

Bancos de Dados NoSQL

Capítulo 01 – Banco de Dados NoSQL

PROF.: RICARDO BRITO ALVES



Banco de Dados NoSQL

Capítulo 01 – Aula 01.01 – Visão geral de NoSQL

PROF.: RICARDO BRITO ALVES

Nesta Aula



- ❑ Definição de NoSQL
- ❑ Características
- ❑ Alguns bancos NoSQL

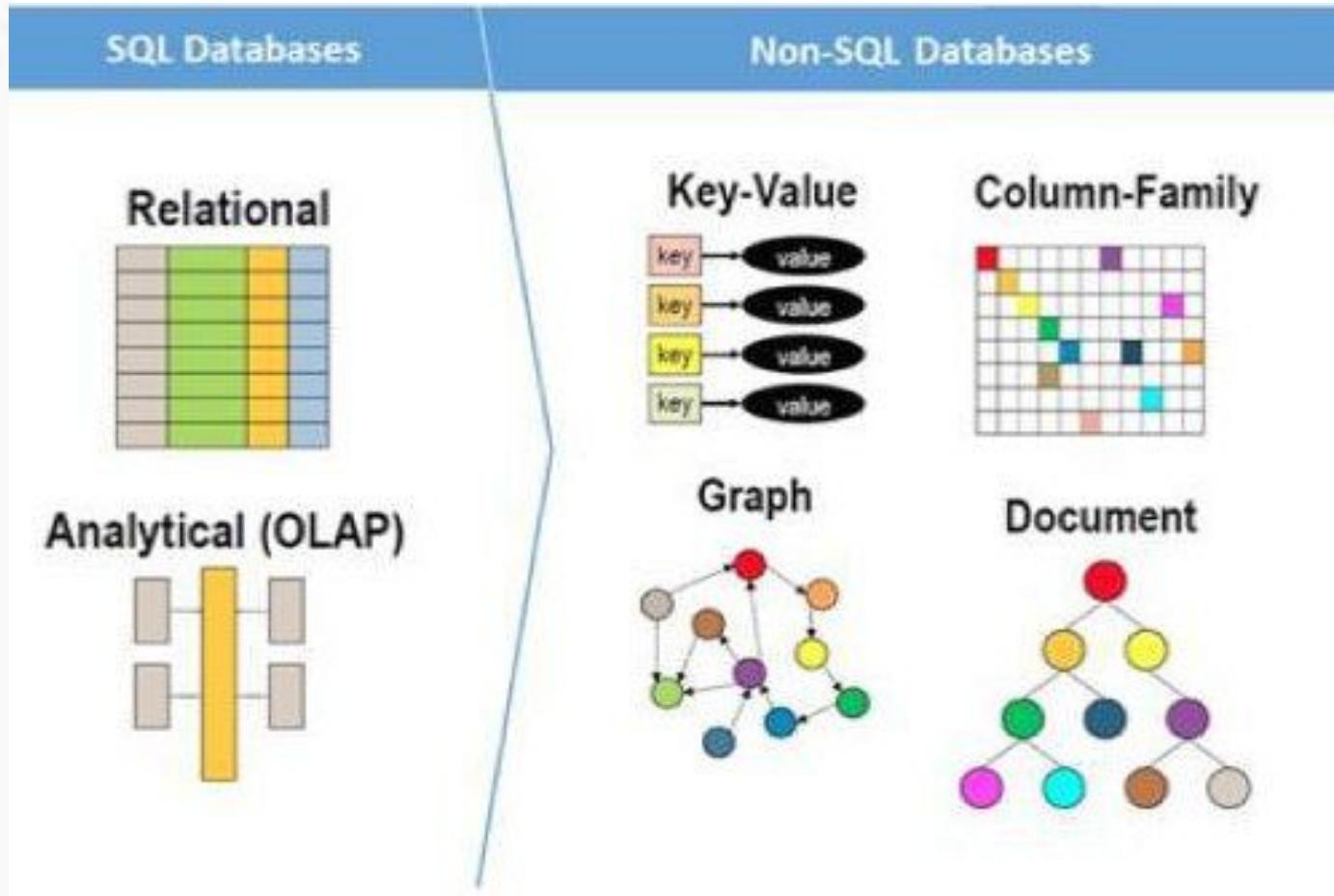
Banco de Dados Relacional



- Os dados são estruturados de acordo com o modelo relacional.
- SQL Server, Oracle, PostgreSQL, MySQL, DB2 e outros.
- Elementos básicos
 - Relações (tabelas) e registros (tuplas)
- Características fundamentais
 - ✓ Restrições de integridade
 - ✓ PK-primary key, FK-foreign key, UK-unique key, CK-check key, NN-not null
 - ✓ Normalização (formas normais)
 - ✓ Linguagem SQL (Structured Query Language)

Banco de Dados Relacional

IGTI



Ranking Banco de Dados

<https://db-engines.com/en/ranking>

Rank Aug 2020	Jul 2020	Aug 2019	DBMS	Database Model	Score		
					Aug 2020	Jul 2020	Aug 2019
1.	1.	1.	Oracle +	Relational, Multi-model 	1355.16	+14.90	+15.68
2.	2.	2.	MySQL +	Relational, Multi-model 	1261.57	-6.93	+7.89
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model 	1075.87	+16.15	-17.30
4.	4.	4.	PostgreSQL +	Relational, Multi-model 	536.77	+9.76	+55.43
5.	5.	5.	MongoDB +	Document, Multi-model 	443.56	+0.08	+38.99
6.	6.	6.	IBM Db2 +	Relational, Multi-model 	162.45	-0.72	-10.50
7.	↑ 8.	↑ 8.	Redis +	Key-value, Multi-model 	152.87	+2.83	+8.79
8.	↓ 7.	↓ 7.	Elasticsearch +	Search engine, Multi-model 	152.32	+0.73	+3.23
9.	9.	↑ 11.	SQLite +	Relational	126.82	-0.64	+4.10
10.	↑ 11.	↓ 9.	Microsoft Access	Relational	119.86	+3.32	-15.47
11.	↓ 10.	↓ 10.	Cassandra +	Wide column	119.84	-1.25	-5.37
12.	12.	↑ 13.	MariaDB +	Relational, Multi-model 	90.92	-0.21	+5.96
13.	13.	↓ 12.	Splunk	Search engine	89.91	+1.64	+4.03
14.	↑ 15.	↑ 15.	Teradata +	Relational, Multi-model 	76.78	+0.81	+0.14
15.	↓ 14.	↓ 14.	Hive	Relational	75.29	-1.14	-6.51

NoSQL

IGTI

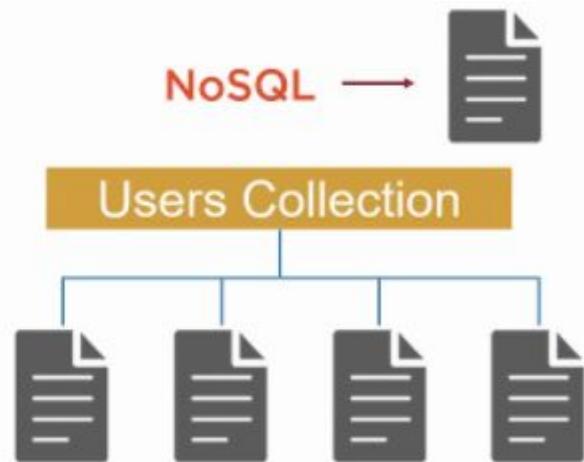
NoSQL é um termo genérico que define **bancos de dados não-relacionais**.

A tecnologia NoSQL foi iniciada por companhias líderes da Internet como Google, Facebook, Amazon e LinkedIn, para superar as limitações de banco de dados relacional para aplicações web modernas.

SQL

Table: Users			
	id	name	createdAt
	Filter	Filter	Filter
1	1	Ashton	2018-08-12 1...
2	2	Jakeem	2018-08-12 1...
3	3	Uma	2018-08-12 1...

NoSQL



NoSQL

Não quer dizer que seus modelos não possuem relacionamentos e sim que *não são orientados a tabelas*.

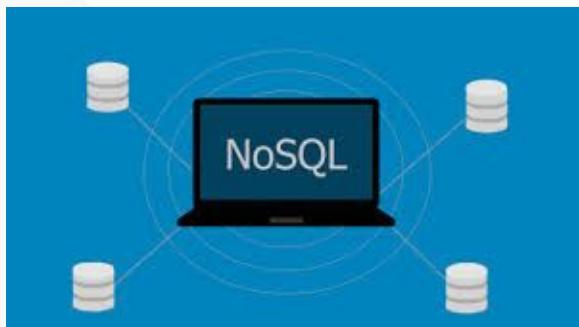
Not Only SQL (não apenas SQL).

Bancos de dados NoSQL são cada vez mais usados em *Big Data* e *aplicações web de tempo real*.



NoSQL - Características

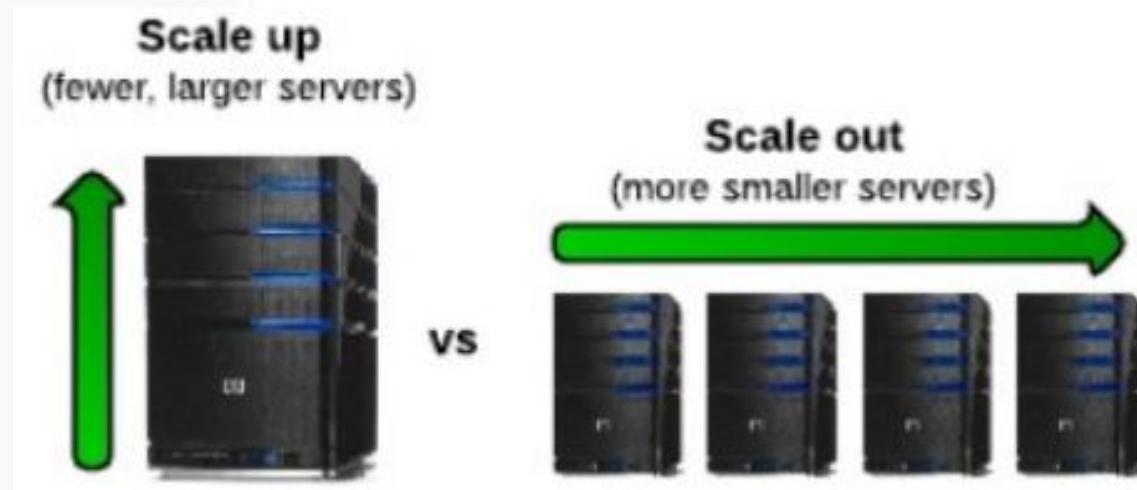
- Escalabilidade horizontal.
- Ausência de esquema ou esquema flexível.
- Suporte a replicação.
- API simples.
- Nem sempre prima pela consistência.



Escalabilidade Horizontal

À medida em que o volume de dados cresce, aumenta-se a necessidade de escalabilidade e melhoria do desempenho.

Podemos escalar **verticalmente** (adicionar CPU, memória, disco) ou podemos escalar **horizontalmente** (adicionar mais nós).

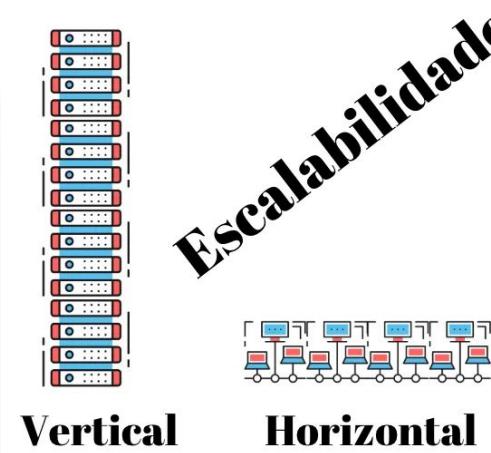


Escalabilidade Horizontal

Dentre todas as possibilidades para esta solução, **a escalabilidade horizontal se torna a mais viável, porém requer diversas threads ou que processos de uma tarefa sejam criadas e distribuídas.**

Não é que os bancos de dados relacionais não escalam, eles não escalam facilmente.

Como nos modelos NoSQL não existe bloqueios, esse tipo de escalabilidade se torna mais viável.



Escalabilidade Horizontal

Uma alternativa muito utilizada para alcançar a escalabilidade horizontal é o **Sharding**, que *divide os dados em múltiplas tabelas a serem armazenadas ao longo de diversos nós na rede*.

Um shard de banco de dados, ou literalmente um fragmento de banco de dados, é uma partição horizontal de dados em um banco de dados ou mecanismo de busca.

Cada partição individual é referenciada como um shard ou shard de banco de dados.

Sharding

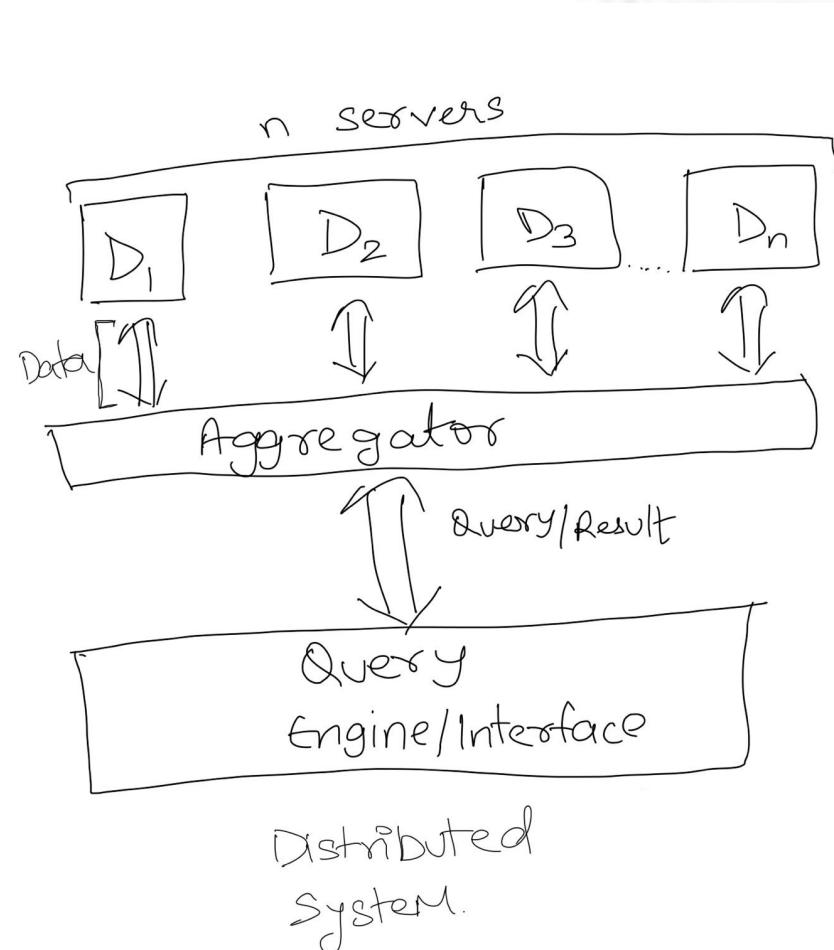
Como o **Sharding** ajuda a obter

Você pode particionar um índice em um servidor separado.

Se você consultar um servidor, obterá um conjunto completo de dados. **O sharding é distribuída usa um agregador**, que combina os resultados de todos os servidores. **Um agregador é um servidor.**

Esse programa agregador é responsável por combinar os resultados de todos os servidores.

Em outras palavras, **Sistemas**



Ausência de Esquema

Ausência de esquema (Schema-free) ou esquema flexível é uma outra característica em bancos de dados NoSQL.

Basicamente é a *ausência parcial ou total de esquema que define a estrutura de dados*.

É justamente essa ausência de esquema que **facilita uma alta escalabilidade** e alta disponibilidade, *mas em contrapartida não há a garantia de integridade dos dados, fato este que não ocorre no Modelo Relacional*.

Ausência de Esquema

ID	Name	IsActive	Dob
1	John Smith	True	8/30/1964
2	Sarah Jones	False	2/18/2002
3	Adam Stark	True	7/13/1987

Document 1

```
{  
  "id": "1",  
  "name": "John Smith",  
  "isActive": true,  
  "dob": "1964-30-08"  
}
```

Document 2

```
{  
  "id": "2",  
  "fullName": "Sarah Jones",  
  "isActive": false,  
  "dob": "2002-02-18"  
}
```

Document 3

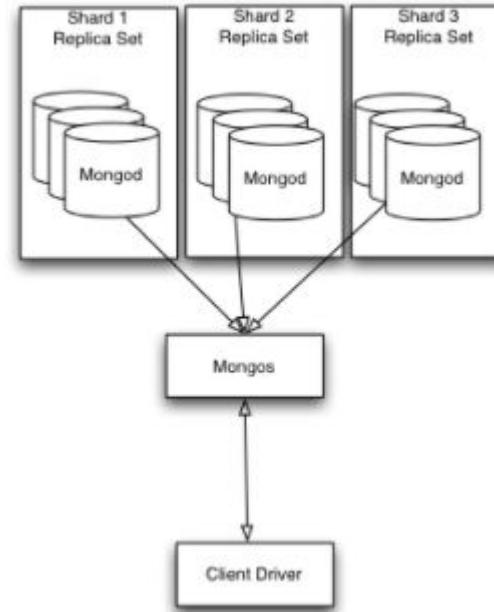
```
{  
  "id": "3",  
  "fullName":  
  {  
    "first": "Adam",  
    "last": "Stark"  
  },  
  "isActive": true,  
  "dob": "2015-04-19"  
}
```

Replicação

Supor o nativo a replic o o  o o outra forma de prover a escalabilidade, pois, no momento em que permitimos a replic o o de forma nativa o tempo gasto para recuperar informa o es  o o reduzido.

Replicação refere-se ao armazenamento de dados e a estratégia de backups entre computadores em locais distintos.

É um conjunto de tecnologias utilizadas ***para copiar e distribuir objetos e dados de um banco de dados para outro banco de dados***, sincronizando estes dados com a ***finalidade de se manter a consistência***.



Replicação

No caso dos bancos de dados SQL que possuem as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) o modelo normalmente utilizado é o ***master-slave (mestre-escravo)*** em que um servidor atua como master e os demais atuam como slave onde todas as operações que alteram algum dado no master são repassadas de imediato para os servidores slaves.

Um exemplo de banco de dados que utiliza este tipo de replicação é o MySQL.

Replicação

No caso dos bancos de dados NoSQL que trabalham com a propriedade BASE (Basicamente Disponível Eventualmente Consistente) o modelo normalmente adotado é o ***multimaster*** onde as operações de leitura e gravação podem ser realizadas através de qualquer nó.

Esse tipo de banco de dados possui fraca consistência e tem como fator principal a disponibilidade.

Um exemplo de banco de dados que trabalha com este tipo de replicação é o ***Cassandra*** que ***usa o particionamento dos dados através da estratégia de replicação e da topologia dos clusters (distribuição dos nós).***

API

API simples para acessar o banco de dados, ou seja, em banco de dados NoSQL, o foco não está no armazenamento dos dados e sim como recuperar estes dados de forma eficiente.

Pensando nisso, é fundamental ter APIs desenvolvidas para facilitar o acesso às informações de forma rápida e eficiente.

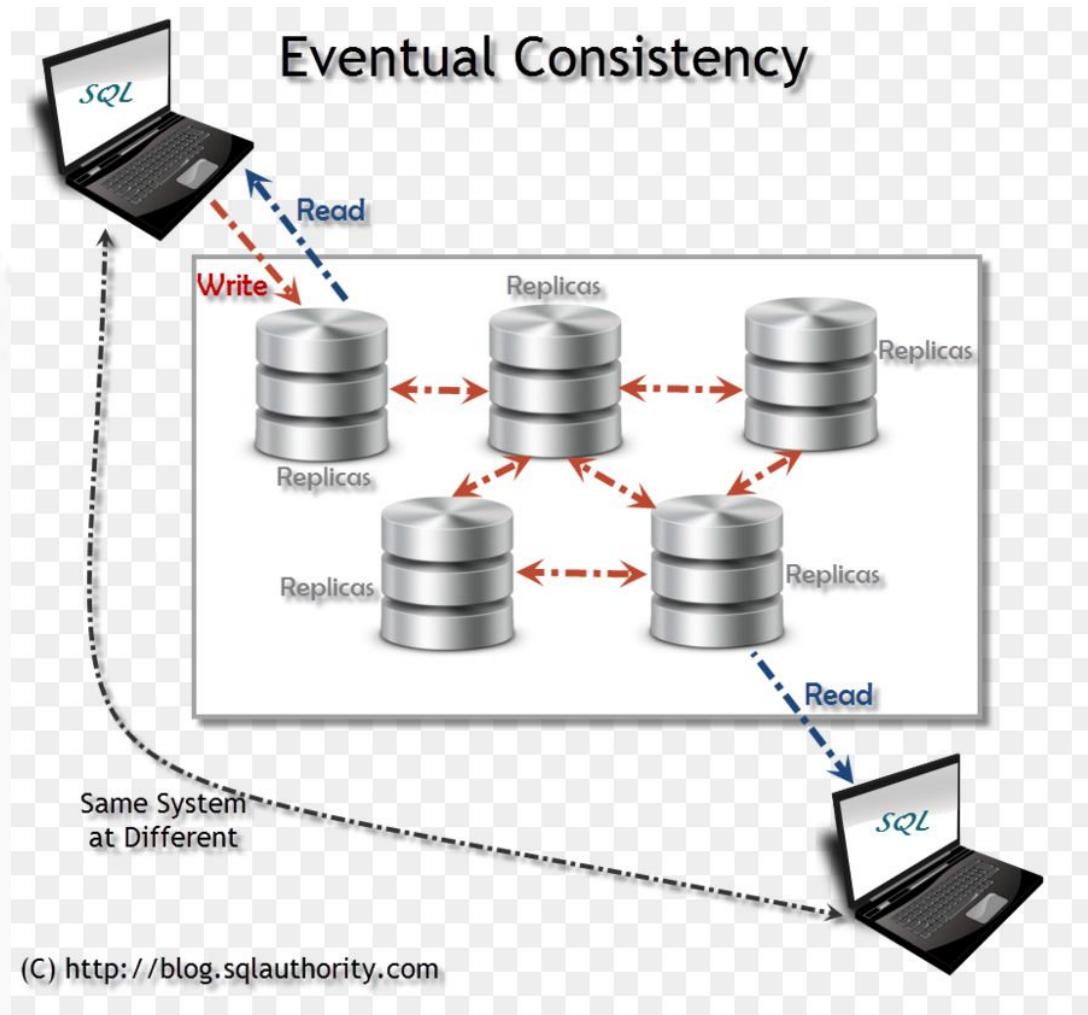


Consistência

Consistência eventual é outra característica particular de bancos NoSQL onde nem sempre a consistência dos dados é mantida.

É necessário haver um planejamento para que o sistema possa tolerar inconsistências temporárias com o objetivo de priorizar a disponibilidade.

Consistência



Alguns Bancos NoSQL



Aerospike: Banco de dados NoSQL que oferece *uma vantagem de velocidade de memória, atraindo empresas de anúncios de alta escala e aquelas que precisam de tempos de resposta em milissegundo.* Aerospike está apostando em novas categorias, incluindo jogos, e-commerce e segurança, onde a baixa latência é tudo.

Apache Cassandra: Os pontos fortes são a modelagem de dados NoSQL e *escalabilidade linear flexível em hardware* por conta do uso de cluster.

Amazon DynamoDB: foi desenvolvido pela Amazon para incrementar o seu e-commerce, possibilitando ter seus serviços altamente escaláveis. Inspirou o Cassandra, Riak, e outros projetos NoSQL.

Alguns Bancos NoSQL



MongoDB: É o banco de dados mais popular NoSQL, com mais de sete milhões de downloads e centenas de milhares de implantações. ***Sua popularidade se deve à facilidade de desenvolvimento e manejo flexível dos dados.*** Muito utilizado em aplicações de redes sociais web e móvel.

HBase: É o banco de dados que roda em cima do ***HDFS (Hadoop Distributed File System – sistema de arquivos distribuído)***, por isso dá aos usuários a capacidade única de trabalhar diretamente com os dados armazenados no Hadoop. As características incluem grande escalabilidade.

Redis: É o banco de dados NoSQL do tipo chave-valor mais conhecido. No mercado podemos encontrar diversas outras soluções que também são mecanismos de armazenamento baseado em chave-valor.

Conclusão



- ✓ Definição de NoSQL
- ✓ Características do NoSQL
- ✓ Alguns bancos NoSQL

Próxima Aula



01. •

Propriedades de Bancos de
Dados

02. •

03. •

04. •



Banco de Dados NoSQL

Capítulo 01 – Aula 01.02 – Propriedades dos Banco de Dados

PROF.: RICARDO BRITO ALVES

Nesta Aula



- Propriedades
 - ACID x Base
 - Teorema CAP

Propriedades ACID

Atomicidade - estado em que as modificações no BD devem ser todas ou nenhuma feita. Cada transação é dita como “atômica”. Se uma parte desta transação falhar, toda transação falhará.

Consistência - estado que garante que todos os dados serão escritos no BD.

Isolamento - requer que múltiplas transações que estejam ocorrendo “ao mesmo tempo”, não interfiram nas outras.

Durabilidade - garante que toda transação submetida (commit) pelo BD não será perdida



Propriedades BASE

Basically Available – Basicamente Disponível.

Soft-State – Estado Leve

Eventually Consistent – Eventualmente Consistente.

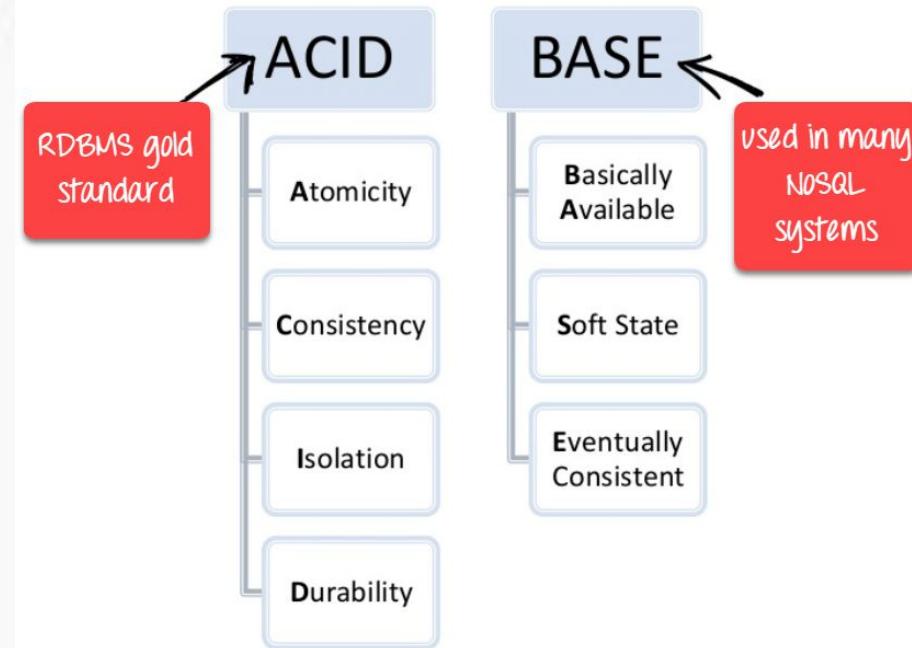
Uma aplicação funciona basicamente todo o tempo (Basicamente Disponível), não tem de ser consistente todo o tempo (Estado Leve) e o sistema torna-se consistente no momento devido (Eventualmente Consistente).



ACID vs BASE

BDs Relacionais trabalham com **ACID** (Atomicidade, Consistência, Isolabilidade, Durabilidade).

BDs NoSQL trabalham com **BASE** (Basicamente Disponível, Estado Leve, Eventualmente consistente).



CAP



Consistency – Consistência

Availability – Disponibilidade

Partition Tolerance – Tolerância ao Particionamento

Consistência

Consistência (**Consistent**)

As escritas são atômicas e todas as requisições de dados subsequentes obtém o novo valor. Assim, é garantido o retorno do dado mais atualizado armazenado, logo após ter sido escrito. Isso independe de qual nó seja consultado pelo cliente – o dado retornado para a aplicação será igual.

Disponibilidade

Disponibilidade (**Available**)

O banco de dados sempre retornará um valor desde que ao menos um servidor esteja em execução – mesmo que seja um valor defasado.

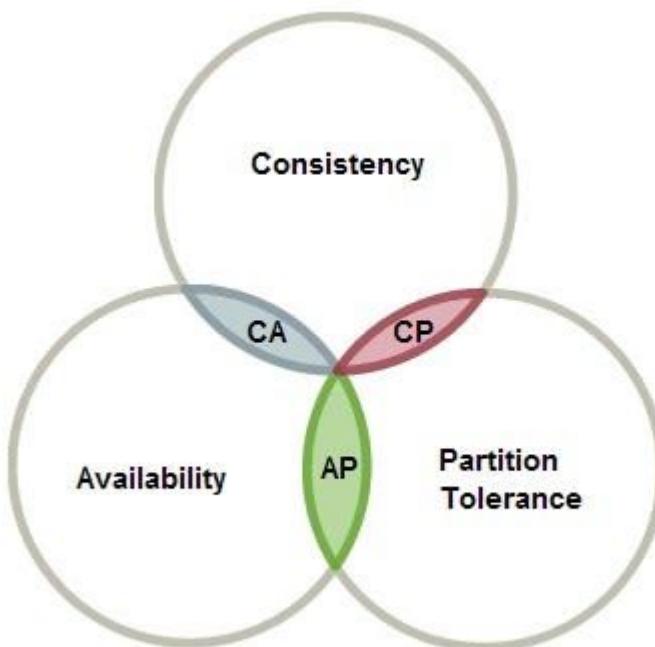
Tolerância ao Particionamento

Tolerância ao Particionamento (**Partition Tolerant**) ou Tolerante a Falhas.

O sistema ainda irá funcionar mesmo se a comunicação com o servidor for temporariamente perdida. Assim, se houver falha de comunicação com um nó da rede distribuída, os outros nós poderão responder às solicitações de consultas de dados dos clientes.

Teorema CAP

De acordo com o teorema **CAP**, *um sistema distribuído de bancos de dados somente pode operar com dois desses comportamentos ao mesmo tempo, mas jamais com os três simultaneamente.*

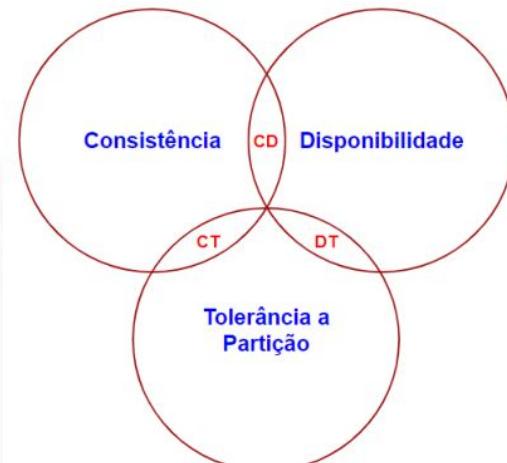


Consistência Eventual

É um conceito interessante derivado do teorema CAP.

O sistema prioriza as escritas de dados (armazenamento), sendo o sincronismo entre os nós do servidor realizado em um momento posterior – o que causa um pequeno intervalo de tempo no qual o sistema como um todo é inconsistente.

Para isso, são implementadas as propriedades **Disponibilidade** e **Tolerância a Partição**.



Consistência Eventual

Exemplos de sistemas de bancos de dados que implementam a consistência eventual são o ***MongoDB, Cassandra e RavenDB (bancos NoSQL)***, entre outros.

Em Bancos Relacionais, é muito comum implementar as propriedades **Consistência** e **Disponibilidade**. Como exemplos, citamos os SGBDRs ***Oracle, MySQL, PostgreSQL, SQL Server*** e outros.

Ao criar um banco de dados distribuído é importante ***ter em mente o teorema CAP***.

Você terá de decidir se o banco será consistente ou disponível, pois bancos de dados distribuídos são sempre tolerantes a partição.

Conclusão



Propriedades dos bancos de dados

- ✓ ACID
- ✓ BASE
- ✓ Teorema CAP

Próxima Aula



01. ••

NewSQL

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 01 – Aula 01.03 – Visão Geral de NewSQL

PROF.: RICARDO BRITO ALVES

Nesta Aula



- ❑ Definição e visão de NewSQL
- ❑ Exemplo de bancos de dados NewSQL

NewSQL

Pode ser definido como uma classe de SGBDs relacionais modernos que buscam fornecer o mesmo desempenho escalonável do NoSQL para cargas de trabalho OLTP e, simultaneamente, garantir a conformidade ACID para transações como no RDBMS.

Basicamente esses sistemas desejam alcançar a escalabilidade do NoSQL sem ter que descartar o modelo relacional com SQL e suporte a transações do DBMS legado.



Conceitos do NewSQL



- *Expansão dividindo um banco de dados em subconjuntos separados chamados partições ou fragmentos*, levando à execução de uma consulta em várias partições e, em seguida, combinar o resultado em um único resultado.
- Os novos sistemas SQL *preservam as propriedades ACID* dos bancos de dados.
- Benefícios do sistema de *controle de simultaneidade aprimorado* em relação aos tradicionais.

Conceitos do NewSQL



- *Presença de índice secundário permitindo NewSQL para suportar tempos de processamento de consulta mais rápidos.*
- Alta disponibilidade e durabilidade de dados forte são possíveis com o uso de *mecanismos de replicação*.
- Novos sistemas SQL podem ser configurados para fornecer *atualizações síncronas de dados* pela WAN.
- *Minimiza o tempo de inatividade, fornece tolerância a falhas com seu mecanismo de recuperação de falha.*

SQL, NoSQL ou NewSQL



- O SQL está em conformidade com as propriedades ACID e funciona bem com ***escalabilidade vertical***.
- O NoSQL oferece seu próprio ***escalonamento horizontal*** e está em conformidade com propriedades BASE. NoSQL, no entanto, não obedece às regras ACID, que são necessárias para manter um banco de dados confiável e consistente.
- As empresas e organizações em ritmo acelerado geram terabytes de dados transacionais diariamente enquanto trabalham em um sistema OLTP. NewSQL é a escolha ideal. ***NewSQL melhora o SQL ao fornecer escalabilidade horizontal enquanto mantém as propriedades ACID***.

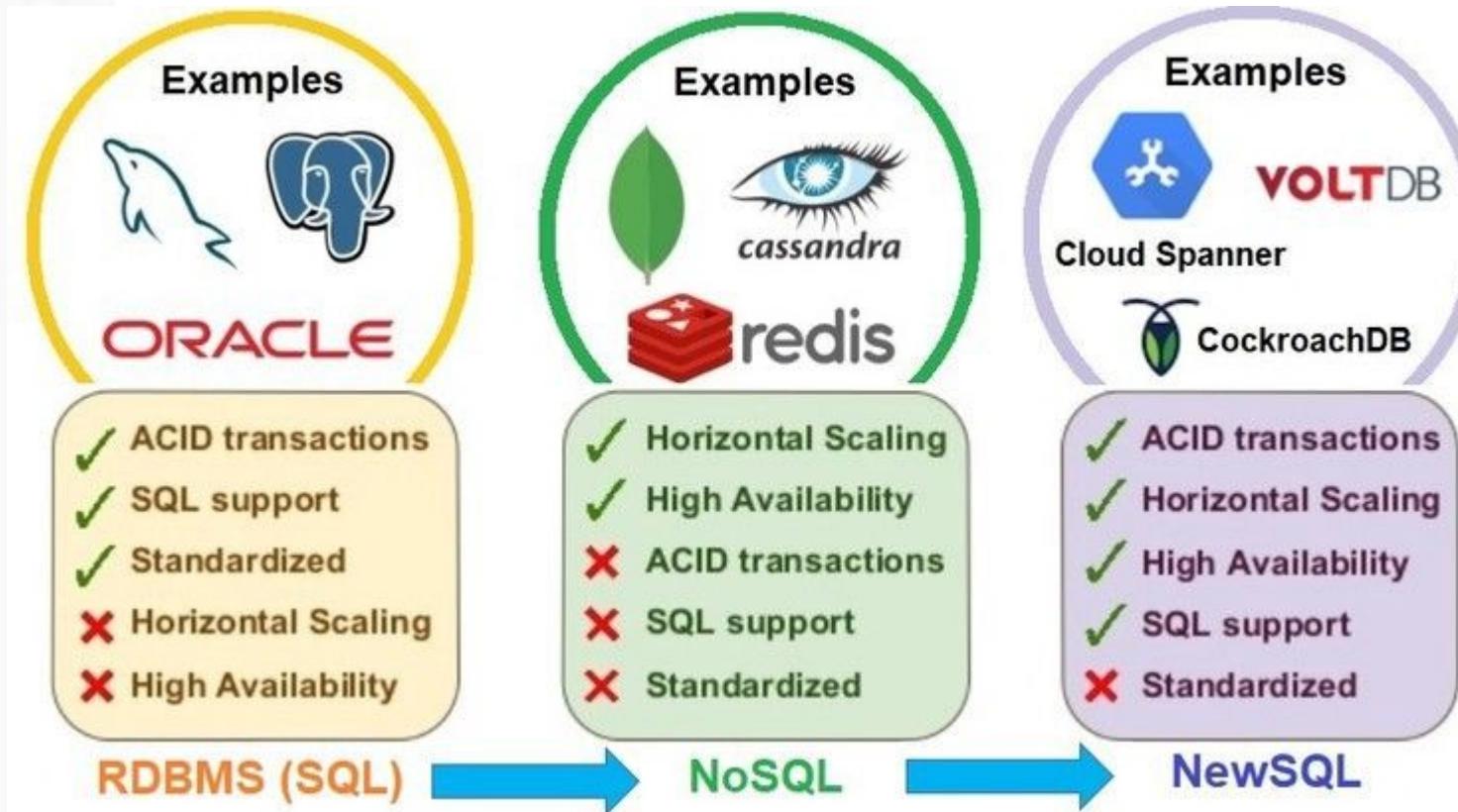
SQL, NoSQL ou NewSQL

IGTI

	Old SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL	Yes	No	Yes
ACID transactions	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Performance / big volume	No	Yes	Yes
Schema-less	No	Yes	No

SQL, NoSQL ou NewSQL

IGTI



Alguns Bancos NewSQL



MemSQL: Como o próprio nome sugere, é *operado em memória*, e é um sistema de banco de dados de alta escala por sua combinação de desempenho e compatibilidade com o SQL transacional e ACID na memória, adicionando uma interface relacional em uma camada de dados in-memory.

VoltDB: Projetado por vários pesquisadores de sistema de banco de dados bem conhecidos, esse banco *oferece a velocidade e a alta escalabilidade dos bancos de dados NoSQL, mas com garantias ACID*, e sua latência em milissegundo e *integração com Hadoop*.

SQLFire: Servidor de *banco de dados NewSQL da VMware*, desenvolvido para escalar em plataformas nas nuvens e tomar as vantagens de infraestrutura virtualizadas.

MariaDB: foi *desenvolvido pelo criador do MySQL* e é totalmente compatível com o MySQL. Também pode interagir com os bancos de dados NoSQL, como Cassandra e LevelDB.

Algunes Bancos NewSQL

IGTI



Conclusão



- ✓ Definição e visão de NewSQL
- ✓ Exemplo de bancos de dados NewSQL

Próxima Aula



01. ••

03. ••

Técnicas dos bancos NoSQL

02. ••

04. ••



Banco de Dados NoSQL

Capítulo 01 – Aula 01.04 – Técnicas em Bancos NoSQL

PROF.: RICARDO BRITO ALVES

Nesta Aula



- ❑ Técnicas utilizadas na implementação de bancos
NoSQL

Técnicas Utilizadas na Implementação



Sobre as principais características nos bancos de dados **NoSQL**, é importante *ressaltar algumas técnicas utilizadas para a implementação de suas funcionalidades*.

Entre elas estão:

- Map/reduce
- Consistent hashing
- MVCC - Multiversion Concurrency Control
- Vector Clocks

Map Reduce

Permite a manipulação de enormes volumes de dados ao longo de nós em uma rede.

Funciona da seguinte forma:

- Na fase **MAP**, os *problemas são particionados em pequenos problemas que são distribuídos em outros nós na rede.*
- Quando chegam à fase **REDUCE**, esses pequenos problemas *são resolvidos em cada nó filho e o resultado é passado para o pai*, que sendo ele consequentemente filho, repassaria para o seu, até chegar à raiz do problema.

O que é Hashing?

Hashing é o processo de mapear um dado, normalmente um objeto de tamanho arbitrário para outro dado de tamanho fixo, normalmente um inteiro, conhecido como código hash ou simplesmente hash. Uma função é geralmente usada para mapear objetos para código hash conhecido como função hash.

Por exemplo, uma função hash pode ser usada para mapear strings de tamanho aleatório para algum número fixo entre 0... N. Dado qualquer string, ela sempre tentará mapeá-la para qualquer número inteiro entre 0 e N. Suponha que N seja 100. Então, por exemplo, para qualquer string, a função hash sempre retornará um valor entre 0 e 100.

Hello ---> 60

Hello World ---> 40

Consistent Hashing

Hashing consistente é um tipo especial de hashing, que *quando uma tabela hash é redimensionada, apenas as chaves precisam ser remapeadas em média, onde é o número de chaves e é o número de slots.*

Suporta mecanismos de armazenamento e recuperação, onde a quantidade de sites está em constante mudança. *É interessante usar essa técnica, pois ela evita que haja uma grande migração de dados entre estes sites, que podem ser alocados ou desalocados para a distribuição dos dados.*

Distributed Hashing

Suponha que um número de funcionários continue crescendo e se torne difícil armazenar todas as informações dos funcionários em uma tabela hash que pode caber em um único computador. Nessa situação, tentaremos distribuir a tabela de hash para vários servidores para evitar a limitação de memória de um servidor. Os objetos (e suas chaves) são distribuídos entre vários servidores.

Este tipo de configuração é muito comum para caches na memória como Memcached, Redis etc.

Distributed Hashing

A chave de partição também não deve ser confundida com uma chave primária, é mais como um identificador único controlado pelo sistema.

NAME	AGE	CAR	GENDER
jim	36	camaro	M
carol	37	345s	F
johnny	12	supra	M
suzy	10	mustang	F

Distributed Hashing

O banco de dados atribui um valor hash a cada chave de partição:

PARTITION KEY	MURMUR3 HASH VALUE
jim	-2245462676723223822
carol	7723358927203680754
johnny	-6723372854036780875
suzy	1168604627387940318

Distributed Hashing

Cada nó no cluster é responsável por um intervalo de dados com base no valor de hash.

