

ARQUITETURA NOSQL

DOCUMENT - BASED DATABASES (MONGODB)

MARCELO MANZANO E REGINA CANTELE



4

LISTA DE FIGURAS

Figura 4.1 – Db-Engines NoSQL Documentos.....	10
Figura 4.2 – The Forrester Wave™: Big Data NoSQL	11
Figura 4.3 – Azure Cosmos DB multimodel.....	12
Figura 4.4 – Azure Cosmos DB monitoração	13
Figura 4.5 – CouchBase.....	14
Figura 4.6 – MongoDB	16
Figura 4.7 – Modelo simples de um JSON	18
Figura 4.8 – Contraste Chave-valor e Documentos	19
Figura 4.9 – Arquitetura MongoDB.....	21
Figura 4.10 – Ops Manager.....	22
Figura 4.11 – Gerenciamento de memória.....	23
Figura 4.12 – MongoDB replica set.....	26
Figura 4.13 – Replica Set regras.....	26
Figura 4.14 – Grupo de instâncias Mongod.....	27
Figura 4.15 – Indexação Mongod.....	28
Figura 4.16 – Modelo de dados.....	29
Figura 4.17 – Modelagem relacional	29
Figura 4.18 – MongoDB	30
Figura 4.19 – MongoDB	31
Figura 4.20 – Criação da conta	33
Figura 4.21 – Criar um cluster	33
Figura 4.22 – Escolher provedor do cluster.....	34
Figura 4.23 – Cluster provisionado.....	34
Figura 4.24 – Criação da conexão	35
Figura 4.25 – Criar um usuário para conexão	35
Figura 4.26 – Acesso com MongoDB Compass.....	35
Figura 4.27 – String conexão	36
Figura 4.28 – MongoDB Compass	36
Figura 4.29 – MongoDB Compass acessando Atlas	37
Figura 4.30 – Tela para download do instalador do MongoDB Community Server	38
Figura 4.31 – Instalador MongoDB Windows	38
Figura 4.32 – Tela para download do instalador do MongoDB Compass.....	39
Figura 4.33 – MongoDB Compass	39
Figura 4.34 – MongoDB interface comando de linha	40
Figura 4.35 – MongoDB	41
Figura 4.36 – Exemplo de uso dos comandos <code>use</code>	41
Figura 4.37 – Resultado da execução do comando <code>createCollection</code>	43
Figura 4.38 – Exemplo de uso dos comandos <code>show collections</code> e <code>createCollection</code>	44
Figura 4.39 – MongoDB - Exemplo de um documento e tipos de dados	46
Figura 4.40 – Exibindo o conteúdo da constante <code>db</code>	47
Figura 4.41 – Resultado da operação de inclusão na coleção emp	49
Figura 4.42 – Resultado da operação de inclusão com <i>array</i> na coleção emp	50
Figura 4.43 – Resultado da operação de inclusão com a junção de outro documento na coleção emp	51
Figura 4.44 – Resultado da operação de inclusão de um <i>array</i> de documentos na coleção emp	53

Figura 4.45 – Resultado parcial da consulta aos dados da coleção emp	56
Figura 4.46 – Resultado de uso do método <code>findOne()</code> em consulta aos dados da coleção emp	56
Figura 4.47 – MongoDB – ObjectId	57
Figura 4.48 – Resultado de uso do método <code>getTimestamp()</code>	58
Figura 4.49 – Resultado da busca por igualdade e do método <code>pretty()</code>	59
Figura 4.50 – Load Sample Dataset	74
Figura 4.51 – Carga com sucesso	74
Figura 4.52 – Associar Data Source	75
Figura 4.53 – Criar um dashboard	75
Figura 4.54 – Chart x Data Source	76
Figura 4.55 – Explorando Airbnb	77
Figura 4.56 – Airbnb Total de locações no Brasil	78
Figura 4.57 – Airbnb Valor Locação x Avaliação	79
Figura 4.58 – Create Database	80
Figura 4.59 – Importar coleção restaurantes	81
Figura 4.60 – Explorando metadados	81
Figura 4.61 – Compass explorando metadados	82
Figura 4.62 – Compass com coordenadas geolocalizadas	82
Figura 4.63 – MQL	83
Figura 4.64 – Compass visualização no mapa	83
Figura 4.65 – Explorando mapas	84
Figura 4.66 – MQL resultante	84
Figura 4.67 – MQL documentos resultantes	85
Figura 4.68 – Dashboard no Chart	85
Figura 4.69 -- Restaurantes por bairro e cozinha	86
Figura 4.70 – Total de restaurantes por bairro e cozinha	86

LISTA DE QUADROS

Quadro 4.1 – Quadro com os principais operadores de comparação	61
Quadro 4.2 – Quadro de equivalência de operadores SQL versus MongoDB	61
Quadro 4.3 – Quadro com os principais operadores lógicos.....	63
Quadro 4.4 – Quadro de equivalência de operadores lógicos SQL versus MongoDB	63
Quadro 4.5 – Quadro com os principais operadores de <i>arrays</i>	65
Quadro 4.6 – Quadro de equivalência de projeção SQL versus MongoDB	66
Quadro 4.7 – Quadro de equivalência de expressões regulares entre SQL versus MongoDB	67
Quadro 4.8 – Equivalência de operadores de atualização entre SQL versus MongoDB	69
Quadro 4.9 – Quadro de equivalência de operações de remoção entre SQL versus MongoDB	71
Quadro 4.10 – Quadro com as principais funções de ajuda do banco de dados MongoDB	71
Quadro 4.11 – Principais usos do comando <code>show</code> do banco de dados MongoDB ..	72
Quadro 4.12 – Símbolos e operadores usados em conjunto com os documentos MongoDB	72
Quadro 4.13 – Comparação entre os termos usados pelo banco de dados relacional Oracle e o banco de dados orientado a documentos MongoDB	73

