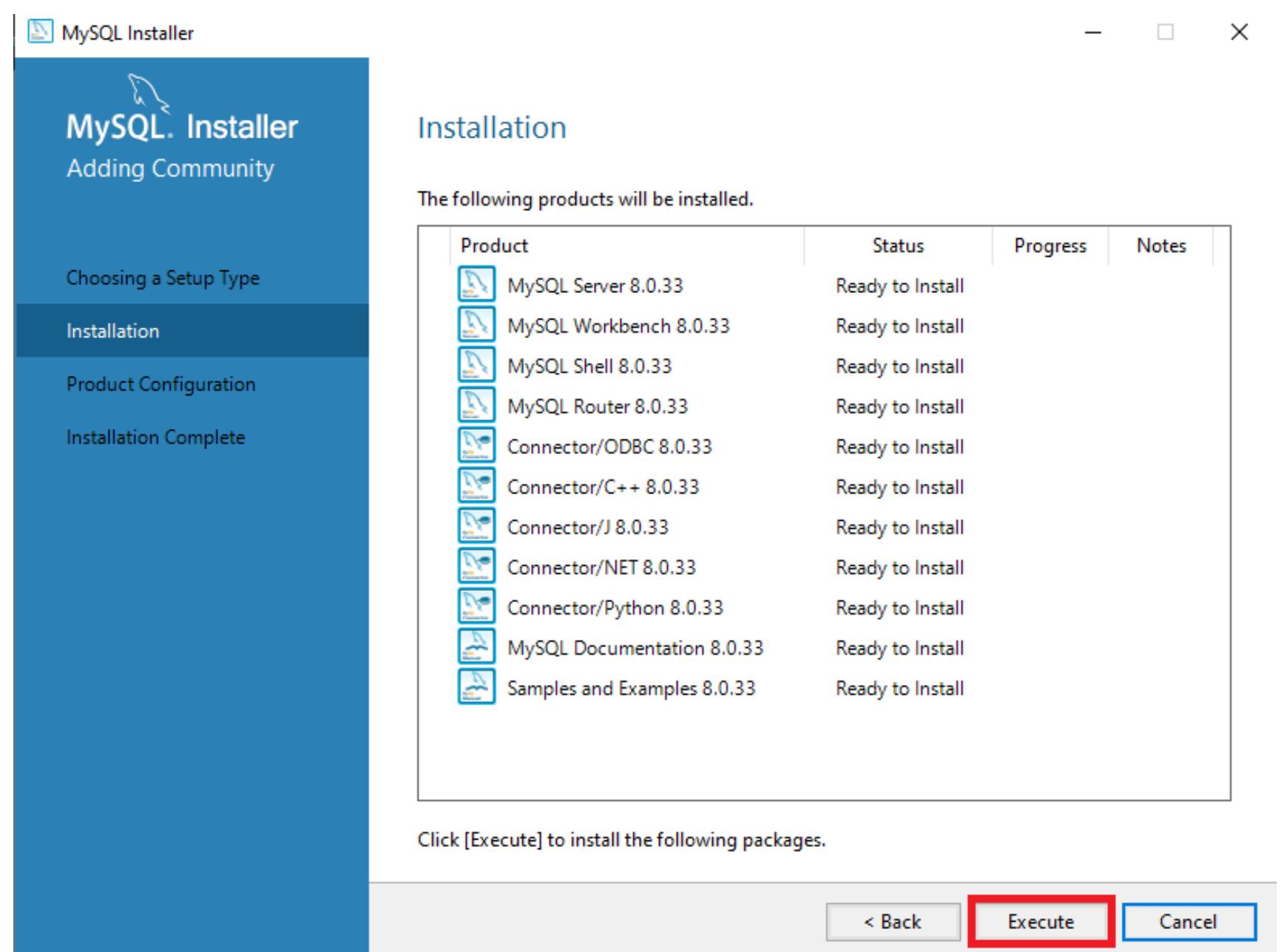
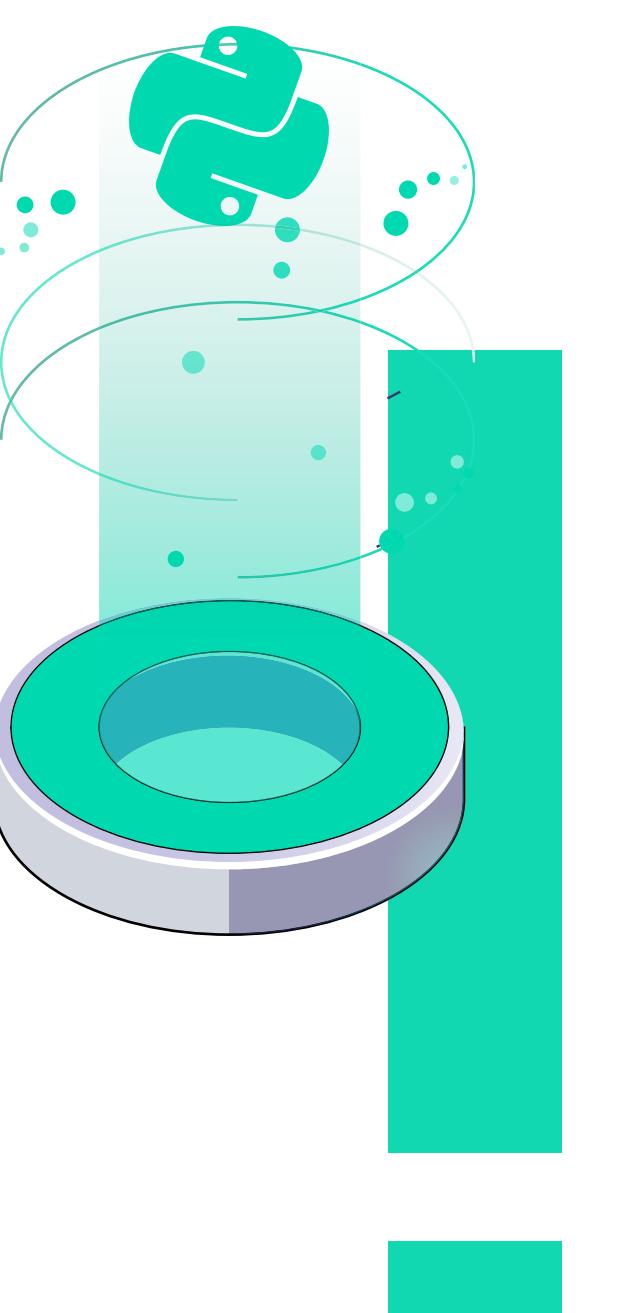
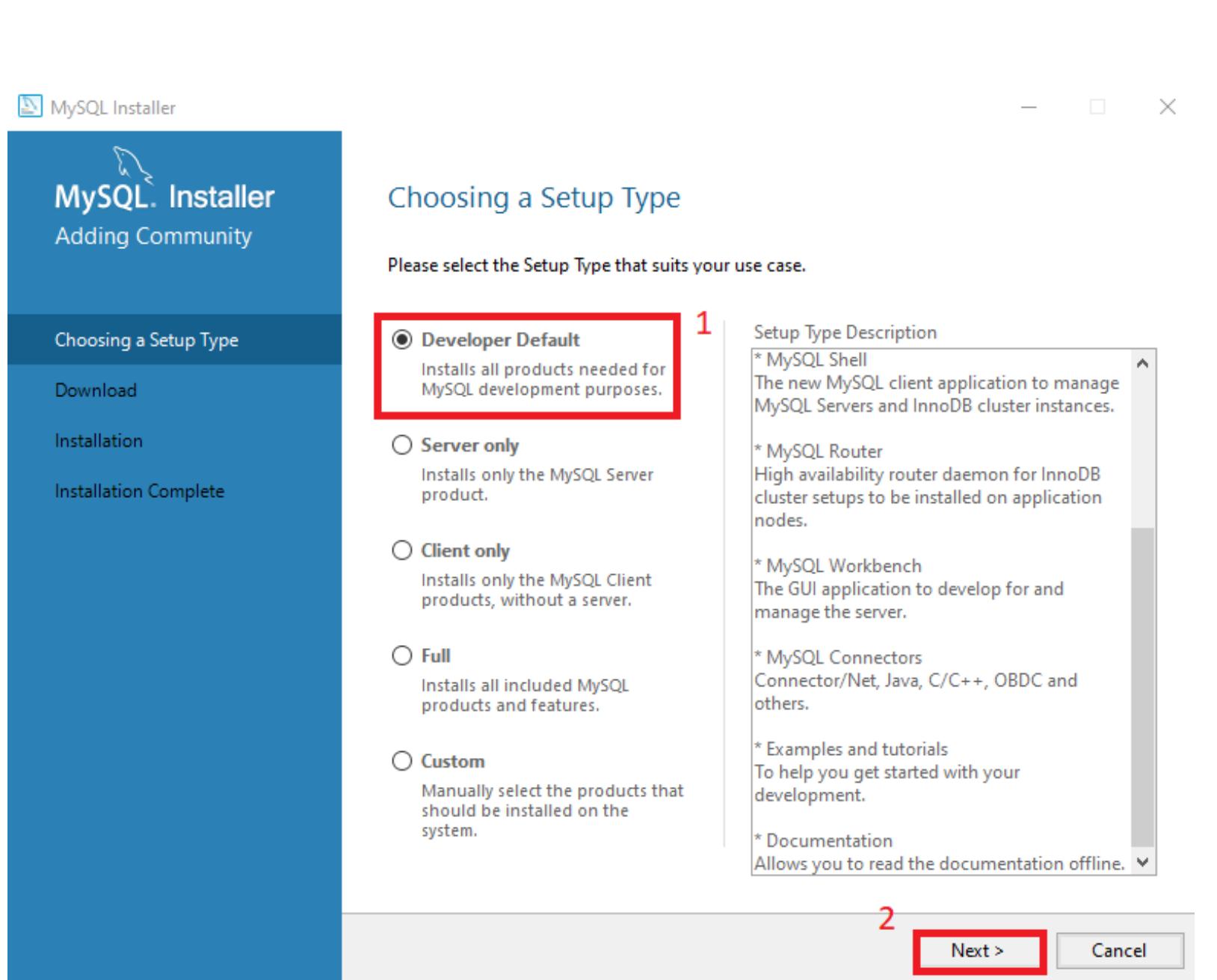
Após clicar no arquivo de instalação, essa tela aparecerá:



Espere, até que ele peça permissão e confirme a instalação.

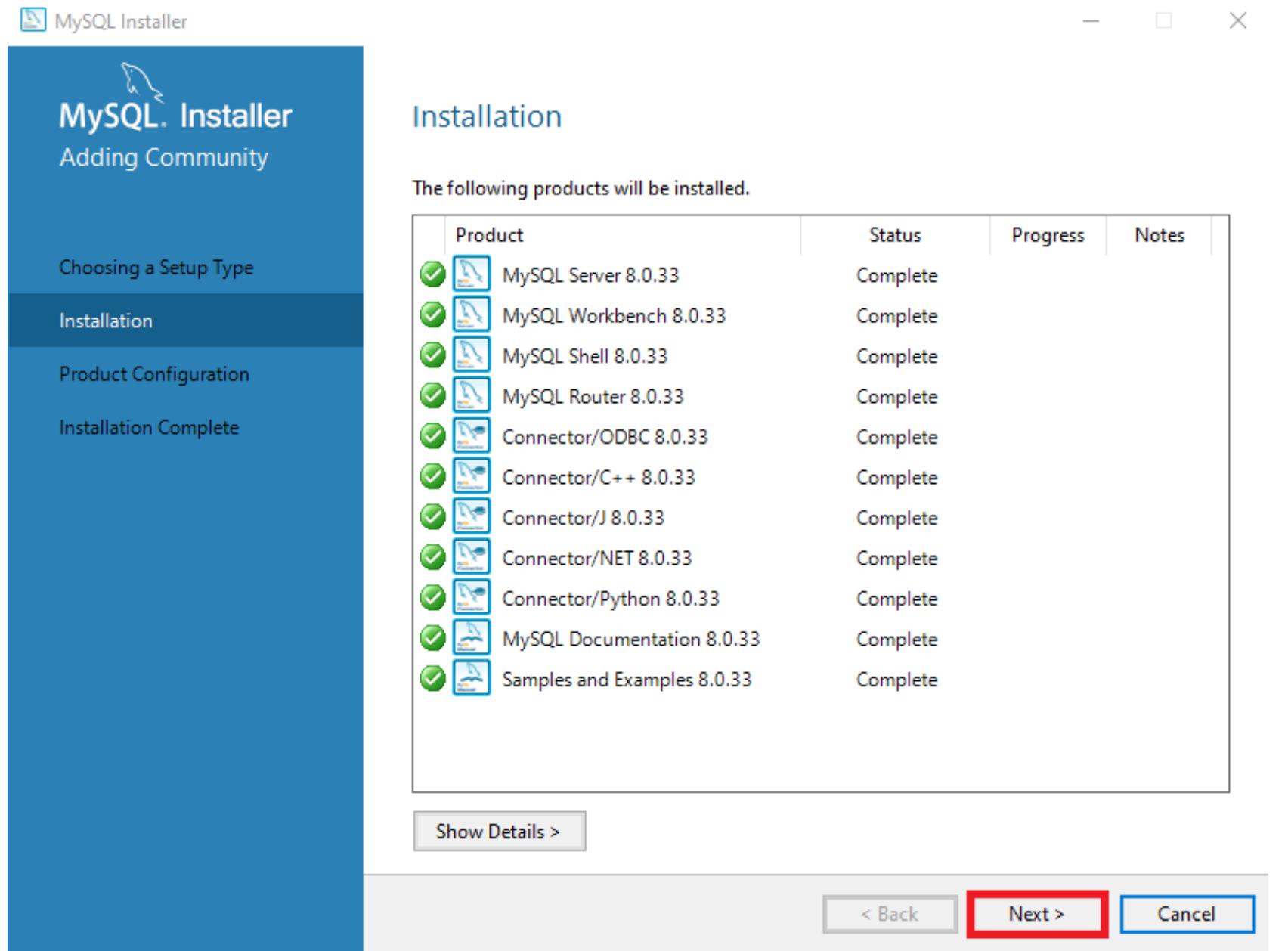
Após essas etapas, a tela a seguir aparecerá, clique em "Developer

Default" e depois em "Next >":

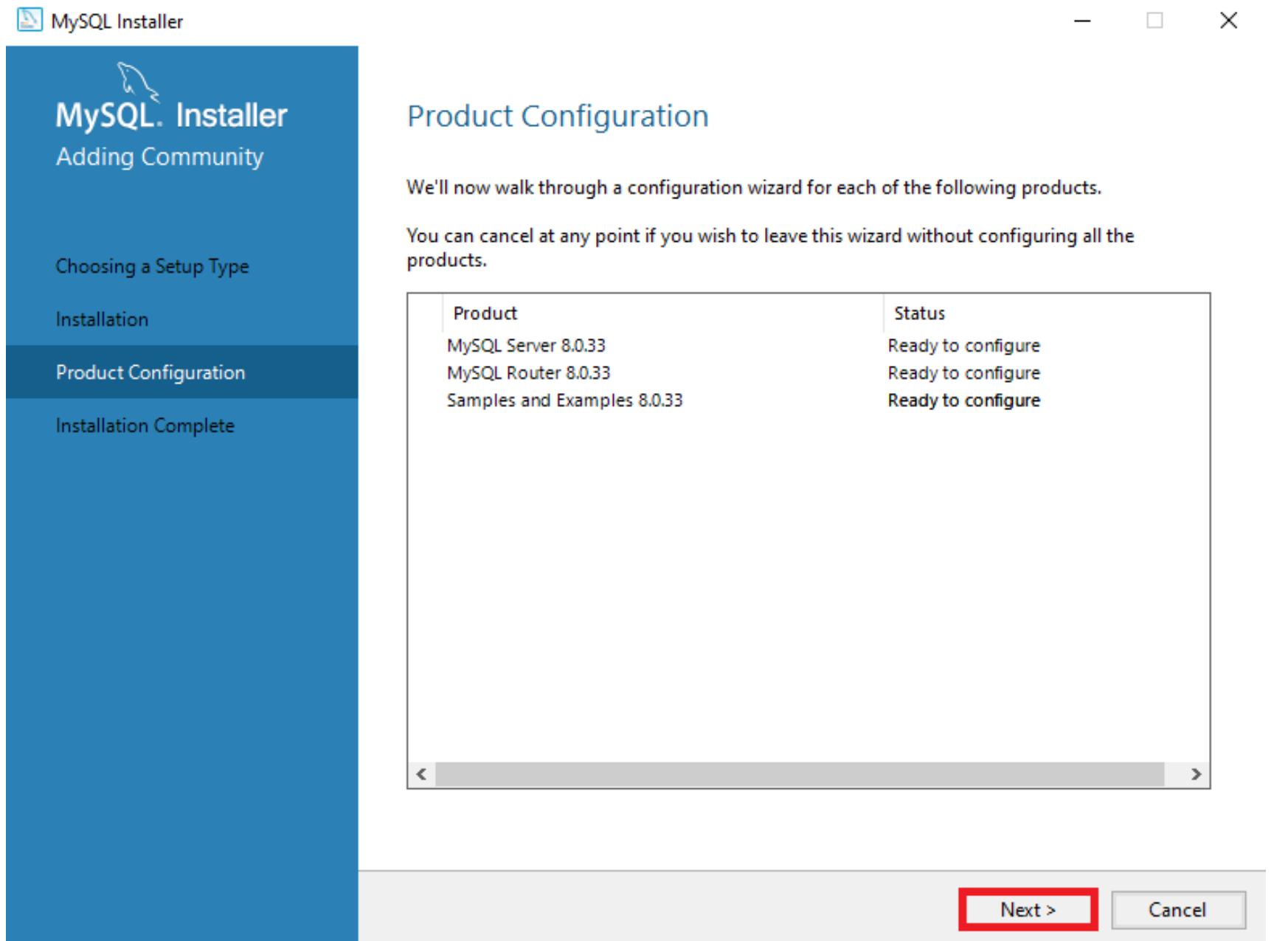


Esses são os arquivos necessários para fazer o MySQL funcionar corretamente no seu computador, clique em "Execute" e espere os arquivos serem instalados no seu computador:

