INTRO TO DATABASE

# SQL x NOSQL

Structure Query Language and Not Only

* Uma tabela de usuários com nome, endereço é um exemplo de SQL.

Contudo, imagine que um cliente quer adicionar e-mail, e outro quer retirar o endereço ?

Tabela : modus operandi de SQL

|  |  |  |
| --- | --- | --- |
| **Nome** | **Endereço** | **Email** |
| Matheus | Rua x | [math@js.com](mailto:math@js.com) |
| Andre | Null | Null |

Isso é o que uma SQL iria fazer... completar lacunas com Null. É uma péssima ideia, imagina mandar e-mail automaticamente para null?

* Com NoSQL é diferente, eu teria objetos diferentes:

**{**

**first\_name : “Matheus”,**

**adress: “rua x”,**

**email: “math@js.com**

**}**

**{**

**first\_name : “andre”**

**}**

* SQL é tradicional, bem estruturado, seguro... já NoSQL é flexível, moderno e adaptável, bom para StartUp.
* SQL é melhor ‘relacional’, e NoSQL é ‘no relacional’. Pense em excel... muito mais fácil de relacionar as variáveis! Posso ter uma tabela de cliente, outra de produtos e outra de compras, e relacioná-las!
* Com NoSQL eu teria que usar ‘embeded objects’ :

**{**

**orderID: order\_01,**

**custumer :**

**{**

**firstName: “Matheus”,**

**address: “rua X”**

**},**

**products :**

**{**

**prodName: “pencil”,**

**prodPrice : 0.80**

**}**

**}**

................... Or ............................

**{**

**orderID: order\_01,**

**customer: person\_01,**

**products: prod\_01**

**}**

**{**

**personID: person\_01,**

**firstName: “matheus”,**

**address: “rua x”**

**}**

**{**

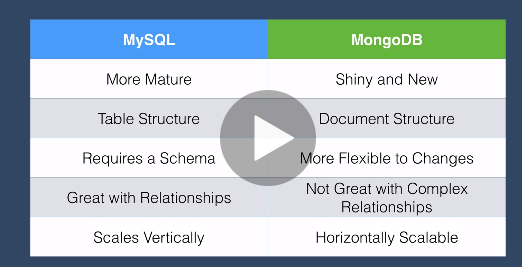
**prodID : prod\_01,**

**prodName: “pencil”,**

**prodPrice : 0.80**

**}**

* O quadro acima não é muito prático.... Se seu foco são dados estruturados, sequenciais e relacionados, use SQL - **MySQL**. Se seus dados são muito dinâmicos, grandes ou complexos, escaláveis, e com previsão de um crescimento constante ( Instagram), use NoSQL - **MongoDB**.
* SQL usa table structure, e NoSQL usa ‘Document Structure’

****

# SQL tutorial

There are a lot of info into w3schools. Os principais commandos são – CRUD:

1. Create
2. Read
3. Update
4. Destroy

CREATE TABLE table\_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
   
);

**CREATE TABLE products (**

**Id INT NOR NULL,**

**name STRING,**

**price MONEY,**

**PRIMARY KEY (id)**

**)**

* PRIMARY KEY define qual dos atributos será a identidade da coluna!
* AGORA vamos ver como inserir data – INSERT DATA –

**INSERT INTO products**

**VALUES (1, "Pen", 1.20)**

**​**

**INSERT INTO products (id,name)**

**VALUES (2, "pencil")**

* A PRIMEIRA FORMA INSERE EM TODAS, A SEGUNDA DEIXA A NÃO ESPECIFICADAS EM BRANCO – NULL
* LEMBRE QUE HÁ DADOS QUE NÃO PODEM SER NULL

**SELECT name, price FROM products;**

* ****Acima, se mostra apenas as colunas ‘name’ e ‘price’****

**SELECT \* FROM products WHERE id=1**

* ****Acima, se mostra apenas a linha do produto de ’ id = 1 ‘****

**SELECT price FROM products WHERE name="pencil"**

* ****Acima, se mostra o preço do ‘pencil’****

**SELECT \* FROM products WHERE name<>"pencil"**

* ****Acima, se mostra todas as linhas de todos produtos, exceto as do produto cujo nome é ‘pencil’****

## 2.1 Update

* ****Eu posso atualizar um dado da tabela :****

**UPDATE products**

**SET price = 0.5**

**WHERE id=2**

* ****Ou eu posso atualizar uma nova info da tabela :****

**ALTER TABLE products**

**ADD stock INT**

## 2.2 Delete

Agora iremos ver como deletar contúdo.