LISTA DE CÓDIGOS-FONTE

Código-fonte 4.1 – Sintaxe do comando <code>use</code>	41
Código-fonte 4.2 – Sintaxe do comando <code>createCollection</code>	42
Código-fonte 4.3 – Exemplo do uso do comando <code>createCollection</code>	42
Código-fonte 4.4 – Exemplo do uso do comando <code>show collections</code>	43
Código-fonte 4.5 – Exemplo de documentos com dois campos	44
Código-fonte 4.6 – Estrutura básica de um comando no banco de dados MongoDB	47
Código-fonte 4.7 – Sintaxe do comando <code>insert</code>	48
Código-fonte 4.8 – Exemplo de inclusão de dados na coleção emp	49
Código-fonte 4.9 – Exemplo de inclusão de dados com <i>array</i> na coleção emp	50
Código-fonte 4.10 – Exemplo de inclusão de dados com a junção de outro documento na coleção emp	51
Código-fonte 4.11 – Exemplo de inclusão de dados com um <i>array</i> de documentos na coleção emp	52
Código-fonte 4.12 – Exemplo de criação de variável com oito documentos	54
Código-fonte 4.13 – Exemplo de inclusão de dados através de variável	55
Código-fonte 4.14 – Sintaxe do método <code>find()</code>	55
Código-fonte 4.15 – Exemplo de consulta aos documentos na coleção emp	55
Código-fonte 4.16 – Sintaxe do método <code>findOne()</code>	56
Código-fonte 4.17 – Exemplo de consulta ao primeiro documento na coleção emp	56
Código-fonte 4.18 – Exemplo de uso do método <code>getTimestamp()</code>	58
Código-fonte 4.19 – Exemplo de consulta por igualdade	58
Código-fonte 4.20 – Exemplo de consulta por igualdade com saída formatada pelo método <code>pretty()</code>	58
Código-fonte 4.21 – Exemplo de uso do método <code>sort()</code>	59
Código-fonte 4.22 – Exemplo de uso do operador <code>\$lt</code>	60
Código-fonte 4.23 – Exemplo de uso do operador <code>\$lte</code>	60
Código-fonte 4.24 – Exemplo de uso do operador <code>\$gt</code>	60
Código-fonte 4.25 – Exemplo de uso do operador <code>\$gte</code>	60
Código-fonte 4.26 – Exemplo de uso do operador <code>\$or</code>	62
Código-fonte 4.27 – Exemplo de uso do operador <code>\$and</code>	62
Código-fonte 4.28 – Exemplo de uso do operador <code>\$nor</code>	62
Código-fonte 4.29 – Exemplo de uso do operador <code>\$ne</code>	62
Código-fonte 4.30 – Exemplo de uso do operador <code>\$not</code>	62
Código-fonte 4.31 – Exemplo de uso do operador <code>\$in</code>	64
Código-fonte 4.32 – Exemplo de uso do operador <code>\$nin</code>	64
Código-fonte 4.33 – Exemplo de uso do operador <code>\$all</code>	64
Código-fonte 4.34 – Exemplo de uso do operador <code>\$exists</code>	64
Código-fonte 4.35 – Exemplo de projeção	65
Código-fonte 4.36 – Sintaxe do método <code>find()</code>	66
Código-fonte 4.37 – Exemplo de expressão regular	67
Código-fonte 4.38 – Exemplo de expressão regular	67
Código-fonte 4.39 – Exemplo de expressão regular	67
Código-fonte 4.40 – Exemplo de expressão regular	67
Código-fonte 4.41 – Exemplo de expressão regular	67

Código-fonte 4.42 – Sintaxe do método <code>update()</code>	68
Código-fonte 4.43 – Exemplo de alteração completa de um documento	68
Código-fonte 4.44 – Exemplo de uso do operador <code>\$inc</code>	69
Código-fonte 4.45 – Exemplo de uso do operador <code>\$set</code>	69
Código-fonte 4.46 – Exemplo de uso do operador <code>\$rename</code>	69
Código-fonte 4.47 – Exemplo de uso do operador <code>\$rename</code>	69
Código-fonte 4.48 – Sintaxe do método <code>remove()</code>	70
Código-fonte 4.49 – Exemplo de remoção documentos de uma coleção	70

EXEMPLO

SUMÁRIO

4 DOCUMENT-BASED DATABASES (MONGODB).....	9
4.1Tecnologias	9
4.2 MongoDB	11
4.3 Azure CosmosDB	12
4.4 CouchBase.....	13
4.5 CouchDB.....	14
4.6 Firebase Realtime Database	15
4.7 Mongoddb	15
4.8 Características e funcionalidades.....	17
4.9 Chave-valor e formato BSON	18
4.10 Vantagens e considerações	19
4.11 Visão geral da arquitetura	21
4.12 Modelo de dados.....	28
4.13 MongoDB Atlas	31
4.14 Construir o ambiente	32
4.15 Acessar o ambiente.....	32
4.16 Criar um cluster	33
4.17 Segurança de conexão de configuração	34
4.18 Acessar cluster com MongoDB Compass	36
4.19 On-Premises	37
4.20 Criando um banco de dados	40
4.21 Coleção	42
4.22 Documento	44
4.23 Tipos de dados.....	45
4.24 Operações CRUD.....	47
4.25 Incluindo dados	47
4.26 Inclusão de documentos por meio de variáveis.....	53
4.27 Consultando dados.....	55
4.28 Atributo _id	57
4.29 Pesquisa por igualdade.....	58
4.30 Classificando a saída dos dados.....	59
4.31 Operadores de comparação.....	60
4.32 Operadores lógicos	61
4.33 Operadores para tratamento arrays	64
4.34 Projeção	65
4.35 Expressões regulares.....	66
4.36 Alterando dados	68
4.37 Eliminando dados.....	70
4.38 Help	71
4.39 Correspondência com o relacional	73
4.40 Exemplo Airbnb Collections.....	73
4.41 Collections.....	74
4.42 Airbnb – Construir visualizações	75
4.43 Restaurantes de Nova York	79
4.43.1 Criar um banco de dados e coleção	79
4.43.2 Importar arquivo	80
4.43.3 Explorar atributos	81

4.43.4 Explorar graficamente dados de geolocalização	82
4.43.5 Utilizar MQL.....	82
4.43.6 Visualizar dados no MongoDB Charts.....	85
REFERÊNCIAS.....	88

EXEMPLO

4 DOCUMENT-BASED DATABASES (MONGODB)

Os bancos de dados de documentos foram projetados para lidar com o armazenamento e o gerenciamento de documentos em grande escala. O termo "orientado a documentos" remete a armazenar “documentos” no senso tradicional – artigos, arquivos do Microsoft Word, entre outros –, porém, um documento pode ser qualquer tipo de “Objeto sem ponteiro”.

Geralmente suportam dados mais complexos quando comparados ao armazenamento de chave-valor, índices secundários, múltiplos tipos de documentos por banco de dados e documentos aninhados ou listas.

Esse tipo do banco de dados atribui uma chave-valor a cada documento. Os documentos podem conter vários pares de chave-valor ou mesmo documentos aninhados. Os documentos são codificados em um formato padrão de troca de dados, como XML, JSON ou BSON (JSON binário), contendo assim os metadados relacionados a esses formatos, permitindo um esquema flexível e irrestrito.

De fato, eles permitem armazenar centenas de atributos em uma única coluna de um esquema de documento. Desse modo, as linhas podem receber vários valores e tipos de atributos.

4.1 Tecnologias

Muitas tecnologias estão associadas ao NoSQL orientados a documentos e o site Db-Engines classifica-os por sua popularidade atual. A popularidade usa parâmetros como número de menções do sistema em sites, interesse geral no sistema no Google Trends, frequência de discussões técnicas sobre o sistema, número de ofertas de emprego nas quais o sistema é mencionado em sites como Indeed e Simply Hired, número de perfis em redes profissionais mais populares como LinkedIn e Upwork, e relevância nas redes sociais como o número de tweets do Twitter.