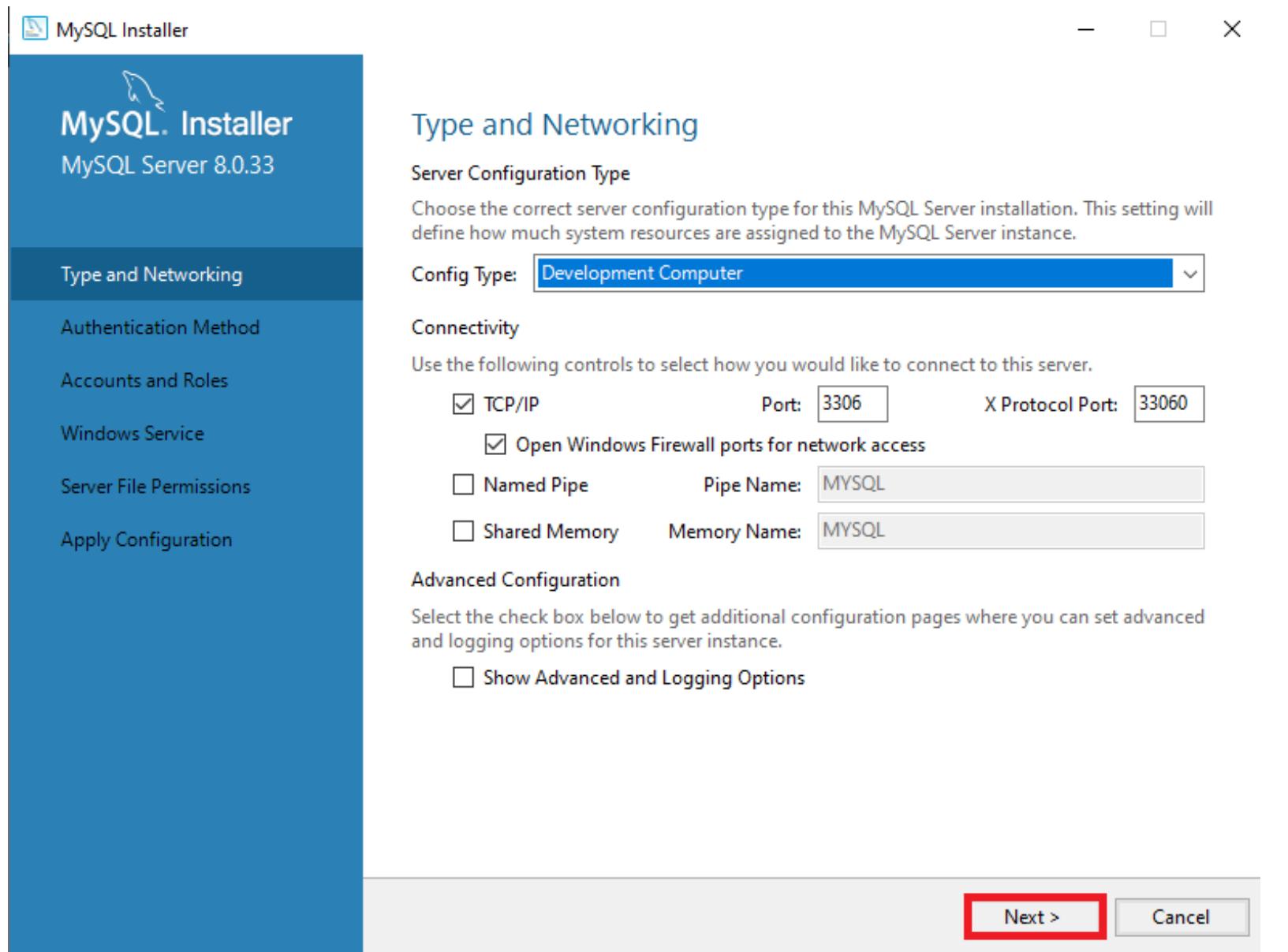
Após os arquivos serem instalados, clique em "Next >":



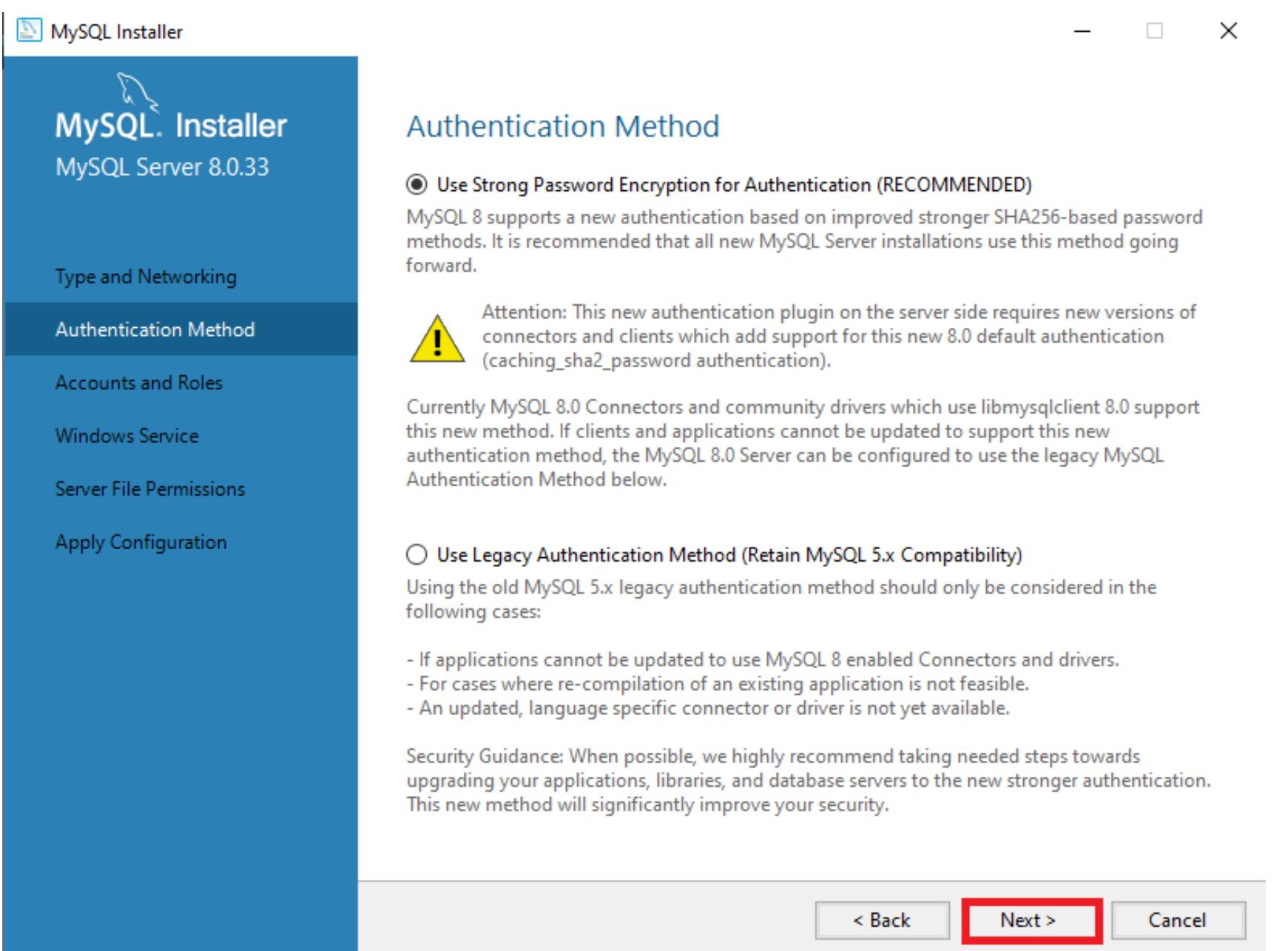
Clique em "Next >":



Clique em "Next>":

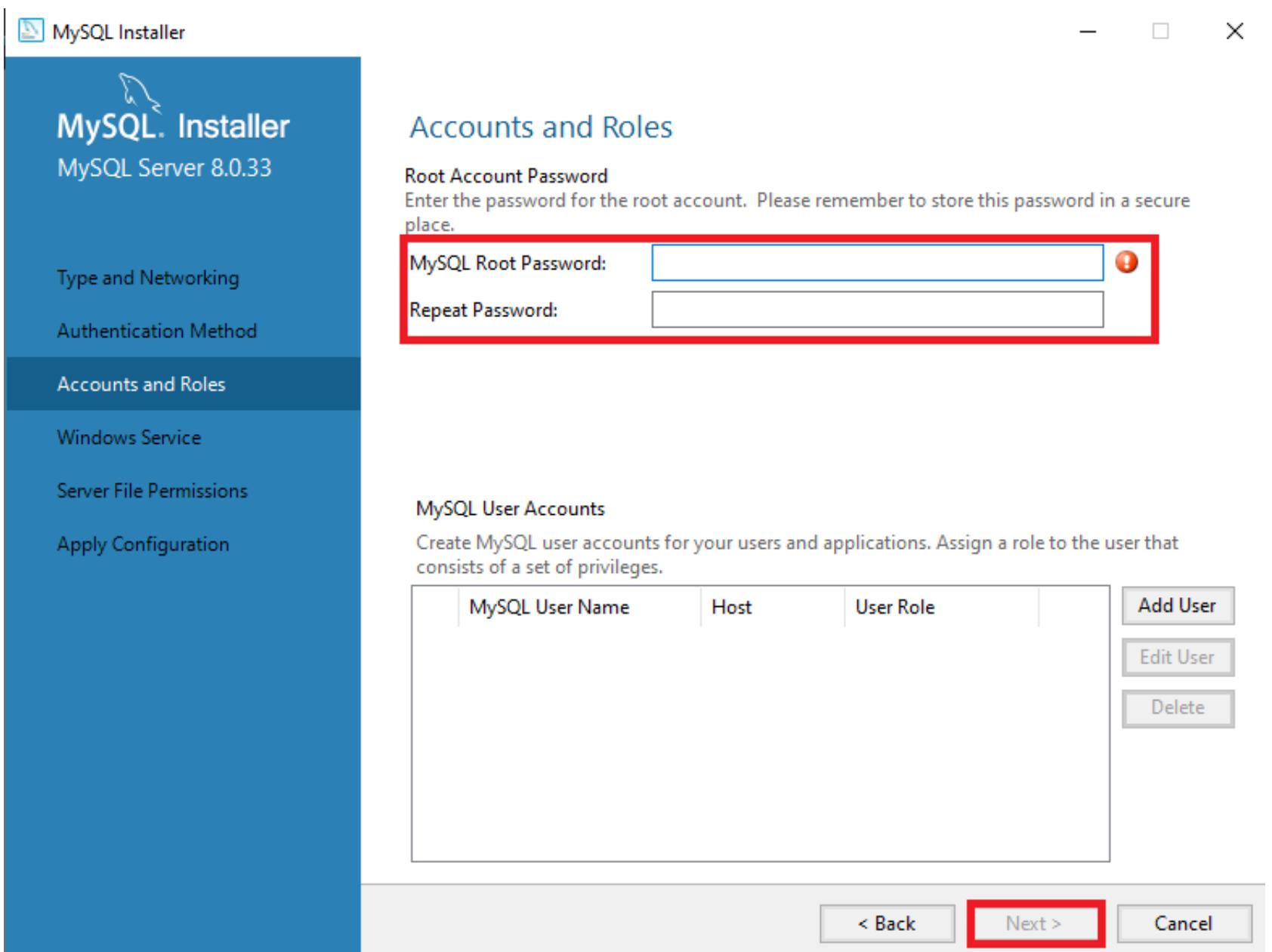


Clique em "Next>":

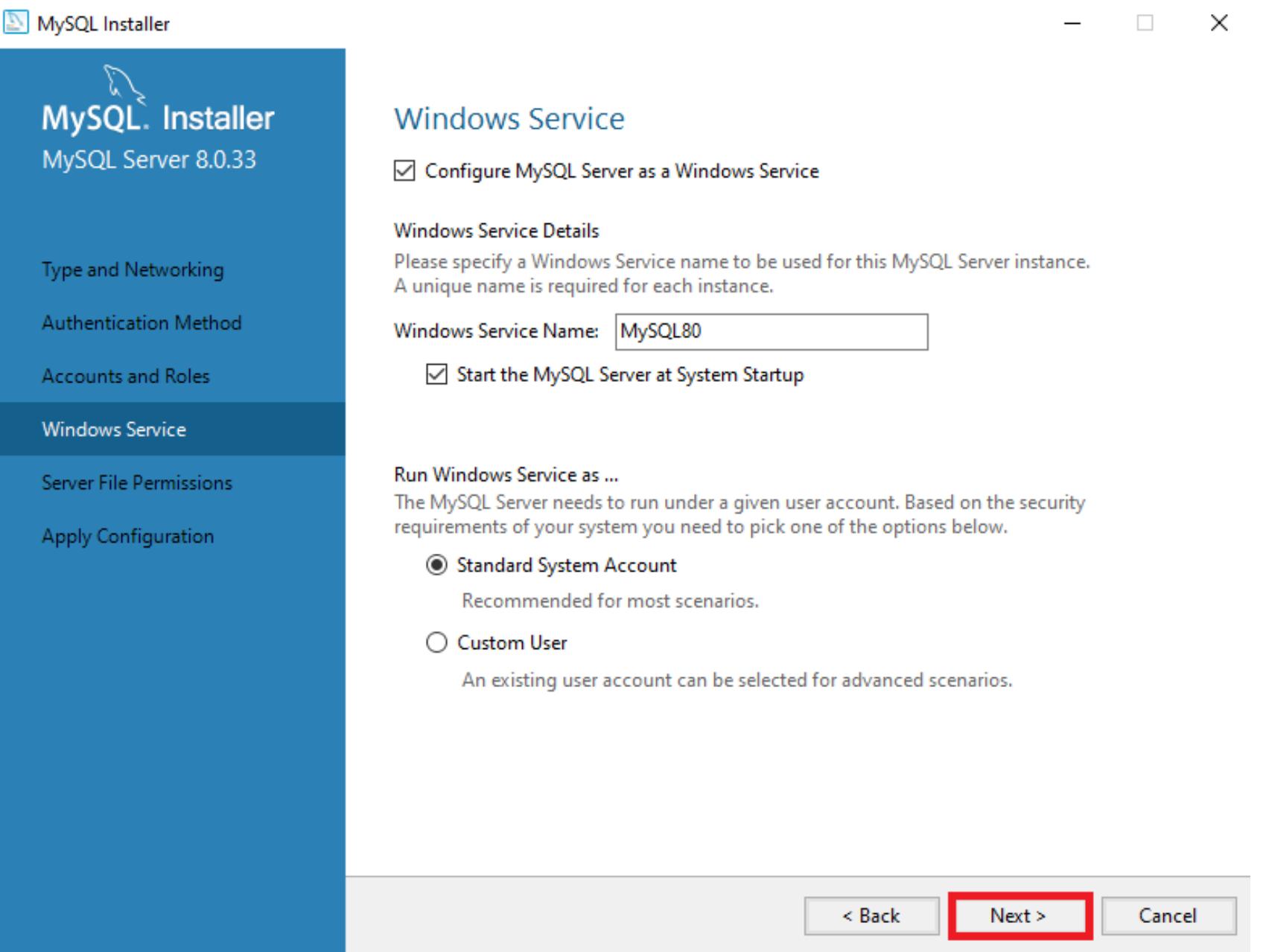


Escolha uma senha, lembre-se de a salvar em algum lugar, ela será importante.