Valores hash em um cluster de quatro nós:

NODE	START RANGE	END RANGE	PARTITION KEY	HASH VALUE
1	-9223372036854775808	-4611686018427387904	johnny	-6723372854036780875
2	-4611686018427387903	-1	jim	-2245462676723223822
3	0	4611686018427387903	suzy	1168604627387940318
4	4611686018427387904	9223372036854775807	carol	772335892720368075

MVCC - Multiversion Concurrency Control



Oferece suporte a transações paralelas em banco de dados. Por não fazer uso de locks para controle de concorrência, faz com que transações de escrita e leitura sejam feitas simultaneamente.

Ao serem iniciados novos processos de leitura de um banco de dados, e nesse mesmo instante existir um outro processo que está atualizando, pode acontecer que o processo de leitura esteja lendo veja apenas uma parte do que está sendo atualizado, ou seja, um dado inconsistente.

Vector Clocks

Vector clocks: *Ordenam eventos que ocorreram em um sistema.* Como existe a possibilidade de várias operações estarem acontecendo simultaneamente, o uso de *um log de operações informando suas datas se faz importante para informar qual versão de um dado é a mais atual.*

Conclusão



Técnicas utilizadas na implementação de bancos NoSQL:

- ✓ Map/reduce
- ✓ Consistent hashing
- ✓ MVCC - Multiversion Concurrency Control
- ✓ Vector Clocks

Próxima Aula



01. ••

Tipo de Bancos NoSQL

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 01 – Aula 01.05 – Tipo de Bancos NoSQL

PROF.: RICARDO BRITO ALVES

Nesta Aula

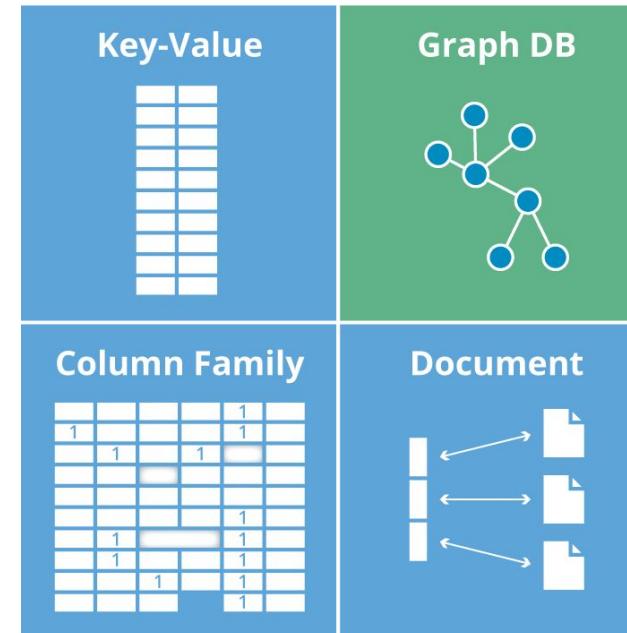


- ❑ Tipo de banco de dados NoSQL

Modelos de NoSQL

Temos quatro categorias do NoSQL:

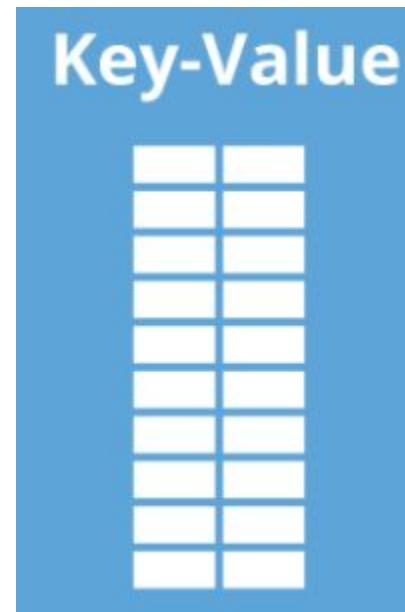
- Chave-valor (key-Value)
- Orientado a Grafos
- Orientado a Coluna (Column Family)
- Orientado a Documentos



Chave-Valor (Key-Value)

- Modelo mais simples.
- Permite a visualização do banco como uma grande tabela.
- Todo o banco é composto por um conjunto de chaves que estão associadas a um único valor.

Chave (Campo)	Valor (Instancia)
Nome	Hélio Rodrigues
Idade	45
Sexo	Masculino
Fone	99 99999999



Chave-Valor (Key-Value)

É um modelo considerado simples e permite a sua visualização através de uma tabela de hash, no qual há uma chave única e um indicador de determinado dado, podendo ser uma String ou um binário.

Este modelo é caracterizado pela sua facilidade ao ser implementado, permitindo que os dados sejam acessados rapidamente através da chave, aumentando também a disponibilidade do acesso aos dados.

Para manipulá-los, utilizamos comandos simples como `get()` e `set()`, que retornam e capturam valores.

Um problema enfrentado por este tipo de banco de dados é *que não permite a recuperação de objetos através de consultas mais complexas*.

Como exemplo, podemos citar o Dynamo que foi desenvolvido pela Amazon.

Orientado a Grafos

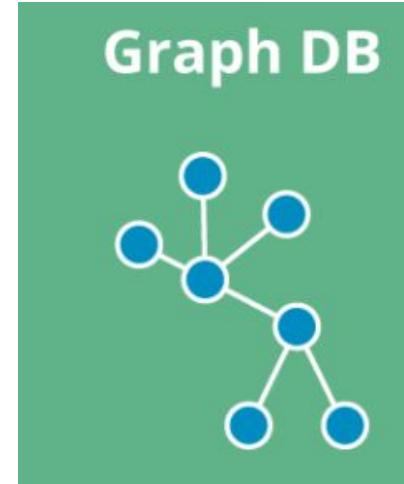
Este modelo possui três componentes básicos:

- Nós (vértices dos grafos).
- Os relacionamentos (arestas).
- As propriedades (conhecidos também como atributos).

É visto como um multigrafo rotulado e direcionado, onde cada par de nós **pode ser conectado por mais de uma aresta.**

A utilização deste modelo é muito útil quando é necessário fazer consultas demasiadamente complexas.

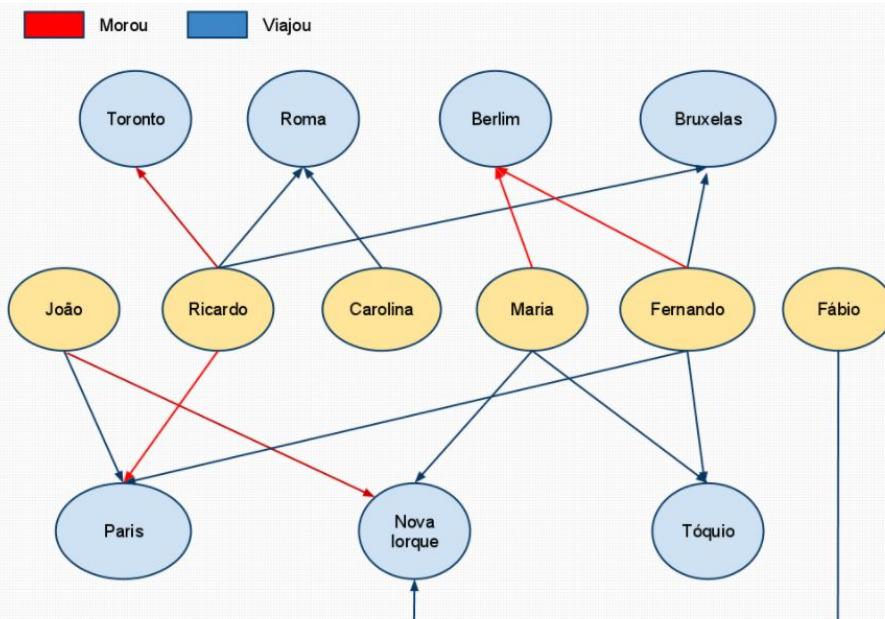
O modelo orientado a grafos possui uma alta performance, permitindo um bom desempenho nas aplicações.



Orientado a Grafos

Imagine uma aplicação que mantêm informações relativas à viagem. Uma consulta pertinente seria: “Quais cidades foram visitadas anteriormente por pessoas que foram para Nova Iorque?”

Temos diversas pessoas: João, Ricardo, Carolina, Maria, Fernando e Fábio que representam nós do grafo e estão conectadas a cidades que visitaram ou residiram.



Orientado a Grafos

O Neo4J trata-se de um banco de dados baseado em grafos desenvolvido em Java.

Além de possuir suporte completo para transactions, ele também trabalha com nós e relacionamentos.



Orientado a Colunas

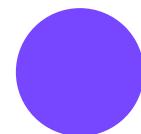
Este tipo de banco de dados *foi criado para armazenar e processar uma grande quantidade de dados distribuídos em diversas máquinas.*

Aqui existem *as chaves, mas neste caso, elas apontam para atributos ou colunas múltiplas.*

Os dados são indexados por uma tripla (coluna, linha e timestamp), *a coluna e linha são identificadas por chaves e o timestamp permite diferenciar múltiplas versões de um mesmo dado.*

Como o próprio nome sugere, as colunas *são organizadas por família da coluna.* Demonstra maior complexidade que o de chave-valor.

Column Family			
1	1	1	1
1	1	1	
			1
1		1	1
1		1	1
	1	1	1
			1



Orientado a Colunas

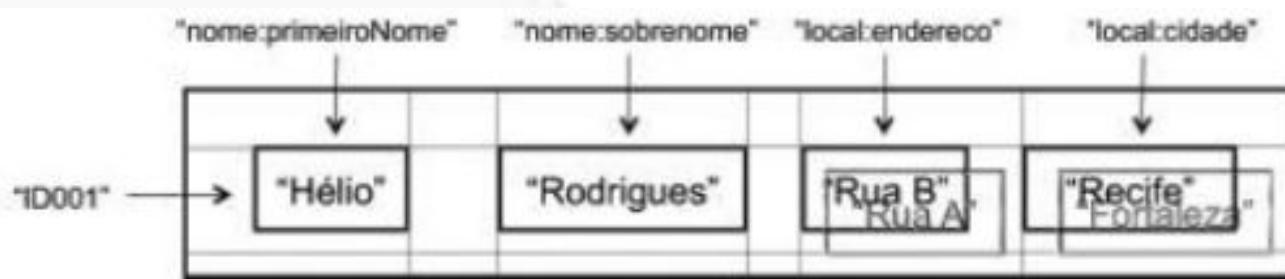
Vale destacar que as operações de escrita e leitura são atômicas, ou seja, os valores associados a uma linha são considerados em sua execução, independente das colunas que estão sendo lidas/escritas.

O conceito associado a este modelo é o de família de colunas, com o objetivo de reunir colunas que armazenam o mesmo tipo de informação.

Orientado a Colunas

Podemos modelar o conceito de amigos, onde o **primeiro nome** e **sobrenome** são colunas pertencentes à família de colunas denominada “**nome**”. Da mesma forma, as colunas **endereço**, **cidade** pertencem à **família local**.

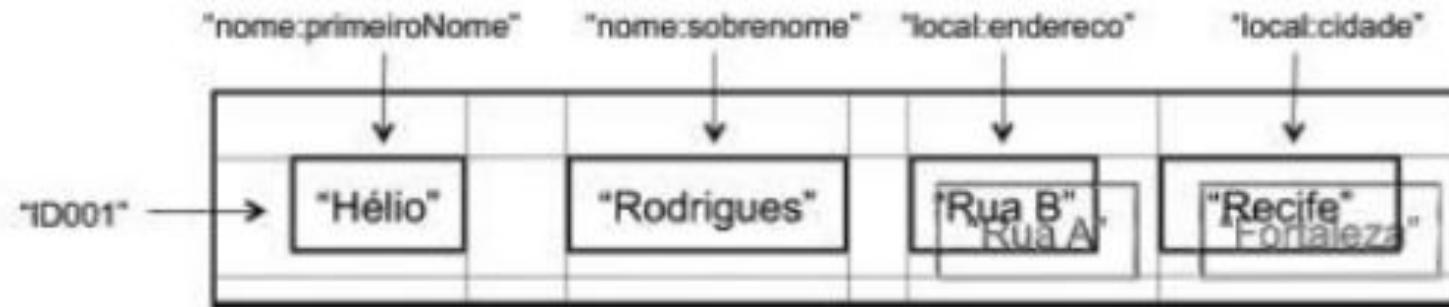
É interessante observar que na linha 001 a pessoa tem diversos endereços. Como a busca neste tipo de banco de dados é atômica, mesmo que o interesse seja buscar o primeiro nome da linha 001, todas as colunas serão retornadas quando esta mesma linha for consultada.



Orientado a Colunas

Este modelo permite ainda o particionamento de dados, oferecendo forte consistência, ***no entanto, a alta disponibilidade é o seu ponto fraco.***

Este modelo de dados surgiu com o BigTable criado pelo Google. Além do BigTable temos também o Cassandra que foi desenvolvido pelo Facebook.

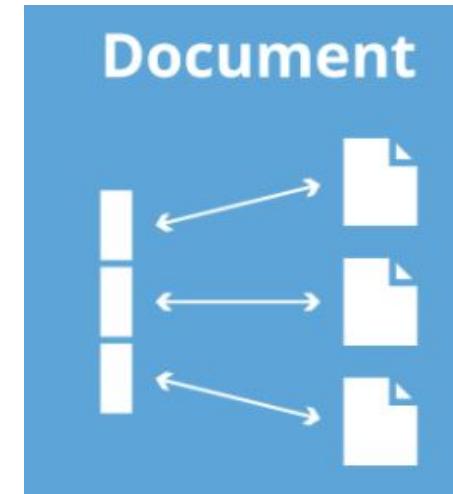


Orientado a Documentos

Como o próprio nome sugere, **este modelo armazena coleções e documentos**.

Um documento é um objeto identificador único e um conjunto de campos que podem ser strings, listas ou documentos aninhados.

Diferente do banco de dados chave-valor onde se cria uma única tabela hash, **neste modelo temos um agrupamento de documentos**, sendo que em cada um destes documentos temos um conjunto de campos e o valor destes campos.



Orientado a Documentos

Neste modelo temos ausência de esquema pré-definido (schema free).

Portanto é possível que haja alterações nos documentos, com a adição de novos campos, por exemplo, sem afetar adversamente outros documentos.

Outra característica interessante é que não é necessário armazenar valores de dados vazios para campos que não possuem um valor.

ID: P001
Assunto: "Eu gosto
Autor: "Hélio"
Data: "27/01/2011
Tags: ["laranjas", "s
Mensagem: "Hoje e

ID: P002
Assunto: "Eu gosto de tomates"
Autor: "Mara"
Data: 15/05/2012
Tags: ["tomate", "suco", "plantas"]
Mensagem: "Hoje eu estou com vontade de tomar suco de tomate."

Orientado a Documentos

Como exemplo de sistema de banco de dados que utiliza este tipo de solução destacamos o **CouchDB** e o **MongoDB**. O CouchDB utiliza o formato JSON e é implementado em Java. Já o mongo usa um formato semelhante ao JSON e é implementado em C++, **permitindo tanto concorrência quanto replicação**.

JSON

É um acrônimo de **JavaScript Object Notation**, e tem um formato compacto, de padrão aberto independente, de troca de dados simples e rápida entre sistemas, especificado por Douglas Crockford em 2000, que utiliza texto legível a humanos, no formato atributo-valor.

```
1 | {  
2 |     "id":1,  
3 |     "nome":"Alexandre Gama",  
4 |     "endereco":"R. Qualquer"  
5 | }
```

Modelos de NoSQL

Nenhum modelo pode ser considerado superior a outro. Um modelo pode ser mais adequado para ser utilizado em certas situações.

- Para a utilização de um banco de dados de manipulação de dados que frequentemente serão escritos, mas não lidos (um contador de hits na Web, por exemplo), pode ser usado um banco de dados orientado a documento como o **MongoDB**.
- Já aplicativos que demandam alta disponibilidade, onde a minimização da atividade é essencial, podemos utilizar um modelo orientado a colunas como o **Cassandra**.
- Aplicações que exigem um alto desempenho em consultas com muitos agrupamentos podem utilizar um modelo orientado a **grafos**.

Conclusão



Tipo de banco de dados NoSQL:

- ✓ Chave-valor (key-Value)
- ✓ Orientado a Grafos
- ✓ Orientado a Coluna (Column Family)
- ✓ Orientado a Documentos

Próxima Aula



01. •

Motivações no uso de bancos
NoSQL

02. •

03. •

04. •

Banco de Dados NoSQL

Capítulo 01 – Aula 01.06 – Motivações no uso de NoSQL

PROF.: RICARDO BRITO ALVES

Nesta Aula



- ❑ Motivações no uso de banco de dados NoSQL

NoSQL- Motivações



Hoje as empresas estão adotando NoSQL para um número crescente de cenários.

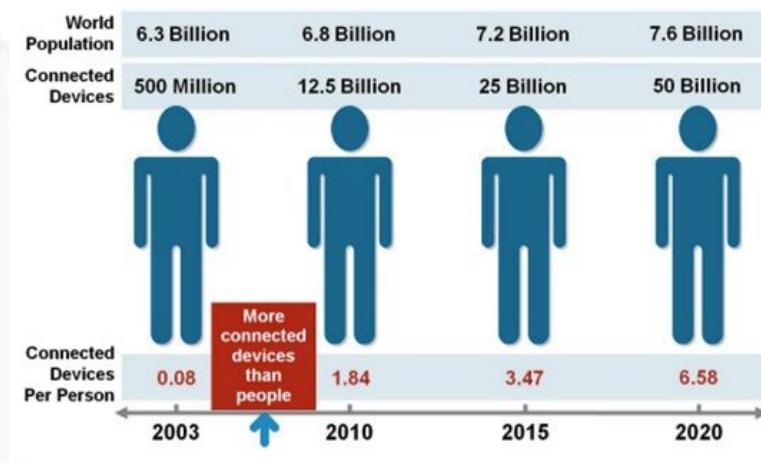
A escolha que é impulsionada por quatro tendências relacionadas :

- Big Users
- Big Data
- Internet das coisas
- Cloud Computing

NoSQL- Big Users

O crescente uso de aplicativos online resultou em um número crescente de operações de banco de dados e uma necessidade de uma maneira mais fácil de escalar bancos de dados para atender a essas demandas.

Um grande número de usuários, combinados com a natureza dinâmica dos padrões de uso está demandando uma tecnologia de banco de dados mais facilmente escalável.



NoSQL- Big Data



Big Data é a área do conhecimento que estuda como tratar, analisar e obter informações a partir de conjuntos de dados demasiadamente grandes para serem analisados por sistemas tradicionais.

Razões para usar o Big Data:

- Entender padrões;
- Prever situações;
- Criar fronteiras;
- Informar coleções de dados;
- Estimar parâmetros escondidos;
- Calibrar.

NoSQL- Big Data



Big Data é um conceito que **descreve o grande volume de dados estruturados e não estruturados** que são gerados a cada segundo.

O diferencial do Big Data está justamente atrelado à **possibilidade e oportunidade em cruzar esses dados por meio de diversas fontes** para obtermos insights rápidos e preciosos. A exigência dos consumidores e o aumento da competitividade em todos os mercados nos força a inovar e ter esse caminho como premissa básica nos negócios.



NoSQL- Tipos de Big Data

IGTI

Types of Big Data

Structured

1001 1010	1001 0101	1100 0110
0011 1100	0110 1001	0011 1010
0011 0011	0101 1100	1001 1001

Unstructured



Semi-Structured



SelectHub

NoSQL- IoT - Internet das Coisas



O termo foi utilizado pela primeira vez em 1999 para descrever um sistema onde os objetos poderiam ser conectados à internet.

Antigamente a internet ligava apenas computadores a computadores de uso exclusivo militar e de algumas poucas faculdades enquanto hoje em dia até um relógio pode ter acesso à internet. Além da necessidade de muito mais endereços de IP(IPv4 para IPv6 – 128 bits), essa revolução na internet traz muitas outras características pro meio.

O fato de ter tantos dispositivos conectados à rede literalmente tem revolucionado o modo como vivemos.

NoSQL- IoT - Internet das Coisas



- 32 bilhões de coisas vão estar conectadas a internet.
- 10% de todos os dados serão gerados por sistemas embarcados (versus 2% atualmente).
- 21% dos mais valiosos dados serão gerados por sistemas embarcados (versus 8% atualmente).
- Dados de telemetria - semi-estruturados e contínuos - representam um desafio para bancos de dados relacionais, que exigem um esquema fixo e dados estruturados.

Cidades Inteligentes

IGTI

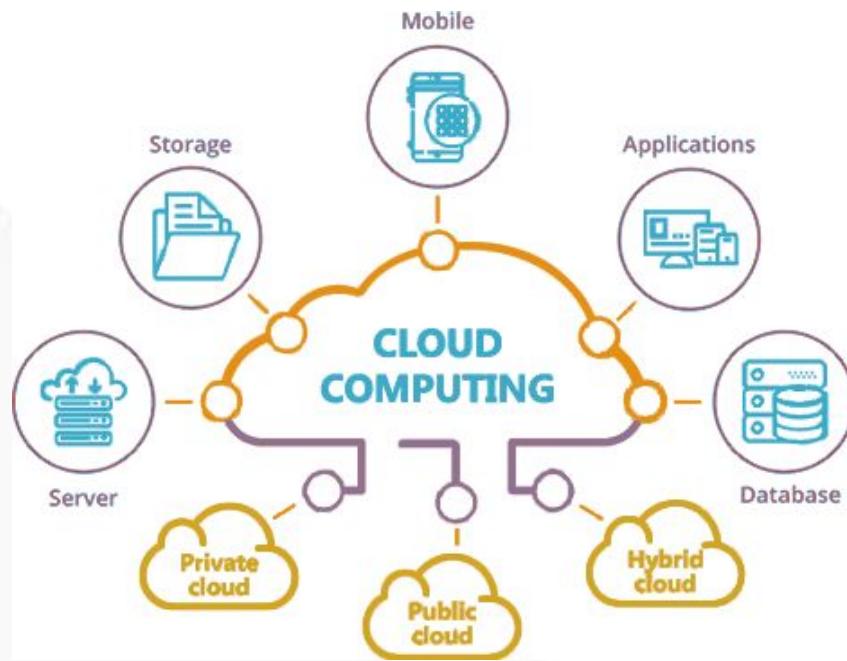
- Poluição Sonora.
- Otimizar coleta de lixo.
- Controle de tráfego.
- Controle de distribuição de energia elétrica.
- Segurança pública.



NoSQL- Cloud Computing



Computação em nuvem, é um termo coloquial para a disponibilidade sob demanda de recursos do sistema de computador, especialmente armazenamento de dados e capacidade de computação, sem o gerenciamento ativo direto do utilizador.



Conclusão



- ✓ Motivações no uso de bancos de dados NoSQL

Próxima Aula



01. ••

Principais bancos de dados
NoSQL

03. ••

02. ••

04. ••

Bancos de Dados NoSQL

Capítulo 02 – Principais Banco de Dados NoSQL

PROF.: RICARDO BRITO ALVES



Banco de Dados NoSQL

Capítulo 02 – Aula 02.01 – Introdução ao Apache Cassandra

PROF.: RICARDO BRITO ALVES

Nesta Aula



- ❑ Apache Cassandra

Colunar

Para mostrar um banco de dados colunar, precisamos relembrar dos bancos relacionais: eles armazenam os dados em linhas (rows) enquanto os bancos colunares persistem os dados por colunas criando um relacionamento através de um id.

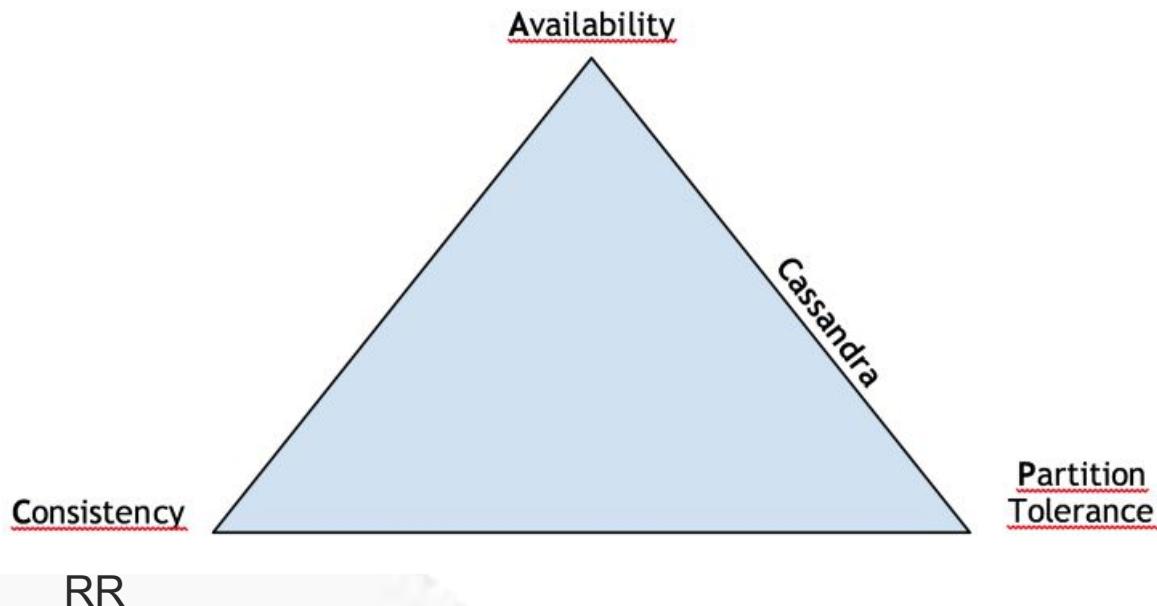
Banco de Dados Relacional:

<u>sku_produto</u>	<u>nome_produto</u>	<u>preco_produto</u>
1	Tênis	100
2	Bola	50
3	Camisa	200
4	Bike	900

Banco de Dados Colunar:

<u>sku_produto</u>		<u>nome_produto</u>		<u>preco_produto</u>	
<u>id</u>	<u>value</u>	<u>id</u>	<u>value</u>	<u>id</u>	<u>value</u>
0	1	0	Tênis	0	100
1	2	1	Bola	1	50
2	3	2	Camisa	2	200
3	4	3	Bike	3	900

Teorema CAP



Resolvendo conflitos de forma transparente para os clients.

Apache Cassandra



Apache Cassandra é um projeto de sistema de banco de dados distribuído altamente escalável de segunda geração, que reúne a arquitetura do DynamoDB, da Amazon Web Services e modelo de dados baseado no BigTable, do Google.

Instalação Apache Cassandra



Faça o download da última versão no site oficial:

<http://incubator.apache.org/cassandra/>

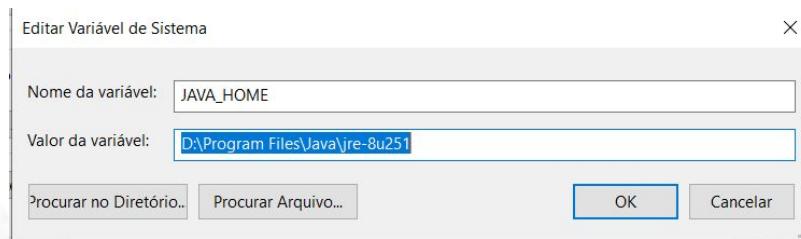
<https://downloads.apache.org/cassandra/3.11.9/apache-cassandra-3.11.9-bin.tar.gz>

- objetivo é fazer uma instalação bem simples para interação com o banco. Feito o download, descompacte o arquivo, os diretórios mais importantes são:
 - bin: contém os scripts de interação com o Cassandra.
 - data: contém os dados que o Cassandra armazena.
 - conf: contém os yml's de configuração do banco.

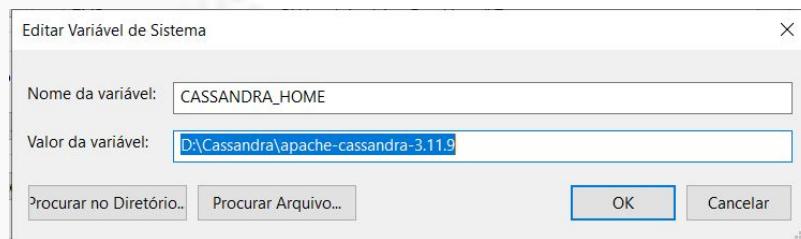
Instalação Apache Cassandra



A premissa é o ter o Java SDK ou JRE instalado na máquina. Como o Cassandra é feito em Java, o ideal é que você tenha a última versão do JRE instalado na sua máquina. Certifique que a variável de ambiente de sistema JAVA_HOME aponte para a instalação java da sua máquina.



Crie uma variável de ambiente chamada CASSANDRA_HOME e aponte para o seu diretório de instalação do Cassandra.



Rodando Apache Cassandra



Abra o command como administrador, acesse o diretório /bin onde
foi descompactado o Cassandra(cd\cassandra\apache-cassandra-3.11.9\bin). No
Windows é preciso liberar os comandos powershell no command.

- powershell Set-ExecutionPolicy Unrestricted
- cassandra.bat –f

Enquanto essa janela do command estiver aberta, o Cassandra
estará rodando localmente em apenas um nó e por default ele sobe
na porta: 9042.

Cqsh - Apache Cassandra



Neste momento vamos utilizar o **cqlsh**, é um sistema de command line que nos permite interagir com o Cassandra.

Abra uma nova janela do command como administrador, acesse o diretório /bin onde foi descompactado o Cassandra (cd\cassandra\apache-cassandra-3.11.9\bin) e execute os seguintes comandos:

- powershell Set-ExecutionPolicy Unrestricted
- cqlsh.bat

```
cqlsh:teste> insert into teste.posts (tag, name, author, description, likes)
... values
... ('apache-cassandra','Cassandra post','Jose','post do cassandra',1);
cqlsh:teste> update posts set likes = 2
... where tag = 'apache-cassandra' and name = 'Cassandra post';
cqlsh:teste> select * from posts;

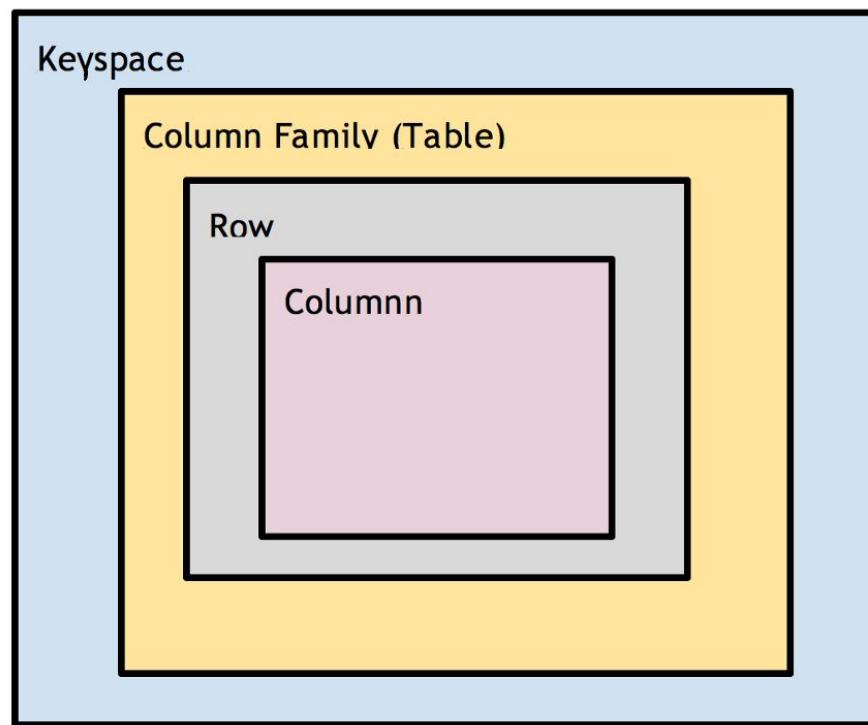
tag          | name        | author | description      | likes
-----+-----+-----+-----+-----+
apache-cassandra | Cassandra post | Jose | post do cassandra | 2

(1 rows)
cqlsh:teste>
```

Arquitetura do Cassandra

IGTI

O Cassandra é segmentado em keyspaces, tables (column families), rows e columns.



Keyspace

É o equivalente a um banco de dados no mundo relacional e agrupa as tabelas do sistema. Suas principais configurações são:

- Replication factor: O fator de replicação define quantas “cópias” existirão dos dados. Caso tenha um cluster com 5 máquinas e o replication factor igual a 2, o Cassandra irá garantir que seus dados estarão em pelo menos 2 máquinas. Essa feature é bem importante para alta disponibilidade do banco.
- Replica placement strategy: É a estratégia de replicação dos dados, existem 2 estratégias, sendo:
 - ✓ SimpleStrategy: Respeitando o fator de replicação o Cassandra simplesmente replica o dado para o próximo nó (node) do cluster.
 - ✓ NetworkTopologyStrategy: Com essa estratégia o Cassandra irá persistir os dados em mais de um datacenter (obrigatoriamente o cluster deve estar configurado para ser multi-datacenter).

Tables (Column family)

É o equivalente à uma tabela no mundo relacional.

Um keyspace pode conter “N” column families, que por sua vez podem conter “N” rows.

A comparação entre column families e tabelas do banco relacional é apenas para facilitar o entendimento.

Vale ressaltar que o comportamento é bem diferente, como por exemplo: JOIN`s não são suportados.

Tables (Column family)

Nas columns families são definidos os seguintes atributos:

- **Primary key:** A primary key é composta de 2 partes:
 - ✓ Partition key: Ela define qual partição será armazenado o dado.
 - ✓ Clustering key: É o restante da chave, esse campo não entra na definição das partições.

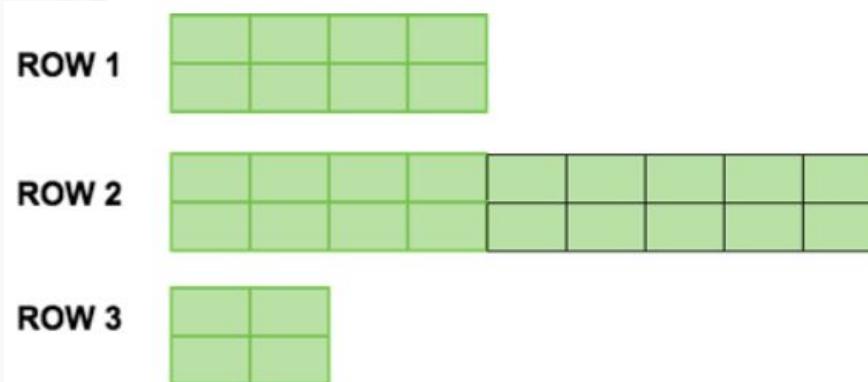
Obs: É possível definir “N” campos como partition key — a junção deles irá definir a partição.

- **Columns:** Representam as demais colunas da tabela, lista de tipos suportados na versão.

Row

As rows são compostas pela Primary key e um conjunto de columns.

Vale ressaltar que o Cassandra persiste apenas os campos que contém valores, diferente dos bancos relacionais onde são alocados recursos para campos nulos. Isso significa que as rows podem conter colunas diferentes.



Column

A column é composta por basicamente 3 campos:

- Column key: Nome da coluna.
- Column value: É o valor que está sendo persistido.
- Timestamp: O Cassandra utiliza esse campo para resolver conflitos e determinar qual é o valor mais atual.

Conclusão



- ✓ Apache Cassandra

Próxima Aula



01. ••

Introdução ao Redis

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 02 – Aula 02.02 – Introdução ao Redis

PROF.: RICARDO BRITO ALVES

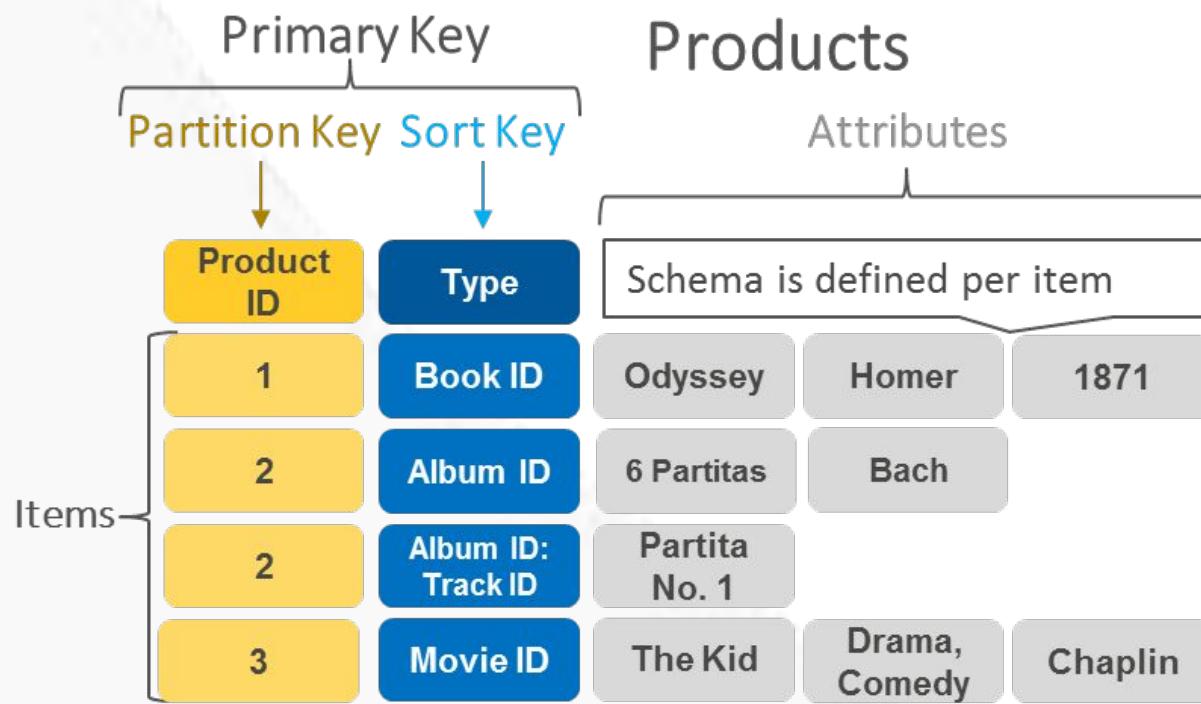
Nesta Aula



- ❑ Redis

Banco Chave-Valor

Banco de dados que trabalham no esquema Chave-Valor (Key-Value) são sistemas distribuídos, também conhecidos como tabelas de hash distribuídas, armazenam objetos indexados por chaves, e possibilitam a busca por esses objetos a partir de suas chaves.



Redis



Redis (Remote Dictionary Server) é um banco de dados de memória e de código aberto que é usado como cache e como intermediário de mensagens. Ele também é conhecido como um servidor de dados estruturados.

Oficialmente o Redis não tem uma distribuição para Windows, porém é possível encontrar uma distribuição paralela no GitHub.

Baixe o Redis do seu site oficial: <https://redis.io/>.

Redis

O Redis é um armazenamento de estrutura de dados de chave-valor de código aberto e na memória. O Redis oferece um conjunto de estruturas versáteis de dados na memória que permite a fácil criação de várias aplicações personalizadas. Os principais casos de uso do Redis incluem cache, gerenciamento de sessões, PUB/SUB e classificações. É o armazenamento de chave-valor mais conhecido atualmente.

- Desempenho muito rápido
- Estruturas de dados na memória
- Versatilidade e facilidade de uso
- Replicação e persistência
- Compatibilidade com as linguagens Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, R, Go e muitas outras.

Redis - Comandos



<https://redis.io/commands>

Set: set name “José”

Get: get name

Keys*: busca todas as Keys setadas

del key: del name

Flushall: deleta todas as keys

Conclusão



- ✓ Redis

Próxima Aula



01. ••

Introdução ao Neo4j

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 02 – Aula 02.03 – Introdução ao Neo4j

PROF.: RICARDO BRITO ALVES

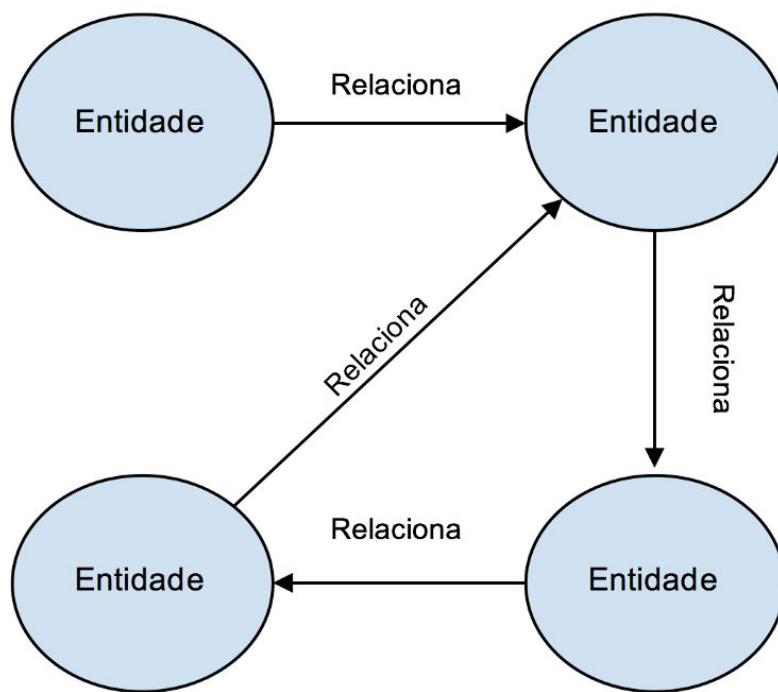
Nesta Aula



- Neo4j

Grafos

Os dados são persistidos em um esquema de grafo, onde um registro “aponta” para o próximo.



Neo4J



Neo4j é um sistema de gerenciamento de banco de dados gráfico desenvolvido pela Neo4j, Inc. Descrito por seus desenvolvedores como um banco de dados transacional compatível com ACID com armazenamento e processamento de gráfico nativo, Neo4j está disponível em uma "edição da comunidade" de código aberto.

O Neo4j é implementado em Java e acessível a partir de softwares escritos em outras linguagens usando a linguagem de consulta Cypher através de um endpoint HTTP transacional ou através do protocolo binário "bolt".

Neo4J



O Neo4J tem como prioridade, tratar seus relacionamentos da melhor forma possível, isso quer dizer que, na medida que os relacionamentos entre “nós” aumentam, sua capacidade de processamento continua estável, diferentemente de bancos de dados tradicionais.