48 systems in ranking, August 2020

Rank			DBMS	Database Model	Score		
Aug 2020	Jul 2020	Aug 2019			Aug 2020	Jul 2020	Aug 2019
1.	1.	1.	MongoDB +	Document, Multi-model T	443.56	+0.08	+38.99
2.	2.	2.	Amazon DynamoDB +	Multi-model T	64.75	+0.17	+8.18
3.	3.	4.	Microsoft Azure Cosmos DB +	Multi-model T	30.73	+0.32	+0.79
4.	4.	3.	Couchbase +	Document, Multi-model T	29.06	+0.36	-4.77
5.	5.	5.	CouchDB	Document	17.39	+0.71	-2.38
6.	6.	7.	Firebase Realtime Database	Document	15.31	+0.20	+3.67
7.	7.	6.	MarkLogic +	Multi-model T	12.22	+0.55	-2.25
8.	8.	8.	Realm +	Document	8.61	+0.02	+1.26
9.	9.	10.	Google Cloud Firestore	Document	7.82	-0.01	+1.56
10.	10.	12.	ArangoDB +	Multi-model T	5.73	-0.11	+0.61
11.	11.	11.	Google Cloud Datastore	Document	5.36	+0.02	-0.54
12.	12.	9.	OrientDB	Multi-model T	5.02	+0.14	-1.27
13.	13.	13.	RavenDB +	Document, Multi-model T	4.21	+0.07	-0.90
14.	14.	15.	RethinkDB	Document	4.02	+0.11	-0.09
15.	15.	14.	Cloudant	Document	3.76	-0.09	-0.82

Figura 4.1 – Db-Engines NoSQL Documentos
Fonte: DB-Engines (2020)

No relatório **The Forrester Wave™: Big Data NoSQL** (FORRESTER, 2019), do qual vários fornecedores de banco de dados NoSQL participaram, os bancos orientados a documentos se destacam também no quadrante de líderes – MongoDB e Couchbase.

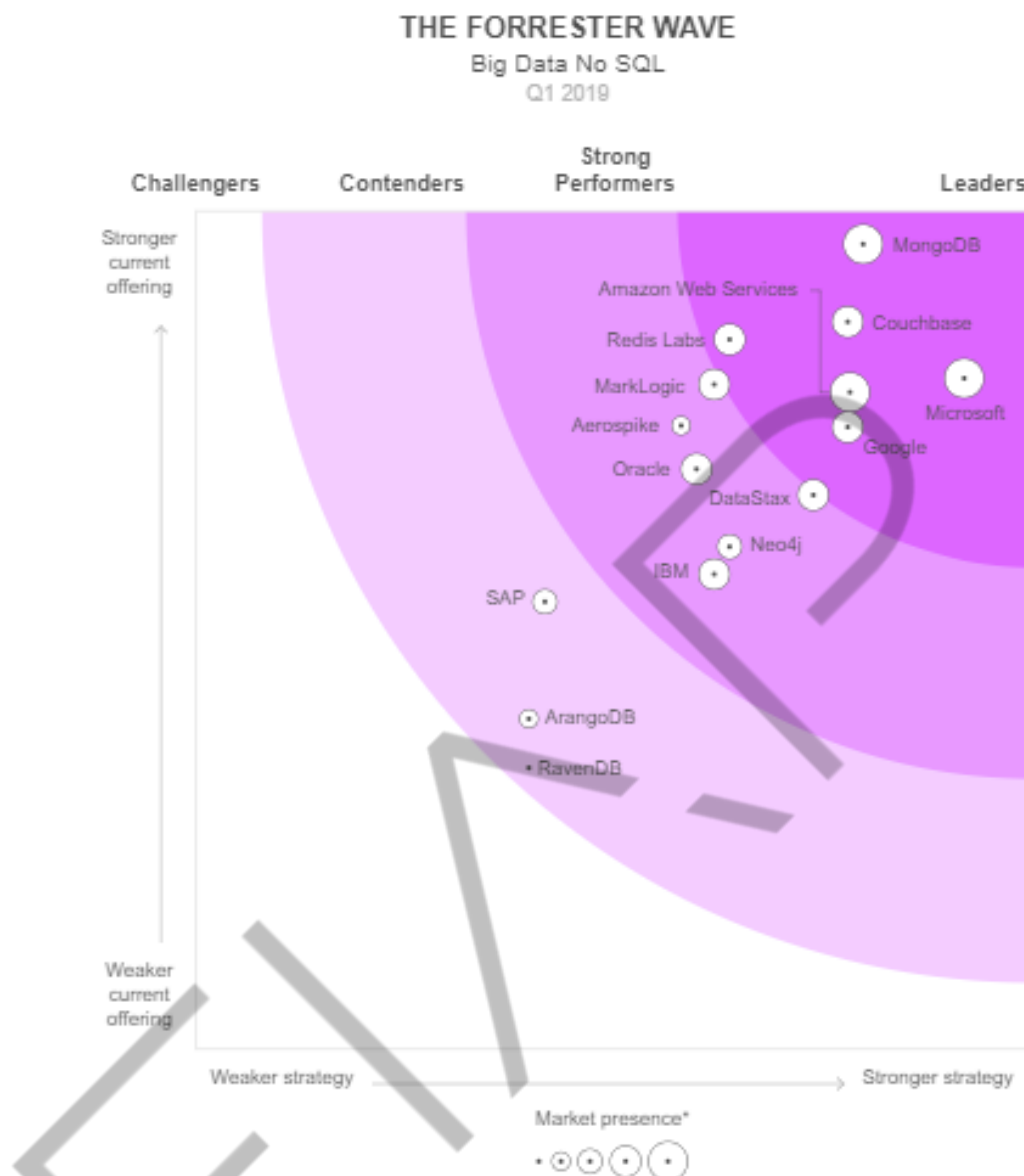


Figura 4.2 – The Forrester Wave™: Big Data NoSQL
Fonte: Forrester (2019)

Segue uma breve descrição de cinco principais: MongoDB, Azure CosmosDB, CouchBase e Firebase Realtime Database.

4.2 MongoDB

O nome MongoDB deriva da palavra inglesa humongous, termo que pode ser traduzido como “gigantesco”. É um banco de dados NoSQL orientado a documentos, escrito em C++, sua linguagem de consulta é JavaScript, desenvolvido no início de 2007, em Nova York pela empresa 10gen. No final de 2009, tornou-se open source

database server. Seu uso é tão difundido e importante que será detalhado em capítulos subsequentes desta disciplina.