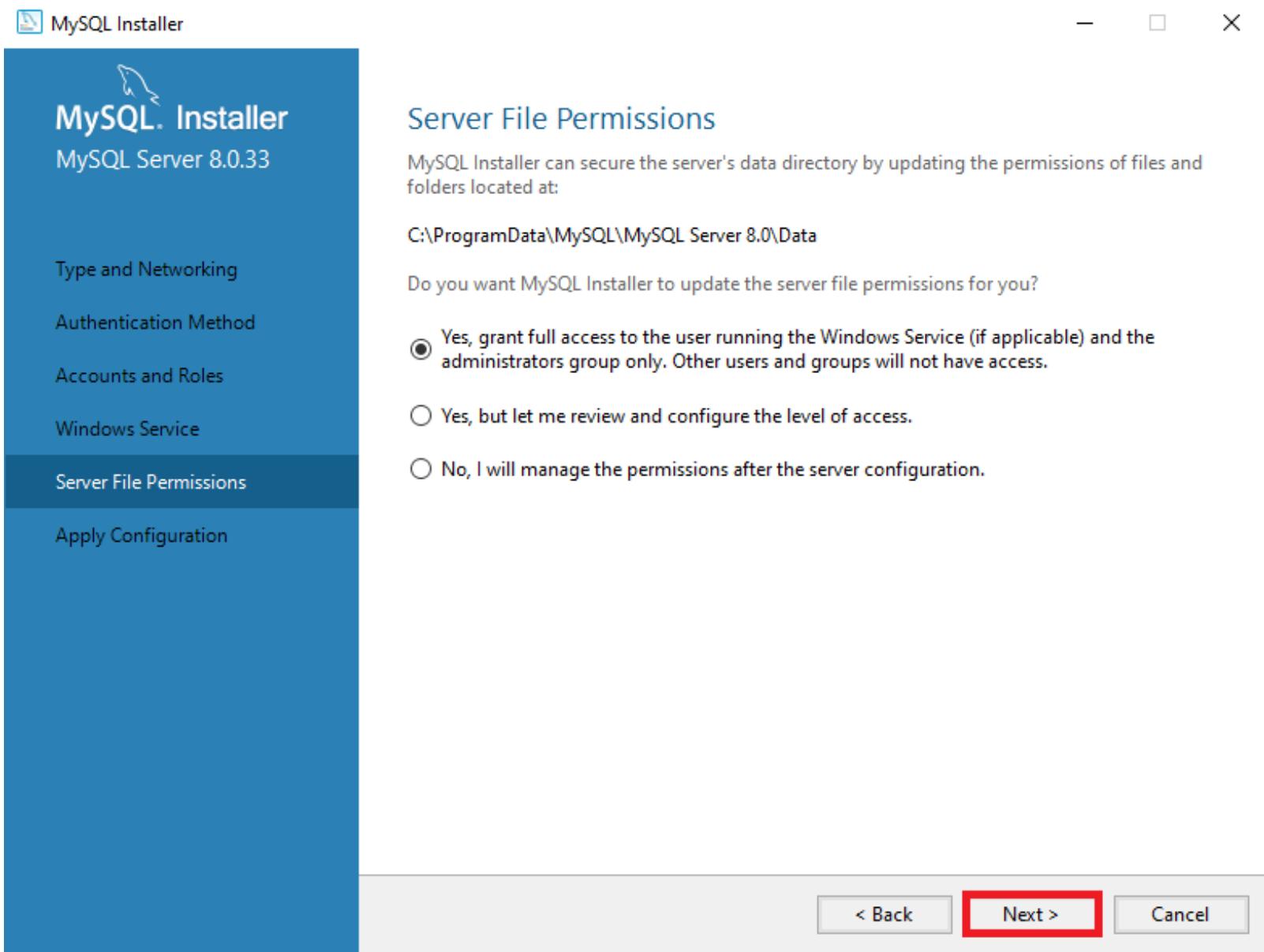
Depois clique em "Next >":



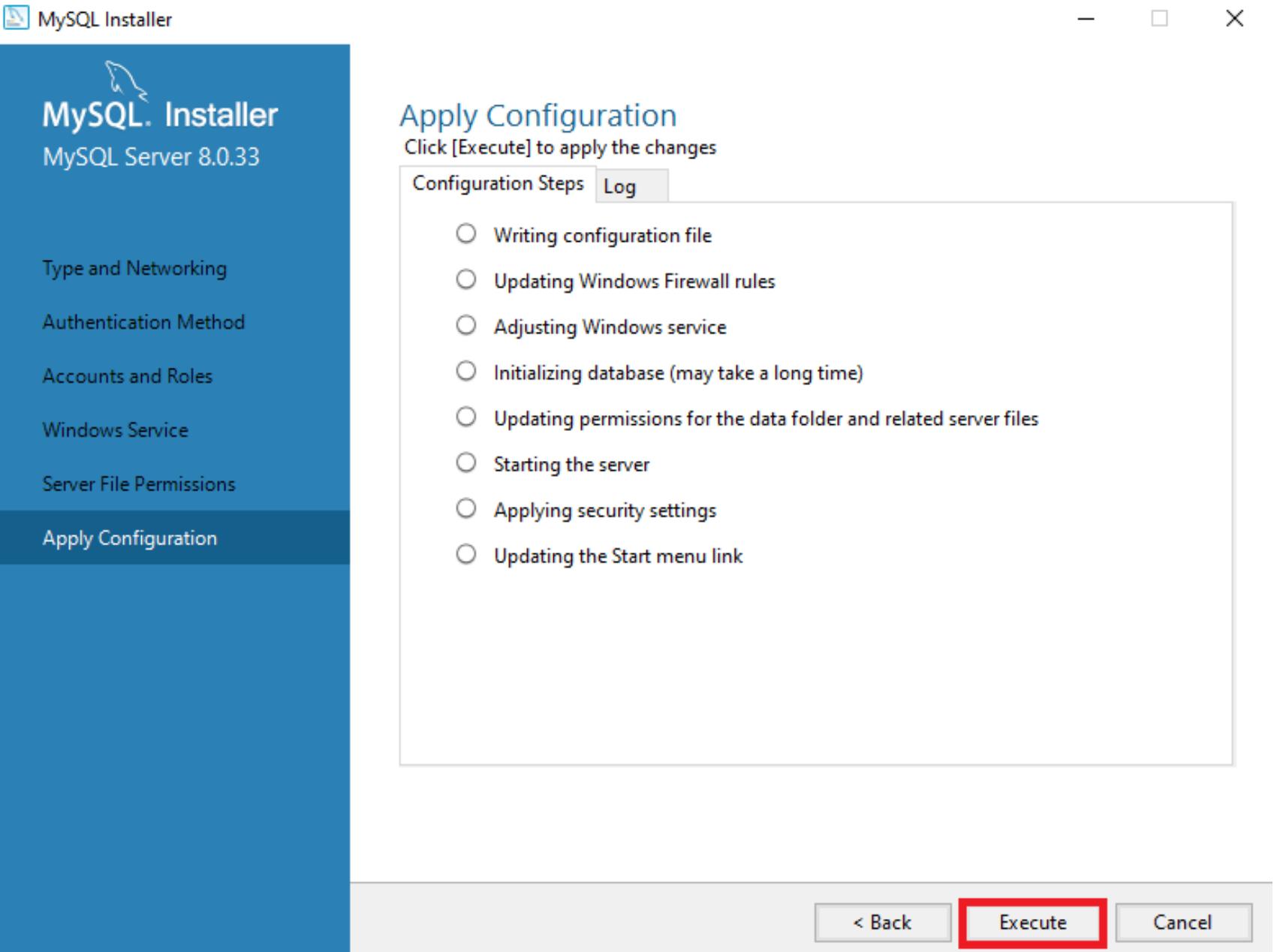
Clique em "Next >":



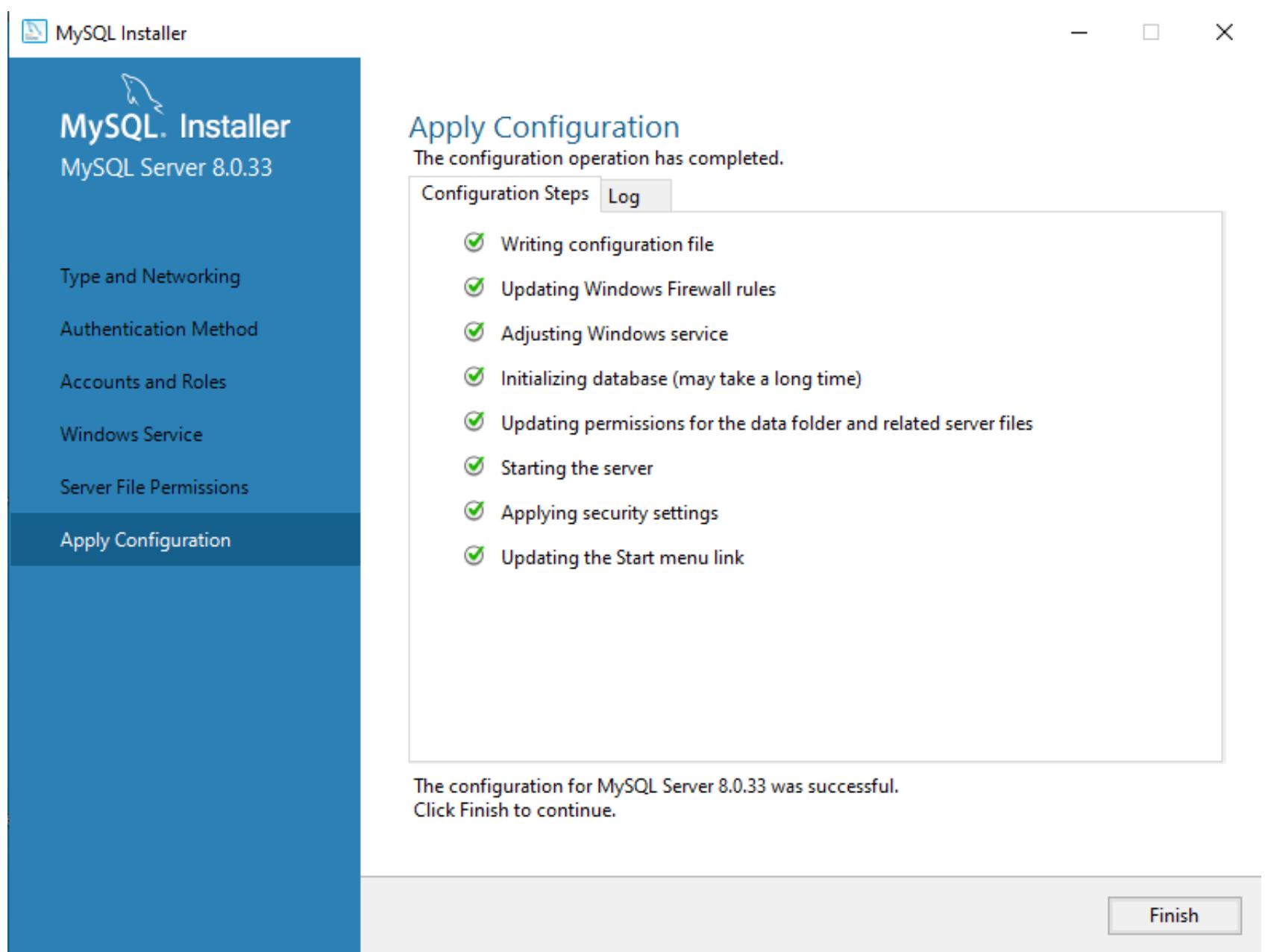
Clique em "Next >":



Clique em "Execute" e espere:



Pronto, instalação finalizada:



### 3.2. Instalando o MySQL no Linux

Passo 1. Abra um terminal (Usando o Dash ou pressionando as teclas CTRL+ALT+T);

Passo 2. Atualize o APT com o comando:

```
>> sudo apt-get update
```

Passo 3. Confira a versão do seu sistema, para isso, use o seguinte comando no terminal:

```
>> lsb_release -rs
```

Passo 4. Se a versão do seu Linux Ubuntu for igual a 20.04 utilize o seguinte comando:

```
>> wget http://cdn.mysql.com/Downloads/MySQLGUITools/mysql-workbench-community_8.0.25-1ubuntu20.04_amd64.deb -O mysql-workbench-community.deb
```

Passo 5. Instale o pacote baixado com o comando a seguir (mesmo que apareça um erro, vá para o próximo passo que ele corrigirá tudo);

```
>> sudo dpkg -i mysql-workbench-community.deb
```

Passo 6. Para instalar as dependências e finalizar a instalação, use o comando:

```
>> sudo apt-get -f install
```

Pronto! Agora, quando quiser iniciar o programa, digite mysql no Dash (ou em um terminal, seguido da tecla TAB).

Caso sua versão seja diferente de 20.04 acesse o seguinte site e siga os passos:

<https://www.edivaldobrito.com.br/como-instalar-o-instalar-mysql-workbench-no-ubuntu-e-derivados/>

### 3.3. Instalando o MySQL no MAC

Acesse a página do instalador do SQL e selecione o seu sistema operacional:

<https://dev.mysql.com/downloads/installer/>

The screenshot shows the MySQL Workbench 8.0.19 download page. At the top, there are tabs for 'General Availability (GA) Releases' (which is selected), 'Archives', and a search icon. Below the tabs, it says 'Select Operating System:' with a dropdown menu set to 'macOS'. A note indicates that packages for Catalina (10.15) are compatible with Mojave (10.14). The main section displays a single download link for 'macOS (x86, 64-bit), DMG Archive' with version 8.0.19, size 108.9M, and a 'Download' button. A note at the bottom suggests using MDS checksums and GnuPG signatures for package integrity.

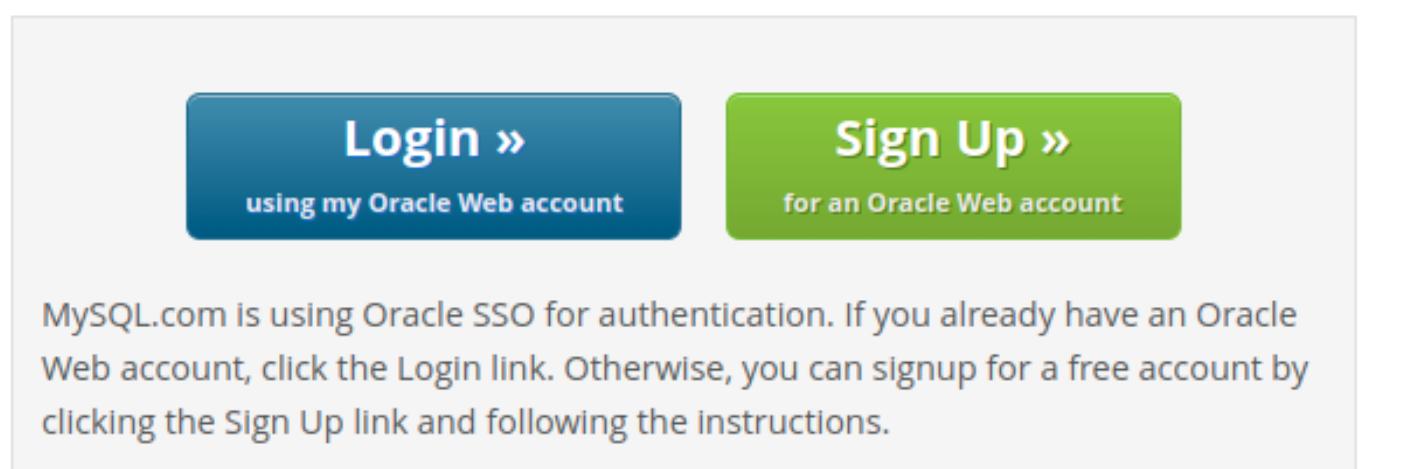
Você será redirecionado para outra página Clique no link abaixo para começar o download automaticamente.

