



# Plataformas de aplica  es Web

## Aula 09 - PetTopStore - Admin - Parte 1



Material Did tico do Instituto Metr pole Digital - IMD

### Termo de uso

Os materiais did ticos aqui disponibilizados est o licenciados atrav s de Creative Commons **Atribui  o-SemDeriva  es-SemDerivados CC BY-NC-ND**. Voc  possui a permiss o para realizar o download e compartilhar, desde que atribua os cr ditos do autor. N o poder  alter -los e nem utiliza-los para fins comerciais.

Atribui  o-SemDeriva  es-SemDerivados

CC BY-NC-ND



<https://creativecommons.org/licenses/by-nc-nd/4.0/>

# Apresentação

Nessa aula vamos iniciar a apresentação do sistema administrativo que usaremos como base para a criação da nossa loja virtual chamada PetTopStore.

Esse projeto que iremos iniciar será uma loja virtual(Pet Shop) chamado Pet Top Store e se trata de uma solução composta pelas seguintes funcionalidades:

- Interface administrativa: Onde o administrador do sistema poderá gerenciar produtos e visualizar relatórios de vendas. Também contempla uma API de acesso seguro aos dados para os outros módulos.
- PDV: Ponto de venda da loja física onde o vendedor poderá cadastrar clientes e realizar novas vendas para ele.
- Loja on-line: Um web site de vendas online onde os clientes podem visualizar e adquirir os produtos da loja

Iremos desenvolver o projeto de forma modular, com plataformas e tecnologias diferentes para cada módulo. Será apresentado nesta aula o sistema administrativo onde os administradores da loja podem acessar o cadastro de produtos e dados de vendas, assim como criar funcionários para a loja. Esse módulo irá contemplar também uma API que será utilizada pelos outros módulos, mas isso não será visto nesta aula ainda.

## Tecnologias do módulo administrativo

O sistema administrativo será criado utilizando:

- Node.js+Express como plataforma back-end
- MySQL como banco de dados
- knex.js como biblioteca de acesso ao banco de dados

O sistema administrativo conterà as funcionalidades mais importantes inicialmente, mas poderá ser incrementada à medida que o projeto vá avançando.