4.3 Azure CosmosDB

O Azure Cosmos DB é um serviço de banco de dados multimodel (coleção, tabela, grafo) distribuído da Microsoft com contratos de nível de serviço (SLAs) para taxa de transferência, latência, disponibilidade e consistência.

Nele, um banco de dados é a unidade de gerenciamento para um conjunto de contêineres – cada contêiner tem itens independentes de esquema. Por exemplo, um item que representa uma pessoa e um item que representa um automóvel podem ser colocados no mesmo contêiner. Por padrão, todos os itens que você adiciona a um contêiner são indexados automaticamente sem a necessidade de um índice explícito ou gerenciamento de esquema.

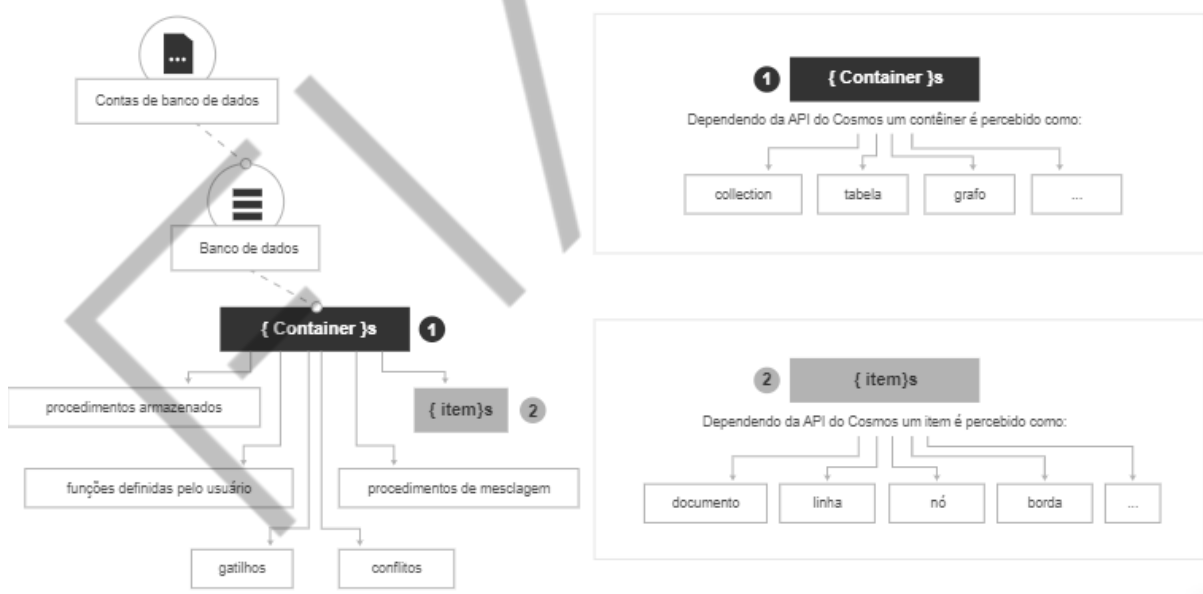


Figura 4.3 – Azure Cosmos DB multimodel
Fonte: Microsoft (2020)

Azure Cosmos DB possui recursos para monitorar a disponibilidade, o desempenho e a operação dos recursos associados ao seu ambiente e para disparar alertas sobre eventos controlados sobre esses dados.

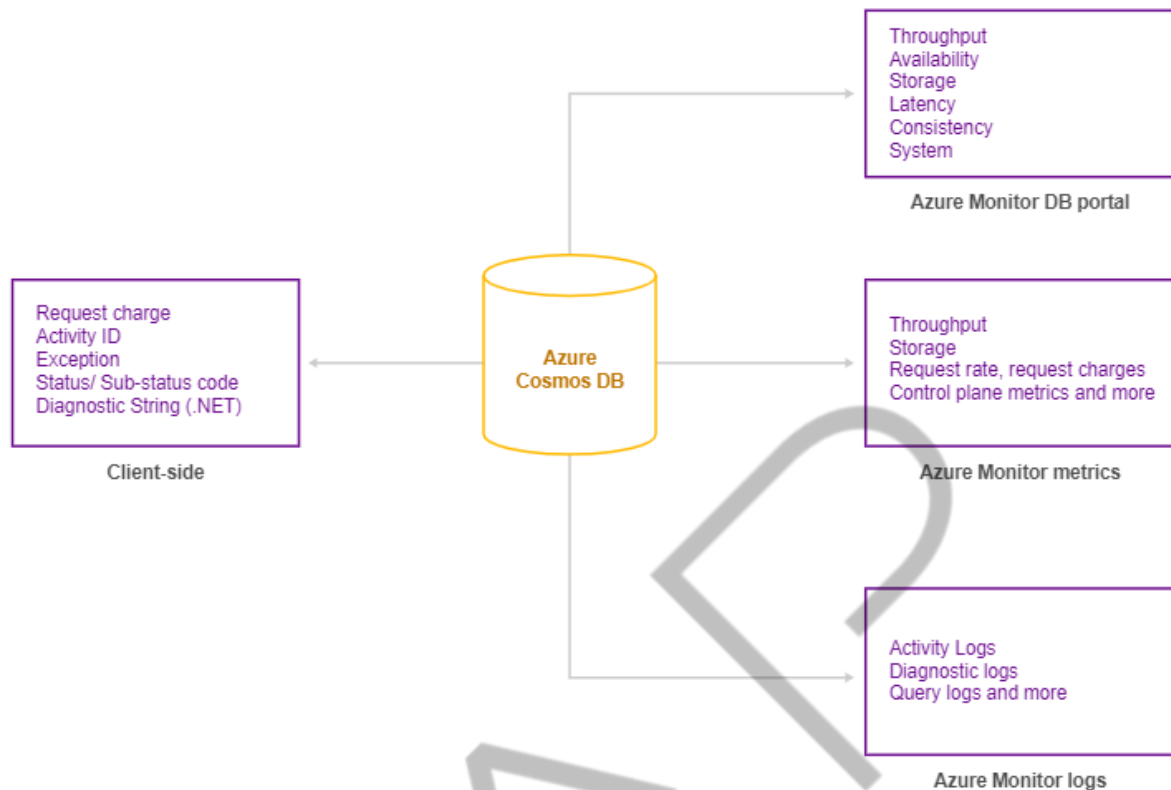


Figura 4.4 – Azure Cosmos DB monitoração
Fonte: Microsoft (2020)

Para popularizar seu uso, a Microsoft permite a gratuidade por trinta dias.

<https://azure.microsoft.com/pt-br/try/cosmosdb/>.

