Introdução ao MongoDB e Banco de Dados NoSQL

Conhecendo os Tipos de Banco de Dados NoSQL

- **Tipos de banco NoSQL**

> **Sandbox - NEO4J**
>
> **https://sandbox.neo4j.com/**
>
> **CREATE (:Client{name : "Bob Esponja", age : 28, hobbies : ['Caça agua-viva, Comer']})**
>
> Criar um nó(dado)

> **MATCH(bob_esponja) RETURN bob_esponja;**
>
> Recuperar todos registro

> **CREATE (:Client{name : "Lula Molusco", age: 30, hobbies : ['musica']}) -[:Bloqueado]→(:Client {name : "Patrick", hobbies: ['Caçar']})**
>
> Cria relacionamento

> **CREATE (:Object)**
>
> Cria um nó

> **MATCH (lula:Client {name: "Lula Molusco}), (patrick: Client {name:"Patrick"}) CREATE (lula)- [:Bloqueado]→(patrick)**
>
> Cria relacionamento Entre objetos

> **MATCH (lula: Client {name: "Lula Molusco})-[relaciona: Bloqueado→]-() DELETE relaciona**
>
> Deleta nó e relacionamento

> **MATCH(lula:Client {name: "Lula Molusco"}) DELETE lula;**
>
> Deleta um nó

> **MATCH (patrick:Client {name: "Patrick"}) SET patrick.hobbies = ['caçar'];**
>
> Atualiza um dado/property

> **MATCH(patrick:Client {name: "Patrick"}) SET patrick:Client_Premium**
>
> Alterar Label

> **MATCH(todos) RETURN todos;**
>
> Recuperar todos registros e objetos

- ## ColunaFamilia de Colunas

- **Cassandra**

> **Sandbox Cassandra**
>
> https://www.katacoda.com/datastax/courses/cassandra-try-it-out/try-cql

> **CREATE KEYSPACE fenda_biquini WITH replication = {'class': 'SimpleStrategy', 'replication_factor':1};**
>
> KEYSPACE é analogo a um database

> **use fenda_biquini;**
>
> Coloca em uso a keyspace

> **CREATE COLUMNFAMILY clients (name TEXT PRIMARY KEY, age INT);**
>
> É análogo a uma tabela

> **INSERT INTO clients (name, age) VALUES ('Bob Esponja', 38);**
>
> Insere registro na COLUMNFAMILY clients

> **INSERT INTO clients JSON '{"name": "Patrick"}';**
>
> Insere registro na COLUMNFAMILY clients porem no formato de JSON

> **UPDATE clients SET age=33 WHERE name ='Patrick';**
>
> Atualiza registro

> **ALTER COLUMNFAMILY clients ADD hobby TEXT;**
>
> Adiciona coluna

> **UPDATE clients SET hobby='caçar' WHERE name ='Patrick';**
>
> Atualiza registro na coluna criada

> **DELETE FROM clients WHERE name = 'Patrick';**
>
> Deleta Registro

> **SELECT * FROM clients;**
>
> Seleciona registros da COLUMNFAMILY clients

> **SELECT age, WRITETIME(age) FROM clients;**
>
> Obtém a hora que foi inserida o registro

> **SELECT * FROM clients WHERE name ='Bob Esponja';**
>
> Consulta com filtro

> **SELECT JSON * FROM clients;**
>
> Retorna a consulta em formato de JSON

- ## Chave-valor

- **REDIS**

> **Sandbox REDIS**
>
> https://try.redis.io/

> **SET user1:name "Bob Esponja"**
>
> Insere valor

> **GET user1:name**
>
> Retorna valor

> **SET user '{"name":"Patrick", "age": 31}'**
>
> Insere no formato JSON

> **GET user**
>
> Retorna registro

> **SET user2:name "Lula Molusco" EX 10**
>
> Cria usuário e expira(apaga) após 10 segundos

> **EXISTS user2:name**
>
> Verifica se o registro existe retornando 0 ou 1

> **LPUSH user1:hobbies "caçar"**
>
> Insere em formato de lista

> **LINDEX user1:hobbies 0**
>
> Retorna o registro da lista no índice 0

> **LRANGE user1:hobbies 0 1**
>
> Retorna o registros da lista no range de 0 a 1

> **TYPE user1:name**
>
> Qual tipo do valor

> **TTL user1:name**
>
> Qual o tempo em que expira uma chave (-1 significa que não tem tempo)