## MySQL Community Downloads

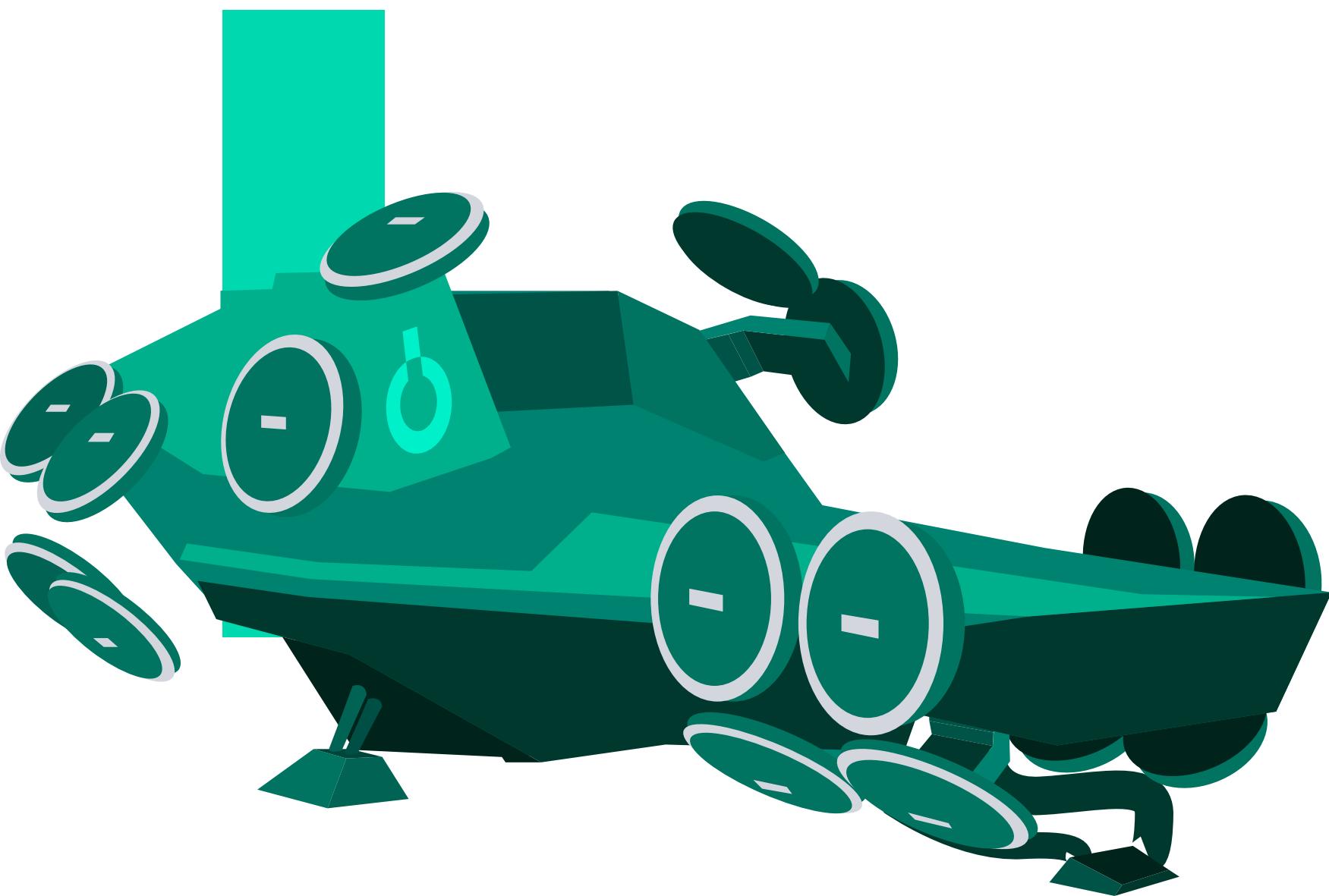
Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system



Depois de completo o download clique no arquivo de instalação na pasta Downloads e quando apresentado com a tela abaixo, arraste o ícone do MySQL Workbench para a pasta Applications.



## Mundo 4

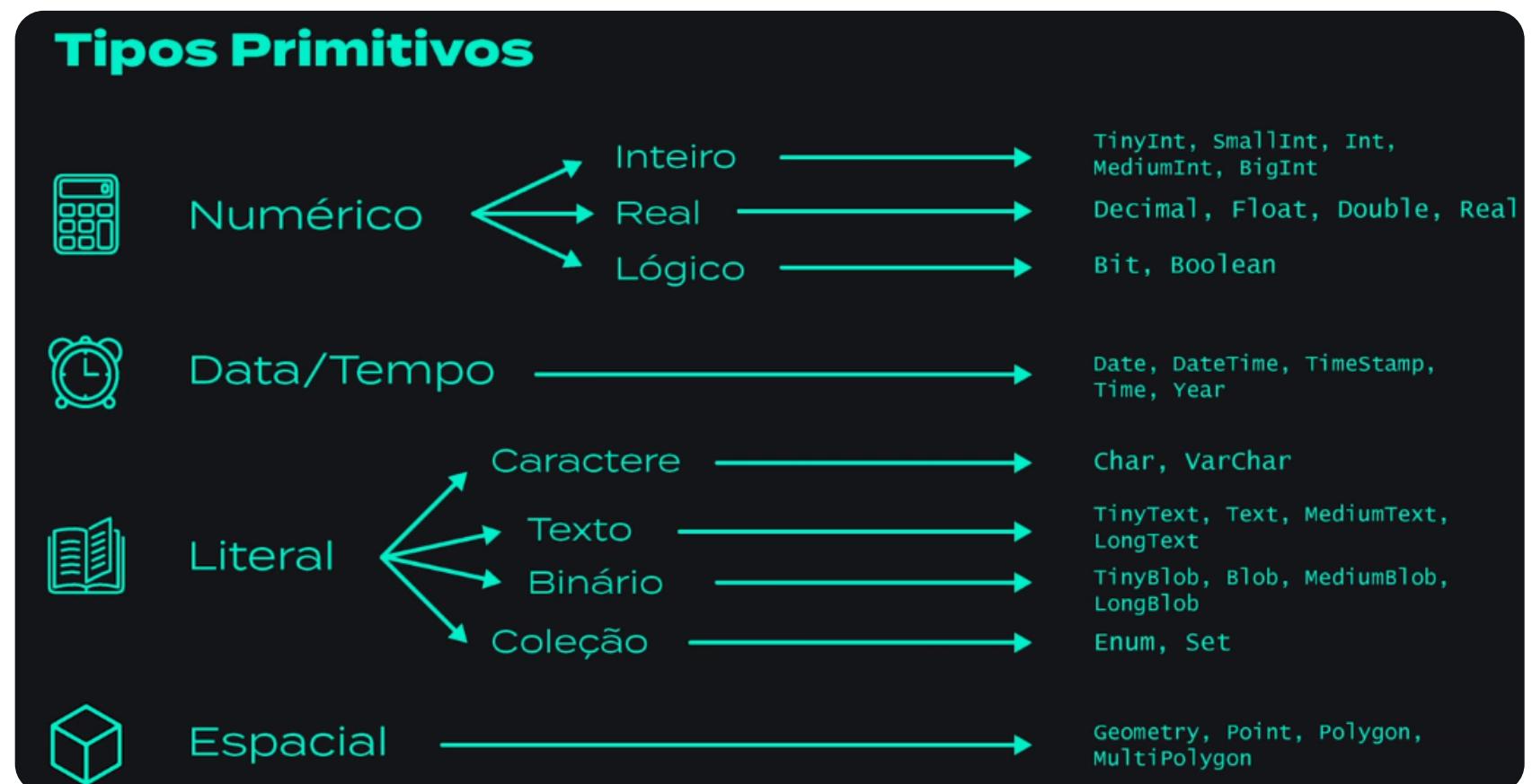
### 4.1. Tipos de dados

#### 4.1.1. Como funciona?

Cada dado no banco de dados SQL ocupa um tamanho de bytes. Quanto mais bytes, maior será o consumo do seu banco de dados. Pensando nisso, foram criados os tipos de dados, para o usuário poder administrar e consumir apenas o necessário.

Hoje em dia, dependendo da sua aplicação, isso não faz tanta diferença quanto há anos atrás, quando as máquinas eram super limitadas e cada gigabyte importava.

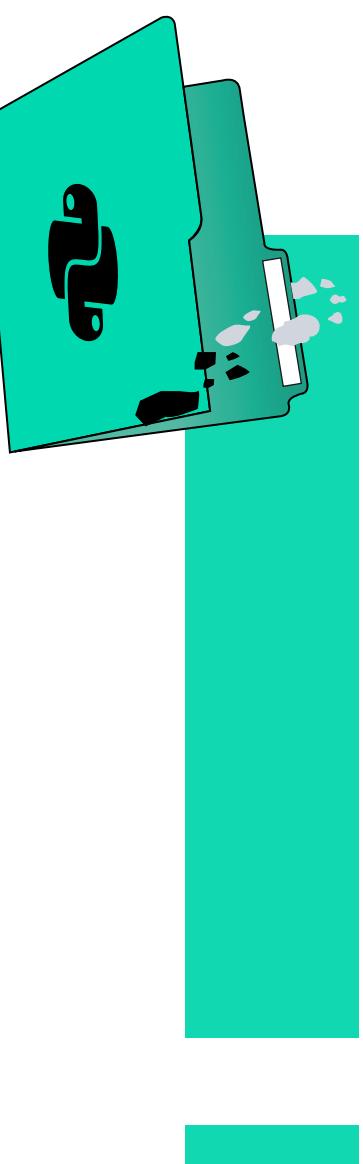
Esses são os tipos de dados:



Como pode ver, o tipo de dado vai muito além de definir o que é um texto e o que é um número. Cada tipo primitivo de dado possui um sub item, que o especifica ainda mais. Mas como você pode saber quando usar cada um?

É simples: pense que você administra o banco de dados de uma corretora e é responsável pelas ordens de compra e venda de clientes, porém seus clientes só aportam até R\$ 10.000,00. Não faz sentido você utilizar um tipo de inteiro TinyInt ou até mesmo um BigInt.

É importante que você saiba que se algum valor estiver fora do intervalo definido pelo tipo, de acordo com a explicação abaixo, o SQL não aceitará o envio dessa informação e retornará um erro.



## 4.1.2. Numéricos

### 4.1.2.1. Inteiro

- **TinyInt**: ocupa 1 byte e pode armazenar valores de -128 a 127 (assumindo que o valor é com sinal) ou de 0 a 255 (assumindo que o valor não possui sinal).
- **SmallInt**: ocupa 2 bytes e pode armazenar valores de -32.768 a 32.767 (assumindo que o valor é assinado) ou de 0 a 65.535 (assumindo que o valor é não assinado).
- **Int**: ocupa 4 bytes e pode armazenar valores de -2.147.483.648 a 2.147.483.647 (assumindo que o valor é assinado) ou de 0 a 4.294.967.295 (assumindo que o valor é não assinado).

- **MediumInt**: ocupa 3 bytes e pode armazenar valores de -8.388.608 a 8.388.607 (assumindo que o valor é assinado) ou de 0 a 16.777.215 (assumindo que o valor é não assinado).
- **BigInt**: ocupa 8 bytes e pode armazenar valores de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807 (assumindo que o valor é assinado) ou de 0 a 18.446.744.073.709.551.615 (assumindo que o valor é não assinado).



#### 4.1.2.2. Real

- **DECIMAL**: é um tipo de dado que armazena valores numéricos exatos, com precisão fixa. A precisão é determinada pela quantidade de dígitos que se pode armazenar e pela escala, que representa o número de casas decimais que podem ser armazenadas. O DECIMAL é útil para armazenar valores monetários, por exemplo, onde a precisão é importante e não se pode ter erros de arredondamento.
- **FLOAT e DOUBLE**: são tipos de dados que armazenam valores numéricos com precisão aproximada (floating point). Ambos são usados para valores com decimais, mas o DOUBLE oferece maior precisão do que o FLOAT. Esses tipos de dados são úteis para cálculos matemáticos, mas podem apresentar pequenas imprecisões nos cálculos devido à forma como a precisão é armazenada.
- **REAL**: é um tipo de dado que armazena valores numéricos com precisão aproximada, semelhante ao FLOAT. Ele usa menos espaço de armazenamento do que o DOUBLE, mas também oferece menor precisão. O REAL é útil para armazenar valores numéricos com precisão reduzida, como coordenadas geográficas.

#### 4.1.2.3. Lógico

- **BIT**: é um tipo de dado que armazena valores booleanos como 0 ou 1. O BIT é usado quando se deseja economizar espaço de armazenamento, pois apenas 1 bit é utilizado para armazenar cada valor booleano. No entanto, o BIT só pode armazenar valores verdadeiros ou falsos e não pode ser usado para representar outros valores booleanos, como NULL ou UNKNOWN.
- **BOOLEAN**: é um tipo de dado que armazena valores booleanos como TRUE ou FALSE. Ao contrário do BIT, o BOOLEAN é uma representação mais semântica do valor booleano e pode ser usado para representar outros valores booleanos, como NULL ou UNKNOWN. No entanto, o BOOLEAN ocupa mais espaço de armazenamento do que o BIT.

#### 4.1.3. Data/Tempo

- **DATE**: é um tipo de dado que armazena apenas informações de data, como ano, mês e dia. A precisão do DATE é de um dia inteiro e o formato de armazenamento é 'YYYY-MM-DD'.
- **DATETIME**: O DATETIME armazena informações de data e hora com uma precisão de um segundo. O formato de armazenamento do DATETIME é 'YYYY-MM-DD HH:MM:SS'.
- **TIMESTAMP**: O TIMESTAMP pode armazenar informações de data e hora com precisão de frações de segundo (geralmente até 6 dígitos após a vírgula). O formato do TIMESTAMP é 'YYYY-MM-DD HH:MM:SS.uuuuuu', onde 'uuuuuu' representa as frações de segundo.
- **TIME**: é um tipo de dado que armazena informações de hora, sem informações de data. A precisão do TIME é de um segundo e o formato de armazenamento é 'HH:MM:SS'.

- **YEAR**: é um tipo de dado que armazena informações de ano, em formato de dois dígitos ou quatro dígitos. O formato de armazenamento do YEAR é 'YYYY' ou 'YY'.