4.4 CouchBase

Lançado em 2010 por ex-alunos do projeto Memcached, o Couchbase é um banco de dados orientado a documentos distribuído podendo ser hospedado localmente ou em todos os principais provedores de nuvem. Favorece a consistência dos dados e tem uma ferramenta de consulta SQL padronizada chamada N1QL. Isso pode facilitar seu manuseio por desenvolvedores familiarizados com as bases SQL. Com uma solução de cache integrada, o Couchbase visa a aplicativos da web. No Couchbase, os dados são replicados no modelo multimestre.

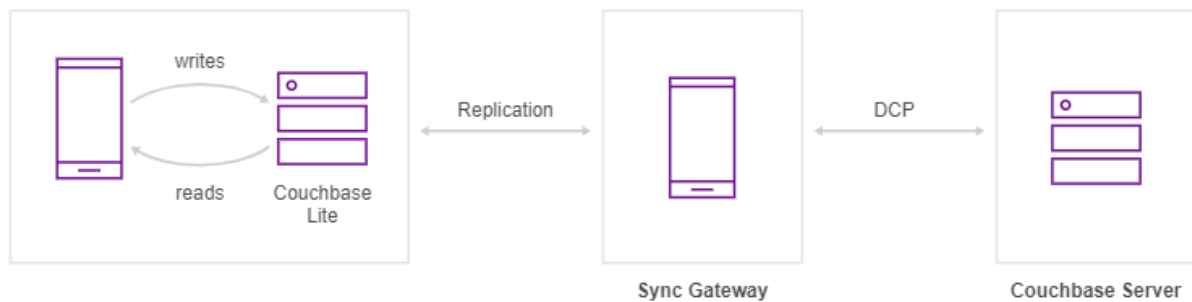


Figura 4.5 – CouchBase
Fonte: Hedgpeth (2019)

O Couchbase Mobile estende o Couchbase fornecendo a capacidade de gerenciar e sincronizar dados com segurança de qualquer nuvem / servidor para cada dispositivo móvel com três principais componentes:

- Couchbase Lite, um banco de dados NoSQL para documentos JSON para os aplicativos móveis compatível com aplicativos iOS, Android, Xamarin e IoT;
- Sync Gateway, um mecanismo de sincronização voltado para a Internet que sincroniza dados com segurança entre clientes móveis e servidor; e
- Couchbase Server, uma plataforma de banco de dados NoSQL distribuída e altamente escalável.

4.5 CouchDB

CouchDB é um banco de dados que abraça completamente a web. Os dados são armazenados em documentos JSON e acessados com o navegador da web. Funciona bem com aplicativos móveis e da Web. Os dados podem ser distribuídos de maneira eficiente com replicação incremental e tolerante a partições, com suporte a configurações multimestre com detecção automática de conflito. Isso significa que alterações podem ser replicadas bidireccionalmente.

Cada documento é nomeado exclusivamente no banco de dados. Documentos são a unidade principal de dados e incluem metadados que são mantidos pelo sistema de banco de dados. Os campos do documento são nomeados exclusivamente e contêm valores de dados variáveis como texto, número, booleano, listas, entre outros, e não há limite definido para o tamanho do texto ou número de elementos.

Apresenta todas as propriedades ACID (Atomic, Consistent, Isolated, Durable) e no disco, o CouchDB nunca substitui dados confirmados ou estruturas associadas, garantindo que o arquivo do banco de dados mantenha sempre um estado consistente.

4.6 Firebase Realtime Database

O Firebase é um kit de ferramentas desenvolvido pelo Google que fornece serviços como análises, bancos de dados, autenticação, entre outros. É uma plataforma confiável usada por empresas conhecidas como Alibaba, New York Times, Duolingo e Shazam.

O Firebase Realtime Database é um banco de dados NoSQL desse kit, que armazena os dados no formato JSON, baseado em nuvem e sincroniza dados em tempo real para cada cliente conectado. Por exemplo, pode armazenar os dados obtidos com o ping de beacons ou dados de localização de dispositivos móveis.

Quando os desenvolvedores criam plataforma para aplicativos em Android, iOS e JavaScript SDK, todos os clientes compartilharão uma instância Firebase Realtime Database e receberão as atualizações de dados mais recentes automaticamente.

Quando os usuários estiverem offline, ainda podem salvar dados no aplicativo e atualizar os dados depois que a rede estiver conectada ao servidor. O servidor atualiza os dados para cada dispositivo conectado automaticamente. Portanto, é responsivo mesmo quando estiver offline. O Firebase Realtime Database pode ser acessado diretamente de um dispositivo móvel ou navegador da web, portanto, não requer um servidor de aplicativos.

Pratique em:

<<https://firebase.google.com/products/realtime-database?hl=pt-br>>.

4.7 Mongodb

O MongoDB é um tipo de servidor de banco de dados orientado a documentos desenvolvido na linguagem de programação C++. A palavra Mongo é derivada de

Humongous. O MongoDB é um produto de servidor de banco de dados de software livre que é usado para armazenamento orientado a documentos.

O MongoDB melhorou o desempenho, porque torna o armazenamento de dados mais rápido e fácil usando esquemas de bancos de dados dinâmicos semelhantes ao JSON, em vez de sistemas de bancos de dados relacionais tradicionais de tabelas e SQL. Por esse motivo, o MongoDB é categorizado como o servidor de banco de dados NoSQL. O esquema do banco de dados dinâmico usado no MongoDB é chamado de BSON.



Figura 4.6 – MongoDB
Fonte: MongoDB (2019)

O MongoDB foi desenvolvido em 2007 por uma organização com sede em Nova York, a 10gen, que agora se chama MongoDB Inc. O MongoDB foi desenvolvido inicialmente como PAAS (Platform as a service).

Mais tarde, no ano de 2009, o MongoDB foi introduzido no mercado como um servidor de banco de dados de código aberto que foi mantido e apoiado pelo MongoDB Inc. Muitas organizações de grande e médio porte como SourceForge, Foursquare, craigslist e eBay estão usando o MongoDB em desenvolvimento de suas aplicações de banco de dados.

Esse Banco de Dados tem como característica ser código-fonte aberto licenciado pela GNU AGPL (Aferro General Public License) versão 3.0, possuir alta performance, não possuir esquemas. Diversas linguagens e plataforma já possuem drivers para o MongoDB, entre elas destacam-se: C, C#, C++, Haskell, Java,

JavaScript, Perl, **PHP**, Python, Ruby e Scala. Além disso, o MongoDB possui binários para diversas plataformas como Windows, Mac OS X, Linux e Solaris.

4.8 Características e funcionalidades

O MongoDB é um produto de servidor de banco de dados de software livre que é usado para armazenamento orientado a documentos. O MongoDB melhorou o desempenho porque torna o armazenamento de dados mais rápido e fácil usando esquemas de bancos de dados dinâmicos semelhantes ao JSON, em vez de sistemas de bancos de dados relacionais tradicionais de tabelas e SQL. Por esse motivo, o MongoDB é categorizado como o servidor de banco de dados NoSQL. O esquema do banco de dados dinâmico usado no MongoDB é chamado de BSON.