> **PTTL user1:name**
>
> Qual o tempo em que expira uma chave em milissegundos

> **PERSIST user2:name**
>
> O comando PERSIST remove o tempo de expiração da chave

> **DEL user2:name**
>
> Deleta registro

- ## MongoDB

- Operações e manipulação de dados

> **show databases;**
>
> Lista databases

> **use fenda_biquini**
>
> cria e coloca em uso o banco de dados

> **db.createCollection("test". {capped: true, max: 2, size: 2});**
>
> Cria uma Collection, capped para ser limitada em 2 documentos

> **show collections;**
>
> Lista as collections criadas

> **db.test.insertOne({"name": "Teste 1"});**
>
> Insere dado na collection

db.test2.insertOne({"age": 10});

Criar collection e insere

db.clients.insert([{"name": "patrick", "age": 38}, {"name": "Bob"}]);

Cria uma collection com dois documentos

db.clients.save({ "_id": Objectid("sfd6f4s6df645fs"),"name": "patrick", "age": 38 })

Atualiza(por completo) ou insere registro caso não exista

db.clients.update({"name": "patrick"}, {$set: {"age": 38}})

Atualiza registro especifico. Atualiza a idade do patrick para 38

db.clients.update({"name": "patrick"}, {$set: {"age": 38}}, {multi: true})

Atualiza todos registros (parametro multi: true) com nome patrick para idade 38

db.clients.updateMany({"age": 44}, {$set: {"age": 55}})

Alternativa ao (parametro multi:true). Atualiza todos registros de 44 para 55

db.clients.find({"age": 44})

Seleciona clientes com idade = 44

db.clients.find({"age": 44}).limit(1)

Seleciona apenas o primeiro cliente com idade = 44

db.clients.find({"age": 44, "name": "patrick"})

Seleciona cliente com idade = 44 E nome patrick

db.clients.find({"age": {$in: [30,41]}})

Seleciona quem tem 30 e 41 anos

db.clients.find({$or: [{"name": "patrick"}, {"age": 44}]})

Seleciona quem chama patrick OU tem 44 anos

> **db.clients.find({"age": {$lt: 55}})**
>
> Seleciona todos registros com idade MENOR que 55 anos

> **db.clients.find({"age": {$lte: 55}})**
>
> Seleciona todos registros com idade MENOR IGUAL que 55 anos

> **db.clients.deleteOne({"age": 55})**
>
> Deleta um registro com idade de 55

> **db.clients.deleteMany({"age": 55})**
>
> Deleta todos registros com idade de 55

> **db.test.find({})**
>
> Lista todos documentos da collection

- Performance e índices

> **for (var i=0; i< 10000; i++){**
>
> **db.clients.insert({name: "Cliente" + i, age:i});**
>
> }
>
> Insere 10000 Registros na collection clients

> **db.getColletion('clients').count({})**
>
> Conta a quantidade de registros

> **db.getColletion('clients').find({ _id: ObjectId("56dfg42fd5df45gdd5d")}).explain(true)**
>
> Busca registro e mostra o plano de execução

> **db.getColletion('clients').createIndex({name: 1}, {"name": "nome_indice"})**
>
> Cria campo de indice de forma ordenada ({name: 1})

- Agregações

db.getCollection('restaurants').count({})

Conta registros

db.getCollection('restaurants').distinct("cuisine")

Faz distinct por campo

db.getCollection('restaurants').aggregate([{$group: {_id: "$cuisine", total: {$sum:1}}}])

Faz a soma por cuisine

db.getCollection('restaurants').aggregate([{$addFields :{"teste": true}}])

Adiciona um campo ao resultado da agregação

db.getCollection('restaurants').aggregate([{$group: {_id: "$cuisine", total: {$sum:1}, id_maximo: {$max: "$restaurant_id"}}}])

Faz a soma por cuisine com restaurant_id maximo

db.getCollection('restaurants').aggregate([{$match :{$and:[{cuisine: "American"}, {borouch: "Brooklyn"}]}}])

Faz um filtro selecionando apenas cuisine: "American" E borouch: "Brooklyn"

db.getCollection('restaurants').aggregate([{$match :{$or:[{cuisine: "American"}, {borouch: "Brooklyn"}]}}])

Faz um filtro selecionando apenas cuisine: "American" OU borouch: "Brooklyn"