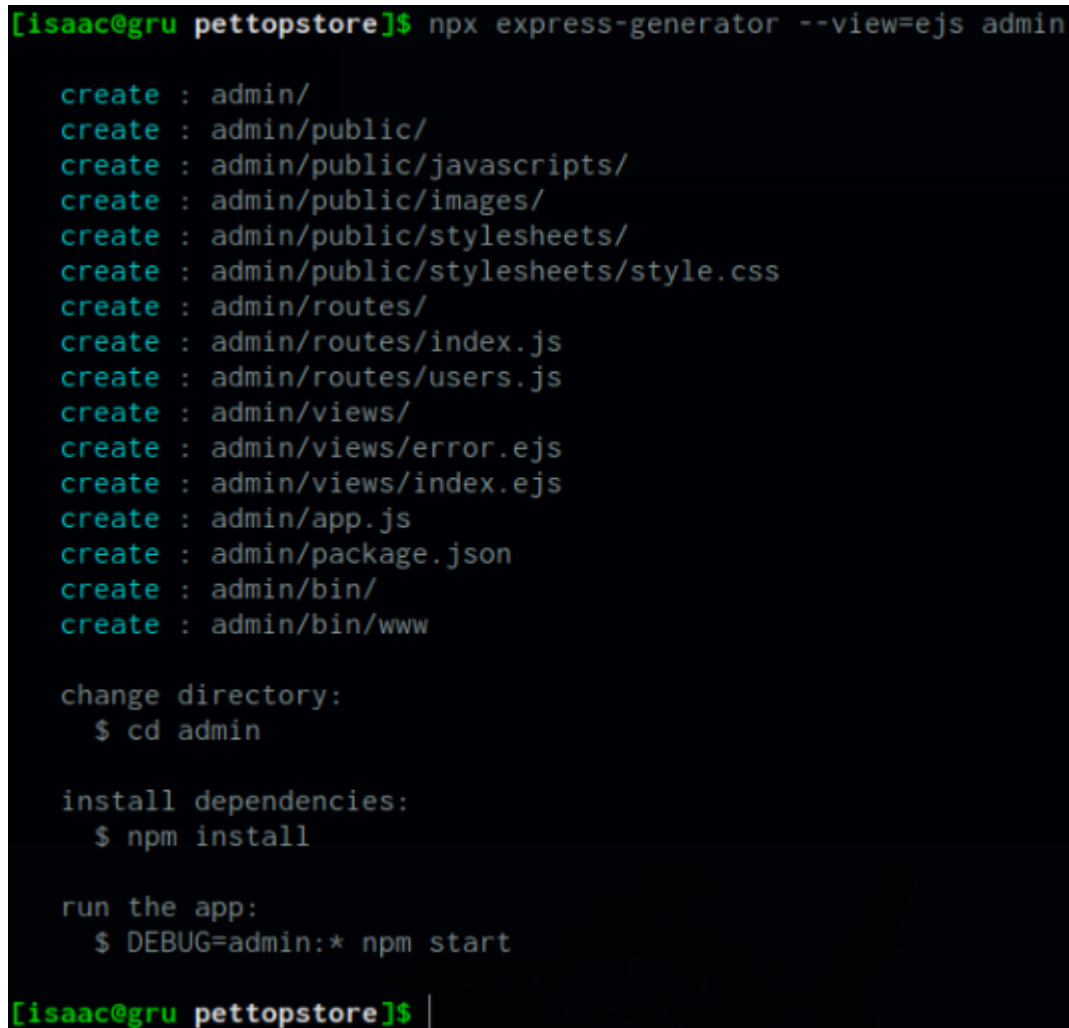
## Criação da aplicação

Inicialmente crie uma pasta principal para os módulos da aplicação, chamada pettopstore:

```
mkdir pettopstore  
cd pettopstore
```

A aplicação Express será criada com utilizando a ferramenta Express generator com o comando:

```
npx express-generator --view=ejs admin
```



```
[isaac@gru pettopstore]$ npx express-generator --view=ejs admin

  create : admin/
  create : admin/public/
  create : admin/public/javascripts/
  create : admin/public/images/
  create : admin/public/stylesheets/
  create : admin/public/stylesheets/style.css
  create : admin/routes/
  create : admin/routes/index.js
  create : admin/routes/users.js
  create : admin/views/
  create : admin/views/error.ejs
  create : admin/views/index.ejs
  create : admin/app.js
  create : admin/package.json
  create : admin/bin/
  create : admin/bin/www

change directory:
  $ cd admin

install dependencies:
  $ npm install

run the app:
  $ DEBUG=admin:* npm start

[isaac@gru pettopstore]$ |
```

Foi criado o esqueleto da aplicação, como visto na imagem acima com o express generator passando a opção do EJS como View Engine e com o nome "admin" para o novo projeto criado. A partir de agora trabalharemos nessa pasta "admin" criada.

Entre na pasta "admin", instale as dependências e execute a aplicação:

```
cd admin
npm install
npm start
```