Basicamente, nesse tipo de banco (document-based ou document-oriented), temos coleções de documentos, nas quais cada documento é autossuficiente, ele contém todos os dados que possa precisar, em vez do conceito de não repetição + chaves estrangeiras do modelo relacional.

A ideia é que você não tenha de fazer JOINS, pois eles prejudicam muito a performance em suas queries (são um mal necessário no modelo relacional, infelizmente). Você modela a sua base de forma que a cada query você vai uma vez ao banco e com apenas uma chave primária pega tudo o que precisa.

O escalonamento horizontal com Sharding é muito bem implementado no MongoDB. Sharding é utilizado quando temos muitos dados e estamos no limite do disco, dessa forma, dividimos esses dados entre várias máquinas, assim temos mais rendimento e maior capacidade de armazenamento em disco.

Portanto, quanto mais Shards, maior será o armazenamento e o desempenho. O MongoDB disponibiliza essa opção. Para mais detalhes sobre Sharding podemos consultar a documentação oficial na página oficial do [MongoDB](#). Banco de dados relacionais muito utilizados como o MySQL não suportam esse tipo de solução por padrão, para isso teríamos que manipular os dados em uma camada acima da base de dados, sendo muito mais trabalhoso.

4.9 Chave-valor e formato BSON

Dados desestruturados são um problema para a imensa maioria dos bancos de dados relacionais, mas não tanto para o MongoDB. Quando o seu schema é variável, é livre, usar MongoDB vem muito bem a calhar. Os documentos BSON (JSON binário) do Mongo são schemaless e aceitam quase qualquer coisa que você quiser armazenar, sendo um mecanismo de persistência perfeito para uso com tecnologias que trabalham com JSON nativamente, como JavaScript (e consequentemente Node.js).