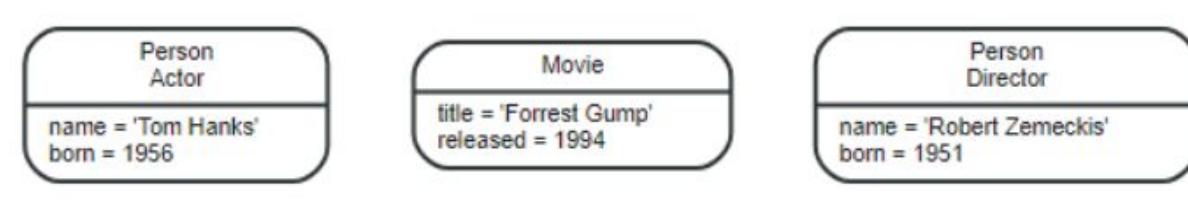
A linguagem do Neo4J é a **cypherQuery**, e a curva de aprendizagem não é tão grande, um case de sucesso é o facebook, que aderiu o sistema de grafos em um dos seus módulos, pois encontrou um desempenho maior para criar relacionamentos dentro da plataforma.

Neo4J

IGTI

Labels: É uma espécie de template para os nós, a grosso modo, seria uma classe. Na imagem, temos as labels:

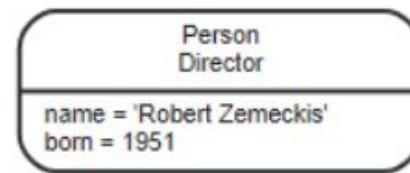
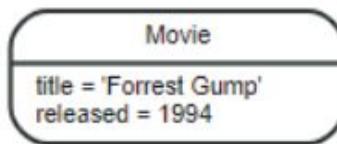
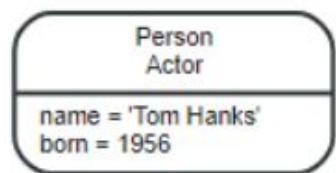
PersonActor, Movie e PersonDirector.



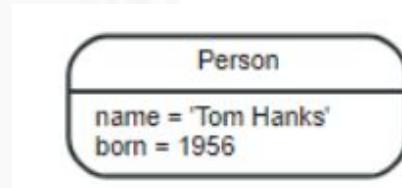
Neo4J

IGTI

Propriedades (Properties): são parâmetros baseados em chave-valor, para designar um nó, ou uma relacionamento. Na imagem há varias propriedades, tais como: name, born, title e released, eles demonstram características ou descrições de uma label.



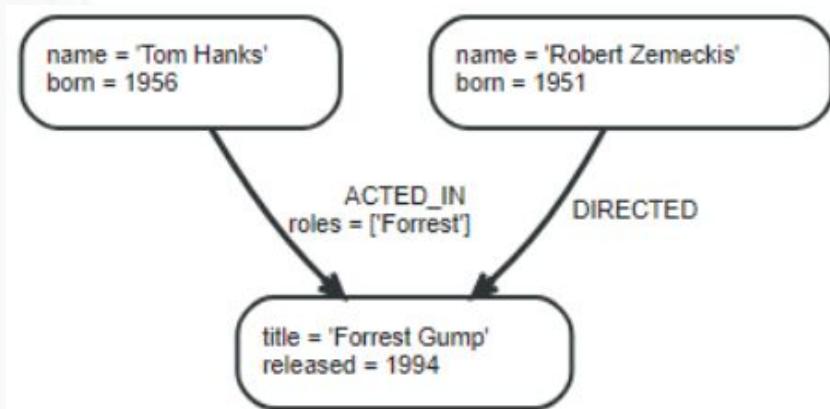
Instâncias: São como entidades criadas baseadas em um template pronto (label), se parecem instâncias de objetos de uma classe.



Neo4J

IGTI

Relacionamentos: Representa uma conexão entre nós, é possível criar propriedades dentro de um relacionamento. No exemplo abaixo, há dois relacionamentos, o primeiro se chama ACTED_IN, que possui uma propriedade chamada roles, o segundo relacionamento se chama DIRECTED, que não possui propriedades.



Instalação do Neo4J



Faça o download da versão que desejar no Neo4j Download Center.

<https://neo4j.com/download-center/>

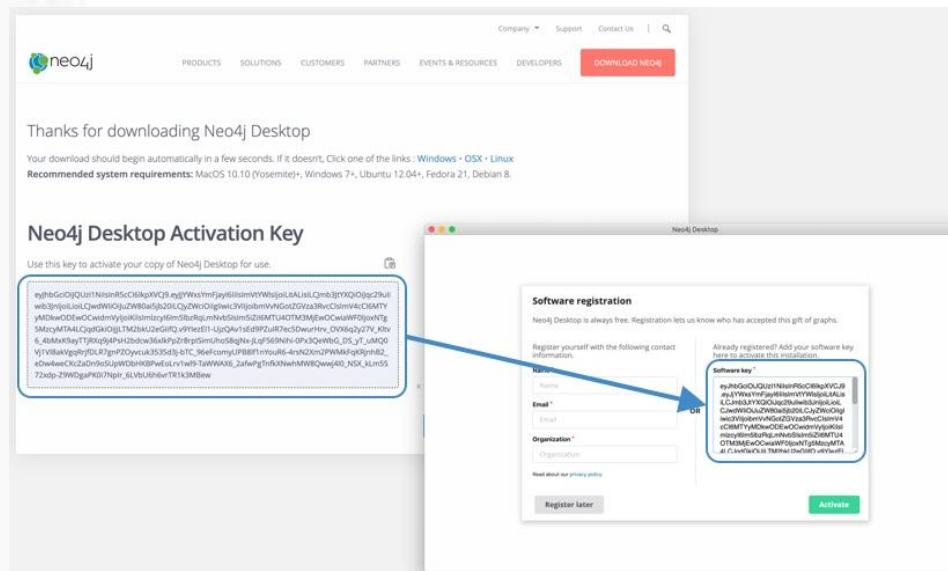
Current Releases

Enterprise Server	Community Server	Neo4j Desktop
Current Release		

Neo4J

IGTI

Se você estiver abrindo o Neo4j Desktop pela primeira vez, ele deverá solicitar que você registre o software com uma chave de ativação. Esta chave de ativação é gerada quando você faz o download do Neo4j Desktop pela primeira vez e será exibida na página de confirmação do download. Mantenha-o em um lugar seguro.

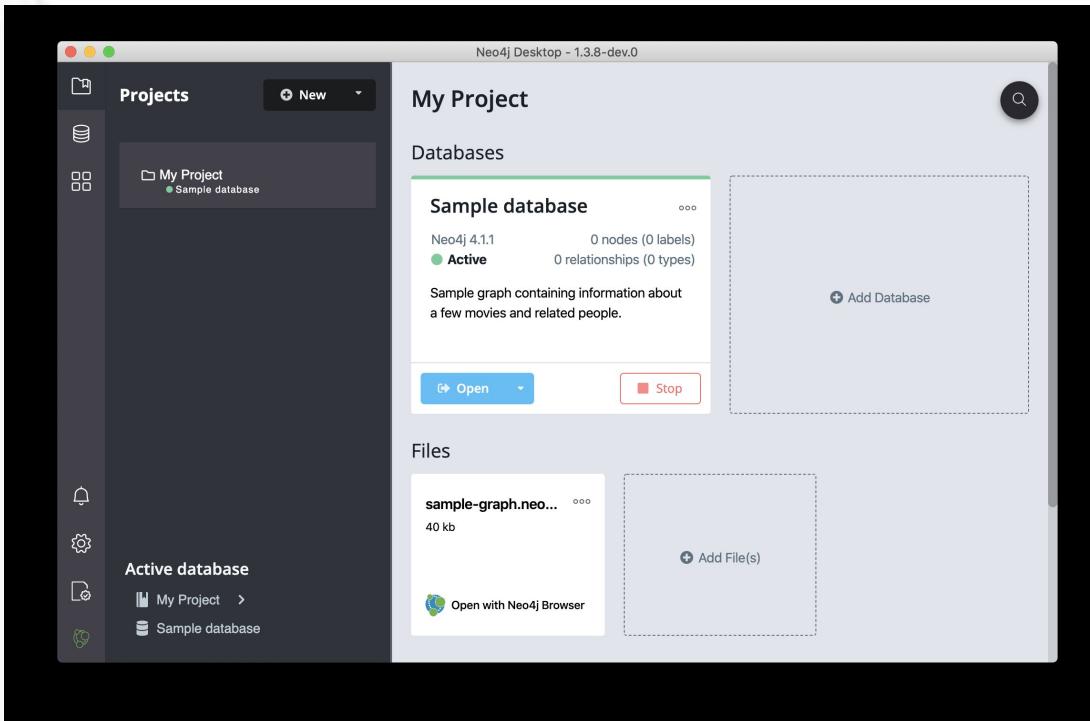


Neo4J

IGTI

Como um novo usuário do Neo4j Desktop, você será saudado com um novo banco de dados de amostra contendo Filmes e Atores.

<https://neo4j.com/developer/neo4j-desktop/>



Neo4J



É possível perceber a diferença entre os bancos relacionais e os bancos em grafo no que diz respeito à recuperação de informação, ao comparar como a mesma consulta é escrita para cada sistema.

Considere uma consulta que retorna todos os usuários que curtem a página com o nome “The Beatles”.

```
1  SELECT u.name, u.gender, u.age
2      FROM User u, Likes c, Page p
3      WHERE p.name = "The Beatles"
4          AND p.ID = c.ID_Page
5          AND c.ID_User = u.ID
```

[graph_rdbms.sql](#) hosted with ❤ by [GitHub](#)

[view raw](#)

```
1  MATCH (users) -[:LIKES]-> (:Page {name:"The Beatles"})
2  RETURN users
```

[graph_rdbms.cql](#) hosted with ❤ by [GitHub](#)

[view raw](#)

Neo4J



As palavras chave mais importantes do **Cypher** são:

MATCH: Busca no grafo, dados que correspondem a um padrão fornecido

WHERE: Filtra os resultados com predicados, como no SQL.

RETURN: Informa qual o retorno da sua consulta

CREATE: Cria elementos (nodes e relacionamentos) com rótulos e propriedades.

MERGE: É uma combinação de MATCH e CREATE.

Conclusão



- ✓ Neo4j

01. •

Introdução ao MongoDB

02. •

03. •

04. •



Banco de Dados NoSQL

Capítulo 02 – Aula 02.04 – Introdução ao MongoDB

PROF.: RICARDO BRITO ALVES

Nesta Aula

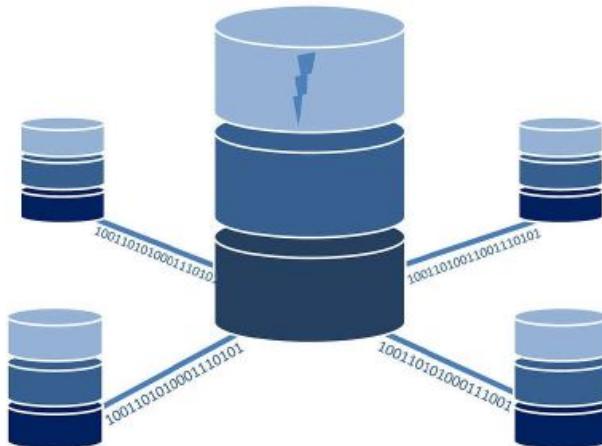


- ❑ MongoDB

NoSQL – Banco de Dados de Documentos



- Armazenam chave/valor.
- O valor é um documento estruturado e indexado, com metadados.
- Valor (Documento), pode ser consultado.
- JSON: JavaScript Object Notation.
 - Feito para troca de dados.
 - Mais compacto e legível que XML.



NoSQL - JSON

“ Estrutura nome/valor, entre aspas duplas



Dados separados por vírgula

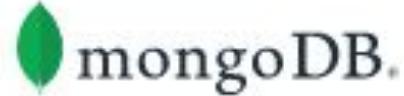


Chaves separam objetos



Vetores entre colchetes

MongoDB



MongoDB é um software de banco de dados orientado a documentos, de código aberto e multiplataforma, escrito na linguagem C++.

Classificado como um programa de banco de dados NoSQL, o MongoDB usa documentos semelhantes a JSON com esquemas.

MongoDB

- ✓ Open source.
- ✓ Multiplataforma.
- ✓ Escalável.



Relacional

Banco de Dados

Tabela

Linha

Coluna

MongoDB

Banco de Dados

Coleção

Documento

Campo

IGTI

Instalação MongoDB

Baixar a versão free Community no site:

<https://www.mongodb.com/try/download/community>

Windows: execute o instalador do MongoDB - clique duas vezes no arquivo “.msi” baixado.

Siga o assistente de instalação do MongoDB Community Edition. O assistente o orienta durante a instalação do MongoDB e MongoDB Compass.

Configuração de Serviço: à partir do MongoDB 4.0, você pode configurar o MongoDB como um serviço do Windows durante a instalação ou apenas instalar os arquivos e os iniciar.

MongoDB Compass



Interface gráfica do MongoDB.

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar with connection details: HOST localhost:27017, CLUSTER Standalone, and EDITION MongoDB 4.4.0 Enterprise. Below this is a search bar and a list of databases: DBElec, DBElections, DBEleicao, admin, config, and local. The main area displays a table of databases with columns: Database Name, Storage Size, Collections, and Indexes. Each database row has a trash icon in the last column.

Database Name	Storage Size	Collections	Indexes
DBElec	28.0KB	3	3
DBElections	4.0KB	1	1
DBEleicao	20.0KB	1	1
admin	20.0KB	0	1
config	24.0KB	0	2
local	44.0KB	1	1

Instalação MongoDB



MongoDB Service MongoDB

The following installs and configures MongoDB as a Windows service.

Starting in MongoDB 4.0, you can configure and start MongoDB as a Windows service during the install, and the MongoDB service is started upon successful installation.

This screenshot shows the 'Service Configuration' dialog box for MongoDB 4.0. The title bar reads 'MongoDB 4.0.0 2008R2Plus SSL (64 bit) Service Customization'. The main area is titled 'Service Configuration' with the sub-instruction 'Specify optional settings to configure MongoDB as a service.' Below this, there are two radio button options: 'Install MongoDB as a Service' (selected) and 'Run service as Network Service user' (unchecked). Under the local or domain user option, fields for 'Account Domain' (empty), 'Account Name' ('MongoDB'), and 'Account Password' (empty) are present. The 'Service Name' field is set to 'MongoDB'. At the bottom, the 'Data Directory' is set to 'C:\Program Files\MongoDB\Server\4.0\data\' and the 'Log Directory' is set to 'C:\Program Files\MongoDB\Server\4.0\log\'. Navigation buttons at the bottom include '< Back', 'Next >', and 'Cancel'.

Instalação MongoDB



The following installs MongoDB only and does not configure MongoDB as a Windows service.

If you choose not to configure MongoDB as a Windows service, uncheck the **Install MongoDB as a Service**.

Executando MongoDB

Execute o MongoDB – pelo command (execute como administrador) vá no diretório bin e execute o arquivo “mongo.exe”.

"C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe"

```
D:\Program Files\MongoDB\Server\4.4\bin>mongo
MongoDB shell version v4.4.0
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("1843fa47-4730-4fae-9668-05bf5e03d650") }
MongoDB server version: 4.4.0
---
The server generated these startup warnings when booting:
    2020-11-22T11:03:56.207-03:00: ***** SERVER RESTARTED *****
    2020-11-22T11:03:59.973-03:00: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted
---
MongoDB Enterprise > ls collections
```

MongoDB - Definições

MongoDB é um banco de dados multi-plataforma orientado ao documento que fornece alta performance, alta disponibilidade e fácil escalabilidade.

MongoDB funciona no conceito de coleções de documentos.

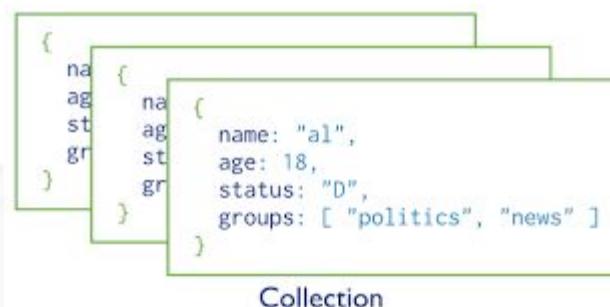
Banco de Dados

Banco de dados é um recipiente físico para coleções. Cada banco de dados tem o seu próprio conjunto de arquivos no sistema de arquivos. Um único servidor MongoDB normalmente tem vários bancos de dados.

MongoDB - Collection

Coleções (Collection)

- Collection é um grupo de documentos MongoDB.
- É o equivalente de uma tabela de RDBMS.
- Uma coleção existe dentro de um único banco de dados. Coleções não impõem um esquema.
- Documentos dentro de uma coleção pode ter diferentes campos. Normalmente, todos os documentos em uma coleção são propositalmente semelhantes ou afins.



MongoDB - Document

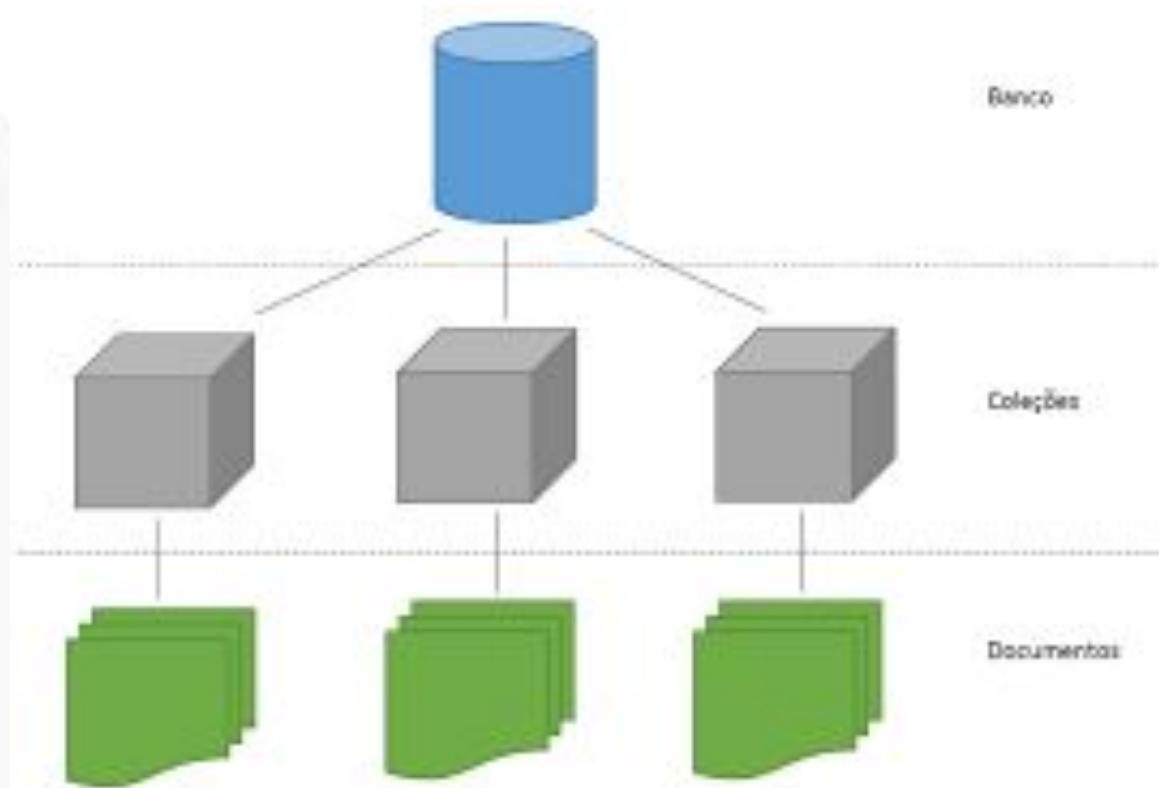
Document

- Um documento é um conjunto de pares de valores-chave.
- Documentos têm esquema dinâmico. Esquema dinâmico significa que os documentos na mesma coleção não precisam ter o mesmo conjunto de campos ou estrutura e campos comuns em documentos de uma coleção podem conter diferentes tipos de dados.

```
db.users.insertOne(  
  {  
    name: "sue",      ← field: value  
    age: 26,         ← field: value  
    status: "pending" ← field: value  
  })                ← collection
```

MongoDB - Definições

IGTi



MongoDB - Comandos

<https://docs.mongodb.com/manual/reference/command/>

```
db.posts.insert([
  {nome:"André",postagem:"Produtos caros", data:"12-01-2019",idade:25},
  {nome:"Ricardo",postagem:"Produtos caros", data:"14-07-2019", idade:12}])

#idade menor ou igual a 12
> db.posts.find({postagem:"Produtos caros",idade: {$lte: 12}})
{ "_id" : ObjectId("5d0911600ee1100c307004da"), "nome" : "Ricardo", "postagem"
: "Produtos caros", "data" : "14-07-2019", "idade" : 12 }
```

- **\$lt: menor que**
- **\$lte: menor ou igual que**
- **\$ne: diferente de**
- **\$in: contém**
- **\$nin: não contém**

MongoDB - Comandos



```
> db.posts.find()

{ "_id" : ObjectId("5d090bc10ee1100c307004d4"),
  "nome" : "José", "postagem" : "Bons Produtos!",
  "data" : "31-06-2019" }

{ "_id" : ObjectId("5d090cd10ee1100c307004d5"),
  "nome" : "Antonio", "postagem" : "Minha bike
quebrou", "data" : "26-05-2019" }

{ "_id" : ObjectId("5d090cd10ee1100c307004d6"),
  "nome" : "Maria Silva", "postagem" : "Encontrei
tudo que procurava", "data" : "14-06-2019" }

{ "_id" : ObjectId("5d090cd10ee1100c307004d7"),
  "nome" : "Lucas Andrade", "postagem" : "Ótimo
atendimento!", "data" : "12-04-2019" }
```

Conclusão



- ✓ MongoDB

Próxima Aula



01.

03.

Prática com MongoDB

02.

04.



Banco de Dados NoSQL

Capítulo 02 – Aula 02.05 – Prática com MongoDB

PROF.: RICARDO BRITO ALVES

Nesta Aula



- Prática com MongoDB

MongoDB – Create Database

MongoDB Create Database - criar um banco de dados MongoDB dinamicamente.

O comando MongoDB **USE DATABASE** é usado não apenas para **selecionar um banco de dados para executar consultas**, mas **também para criar um banco de dados**. Se o nome do banco de dados fornecido para o comando USE Database ainda não estiver presente no MongoDB, um novo banco de dados com o nome será criado quando você inserir um Documento em uma Coleção nesse banco de dados.

```
use <database_name>
```

MongoDB – Create Database



```
> show dbs;  
  
> use teste_aula  
  
> show dbs;  
  
> db.users.insertOne( { name: "Foo", age: 34, cars: [ "BMW  
320d", "Audi R8" ] } )  
  
> db.users.insertOne( { name: "TFC-test", age: 31, cars: [ "BMW  
320d", "Audi R8" ] } )  
  
> show dbs;
```

MongoDB – Delete Database

Para excluir ou descartar um banco de dados do MongoDB, siga as etapas abaixo:

Selecione o banco de dados que deseja excluir com a ajuda do comando USE <database>.

Elimine o banco de dados com a ajuda do comando db.dropDatabase () .

```
> show dbs  
  
> use teste_aula  
  
> db.dropDatabase()  
  
> show dbs
```

MongoDB – Collection

- A Collection é um armazenamento para documentos.
- É análogo a uma tabela no MySQL que armazena registros.
- Ao contrário dos bancos de dados relacionais, o MongoDB não tem um esquema. Um documento no MongoDB em uma collection pode ter quaisquer campos. Portanto, mudanças no documento não afetam documentos anteriores armazenados na mesma collection.

MongoDB – Collection

MongoDB – Create Collection

```
db.createCollection(name, options)
```

name	[mandatory] name of collection
options	[optional] mongodb document specifying information about collection

Options

Field	Type	Description
capped	Boolean	If true then collection is limited in size. i.e., Number of documents that could be stored in the collection is limited.
autoIndexId	Boolean	[deprecated]
size	Number	Size of Collection in Bytes
max	Number	Number of MongoDB Documents that could be stored in the collection

MongoDB – Collection

MongoDB – Create Collection

```
> use aula  
  
> show collections  
  
> db.customers.insert({ name: "Honey", age: 25, cars:  
    [ "Audi R8" ] })  
  
> show collections
```

MongoDB – Collection

MongoDB – Delete Collection

> use aula

> show collections

> db.customers.drop()

> show collections

Se a coleção não existe o MongoDB retorna false

> db.exemplo.drop()

MongoDB – Documents

- **Documents** em MongoDB é uma entidade na qual zero ou mais pares de valores de campo ordenados são armazenados.
- Em comparação com bancos de dados relacionais, é análogo a um registro ou linha na tabela.
- O documento no MongoDB segue as especificações BSON [<http://bsonspec.org/>]. BSON é a serialização com codificação binária de documentos semelhantes a JSON. Com o BSON, os documentos do MongoDB podem ser acessados facilmente. Como BSON usa tipos de dados C, codificar dados para BSON ou decodificar de BSON é mais fácil na maioria das linguagens de programação.

MongoDB – Documents

Um documento pode ter documentos aninhados nele. Os documentos do MongoDB são os blocos de construção de uma coleção do MongoDB.

```
{  
  name: "Midhuna",  
  age: 23,  
  place: "New York",  
  hobbies: ["Singing", "Reading Books"]  
}
```

```
{  
  name: "Midhuna",  
  age: 23,  
  place: "New York",  
  hobbies: ["Singing", "Reading Books"]  
  spouse: {  
    name: "Akash",  
    age: 25  
  }  
}
```

MongoDB – Insert Documents

```
> use aula  
  
> db.customers.insertOne( { name: "Abhi", age: 34, cars: [  
    "BMW 320d", "Audi R8" ] } )
```

MongoDB – Insert Documents



```
db.customers.insertMany(  
  [  
    { name: "Midhuna", age: 23, cars: [ "BMW 320d", "Audi R8" ],  
      place:"Amaravati" },  
    { name: "Akhil", age: 24, cars: [ "Audo A7", "Agera R" ],  
      place:"New York" },  
    { name: "Honey", age: 25, cars: [ "Audi R8" ] }  
    { name: "Paul", age: 25, cars: [ "Audi R8" ] }  
    { name: "Stuart", age: 25, cars: [ "Audi R8" ] }  
  ]  
)
```

MongoDB – Query Documents



`db.<collection>.find({})` – recupera todos documentos de
uma collection

> `use aula`

> `db.customers.find({})`

> `criteria={ age:23 }`

> `db.customers.find(criteria)`

MongoDB – Query Documents



```
db.collection.find(query_document, projection_document)
```

```
{field1:projection_value, field1:projection_value, ...}
```

Projection_value	Description
1	Include the field:value in Result
0	Do not include the field:value in Result

- use aria

```
> projection_doc={"name":1,"age":1,"cars":1,_id:0}
```

```
> db.customers.find({},projection_doc)
```

```
> db.customers.find({}, {"name":1, "age":1})
```

```
> db.customers.find({}, {"name":0, "age":0})
```

MongoDB – Update Documents



db.collection_name.update(criteria, update, options)

collection_name	Name of the collection in which you update document
criteria	[mandatory] Criteria to select documents for update
update	[mandatory] Updates to the fields for the selected documents

Option	[Type][DefaultValue] DefaultAction
upsert	[boolean][false] Does not insert a new document when no match found
multi	[boolean][false] Updates only one document.
collation	[document][default] Uses binary comparison for comparing strings.
WriteConcern	[document][default] Refer Write Concern [https://docs.mongodb.com/manual/reference/write-concern/].

MongoDB – Update Documents



```
> db.customers.find({})  
  
> criteria={name:"Akhil"}  
  
> db.customers.find(criteria)  
  
> update={name:"Akhil xxx",age:28}  
  
> db.customers.update(criteria,update)  
  
> db.customers.find(criteria)
```

MongoDB – Update Documents



```
> criteria={age:25}  
  
> db.customers.find(criteria).count()  
  
> update={$set:{cars: "Audi R8 xxx" }}  
  
> options={multi:true}  
  
> db.customers.update(criteria,update,options)  
  
> db.customers.find(criteria)
```

MongoDB – Delete Documents



`db.collection_name.remove(CRITERIA, JUST_ONE)`

collection_name	String	[mandatory] Name of the Collection from which you would like to remove Documents
CRITERIA	Document	[optional] The criteria that selects the required documents to delete
JUST_ONE	1 or true	[optional] If one or true, deletes only one document from the selection

MongoDB – Limitando Documents

- MongoDB Limit Documents - Para limitar o número de registros que uma consulta retorna no resultado, use o método cursor.limit () .
- O método Limit aceita um número como argumento que indica o limite do número de registros no resultado da consulta.

> db.customers.find().limit(2)

- Skipping é uma forma de ignorar registros e geralmente é útil quando você já mostrou os primeiros N documentos e está interessado em mostrar apenas os documentos restantes.

> db.customers.find().skip(2)

MongoDB – Sort Documents

- MongoDB Sort Documents - Para classificar documentos em uma coleção com base em um campo, use o método cursor.sort () .
- O método de classificação aceita pares de campo e pedido em um documento como argumento.

```
cursor.sort({field:order, field:order [, field:order]}
```

Order	Description
1	Ascending (Increasing) Order
-1	Descending (Decreasing) Order

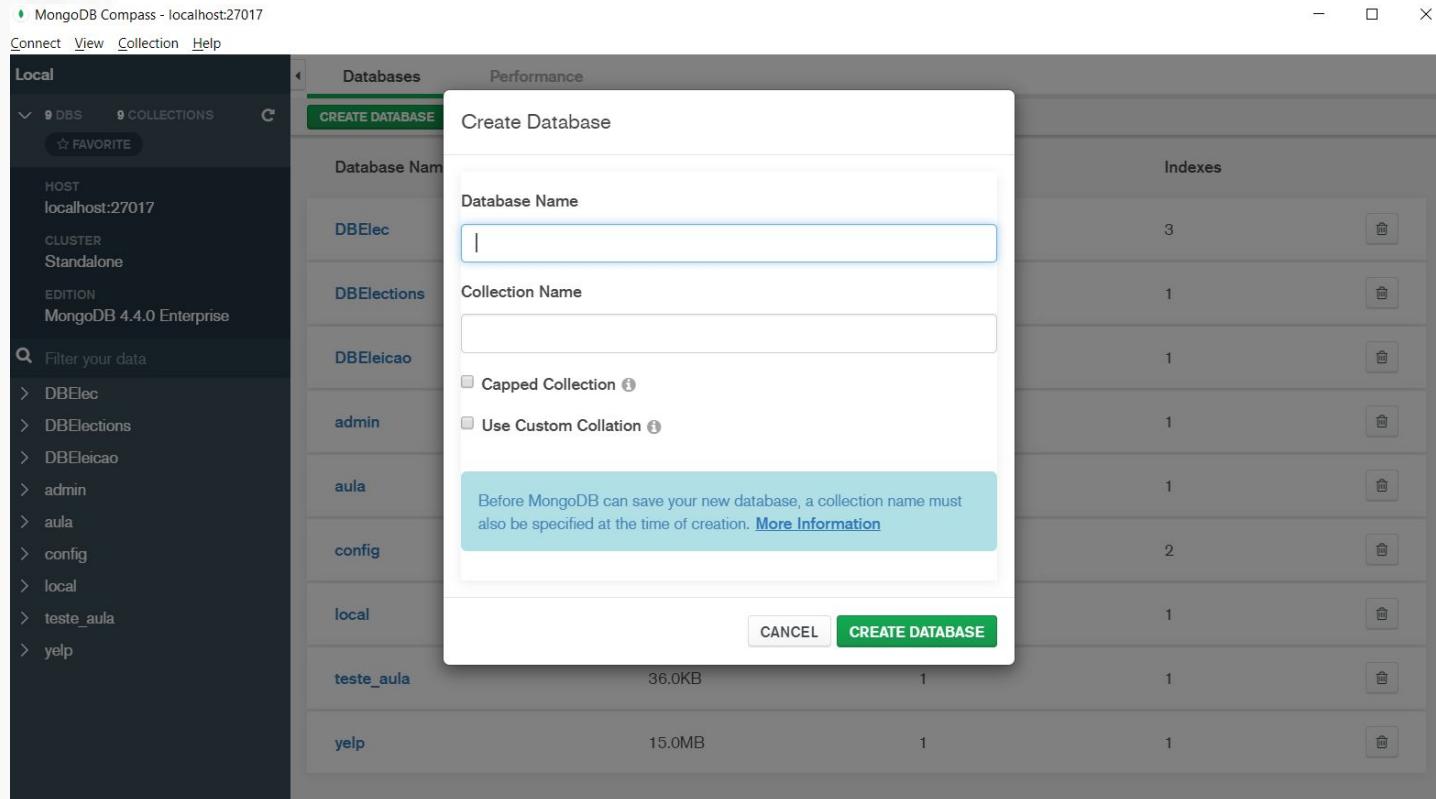
```
> db.customers.find().sort({"age":1});
```

```
> db.customers.find().sort({});
```

```
> db.customers.find().sort({"age": -1, "name": 1});
```

MongoDB – Import Compass

IGTI



Conclusão



- ✓ Prática com o banco de dados NoSQL MongoDB

Próxima Aula



01. ••

CRUD no MongoDB

02. ••

03. ••

04. ••

Bancos de Dados NoSQL

Capítulo 03 – CRUD no MongoDB

PROF.: RICARDO BRITO ALVES



Banco de Dados NoSQL

Capítulo 03 – Aula 03.01 – Logs

PROF.: RICARDO BRITO ALVES

Nesta aula



- Dicas no prompt de comandos
- Mongod.exe
- Logs

Tab e outros atalhos

- Para obter ajuda: db.g<Tab>

```
MongoDB Enterprise > db.  
db.adminCommand(          db.dropRole(          db.getQueryOptions(    db.killOp(          db.serverBits(  
db.aggregate(            db.dropUser(          db.getReplicationInfo( db.killOp(          db.serverBuildInfo(  
db.auth(                db.enableFreeMonitoring( db.getRole(          db.listCommands( db.serverCmdLineOpts(  
db.changeUserPassword( db.eval(          db.getRoles(        db.loadServerScripts( db.serverStatus(  
db.cloneDatabase(       db.forceError(        db.getSession(      db.logout(          db.setLevel(  
db.commandHelp(         db.fsyncLock(        db.getSiblingDB( db.printCollectionStats( db.setProfilingLevel(  
db.constructor(        db.fsyncUnlock(       db.getSiblingDB( db.printReplicationInfo( db.setSlaveOk(  
db.copyDatabase(       db.getCollection(     db.getSlaveOk(       db.printShardingStatus( db.setWriteConcern(  
db.createCollection(   db.getCollectionInfos( db.getUser(        db.printSlaveReplicationInfo( db.shutdownServer(  
db.createRole(          db.getCollectionNames( db.getUsers(        db.propertyIsEnumerable( db.stats(  
db.createUser(           db.getFreeMonitoringStatus( db.getWriteConcern( db.prototype(          db.toLocaleString(  
db.createView(          db.getLastError(       db.grantPrivilegesToRole( db.removeUser(        db.toString(  
db.currentOp(          db.getLastErrorCmd(     db.grantRolesToRole( db.resetError(        db.toJson(  
db.currentOp(          db.getLastErrorObj(    db.grantRolesToUser( db.revokePrivilegesFromRole( db.unsetWriteConcern(  
db.dbEval(              db.getLogComponents( db.groupeval(        db.revokeRolesFromRole( db.updateRole(  
db.disableFreeMonitoring( db.getMongo(          db.hasOwnProperty( db.revokeRolesFromUser( db.updateUser(  
db.dropAllRoles(        db.getName(          db.help(          db.runCommand(        db.valueOf(  
db.dropAllUsers(        db.getProfilingLevel( db.hostInfo(        db.runCommandWithMetadata( db.version(  
db.dropDatabase(        db.getProfilingStatus( db.isMaster(        db.runReadCommand( db.watch(  
                                         db.
```

- Use setas para cima ou para baixo para acessar o histórico de comandos.
- CTRL C – copia ou encerra
- CTRL V – botão direito do mouse
- Aspas simples, duplas ou ausente:

```
MongoDB Enterprise > db.aula.insert({"nome": "Jose1"})  
WriteResult({ "nInserted" : 1 })  
MongoDB Enterprise > db.aula.insert({'nome': 'Jose1'})  
WriteResult({ "nInserted" : 1 })  
MongoDB Enterprise > db.aula.insert({nome: 'Jose1'})  
WriteResult({ "nInserted" : 1 })
```

Mongod.exe

Mongod é o processo principal para o sistema MongoDB. Ele lida com solicitações de dados, gerencia o acesso aos dados e executa operações de gerenciamento em segundo plano.

Fica na pasta bin.

--help, -h: retorna as informações e opções de uso do mongod

--version: retorna a versão de uso do mongod

--config <filename>, -f <filename>

...

```
C:\ Administrador: Prompt de Comando
C:\Program Files\MongoDB\Server\4.4\bin>mongod --version
db version v4.4.0
Build Info: {
    "version": "4.4.0",
    "gitVersion": "563487e100c4215e2dce98d0af2a6a5a2d67c5cf",
    "modules": [
        "enterprise"
    ],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}

C:\Program Files\MongoDB\Server\4.4\bin>
```

Logs - MongoDB



e Computador > DATA (D:) > Program Files > MongoDB > Server > 4.4 > log

Nome	Data de modificação	Tipo	Tamanho
mongod	24/12/2020 11:46	Documento de Te...	1.323 KB

Pesquisar log

mongod - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

```
{"t":{"$date":"2020-12-24T11:09:57.474-03:00"}, "s":"I", "c":"NETWORK", "id":22944, "ctx":"conn4","msg":"connection ended","attr":{"remoteAddress":"127.0.0.1:51803"}}, {"t":{"$date":"2020-12-24T11:09:57.474-03:00"}, "s":"I", "c":"NETWORK", "id":22944, "ctx":"conn11","msg":"connection ended","attr":{"remoteAddress":"127.0.0.1:51800"}}, {"t":{"$date":"2020-12-24T11:09:57.474-03:00"}, "s":"I", "c":"NETWORK", "id":22944, "ctx":"conn5","msg":"connection ended","attr":{"remoteAddress":"127.0.0.1:51800"}}, {"t":{"$date":"2020-12-24T11:09:57.474-03:00"}, "s":"I", "c":"NETWORK", "id":22944, "ctx":"conn7","msg":"connection ended","attr":{"remoteAddress":"127.0.0.1:51800"}}, {"t":{"$date":"2020-12-24T11:09:57.475-03:00"}, "s":"I", "c":"NETWORK", "id":22944, "ctx":"conn2","msg":"connection ended","attr":{"remoteAddress":"127.0.0.1:51800"}}, {"t":{"$date":"2020-12-24T11:09:57.479-03:00"}, "s":"I", "c":"NETWORK", "id":22944, "ctx":"conn1","msg":"connection ended","attr":{"remoteAddress":"127.0.0.1:51800"}}, {"t":{"$date":"2020-12-24T11:11:36.364-03:00"}, "s":"I", "c":"COMMAND", "id":51803, "ctx":"LogicalSessionCacheRefresh","msg":"Slow query"}, {"t":{"$date":"2020-12-24T11:40:32.730-03:00"}, "s":"I", "c":"NETWORK", "id":22943, "ctx":"listener","msg":"connection accepted","attr":{}}, {"t":{"$date":"2020-12-24T11:40:32.897-03:00"}, "s":"I", "c":"NETWORK", "id":51800, "ctx":"conn14","msg":"client metadata","attr":{}}, {"t":{"$date":"2020-12-24T11:40:32.988-03:00"}, "s":"I", "c":"NETWORK", "id":22943, "ctx":"listener","msg":"connection accepted","attr":{}}, {"t":{"$date":"2020-12-24T11:40:32.990-03:00"}, "s":"I", "c":"NETWORK", "id":51800, "ctx":"conn15","msg":"client metadata","attr":{}}, {"t":{"$date":"2020-12-24T11:40:33.132-03:00"}, "s":"I", "c":"COMMAND", "id":51803, "ctx":"conn15","msg":"Slow query","attr":{"type":"command"}}, {"t":{"$date":"2020-12-24T11:41:36.884-03:00"}, "s":"I", "c":"WRITE", "id":51803, "ctx":"LogicalSessionCacheRefresh","msg":"Slow query"}, {"t":{"$date":"2020-12-24T11:41:36.885-03:00"}, "s":"I", "c":"COMMAND", "id":51803, "ctx":"LogicalSessionCacheRefresh","msg":"Slow query"}, {"t":{"$date":"2020-12-24T11:42:53.085-03:00"}, "s":"I", "c":"COMMAND", "id":51803, "ctx":"conn15","msg":"Slow query","attr":{"type":"command"}}, {"t":{"$date":"2020-12-24T11:43:17.859-03:00"}, "s":"I", "c":"NETWORK", "id":22943, "ctx":"listener","msg":"connection accepted","attr":{}}, {"t":{"$date":"2020-12-24T11:43:17.864-03:00"}, "s":"I", "c":"NETWORK", "id":51800, "ctx":"conn16","msg":"client metadata","attr":{}}, {"t":{"$date":"2020-12-24T11:43:17.872-03:00"}, "s":"I", "c":"NETWORK", "id":22943, "ctx":"listener","msg":"connection accepted","attr":{}}, {"t":{"$date":"2020-12-24T11:43:17.874-03:00"}, "s":"I", "c":"NETWORK", "id":51800, "ctx":"conn17","msg":"client metadata","attr":{}}, {"t":{"$date":"2020-12-24T11:43:18.081-03:00"}, "s":"I", "c":"STORAGE", "id":20320, "ctx":"conn17","msg":"createCollection","attr":{"name":"test"}}, {"t":{"$date":"2020-12-24T11:43:18.834-03:00"}, "s":"I", "c":INDEX", "id":20345, "ctx":"conn17","msg":"Index build: done building", "attr":{}}, {"t":{"$date":"2020-12-24T11:43:18.834-03:00"}, "s":"I", "c":COMMAND, "id":51803, "ctx":"conn17","msg":"Slow query","attr":{"type":"command"}}, {"t":{"$date":"2020-12-24T11:43:20.451-03:00"}, "s":"I", "c":COMMAND, "id":51803, "ctx":"conn17","msg":"Slow query","attr":{"type":"command"}}, {"t":{"$date":"2020-12-24T11:43:25.488-03:00"}, "s":"I", "c":NETWORK, "id":22944, "ctx":"conn17","msg":"connection ended","attr":{}}, {"t":{"$date":"2020-12-24T11:43:25.489-03:00"}, "s":"I", "c":NETWORK, "id":22944, "ctx":"conn16","msg":"connection ended","attr":{}}, {"t":{"$date":"2020-12-24T11:46:48.901-03:00"}, "s":"I", "c":NETWORK, "id":22943, "ctx":"listener","msg":"connection accepted","attr":{}}, {"t":{"$date":"2020-12-24T11:46:48.903-03:00"}, "s":"I", "c":NETWORK, "id":51800, "ctx":"conn18","msg":"client metadata","attr":{}}
```

Ln 1, Col 1 100% Unix (LF) UTF-8

Logs

MongoDB possui um recurso sofisticado de criação de perfil. O log acontece na system.profile. Os logs podem ser vistos em:

`db.system.profile.find()`

Existem 3 níveis de log (origem):

- **Nível 0** - o *criador de perfil está desativado*, não coleta nenhum dado. O mongod sempre grava operações mais longas do que o limite slowOpThresholdMs em seu log. Este é o nível padrão do criador de perfil.
- **Nível 1** - coleta dados de criação de perfil *apenas para operações lentas*. *Por padrão, operações lentas são aquelas mais lentas que 100 milissegundos*. Você pode modificar o limite para operações "lentas" com a opção de tempo de execução slowOpThresholdMs ou o comando setParameter. Consulte a seção Especificar o limite para operações lentas para obter mais informações.
- **Nível 2** - coleta dados de criação de perfil *para todas as operações do banco de dados*.