#### 4.1.4. Literal

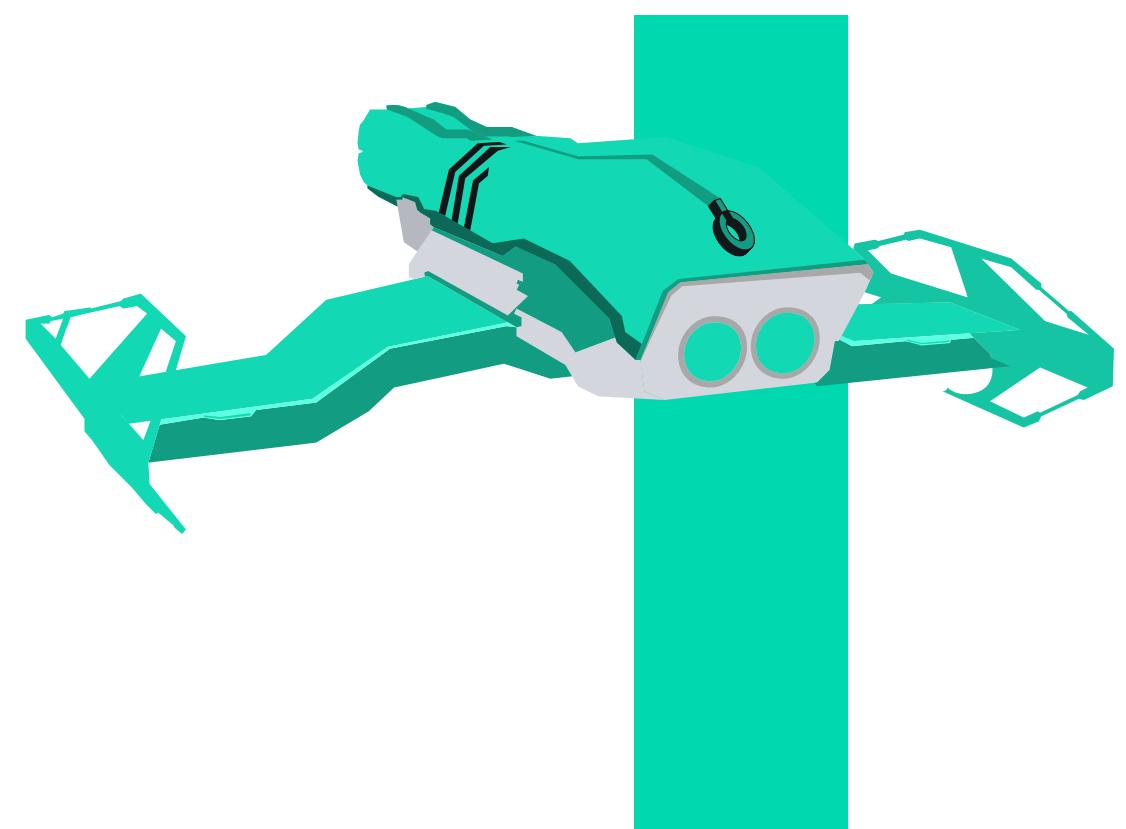
##### 4.1.4.1. Caractere

- **CHAR**: é um tipo de dado que armazena um tamanho fixo de caracteres. Por exemplo, se um campo CHAR for definido como CHAR(10), ele sempre ocupará 10 bytes, independentemente de quantos caracteres ele contenha. Esse tipo de dado é ideal para armazenar valores de texto que tenham um tamanho previsível e constante, como códigos de produtos ou números de telefone.

- **VARCHAR**: é um tipo de dado que armazena um tamanho variável de caracteres. Por exemplo, se um campo VARCHAR for definido como VARCHAR(10), ele pode armazenar de 1 a 10 caracteres. O espaço em disco utilizado é proporcional ao tamanho do valor armazenado. Esse tipo de dado é ideal para armazenar valores de texto que tenham um tamanho variável, como descrições de produtos ou mensagens de texto.

##### 4.1.4.2. Texto

- **TINYTEXT**: é um tipo de dado que pode armazenar até 255 caracteres.
- **TEXT**: é um tipo de dado que pode armazenar até 65.535 caracteres.
- **MEDIUMTEXT**: é um tipo de dado que pode armazenar até 16.777.215 caracteres.
- **LONGTEXT**: é um tipo de dado que pode armazenar até 4.294.967.295 caracteres.



#### 4.1.4.3. Binário

São usados em bancos de dados SQL para armazenar dados binários, como imagens, arquivos de áudio ou vídeo, documentos PDF, entre outros. A diferença entre eles está no tamanho máximo dos dados que podem ser armazenados.

- **TINYBLOB**: é um tipo de dado que armazena dados binários de tamanho máximo de 255 bytes.
- **BLOB**: é um tipo de dado que armazena dados binários de tamanho máximo de 65.535 bytes (64KB).
- **MEDIUMBLOB**: é um tipo de dado que armazena dados binários de tamanho máximo de 16.777.215 bytes (16MB).
- **LONGBLOB**: é um tipo de dado que armazena dados binários de tamanho máximo de 4.294.967.295 bytes (4GB).



#### 4.1.4.4. Coleção

- **ENUM**: é um tipo de dado que permite selecionar um valor de uma lista predefinida. O ENUM pode armazenar até 65.535 valores diferentes, mas é importante lembrar que cada valor deve ser definido explicitamente. Os valores ENUM são armazenados internamente como números inteiros, associados com cada opção em ordem numérica a partir de 1.

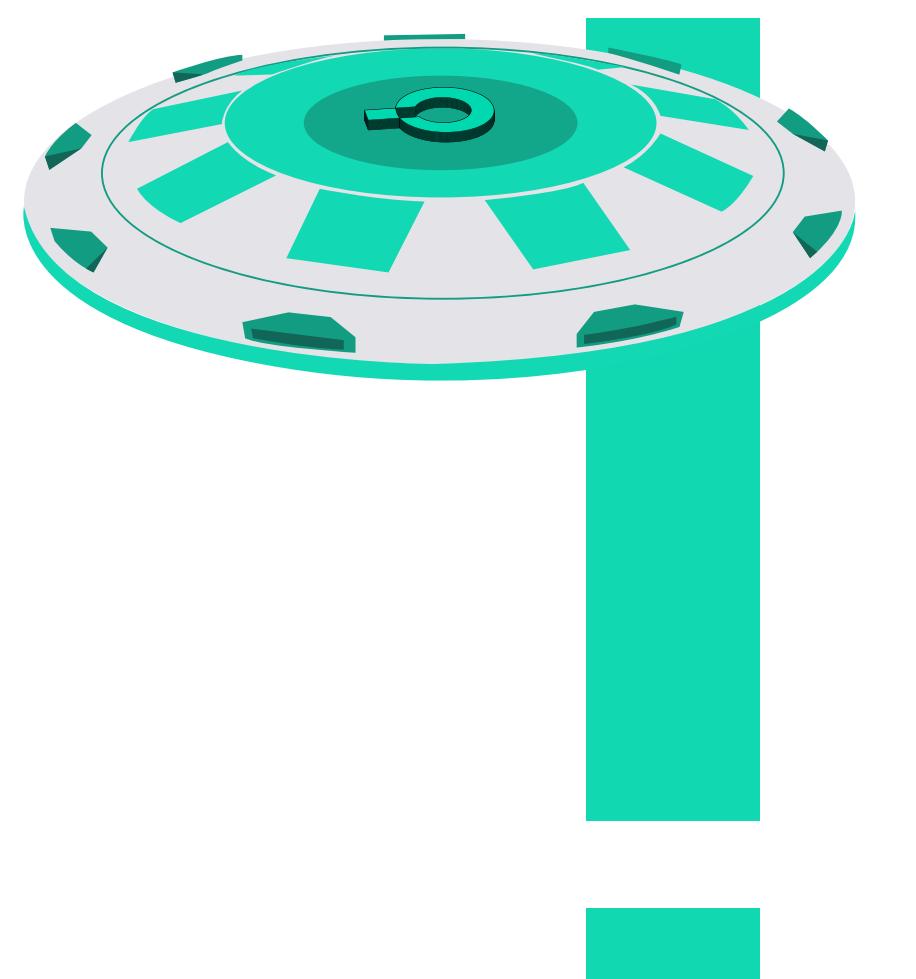
- **SET**: é um tipo de dado que permite selecionar múltiplos valores de uma lista predefinida. O SET pode armazenar até 64 valores diferentes, cada um definido explicitamente. Os valores do SET são armazenados como uma string de bits, em que cada bit representa um valor de opção. Por exemplo, um SET com as opções 'Opção 1', 'Opção 2' e 'Opção 3' pode ser representado pelos seguintes valores binários: 001, 010, 100, 011, 101, 110, 1.

#### 4.1.4.4. Espacial

- **GEOMETRY**: é um tipo de dados genérico que pode armazenar qualquer tipo de objeto espacial.
- **POINT**: é um tipo de dados que representa um ponto no espaço.
- **POLYGON**: é um tipo de dados que representa uma forma geométrica fechada, composta por uma sequência de pontos que definem sua borda.
- **MULTIPOLYGON**: é um tipo de dados que representa uma coleção de formas geométricas fechadas, compostas por uma sequência de pontos que definem suas bordas.

Esses tipos de dados podem ser usados para armazenar informações geográficas como coordenadas de latitude e longitude, limites de áreas geográficas, mapas, entre outras informações. A principal diferença entre eles está na forma como armazenam informações geográficas. Enquanto o POINT armazena informações de um único

ponto no espaço, o POLYGON armazena informações de uma forma geométrica fechada e o MULTIPOLYGON pode armazenar várias formas geométricas fechadas.



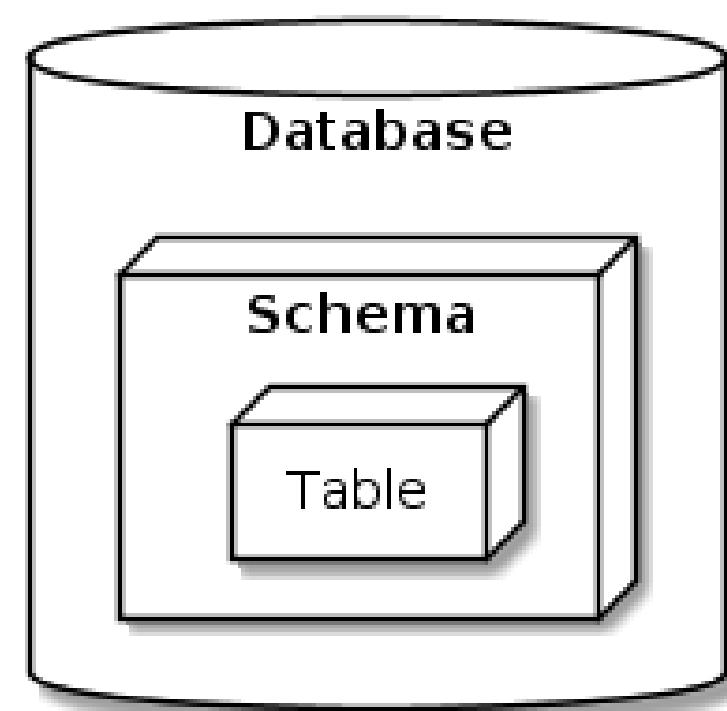
#### 4.2. Níveis do banco de dados SQL

É assim que funciona o nível hierárquico no banco de dados SQL, sendo:

**Database**: Banco de dados

**Schema**: Conjunto de Tabelas

**Table**: Tabela



## Mundo 5

### 5.1. Comandos dentro do SQL

Os comando dentro do SQL são separados por “ ; ”

Se você quiser rodar um comando é só clicar na seguinte sequência:

**Ctrl + Enter.**

Caso você queira rodar todos os comandos basta digitar **Ctrl + Shift + Enter.**

#### 5.1.1. Criando um schema

O **schema** é o lugar onde fica armazenado nossas tabelas. É importante que você separe os tipos de tabelas para certos bancos de dados. Por exemplo, não misture em um schema informações sobre as ações da B3 e os dados dos funcionários, tenha sempre tudo bem estruturado.

**CREATE DATABASE \_nome\_do\_schema**

#### 5.1.2. Excluindo um schema

**DROP DATABASE \_nome\_do\_schema**

#### 5.1.3 Definindo o schema a ser usado

**USE \_nome\_do\_schema**

#### 5.2. Criando uma tabela

Antes de criar uma tabela no schema, você precisa se certificar que está utilizando o schema correto. Então para isso, antes de qualquer comando, vá até o schema desejado utilizando o:

**USE \_nome\_do\_schema;**

Existem várias formas de criar uma tabela, por isso, vou te dar os componentes necessários para se criar uma:

**Primary Key:** É a chave de cada linha que certificará que não existem informações duplicadas dentro do seu banco de dados. Caso você insira informações com a Primary Key duplicada, gerará um erro.

Você só pode definir uma Primary Key por tabela.

**NOT NULL:** É uma propriedade do banco de dados que certificará que não entrarão linhas vazias dentro do seu banco de dados. Caso você insira alguma informação vazia, o banco de dados gerará um erro.

Você pode definir essa propriedade em quantas colunas quiser.

**AUTO\_INCREMENT:** É uma propriedade do banco de dados que adiciona automaticamente, valores inteiros, a cada linha. Isso é utilizado quando queremos um id numérico para nossas informações.

Cuidado: ao definir Auto\_Increment como Primary Key, você pode gerar colunas duplicadas sem você notar. Isso acontece porque o banco automaticamente irá gerar uma nova chave aleatória a cada input, mesmo que todas as outras colunas (que de fato são úteis) sejam iguais.

Você pode definir essa propriedade uma vez na tabela.

**DEFAULT:** É uma propriedade do banco de dados que define um valor para a coluna caso a informação seja nula, seria um valor padrão que todas as informações vão ter no primeiro momento.

Você pode definir essa propriedade em quantas colunas quiser.

**ENUM:** É uma propriedade do banco de dados que define quais valores poderão ser atribuídos para aquela coluna, caso um valor diferente seja atribuído retornará um erro.

Você pode definir essa propriedade em quantas colunas quiser.

**UNSIGNED:** É uma propriedade do banco de dados que impõe que os valores negativos dentro da coluna, caso um valor negativo seja atribuído retornará um erro.

Você pode definir essa propriedade em quantas colunas quiser.

Agora que você já sabe as principais propriedades dentro do banco de dados, podemos partir para a criação de tabelas. Nesta parte você vai utilizar, principalmente, conhecimentos do Mundo 4 onde falamos de tipos de dados.

### 5.2.1. Estrutura primária para se criar uma tabela:

```
CREATE TABLE nome_do_banco_de_dados.nome_da_tabela (
    nome_coluna1 tipo_coluna1(),
    nome_coluna2 tipo_coluna2(),
    nome_coluna3 tipo_coluna3()
);
```

Dentro do SQL, você pode especificar o nome do banco de dados antes do nome da tabela, para identificar de onde vem aquela tabela. Isso substitui o comando: **USE nome\_da\_tabela**

### 5.2.2. Exemplo de uma tabela sendo criada:

```
CREATE TABLE transactions (
    id INTEGER PRIMARY KEY,
    date DATE,
    type ENUM('compra','venda'),
    symbol VARCHAR(10),
    shares INTEGER UNSIGNED
    price DECIMAL(10,2),
    customer_id INTEGER
);
```

## Mundo 6

### 6.1. Inserindo dados na tabela

#### 6.1.1. Estrutura primária para inserir dados:

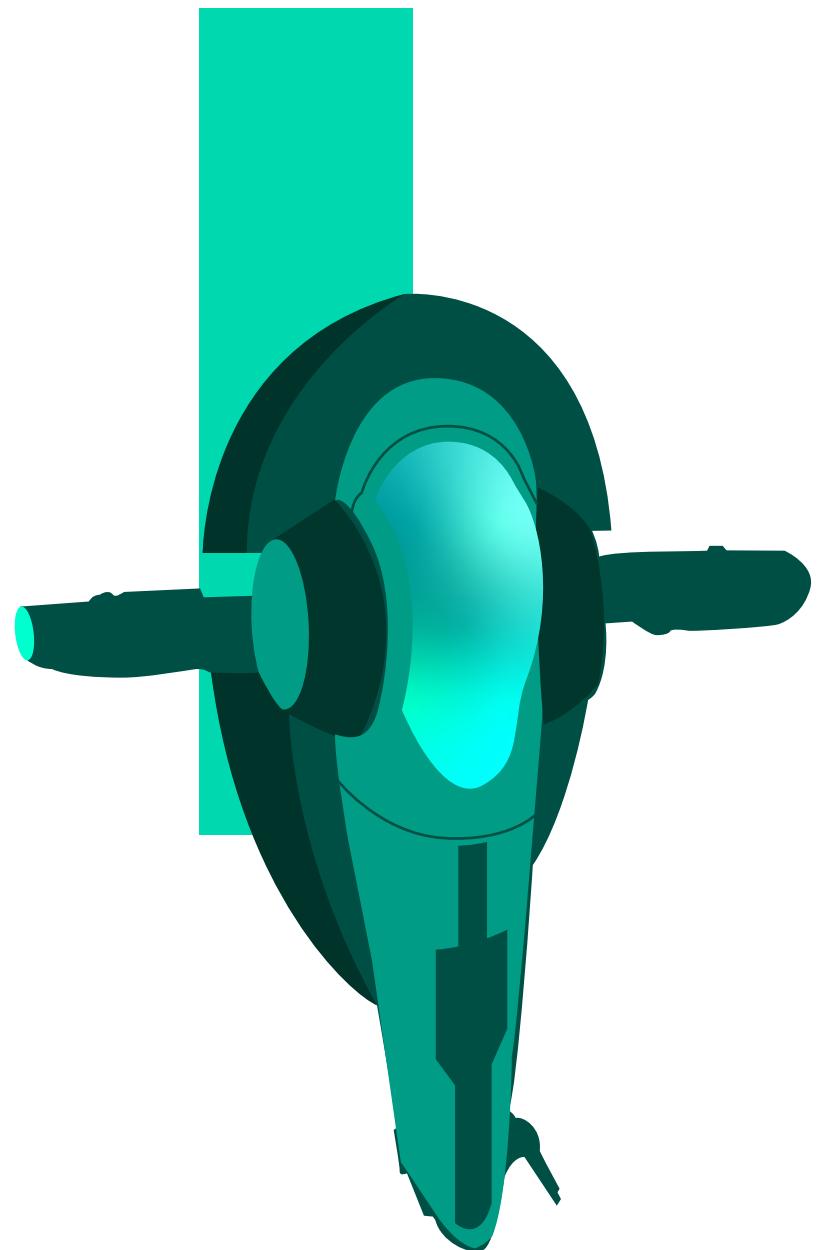
```
INSERT INTO nome_tabela (nome_coluna1,nome_coluna2...) VALUES ("valores1","valores2",...)
```

#### 6.1.2. Inserindo dados:

```
INSERT INTO codigopy.stocks (symbol, name, c  
sector, price, volume) VALUES ("AAPL", "Apple Inc.",  
"Tecnologia", "102.34", "123456")
```

#### 6.1.3. Inserindo múltiplos dados:

```
INSERT INTO codigopy.stocks (symbol, name, c  
sector, price, volume ) VALUES ("AAPL", "Apple Inc.",  
"Tecnologia", "102.34", "123456"),("GOOG", "Google  
LLC", "Tecnologia", "102.34", "123456"),("FB", "Fa-  
cebook Inc.", "Tecnologia", "102.34", "123456"),
```



## Mundo 7

### 7.1. Alterando a estrutura da tabela

Com esta propriedade mudaremos as propriedades de uma tabela que já existe.