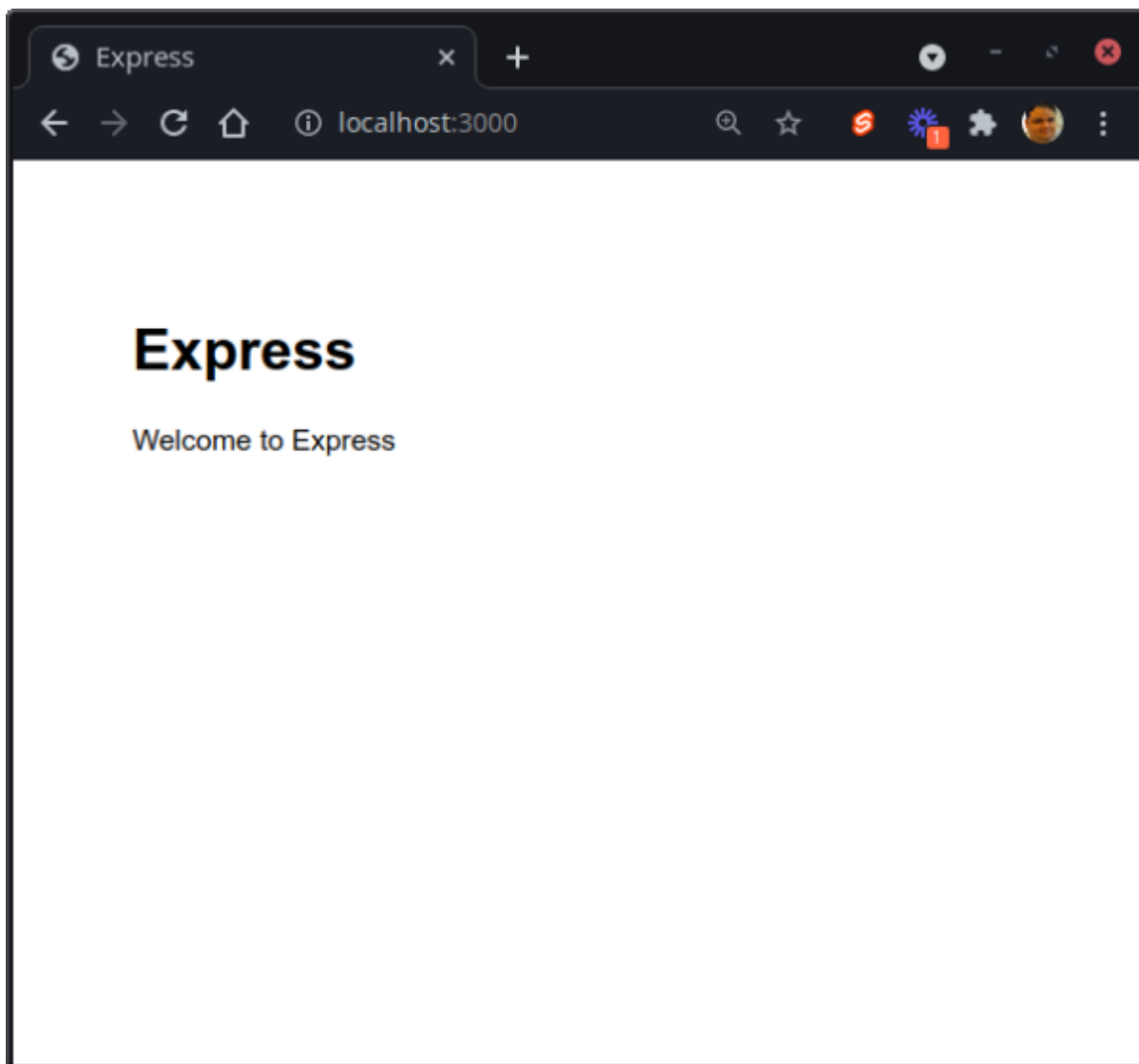
```
[isaac@gru pettopstore]$ cd admin
[isaac@gru admin]$ npm install

added 54 packages, and audited 55 packages in 4s

found 0 vulnerabilities
[isaac@gru admin]$ npm start

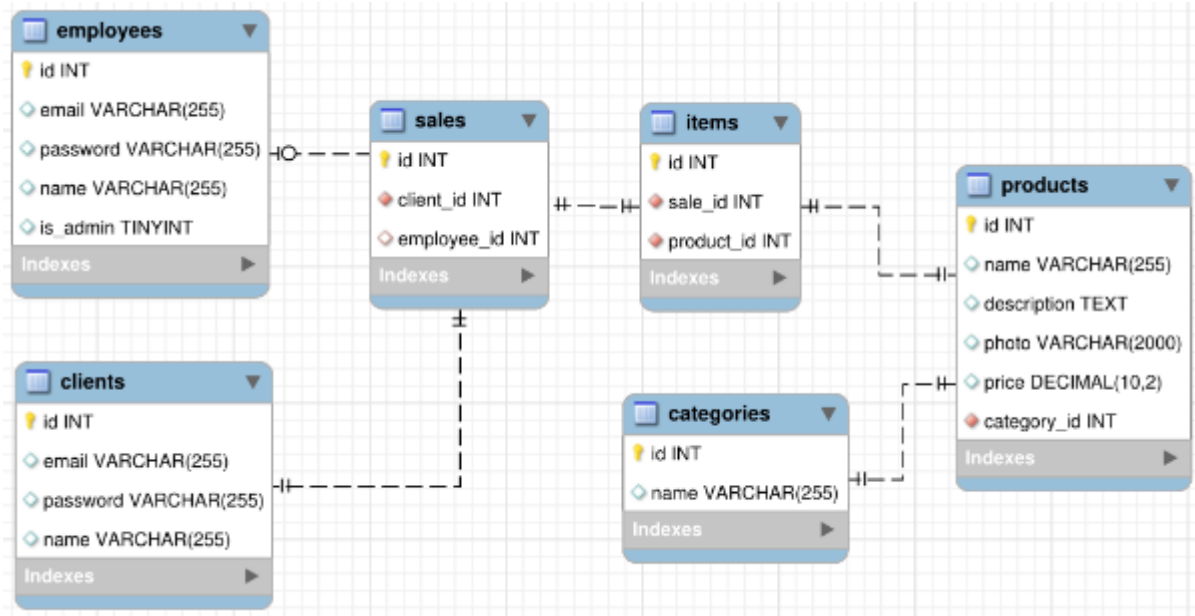
> admin@0.0.0 start
> node ./bin/www
```