**DELETE FROM products**

**WHERE name="pencil"**

Isto irá deletar toda a linha

## 2.3 Relacionando os dados

Ao lado temos uma tabela com uma tabela de comandos de pedidos, relacionando os produtos e os clientes pedintes

**CREATE TABLE orders (**

**id INT NOR NULL,**

**order\_number INT,**

**custumer\_id INT,**

**product\_id INT,**

**PRIMARY KEY (id),**

**FOREIGN KEY (custumer\_id) REFERENCES custumers(id),**

**FOREIGN KEY (product\_id) REFERENCES products(id)**

**)**

* Agora, precisamos usar o ‘join’ para linkar tudo – leia sobre as várias formas de join –

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2  
ON table1.column\_name = table2.column\_name;

**SELECT orders.order\_number, customers.first\_name, customers.last\_name, customers.address**

**FROM orders**

**INNER JOIN customers ON orders.custumer\_id = customers.id**

* E é assim que ficaria nossa nova tabela, com o pedido e os dados do usuário que pediu! veja que o ‘ON’ faz o match da FOREIGN KEY com sua REFERENCE
* FROM the foreign INNER JOIN reference

## MongoDB

After installing, it is a good idea to study the CRUD of that system

* We will run the server with the line :

**mongod**

* In a new prompt, we run the Mongo Shell, which allows us to CRUD:

**mongo**

* We can always tap the line above to search for the shortcuts

**help**

* It will show us all the DBs that contains collections

**show dbs**

* And that’s how we can create a new DB, called “shop”

**use shopDB**

* IF, it already exists, we type

**use shop**

* And how to insert a collection of objects

**db.products.insertOne({ \_id: 1, name: "Pen", price: 1.50 })**

* Show will always tell us what the keyword contains

**show colections**

* And if you look into the documentation, we can add many as well

**db.collections.insertMany ….**

## ****3.1 Reading Data****

* The find() method will help us to read the collections. I can use parameters or simply say products.find() and it will return all the items - remember, products is the name of my collection -

**db.products.find()**

* If I want to see the products that match the name = “Pencil” ….

**db.products.find({name: “Pencil”})**

* Now, lets see the products that have the price Greater Then : 1
* – the query will be ‘price’ –

**db.products.find ( { price: {$ gt: 1}})**

* Now, I want the name of the products that has the id =1
* – the query will be ‘\_id’ –
* - the projection will be ‘name, if true’ –
* By default, the id will always be a return, which means, a projection. Unless you put ‘\_id = 0 ’

**db.products.find({\_id: 1}, {name: 1})**

## 3.2 Update Data

* Now, I want to update a field, let’s say, the STOCK of a particular product.
* I need the first input, that is the query selector
* The second input is the updating data

**db.products.updateOne({ \_id: 1}, {$set: {stock: 32}})**

## 3.3 Delete Data

* We can use ‘deleteOne()’ or ‘deleteMany()’
* If I want to delete an entire product:

**db.products.deleteOne({ \_id: 1})**

## 3.4 Relations among Data

* There are two ways, depending on the architecture of your data
* First, let’s intro a product that has an data called “reviews”, which is dynamic, once that could be several reviews to a single product

**db.products.insert({**

**\_id: 3,**

**name: "Rubber",**

**price: 1.30,**

**stock: 43,**

**reviews: [**

**{**

**authorName: "Sally",**

**rating: 5,**

**review: "Best Ever!"**

**},**

**{**

**authorName: "Bob",**

**rating: 4,**

**review: "good"**

**}**

**]**

**})**

* Another insertion…

**db.products.insert({**

**\_id: 2,**

**name: "Pencil",**

**price: 0.80,**

**stock: 12,**

**reviews: [**

**{**

**authorName: "Mat",**

**rating: 3,**

**review: "not bad"**

**},**

**{**

**authorName: "Cintia",**

**rating: 4,**

**review: "good"**

**}**

**]**

**})**