Logs

Para ver em que nível de perfil o banco de dados está sendo executado, use: ***db.getProfilingLevel()***

Para ver o status: ***db.getProfilingStatus()***

Para alterar o status da criação de perfil, use o comando

db.setProfilingLevel(level, milliseconds)

onde level refere-se ao nível de criação de perfil e milliseconds é o ms de qual duração as consultas precisam ser registradas.

Para desativar o log, use: ***db.setProfilingLevel(0)***

A consulta para procurar na coleção de perfis do sistema todas as consultas que levaram mais de um segundo, ordenadas por carimbo de data e hora decrescente, será:

db.system.profile.find({ millis : { \$gt:1000 } }).sort({ ts : -1 })

Logs



```
MongoDB Enterprise > db.setProfilingLevel(2)
{ "was" : 2, "slowms" : 100, "sampleRate" : 1, "ok" : 1 }
MongoDB Enterprise > db.ts.insert({'nome':'Jose'})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.ts.insert({'nome':'Pedro'})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.system.profile.find()
{
  "op" : "command", "ns" : "local.ts", "command" : { "drop" : "ts", "lsid" : { "id" : UUID("09ac33cb-25ee-43d9-8e9b-0f94cbceb677") }, "$db" : "local",
  "locks" : { "ParallelBatchWriterMode" : { "acquireCount" : { "r" : NumberLong(3) } }, "ReplicationStateTransition" : { "acquireCount" : { "w" : Num-
  berLong(1) }, "Global" : { "acquireCount" : { "w" : NumberLong(3) } }, "Database" : { "acquireCount" : { "w" : NumberLong(3) } }, "Collection" : { "acquireCount" : { "w" :
  NumberLong(1) } }, "Mutex" : { "acquireCount" : { "r" : NumberLong(3) } } }, "flowControl" : { "acquireCount" : NumberLong(3), "timeAcquiringMicros" : Number-
  Long(6) }, "responseLength" : 72, "protocol" : "op_msg", "millis" : 13, "ts" : ISODate("2020-12-31T14:53:58.935Z"), "client" : "127.0.0.1", "appName" :
  "MongoDB Shell", "allUsers" : [ ], "user" : "" },
  "op" : "insert", "ns" : "local.ts", "command" : { "insert" : "ts", "ordered" : true, "lsid" : { "id" : UUID("09ac33cb-25ee-43d9-8e9b-0f94cbceb677") },
  "ninserted" : 1, "keysInserted" : 1, "numYield" : 0, "locks" : { "ParallelBatchWriterMode" : { "acquireCount" : { "r" : NumberLong(3) } }, "Repli-
  tion" : { "acquireCount" : { "w" : NumberLong(3) } }, "Global" : { "acquireCount" : { "w" : NumberLong(3) } }, "Database" : { "acquireCount" : { "w" :
  NumberLong(1) }, "Global" : { "acquireCount" : { "w" : NumberLong(3) } }, "Database" : { "acquireCount" : { "w" : NumberLong(3) } }, "Collection" : { "acquireCount" : { "r" :
  NumberLong(1), "w" : NumberLong(3) } }, "Mutex" : { "acquireCount" : { "r" : NumberLong(3) } } }, "flowControl" : { "acquireCount" : NumberLong(3),
  "timeAcquiringMicros" : NumberLong(4) }, "responseLength" : 45, "protocol" : "op_msg", "millis" : 11, "ts" : ISODate("2020-12-31T14:53:58.935Z"),
  "client" : "127.0.0.1", "appName" : "MongoDB Shell", "allUsers" : [ ], "user" : "" },
  "op" : "insert", "ns" : "local.ts", "command" : { "insert" : "ts", "ordered" : true, "lsid" : { "id" : UUID("09ac33cb-25ee-43d9-8e9b-0f94cbceb677") },
  "ninserted" : 1, "keysInserted" : 1, "numYield" : 0, "locks" : { "ParallelBatchWriterMode" : { "acquireCount" : { "r" : NumberLong(1) } }, "Repli-
  tion" : { "acquireCount" : { "w" : NumberLong(1) } }, "Global" : { "acquireCount" : { "w" : NumberLong(1) } }, "Database" : { "acquireCount" : { "w" :
  NumberLong(1) }, "Global" : { "acquireCount" : { "w" : NumberLong(1) } }, "Database" : { "acquireCount" : { "w" : NumberLong(1) } }, "Collection" : { "acquireCount" : { "w" :
  NumberLong(1) } }, "Mutex" : { "acquireCount" : { "r" : NumberLong(1) } } }, "flowControl" : { "acquireCount" : NumberLong(1),
  "timeAcquiringMicros" : NumberLong(1) }, "responseLength" : 45, "protocol" : "op_msg", "millis" : 0, "ts" : ISODate("2020-12-31T14:54:24.368Z"),
  "client" : "127.0.0.1", "appName" : "MongoDB Shell", "allUsers" : [ ], "user" : "" },
MongoDB Enterprise > db.setProfilingLevel(0)
{ "was" : 2, "slowms" : 100, "sampleRate" : 1, "ok" : 1 }
```

Conclusão



- ✓ Dicas de comandos
- ✓ Mongod.exe
- ✓ Logs

Próxima aula



01. ••

Tipos para os Fields

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 03 – Aula 03.02 – Tipos - Fields

PROF.: RICARDO BRITO ALVES

Nesta aula



- Tipos Fields nos documentos
- Object Id

Tipos nos Documentos

MongoDB suporta vários tipos de dados.

- String
- Integer
- Boolean
- Double
- Min/ Max keys
- Arrays
- Timestamp
- Null
- Symbol
- Date
- Object ID
- Binary data
- Code
- Regular expression

Tipos nos Documentos

String: Este é o tipo de dados mais comumente usado para armazenar dados. String no mongodb deve ser um UTF-8 válido.

Integer: Este tipo é usado para armazenar um valor numérico. Integer pode ser 32 bits ou 64 bits dependendo do seu servidor.

Boolean: Este tipo é usado para armazenar um valor booleano (verdadeiro/falso).

Double: Este tipo é usado para armazenar valores de ponto flutuante.

Min/ Max keys: Este tipo é usado para comparar um valor contra os elementos mais baixos e mais altos de um BSON.

MinKey e MaxKey são usados em operações de comparação e existem principalmente para uso interno. Para todos os valores possíveis do elemento BSON, MinKey sempre será o menor valor, enquanto MaxKey sempre será o maior valor.

Tipos nos Documentos



Arrays: Este tipo é usado para armazenar arrays, listas ou múltiplos valores dentro de uma chave(key).

Timestamp: ctimestamp. Isto pode ser útil para a gravação de quando um documento foi modificado ou acrescentado.

Object: Este tipo de dado é usado para incorporar documentos.

Null: Este tipo é usado para armazenar um valor nulo(null).

Symbol: Este tipo de dados é usado de forma idêntica ao String, porém ele é geralmente reservado para linguagens que usam um tipo de símbolo específico.

Tipos nos Documentos

Date: Este tipo de dados é utilizado para armazenar a data ou a hora atual no formato de UNIX. Você pode especificar o seu próprio date_time através da criação do objeto Date e passando o dia, mês e ano para ele.

Object ID: Este tipo de dados é usado para armazenar os identificadores(ID) dos documentos.

Binary data: Este tipo de dados é usado para armazenar um dado binário.

Code: Este tipo de dados é usado para armazenar código javascript dentro do documento.

Regular expression: Este tipo de dados é usado para armazenar expressões regulares.

Tipos nos Documentos



Chave - valor

```
{  
  "nome": "Pedro Pereira",  
  "idade": 5,  
  "esportes": ["tênis", "futebol", "videogame"],  
  "estuda": true,  
  "foto": ddkfso242dsdsIsslSdd  
  "matérias_notas":  
  {  
    "matemática": 9,1,  
    "português": 7,8,  
    "geografia": 8,2  
  }  
}
```

The code illustrates various data types in a JSON-like object:

- `"nome": "Pedro Pereira"` // string
- `"idade": 5` // number
- `"esportes": ["tênis", "futebol", "videogame"]` // array
- `"estuda": true` // boolean
- `"foto": ddkfso242dsdsIsslSdd` // binary
- `"matérias_notas":` // object
 - `"matemática": 9,1` // number
 - `"português": 7,8` // number
 - `"geografia": 8,2` // number

Tipos nos Documentos



Chave - valor

```
{  
    "nome": "Pedro Pereira",  
    "datanascimento": new ISODate('2012-05-01 12:30:00'),  
    "esportes": ["tênis", "futebol", "videogame"],  
    "estuda": true,  
    "matérias_notas":  
    {  
        "matemática": 9.1,  
        "portugues": 7.8,  
        "geografia": 8.2  
    }  
}
```

Tipos nos Documentos - Datas



Para trabalhar com a função ***ISODate()*** devemos especificar a data no formato “***YYYY-MM-DD hh:mm:ss***”, dentre outros possíveis. Por exemplo: a inserção de um documento com a data de 1º de maio de 2012 às 12h30 ficaria:

```
db.ColDate.insert({ aDate: new ISODate('2012-05-01 12:30:00') })
```

Para obter partes da data, podemos utilizar os seguintes métodos do tipo de dados Date:

- ***getDate()*** obtém o dia do mês: > doc.aDate.getDate()
- ***getMonth()*** obtém o mês - Janeiro=0 e Dezembro11:
> doc.aDate.getMonth()
- ***getFullYear()*** obtém a ano: > doc.aDate.getFullYear()

ObjectId

- Em bancos de dados relacionais temos as primary Keys, que são chaves primárias. É um identificador único da linha da tabela.
- No MongoDB temos o ObjectId.

```
MongoDB Enterprise > db.teste.find()
{ "_id" : ObjectId("5fed3897b1446e3c5d4502ae"), "nome" : "Pedro Pereira", "datanascimento" : ISODate("2012-05-01T12:30:00Z"), "esportes" : [ "tênis", "game" ], "estuda" : true, "matérias_notas" : { "matemática" : 9.1, "português" : 7.8, "geografia" : 8.2 } }
{ "_id" : ObjectId("5fed38e6b1446e3c5d4502af"), "nome" : "Pedro Pereira", "datanascimento" : ISODate("2012-05-01T12:30:00Z"), "esportes" : [ "tênis", "game" ], "estuda" : true, "matérias_notas" : { "matemática" : 9.1, "português" : 7.8, "geografia" : 8.2 } }
MongoDB Enterprise > db.teste.find().pretty()
{
  "_id" : ObjectId("5fed3897b1446e3c5d4502ae"),
  "nome" : "Pedro Pereira",
  "datanascimento" : ISODate("2012-05-01T12:30:00Z"),
  "esportes" : [
    "tênis",
    "futebol",
    "videogame"
  ],
  "estuda" : true,
  "matérias_notas" : {
    "matemática" : 9.1,
    "português" : 7.8,
    "geografia" : 8.2
  }
}
{
  "_id" : ObjectId("5fed38e6b1446e3c5d4502af"),
  "nome" : "Pedro Pereira",
  "datanascimento" : ISODate("2012-05-01T12:30:00Z"),
  "esportes" : [
    "tênis",
    "futebol",
    "videogame"
  ],
  "estuda" : true,
  "matérias_notas" : {
    "matemática" : 9.1,
    "português" : 7.8,
    "geografia" : 8.2
  }
}
```

ObjectId

ObjectId (<hexadecimal>)

O valor ObjectId de 12 bytes consiste em:

- Um valor de ***timestamp*** (carimbo pela data/hora) de 4 bytes, representando a criação do ObjectId, medido em segundos.
- Um ***valor aleatório*** de 5 bytes (elimina conflitos em casos de inserção de registro no mesmo instante).
- Um ***contador*** de incremento de 3 bytes, inicializado com um valor aleatório.

_id



\$ mongo

```
MongoDB Enterprise > db.teste.insert(  
... {  
...   '_id': 1,  
...   'nome': 'José Pereira',  
...   'datanascimento': new ISODate('2013-05-01 12:30:00'),  
...   'esportes': ['tênis', 'futebol', 'videogame'],  
...   'estuda': true,  
...   'matérias_notas':  
...     {  
...       'matemática': 5.1,  
...       'portugues': 4.8,  
...       'geografia': 8.3  
...     }  
... })  
WriteResult({  
    "nInserted" : 0,  
    "writeError" : {  
        "code" : 11000,  
        "errmsg" : "E11000 duplicate key error collection: aula.teste index: _id_ dup key: { _id: 1.0 }"  
    }  
})
```

ObjectId

- **ObjectId.getTimestamp ()**: retorna a parte do carimbo de data / hora do objeto como uma Data.
- **ObjectId.toString ()**: retorna a representação JavaScript na forma de um literal de string “ObjectId (...)”.
- **ObjectId.valueOf ()**: retorna a representação do objeto como uma string hexadecimal. A string retornada é o atributo str.

Observação: o ObjectId **usa da data do cliente e não do server**. Se o cliente estiver com datas erradas, você refletirá isso no ObjectId.

ObjectId



```
MongoDB Enterprise > db.teste.find({})
{ "_id" : ObjectId("5fed3897b1446e3c5d4502ae"), "nome" : "Pedro Pereira", "datanascimento" : ISODate("2012-05-01T12:30:00Z"), "esportes" : [ "tênis", "game" ], "estuda" : true, "matérias_notas" : { "matemática" : 9.1, "portugues" : 7.8, "geografia" : 8.2 } }
{ "_id" : ObjectId("5fed38e6b1446e3c5d4502af"), "nome" : "Pedro Pereira", "datanascimento" : ISODate("2012-05-01T12:30:00Z"), "esportes" : [ "tênis", "game" ], "estuda" : true, "matérias_notas" : { "matemática" : 9.1, "portugues" : 7.8, "geografia" : 8.2 } }
{ "_id" : 1, "nome" : "Pedro Pereira", "datanascimento" : ISODate("2012-05-01T12:30:00Z"), "esportes" : [ "tênis", "futebol", "videogame" ], "estuda" _notas" : { "matemática" : 9.1, "portugues" : 7.8, "geografia" : 8.2 } }
{ "_id" : 2, "nome" : "José Pereira", "datanascimento" : ISODate("2013-05-01T12:30:00Z"), "esportes" : [ "tênis", "futebol", "videogame" ], "estuda" notas" : { "matemática" : 5.1, "portugues" : 4.8, "geografia" : 8.3 } }
MongoDB Enterprise > ObjectId("5fed38e6b1446e3c5d4502af").getTimestamp()
ISODate("2020-12-31T02:35:18Z")
MongoDB Enterprise > ObjectId("5fed38e6b1446e3c5d4502af").toString()
ObjectId("5fed38e6b1446e3c5d4502af")
MongoDB Enterprise > ObjectId("5fed38e6b1446e3c5d4502af").valueOf()
5fed38e6b1446e3c5d4502af
```

```
MongoDB Enterprise > db.teste.findOne()
{
    "_id" : ObjectId("5fed3897b1446e3c5d4502ae"),
    "nome" : "Pedro Pereira",
    "datanascimento" : ISODate("2012-05-01T12:30:00Z"),
    "esportes" : [
        "tênis",
        "futebol",
        "videogame"
    ],
    "estuda" : true,
    "matérias_notas" : {
        "matemática" : 9.1,
        "portugues" : 7.8,
        "geografia" : 8.2
    }
}
MongoDB Enterprise > db.teste.findOne()._id
ObjectId("5fed3897b1446e3c5d4502ae")
MongoDB Enterprise > db.teste.findOne()._id.getTimestamp()
ISODate("2020-12-31T02:33:59Z")
```

Conclusão



- ✓ Tipos Fields nos documentos
- ✓ Object Id

Próxima aula



01. ••

Inserts

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 03 – Aula 03.03 – Inserts

PROF.: RICARDO BRITO ALVES

Nesta aula



- CRUD
- Inserts

CRUD



✓ Create

```
db.collection.insert( <document> )
db.collection.save( <document> )
db.collection.update( <query>, <update>, { upsert: true } )
```

✓ Read

```
db.collection.find( <query>, <projection> )
db.collection.findOne( <query>, <projection> )
```

✓ Update

```
db.collection.update( <query>, <update>, <options> )
```

✓ Delete

```
db.collection.remove( <query>, <justOne> )
```

CRUD



```
> db.user.insert({  
  first: "John",  
  last : "Doe",  
  age: 39  
})
```

```
> db.user.find ()  
{  
  "_id" : ObjectId("51..."),  
  "first" : "John",  
  "last" : "Doe",  
  "age" : 39  
}
```

```
> db.user.update(  
  {"_id" : ObjectId("51...")},  
  {  
    $set: {  
      age: 40,  
      salary: 7000)  
  }  
)
```

```
> db.user.remove({  
  "first": /^J/  
})
```

Inserts

- ✓ `Insert({})`
- ✓ `InsertOne({})`
- ✓ `insertMany([{}, {}, {}])`

Insert ({})

- ✓ `db.crud.insert({field_A: 123})`
- ✓ `db.crud.insert([{field_A: 123}, {field_B: 223}])` //array com mais de um insert

```
MongoDB Enterprise > db.crud.insert({field_A: 123})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.crud.insert([{"field_A": 123}, {"field_B": 223}])
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 2,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
```

InsertOne ({ })



```
db.collection.insertOne(  
  <document>,  
  {  
    writeConcern: <document>  
  }  
)
```

writeConcern - Opcional. Um documento expressando a preocupação de gravação. Omita o uso da preocupação de gravação padrão, *não defina explicitamente a preocupação de gravação para a operação se executado em uma transação.*

Retorna um booleano como verdadeiro se a operação foi executada com preocupação de gravação ou falso se a preocupação com gravação foi desabilitada.

Write Concern

O objeto passado como argumento possui duas chaves: **w** e **j**.

A opção **w** define a necessidade de um cliente em garantir a escrita de dados **em um, nenhum, ou mais de um servidor mongodb**.

Para entender de fato o funcionamento dessa opção, é **necessário entender um pouco sobre Replica Sets**.

De forma resumida:

- A opção **{w: 1}** (padrão no mongodb) exige que a informação seja escrita em uma instância do mongodb para retornar um código de sucesso ao cliente inserindo o dado.
- A opção **{w: 0}** exibe o código de sucesso sem a garantia da persistência de dados.
- Já a opção **{w: "Majority"}** faz com que os dados sejam inseridos na maioria dos membros de um Replica Set (conjunto de servidores mongodb com intuito de replicar dados e fornecer alta-disponibilidade).

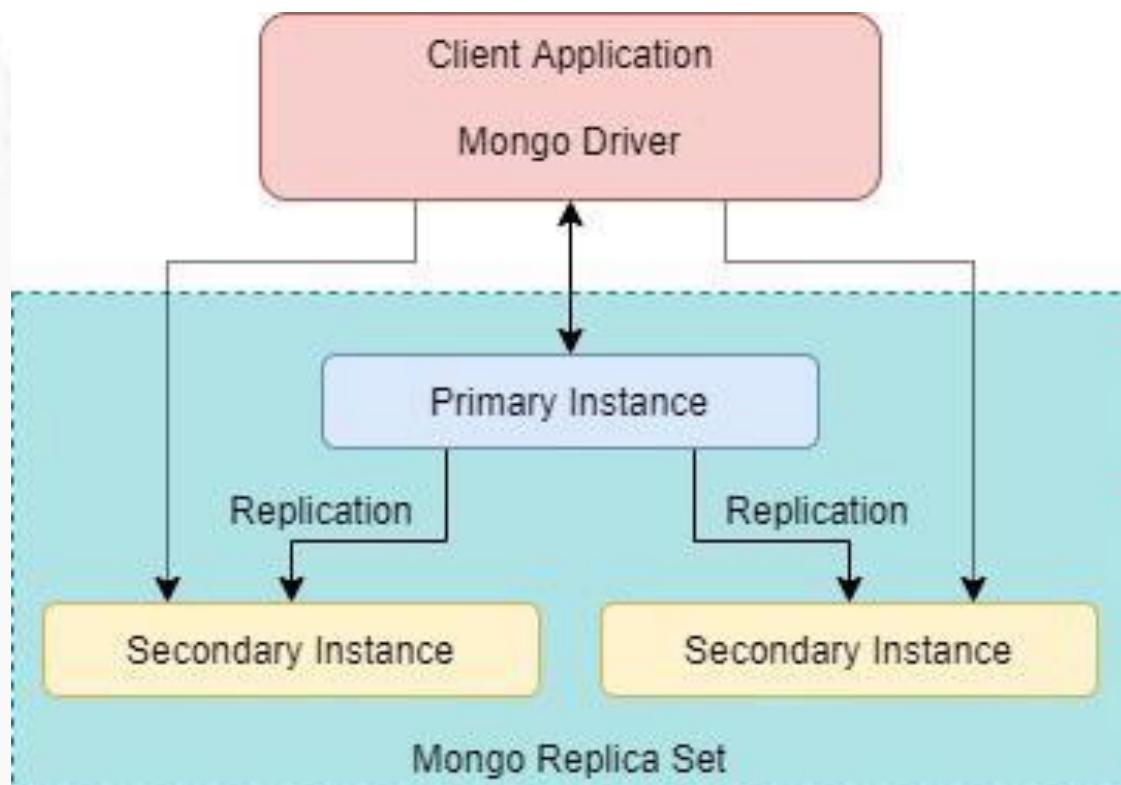
Write Concern

A opção *j* define a necessidade de um cliente ***em garantir a escrita de dados em disco.***

Caso você não especifique opção *j*, os dados escritos permanecem *em memória RAM, até que eventualmente o mongoDB grave os dados em seu log de transação, chamado oplog.*

Para garantir a persistência dos dados de forma persistente, passe como argumento o opção {*j*: True} a cada operação de escrita.

Replica Set



InsertOne ({ })

A partir da versão 3.2 do MongoDB

- ✓ db.crud.insertOne({field_A: 2123})

```
MongoDB Enterprise > db.crud.insertOne([{field_A: 2123}, {field_B: 2223}])
uncaught exception: Error: operation passed in cannot be an Array :
addOperationsList@src/mongo/shell/bulk_api.js:604:23
Bulk/this.insert@src/mongo/shell/bulk_api.js:650:20
DBCollection.prototype.insertOne@src/mongo/shell/crud_api.js:260:5
@(shell):1:1
MongoDB Enterprise > db.crud.insertOne({field_A: 2123})
{
    "acknowledged" : true,
    "insertedId" : ObjectId("5fedf44469910c9912e213e4")
}
```

InsertMany ([{ }, { }, { }])



A partir da versão 3.2 do MongoDB

- ✓ `insertMany ([{ }, { }, { }])`
- ✓ `db.crud.insertOne([{field_A: 2123}, {field_B: 2223}])`

InsertMany ([{ }, { }, { }])



```
db.collection.insertMany(  
  [ <document 1> , <document 2>, ... ],  
  {  
    writeConcern: <document>,  
    ordered: <boolean>  
  }  
)
```

Ordered

- Ordered é um booleano opcional.
- Especifica se a instância do mongod **deve executar uma inserção ordenadamente ou não ordenadamente**.
- O padrão é true.

Ordered



```
MongoDB Enterprise > db.crudMany.insertMany([
... {_id:1, a: 123},
... {_id:2, b: 456},
... {_id:3, c: 789}])
{ "acknowledged" : true, "insertedIds" : [ 1, 2, 3 ] }
MongoDB Enterprise > db.crudMany.find()
{ "_id" : 1, "a" : 123 }
{ "_id" : 2, "b" : 456 }
{ "_id" : 3, "c" : 789 }
MongoDB Enterprise > db.crudMany.insertMany([
... {_id:2, a: 4123},
... {_id:7, b: 4456}])
uncaught exception: BulkWriteError{
  "writeErrors" : [
    {
      "index" : 0,
      "code" : 11000,
      "errmsg" : "E11000 duplicate key error collection: te
      "op" : {
        "_id" : 2,
        "a" : 4123
      }
    }
  ]
}
```

Ordered



```
MongoDB Enterprise > db.crudMany.insertMany([ {_id:2, a: 4123}, {_id:7, b: 4456}], {ordered: false})
uncaught exception: BulkWriteError({
  "writeErrors" : [
    {
      "index" : 0,
      "code" : 11000,
      "errmsg" : "E11000 duplicate key error collection: test.crudMany index: _id_ dup key: { _id: 2.0 }",
      "op" : {
        "_id" : 2,
        "a" : 4123
      }
    }
  ],
  "writeConcernErrors" : [ ],
  "nInserted" : 1, ←
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
}) :
```

```
MongoDB Enterprise > db.crudMany.find()
{ "_id" : 1, "a" : 123 }
{ "_id" : 2, "b" : 456 }
{ "_id" : 3, "c" : 789 }
{ "_id" : 7, "b" : 4456 }
```

Conclusão



- ✓ Insert
- ✓ InsertOne
- ✓ InsertMany

Próxima aula



01. ••

03. ••

Queries – consultas no MongoDB

02. ••

04. ••



Banco de Dados NoSQL

Capítulo 03 – Aula 03.04 – Read - Query

PROF.: RICARDO BRITO ALVES

Nesta aula



- ❑ Consultas no MongoDB

Read - Queries



- ✓ `find()`
- ✓ `findOne()`

Find()

`db.collection.find(query, projection)`

- ✓ **Query** – opcional: especifica o filtro de seleção usando operadores de consulta. Para retornar todos os documentos em uma coleção, omita este parâmetro ou passe um documento vazio ({}).
- ✓ **Projeção** – opcional: especifica os campos a serem retornados nos documentos que correspondem ao filtro da consulta. Para retornar todos os campos nos documentos correspondentes, omita este parâmetro.

Find()

```
MongoDB Enterprise > db.crud2.insertMany([
... {a: 123},
... {b: 456},
... {c: 789},
... {a: 123},
... {b: 456},
... {c: 789},
... {a: 123},
... {b: 456},
... {c: 789},
... {e: 123},
... {f: 456},
... {g: 789},
... ])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5fee018169910c9912e21400"),
        ObjectId("5fee018169910c9912e21401"),
        ObjectId("5fee018169910c9912e21402"),
        ObjectId("5fee018169910c9912e21403"),
        ObjectId("5fee018169910c9912e21404"),
        ObjectId("5fee018169910c9912e21405"),
        ObjectId("5fee018169910c9912e21406"),
        ObjectId("5fee018169910c9912e21407"),
        ObjectId("5fee018169910c9912e21408"),
        ObjectId("5fee018169910c9912e21409"),
        ObjectId("5fee018169910c9912e2140a"),
        ObjectId("5fee018169910c9912e2140b")
    ]
}
```

Find()

```
db.crud2.find()
```

```
db.crud2.find
```

```
(
```

```
{ b: 456}
```

```
)
```

```
MongoDB Enterprise > db.crud2.find({b:456})
{ "_id" : ObjectId("5fee018169910c9912e21401"), "b" : 456 }
{ "_id" : ObjectId("5fee018169910c9912e21404"), "b" : 456 }
{ "_id" : ObjectId("5fee018169910c9912e21407"), "b" : 456 }
```

FindOne()

```
db.crud2.findOne()
```

```
db.crud2.findOne
```

```
(
```

```
{ b: 456}
```

```
)
```

```
MongoDB Enterprise > db.crud2.findOne({b:456})  
{ "_id" : ObjectId("5ff1174afe368a5f724ef86c"), "b" : 456 }
```

Projection



```
SELECT nome, endereço  
FROM tabx  
WHERE  
... Seus filtros (nome = “Jose Pereira”)
```

```
Db.collect.find ( { where-filtros}, { select-projection } )  
  
Db.collect.find ( { nome:'Pedro'}, { nome:1, endereço:1 } )  
  
Db.collect.find ( { nome:'Pedro'}, { nome:1, endereço:1, _id:0 } )  
  
Db.collect.find ( { nome:'Pedro'}, { nome:1, endereço:0, _id:0 } )
```

Projection



```
MongoDB Enterprise > db.teste.findOne({estuda:true}, {nome:1, esportes:1})
{
    "_id" : ObjectId("5fed3897b1446e3c5d4502ae"),
    "nome" : "Pedro Pereira",
    "esportes" : [
        "tênis",
        "futebol",
        "videogame"
    ]
}
MongoDB Enterprise > db.teste.findOne({estuda:true}, {nome:1, esportes:1, _id:0})
{
    "nome" : "Pedro Pereira",
    "esportes" : [
        "tênis",
        "futebol",
        "videogame"
    ]
}
MongoDB Enterprise > db.teste.findOne({estuda:true}, {nome:1, esportes:0, _id:0})
uncaught exception: Error: error: {
    "ok" : 0,
    "errmsg" : "Cannot do exclusion on field esportes in inclusion projection",
```

Projection



```
MongoDB Enterprise > projet = {nome:1, esportes:1, _id:0}
{ "nome" : 1, "esportes" : 1, "_id" : 0 }
MongoDB Enterprise > filt = {estuda: true}
{ "estuda" : true }
MongoDB Enterprise > db.teste.findOne(filt, projet)
{
  "nome" : "Pedro Pereira",
  "esportes" : [
    "tênis",
    "futebol",
    "videogame"
  ]
}
```

Conclusão



Consultas no MongoDB

- ✓ `Find()`
- ✓ `FindOne()`

Próxima aula



01. ••

Updates

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 03 – Aula 03.05 – Updates

PROF.: RICARDO BRITO ALVES

Nesta aula



- ❑ Updates no MongoDB

Updates

- ✓ UpdateOne
- ✓ UpdateMany
- ✓ ReplaceOne

UpdateOne

Desde a versão 3.2.

Definição: db.collection.updateOne(filter, update, options)

```
db.collection.updateOne(  
  <filter>,  
  <update>,  
  {  
    upsert: <boolean>, // se não encontrar, vai inserir  
    writeConcern: <document>,  
    collation: <document>,  
    arrayFilters: [ <filterdocument1>, ... ],  
    hint: <document|string> // Available starting in MongoDB 4.2.1  
  }  
)
```

UpdateOne



Definição: db.collection.updateOne(filter, update, options)

```
MongoDB Enterprise > db.funcionarios.find()
{ "_id" : ObjectId("5fee12cae88d88a848a7f5b6"), "nome" : "Jose Pereira", "sexo" : "M", "escolaridade" : "S", "depto" : { "deptnome" : "A", "Local" : "X" } }
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b7"), "nome" : "Carlos Pereira", "sexo" : "M", "escolaridade" : "M", "depto" : { "deptnome" : "A", "Local" : "Y" } }
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b8"), "nome" : "Maria Pereira", "sexo" : "F", "escolaridade" : "M", "depto" : { "deptnome" : "A", "Local" : "Y" } }
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b9"), "nome" : "Perla Nascimento", "sexo" : "F", "escolaridade" : "S", "depto" : { "deptnome" : "B", "Local" : "Y" } }
```

```
MongoDB Enterprise > db.funcionarios.findOne({nome: /Perla/})
{
  "_id" : ObjectId("5fee12d6e88d88a848a7f5b9"),
  "nome" : "Perla Nascimento",
  "sexo" : "F",
  "escolaridade" : "S",
  "depto" : {
    "deptnome" : "B",
    "Local" : "Y"
  }
}
MongoDB Enterprise > db.funcionarios.updateOne({nome: /Perla/}, {$set:{salario: 1000}})
{
  "acknowledged" : true,
  "matchedCount" : 1,
  "modifiedCount" : 1
}
```

UpdateMany



Desde a versão 3.2.

```
db.funcionarios.updateMany({nome: /a/}, {$set:{salario: 2000}})
```

```
MongoDB Enterprise > db.funcionarios.find({nome: /a/})
{ "_id" : ObjectId("5fee12cae88d88a848a7f5b6"), "nome" : "Jose Pereira", "sexo" : "M", "escolaridade" : "S", "depto" : { "deptnome" : "A", "Local" : "X" } }
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b7"), "nome" : "Carlos Pereira", "sexo" : "M", "escolaridade" : "M", "depto" : { "deptnome" : "A", "Local" : "Y" } }
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b8"), "nome" : "Maria Pereira", "sexo" : "F", "escolaridade" : "M", "depto" : { "deptnome" : "A", "Local" : "Y" } }
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b9"), "nome" : "Perla Nascimento", "sexo" : "F", "escolaridade" : "S", "depto" : { "deptnome" : "B", "Local" : "Y" }, "salario" : 1000 }
MongoDB Enterprise > db.funcionarios.updateMany({nome: /a/}, {$set:{salario: 2000}})
{ "acknowledged" : true, "matchedCount" : 4, "modifiedCount" : 4 }
MongoDB Enterprise > db.funcionarios.find({nome: /a/})
{ "_id" : ObjectId("5fee12cae88d88a848a7f5b6"), "nome" : "Jose Pereira", "sexo" : "M", "escolaridade" : "S", "depto" : { "deptnome" : "A", "Local" : "X" }, "salario" : 2000 }
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b7"), "nome" : "Carlos Pereira", "sexo" : "M", "escolaridade" : "M", "depto" : { "deptnome" : "A", "Local" : "Y" }, "salario" : 2000 }
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b8"), "nome" : "Maria Pereira", "sexo" : "F", "escolaridade" : "M", "depto" : { "deptnome" : "A", "Local" : "Y" }, "salario" : 2000 }
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b9"), "nome" : "Perla Nascimento", "sexo" : "F", "escolaridade" : "S", "depto" : { "deptnome" : "B", "Local" : "Y" }, "salario" : 2000 }
```

ReplaceOne



Desde a versão 3.2.

```
db.funcionarios.replaceOne({nome: /Perla/}, {salario: 5000})
```

```
MongoDB Enterprise > db.funcionarios.find({nome: /Perla/})
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b9"), "nome" : "Perla Nascimento", "sexo" : "F", "escolaridade" : "S", "depto" : { "deptnome" : "B", "Local" : "Y" }, "salario" : 2000 }
MongoDB Enterprise > db.funcionarios.replaceOne({nome: /Perla/}, {salario: 5000})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Conclusão



Updates em dados no MongoDB

- ✓ UpdateOne
- ✓ UpdateMany
- ✓ ReplaceOne

Próxima aula



01. ••

Deletes

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 03 – Aula 03.06 – Deletes

PROF.: RICARDO BRITO ALVES

Nesta aula



- ❑ Deletes no MongoDB

Deletes

- ✓ DeleteOne
- ✓ DeleteMany

DeleteOne

Desde a versão 3.2.

Definição: db.collection.deleteOne(filter)

```
const query = { "name": "lego" };
```

```
itemsCollection.deleteOne(query)
```

DeleteOne

```
db.crud3.deleteOne({a: 123})
```

```
MongoDB Enterprise > db.crud3.insertMany([
... {a: 123},
... {b: 456},
... {c: 789},
... {a: 123},
... {b: 456},
... {c: 789},
... {a: 123},
... {b: 456},
... {c: 789},
... {e: 123},
... {f: 456},
... {g: 789},
... ])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5fee31e2e88d88a848a7f5ba"),
    ObjectId("5fee31e2e88d88a848a7f5bb"),
    ObjectId("5fee31e2e88d88a848a7f5bc"),
    ObjectId("5fee31e2e88d88a848a7f5bd"),
    ObjectId("5fee31e2e88d88a848a7f5be"),
    ObjectId("5fee31e2e88d88a848a7f5bf"),
    ObjectId("5fee31e2e88d88a848a7f5c0"),
    ObjectId("5fee31e2e88d88a848a7f5c1"),
    ObjectId("5fee31e2e88d88a848a7f5c2"),
    ObjectId("5fee31e2e88d88a848a7f5c3"),
    ObjectId("5fee31e2e88d88a848a7f5c4"),
    ObjectId("5fee31e2e88d88a848a7f5c5")
  ]
}
```

```
MongoDB Enterprise > db.crud3.find({a: 123})
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5ba"), "a" : 123 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bd"), "a" : 123 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c0"), "a" : 123 }
```

```
MongoDB Enterprise > db.crud3.deleteOne({a: 123})
{ "acknowledged" : true, "deletedCount" : 1 }
MongoDB Enterprise > db.crud3.find({a: 123})
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bd"), "a" : 123 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c0"), "a" : 123 }
```

DeleteMany

```
db.crud3.deleteMany({a: 123})
```

```
MongoDB Enterprise > db.crud3.find({a: 123})
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bd"), "a" : 123 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c0"), "a" : 123 }
MongoDB Enterprise > db.crud3.deleteMany({a: 123})
{ "acknowledged" : true, "deletedCount" : 2 }
MongoDB Enterprise > db.crud3.find({a: 123})
```

DeleteMany

```
db.crud3.deleteMany({b: {$exists:true}})
```

```
MongoDB Enterprise > db.crud3.find({b: {$exists:true}})
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bb"), "b" : 456 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5be"), "b" : 456 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c1"), "b" : 456 }
MongoDB Enterprise > db.crud3.deleteMany({b: {$exists:true}})
{ "acknowledged" : true, "deletedCount" : 3 }
MongoDB Enterprise > db.crud3.find({b: {$exists:true}})
```

DeleteMany

Como um delete sem where...

```
db.crud3.deleteMany({ })
```

```
MongoDB Enterprise > db.crud3.find()
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bc"), "c" : 789 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bf"), "c" : 789 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c2"), "c" : 789 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c3"), "e" : 123 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c4"), "f" : 456 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c5"), "g" : 789 }
MongoDB Enterprise > db.crud3.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 6 }
MongoDB Enterprise > db.crud3.find()
```

Conclusão



Deletes no MongoDB

- ✓ DeleteOne
- ✓ DeleteMany

Próxima aula



01. ••

Queries com Operadores

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 03 – Aula 03.07 – Queries com Operadores

PROF.: RICARDO BRITO ALVES

Nesta aula



- ❑ Consultas mais elaboradas utilizando operadores

Like



```
db.funcionarios.insertOne(  
  { nome: 'Jose Pereira', sexo: 'M', escolaridade: 'S',  
    depto: {deptnome: 'A', Local: 'X' } })
```

```
db.funcionarios.insertMany([  
  {nome: 'Carlos Pereira', sexo: 'M', escolaridade: 'M',  
    depto: {deptnome: 'A', Local: 'Y' }},  
  {nome: 'Maria Pereira', sexo: 'F', escolaridade: 'M',  
    depto: {deptnome: 'A', Local: 'Y' }},  
  {nome: 'Perla Nascimento', sexo: 'F', escolaridade: 'S',  
    depto: {deptnome: 'B', Local: 'Y' } } ])
```

```
MongoDB Enterprise > db.funcionarios.find()  
{ "_id" : ObjectId("5fee12cae88d88a848a7f5b6"), "nome" : "Jose Pereira", "sexo" : "M", "escolaridade" : "S", "depto" : { "deptnome" : "A", "Local" : "X" } }  
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b7"), "nome" : "Carlos Pereira", "sexo" : "M", "escolaridade" : "M", "depto" : { "deptnome" : "A", "Local" : "Y" } }  
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b8"), "nome" : "Maria Pereira", "sexo" : "F", "escolaridade" : "M", "depto" : { "deptnome" : "A", "Local" : "Y" } }  
{ "_id" : ObjectId("5fee12d6e88d88a848a7f5b9"), "nome" : "Perla Nascimento", "sexo" : "F", "escolaridade" : "S", "depto" : { "deptnome" : "B", "Local" : "Y" } }
```

Like



Consulta SQL

```
MongoDB Enterprise > db.funcionarios.find({}, {nome:1, _id:0})
{ "nome" : "Jose Pereira" }
{ "nome" : "Carlos Pereira" }
{ "nome" : "Maria Pereira" }
{ "nome" : "Perla Nascimento" }
MongoDB Enterprise > db.funcionarios.find({nome:/ereira/}, {nome:1, _id:0})
{ "nome" : "Jose Pereira" }
{ "nome" : "Carlos Pereira" }
{ "nome" : "Maria Pereira" }
```

db.funcionarios.find({nome:/ereira/})

ou

db.getCollection('funcionarios').find({nome:/ereira/})

OR



O operador **\$or** executa uma operação lógica OR em uma matriz de duas ou mais <expressions> e seleciona os documentos que satisfazem pelo menos uma das <expressions>.