Acesse <http://localhost:3000> no navegador e você deverá ver o seguinte resultado:



## Modelo do banco de dados

Para o sistema, iremos utilizar o seguinte modelo de banco de dados:



As tabelas que iremos criar são:

- employees
- clients
- products
- categories
- sales
- items

A tabela **employees** representa os funcionários da loja com os campos email, password, name e um booleano chamado *is\_admin* que determina se esse funcionário tem ou não acesso à área administrativa do sistema.

A tabela **clients** guarda a lista dos clientes da empresa, cadastrados pelo site ou por um funcionário que está operando o PDV (Ponto de Venda) da loja física.

A tabela **products** tem a lista de produtos da loja virtual com o nome, descrição, caminho para uma foto, preço e uma relação com a categoria desse produto pela chave *category\_id* que aponta para o id da tabela **categories**.

A tabela **categories** armazena as categorias dos produtos e é utilizada no seu cadastro e também na loja virtual para apresentar o menu de produtos separado por categorias.

A tabela **sales** tem as vendas realizadas no site ou na loja física. Ela guarda somente o *client\_id* e o *employee\_id* (campo opcional pois em uma compra pelo site ele não é preenchido). Os itens (ids dos produtos) de uma venda(sales) serão armazenados em uma tabela que relaciona products e sales, chamada **items**.

A tabela **items** relaciona **products** e **sales** (N-N) armazenando dois ids (*sale\_id* e *product\_id*) e com ela conseguimos saber que produtos estão em cada venda.