BSON Document: cenários altamente recomendados e utilizados atualmente são os catálogos de produtos de e-commerces. Telas de detalhes de produto em ecommerces são extremamente complicadas devido à diversidade de informações aliada a milhares de variações de características entre os produtos, que acabam resultando em dezenas de tabelas se aplicado sobre o modelo relacional. Em MongoDB essa problemática é tratada de uma maneira muito mais simples, que explicarei mais adiante.



```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

← field: value
← field: value
← field: value
← field: value

Figura 4.7 – Modelo simples de um JSON
Fonte: MongoDB (2020)

Além do formato de documentos utilizado pelo MongoDB ser perfeitamente intercambiável com o JSON serializado do JS, MongoDB opera basicamente de maneira assíncrona em suas operações, assim como o próprio Node.js, o que nos permite ter uma persistência extremamente veloz, aliado a uma plataforma de programação igualmente rápida.

Os bancos de dados orientados a documentos são uma das principais categorias de bancos de dados NoSQL. Os bancos de dados de gráficos são semelhantes, mas adicionam outra camada, o relacionamento, o que lhes permite vincular documentos para uma passagem rápida.

Os bancos de dados orientados a documentos são inerentes aos bancos de dados chave-valor. A diferença reside na forma como os dados são processados; em um banco de dados chave-valor, os dados são considerados inerentemente para o banco de dados, enquanto um sistema orientado a documentos depende da estrutura interna no documento para extrair metadados que o mecanismo de banco de dados usa para otimização adicional.

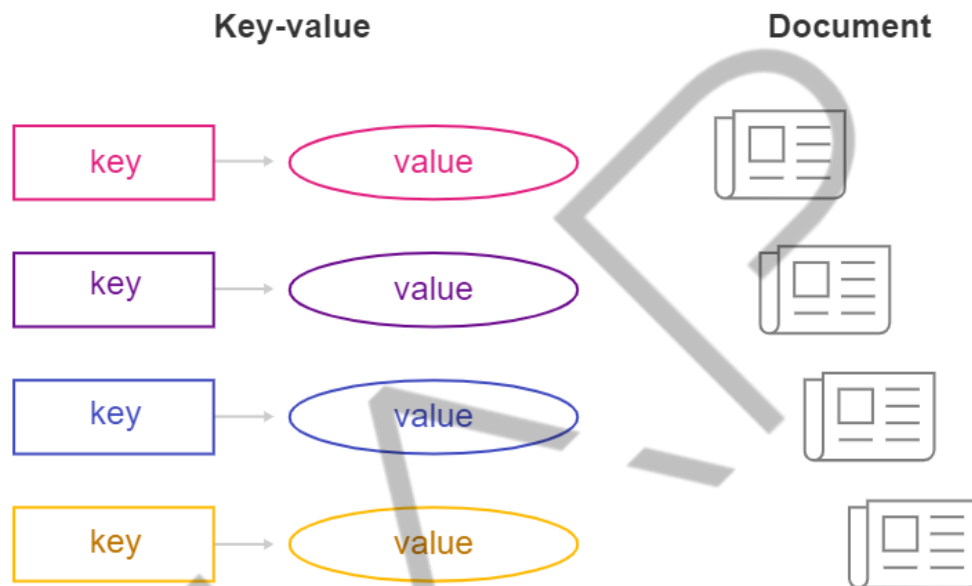


Figura 4.8 – Contraste Chave-valor e Documentos
Fonte: Elaborado pelo autor (2019)

Os bancos de dados de documentos armazenam todas as informações para um determinado objeto em uma única instância. Isso faz com que o mapeamento de objetos no banco de dados seja uma tarefa simples, eliminando o mapeamento objeto-relacional, tornando-os atraentes para aplicativos web, que estão sujeitos a mudanças contínuas, sendo a velocidade uma questão importante.

4.10 Vantagens e considerações

São várias as vantagens de usar o MongoDB em servidores de banco de dados contemporâneos.

- **Simplicidade:** o MongoDB sendo um sistema de gerenciamento de banco de dados não sql é muito mais simples, e as complexidades que vêm com bancos de dados relacionais são removidas no MongoDB. Como JSON, o

armazenamento orientado a documentos simplifica os sistemas de banco de dados.

- **Replicação e Confiabilidade de Dados:** o MongoDB permite que os usuários repliquem dados em vários servidores espelhados, o que garante a confiabilidade dos dados. No caso de um servidor travar, seu espelho ainda estará disponível e o processamento do banco de dados permanecerá inalterado.
- **Consultas NoSQL:** o MongoDB possui um mecanismo de consulta não sql que resulta em funcionalidades de armazenamento e recuperação de dados extremamente rápidas. As consultas orientadas a documentos com base em JSON são extremamente rápidas em comparação com as consultas sql tradicionais.
- **Migrações Gratuitas de Esquema:** no MongoDB, o esquema é definido pelo código. Portanto, no caso de migrações de banco de dados, não ocorre nenhum problema de compatibilidade de esquema.
- **Escalabilidade Horizontal Eficiente:** como o MongoDB é um banco de dados não relacional, ele é mais adequado para os cenários em que a escalabilidade horizontal é importante.
- **Código aberto:** o MongoDB é um servidor de banco de dados que é de código aberto e personalizável de acordo com os requisitos da organização.

O MongoDB é altamente recomendado em cenários em que o processamento rápido e a simplicidade são a chave. Devido ao seu mecanismo de consulta baseado no NoSQL, ele é robusto, escalável e altamente eficiente. Uma desvantagem é quando queremos alterar todos os registros relacionados a uma unidade semântica, nesse caso é preciso tratar um a um.

Uma pergunta a ser respondida ainda é sobre a dúvida que todos os usuários têm em relação ao suporte do MongoDB que todas as grandes empresas e projetos de alto valor necessitam. Até que ponto o MongoDB oferece esse suporte necessário? Outro ponto é em relação à disponibilidade do serviço. Essas e outras perguntas serão respondidas ao longo do tempo, o fato é que muitas empresas de grande porte têm utilizado em projetos de complexidade média, uma escolha natural devido ao pequeno

tempo que essa nova tecnologia está no mercado e os resultados têm sido muito satisfatórios.

Este é um vacilo muito comum por aqueles que se apaixonam pelo paradigma documental: armazenar objetos ultra complexos. Tenha em mente algumas limitações do MongoDB:

- Seu documento pode ter no máximo 16 Mb de tamanho.
- O nível máximo de profundidade de um documento é 100.

Convenhamos: escrever um documento dessa magnitude não é uma boa ideia em lugar algum, mas já ouvi alguns casos nos quais isso ocorreu. Sendo assim tenha isso em mente.