```
{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }
```

Consulta SQL

Select * from funcionário where escolaridade = "S" or escolaridade = "M"

```
MongoDB Enterprise > db.funcionarios.find({}, {_id:0, nome:1, escolaridade:1})
{ "nome" : "Jose Pereira", "escolaridade" : "S" }
{ "nome" : "Carlos Pereira", "escolaridade" : "M" }
{ "nome" : "Maria Pereira", "escolaridade" : "M" }
{ "nome" : "Perla Nascimento", "escolaridade" : "S" }
```

OR



{ \$or: [{ <expression1> }, { <expression2> }, ... , { <expressionN> }] }

```
db.funcionarios.find({$or:  [{'escolaridade':  'M',  'escolaridade':  'S'}]},  
{_id:false, nome:true, endereço:true} )
```

```
db.funcionarios.find({$or:  [{'escolaridade':  'M'},  {'escolaridade':  'S'}]  },  
{_id:0, nome:1, endereço:1} )
```

```
MongoDB Enterprise > db.funcionarios.find({$or:  [{'escolaridade':  'M'},  {'escolaridade':  'S'}]  },  {_id:false, nome:true,  
escolaridade:true} )  
{ "nome" : "Jose Pereira", "escolaridade" : "S" }  
{ "nome" : "Carlos Pereira", "escolaridade" : "M" }  
{ "nome" : "Maria Pereira", "escolaridade" : "M" }
```

In, Not In

Consulta SQL

```
Select * from funcionário where escolaridade in ("S","M")
```

MongoDB

```
db.funcionarios.find
```

```
(  
  {escolaridade: {$in: ['M','S']}},  
  {_id:false, nome:true, endereço:true}  
)
```

```
db.funcionarios.find( {escolaridade: {$nin: ['M','S']}}, {_id:false, nome:true,  
endereço:true} )
```

```
MongoDB Enterprise > db.funcionarios.find( {escolaridade: {$in: ['M','S']}}, {_id:false, nome:true, endereço:true} )  
{ "nome" : "Jose Pereira" }  
{ "nome" : "Carlos Pereira" }  
{ "nome" : "Maria Pereira" }  
{ "nome" : "Perla Nascimento" }
```

Operadores



Name	Description
\$eq	Matches values that are equal to a specified value.
\$gt	Matches values that are greater than a specified value.
\$gte	Matches values that are greater than or equal to a specified value.
\$in	Matches any of the values specified in an array.
\$lt	Matches values that are less than a specified value.
\$lte	Matches values that are less than or equal to a specified value.
\$ne	Matches all values that are not equal to a specified value.
\$nin	Matches none of the values specified in an array.

Operadores



```
db.funcionarios.find
```

```
(  
  {escolaridade: {$ne: 'M'}, sexo: {$gte: 'M'}},  
  {_id:0, nome:1, endereço:1}  
)
```

```
MongoDB Enterprise > db.funcionarios.find( {}, {_id:0, nome:1, endereço:1} )  
{ "nome" : "Jose Pereira" }  
{ "nome" : "Carlos Pereira" }  
{ "nome" : "Maria Pereira" }  
{ "nome" : "Perla Nascimento" }  
MongoDB Enterprise > db.funcionarios.find( {escolaridade: {$ne: 'M'}, sexo: {$gte: 'M'}} , {_id:0, nome:1, endereço:1} )  
{ "nome" : "Jose Pereira" }
```

Conclusão



- ✓ Consultas mais elaboradas utilizando operadores

Próxima aula



01. ••

Aggregation

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 03 – Aula 03.08 – Aggregation

PROF.: RICARDO BRITO ALVES

Nesta aula



- Agregação

Aggregate Count



Consulta SQL

```
Select count(*) from funcionário
```

MongoDB

```
db.funcionarios.aggregate(  
  {$group: {'_id': null, 'count': {$sum:1}} }  
)
```

```
MongoDB Enterprise > db.funcionarios.aggregate( {$group: {'_id': null, 'count': {$sum:1}} } )  
[{"_id" : null, "count" : 4}]
```

```
db.funcionarios.aggregate(  
  {$group: {'_id': '$salario', 'count': {$sum:1}} }  
)
```

```
MongoDB Enterprise > db.funcionarios.aggregate( {$group: {'_id': '$salario', 'count': {$sum:1}} } )  
[{"_id" : 5000, "count" : 1}, {"_id" : 2000, "count" : 3}]
```

Aggregate Avg



```
db.funcionarios.aggregate  
(  
  {$group: {'_id': '$salario', 'media': {$avg:'$salario'}} }  
)
```

```
MongoDB Enterprise > db.funcionarios.aggregate( {$group: {'_id': '$salario', 'media': {$avg:'$salario'}} } )  
{ "_id" : 5000, "media" : 5000 }  
{ "_id" : 2000, "media" : 2000 }
```

Aggregate Sum



```
db.funcionarios.aggregate  
(  
{$group: {'_id': '$salario', 'soma': {$sum:'$salario'}} }  
)
```

```
MongoDB Enterprise > db.funcionarios.aggregate( {$group: {'_id': '$salario', 'soma': {$sum:'$salario'}} } )  
{ "_id" : 5000, "soma" : 5000 }  
{ "_id" : 2000, "soma" : 6000 }
```

Conclusão



- ✓ Exemplos de comandos de agregação

Próxima aula



01. ••

Lookup

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 03 – Aula 03.09 – Lookup (Aggregation)

PROF.: RICARDO BRITO ALVES

Nesta aula



- ❑ Lookup - agregação

\$lookup (aggregation)

Executa uma junção entre collections.

Para cada documento de entrada, o estágio \$ lookup adiciona um novo campo de array cujos elementos são os documentos correspondentes da coleção “unida”.

```
{  
  $lookup:  
    {  
      from: <collection to join>,  
      localField: <field from the input documents>,  
      foreignField: <field from the documents of the "from" collection>,  
      as: <output array field>  
    }  
}
```

\$lookup (aggregation)

Essa operação poderia corresponder ao seguinte pseudo-SQL:

```
SELECT *, <output array field>
FROM collection
WHERE <output array field> IN (SELECT *
    FROM <collection to join>
    WHERE <foreignField>= <collection.localField>);
```

\$lookup - exemplo

```
db.orders.insertMany([
  { "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2 },
  { "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1 },
  { "_id" : 3 }
])

db.inventory.insertMany([
  { "_id" : 1, "sku" : "almonds", "description": "product 1", "instock" : 120 },
  { "_id" : 2, "sku" : "bread", "description": "product 2", "instock" : 80 },
  { "_id" : 3, "sku" : "cashews", "description": "product 3", "instock" : 60 },
  { "_id" : 4, "sku" : "pecans", "description": "product 4", "instock" : 70 },
  { "_id" : 5, "sku": null, "description": "Incomplete" },
  { "_id" : 6 }
])
```

\$lookup - exemplo

```
db.orders.aggregate([
  {
    $lookup:
      {
        from: "inventory",
        localField: "item",
        foreignField: "sku",
        as: "inventory_docs"
      }
  }
])
```

\$lookup - exemplo

```
MongoDB Enterprise > db.orders.insertMany([
...   { "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2 },
...   { "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1 },
...   { "_id" : 3 }
... ])
{ "acknowledged" : true, "insertedIds" : [ 1, 2, 3 ] }
MongoDB Enterprise > db.inventory.insertMany([
...   { "_id" : 1, "sku" : "almonds", "description": "product 1", "instock" : 120 },
...   { "_id" : 2, "sku" : "bread", "description": "product 2", "instock" : 80 },
...   { "_id" : 3, "sku" : "cashews", "description": "product 3", "instock" : 60 },
...   { "_id" : 4, "sku" : "pecans", "description": "product 4", "instock" : 70 },
...   { "_id" : 5, "sku": null, "description": "Incomplete" },
...   { "_id" : 6 }
... ])
{ "acknowledged" : true, "insertedIds" : [ 1, 2, 3, 4, 5, 6 ] }
```

\$lookup - exemplo

```
MongoDB Enterprise > db.orders.aggregate([
...   {
...     $lookup:
...       {
...         from: "inventory",
...         localField: "item",
...         foreignField: "sku",
...         as: "inventory_docs"
...       }
...   }
... ])
{
  "_id" : 1,
  "item" : "almonds",
  "price" : 12,
  "quantity" : 2,
  "inventory_docs" : [
    {
      "_id" : 1,
      "sku" : "almonds",
      "description" : "product 1",
      "instock" : 120
    }
  ]
}
{
  "_id" : 2,
  "item" : "pecans",
  "price" : 20,
  "quantity" : 1,
  "inventory_docs" : [
    {
      "_id" : 4,
      "sku" : "pecans",
      "description" : "product 4",
      "instock" : 70
    }
  ]
}
{
  "_id" : 3,
  "inventory_docs" : [
    {
      "_id" : 5,
      "sku" : null,
      "description" : "Incomplete"
    },
    {
      "_id" : 6
    }
  ]
}
```

```
MongoDB Enterprise > db.orders.aggregate([
...   {
...     $lookup:
...       {
...         from: "inventory",
...         localField: "item",
...         foreignField: "sku",
...         as: "inventory_docs"
...       }
...   }
... ]).pretty()
{
  "_id" : 1,
  "item" : "almonds",
  "price" : 12,
  "quantity" : 2,
  "inventory_docs" : [
    {
      "_id" : 1,
      "sku" : "almonds",
      "description" : "product 1",
      "instock" : 120
    }
  ]
}
```

Conclusão



- ✓ Lookup

Próxima aula



01. ••

Modelagem

02. ••

03. ••

04. ••

Bancos de Dados NoSQL

Capítulo 04 – Modelagem e Relacionamento

PROF.: RICARDO BRITO ALVES



Banco de Dados NoSQL

Capítulo 04 – Aula 04.01 – Modelagem de Dados

PROF.: RICARDO BRITO ALVES



Banco de Dados NoSQL

Capítulo 04 – Aula 04.02 – MER e DER

PROF.: RICARDO BRITO ALVES

Nesta aula



- MER
- DER

MER x DER



MER – Modelo Entidade-Relacionamento é um diagrama simplificado:

- Não possui atributos.
- Não possui cardinalidade.

DER – Diagrama Entidade-Relacionamento é mais detalhado:

- Possui atributos.
- Possui cardinalidade.

MER x DER



MER - Modelo Entidade-Relacionamento: Conjunto de conceitos e elementos de modelagem que o projetista de banco de dados precisa conhecer. Modelo lógico.

DER – Diagrama Entidade-Relacionamento: resultado do processo de modelagem executado pelo projetista de dados que conhece o MER. Modelo físico.

Entidades

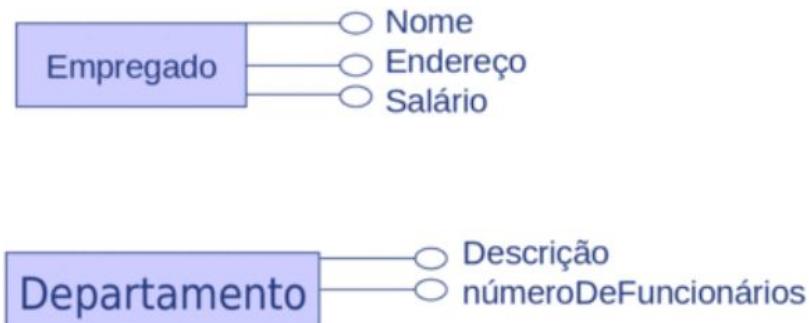
É um conjunto de objetos do mundo real sobre a qual deseja-se manter informações no banco de dados.

Exemplos:

- Cliente.
- Fornecedor.
- Departamento.
- Estoque.

Atributos

São informações a respeito de uma Entidade.



Atributo Chave: toda Entidade deve ter um atributo chave para identifica-la de forma única. Pode ser um conjunto de atributos (chave composta).

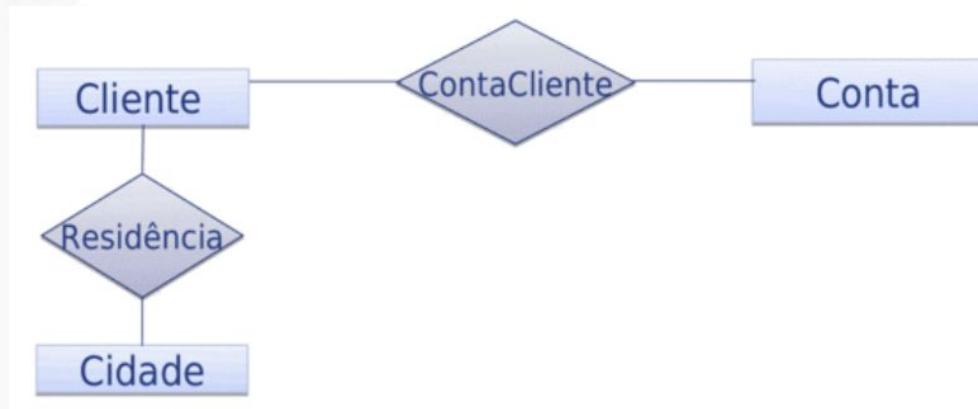
Relacionamentos

- É uma associação entre Entidades.
- Pode ser representado por um losango.



Relacionamentos

- Exemplos:



Cardinalidade

A Cardinalidade define a quantidade de elementos de uma Entidade associada com a quantidade de elementos de outra Entidade.

- 1:1 (um para um)
- 1:N (um para muitos)
- N:N (muitos para muitos)

Um para Um (1:1)

Temos exemplo da figura abaixo apenas ilustrativo, pois na prática não é o que acontece.

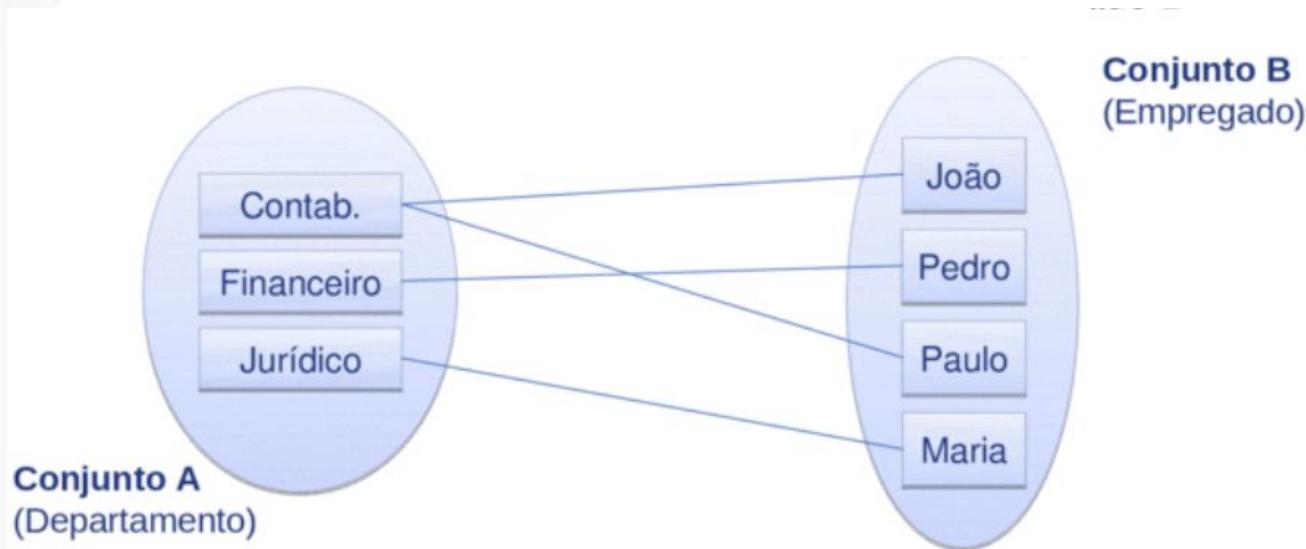


Seria melhor exemplificar a relação entre as entidades

Aluno x ID_Aluno.

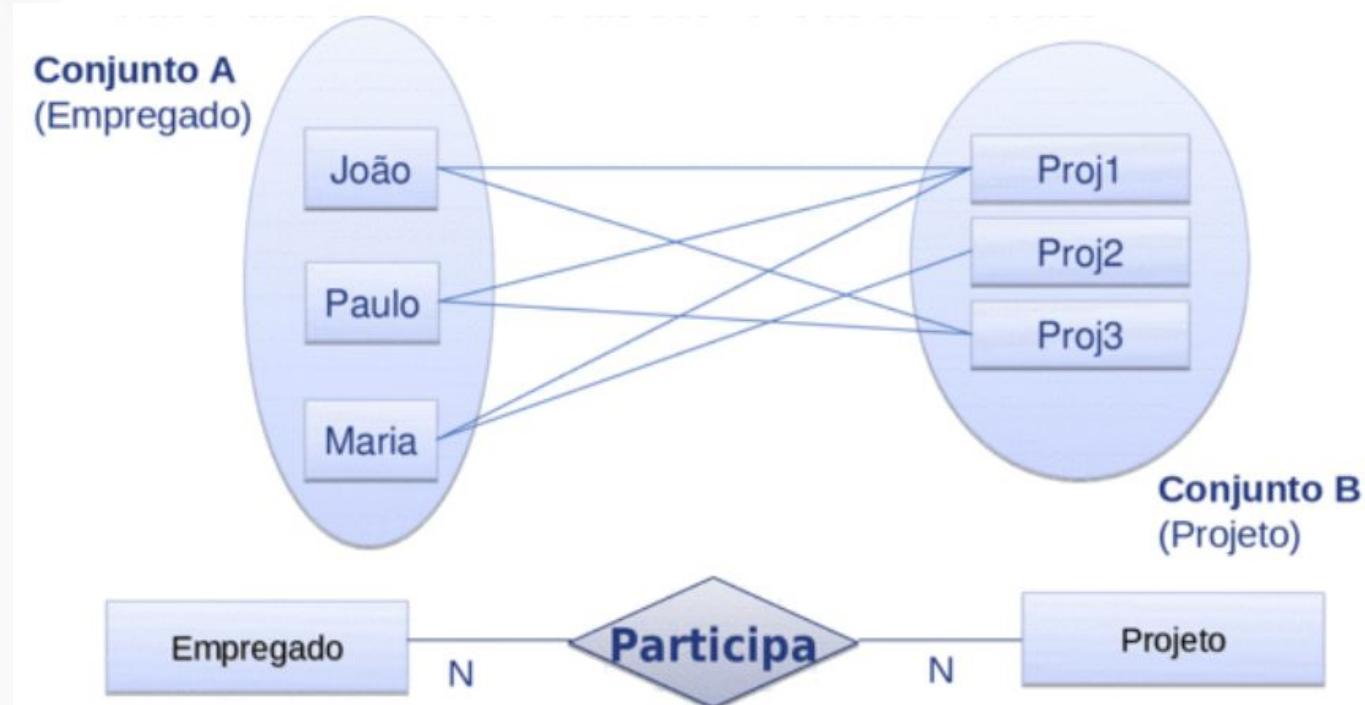
Um para Muitos (1:N)

IGTI



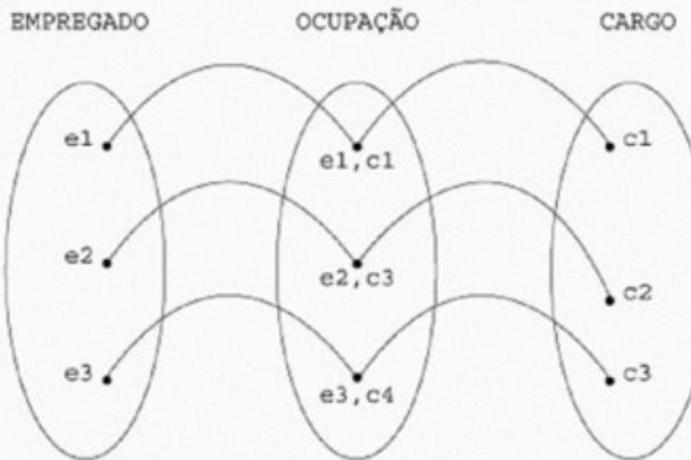
Muitos para Muitos (N:N)

IGTI



Muitos para Muitos (N:N)

IGTI



Conclusão



- ✓ MER
- ✓ DER

Próxima aula



01.

03.

Relação Um para Um

02.

04.



Banco de Dados NoSQL

Capítulo 04 – Aula 04.03 – Um para Um

PROF.: RICARDO BRITO ALVES

Nesta aula



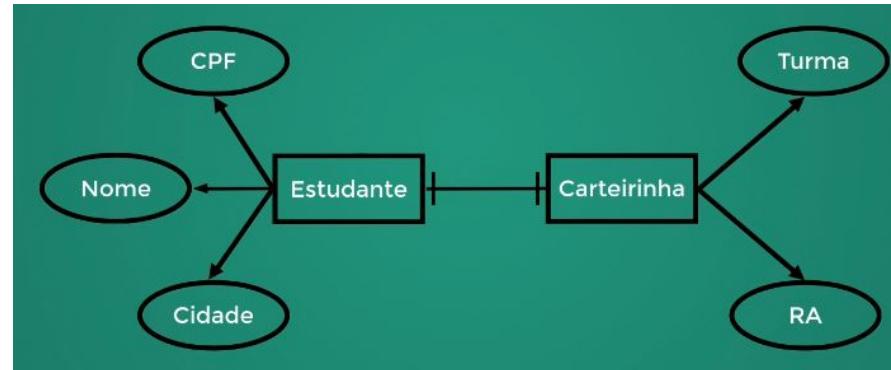
- ❑ Relação Um para Um – One to One

Um para Um

One to One

Estudante

- id
- nome
- cpf
- cidade
- carteirinha_id



Carteirinha

- id_carteirinha
- turma
- RA

Um para Um



One to One

```
{  
  "id": 1,  
  "nome": "Pedro Pereira",  
  "cpf": 12345645,  
  "cidade": "São Paulo",  
  "carteirinha_id": 8  
}  
  
{  
  "id_carteirinha": 8,  
  "turma": "210B",  
  "RA": 1234  
}
```

Um para Um



Embedded documents

```
{  
  "id": 1,  
  "nome": "Pedro Pereira",  
  "cpf": 12345645,  
  "cidade": "São Paulo",  
  "carteirinha": {  
    "_id": 8,  
    "turma": "210B",  
    "RA": 1234  
  }  
}
```

Conclusão



- ✓ Um para Um

Próxima aula



01. ••

Relação Um para Muitos

03. ••

02. ••

04. ••



Banco de Dados NoSQL

Capítulo 04 – Aula 04.04 – Um para Muitos

PROF.: RICARDO BRITO ALVES

Nesta aula



- ❑ Relação Um para Muitos – One to Many

Um para Muitos

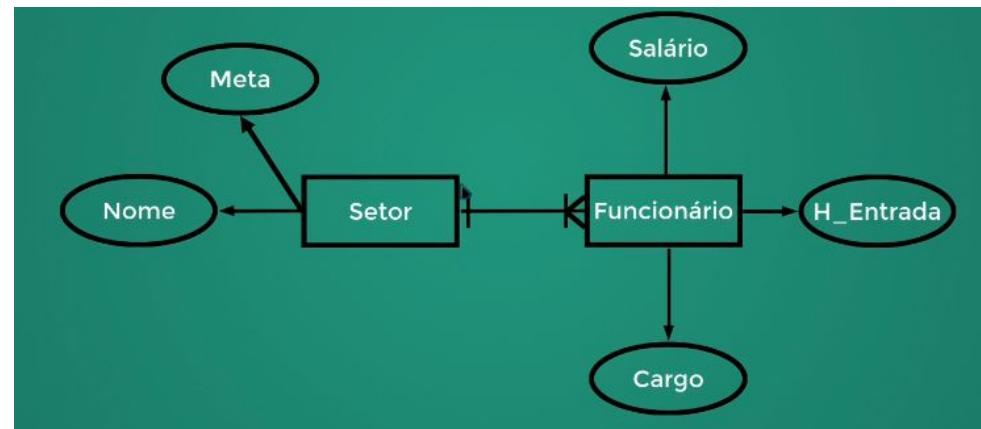
One to Many

Setor

- id
- meta
- nome

Funcionario

- id
- salario
- turno
- cargo
- id_setor



Um para Muitos



One to Many

```
{  
  "id": 1,  
  "nome": "José Pereira",  
  "salario": 1400,  
  "turno": "tarde",  
  "cargo": "vendedor pleno",  
  "id_setor": 3  
}  
  
{  
  "id": 3,  
  "meta": "40000",  
  "nome": "depto Vendas"  
}
```

Um para Muitos - Embedded Documents



```
{   "id": 3,  
    "meta": "40000",  
    "nome": "depto Vendas"  
    "funcionarios": {  
        "id": 1,  
        "nome": "José Pereira",  
        "salario": 1400,  
        "turno": "tarde",  
        "cargo": "vendedor pleno"  
    },  
    {  
        "id": 2,  
        "nome": "Ana Pereira",  
        "salario": 1100,  
        "turno": "tarde",  
        "cargo": "vendedor pleno"  
    },  
    {  
        "id": 3,  
        "nome": "Carlos Pereira",  
        "salario": 2100,  
        "turno": "manhã",  
        "cargo": "vendedor senior"  
    }  
}
```

Um para Muitos - Considerações



Embedded Documents:

- Muitos funcionários por área.
- Embedded levando a área para dentro do funcionário.
- Dependendo use relacionamento (referência e join).

Conclusão



- ✓ Um para Muitos

Próxima aula



01. ••

Relação Muitos para Muitos

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 04 – Aula 04.05 – Muitos para Muitos

PROF.: RICARDO BRITO ALVES

Nesta aula



- ❑ Relação Muito para Muitos – Many to Many

Muitos para Muitos

IGTI

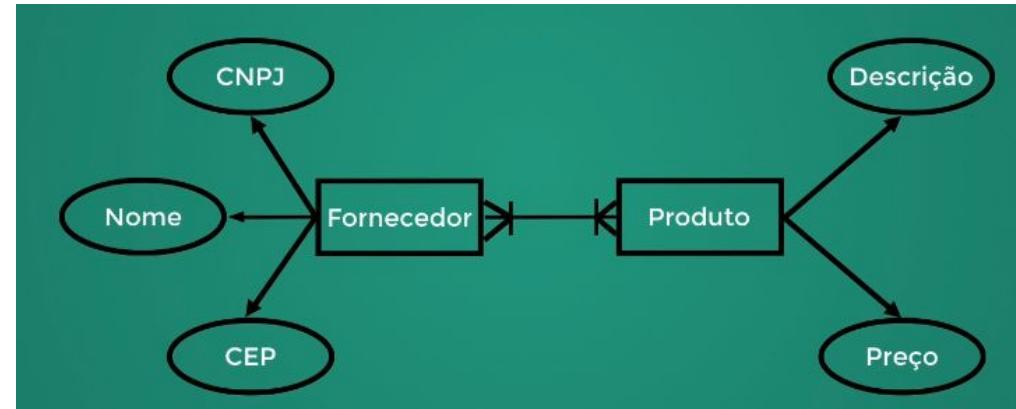
Many to Many

Fornecedor

- id
- cnpj
- nome
- cep

Produto

- id
- descricao
- preco



Muitos para Muitos



```
// Collection Fornecedores
{
  "_id": "f04",
  "cnpj": "165486984",
  "nome": "Fornecedor 1",
  "cep": "1098465"
}

{
  "_id": "f07",
  "cnpj": "98498151",
  "nome": "Fornecedor 2",
  "cep": "198498"
}
```

Muitos para Muitos

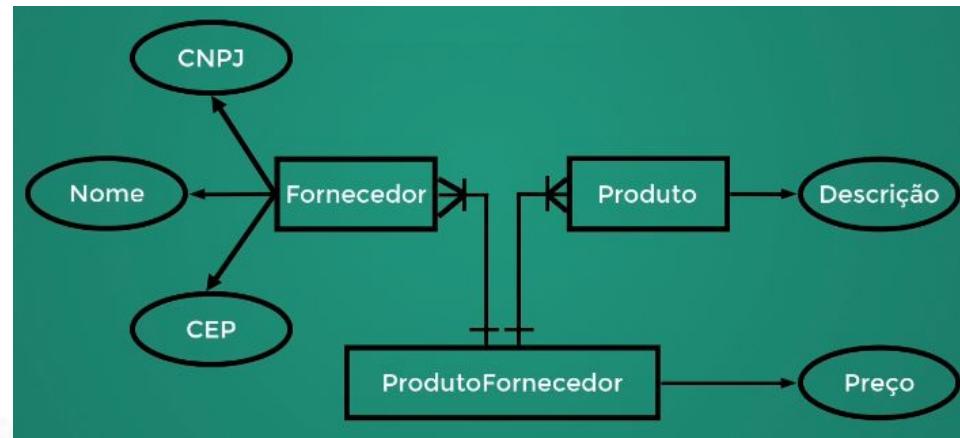


```
// Collection Produto
{
  "_id": "p16",
  "descricao": "Panela",
  "preco": 35.50
}
{
  "_id": "p21",
  "descricao": "Prato",
  "preco": 13
}
{
  "_id": "p47",
  "descricao": "Faqueiro",
  "preco": 117.50
}
```

Muitos para Muitos

FornecedorProduto

- fornecedor_id
- produto_id



Muitos para Muitos com Arrays



```
// Collection Fornecedores
{
    "_id": "f04",
    "cnpj": "165486984",
    "nome": "Fornecedor 1",
    "cep": "1098465",
    "produto_ids": ["p16", "p21"]
}

{
    "_id": "f07",
    "cnpj": "98498151",
    "nome": "Fornecedor 2",
    "cep": "198498",
    "produto_ids": ["p21", "p47"]
}
```

Muitos para Muitos com Arrays



// Collection Produtos

```
{  "_id": "p16",
  "descricao": "Panela",
  "preco": 45.50,
  "fornecedor_ids": ["f04"]
}
{  "_id": "p21",
  "descricao": "Prato",
  "preco": 14,
  "fornecedor_ids": ["f04", "f07"]
}
{  "_id": "p47",
  "descricao": "Faqueiro",
  "preco": 127.46,
  "fornecedor_ids": ["f07"]
}
```

Muitos para Muitos com Arrays

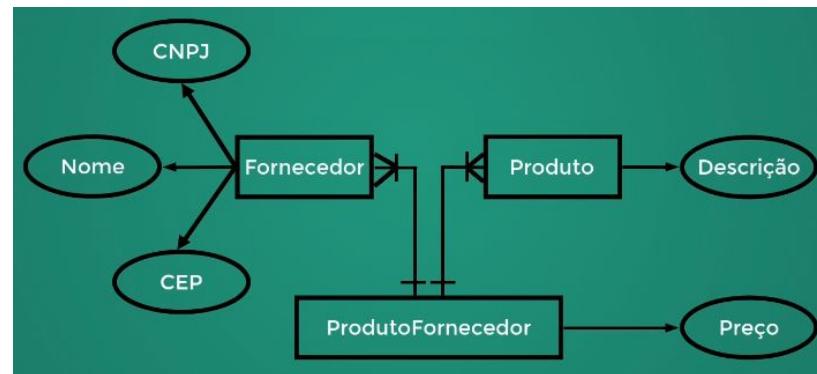
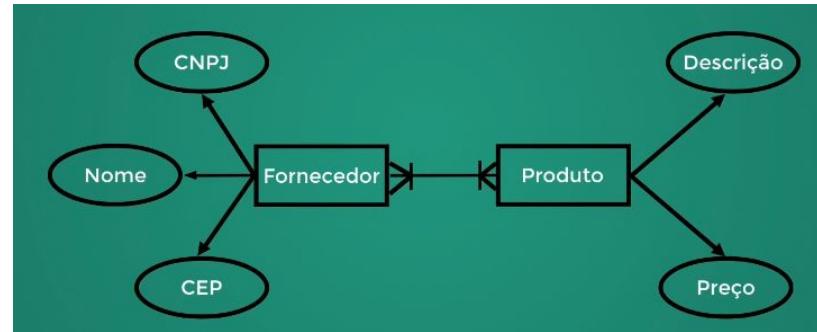


Integridade deve ser priorizada para que o dado não fique desatualizado entre as duas collections.

Muitos para Muitos

Conseguimos resolver a relação apenas com duas collections e não três.

Problema: Preço de referência no fornecedor e preço do produto (desconto).



Muitos para Muitos



Híbrido de muitos para muitos e embedded.

```
// Collection Fornecedor
{
    "_id": "f07",
    "cnpj": "98498151",
    "nome": "Fornecedor 2",
    "cep": "198498",
    "produtos": [          // array
        {
            "_id": "p21",
            "preco": 16
        },
        {
            "_id": "p47",
            "preco": 127.46
        }
    ]
}
```

Muitos para Muitos



```
// Collection Produto
{
  "_id": "p21",
  "descricao": "Prato",
  "fornecedores": [          // array
    {
      "_id": "f04",
      "preco": 12
    },
    {
      "_id": "f07",
      "preco": 16
    }
  ]
}
```

Conclusão



- ✓ Muitos para Muitos

Próxima aula



01. ••

Schema e Validation

02. ••

03. ••

04. ••

Bancos de Dados NoSQL

Capítulo 05 – Schema e Validation

PROF.: RICARDO BRITO ALVES



Banco de Dados NoSQL

Capítulo 05 – Aula 05.01 – Database e Collections

PROF.: RICARDO BRITO ALVES

Nesta aula

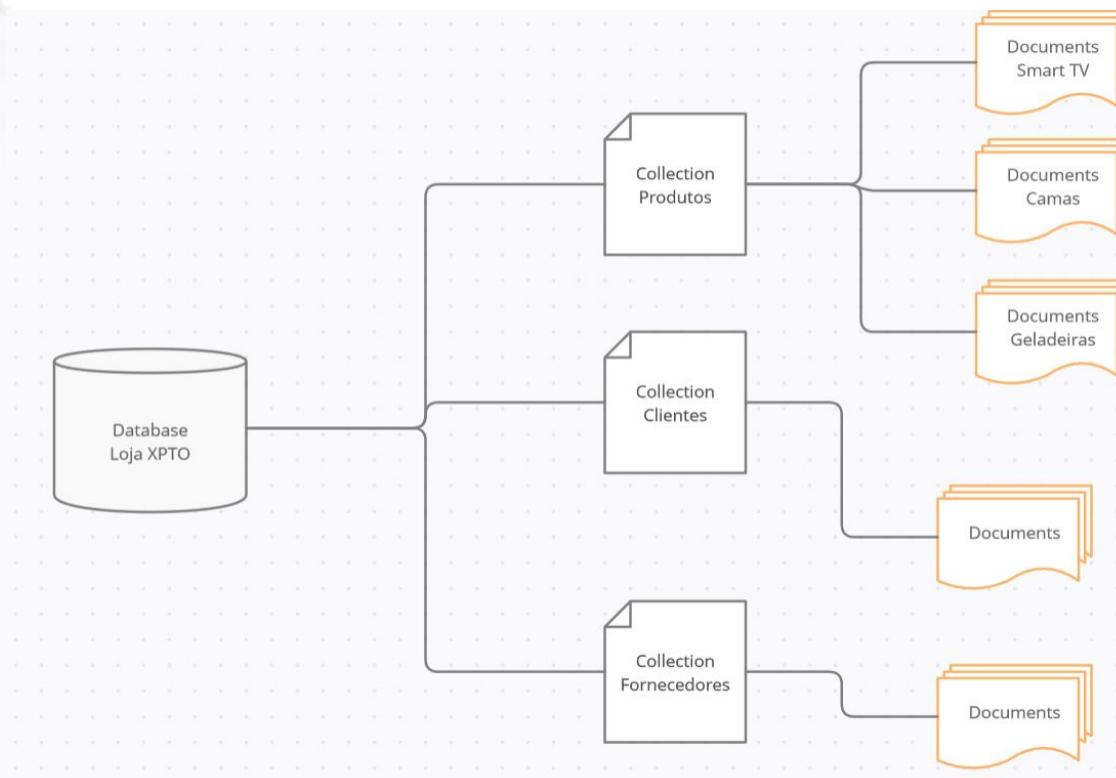


- Database
- Collections

Database x Collection x Documents



O MongoDB abstrai diversos comandos DDL.
Estruturas são criadas conforme estas se tornam necessárias.



Database x Collection x Documents

IGTI

Relational Database

Student_Id	Student_Name	Age	College
1001	Chaitanya	30	Beginnersbook
1002	Steve	29	Beginnersbook
1003	Negan	28	Beginnersbook



MongoDB

```
{  
    "_id": ObjectId("....."),  
    "Student_Id": 1001,  
    "Student_Name": "Chaitanya",  
    "Age": 30,  
    "College": "Beginnersbook"  
}  
{  
    "_id": ObjectId("....."),  
    "Student_Id": 1002,  
    "Student_Name": "Steve",  
    "Age": 29,  
    "College": "Beginnersbook"  
}  
{  
    "_id": ObjectId("....."),  
    "Student_Id": 1003,  
    "Student_Name": "Negan",  
    "Age": 28,  
    "College": "Beginnersbook"  
}
```

Database

Para se criar um database é só executar o comando `use <database>`

Use testeigti

Comando ***Show dbs*** mostra os databases existentes.

Comando ***db*** mostra o database corrente.

Como estamos usando um cliente (local) e não foi gravado nada nesse database ***testeigti*** (collections com dados) o Mongo Server não criou o database.

```
MongoDB Enterprise > use testeigti
switched to db testeigti
MongoDB Enterprise > show dbs
admin 0.000GB
config 0.000GB
igti 0.000GB
local 0.000GB
MongoDB Enterprise > db
testeigti
MongoDB Enterprise > show dbs
admin 0.000GB
config 0.000GB
igti 0.000GB
local 0.000GB
MongoDB Enterprise >
```

Database

Se executarmos o comando para criar uma collection, o database também será criado.

```
db.minhaCollection.insert({"valor": 123})
```

```
MongoDB Enterprise > db.minhaCollection.insert({"valor": 123})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > show dbs
admin      0.000GB
config     0.000GB
igti       0.000GB
local      0.000GB
testeigti  0.000GB
MongoDB Enterprise >
```

admin Database



```
MongoDB Enterprise > use admin
switched to db admin
MongoDB Enterprise > db.createUser(
...   {
...     user: "ricardo",
...     pwd: "igli",
...     roles: [ "userAdminAnyDatabase",
...               "dbAdminAnyDatabase",
...               "readWriteAnyDatabase"]
...   }
...
Successfully added user: {
  "user" : "ricardo",
  "roles" : [
    "userAdminAnyDatabase",
    "dbAdminAnyDatabase",
    "readWriteAnyDatabase"
  ]
}
MongoDB Enterprise > db.auth("ricardo", "igli"
1
MongoDB Enterprise > db.dropUser("ricardo")
true
MongoDB Enterprise >
```

```
MongoDB Enterprise > show collections
system.users
system.version
MongoDB Enterprise > db.system.users.find().pretty()
{
  "_id" : "admin.ricardo",
  "userId" : UUID("00205031-9459-4e9a-b5ae-60f7914fcraf"),
  "user" : "ricardo",
  "db" : "admin",
  "credentials" : {
    "SCRAM-SHA-1" : {
      "iterationCount" : 10000,
      "salt" : "QE/91JhE3q8IOLydqWEjmw==",
      "storedKey" : "wkAY5srG+6gRqCAjc5i7I8nrx70=",
      "serverKey" : "ej0TePJbg/3gPSbLKdWuIbTV4ao="
    },
    "SCRAM-SHA-256" : {
      "iterationCount" : 15000,
      "salt" : "MhvE118MfgAvThtgWJqoHg/LCRUoNjeATbvA4w==",
      "storedKey" : "i6PppPey22CtoK0wrGPzu7JT0GiOs4zMfc56uL95cE=",
      "serverKey" : "LRLNliBiDu3ErW6jupz3eoKk6ml15bFqCZFFja9jYiu="
    }
  },
  "roles" : [
    {
      "role" : "userAdminAnyDatabase",
      "db" : "admin"
    },
    {
      "role" : "dbAdminAnyDatabase",
      "db" : "admin"
    },
    {
      "role" : "readWriteAnyDatabase",
      "db" : "admin"
    }
  ]
}
MongoDB Enterprise >
```

config Database



```
MongoDB Enterprise > show collections
startup_log
MongoDB Enterprise > show dbs
admin      0.000GB
config     0.000GB
igti       0.000GB
local      0.000GB
testeigti  0.000GB
MongoDB Enterprise > use config
switched to db config
MongoDB Enterprise > show collections
system.sessions
MongoDB Enterprise > db.system.sessions.find().pretty()
{
    "_id" : {
        "id" : UUID("0dece178-779a-4a3c-8f8b-cbae4ee298c5"),
        "uid" : BinData(0,"47DEQpj8HB5a+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=")
    },
    "lastUse" : ISODate("2020-12-31T00:25:03.115Z")
}
{
    "_id" : {
        "id" : UUID("d08cadf7-279e-48c7-a6d9-e2db4ae3130b"),
        "uid" : BinData(0,"47DEQpj8HB5a+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=")
    },
    "lastUse" : ISODate("2020-12-31T00:39:22.042Z")
}
{
    "_id" : {
        "id" : UUID("4064017c-e64f-4b52-8a5e-865eceae4a30"),
        "uid" : BinData(0,"47DEQpj8HB5a+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=")
    },
    "lastUse" : ISODate("2020-12-31T00:53:18.373Z")
}
MongoDB Enterprise >
```

local Database

Cada instância do mongod tem seu próprio banco de dados local, que armazena os dados usados no processo de replicação e outros dados específicos da instância. local.startup_log

```
MongoDB Enterprise > use local
switched to db local
MongoDB Enterprise > show collections
startup_log
MongoDB Enterprise > db.startup_log.find().pretty()
```

Collection x Documents



Para se criar uma coleção, basta inserir um documento nela.

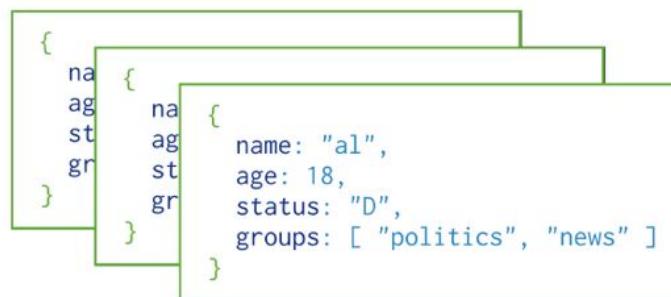
```
db.customers.insert({ nome: "sue", idade: 26, status: "A" })
```

Se quiser verificar quais coleções existem no banco de dados é só executar o comando **show collections** ou **show tables**.

One *document*



One *Collection*



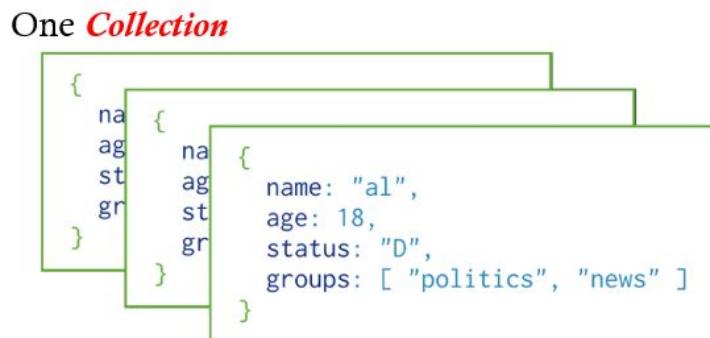
Collection

Para se criar uma coleção, basta inserir um documento nela.