Com o MySQL (ou MariaDB) já instalado na sua máquina, acesse o servidor e crie um banco de dados novo para a aplicação (somente o banco, sem as tabelas por enquanto):

```
[isaac@gru admin]$ mysql -uroot -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 12
Server version: 10.6.3-MariaDB Arch Linux

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database pettopstore;
Query OK, 1 row affected (0.003 sec)

MariaDB [(none)]> exit
Bye
[isaac@gru admin]$ |
```

Na imagem acima conectamos no banco de dados pelo Linux com o cliente "mysql" no modo texto usando o usuário *root* e com uma senha criada na instalação. Você pode utilizar a ferramenta/usuário/senha que desejar para realizar a mesma tarefa. O comando "create database pettopstore;" cria um novo banco de dados chamado "pettopstore".

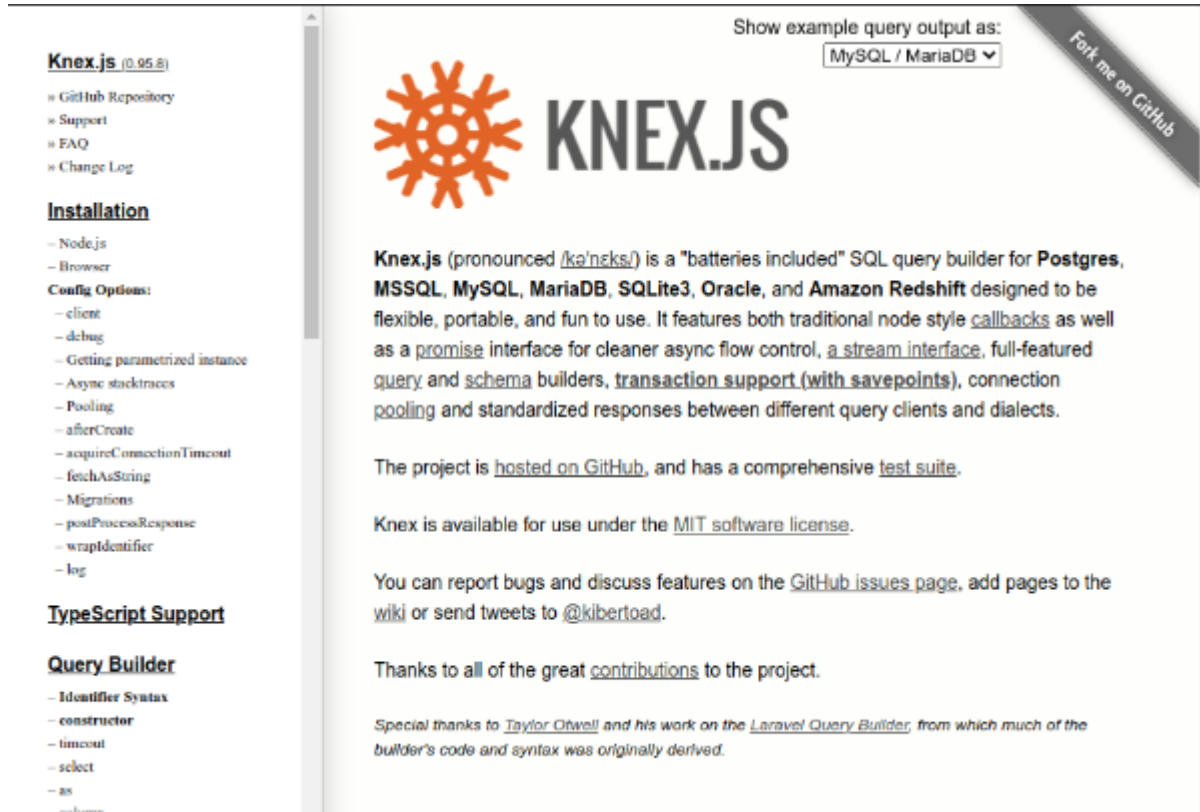
Não precisa criar as tabelas agora pois vamos utilizar outra ferramenta para isso, o knex.js.