#### 7.1.1. Adicionando uma coluna nova:

##### 7.1.1.1. Estrutura primária para adicionar uma coluna:

```
ALTER TABLE nome_tabela ADD COLUMN nome_coluna DATE AFTER nome_da_coluna_anterior;
```

##### 7.1.1.2. Adicionando uma coluna:

```
ALTER TABLE stocks ADD COLUMN price_date  
DATE AFTER symbol;
```

### 7.1.2. Deletando uma coluna:

#### 7.1.2.1. Estrutura primaria para deletar uma coluna:

```
ALTER TABLE nome_tabela DROP COLUMN nome_coluna;
```

#### 7.1.2.2. Deletando uma coluna:

```
ALTER TABLE stocks DROP COLUMN price_date;
```

### 7.1.3 Modificando o tipo da coluna:

#### 7.1.3.1 Estrutura primaria para modificar o tipo da coluna:

```
ALTER TABLE nome_tabela  
MODIFY COLUMN nome_coluna TIPO_COLUNA;
```

#### 7.1.3.1 Estrutura primaria para modificar o tipo da coluna:

```
ALTER TABLE nome_tabela  
MODIFY COLUMN nome_coluna VARCHAR(20);
```

### 7.1.4. Mudando o nome da coluna:

#### 7.1.4.1. Estrutura primaria para mudar o nome da coluna:

```
ALTER TABLE nome_tabela  
CHANGE COLUMN nome_antigo nome_novo  
NOVO_TIPO;
```

**7.1.4.2. Mudando o nome da coluna:**

```
ALTER TABLE stocks  
CHANGE COLUMN symbol simbolo  
VARCHAR(20);
```

**7.1.5.2 Mudando o nome da coluna:**

```
ALTER TABLE stocks  
RENAME TO aces;
```

**7.1.5. Mudando o nome da tabela:****7.1.5.1. Estrutura primaria para mudar o nome da tabela:**

```
ALTER TABLE nome_tabela  
RENAME TO nome_novo;
```

## Mundo 8

### 8.1. Manipulação de Linhas

Neste mundo aprenderemos como manipular linhas conforme as condições.

#### 8.1.1. Manipulando apenas um campo (chave primária):

##### 8.1.1.1. Estrutura primaria para manipular apenas um campo:

```
UPDATE nome_tabela  
SET coluna = valor_novo  
WHERE condição_de_troca
```

#### 8.1.1.2. Estrutura primaria para manipular apenas um campo:

```
UPDATE acoes  
SET sector = "Petroleo"  
WHERE id = 1;
```

#### 8.1.2. Manipulando apenas um campo em várias linhas:

Antes de manipular várias linha de uma vez, o SQL pede uma confirmação através do comando:

```
SET SQL_SAFE_UPDATES = 0;
```

### 8.1.2.1. Estrutura primaria para manipular apenas um campo em várias linhas:

```
SET SQL_SAFE_UPDATES = 0;  
  
UPDATE nome_tabela  
SET coluna = valor_novo  
WHERE condição_de_troca
```

### 8.1.2.2. Estrutura primaria para manipular apenas um campo em várias linhas:

```
SET SQL_SAFE_UPDATES = 0;  
  
UPDATE acoes  
SET sector = "Petroleo"  
WHERE symbol = "PETR4";
```

### 8.1.3. Manipulando vários campos em várias linhas:

Antes de manipular várias linha de uma vez, o SQL pede uma confirmação através do comando:

```
SET SQL_SAFE_UPDATES = 0;
```

### 8.1.3.1 Estrutura primaria para manipular vários campos em várias linhas:

```
SET SQL_SAFE_UPDATES = 0;  
  
UPDATE nome_tabela  
SET coluna = valor_novo, coluna2 =  
valor_novo2  
WHERE condição_de_troca
```

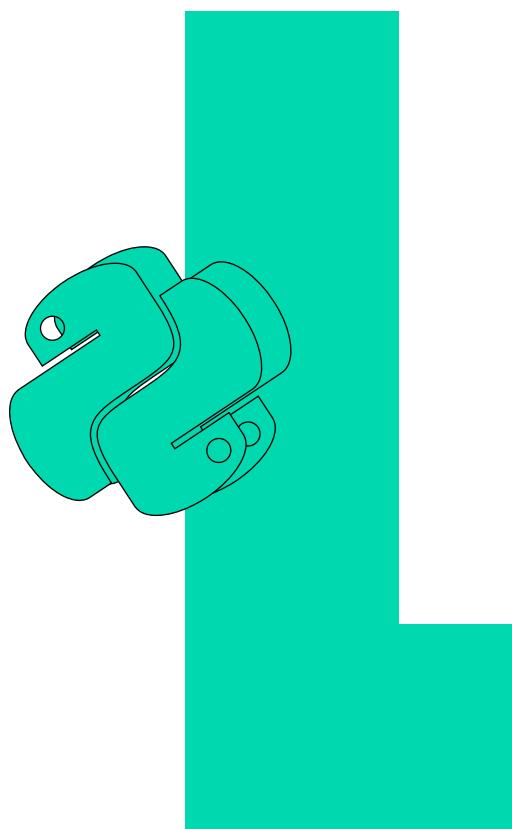
### 8.1.3.2. Estrutura primaria para manipular vários campos em várias linhas:

```
SET SQL_SAFE_UPDATES = 0;  
  
UPDATE acoes  
SET sector = "Petroleo", name = "Petrobras",  
WHERE symbol = "PETR4";
```

### 8.1.4. Manipulando vários campos em várias linhas:

Antes de manipular várias linha de uma vez, o SQL pede uma confirmação através do comando:

```
SET SQL_SAFE_UPDATES = 0;
```



### 8.1.4.1. Estrutura primaria para manipular vários campos em várias linhas:

```
SET SQL_SAFE_UPDATES = 0;  
  
UPDATE nome_tabela  
SET coluna = valor_novo, coluna2 = valor_novo2  
WHERE condição_de_troca
```

### 8.1.4.2. Estrutura primaria para manipular vários campos em várias linhas:

```
SET SQL_SAFE_UPDATES = 0;  
  
UPDATE acoes  
SET sector = "Petroleo", name = "Petrobras",  
WHERE symbol = "PETR4";
```

### 8.1.5. Deletando linhas com condição:

Antes de manipular várias linha de uma vez, o SQL pede uma confirmação através do comando:

```
SET SQL_SAFE_UPDATES = 0;
```

### 8.1.5.1. Estrutura primaria para manipular vários campos em várias linhas:

```
SET SQL_SAFE_UPDATES = 0;  
  
DELETE FROM nome_tabela  
WHERE condição_de_exclusão
```

### 8.1.5.2. Manipulação de vários campos em várias linhas:

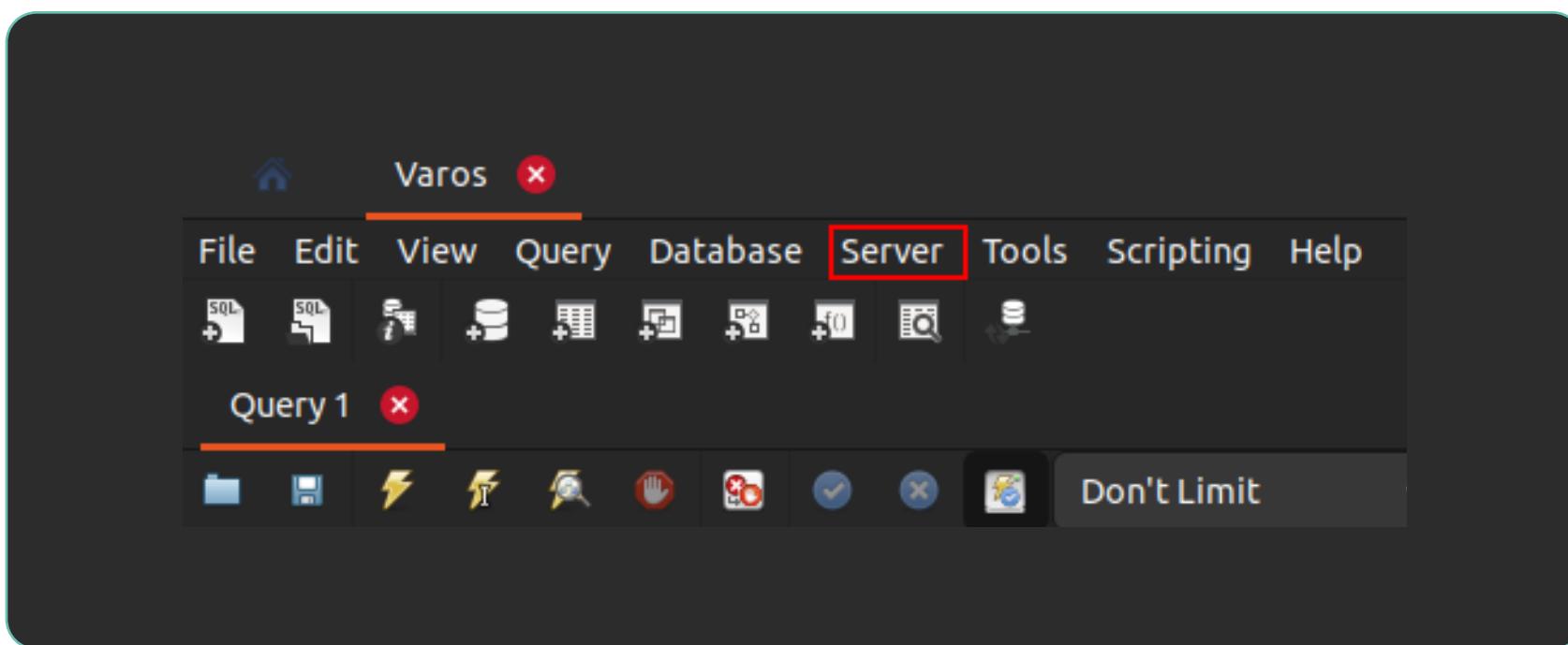
```
SET SQL_SAFE_UPDATES = 0;  
  
DELETE FROM acoes  
WHERE symbol = "PETR4";
```

# Mundo 9

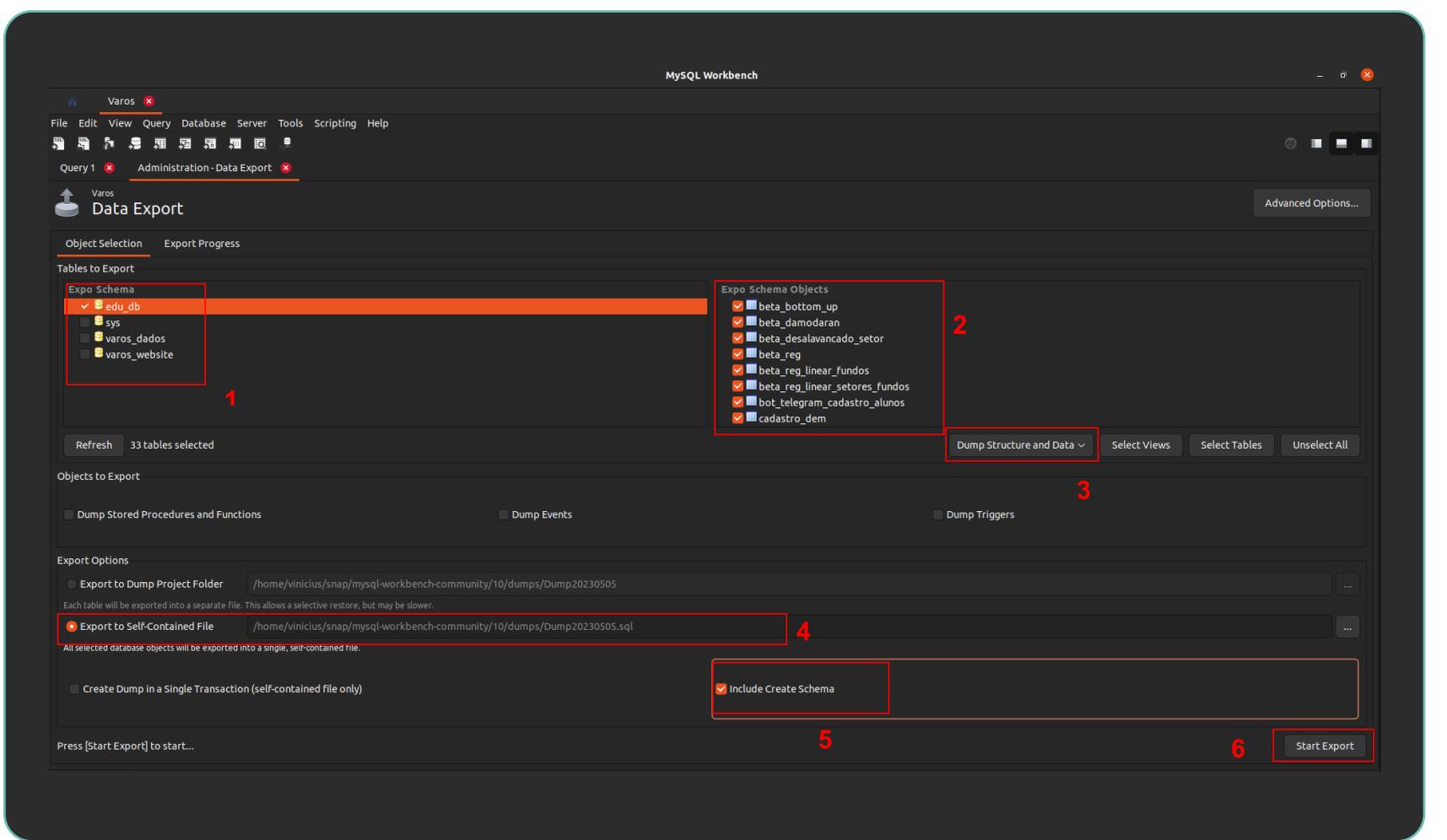
## 9.1. Gerando cópias de segurança:

Gerar backups é de extrema importância, principalmente quando você está trabalhando com um software que não permite retomar algum comando, ou seja, se você deletar uma tabela não poderá voltar atrás sem um backup.

1) Clique em "Server" e depois em "Data Export"



2) Siga os seguintes passos conforme a sequência numérica:



2.1) Primeiramente escolha o banco de dados que você quer exportar.

2.2) Depois escolha a tabela, ou tabelas, que deseja exportar.