```
db.customers.insert({ nome: "sue", idade: 26, status: "A" })
```

Ou executar o comando:

```
db.createCollections()
```



createCollection

```
db.createCollection( <name>,
{
  capped: <boolean>, // capped Collection
  autoIndexId: <boolean>, // está depreciado – não utilizar
  size: <number>, // capped Collection
  max: <number>, // capped Collection
  storageEngine: <document>,
  validator: <document>,
  validationLevel: <string>,
  validationAction: <string>,
  indexOptionDefaults: <document>,
  viewOn: <string>,      // Added in MongoDB 3.4
  pipeline: <pipeline>,    // Added in MongoDB 3.4
  collation: <document>,    // Added in MongoDB 3.4
  writeConcern: <document>
}
)
```

Conclusão



- ✓ Database
- ✓ Collection

Próxima aula



01. ••

Collection com validação

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 05 – Aula 05.02 – Collection com Validação

PROF.: RICARDO BRITO ALVES

Nesta aula



- ❑ Collection com validação

Collection



- Vamos criar uma collection chamada “carro” para análise.
- Cadastraremos o ano como inteiro e como string.

```
MongoDB Enterprise > db.carro.insert({modelo: 'KA', fabricante: 'Ford', ano: 2016})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.carro.insert({modelo: 'Onix', fabricante: 'Chevrolet', ano: '2017'})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.carro.insert({modelo: 'HB20', fabricante: 'Hyundai', ano: 2020})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.carro.find({ano:2016})
{ "_id" : ObjectId("5ff06fd6c6128ce3f0468150"), "modelo" : "KA", "fabricante" : "Ford", "ano" : 2016 }
MongoDB Enterprise > db.carro.find({ano:2020})
{ "_id" : ObjectId("5ff06fe5c6128ce3f0468152"), "modelo" : "HB20", "fabricante" : "Hyundai", "ano" : 2020 }
MongoDB Enterprise > db.carro.find({ano:2017})
MongoDB Enterprise > db.carro.find({ano:'2017'})
{ "_id" : ObjectId("5ff06fdec6128ce3f0468151"), "modelo" : "Onix", "fabricante" : "Chevrolet", "ano" : "2017" }
```

```
MongoDB Enterprise > db.carro.find({ano: {$gte: 2015}})
{ "_id" : ObjectId("5ff06fd6c6128ce3f0468150"), "modelo" : "KA", "fabricante" : "Ford", "ano" : 2016 }
{ "_id" : ObjectId("5ff06fe5c6128ce3f0468152"), "modelo" : "HB20", "fabricante" : "Hyundai", "ano" : 2020 }
```

Collection - Validator

```
db.createCollection("students", {  
  validator: {  
    $jsonSchema: {  
      bsonType: "object",  
      required: [ "name", "year", "major", "address" ],  
      properties: {  
        name: {  
          bsonType: "string",  
          description: "must be a string and is required"  
        },  
        year: {  
          bsonType: "int",  
          minimum: 2017,  
          maximum: 3017,  
          description: "must be an integer in [ 2017, 3017 ] and is required"  
        }  
      }  
    }  
  }  
})
```

Collection - Validator



```
MongoDB Enterprise > db.carro.drop()
true
MongoDB Enterprise > db.createCollection("carro", {
...   validator: {
...     $jsonSchema: {
...       bsonType: "object",
...       required: ["modelo", "ano"],
...       properties: {
...         modelo: {
...           bsonType: "string",
...           description: "Modelo é string e é obrigatório."
...         },
...         fabricante: {
...           bsonType: "string",
...           minLength: 3,
...           maxLength: 30,
...           description: "Deve ser string"
...         },
...         ano: {
...           bsonType: "int",
...           minimum: 2015,
...           maximum: 2025,
...           description: "Inteiro entre [ 2015, 2025 ] e é obrigatório."
...         }
...       }
...     }
...   }
... })
{ "ok" : 1 }
```

Collection - Validator



```
MongoDB Enterprise > db.carro.insert({modelo: 'HB20', fabricante: 'Hyundai', ano: 2020})
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 121,
        "errmsg" : "Document failed validation"
    }
})
```

```
MongoDB Enterprise > db.carro.insert({modelo: 'HB20', fabricante: 'Hyundai', ano: NumberInt(2020)})
WriteResult({ "nInserted" : 1 })
```

Collection - Validator



```
MongoDB Enterprise > db.carro.insert({modelo: 'HB20', fabricante: 'Hyundai', ano: NumberInt(2020)})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.carro.insert({modelo: 'KA', fabricante: 'Ford', ano: NumberInt(2017)})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.carro.insert({modelo: 'S10', fabricante: 'GM', ano: NumberInt(2017)})
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 121,
        "errmsg" : "Document failed validation"
    }
})
```

```
MongoDB Enterprise > db.carro.insert({modelo: 'Onix', fabricante: 'Chevrolet', ano: '2017'})
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 121,
        "errmsg" : "Document failed validation"
    }
})
MongoDB Enterprise > db.carro.insert({modelo: 'Onix', fabricante: 'Chevrolet', ano: NumberInt('2017'))}
WriteResult({ "nInserted" : 1 })
```

Conclusão



- ✓ Collection com validação

Próxima aula



01. ••

Capped Collection

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 05 – Aula 05.03 – Capped Collection

PROF.: RICARDO BRITO ALVES

Nesta aula



- Capped Collection

Capped Collection



```
db.createCollection( <name>,
{
  capped: <boolean>,
  autoIndexId: <boolean>,
  size: <number>,
  max: <number>,
  storageEngine: <document>,
  validator: <document>,
  validationLevel: <string>,
  validationAction: <string>,
  indexOptionDefaults: <document>,
  viewOn: <string>,           // Added in MongoDB 3.4
  pipeline: <pipeline>,       // Added in MongoDB 3.4
  collation: <document>,      // Added in MongoDB 3.4
  writeConcern: <document>
}
)
```

Capped Collection

Capped: boolean, opcional. Para criar uma coleção limitada, especifique true. Se você especificar true, também deve definir um tamanho máximo no campo size.

Size: number, opcional. Especifique um tamanho máximo em bytes para uma coleção limitada. Quando uma coleção limitada atinge seu tamanho máximo, o MongoDB remove os documentos mais antigos para liberar espaço para os novos documentos. O campo de tamanho é obrigatório para coleções limitadas e ignorado para outras coleções.

Max: number, opcional. O número máximo de documentos permitidos na coleção limitada. ***O limite de tamanho tem precedência sobre este limite.*** Se uma coleção limitada atingir o limite de tamanho antes de atingir o número máximo de documentos, o MongoDB remove os documentos antigos. Se você preferir usar o limite máximo, certifique-se de que o limite de tamanho, que é necessário para uma coleção limitada, seja suficiente para conter o número máximo de documentos.

Capped Collection



Max: number, opcional. O número máximo de documentos permitidos na coleção limitada. ***O limite de tamanho tem precedência sobre este limite.***

```
db.createCollection("log_simulado", {  
    capped: true,  
    size: 1024, // em bytes  
    max: 5  
})
```

```
db.log_simulado.insert({n: 1, desc: '1xxxxxx'})  
db.log_simulado.insert({n: 2, desc: '2xxxxxx'})  
db.log_simulado.insert({n: 3, desc: '3xxxxxx'})  
db.log_simulado.insert({n: 4, desc: '4xxxxxx'})  
db.log_simulado.insert({n: 5, desc: '5xxxxxx'})  
db.log_simulado.insert({n: 6, desc: '6xxxxxx'})
```

Capped Collection



```
MongoDB Enterprise > db.createCollection("log_simulado", {  
... capped: true,  
... size: 1024, // em bytes  
... max: 5  
... })  
{ "ok" : 1 }  
MongoDB Enterprise > db.log_simulado.insert({n: 1, desc: '1xxxxxx'})  
WriteResult({ "nInserted" : 1 })  
MongoDB Enterprise > db.log_simulado.insert({n: 2, desc: '2xxxxxx'})  
WriteResult({ "nInserted" : 1 })  
MongoDB Enterprise > db.log_simulado.insert({n: 3, desc: '3xxxxxx'})  
WriteResult({ "nInserted" : 1 })  
MongoDB Enterprise > db.log_simulado.insert({n: 4, desc: '4xxxxxx'})  
WriteResult({ "nInserted" : 1 })  
MongoDB Enterprise > db.log_simulado.insert({n: 5, desc: '5xxxxxx'})  
WriteResult({ "nInserted" : 1 })  
MongoDB Enterprise > db.log_simulado.find()  
{ "_id" : ObjectId("5ff08879c6128ce3f046816b"), "n" : 1, "desc" : "1xxxxxx" }  
{ "_id" : ObjectId("5ff08880c6128ce3f046816c"), "n" : 2, "desc" : "2xxxxxx" }  
{ "_id" : ObjectId("5ff08886c6128ce3f046816d"), "n" : 3, "desc" : "3xxxxxx" }  
{ "_id" : ObjectId("5ff0888bc6128ce3f046816e"), "n" : 4, "desc" : "4xxxxxx" }  
{ "_id" : ObjectId("5ff08890c6128ce3f046816f"), "n" : 5, "desc" : "5xxxxxx" }  
MongoDB Enterprise > db.log_simulado.insert({n: 6, desc: '6xxxxxx'})  
WriteResult({ "nInserted" : 1 })  
MongoDB Enterprise > db.log_simulado.find()  
{ "_id" : ObjectId("5ff08880c6128ce3f046816c"), "n" : 2, "desc" : "2xxxxxx" }  
{ "_id" : ObjectId("5ff08886c6128ce3f046816d"), "n" : 3, "desc" : "3xxxxxx" }  
{ "_id" : ObjectId("5ff0888bc6128ce3f046816e"), "n" : 4, "desc" : "4xxxxxx" }  
{ "_id" : ObjectId("5ff08890c6128ce3f046816f"), "n" : 5, "desc" : "5xxxxxx" }  
{ "_id" : ObjectId("5ff088a1c6128ce3f0468170"), "n" : 6, "desc" : "6xxxxxx" }
```

Conclusão



- ✓ Capped Collection

Próxima aula



01. ••

Object Document Mapper

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 05 – Aula 05.04 – Object Document Mapper (ODM)

PROF.: RICARDO BRITO ALVES

Nesta aula



- ORM – Object Relational Mapper
- ODM – Object Document Mapper

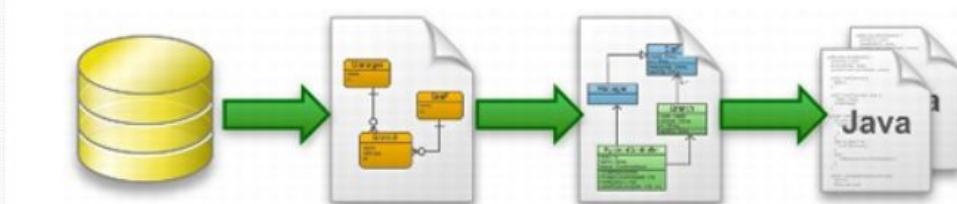
ORM, ODM

Qual o melhor jeito de interagir (conectar) com um banco de dados?

Existem duas abordagens:

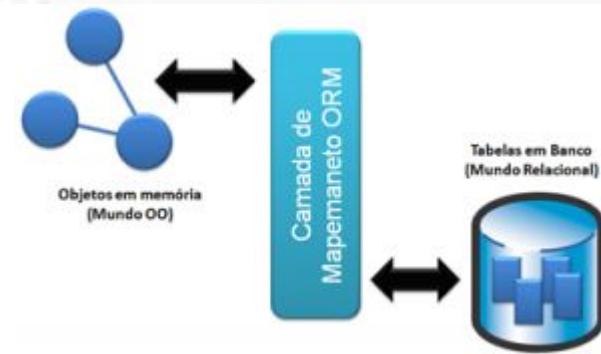
- Usando a linguagem de consulta nativa dos bancos de dados (por exemplo, SQL).
- Usando um ORM ou um ODM.

Segundo Krakowiak (2003), em um sistema de computação distribuída, middleware é definido como uma camada de software que reside entre o sistema operacional e as aplicações em cada ponto do sistema.



ORM, ODM

- ORM (Mapeamento Objeto Relacional).
- ODM (Documento Objeto Mapper).



São técnicas de programação para conversão de dados entre sistemas de tipos incompatíveis em bancos de dados e linguagens de programação orientadas a objetos.

Isso cria um "banco de dados de objeto virtual" que pode ser usado a partir de dentro da linguagem de programação.

- ORM - para bancos de dados relacionais.
- ODM - para NoSQL bancos de dados.

ORM



ORM (Object Relational Mapper) significa mapear um objeto com um mundo relacional.

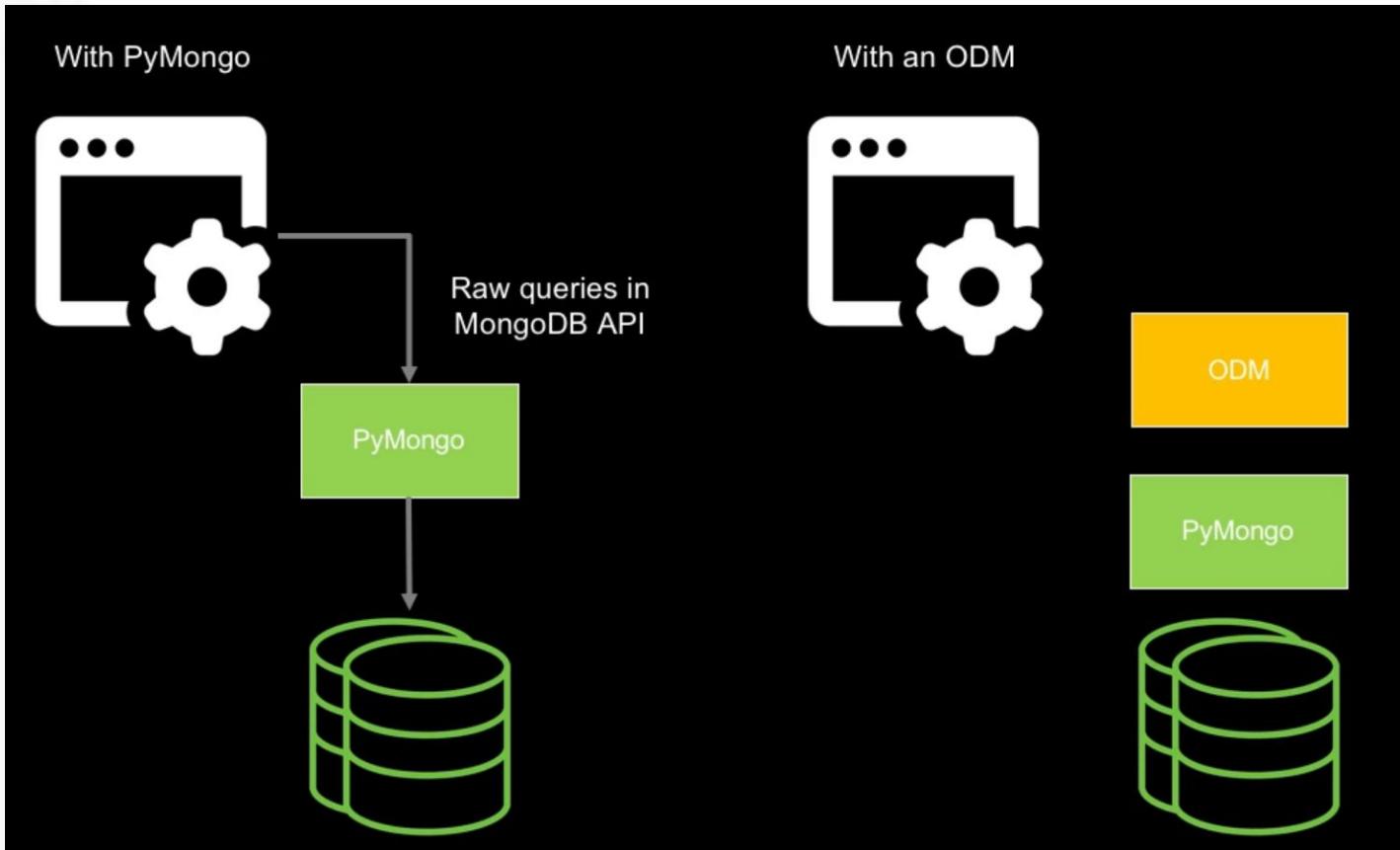
Basicamente converte dados entre tipos incompatíveis em linguagens de programação orientadas a objetos.

ORM envolve os detalhes específicos de implementação de drivers de armazenamento em uma API (interface de programa de aplicativo) e mapeia os campos relacionais para membros de um objeto.

ODM (Object Document Mapper) é um “mapeador” de documentos do MongoDB.

Mapper

IGTi



ORM, ODM

- Um melhor desempenho pode ser obtido usando SQL ou qualquer linguagem de consulta suportada pelo banco de dados.
- ORM / ODMs são frequentemente mais lentos porque usam código de tradução para mapear entre objetos e o formato de banco de dados, o que pode não usar as consultas de banco de dados mais eficientes.
- O benefício de usar um ORM / ODM é que os programadores podem continuar a pensar em termos de objetos JavaScript em vez da semântica do banco de dados - isso é particularmente verdadeiro se você precisar trabalhar com bancos de dados diferentes (no mesmo site ou em diferentes sites).
- Eles também fornecem um local centralizado para realizar a validação e verificação de dados.

Soluções ORM, ODM

Existem muitas soluções ODM / ORM disponíveis.

- **Mongoose**: ferramenta de modelagem de objetos MongoDB projetada para funcionar em um ambiente assíncrono.
- **Waterline**: é um ORM extraído da estrutura web **Sails.js** baseada no Express. Ele fornece uma API uniforme para acessar vários bancos de dados diferentes, incluindo Redis, MySQL, LDAP, MongoDB e Postgres.
- **Bookshelf.js**: Apresenta interfaces de retorno de chamada tradicionais e baseadas em promessa, fornecendo suporte a transações, carregamento de relação ansioso/aninhado, associações polimórficas e suporte para relações um-para-um, um-para-muitos e muitos-para-muitos. Funciona com PostgreSQL, MySQL e SQLite3.
- **Objection.js**: torna o mais fácil possível o uso de todo o poder do SQL e do mecanismo de banco de dados subjacente (suporta SQLite3, Postgres e MySQL).

ORM, ODM



- **Sequelize:** ORM para Node.js e io.js. Ele suporta PostgreSQL, MySQL, MariaDB, SQLite e MSSQL e oferece suporte a transações sólidas, relações, replicação de leitura entre outros
- **Node ORM2:** é um ORM para NodeJS. Ele suporta MySQL, SQLite e Progress, ajudando a trabalhar com o banco de dados usando uma abordagem orientada a objetos.
- **JugglingDB:** é um ORM de banco de dados cruzado para NodeJS, fornecendo uma interface comum para acessar os formatos de banco de dados mais populares. Atualmente com suporte a MySQL, SQLite3, Postgres, MongoDB, Redis e js-memory-storage (mecanismo de escrita própria apenas para teste).

O Mongoose é o mais popular ODM no momento e é uma escolha razoável se você estiver usando o MongoDB para seu banco de dados.

Mongoose

- O Mongoose fornece um método simples e baseado em solução de esquema para modelar seus dados no aplicativo. Inclui seleção de tipo, validação, construção de consulta, lógica de negócios.
- O Mongoose é instalado no seu projeto (`package.json`) assim como outra dependência qualquer — usando NPM. Para instalá-lo, use a seguinte linha de comando dentro da pasta do seu projeto: ***npm install mongoose***.
- A instalação do Mongoose adiciona todas as suas dependências, incluindo o driver de banco de dados MongoDB, mas não instala o MongoDB propriamente dito.

Mongoose - Schema

Além de definir a estrutura de seus documentos e os tipos de dados que você está armazenando, um Schema lida com a definição de:

- Validators (async and sync)
- Defaults
- Getters
- Setters
- Indexes
- Middleware
- Methods definition
- Statics definition
- Plugins
- Pseudo-JOINs

Mongoose - Schema



```
const Comment = new Schema({  
  name: { type: String, default: 'hahaha' },  
  age: { type: Number, min: 18, index: true },  
  bio: { type: String, match: /[a-z]/ },  
  date: { type: Date, default: Date.now },  
  buff: Buffer  
});  
  
// a setter  
Comment.path('name').set(function (v) {  
  return capitalize(v);  
});  
  
// middleware  
Comment.pre('save', function (next) {  
  notify(this.get('email'));  
  next();  
});
```

Conclusão



- ✓ ORM
- ✓ ODM

Próxima aula



01. ••

03. ••

02. ••

04. ••

Ferramentas de Gerenciamento



Banco de Dados NoSQL

Capítulo 05 – Aula 05.05 – Ferramentas de Gerenciamento

PROF.: RICARDO BRITO ALVES

Nesta aula



- ❑ Ferramentas de Gerenciamento

Ferramentas de Gerenciamento



- Existem diversas ferramentas disponíveis no mercado para administrar o banco de dados MongoDB.
- Elas melhoraram a produtividade em tarefas de desenvolvimento, administração e ajudam também a diminuir a curva de aprendizado para quem está começando com banco de dados NoSQL ou vindo do SQL.

NoSQLBooster (antigo MongoBooster)



O NoSQLBooster (anteriormente MongoBooster) é uma ferramenta popular para administrar seu banco de dados. Disponível em diversas plataformas (Mac, Linux e Windows) ela é centrada na linha de comando mongo e fornece recursos de monitoramento, construtor de consultas com autocomplete, consultas SQL e suporte à sintaxe ES2017.

Principais recursos

- Autocomplete inteligente.
- Consultas em SQL.
- Construção de consultas encadeadas.
- Análise de Schemas.
- Monitoramento de servidor.

NoSQLBooster (antigo MongoBooster)



The screenshot shows the NoSQLBooster for MongoDB interface. On the left, the Connection Tree displays a database named 'test' with a size of 1.8GB, containing collections like 'categories', 'movieDetails', 'objectType', 'orders2', 'sensor_readings', 'testCollection', 'transaction', 'unicorns', 'users', and 'roles'. A specific collection, 'unicorns', is highlighted with a yellow selection bar. Below the tree, there are 'My Queries' and 'Samples' sections, with 'Samples' currently selected. Under 'Samples', several JavaScript files are listed, including '01.intro.js', which is also highlighted with a yellow selection bar. In the center, the main workspace shows a connection to 'localhost_40017' and a database 'test'. A query editor window titled 'test:unicorns' is open, showing a list of MongoDB commands such as 'var', 'DBQuery', 'findById', 'find', 'show dbs', 'use <db>', 'findByObjectId', 'find', 'show databases', and 'findWhereObjectIdFromDate'. The 'var' command is currently selected and highlighted with a blue background.

NoSQL Manager



Essa ferramenta de administração mescla a interface amigável e o poder da linha de comando. Oferece alto desempenho com suporte para todos os recursos mais recentes do MongoDB e MongoDB Enterprise.

Principais recursos

- Interface gráfica para linha de comando do MongoDB com preenchimento automático de código.
- Três modos de exibição: Árvore, Tabela e JSON.
- Importação de tabelas de bancos de dados MySQL e SQL Server.
- Exportação de documentos para os formatos CSV, XML, XLSX e JSON.

NoSQL Manager



The screenshot shows the NoSQL Manager application interface. At the top, there is a toolbar with various icons and a menu bar with tabs for "Script 1" through "Script 6" and "Favorite Scripts". Below the toolbar is a code editor window containing a MongoDB query:

```
1 db.orders.find({  
2     ShipToCountry : "United Kingdom",  
3     ItemsTotal : {$gt:100}  
4 })  
5 .sort({ SalesDate : -1})  
6 .limit(2000)  
7  
8
```

Below the code editor is an "Output" tab, which displays the results of the query. The results are presented in a table with three columns: "Document", "Data", and "Type". The "Data" column shows the document structure with fields like "_id", "SaleDate", "ShipToCity", "ShipToCountry", "PaymentMethod", "ItemsTotal", and "agent". The "Type" column indicates the type of each field. The table shows five documents, with the first one expanded to show its full structure. The bottom of the output tab shows the message "Documents fetched: 21, skipped: 0" and "Focused document id: 5c7c3dc483545669402b7217".

At the bottom left, it says "Execution time: 0.3s".

MongoDB Compass



O MongoDB Compass é desenvolvido pela própria equipe do MongoDB. Ele fornece aos usuários uma visualização gráfica de seu esquema MongoDB sem a necessidade da linguagem de consulta por linha de comando.

Principais recursos

- O MongoDB Compass analisa documentos e exibe estruturas chaves em uma coleção usando consultas ad-hoc em segundos.
- Oferece suporte a informações rápidas sobre o status do servidor e o desempenho da consulta.
- Permite visualizar o desempenho de consultas.
- Ajuda os usuários a tomar decisões sobre indexação, validação de documentos e muito mais.
- Não há necessidade de escrever na linha de comando.

MongoDB Compass

IGTI

localhost:27017
Community version 3.1.8

8 DBs | 15 Collections | C

Q filter

flightStats-cut

mongodb://localhost:27017/test.flightStats-cut (localhost:27017)

DOCUMENTS 10.0k total size 6.5 MB avg. size 684 B | INDEXES 1 total size 566.9 KB avg. size 566.9 KB

{ } APPLY RESET

Query returned 9,993 documents. This report is based on a sample of 100 documents (1.00%). ⓘ

flightStats-cut

_id string

EWR-EV-4467-542273341 EWR-EV-4382-544626614
LGA-9E-3457-542758157 JFK-9E-4093-544640472
LGA-ZW-3910-545111616 JFK-SE-41-544640167
EWR-YX-4904-544626872 LGA-ZW-3815-544184788

arrivalAirportF String (100%)

String

0% 3.5% 7%

carrierFsCode

String

19% 9.5% 0%

codeshares

Array of documents with 3 nested fields
Array lengths
min: 1, average: 2.73, max: 11

array undefined document

This screenshot shows the MongoDB Compass interface for the 'flightStats-cut' database. The left sidebar lists databases and collections, with 'flightStats-cut' selected. The main area displays statistics for the collection: 10.0k documents, 6.5 MB total size, and 1 index. A sample report for 100 documents is shown, with 9,993 actual documents. Below, sample data for the '_id' field is listed, and histograms show the distribution of 'arrivalAirportF' and 'carrierFsCode'. The 'codeshares' field is shown as an array of documents with nested fields.

Robo 3T (antigo Robomongo)



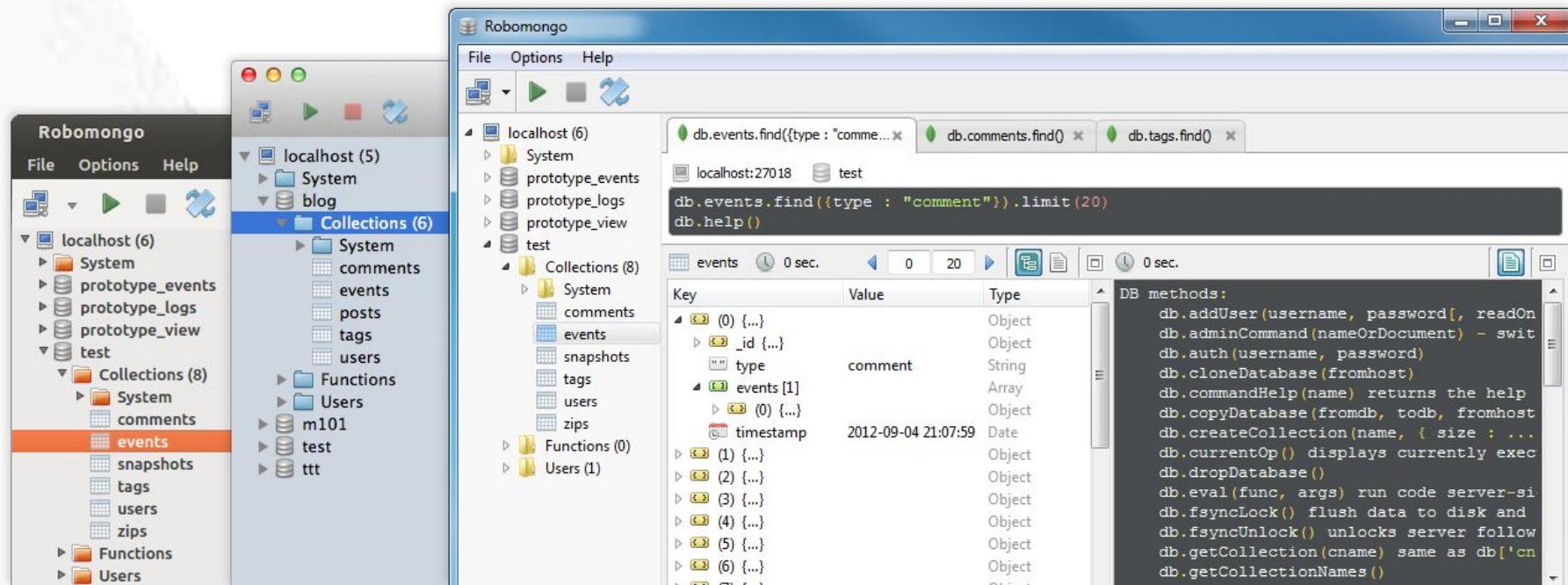
Robo 3T (anteriormente Robomongo) é uma das mais populares interfaces gráficas e gratuita para os amantes do MongoDB. Leve e de código aberto, oferece suporte a várias plataformas (Mac, Linux e Windows) e incorpora também a linha de comando mongo em sua aplicação para fornecer interação para os usuários mais avançados. É desenvolvido pela 3T Software, a equipe por trás da IDE Studio 3T .

Principais recursos

- Linha de comando incorporada.
- Interface assíncrona e sem bloqueios.
- Suporte para MongoDB 4.0+.

Robo 3T

igti



Conclusão



Ferramentas de Gerenciamento

- ✓ NoSQL Booster
- ✓ NoSQL Manager
- ✓ MongoDB Compass
- ✓ Robo 3T

Próxima aula



01. ••

Banco de Dados NoSQL em
Nuvem

02. ••

Sistemas de Arquivos

03. ••

04. ••

Bancos de Dados NoSQL

Capítulo 06 – Banco de Dados NoSQL em Nuvem e Sistemas de Arquivos

PROF.: RICARDO BRITO ALVES



Banco de Dados NoSQL

Capítulo 06 – Aula 06.01 – Introdução a Banco de Dados em Nuvem

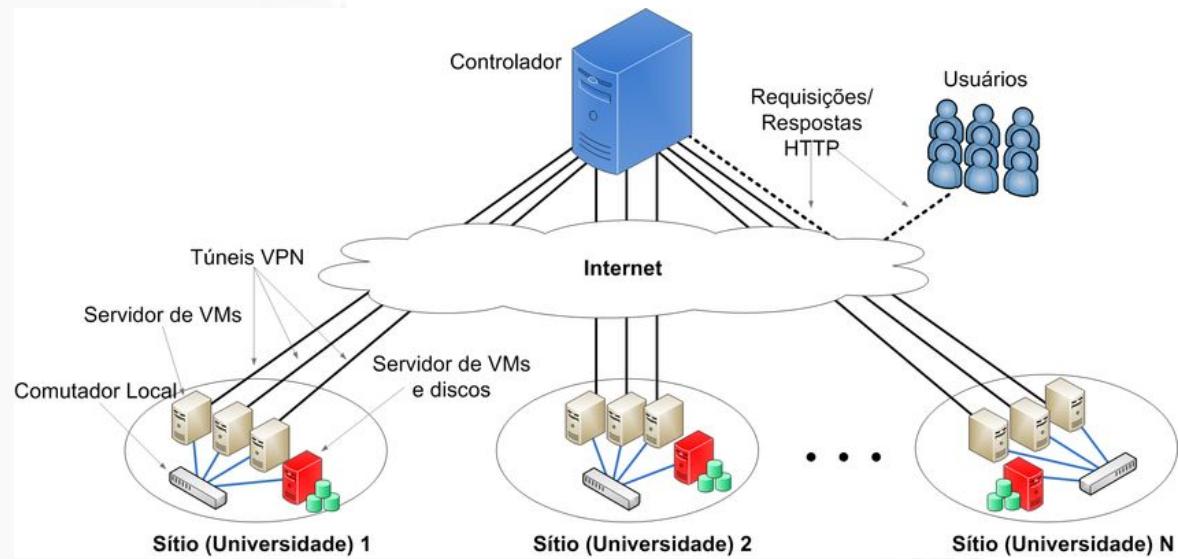
PROF.: RICARDO BRITO ALVES

- Modelos em Nuvem
- DBaaS
- Tipos de Nuvem
- Principais Fornecedores

Computação em Nuvem

IGTI

- Provisionamento dinâmico de recursos sob demanda
- Escalabilidade
- Cobrança baseada no uso do recurso ao invés de uma taxa fixa
- Distribuição geográfica dos recursos



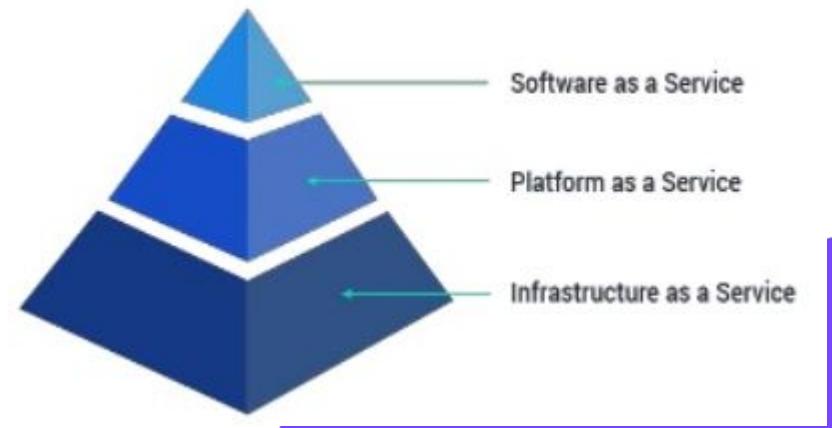
Modelos em Nuvem



SaaS é caracterizado principalmente pela não-aquisição de licenças de software.

PaaS envolve um ambiente virtual para criação, hospedagem e controle de softwares e bancos de dados.

IaaS apenas abstraí aspectos relacionados à parte física de servidores e redes.



IaaS — Infrastructure as a Service (Infraestrutura como Serviço)



Nesse exemplo dos modelos de nuvem, **a empresa contrata uma capacidade de hardware que corresponde a memória, armazenamento, processamento, etc.** Podem entrar nesse pacote de contratações os servidores, roteadores, racks, entre outros.

Dependendo do fornecedor e do modelo escolhido, a sua empresa pode ser tarifada, por exemplo, pelo número de servidores utilizados e pela quantidade de dados armazenados ou trafegados. **Em geral, tudo é fornecido por meio de um data center com servidores virtuais, em que você paga somente por aquilo que usar.**

PaaS — Platform as a Service (Plataforma como Serviço)



Nesse cenário, o PaaS surge como o ideal porque é, como o próprio nome diz, *uma plataforma que pode criar, hospedar e gerir esse aplicativo*.

Nesse modelo de nuvem, contrata-se um ambiente completo de desenvolvimento, no qual é possível criar, modificar e otimizar softwares e aplicações.

Aqui, a grande vantagem é que *a equipe de desenvolvimento só precisa se preocupar com a programação do software, pois o gerenciamento, manutenção e atualização da infraestrutura ficam a cargo do fornecedor e as várias ferramentas de desenvolvimento de software são oferecidas na plataforma*.

SaaS — Software as a Service (Software como Serviço)



Nesse modelo de nuvem **você pode ter acesso a um software sem comprar a sua licença, utilizando-o a partir da Cloud Computing**.

Muitos CRMs ou ERPs trabalham no sistema SaaS.

Assim, o acesso a esses softwares é feito usando a internet. Os dados, contatos e demais informações podem ser acessados de qualquer dispositivo, dando mais mobilidade à equipe.

O Google Docs e o Office 365 funcionam dessa maneira.

On-Premises

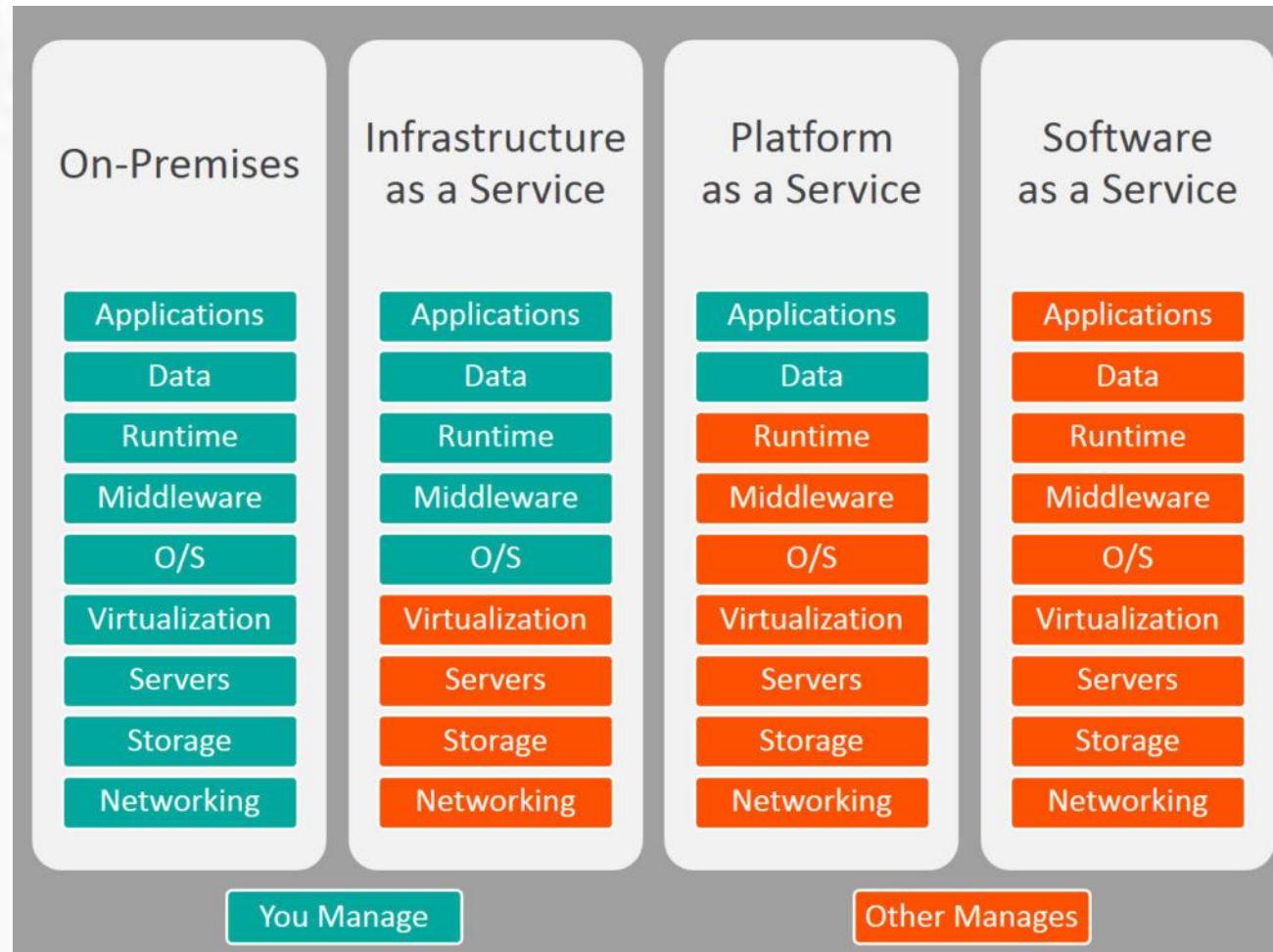


Um servidor ***on-premises*** é aquele em que a própria empresa tem a responsabilidade de processar suas aplicações de hardware e software.

Em outras palavras, toda a infraestrutura, customização, configuração e atualização é feita internamente.

Computação em Nuvem

IGTI



Banco de Dados em Nuvem



Vantagens

- Menor TCO (Total Cost of Ownership, significa Custo Total de Propriedades).
- Facilidade de manutenção.
- Maior elasticidade.
- Menor investimento inicial.

Desvantagens

- Maior percepção de custo.
- Segurança e privacidade (dados enviados para um provedor externo).

BDaaS - Banco de Dados Como Serviço



Virtualização: permite que banco de dados seja instalado em uma máquina virtual

DBaaS: fornece uma plataforma flexível escalável, sob demanda que está orientada para o autoserviço e gerenciamento fácil, particularmente em termos de provisionamento de um negócio no próprio ambiente

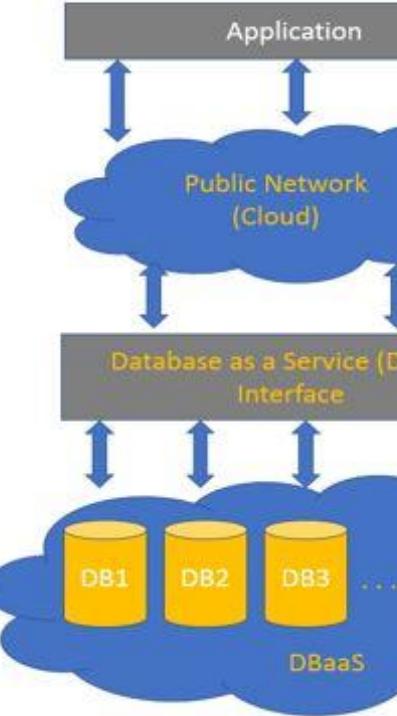


Banco de Dados Como Serviço

DBaaS oferece a funcionalidade de um banco de dados **semelhante ao que é encontrado em sistemas de gerenciamento de banco de dados relacionais (RDBMS), como o SQL Server, MySQL e Oracle.**

Sendo baseado em nuvem, por outro lado o **DBaaS fornece uma plataforma flexível escalável**, sob demanda que está orientada para o autoserviço e gerenciamento fácil, particularmente em termos de provisionamento de um negócio no próprio ambiente.

Produtos DBaaS normalmente fornecem capacidades de monitoramento suficientes para acompanhar o desempenho e o consumo e para alertar os usuários sobre possíveis problemas.



Cloudbase Database Arch

Banco de Dados Como Serviço



As *desvantagens* para o modelo DBaaS *incluem a falta de controle sobre os problemas de desempenho de rede, tais como falhas de latência e de aplicação inaceitáveis.*

Além disso, *alguns produtos DBaaS não suportam capacidades dos RDBMS típicos, tais como compressão de dados e partições de tabela.*

Antes de contratar um DBaaS, é essencial que a empresa avalie suas necessidades específicas e garanta que elas sejam satisfatoriamente resolvidas.

Principais Fornecedores

IGTI



Computação em Nuvem



Como se diferenciam:

	Nuvem pública	Nuvem privada	Servidores dedicados	Nuvem híbrida
Descrição	Ambiente multi-inquilino com escalabilidade de pagamento pelo uso.	Escalabilidade, mais a segurança e o controle aprimorados de um ambiente de inquilino único	Para cargas de trabalho previsíveis que exigem segurança e controle aprimorados	Conecte a nuvem pública à sua nuvem privada ou a servidores dedicados — até mesmo em seu próprio data center
Hardware físico	Compartilhado	Dedicada	Dedicada	Compartilhado + dedicado
Melhor para	Operações não sigilosas, voltadas para o público, e tráfego imprevisível	Operações sigilosas críticas	Operações sigilosas críticas e requisitos exigentes de desempenho, segurança e conformidade	Combine nuvem privada, pública e/ou servidores dedicados para o melhor de cada um deles

Computação em Nuvem

IGTI

Como se diferenciam (Rackspace)

	Nuvem pública	Nuvem privada	Servidores dedicados	Nuvem híbrida
Descrição	Ambiente multi-inquilino com escalabilidade de pagamento pelo uso.	Escalabilidade, mais a segurança e o controle aprimorados de um ambiente de inquilino único	Para cargas de trabalho previsíveis que exigem segurança e controle aprimorados	Conecte a nuvem pública à sua nuvem privada ou a servidores dedicados — até mesmo em seu próprio data center
Expansível	✓	✓	✗	✓
Custo baixo, cobrança como serviços públicos	✓	✗	✗	✓
Flexibilidade	✓	✗	✗	✓
Personalizável	✗	✓	✓	✓
Alto desempenho	✗	✓	✓	✓
Segurança e controle aprimorados	✗	✓	✓	✓
Custo previsível	✗	✓	✓	✓

Computação em Nuvem

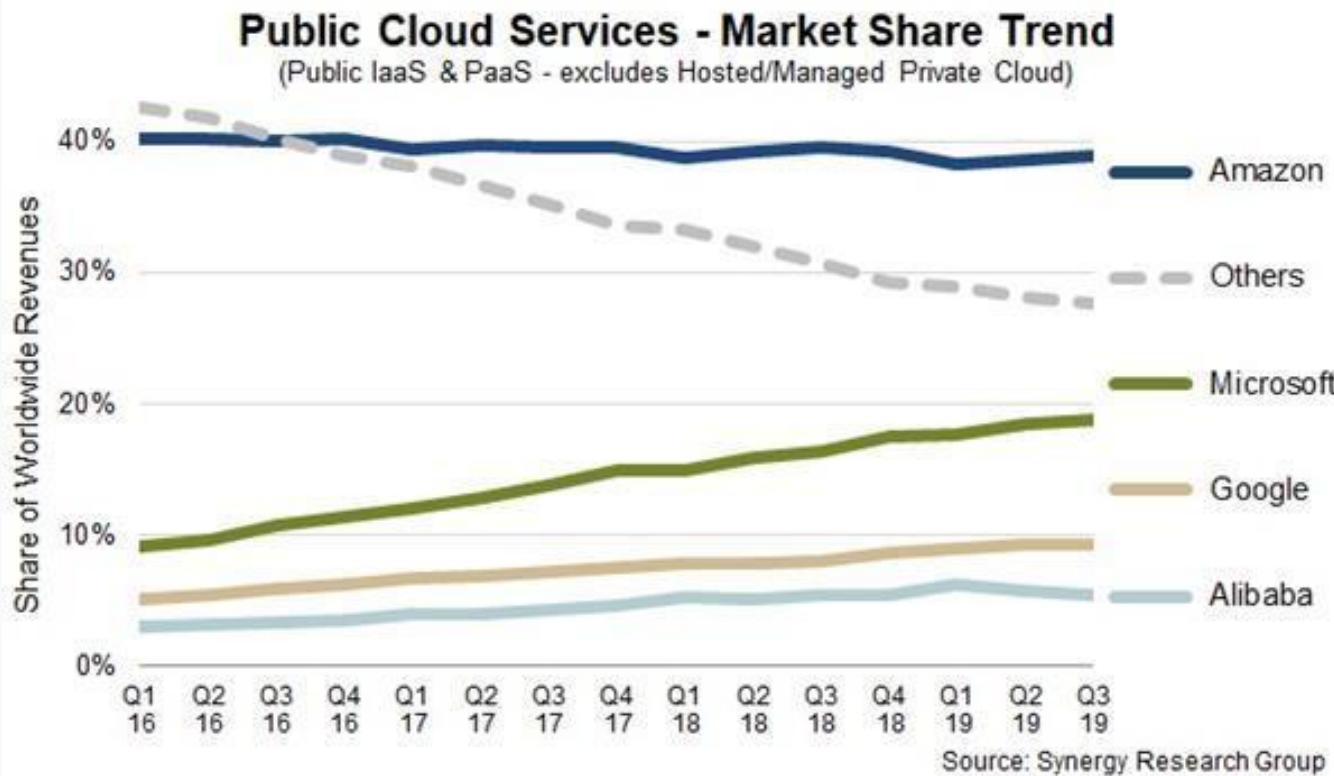


Nós temos três principais fornecedores de computação em nuvem, **AWS**, **Microsoft Azure** e **Google Cloud**, que possuem pontos fortes e fracos que os tornam ideais para diferentes cenários.



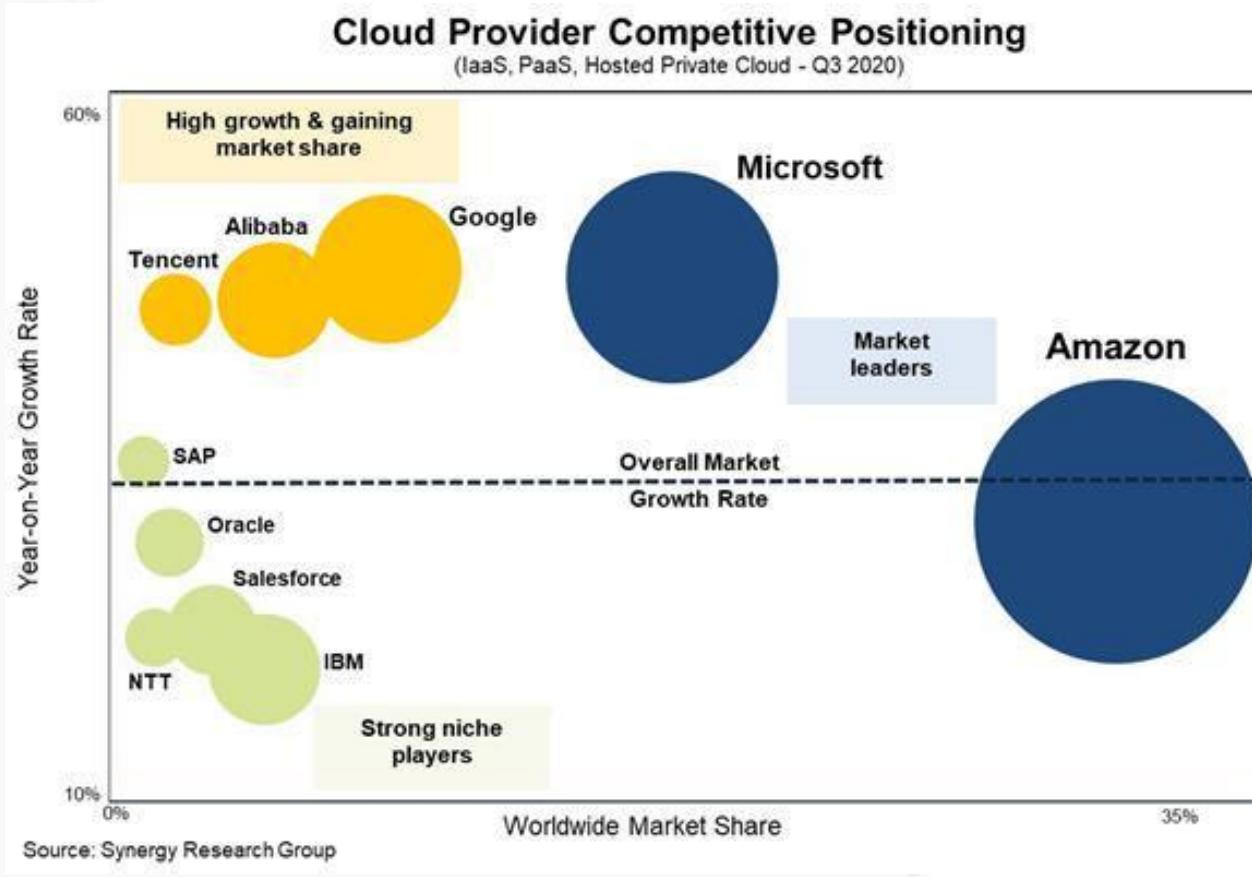
Synergy Research Group

IGTI



Synergy Research Group

IGTI



Synergy Research Group



APAC – Ásia Pacífico

Cloud Services Leadership – APAC Region

Rank	Total Region	China	Japan	Rest of East Asia	South & Southeast	Oceania
Leader	Amazon	Alibaba	Amazon	Amazon	Amazon	Amazon
#2	Alibaba	Tencent	Fujitsu	Microsoft	Microsoft	Microsoft
#3	Microsoft	Baidu	Microsoft	Alibaba	Google	Telstra
#4	Tencent	China Telecom	NTT	Google	Alibaba	Google
#5	Google	Sinnet-AWS	Google	Naver	NTT	Alibaba
#6	NTT	China Unicom	Softbank	KT	IBM	IBM

Based on IaaS, PaaS and hosted private cloud revenues in Q2 2020

Source: Synergy Research Group

Conclusão



Vimos os principais conceitos de nuvem.

- ✓ Modelos em Nuvem
- ✓ DBaaS
- ✓ Tipos de Nuvem
- ✓ Principais Fornecedores

Próxima Aula



01. ••

Introdução a AWS

02. ••

03. ••

04. ••



- AWS
- Principais Serviços

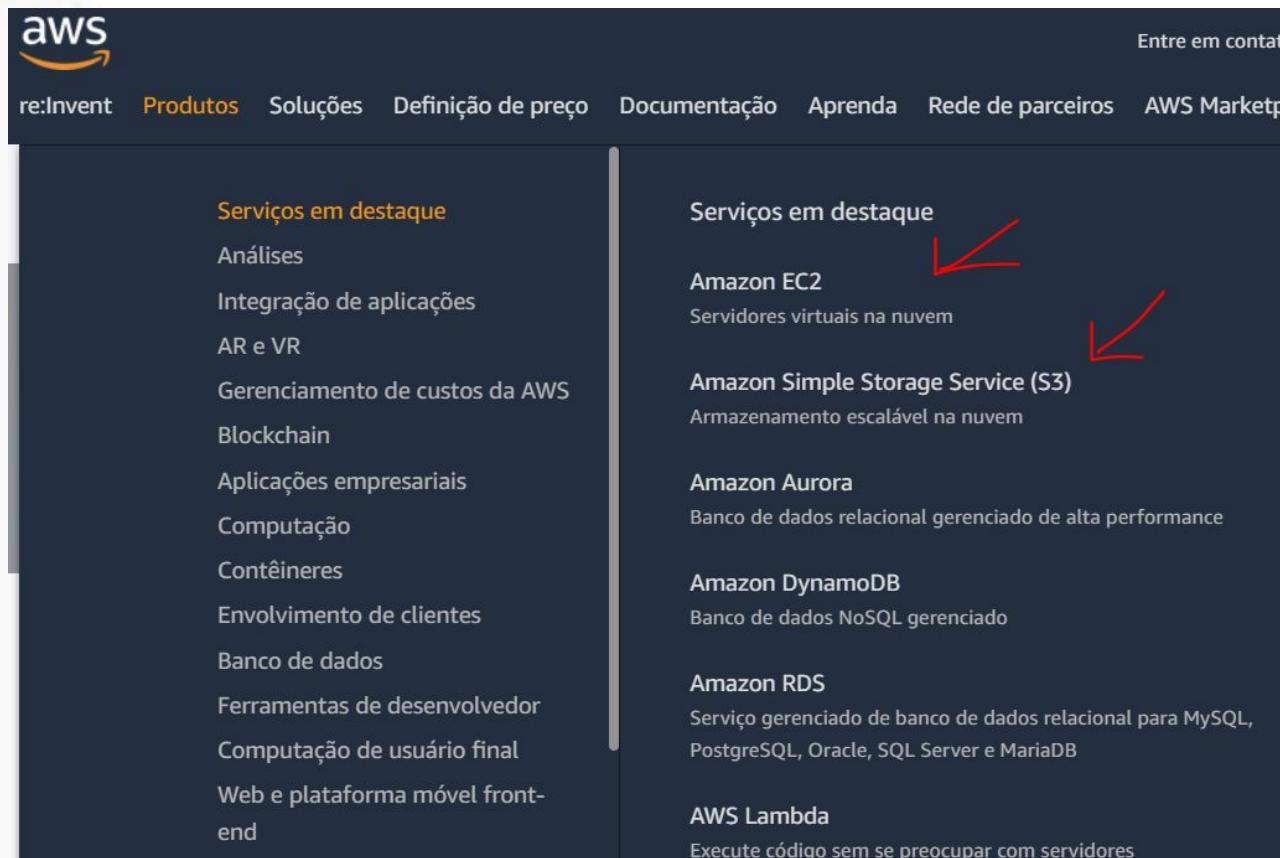


Banco de Dados NoSQL

Capítulo 06 – Aula 06.02 – Introdução a AWS

PROF.: RICARDO BRITO ALVES

<https://aws.amazon.com/pt/getting-started/>



The screenshot shows the AWS Getting Started page. On the left, there's a sidebar with a list of services. On the right, there's a main content area with a list of featured services. Two red arrows point from the sidebar to the main content area, highlighting specific services.

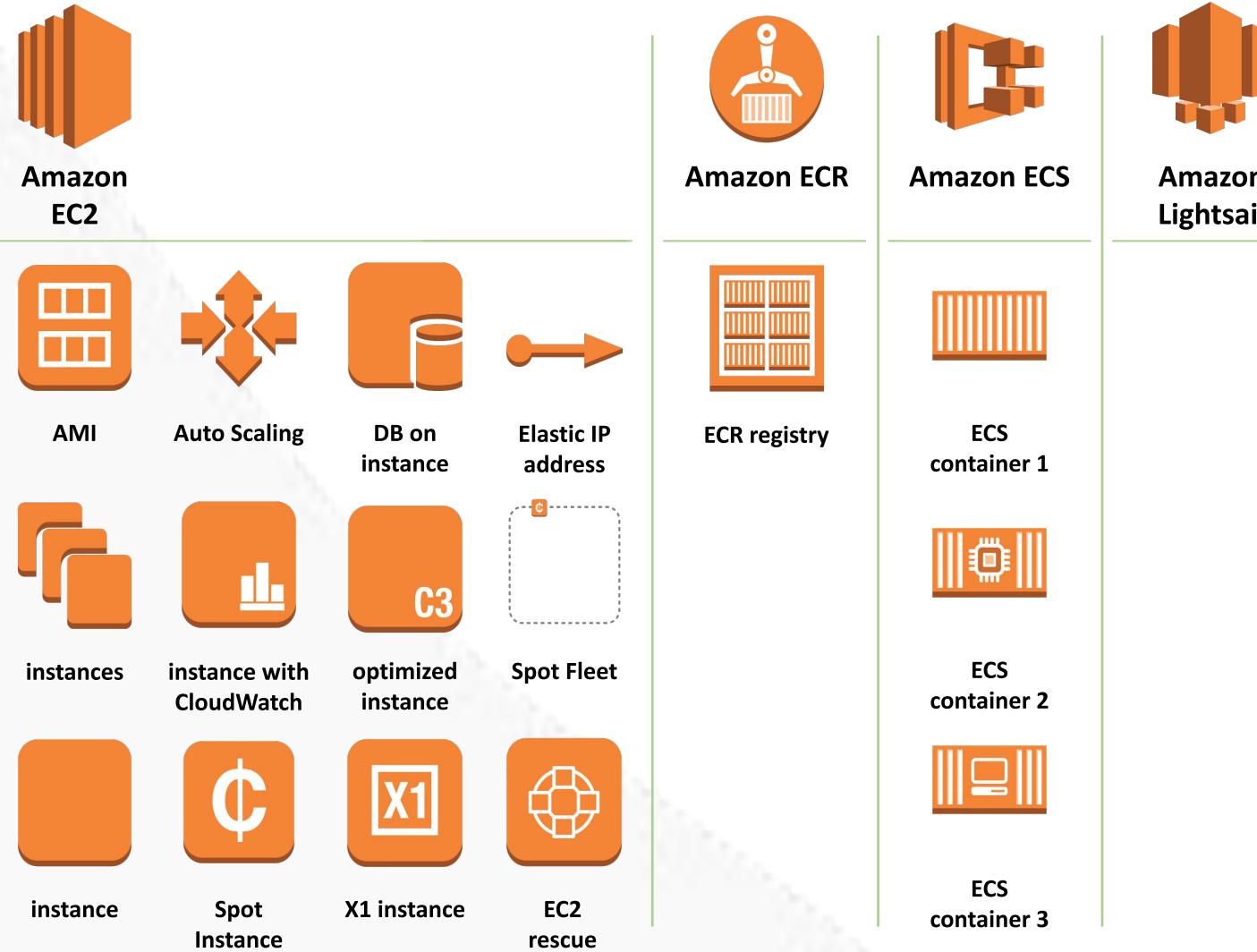
Serviços em destaque

- Análises
- Integração de aplicações
- AR e VR
- Gerenciamento de custos da AWS
- Blockchain
- Aplicações empresariais
- Computação
- Contêineres
- Envolvimento de clientes
- Banco de dados
- Ferramentas de desenvolvedor
- Computação de usuário final
- Web e plataforma móvel front-end

Serviços em destaque

- Amazon EC2**
Servidores virtuais na nuvem
- Amazon Simple Storage Service (S3)**
Armazenamento escalável na nuvem
- Amazon Aurora**
Banco de dados relacional gerenciado de alta performance
- Amazon DynamoDB**
Banco de dados NoSQL gerenciado
- Amazon RDS**
Serviço gerenciado de banco de dados relacional para MySQL, PostgreSQL, Oracle, SQL Server e MariaDB
- AWS Lambda**
Execute código sem se preocupar com servidores

Computacional



AWS – EC2



O Amazon *Elastic Compute Cloud* (Amazon EC2) oferece uma capacidade de computação escalável na Nuvem da Amazon Web Services (AWS).

O uso do Amazon EC2 ***elimina a necessidade de investir em hardware inicialmente***, portanto, você pode desenvolver e implantar aplicativos com mais rapidez.

Você ***pode usar o Amazon EC2 para executar o número de servidores virtuais que precisar***, configurar a segurança e a rede, e gerenciar o armazenamento.

O Amazon EC2 também permite a expansão ou a redução para gerenciar as alterações de requisitos ou picos de popularidade, reduzindo, assim, a sua necessidade de prever o tráfego do servidor.

AWS – EC2



<https://aws.amazon.com/pt/ec2/instance-types/>

AWS – EC2



<https://aws.amazon.com/pt/ec2/instance-types/>

A1	T4g	T3	T3a	T2	M6g	M5	M5a	M5n	M4	
----	-----	----	-----	----	-----	----	-----	-----	----	--

As instâncias M4 oferecem recursos equilibrados de computação, memória e rede e são uma boa opção para muitos aplicativos.

Recursos:

- Processadores Intel Xeon® E5-2686 v4 (Broadwell) de 2,3 GHz ou processadores Intel Xeon® E5-2676 v3 (Haswell) de 2,4 GHz
- Otimizado para EBS por padrão, sem custo adicional
- Suporte a redes avançadas
- Equilíbrio entre recursos de computação, memória e rede

Instância	vCPU*	Mem (GiB)	Armazenamento	Largura de banda dedicada do EBS (Mbps)	Performance de rede
m4.large	2	8	Somente EBS	450	Moderada
m4.xlarge	4	16	Somente EBS	750	Alta
m4.2xlarge	8	32	Somente EBS	1.000	Alta
m4.4xlarge	16	64	Somente EBS	2.000	Alta
m4.10xlarge	40	160	Somente EBS	4.000	10 Gigabit
m4.16xlarge	64	256	Somente EBS	10.000	25 Gigabit

AWS – EC2

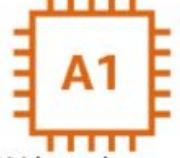
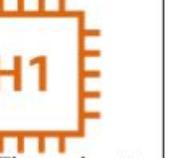
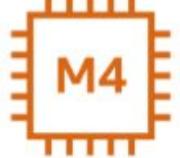
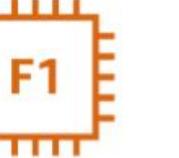


<https://aws.amazon.com/pt/ec2/instance-types/>

Instance	EC2 Compute Units	Memory (GiB)	Storage (GB)	Price per Hour
t2.small	variable	2.0	EBS	\$0.023
m5a.large	4	8.0	EBS	\$0.086
m5a.xlarge	8	16.0	EBS	\$0.172
c5.large	9	4.0	EBS	\$0.085
c5.xlarge	17	8.0	EBS	\$0.170

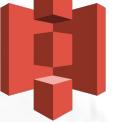
EC2 - Instâncias

IGTI

General Purpose	Compute Optimised	Memory Optimised	Accelerated Computing	Storage Optimised
 ARM based core and custom silicon	 Compute - CPU intensive apps and DBs	 RAM - Memory intensive apps and DB's	 Processing optimised-Machine Learning	 High Disk Throughput - Big data clusters
 Tiny - Web servers and small DBs		 Xtreme RAM - For SAP/Spark	 Graphics Intensive - Video and streaming	 IOPS - NoSQL DBs
 Main - App servers and general purpose		 High Compute and High Memory - Gaming	 Field Programmable - Hardware acceleration	 Dense Storage - Data Warehousing

Storage

IGTI

Amazon S3	Amazon EFS	Amazon Glacier	AWS Storage Gateway	AWS Snowball*	Amazon EBS
					
bucket	file system	archive	cached volume	import/export	snapshot
					
bucket with objects		vault	non-cached volume	virtual tape library	volume
					
object					

AWS – S3



O Amazon ***Simple Storage Service*** é armazenamento para a Internet. Ele foi projetado para facilitar a computação de escala na web para os desenvolvedores.

O Amazon S3 tem uma interface simples de serviços da web que você pode usar para armazenar e recuperar qualquer quantidade de dados, a qualquer momento, em qualquer lugar da web.

Ela concede acesso a todos os desenvolvedores para a mesma infraestrutura altamente dimensionável, confiável, segura, rápida e econômica que a Amazon utiliza para rodar a sua própria rede global de sites da web.

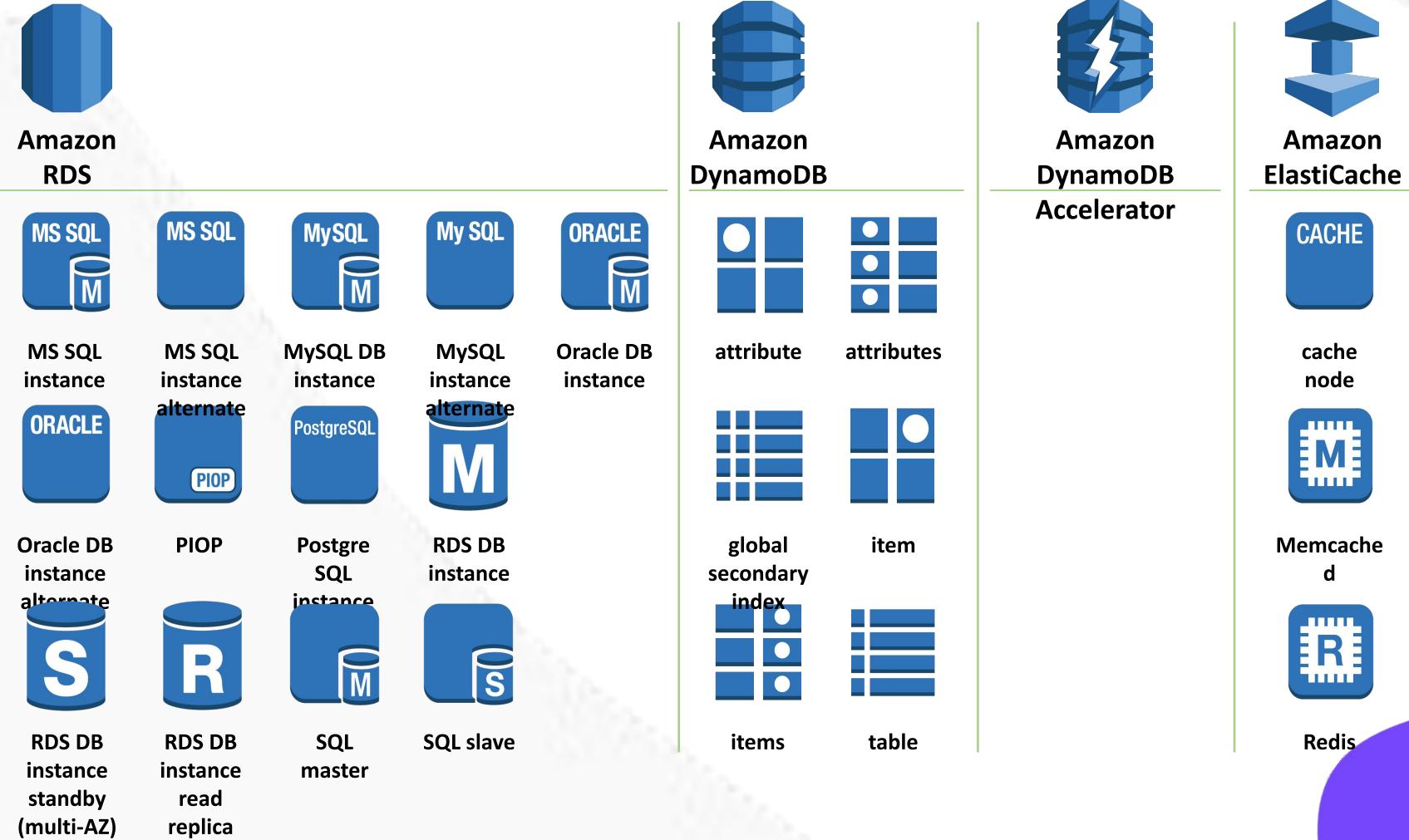
O serviço visa maximizar os benefícios de escala e poder passar esses benefícios para os desenvolvedores.

AWS – S3



	S3 Standard	S3 Intelligent-Tiering*	S3 Standard-IA	S3 One Zone-IA†	S3 Glacier	S3 Glacier Deep Archive**
Projetado para resiliência	99,999999999% (11 9s)	99,999999999% (11 9's)				
Projetado para disponibilidade	99,99%	99,9%	99,9%	99,5%	99,99%	99,99%
Acordo de nível de serviço de disponibilidade	99,9%	99%	99%	99%	99,9%	99,9%
Zonas de disponibilidade	≥3	≥3	≥3	1	≥3	≥3
Cobrança mínima de capacidade por objeto	N/D	N/D	128 KB	128 KB	40 KB	40 KB
Cobrança mínima de duração de armazenamento	N/D	30 dias	30 dias	30 dias	90 dias	180 dias
Taxa de recuperação	N/D	N/D	por GB recuperado	por GB recuperado	por GB recuperado	por GB recuperado
Latência de primeiro byte	milissegundos	milissegundos	milissegundos	milissegundos	selecione minutos ou horas	selecione horas
Tipo de armazenamento	Objeto	Objeto	Objeto	Objeto	Objeto	Objeto
Transições de ciclo de vida	Sim	Sim	Sim	Sim	Sim	Sim

Database



AWS – RDS



O Amazon *Relational Database Service* (Amazon RDS) *facilita a configuração, a operação e a escalabilidade de bancos de dados relacionais na nuvem.*

O serviço *oferece capacidade econômica e redimensionável e automatiza tarefas demoradas de administração, como provisionamento de hardware, configuração de bancos de dados, aplicação de patches e backups.*

Dessa forma, você pode se concentrar na performance rápida, alta disponibilidade, segurança e conformidade que os aplicativos precisam.

AWS – RDS



Mecanismos de banco de dados do Amazon RDS



AWS

IGTI

COMPUTE	
Free Tier	12 MONTHS FREE
Amazon EC2 750 Hours per month	
Resizable compute capacity in the Cloud.	

STORAGE	
Free Tier	12 MONTHS FREE
Amazon S3 5 GB of standard storage	
Secure, durable, and scalable object storage infrastructure.	

DATABASE	
Free Tier	12 MONTHS FREE
Amazon RDS 750 Hours per month of db.t2.micro database usage (applicable DB engines)	
Managed Relational Database Service for MySQL, PostgreSQL, MariaDB, Oracle BYOL, or SQL Server.	

DATABASE	
Free Tier	ALWAYS FREE
Amazon DynamoDB 25 GB of storage	
Fast and flexible NoSQL database with seamless scalability.	

MACHINE LEARNING	
Free Tier	FREE TRIAL
Amazon SageMaker 250 Hours per month of t2.medium notebook usage for the first two months	
Fully managed platform to build, train, and deploy machine learning models.	

COMPUTE	
Free Tier	ALWAYS FREE
AWS Lambda 1 Million free requests per month	
Compute service that runs your code in response to events and automatically manages the compute resources.	

Conclusão



- ✓ AWS
- ✓ Principais Serviços

Próxima Aula



01. •

Introdução ao Google Cloud

02. •

03. •

04. •



Banco de Dados NoSQL

Capítulo 06 – Aula 06.03 - Introdução ao Google Cloud

PROF.: RICARDO BRITO ALVES



- Google Cloud
- Principais Serviços

Google Cloud Platform



Google Cloud *Platform* é uma suíte de computação em nuvem oferecida pelo Google, funcionando na mesma infraestrutura que a empresa usa para seus produtos dirigidos aos usuários, dentre eles o *Buscador Google* e o *Youtube*.

Juntamente com um conjunto de ferramentas de gerenciamento modulares, fornecem uma série de serviços *incluindo, computação, armazenamento de dados, análise de dados e aprendizagem de máquina.*

Compute Engine



Criação de máquinas virtuais

O Google Compute Engine é o componente ***Infraestrutura como serviço do Google Cloud Platform***, construído sobre a infraestrutura global que executa o mecanismo de pesquisa do Google, Gmail, YouTube e outros serviços.

O Google Compute Engine ***permite que os usuários iniciem máquinas virtuais sob demanda***. Uma máquina virtual é um computador emulado em outra máquina. Nesse caso, o servidor em nuvem executa vários sistemas operacionais em diversas outras máquinas, em vez de cada computador ter um sistema operacional próprio.

Compute Engine



O Google Compute Engine *oferece máquinas virtuais em execução nos centros de dados inovadores do Google e na rede mundial de fibra*. O suporte a ferramentas e fluxo de trabalho do Compute Engine permite dimensionar de instâncias únicas para computação em nuvem balanceada de carga global. *As VMs do Compute Engine são inicializadas rapidamente, vêm com opções de disco persistentes e locais de alto desempenho, oferecem desempenho consistente.*

- Instâncias.
- Configurações.
- Deploy de aplicação e Jobs.
- Alertas automáticos.

Storage & Databases



Amplo armazenamento de dados em nuvem.

A ideia de Plataforma como um Serviço (PaaS) já é bem difundida em muitos setores, especialmente no que diz respeito à gestão e ao armazenamento de dados.

É similar ao Google Drive, porém, em escala bem maior, o que permite a você armazenar e organizar todos os arquivos da empresa e acessá-los a partir de qualquer máquina com permissão de acesso. É ótimo para evitar a perda de documentos e para minimizar o uso do espaço físico em seu negócio.

Storage & Databases



Armazenamento de produtos escaláveis, resilientes e de alto desempenho. Bancos de dados para suas aplicações.

- Armazenamento de arquivos.
- Gerenciamento de buckets.
- Bucket Locations.
- Multi Regional.
- Regional.
- NearLine.
- CodeLine.
- Linha de comando, API, Console.

App Engine



Plataforma para criação de aplicativos Web escaláveis e backends para dispositivos móveis. O App Engine fornece serviços integrados e APIs, como datastores NoSQL, memcache e uma API de autenticação de usuário, comum à maioria dos aplicativos.

- Construir aplicações de desenvolvimentos escaláveis.
- Instâncias automáticas.
- Integrações.
- Deploy automatizado.

Big Data



Quando precisamos tomar uma decisão importante para a empresa, é importante ter dados que deem suporte a essa escolha. Ainda mais, atualmente, em um mercado cada vez mais complexo e volátil.

Totalmente gerenciado data warehousing, lote e processamento de fluxo, exploração de dados, Hadoop / Spark, e mensagens confiáveis.

- Análise de dados em bancos NoSQL.
- Construir bancos via: linha de comando, console e API's.
- Importar dados: csv, txt, integrações.

Machine Learning



Serviços de ML rápidos, escaláveis e fáceis de usar. Use modelos pré-treinados ou treine modelos personalizados em seus dados.

- Máquina de conhecimento, como funciona.
- Exemplo Case Google: E-mail e Spam.
- Google Cloud Vision API.
- Google Translate API.
- Google Speech API.

Google Cloud Platform



Google Cloud Pricing Calculator

Prices are up to date. Last update: 16-November-2020

COMPUTE ENGINE APP ENGINE KUBERNETES ENGINE CLOUD RUN VMWARE ENGINE CLOUD STORAGE NETWORKING EGRESS CLOUD LOAD BALANCING INTE & CL

Estimate

Search for a product you are interested in.

Instances

Number of instances *

What are these instances for?

Operating System / Software

Free: Debian, CentOS, CoreOS, Ubuntu, or other User Provided OS

Machine Class

Regular

Machine Family

General purpose

Google Cloud vs AWS



O Google Cloud e o AWS são bem semelhantes. Para fazermos uma comparação seria necessário observar as categorias abaixo:

- Compute.
- Armazenamento em Blocos.
- Rede.
- Faturamento e Preços.
- Suporte e tempo de atividade.
- Segurança.

Google Cloud vs AWS



Tipo de Máquina/Instância	Google Compute Engine	AWS EC2
Compartilhado	f1-micro g1-small	t2.nano – t2.2xlarge
Padrão	n1-standard-1 – n1-standard-96 (beta)	m3.medium – m3.2xlarge m4.large – m4.16xlarge