## Knex.js

Web site: <https://knexjs.org/>

O knex.js é uma ferramenta de acesso a banco de dados para Node.js que suporta os seguintes bancos:

- Postgres
- MSSQL
- MySQL
- MariaDB
- SQLite3
- Oracle
- Amazon Redshift



O knex.js é definido como um "query builder", ou seja, um construtor de consultas para o banco de dados que ele se conecta. É pensado para ser flexível e simples de usar, fornecendo um conjunto de funcionalidades básicas para realizar qualquer tipo de operação que você desejar no seu banco de dados.

Vamos usar o knex.js no nosso projeto e apresentaremos pontualmente as funcionalidades que estaremos utilizando. Para saber mais sobre o knex.js visite: <https://knexjs.org/>

## Integrando knex.js no Admin do PetTopStore

Para integrar o knex.js no admin primeiramente instale-o no projeto junto com o driver do mysql:

```
npm install knex mysql
```

```
[isaac@gru admin]$ npm install knex mysql2
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated

added 187 packages, and audited 242 packages in 24s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
[isaac@gru admin]$
```

Note que estamos instalando o driver mysql com o knex, que suporta tanto o banco de dados MySQL como o MariaDB.

## Configurando o knex.js e criando uma tabela no banco

Para configurar o knex.js, primeiramente crie um arquivo no projeto chamado **knexfile.js** com o seguinte conteúdo:

```
module.exports = {
  client: 'mysql',
  connection: {
    host : '127.0.0.1',
    user : 'seu_usuario',
    password : 'sua_senha',
    database : 'pettopstore'
  }
};
```

É importante trocar o valor de *user* e *password* para o seu usuário e senha do banco de dados.

Agora vamos testar nossa configuração criando tabelas do nosso banco de dados utilizando um recurso chamado "migrations" do knex.js.

Migrations são arquivos Javascript que são criados por uma ferramenta de linha de comando do knex chamada 'knex cli' e tem por objetivo organizar de forma sequencial as alterações no esquema do seu banco de dados, como por exemplo a criação de tabelas, alteração de colunas, remoção de colunas, etc.

Para mais informações sobre migrations acesse: <https://knexjs.org/#Migrations>

Para criar uma migration digite o seguinte comando:

```
npx knex migrate:make create_employees
```

Esse comando irá criar uma migração chamada "create\_employees". Repare que esse nome foi passado por nós, e poderia ser qualquer nome, mas escolhemos um nome que representa o que essa migração realmente fará (criar a tabela de employees). O comando cria um arquivo Javascript na pasta "migrations" dentro da aplicação com o nome `TIMESTAMP_create_employees.js` (esse `TIMESTAMP`) é um número que representa a data/hora atual e faz parte do nome do arquivo, por exemplo: `migrations/20210820011759_create_employees.js`



O arquivo de migração é um conjunto de duas funções, uma chamada "up" e uma chamada "down" que vem vazias e você deve preencher com os comandos que desejar, como visto abaixo:

```
migrations > 20210820011759_create_employees.js > ...  
1  
2  exports.up = function(knex) {  
3  
4  };  
5  
6  exports.down = function(knex) {  
7  
8  };
```

A função "up" será executada quando a migração for executada e ela deverá nesse caso ter os comandos do knex necessários para criar a tabela "employees".

A função "down" deve ter os comandos opostos aos da função "up", ou seja, no nosso caso ela deve implementar um código que remove a tabela "employees".

Migrations podem ser executadas para cima "up" ou para baixo "down" o que aplica/desfaz mudanças no banco de dados, por isso a necessidade de duas funções que você deve implementar.

Implementando as funções up e down temos o seguinte conteúdo para nosso arquivo de migration:

```
exports.up = function(knex) {  
  return knex.schema.createTable('employees', function (table) {  
    table.increments('id');  
    table.string('email');  
    table.string('password');  
    table.string('name');  
    table.boolean('is_admin');  
  })  
};  
  
exports.down = function(knex) {  
  return knex.schema.dropTable('employees');  
};
```

Repare que para a função "up" usamos o comando *knex.schema.createTable* que

recebe o nome da tabela que desejamos criar e uma função recebendo uma variável chamada table. Nessa variável table podemos adicionar campos com os métodos table.increments('id') para criar um campo chamado 'id' inteiro o autoincremental, o método table.string que recebe o nome do campo de texto que queremos criar e o método table.boolean que recebe o nome do campo booleano que estamos criando.