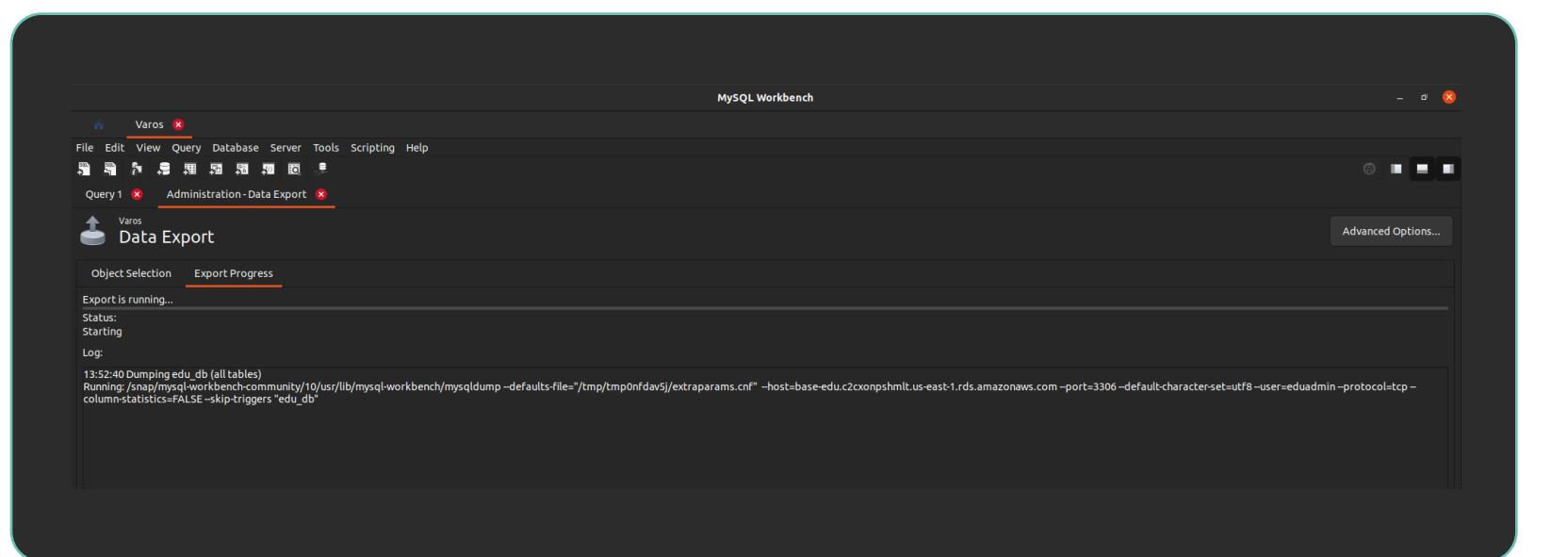
2.3) Confirme que a opção está "Dump Structure and Data". Você pode exportar também só a estrutura ou só os dados, no nosso caso exportamos os dois.

2.4) Selecione a opção onde o backup ficará salvo em um arquivo em um local escolhido por você.

2.5) Selecione para ele criar um banco de dados automaticamente na hora da importação dos dados.

2.6) Por fim, dê um start na exportação e espere.

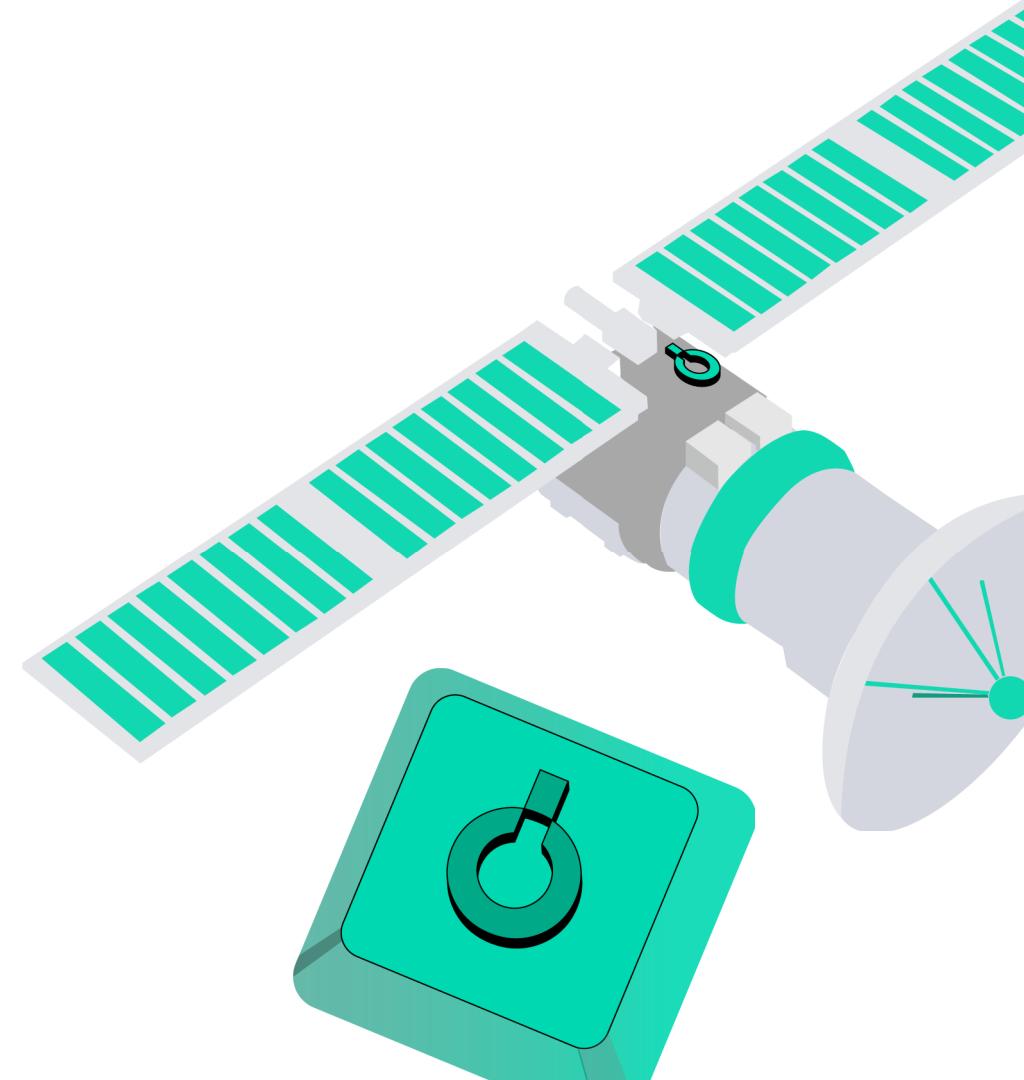
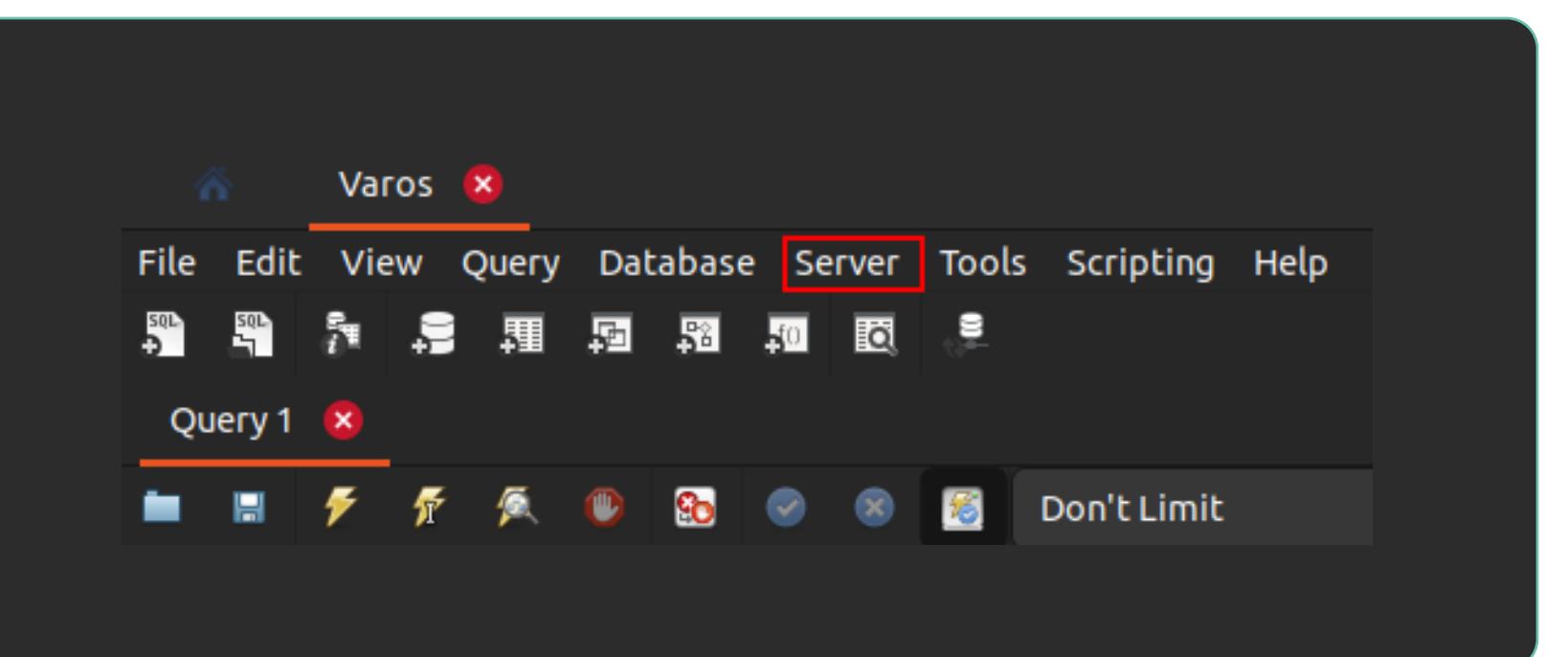
3) Espere a exportação ficar pronta.



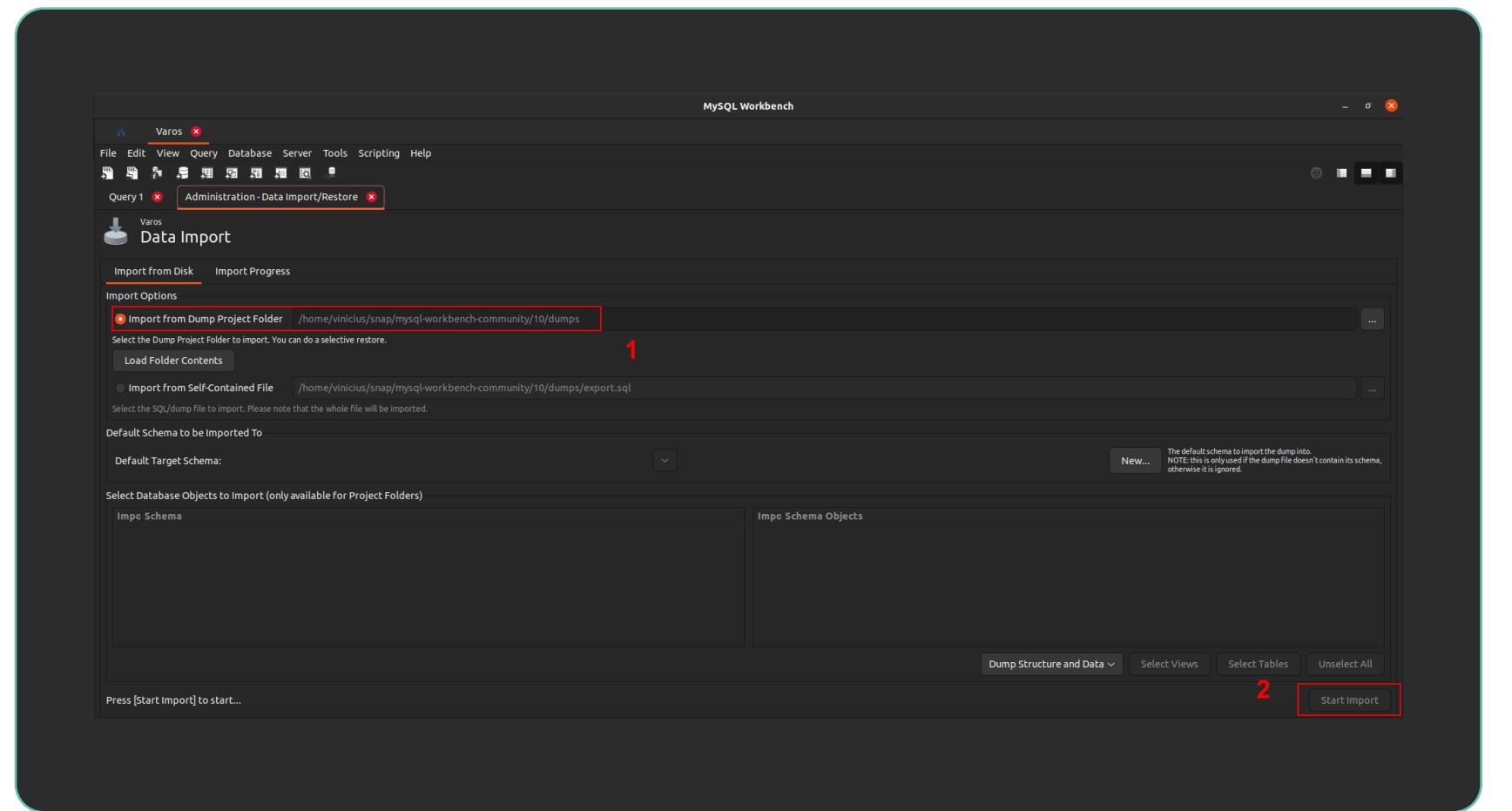
4) Depois é só ir onde salvou o arquivo.

## 9.2. Carregando cópias de segurança:

1) Clique em "Server" e depois em "Data Import".



2) Escolha o local do arquivo e dê start na importação



## Mundo 10

### 10.1. Onde achar Data sets:

Acesse o <https://datasetsearch.research.google.com/> e procure pelo Data set que você deseja, nesta aula procuremos sobre: "Mercado Brasileiro Ações".

### 10.2. FOREIGN KEY

Foreign Key, que pode ser lida como chave estrangeira, é a propriedade que estabelece o relacionamento entre duas tabelas. Deve ser estabelecida durante a criação da tabela:

É uma propriedade que serve para proteger seus dados, evitando que eles fiquem inconsistentes.

```
CREATE TABLE nome_do_banco_de_dados.  
nome_da_tabela (  
  
    nome_coluna1 tipo_coluna1(),  
    nome_coluna2 tipo_coluna2(),  
    nome_coluna3 tipo_coluna3(),  
    FOREIGN KEY (coluna1) REFERENCES tabela_referencia(coluna2)  
  
);
```

### 10.3. CONSTRAINT

CONSTRAINT é a propriedade que define o nome de uma Primary key. Deve ser estabelecida durante a criação da tabela:

Essa propriedade serve para identificação da sua Primary Key.

```
CREATE TABLE nome_do_banco_de_dados.  
nome_da_tabela (  
  
    nome_coluna1 tipo_coluna1(),  
    nome_coluna2 tipo_coluna2(),  
    nome_coluna3 tipo_coluna3(),  
    CONSTRAINT (nome_PK) PRIMARY KEY(nome_coluna_PK)  
  
);
```

### 10.3.1. CONSTRAINT com duas Primary Key

Você pode também definir duas colunas como sendo uma Primary Key, isso é importante, por exemplo, em cotações. Porque você só pode ter um dado de fechamento diário por empresa, então você poderia definir sua chave primária como sendo o ticker e a data.

Em resumo, o conjunto das suas colunas deve dar um valor único.

```
CREATE TABLE nome_do_banco_de_dados.  
nome_da_tabela(  
  
    nome_coluna1 tipo_coluna1(),  
    nome_coluna2 tipo_coluna2(),  
    nome_coluna3 tipo_coluna3(),  
    CONSTRAINT (nome_PK) PRIMARY KEY(nome_  
        coluna_PK,nome_coluna2_PK)  
  
);
```

### 10.4. Conectando o SQL ao Python

Para fazer uma integração direta com Python e SQL você precisa instalar os seguintes pacotes através dos comandos:

```
pip install pymysql  
pip install sqlalchemy  
pip install mysql-connector-python
```

Após ter feito isso, você pode definir suas variáveis que serão importantes na hora de efetuar a integração:

Onde:

Senha: É a senha definida na instalação do SQL.