Alta memória	n1-higmem-2 – n1-higmem-96 (beta)	r3.large – r3.8xlarge r4.large – r4.16xlarge x1.16xlarge – x1e.32xlarge
Alto CPU	n1-highcpu-2 – n1-highcpu-96 (beta)	c3.large – c3.8xlarge c4.large – c4.8xlarge
GPU	You can add GPUs to machine types	g2.2xlarge g2.8xlarge
Armazenamento SSD	n1-standard-1 – n1-standard-32 n1-higmem-2 – n1-higmem-32 n1-highcpu-2 – n1-highcpu-32	i2.xlarge – i2.8xlarge
Armazenamento denso	N/A	d2.xlarge – d2.8xlarge

Google Cloud vs AWS

IGTI

Armazenamento em bloco	Google Cloud Platform	AWS
Serviço	SSD	SSD IOPS geral e provisionado
Tamanhos	1 GB até 64 TB	1 GB até 16 TB IOPS provisionados de 4 GB a 16 TB
IOPS máximos por volume	40,000 ler, 30,000 escrever	10.000 (20.000 para IOPS provisionados) IOPS máximo de 75.000 / instância
Rendimento Máximo por Volume (MB/s)	800 ler, 400 escrever	160 (320 para IOPS provisionados)
Replicação	Redundância incorporada	RAID-1
Redundância de snapshots	Múltiplas localizações	Múltiplas localizações
Encriptação	SSE 256-bit AES	SSE 256-bit AES
Encriptação	SSE 256-bit AES	SSE 256-bit AES
Preço Magnético (por GB / mês)	\$0.040 (disco padrão)	\$0.045
Preço de SSD (por GB/mês)	\$0.170	\$0.10
Preços de SSD PIOPS (por GB/mês)	N/A	\$0.125

Conclusão



- ✓ Google Cloud
- ✓ Principais Serviços

Próxima Aula



01. •

Introdução a Microsoft Azure

02. •

03. •

04. •



Banco de Dados NoSQL

Capítulo 06 – Aula 06.04 - Introdução a Microsoft Azure

PROF.: RICARDO BRITO ALVES



- Microsoft Azure
- Principais Serviços

Microsoft Azure



O Microsoft Azure é uma plataforma destinada à execução de aplicativos e serviços, baseada nos conceitos da computação em nuvem.

A apresentação do serviço foi feita no dia 27 de outubro de 2008 durante a Professional Developers Conference, em Los Angeles e lançado em 1 de Fevereiro de 2010 como Windows Azure, para então ser renomeado como Microsoft Azure em 25 de Março de 2014.

Funcionamento

Sua computação em nuvem é definida como uma combinação de software como serviço (SaaS) com computação em grid.

A computação em grid dá o poder de computação e alta escalabilidade oferecida para as aplicações, através de milhares de máquinas (hardware) disponíveis em centros de processamento de dados de última geração. De software como serviço se tem a capacidade de contratar um serviço e pagar somente pelo uso, permitindo a redução de custos operacionais, com uma configuração de infraestrutura realmente mais aderente às necessidades.

Recursos

Além dos recursos de computação, armazenamento e administração oferecidos pelo Microsoft Azure, a plataforma também disponibiliza uma série de serviços para a construção de aplicações distribuídas, além da total integração com a solução on-premise (local) baseada em plataforma .NET.

Entre os principais serviços da plataforma Windows Azure há o SQL Azure Database, Azure AppFabric Platform e uma API de gerenciamento e monitoramento para aplicações colocadas na nuvem.

Microsoft Azure x AWS



Alguns pontos em comum entre a AWS e Azure:

- Autonomia e provisionamento
- Escalabilidade instantânea
- Segurança
- Conformidade
- Gerenciamento de identidade

As formas de cobrança da AWS podem ser:

On-demand: O cálculo é feito em cima de horas ou segundos utilizados (no mínimo 60 segundos) e somente as instâncias EC2 que forem utilizadas;

Instâncias reservadas (RIs): O preço por hora é fixo, independentemente do uso, e existe um prazo pré-determinado de contratação. Essa forma de pagamento tem o benefício de obter desconto, uma vez que o cliente tem o compromisso de um a três anos;

Instâncias spot: Por serem instâncias extras, ou seja, instâncias de capacidade extra na Nuvem AWS, o preço é muito mais atrativo. Por outro lado, se o EC2 precisar de capacidade, o cliente que utiliza Instância Spot será notificado 2 minutos antes que suas instâncias serão interrompidas.

Microsoft Azure



As formas de cobrança da Azure podem ser:

On-demand: Seus custos são realizados em cima dos minutos utilizados.

Neste modelo, não é necessário compromisso de tempo mínimo de contratação. Como o pagamento é feito de acordo com a utilização, é possível aumentar e diminuir recursos sem limite.

Contrato pré-definido: Como um determinado tempo de utilização é acordado, o custo é reduzido.

Acordo empresarial: Nessa modalidade, o pagamento é realizado antecipadamente, por esse motivo, há benefícios e desconto. O uso adicional é pago de forma separada, mas com desconto nas taxas.

Conclusão



- ✓ Microsoft Azure
- ✓ Principais Serviços

Próxima Aula



01. ••

Arquitetura Microserviços

02. ••

03. ••

04. ••



Banco de Dados NoSQL

Capítulo 06 – Aula 06.05 – Arquitetura Microserviços

PROF.: RICARDO BRITO ALVES

- Arquitetura Monolítica
- Arquitetura Microserviços
- Containers
- Orquestradores de Container

Arquiteturas Monolíticas



Com as arquiteturas monolíticas, todos os processos são altamente acoplados e executam como um único serviço.

Isso significa que se um processo do aplicativo apresentar um pico de demanda, toda a arquitetura deverá ser escalada. A complexidade da adição ou do aprimoramento de recursos de aplicativos monolíticos aumenta com o crescimento da base de código. Essa complexidade limita a experimentação e dificulta a implementação de novas ideias.

As arquiteturas monolíticas aumentam o risco de disponibilidade de aplicativos, pois muitos processos dependentes e altamente acoplados aumentam o impacto da falha de um único processo.

Arquitetura de Microserviços

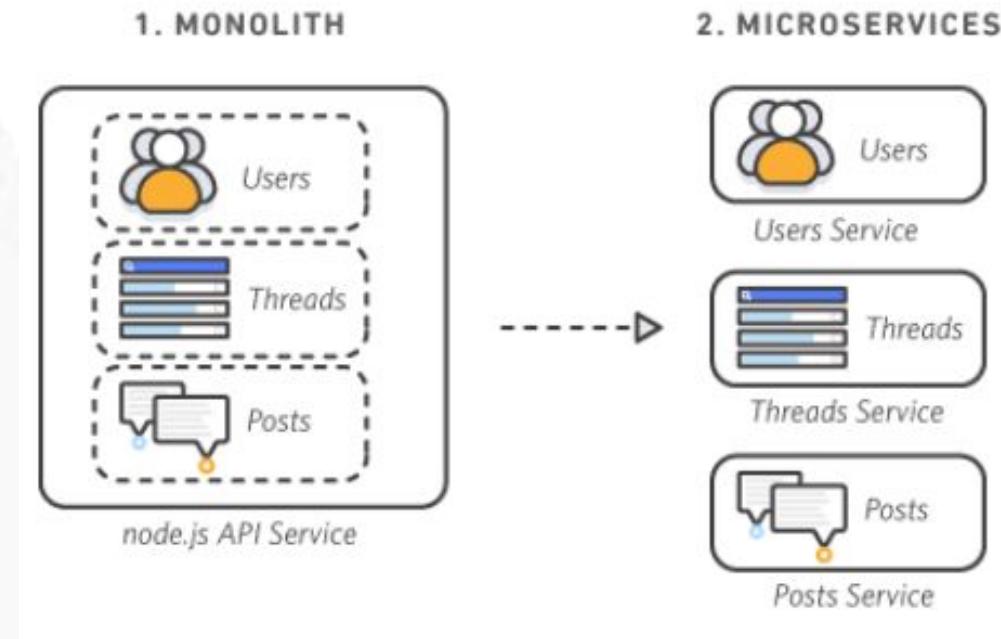


Com uma arquitetura de microserviços, um aplicativo é criado como componentes independentes que executam cada processo do aplicativo como um serviço.

Esses serviços se comunicam por meio de uma interface bem definida usando APIs leves. Os serviços são criados para recursos empresariais e cada serviço realiza uma única função. Como são executados de forma independente, cada serviço pode ser atualizado, implantado e escalado para atender a demanda de funções específicas de um aplicativo.

Arquitetura de Microserviços

IGTI

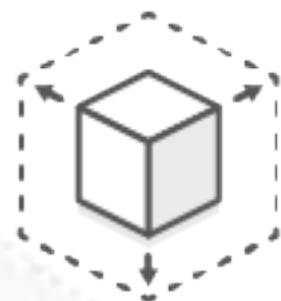


Arquitetura de Microserviços



Escalabilidade flexível

Os microserviços permitem que cada serviço seja escalado de forma independente para atender à demanda do recurso de aplicativo oferecido por esse serviço. Isso permite que as equipes dimensionem corretamente as necessidades de infraestrutura, meçam com precisão o custo de um recurso e mantenham a disponibilidade quando um serviço experimenta um pico de demanda.



Container

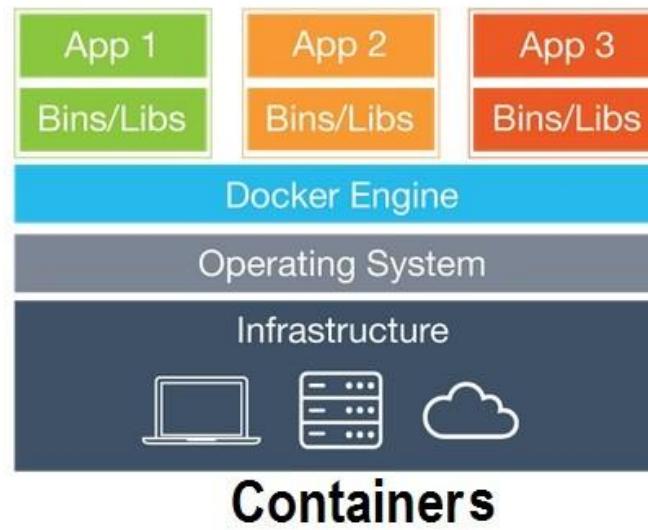
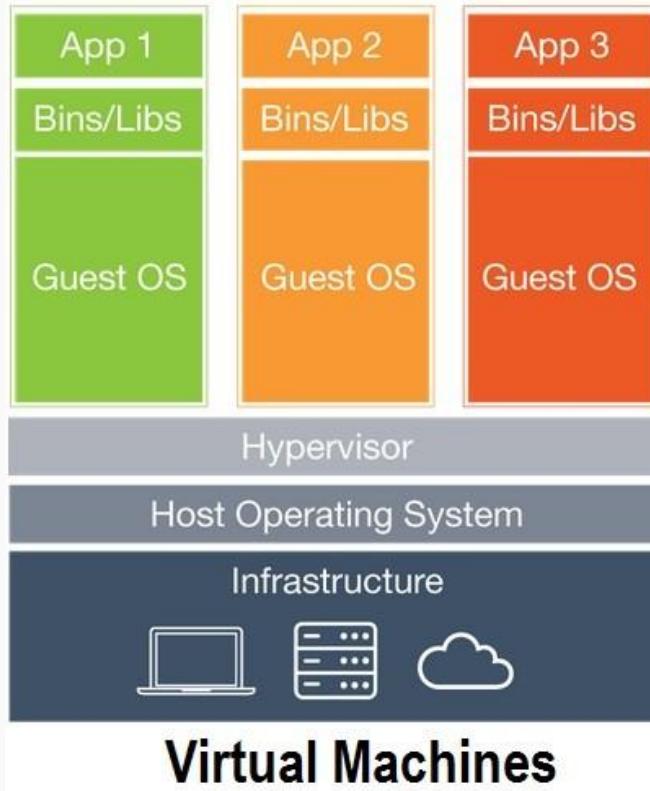


Em vez de usar um sistema operacional para cada estrutura, como na virtualização, os Containers são blocos de espaços divididos pelo Docker em um servidor, o que possibilita a implementação de estruturas de Microserviços que compartilham o mesmo sistema operacional. Porém, de forma limitada (conforme a demanda por capacidade).

O fato de os Containers não terem seus próprios sistemas operacionais, permite que eles consumam menos recursos e, com isso, sejam mais leves.

Container

IGTI



Ferramentas de Orquestração de Containers



As ferramentas de orquestração de containers são aplicações em nuvem que permitem fazer o gerenciamento de múltiplos contêineres.

Seus principais objetivos são:

- Cuidar do ciclo de vida dos containers de forma autônoma, subindo e distribuindo, conforme nossas especificações ou demandas;
- Gerenciar volumes e rede, que podem ser local ou no cloud provider de sua preferência.

Orquestradores de Containers



O Kubernetes, ECS e o Docker são as principais plataformas de gerenciamento de contêineres.

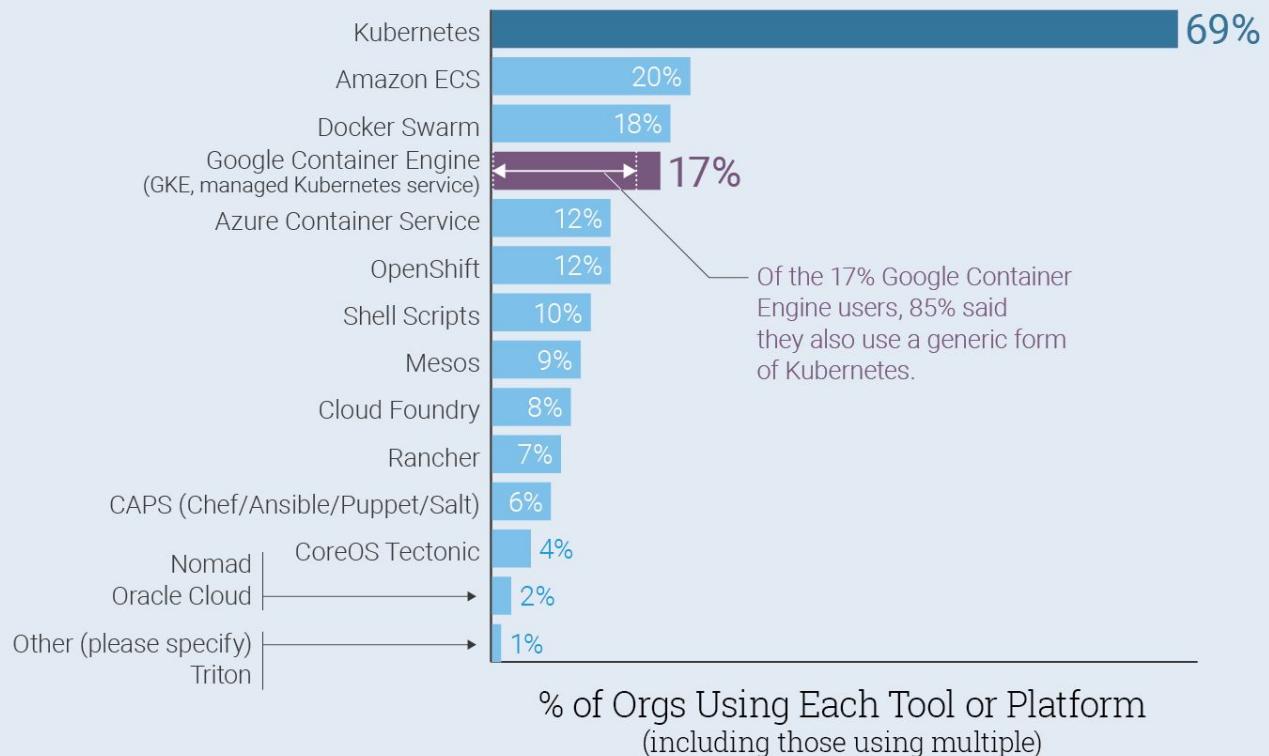
Dessa forma, essa é uma ferramenta para viabilizar a utilização de Containers e Microserviços em servidores com mais facilidade, pois permite empacotar os aplicativos para que possam ser movimentados facilmente.

O Docker permite, por exemplo, que uma biblioteca possa ser instalada em diferentes Containers sem que haja qualquer interdependência entre eles. Essa característica tem o objetivo de facilitar o gerenciamento de códigos e aplicativos.

Orquestradores de Containers

IGTI

Kubernetes Manages Containers at 69% of Organizations Surveyed



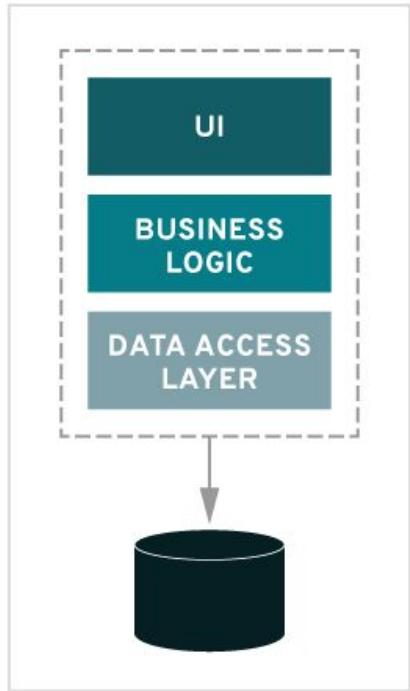
Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. Your organization manages containers with... (check all that apply)? n=763.

THE NEW STACK

Arquitetura

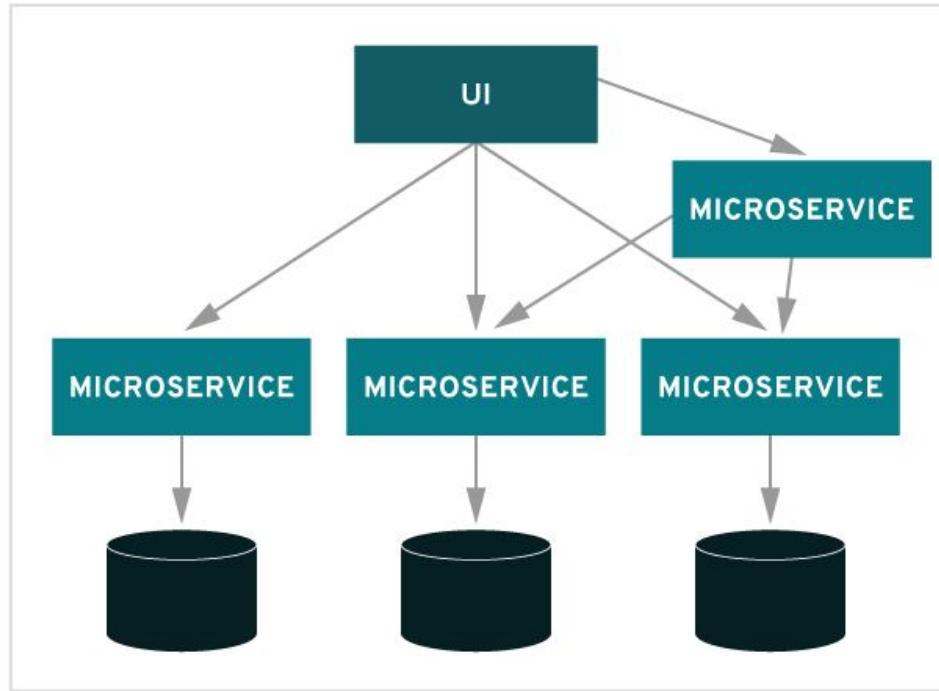
IGTI

MONOLITHIC



VS.

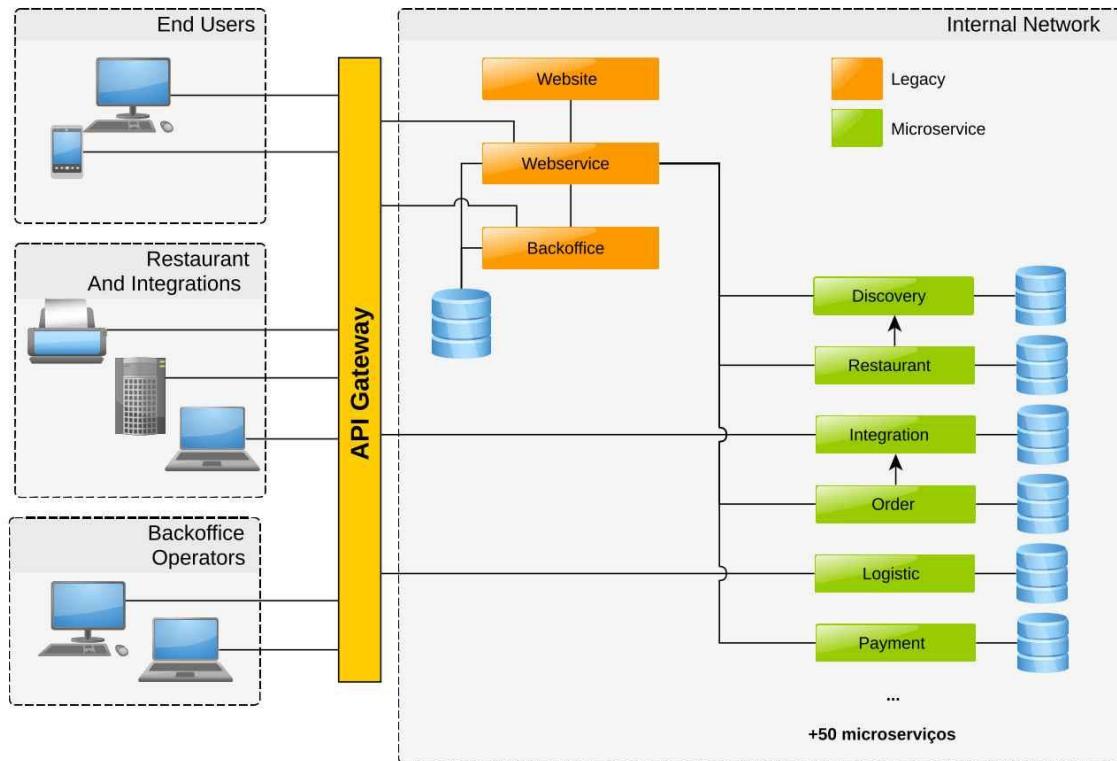
MICROSERVICES



Arquitetura

IGTI

Arquitetura de micro-serviços



Conclusão



- ✓ Arquitetura Monolítica
- ✓ Arquitetura Microserviços
- ✓ Containers
- ✓ Orquestradores de Container

Próxima Aula



01. ••

03. ••

Sistema de Arquivos Distribuídos

02. ••

04. ••

Banco de Dados NoSQL

Capítulo 06 – Aula 06.06 – Sistemas de Arquivo Distribuídos

PROF.: RICARDO BRITO ALVES

- Sistemas de Arquivo
- Sistemas de Arquivo Distribuído
- Apache Hadoop
- HDFS

Sistemas de Arquivos



- Foram originalmente desenvolvidos como um *recurso do S.O* que fornece uma interface de programação conveniente para armazenamento em disco.
- *São responsáveis pela organização, armazenamento, recuperação, atribuição de nomes, compartilhamento e proteção de arquivos.*
- Projetados para *armazenar e gerenciar um grande número de arquivos*, com recursos para criação, atribuição de nomes e exclusão de arquivos.

Sistemas de Arquivos

- Diretório é um arquivo de tipo especial;
- Fornece um mapeamento dos nomes textuais para identificadores internos;
- Podem incluir nomes de outros diretórios.

Tamanho do Arquivo
Horário de Criação
Horário de Acesso (Leitura)
Horário de Modificação (Escrita)
Horário de Alteração de Atributo
Contagem de Referência
Proprietário
Tipo de Arquivo
Lista de Controle de Acesso

Sistemas de Arquivos Distribuídos (DFS ou SAD)



Um sistema de arquivos distribuídos *permite aos programas armazenarem e acessarem arquivos remotos exatamente como se fossem locais*, possibilitando que os usuários acessem arquivos a partir de qualquer computador em uma rede.” (COULOURIS, et. al, p. 284).

- **Objetivo:** permitir que os programas armazenem e acessem arquivos remotos exatamente como se fossem locais.
- **Permitem que vários processos** compartilhem dados por longos períodos, de modo seguro e confiável.
- **O desempenho e segurança** no acesso aos arquivos armazenados em um servidor devem ser compatíveis aos arquivos armazenados em discos locais.

Requisitos de um Sistemas de Arquivos Distribuídos



- Transparência.
- Atualização concorrente de arquivos.
- Replicação de arquivos.
- Heterogeneidade.
- Tolerância a falha.
- Consistência.
- Segurança.
- Eficiência.

Arquitetura do SAD

- Modelo abstrato de arquitetura que serve para o ***Network File System (NFS)*** e ***Andrew File System (AFS)***.
- Divisão de responsabilidades entre três módulos.
 - Cliente.
 - Serviço de arquivos planos.
 - Serviço de diretórios.
- Design aberto
 - Diferentes módulos cliente podem ser utilizados para implementar diferentes interfaces.
 - Simulação de operações de arquivos de diferentes S.O.
 - Otimização de performance para diferentes configurações de hardware de clientes e servidores.



Sistemas de Arquivos Distribuídos (DFS ou SAD)



Funções de um Sistema de Arquivos Distribuído:

- ***Armazenar e compartilhar programas e dados***
 - Funções idênticas às de um sistema centralizado (local).
- ***Ênfase na disponibilidade, confiabilidade e segurança.***
- ***Desempenho***
 - Questão importante porque acessos remotos podem ser significativamente mais lentos que os locais.
 - Não se pretende em geral que o SAD seja mais rápido que um SA local, mas sim que a degradação seja aceitável.

Sistemas de Arquivos Distribuídos (DFS ou SAD)



Sistemas de arquivo distribuídos *devem ser vistos pelos clientes como um sistema de arquivo local.*

A *transparência* é muito importante para seu bom funcionamento.

É necessário *um bom controle de concorrência* no acesso.

Cache é importante.

Apache Hadoop



Hadoop é uma *plataforma de software de código aberto para o armazenamento e processamento distribuído de grandes conjuntos de dados, utilizando clusters de computadores com hardware commodity.*

Os serviços do Hadoop fornecem armazenamento , processamento, acesso, governança, segurança e operações de Dados.

Benefícios do Apache Hadoop



Algumas das razões para se usar Hadoop é a sua “capacidade de armazenar, gerenciar e analisar grandes quantidades de dados estruturados e não estruturados de forma rápida, confiável, flexível e de baixo custo.

- **Escalabilidade e desempenho** – distribuídos tratamento de dados local para cada nó em um cluster Hadoop permite armazenar, gerenciar, processar e analisar dados em escala petabyte.
- **Confiabilidade** – clusters de computação de grande porte são propensos a falhas de nós individuais no cluster. Hadoop é fundamentalmente resistente – quando um nó falha de processamento é redirecionado para os nós restantes no cluster e os dados são automaticamente re-replicado em preparação para falhas de nó futuras.

Benefícios do Apache Hadoop



- **Flexibilidade** – ao contrário de sistemas de gerenciamento de banco de dados relacionais tradicionais, você não tem que esquemas estruturados criados antes de armazenar dados. *Você pode armazenar dados em qualquer formato, incluindo formatos semi-estruturados ou não estruturados*, e em seguida, analisar e aplicar esquema para os dados quando ler.
- **Baixo custo** – ao contrário de software proprietário, *o Hadoop é open source* e é executado em hardware commodity de baixo custo.

HDFS



O HDFS (Hadoop Distributed File System) é um sistema de arquivos distribuído, *projeto para armazenar arquivos muito grandes*, com padrão de acesso aos dados streaming , utilizando clusters de servidores facilmente encontrados no mercado e de baixo ou médio custo.

Não deve ser utilizado para aplicações que precisem de acesso rápido a um determinado registro e sim para aplicações nas quais é necessário ler uma quantidade muito grande de dados.

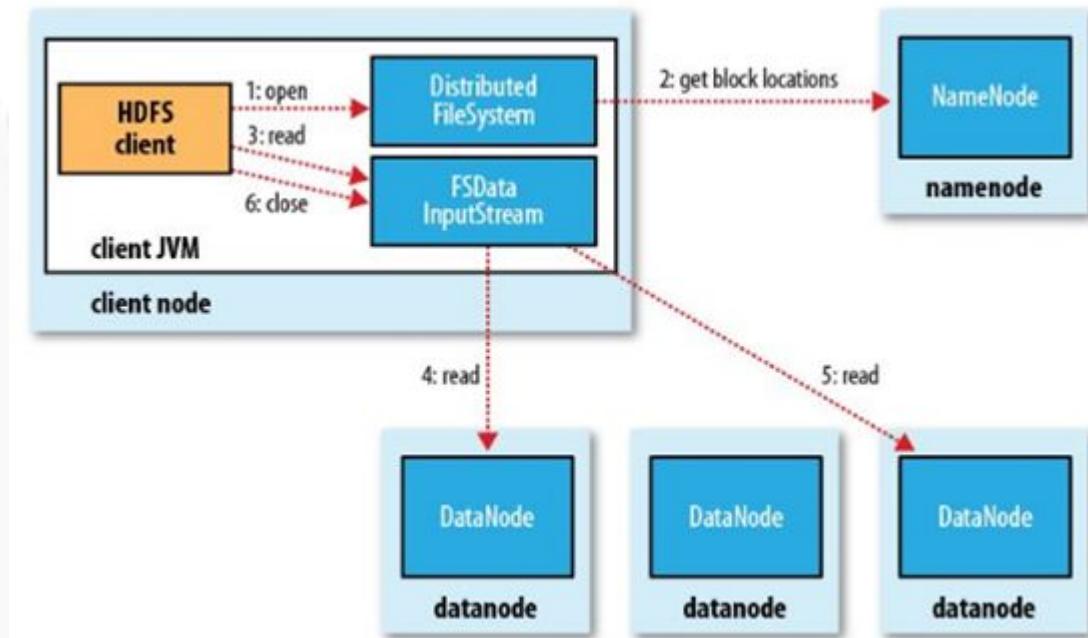
Outra questão que deve ser observada é que *não deve ser utilizado para ler muitos arquivos pequenos*, tendo em vista o overhead de memória envolvido.

Recursos do HDFS

- O HDFS tem 2 tipos de Nós : **Master** (ou Namenode) e **Worker** (ou Datanode).
 - O **Master** armazena informações da distribuição de arquivos e metadados.
 - Já o **Worker** armazena os dados propriamente ditos. Logo o Master precisa sempre estar disponível. Para garantir a disponibilidade podemos ter um backup (similar ao Cold Failover) ou termos um Master Secundário em um outro servidor. Nesta segunda opção, em caso de falha do primário, o secundário pode assumir o controle muito rapidamente.
- Tal como um sistema Unix, é possível utilizar o HDFS via linha de comando.

HDFS

IGTI



Conclusão



- ✓ Sistemas de Arquivo
- ✓ Sistemas de Arquivo Distribuído
- ✓ Apache Hadoop
- ✓ HDFS

Próxima Aula



01.

03.

MongoDB na Nuvem

02.

04.



Banco de Dados NoSQL

Capítulo 06 – Aula 06.07 – MongoDB na Nuvem

PROF.: RICARDO BRITO ALVES

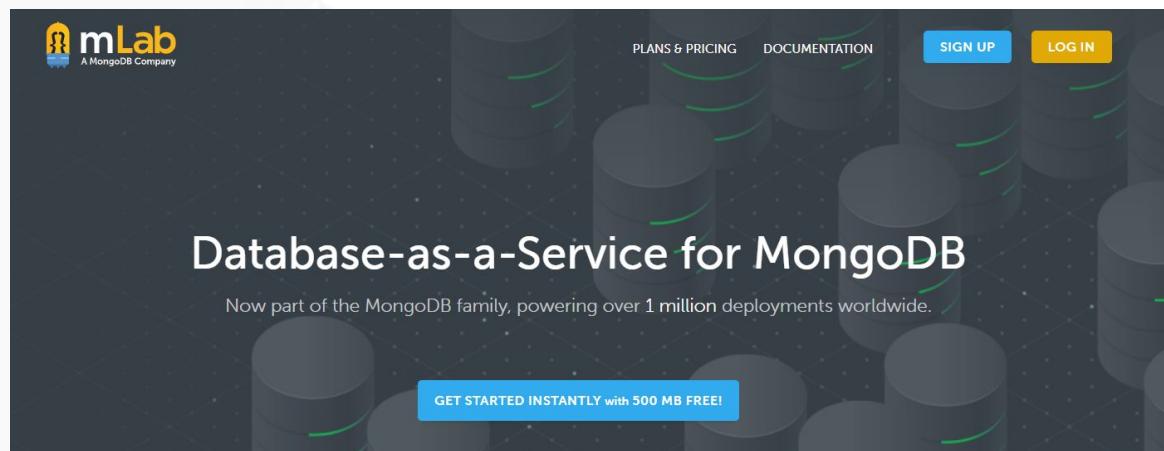
- Configurando mLab
- Conectando ao mLab

MongoDB Gratuito Hospedado na Nuvem



Conhecido um tempo atrás como MongoLab, o mLab é um serviço de banco de dados gerenciável que hospeda na nuvem um banco de dados MongoDB e é executado em provedores como a Amazon Web Services (AWS), Google Cloud e Microsoft Azure.

A parte mais interessante é que o serviço tem um plano gratuito que oferece 0,5 Gb para armazenamento de dados.



Criando uma Conta

Para que possamos utilizar o serviço, precisamos criar uma conta. O processo é bem simples e pode ser feito neste link (<https://mlab.com/>).

Nele você encontrará um formulário, basta preencher os dados e confirmar a conta no e-mail. É bem simples.

mLab is now part of the MongoDB family

If you're looking for a cloud-hosted MongoDB service similar to mLab, sign up for MongoDB Atlas, a fully-managed database-as-a-service available on AWS, Azure, and GCP

Create your account to start building your first cluster:

- Pick your preferred cloud provider: AWS, Azure, or Google
- Choose from over 60 cloud regions around the world
- Select your cluster tier and customize your storage
- Enable multi-region, workload isolation, and replication options for dedicated clusters
- Configure additional settings, including backup snapshots, sharded clusters, advanced security, and more

For more information on MongoDB Atlas pricing, features, and support, visit the [MongoDB Atlas page](#).

Try MongoDB Atlas

Used by millions of developers around the world.

Your Company (optional)

How are you using MongoDB? I'm learning MongoDB

Your Work Email

First Name Ricardo

Last Name Alves

Password

✓ 8 characters minimum

I agree to the [terms of service](#) and [privacy policy](#).

Get Started with 512 MB Free

Primeiro Acesso



Na próxima tela você terá duas escolhas a fazer: **o provedor e o plano**. Dos provedores, temos três possibilidades:

- Amazon Web Services (AWS)
- Google Cloud Platform
- Microsoft Azure

Cloud Provider & Region

AWS, N. Virginia (us-east-1) ▾

The screenshot shows a user interface for selecting a cloud provider and region. At the top, there are three buttons: AWS (highlighted with a green border), Google Cloud, and Azure. Below this, a section titled "Cloud Provider & Region" shows "AWS, N. Virginia (us-east-1)" with a dropdown arrow. The interface is divided into three main regions: ASIA, NORTH AMERICA, and EUROPE. Under ASIA, there are two options: Singapore (ap-southeast-1)★ and Mumbai (ap-south-1). Under NORTH AMERICA, there are three options: N. Virginia (us-east-1)★ (highlighted with a green border), Oregon (us-west-2)★, and Frankfurt (eu-central-1)★. Under EUROPE, there are two options: Ireland (eu-west-1)★ and Frankfurt (eu-central-1)★.

Region	Provider	Region ID	Status
ASIA	Singapore	(ap-southeast-1)	★ Recommended region
	Mumbai	(ap-south-1)	★
NORTH AMERICA	N. Virginia	(us-east-1)	★ Recommended region
	Oregon	(us-west-2)	★
	Frankfurt	(eu-central-1)	★
EUROPE	Ireland	(eu-west-1)	★
	Frankfurt	(eu-central-1)	★

Primeiro Acesso



Dos planos, também temos outras três opções:

Choose a path. Adjust anytime.

Available as a fully managed service across 60+ regions on AWS, Azure, and Google Cloud

Dedicated Multi-Region Clusters

For teams developing world-class applications that require multi-region resiliency or ultra-low latency.

- ✓ Includes all features from Shared and Dedicated Clusters
- ✓ Replicate data across multiple regions
- ✓ Globally distributed read and write operations
- ✓ Control data residency at the document level

Create a cluster

Starting at **\$0.13/hr***

*estimated cost \$98.55/month

Dedicated Clusters

For teams building applications that need advanced development and production-ready environments.

- ✓ Includes all features from Shared Clusters
- ✓ Auto-scaling
- ✓ Network isolation
- ✓ Realtime performance metrics

Create a cluster

Starting at **\$0.08/hr***

*estimated cost \$56.94/month

Shared Clusters

For teams learning MongoDB or developing small applications.

- ✓ Highly available auto-healing cluster
- ✓ End-to-end encryption
- ✓ Role-based access control

Create a cluster

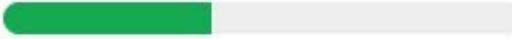
Starting at **FREE**

Guia



Connect to Atlas

Follow this checklist to get started.

40% 

- Build your first cluster
- Create your first database user
- Whitelist your IP address
- Load Sample Data (Optional)
- Connect to your cluster

No thanks

Criando Clusters



RICARDO'S ORG - 2020-11-29 > PROJECT 0

Clusters

SANDBOX

Cluster0

Version 4.2.10

[CONNECT](#) [METRICS](#) [COLLECTIONS](#) [...](#)

CLUSTER TIER

M0 Sandbox (General)

REGION

AWS / N. Virginia (us-east-1)

TYPE

Replica Set - 3 nodes

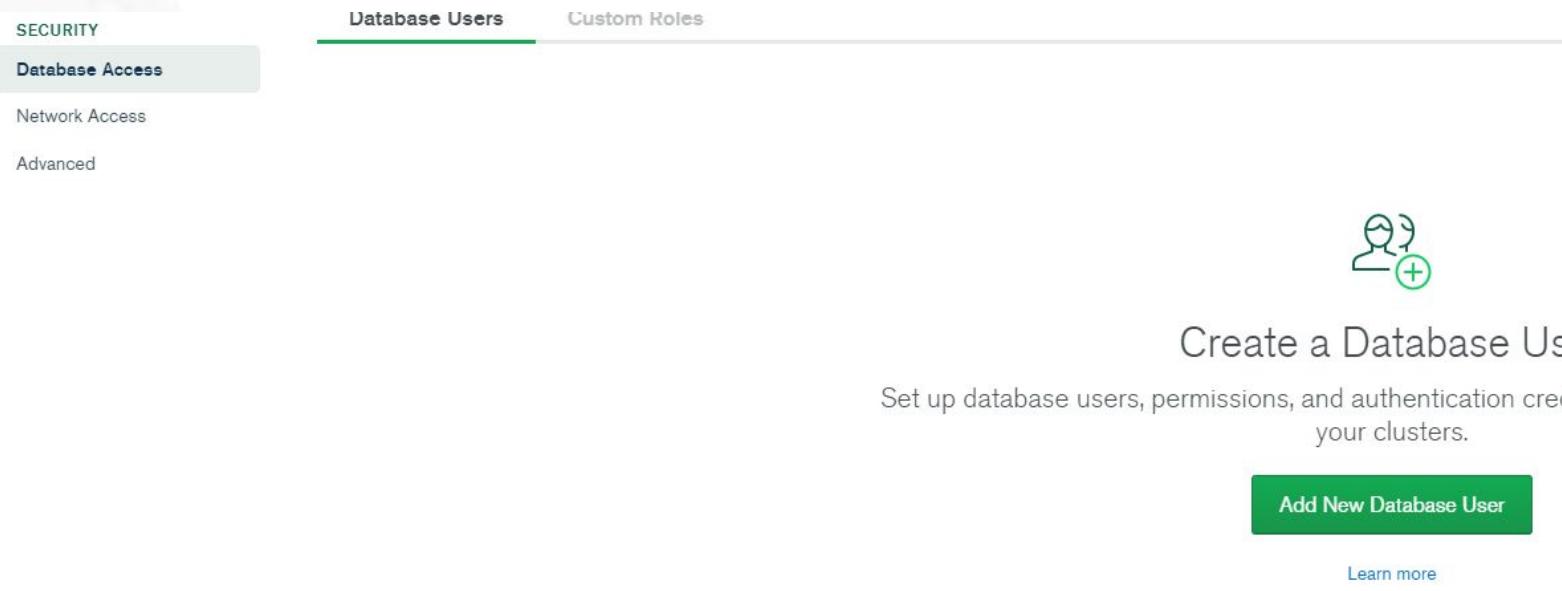
LINKED REALM APP

None Linked

Your cluster is being created

New clusters take between 1-3 minutes to provision.

Criando o Database User



The screenshot shows the AWS Database User creation interface. On the left, there's a sidebar with 'SECURITY' at the top, followed by 'Database Access' (which is highlighted in green), 'Network Access', and 'Advanced'. The main area has three tabs: 'Database Users' (highlighted in green), 'Custom Roles', and 'Custom Policies'. Below the tabs, there's a large icon of a user profile with a plus sign. The text 'Create a Database User' is centered, followed by the sub-instruction 'Set up database users, permissions, and authentication credentials for your clusters.' A green button labeled 'Add New Database User' is at the bottom, and a 'Learn more' link is just below it.

SECURITY

Database Access

Custom Roles

Database Access

Network Access

Advanced

Create a Database User

Add New Database User

Learn more

IP Access



Add IP Access List Entry

Atlas only allows client connections to a cluster from entries in the project's IP Access List. Each entry should either be a single IP address or a CIDR-notated range of addresses. [Learn more](#).

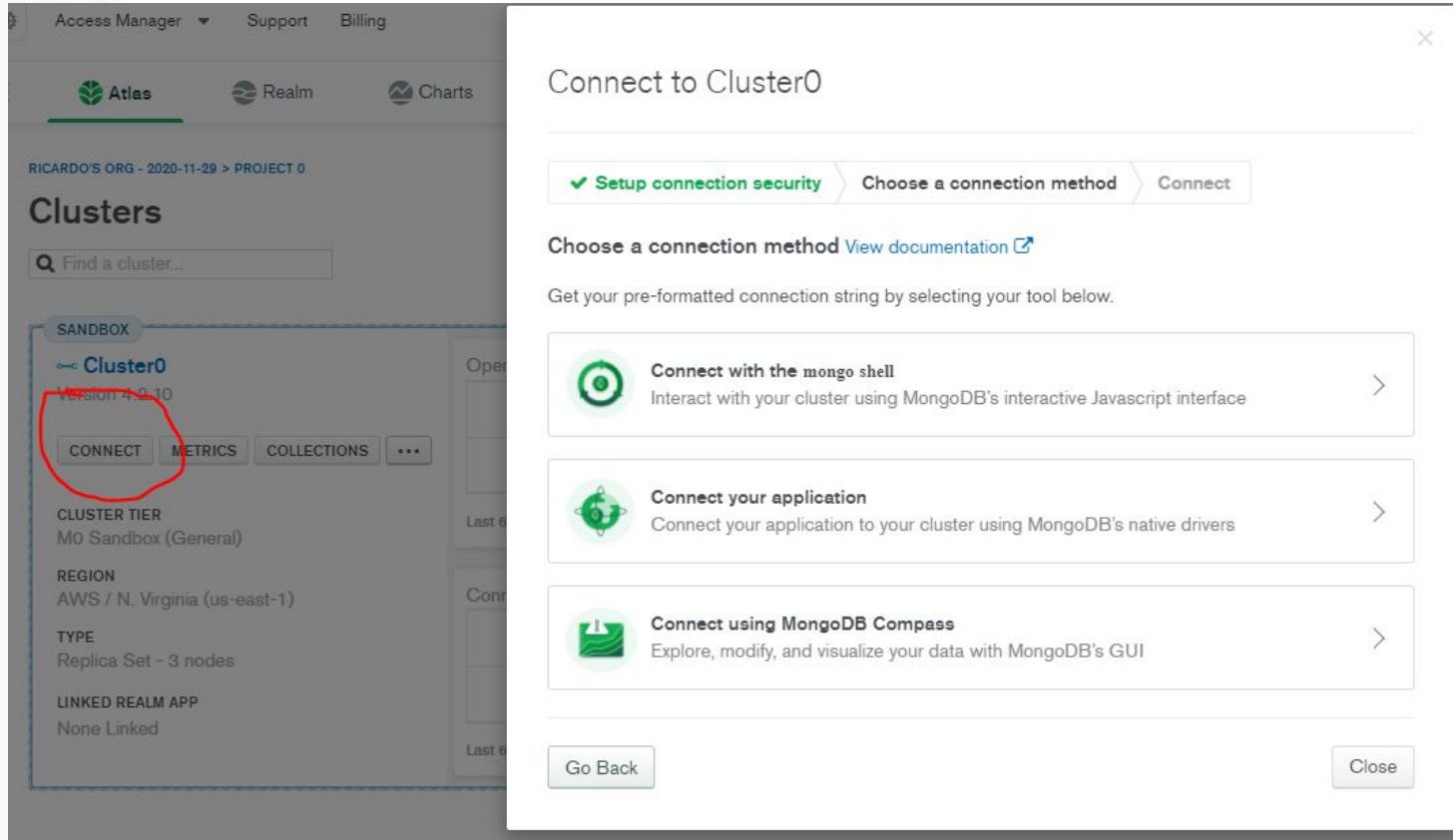
Access List Entry:

Comment:

This entry is temporary and will be deleted in

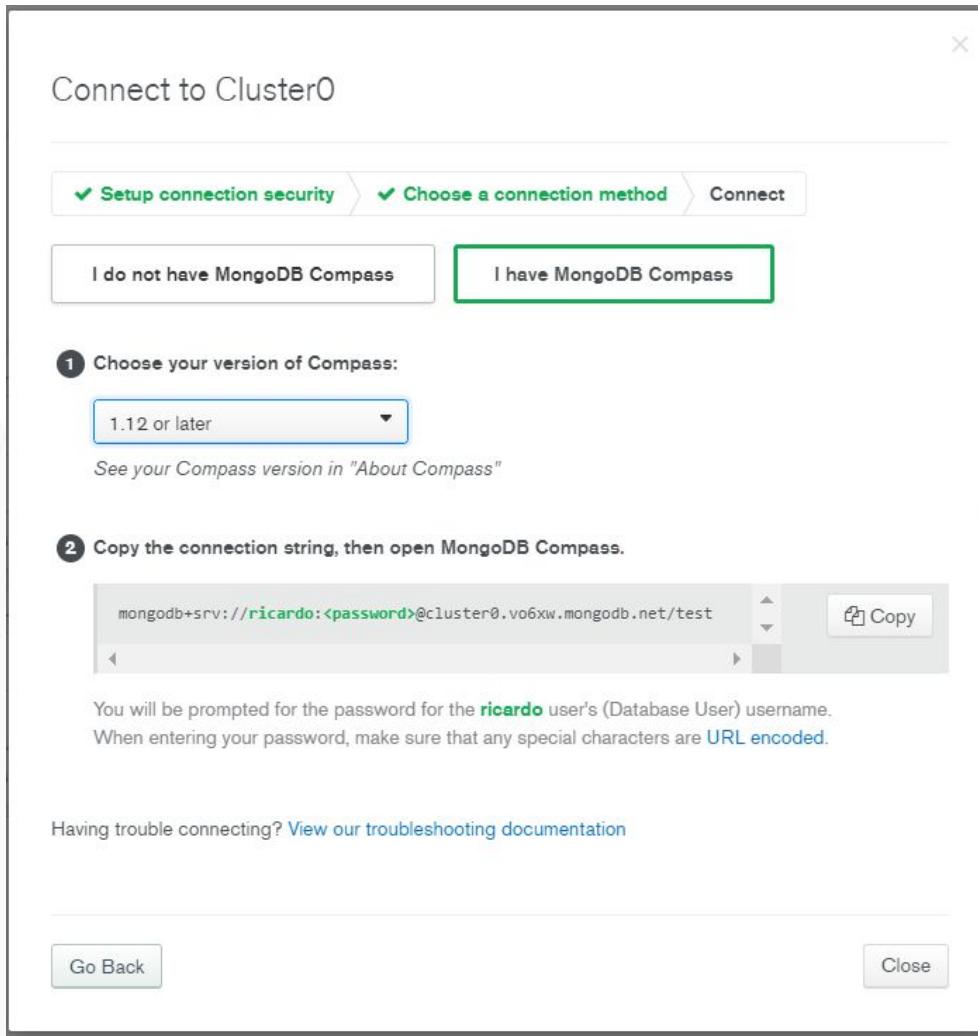
Conexão ao Cluster

IGTI



The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with 'Access Manager', 'Support', and 'Billing'. Below that is the 'Atlas' tab, which is underlined, followed by 'Realm' and 'Charts'. Under 'Clusters', it says 'RICARDO'S ORG - 2020-11-29 > PROJECT 0'. A search bar says 'Find a cluster...'. In the main area, there's a 'Sandbox' section with a 'Cluster0' entry. This entry has a red circle around the 'CONNECT' button. Below it are 'CLUSTER TIER' (M0 Sandbox (General)), 'REGION' (AWS / N. Virginia (us-east-1)), 'TYPE' (Replica Set - 3 nodes), and 'LINKED REALM APP' (None Linked). To the right of the 'Cluster0' entry is a 'Logs' section with 'Last 6 hours' and 'Conn' buttons. A modal window titled 'Connect to Cluster0' is open. It has a progress bar at the top: 'Setup connection security' (green checkmark), 'Choose a connection method' (grey), and 'Connect' (disabled grey). The 'Choose a connection method' section contains three options: 'Connect with the mongo shell' (description: 'Interact with your cluster using MongoDB's interactive Javascript interface'), 'Connect your application' (description: 'Connect your application to your cluster using MongoDB's native drivers'), and 'Connect using MongoDB Compass' (description: 'Explore, modify, and visualize your data with MongoDB's GUI'). At the bottom of the modal are 'Go Back' and 'Close' buttons.

Conexão ao Cluster



MongoDB Compass

IGTI

MongoDB Compass - cluster0.vo6xw.mongodb.net:27017

Connect View Help

Local

3 DBS 7 COLLECTIONS C

☆ FAVORITE

HOSTS

- cluster0-shard-00-01.vo6x...
- cluster0-shard-00-02.vo6x...
- cluster0-shard-00-00.vo6x...

CLUSTER

Replica Set (atlas-13e3rw-...
3 Nodes

EDITION

MongoDB 4.2.10 Enterprise

Filter your data

> admin
> config
> local

Databases Performance

CREATE DATABASE

Database Name	Storage Size	Collections	Indexes
admin	0.0B	0	0
config	0.0B	1	0
local	0.0B	6	0

Conclusão



MongoDB na nuvem:

- ✓ Configurando mLab
- ✓ Conectando ao mLab

Próxima Aula



01. ••

Data Warehouse e Data Lake

02. ••

03. ••

04. ••