Existem outros métodos para manipulação do esquema do banco de dados que podem ser conferidos na documentação do knex.

No método "down" simplesmente removemos a tabela criada com o comando "return knex.schema.dropTable('employees')".

Você agora somente criou os comandos da migration. Falta executar. Para isso execute o comando no terminal:

```
npx knex migrate:up
```

Isso fará com que o método "up" da próxima migração ainda não executada seja rodado. Como no nosso caso só temos uma, ela será a executada. Repare que se você rodar esse comando mais de uma vez, a migração não executará novamente pois o knex guarda a informação de que migrações já rodaram:

```
[isaac@gru admin]$ npx knex migrate:up
Batch 1 ran the following migrations:
20210820011759_create_employees.js
[isaac@gru admin]$ npx knex migrate:up
Already up to date
[isaac@gru admin]$ |
```

Se tudo deu certo, a tabela "employees" foi criada no banco de dados. Verifique utilizando a ferramenta que desejar:

```
[isaac@gru admin]$ mysql -uroot -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 19
Server version: 10.6.3-MariaDB Arch Linux

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use pettopstore
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [pettopstore]> show tables;
+-----+
| Tables_in_pettopstore |
+-----+
| employees              |
| knex_migrations        |
| knex_migrations_lock   |
+-----+
3 rows in set (0.000 sec)

MariaDB [pettopstore]> desc employees;
+-----+-----+-----+-----+-----+-----+
| Field      | Type                | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id         | int(10) unsigned    | NO   | PRI | NULL    | auto_increment |
| email      | varchar(255)        | YES  |     | NULL    |                |
| password   | varchar(255)        | YES  |     | NULL    |                |
| name       | varchar(255)        | YES  |     | NULL    |                |
| is_admin   | tinyint(1)          | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.004 sec)
```

Repare que, usando o cliente do mysql no linux, usamos os comandos "use pettopstore" para usar o banco correto, o comando "show tables" para mostrar as tabelas do banco (realmente a tabela employees está nessa lista) e por último o comando "desc employees" para exibir quais campos foram criados, confirmando que está tudo ok. Esses são comandos padrão do MySQL para verificar os campos das tabelas no banco.

Note que além da tabela "employees" existem duas outras ("knex\_migrations" e "knex\_migrations\_lock"). Essas tabelas são de uso interno do knex para que ela consiga controlar que migrations já foram executadas ou não no banco de dados. Não remova essas tabelas.

Pronto, você criou e executou uma migration que foi responsável pela criação da tabela employees no nosso banco de dados. O knex.js foi responsável por criar o comando "CREATE TABLE" para você que especificou que campos criar na migrations.

O uso de migrations é uma prática muito útil, principalmente quando temos equipes trabalhando no mesmo projeto ou projetos com diversas implantações em

diferentes servidores e bancos de dados, fazendo com que exista um controle automatizado do que falta em cada banco quando o sistema vai evoluindo.

## Criando as outras tabelas

Vamos repetir o processo acima e criar as tabelas restantes com uma migration para cada.

Crie as migrations restantes assim:

**Atenção: Execute um comando por vez (NÃO copie e cole mais de um no terminal) e na mesma ordem que está no material, pois assim você terá garantias de que cada arquivo ficará na ordem certa de criação das tabelas no banco. O knex adiciona um timestamp antes do nome do arquivo e, para garantir a consistência, você não pode rodar todos de uma vez só ou irá ter tabelas criadas na ordem incorreta, o que irá acarretar erros nos relacionamentos.**

Abaixo segue o conteúdo de cada arquivo de migração criado (menos o de employees, que já foi criado anteriormente). Os timestamps dos nomes dos arquivos vão variar no seu projeto):