Banco\_de\_dados: O nome do banco de dados a ser utilizado.

Obs: O restante das informações não precisam ser modificadas.

```
user = "root"
senha = "1234"
banco_de_dados = "mercado_br"
host = "localhost"
porta = 3306

conexao = create_engine(url=f"mysql+pymysql://{user}:{senha}@{host}:{porta}/{banco_de_dados}")
```

É através do comando “create\_engine()” que será feita a integração, sendo:

#### Parâmetros:

**url:** **dialect+driver://username:password@host:port/database**

#### encoding:

Esse parâmetro representa o tipo de codificação utilizado pelo banco de dados.

Esse parâmetro é **opcional**. Pode receber a identificação na formatação de uma [string](#).

#### echo:

Esse parâmetro determina se as instruções SQL serão exibidas no console. Por padrão, o sqlalchemy define “echo = False”.

Esse parâmetro é **opcional** e recebe um booleano: [True](#) ou [False](#).

#### url:

Esse parâmetro é obrigatório e utiliza uma url pré-definida com parâmetros das configurações. É importante notar que cada banco de dados possui uma url diferente. Segue a sintaxe da url:

**url: dialect+driver://username:password@host:port/database**

Onde:

#### dialect:

É o nome do serviço utilizado como gerenciamento de banco de dados. Exemplos: **mysql, oracle, postgresql, etc..**

**driver:**

É o nome da ignição criada para integrar o Python com o gerenciador de banco de dados. No nosso caso, como utilizamos o mysql, poderia ser tanto o **pymysql** ou o **mysqlDb**, que pode ser melhor utilizado dependendo da forma que você vai trabalhar.

**username:**

É o nome do usuário do gerenciador do banco de dados. Por padrão, quando se utiliza o local host, os bancos de dados definem que será **root**.

**password:**

É a senha definida para acessar o gerenciador de banco de dados.

**host:**

É o local onde está armazenado seu banco de dados que, no seu caso será, poderá ser o **localhost**, dentro do computador. Entretanto, poderia ser um serviço na nuvem como AWS. Neste caso seria necessário passar a url do banco de dados.

**port:**

O número da porta é a forma como os bancos de dados se comunicam com o servidor. Ele determina qual programa de banco de dados está sendo usado e quais são as configurações do servidor. Geralmente, um banco de dados utiliza a porta padrão 3306.

**database:**

É o nome do banco de dados localizado dentro do gerenciador de banco de dados.

Esse parâmetro é **obrigatório**. Pode receber a url no formato [string](#).

## 10.5. Jogando informações para o SQL

Para relembrar, peguei um exemplo do nosso mundo sobre Pandas:

```
DataFrame.to_sql(name, con, schema = None, if_exists = "fail",
index = True, index_label = None, chunksize = None, dtype =
None, method = None)
```

Esse método é utilizado para enviar informações ao SQL. Caso não exista a tabela dentro do gerenciador, ele criará uma nova. Esse método tem que ser utilizado em conjunto com a conexão ao SQL.

### Parâmetros:

Só recebe os parâmetros name e con como obrigatórios.

#### name:

Esse parâmetro vai definir o nome da tabela para onde serão enviadas as informações. Caso não exista a tabela, ele criará uma nova.

Esse parâmetro é **obrigatório**. Recebe o nome da tabela no formato string.

#### con:

Esse parâmetro vai definir qual será o conector utilizado para integrar o pandas ao sql.

Esse parâmetro é **obrigatório**. Recebe a variável na qual o conector foi atribuído.

#### schema:

Esse parâmetro vai definir para qual banco de dados a tabela vai ser enviada. Por padrão, o pandas define “schema = None”, ou seja, o pandas vai definir um banco de dados padrão.

Esse parâmetro é **opcional**. Recebe o nome do banco de dados no formato `string`.

**if\_exists:**

Esse parâmetro vai definir o que fazer quando a tabela enviada já existir. Por padrão, o pandas define “`if_exists = ‘fail’`”, ou seja, o envio retornará um erro se a tabela já existir.

Esse parâmetro é **opcional**. Recebe os comandos já pré-definidos no formato `string`. Os formatos são:

`fail` = retornará um erro caso o nome da tabela já exista.

`replace` = substitui a tabela caso o nome da tabela já exista

`append` = adiciona informações à tabela caso o nome da tabela já exista

**index:**

Esse parâmetro vai definir se o index entrará na tabela como uma coluna ou não. Por padrão, o pandas define o “`index = True`”, então ele adiciona o index como uma coluna dentro do SQL.

\* Esse parâmetro só pode ser usado em caso de uma nova tabela ou substituição de uma tabela antiga.

Esse parâmetro é **opcional** e recebe um booleano: `True` ou `False`.

**index\_label:**

Esse parâmetro é utilizado em conjunto com o parâmetro “`index`”. Ele que vai definir o nome da nova coluna que antes era um index. Por padrão, o pandas define o nome da nova coluna como o mesmo nome do index.

Esse parâmetro é **opcional**. Recebe o nome da nova coluna no formato `string`.

**chuncksize:**

Esse parâmetro vai definir quantas linhas serão adicionadas ao banco de dados a cada vez. Por padrão, o pandas define "chunksize = 1", ou seja, ele vai adicionar uma linha de cada vez no gerenciador de banco de dados.

Esse parâmetro é **opcional**. Recebe o valor da quantidade de linhas em [integer](#).

**dtype:**

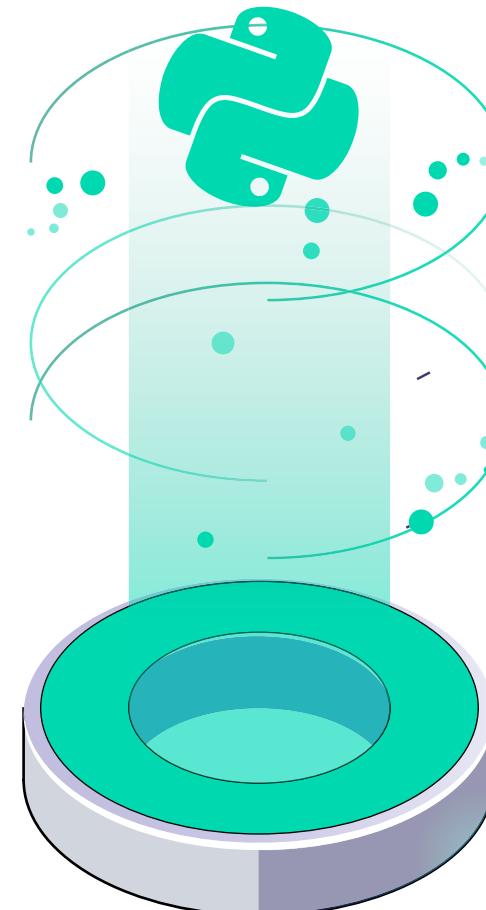
Esse parâmetro vai definir o tipo de cada coluna. Por padrão, o pandas define o tipo [integer](#) para números e o tipo [text](#) para os outros.

Esse parâmetro é **opcional**. Recebe os nomes das colunas e o tipo das colunas no formato [string](#) dentro de um dicionário.

**method:**

O parâmetro "method" na função `to_sql` do pandas determina o método usado para escrever os dados no banco de dados. Os valores possíveis são 'multi' para executar várias instruções SQL em paralelo, 'infer' para tentar inferir o melhor método a partir do tipo de dados e '[None](#)' para usar o método padrão.

Esse parâmetro é **opcional**. Recebe o método utilizado no formato [string](#).



## Mundo 11

### 11.1. Efetuando consultas utilizando o SELECT:

Aprenderemos como selecionar as informações de uma tabela dentro do SQL.

#### 11.1.1. Selecionando todas as informações da tabela:

O \* indica que estamos selecionando todas as colunas.

```
SELECT * FROM nome_tabela;
```

#### 11.1.2. Selecionando colunas específicas:

Escolha apenas as colunas que desejar.

```
SELECT nome_coluna, nome_coluna2  
FROM nome_tabela;
```

#### 11.1.3. Selecionando todas as informações da tabela e ordenando:

Utilizaremos o comando ORDER BY em conjunto com o DESC que indica que será organizado da forma Descendente.

[ASC | DESC] é a forma que seu conjunto de dados será organizado, seja descendente ou ascendente, ou seja, do menor para o maior ou do maior para o menor respectivamente.

```
SELECT * FROM nome_tabela ORDER BY  
nome_coluna DESC;
```

## Mundo 12

### 12.1. Efetuando consultas condicionais utilizando o SELECT e o WHERE:

Aprenderemos como selecionar as informações de uma tabela dentro do SQL a partir das condições que desejamos.

#### 12.1.1. Selecionando todas as informações da tabela com uma condição e organizando:

```
SELECT * FROM nome_tabela  
WHERE nome_coluna = condição  
ORDER BY nome_coluna DESC;
```

#### 12.2.1. Selecionando todas as informações da tabela com duas condição ao mesmo tempo:

```
SELECT * FROM nome_tabela  
WHERE nome_coluna = condição AND  
nome_coluna = condição;
```

#### 12.3.1. Selecionando todas as informações da tabela com uma condição ou outra:

```
SELECT * FROM nome_tabela  
WHERE nome_coluna = condição OR  
nome_coluna = condição;
```

### 12.4.1. Exemplo utilizando mais de uma condição WHERE, AND e OR:

Neste caso estamos selecionando todas as colunas que tem as linhas (dayWeekAbbrevTime = "SEG") ou (dayWeekAbbrevTime = "SAB") e que possuem na linha coincidente, mas em colunas diferentes os valores de (yeartime = 2001) ou (yeartime = 2000)

```
SELECT * FROM descricao_tempo  
WHERE (dayWeekAbbrevTime = "SEG" OR dayWe-  
ekAbbrevTime = "SAB") AND (yeartime = 2000 OR  
yeartime = 2001);
```

### 12.5.1. Selecionando dados que estão em uma lista:

Neste caso damos uma lista com opções que a nossa condição deve respeitar.

```
SELECT * FROM nome_tabela  
WHERE nome_coluna IN (condicao1,condicao2);
```

Ou que não deve respeitar:

```
SELECT * FROM nome_tabela  
WHERE nome_coluna NOT IN (condicao1,-  
condicao2);
```

### 12.6.1. Selecionando dados que estão dentro de um intervalo:

```
SELECT * FROM nome_tabela  
WHERE nome_coluna BETWEEN condicao1  
AND condicao2;
```

### 12.7.1. Selecionando valores únicos:

```
SELECT DISTINCT * FROM nome_tabela  
WHERE nome_coluna condicao;
```

### 12.8.1 Selecionando valores como outros:

```
SELECT DISTINCT * FROM nome_tabela  
WHERE nome_coluna LIKE condicao;
```

### 12.8.1. Selecionando valores como outros utilizando %:

O operador "%" também pode ser usado como um curinga (wildcard) em consultas SQL para efetuar correspondência de padrões em valores de texto. Por exemplo, a consulta abaixo retorna todos os registros onde o campo "nome" começa com a letra "J":

```
SELECT * FROM nome_tabela  
WHERE nome_coluna LIKE "J%";
```

Nesse caso, o "%" significa que qualquer sequência de caracteres pode seguir a letra "J".

## Mundo 13

### 13.1. Juntando tabelas com o JOIN :

Aprenderemos a juntar tabelas a partir das chaves estrangeiras.

#### 13.1.1. Juntando tabelas:

```
SELECT * FROM nome_tabela  
JOIN nome_tabela2;
```

#### 13.1.3. Selecionando colunas ao juntar tabelas:

```
SELECT nome_tabela1.nome_coluna,  
nome_tabela2.nome_coluna FROM nome_  
tabela JOIN nome_tabela2;
```

#### 13.1.4. Selecionando colunas ao juntar tabelas e abreviando os nomes:

O objetivo de abreviar as tabelas é a facilitação da visualização dos dados, por isso abrevie para nomes pequenos como: "dc", "c" etc.

```
SELECT abreviacao1.nome_coluna, abreviacao2.  
nome_coluna FROM nome_tabela AS abreviacao1  
JOIN nome_tabela2 AS abreviacao2;
```

#### 13.1.5. Selecionando colunas ao juntar tabelas e especificando as colunas:

Neste caso você pode escolher quais colunas você deseja que sejam comparadas na hora do JOIN.

```
SELECT abreviacao1.nome_coluna, abreviacao2.  
nome_coluna FROM nome_tabela AS abreviacao1  
JOIN nome_tabela2 AS abreviacao2 ON abrevia-  
cao1.nome_coluna = abreviacao2.nome_coluna;
```

**13.1.6. Selecionando colunas com condicional ao juntar tabelas e especificando as colunas:**

```
SELECT abreviacao1.nome_coluna, abreviacao2.  
nome_coluna FROM nome_tabela AS abreviacao1  
JOIN nome_tabela2 AS abreviacao2 ON abrevia-  
cao1.nome_coluna = abreviacao2.nome_coluna  
WHERE abreviacao1.nome_coluna = condicao;
```

**13.1.7. Selecionando colunas ao juntar três tabelas:**

```
SELECT abreviacao1.nome_coluna, abreviacao2.  
nome_coluna FROM nome_tabela AS abreviacao1  
INNER JOIN nome_tabela2 AS abreviacao2 ON abre-  
viacao1.nome_coluna = abreviacao2.nome_coluna  
INNER JOIN nome_tabela3 AS abreviacao3 ON abre-  
viacao1.nome_coluna = abreviacao3.nome_coluna;
```



### 13.1.8. Exemplo juntando três tabelas:

Neste exemplo estamos juntando as três tabelas com o objetivo final de gerar uma tabela com a cotação diária do dólar.

```
SELECT dt.datetime, c.nameCoin, dc.valueCoin FROM dados_cambio AS dc
INNER JOIN cadastro_moedas AS c ON c.keyCoin = dc.keyCoin
INNER JOIN descricao_tempo AS dt ON dt.keyTime = dc.keyTime
WHERE c.nameCoin = "Dolar" AND dt.datetime BETWEEN 2010
AND 2014;
```

### 13.1.9. DIFERENÇA ENTRE RIGHT, LEFT e INNER JOIN:

**LEFT JOIN:** é uma junção que retorna todos os registros da tabela à esquerda (a tabela que aparece antes da palavra JOIN) e os registros correspondentes da tabela à direita (a tabela que aparece após a palavra JOIN). Se não houver correspondência na tabela à direita, a coluna correspondente será preenchida com NULL.

**RIGHT JOIN:** é uma junção que retorna todos os registros da tabela à direita e os registros correspondentes da tabela à esquerda. Se não houver correspondência na tabela à esquerda, a coluna correspondente será preenchida com NULL.

**INNER JOIN:** é uma junção que retorna apenas os registros que têm correspondência nas duas tabelas. Ou seja, ela retorna apenas os registros onde a chave primária de uma tabela corresponde à chave estrangeira de outra tabela.

## Mundo 14

### 14.1. Puxando tabelas do SQL:

Para isso, utilizaremos um método do pandas.

#### 14.1.1. DataFrame.read\_sql(sql, con):

Esse método é utilizado para puxar informações do SQL. Esse método tem que ser utilizado em conjunto com a conexão ao SQL.

##### Parâmetros:

Só recebe os parâmetros name e con como obrigatórios.

##### sql:

Esse parâmetro definirá o comando do SQL a ser utilizado.

Esse parâmetro é **obrigatório**. Recebe o comando do SQL no formato [string](#).

##### con:

Esse parâmetro vai definir qual será o conector utilizado para integrar o pandas ao sql.

Esse parâmetro é **obrigatório**. Recebe a variável na qual o conector foi atribuído.

**14.1.2. Exemplo prático de como puxar uma tabela do SQL:**

```
from sqlalchemy import create_engine
import pandas as pd

user = "root"
senha = "1234"
banco_de_dados = "mercado_br"
host = "localhost"
porta = 3306

conexao = create_engine(url=f"mysql+pymysql://{user}:{senha}@{host}:{porta}/{banco_de_dados}")

dataframe = pd.read_sql('''SELECT * from cadastro_empresas;''', con=conexao)
```