### Tabela de clientes

```
npx knex migrate:make create_clients
```

Coloque esse conteúdo no arquivo

**migrations/20210820015100\_create\_clients.js**

```
exports.up = function(knex) {
  return knex.schema.createTable('clients', function (table) {
    table.increments('id');
    table.string('email');
    table.string('password');
    table.string('name');
  })
};

exports.down = function(knex) {
  return knex.schema.dropTable('clients');
};
```

## Tabela de categorias

```
npx knex migrate:make create_categories
```

Coloque esse conteúdo no arquivo

**migrations/20210820015127\_create\_categories.js**

```
exports.up = function(knex) {  
  return knex.schema.createTable('categories', function (table) {  
    table.increments('id');  
    table.string('name');  
  })  
};  
  
exports.down = function(knex) {  
  return knex.schema.dropTable('categories');  
};
```

## Tabela de produtos

```
npx knex migrate:make create_products
```

Coloque esse conteúdo no arquivo

**migrations/20210820015133\_create\_products.js**

```
exports.up = function(knex) {  
  return knex.schema.createTable('products', function (table) {  
    table.increments('id');  
    table.string('name');  
    table.text('description');  
    table.string('photo', 2000);  
    table.decimal('price', 10, 2);  
    table.integer('category_id').unsigned().references('categories.id');  
  })  
};  
  
exports.down = function(knex) {  
  return knex.schema.dropTable('products');  
};
```

## Tabela de vendas

```
npx knex migrate:make create_sales
```

Coloque esse conteúdo no arquivo

**migrations/20210820015140\_create\_sales.js**

```
exports.up = function(knex) {
  return knex.schema.createTable('sales', function (table) {
    table.increments('id');
    table.integer('client_id').notNullable().unsigned().references('clients.id');
    table.integer('employee_id').nullable().unsigned().references('employees.id');
  })
};

exports.down = function(knex) {
  return knex.schema.dropTable('sales');
};
```

## Tabela de itens (da vendas)

```
npx knex migrate:make create_items
```

Coloque esse conteúdo no arquivo

**migrations/20210820015146\_create\_items.js**

```
exports.up = function(knex) {
  return knex.schema.createTable('items', function (table) {
    table.increments('id');
    table.integer('sale_id').notNullable().unsigned().references('sales.id');
    table.integer('product_id').notNullable().unsigned().references('products.id');
  })
};

exports.down = function(knex) {
  return knex.schema.dropTable('items');
};
```

## Executando todas as migrações

Para executar as migrações que acabou de criar você pode executar o comando 'npx knex migrate:up' várias vezes (uma para cada migration) ou pode também rodar todas as restantes de uma vez com um só comando:

```
npx knex migrate:latest
```

Conferindo se as tabelas estão no banco de dados:

```
MariaDB [pettopstore]> show tables;
+-----+
| Tables_in_pettopstore |
+-----+
| categories             |
| clients                |
| employees              |
| items                  |
| knex_migrations        |
| knex_migrations_lock   |
| products               |
| sales                  |
+-----+
8 rows in set (0.000 sec)

MariaDB [pettopstore]> |
```

Sucesso. Criamos com o knex.js todas as tabelas do banco de dados utilizando migrations.

Como você notou nas migrations, existem comandos específicos para se criar colunas com as configurações desejadas, podendo ser informado o seu tipo, se são chaves estrangeiras, se podem ou não receber valores nulos, etc.

## Conclusão

Nessa aula criamos o projeto base da área administrativa da nossa loja virtual, modelamos o banco de dados, adicionamos o knex.js no projeto, o configuramos para nosso banco de dados e criamos as tabelas necessárias utilizando o recurso de migrations com o knex cli (ferramenta de linha de comando do knex)

O knex é uma ferramenta muito poderosa e flexível. Não deixe de estudar mais sobre ele no site: <https://knexjs.org/>