4.11 Visão geral da arquitetura

A arquitetura do MongoDB Enterprise Server possui várias camadas e componentes.

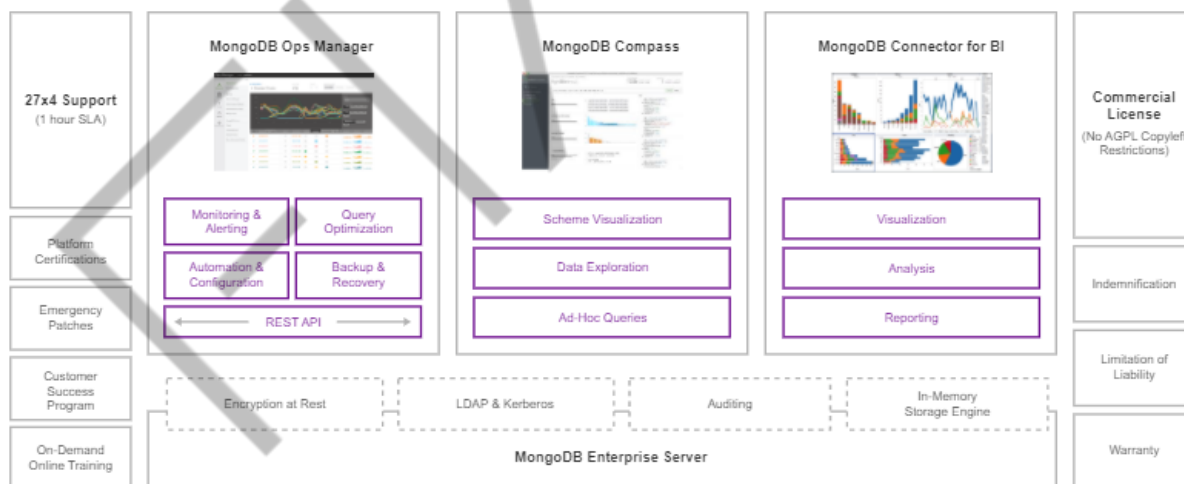


Figura 4.9 – Arquitetura MongoDB
Fonte: MongoDB (2020)

As camadas centrais e mais importantes são Ops Manager, Compass e Connector for BI. O MongoDB Ops Manager é um módulo para realizar o gerenciamento do banco de dados com funcionalidades como monitoração e alertas, otimização de consultas, automação, configuração e backup e recovery.

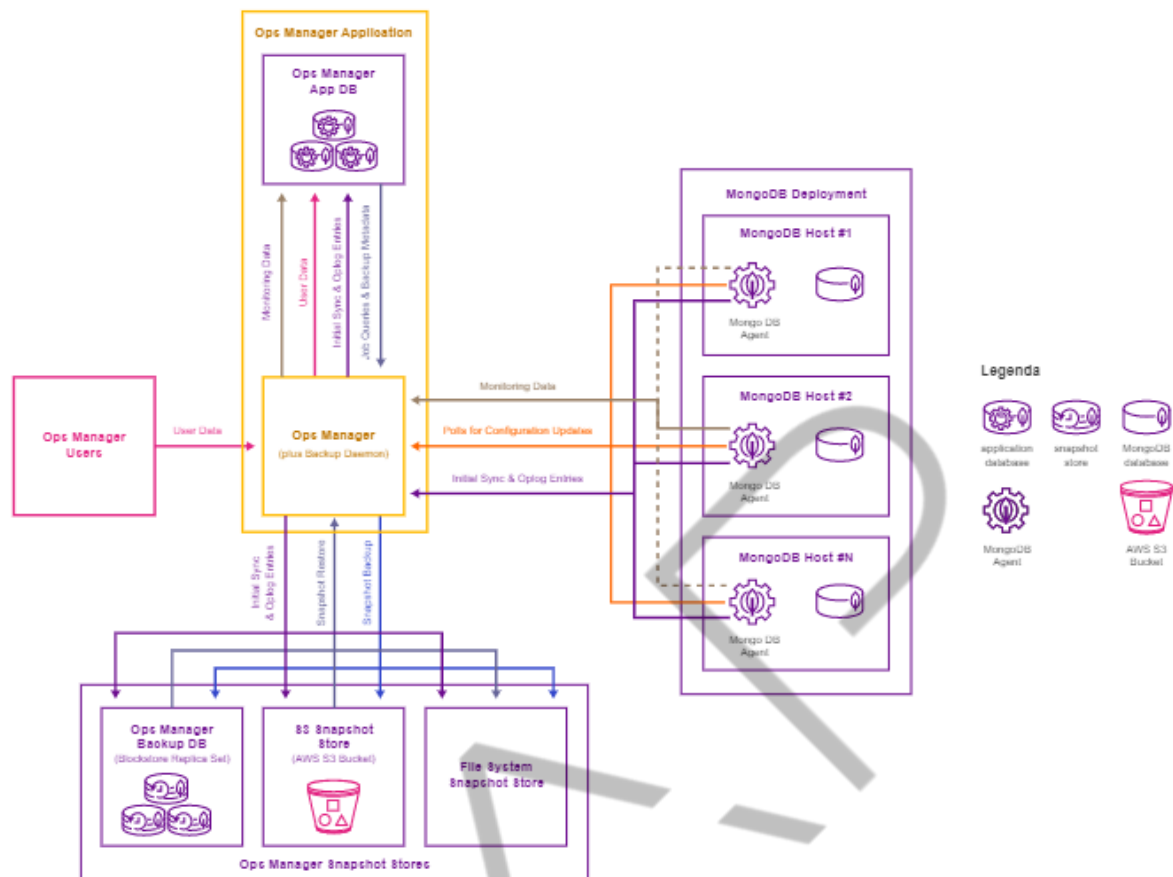


Figura 4.10 – Ops Manager
Fonte: MongoDB (2020)

MongoDB Compass é uma interface gráfica *client* para acessar o banco de dados permitindo a visualização do schema de dados, a exploração dos dados armazenados e consultas ad-hoc.

MongoDB Connector for BI tem diversos dispositivos para conexão com as mais diferentes ferramentas de visualização analíticas do mercado, como Tableau, Microstrateg e Looker. A camada do Enterprise Server tem recursos como engine para gravação em memória, auditoria, autenticação e criptografia.

Os administradores podem usar o log de auditoria nativo do MongoDB para registrar todas as atividades e alterações realizadas no banco de dados. A autenticação para simplificar o controle de acesso ao banco de dados, incluindo LDAP, Windows Active Directory, Kerberos, certificados x.509 e AWS IAM.

A criptografia dos dados está presente no MongoDB enquanto os dados estiverem circulando pela rede, no uso do banco de dados e nos dados armazenados, seja no disco ou em backups.

Uma camada à esquerda destaca suporte 24x7 composta por plataforma para certificações, patches emergenciais, trocas de experiências entre clientes e treinamentos. Uma camada à direita para licenciamento comercial com aspectos relacionados a indenizações, limitação de responsabilidades e garantias.

O MongoDB é altamente escalável e não possui um ponto único de falha. É composto por um nó árbitro (heartbeat), um nó primário (primary ou master) e muitos nós secundários (secondary ou slaves), ou seja, replicação master-slave. Os nós secundários são cópias do nó primário e usados para leituras ou backups.

O MongoDB usa arquivos mapeados de memória para gerenciamento de arquivos, portanto, nesse componente, ele depende muito do lado do sistema operacional. Mas vamos ver isso com base no sistema operacional Linux. O MongoDB mapeia arquivos na memória usando a chamada padrão mmap da glibc.

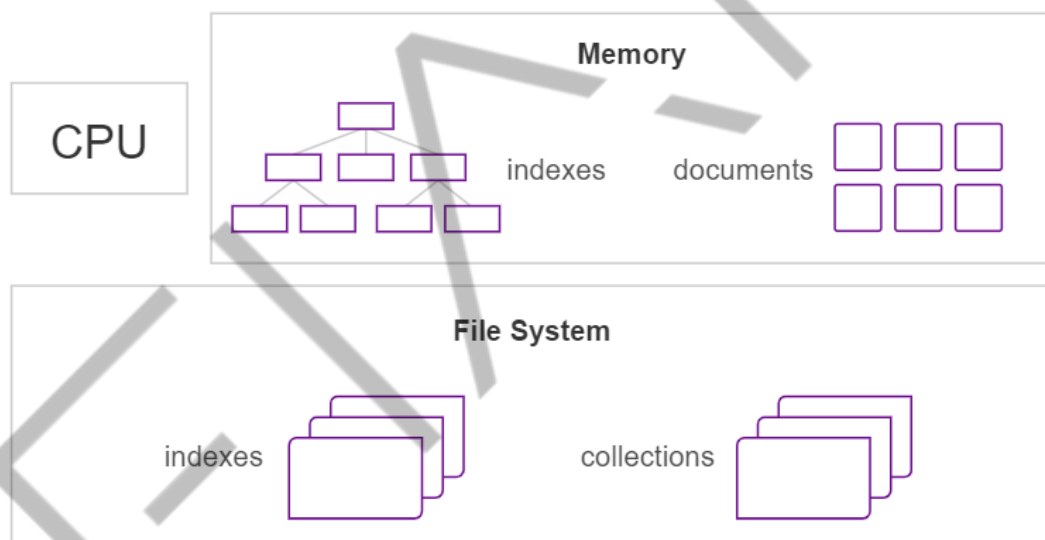


Figura 4.11 – Gerenciamento de memória
Fonte: MongoDB (2020)

Como funciona?

No início, o MongoDB mapeia todos os seus arquivos de armazenamento de dados na memória. RAM não é usado de todo, esse processo só reserva o espaço de endereço. Isso é refletido como uso de memória virtual pelo processo mongod. Eles podem ser encontrados no topo (VIRT) ou no monitor MongoDB (Memory Virt). Basicamente, eles refletem o tamanho dos dados + índices. O uso de memória Res / RSS não reflete nada de relevante.

Então, de certo modo, todos os dados e índices estão na memória, divididos por pequenas páginas de memória. O MongoDB possui ponteiros para cada página, mas eles ainda não estão na RAM.

Quando uma consulta real é feita, o MongoDB localiza a página que contém os dados necessários e lê esses dados a partir da memória em geral, e não importa se os dados estão na RAM ou não. Como não pode saber disso, apenas envia o pedido para o endereço na memória.

O resto vai até o sistema operacional. Inicializa essas páginas na RAM no primeiro uso. Essas páginas inicializadas são conhecidas como páginas de cache, seu tamanho total pode ser visto com o comando 'free' (coluna 'cache'). O problema é que o sistema operacional expira depois de algum tempo. Essa expiração é inevitável, não importa o quanto de RAM esteja disponível. É um processo de todo o sistema operacional baseado em recência de uso.

Existe um processo kswapd que é executado periodicamente e verifica se a página foi acessada desde a execução anterior. Se tiver, não faz nada. Se não, divide a idade por dois. Quando a idade chega a zero, essa página se torna candidata a despejo (Bem, isso nunca se torna zero ou provavelmente perto disso, eu não sei exatamente a matemática envolvida).

Esse mecanismo, arquivo mmap'ing, é usado para todos os arquivos no sistema. Os arquivos de log também são mapeados, portanto, quanto mais logs, mais RAM é usada pelas páginas de cache.

Há alguma dependência no uso de RAM: quanto maior o uso de RAM "ativa" (consulte `vmstat -s`), mais frequentemente o kswapd é executado e mais rápido é o processo de envelhecimento.

O MongoDB informa o tamanho real dos dados na RAM e a idade máxima desses dados. Assim, podemos ver que, no caso do gerenciamento de memória, mongodb depende muito do lado do sistema operacional e isso poderia causar alguns efeitos negativos, como expiração do cache e eliminação de dados da RAM, apenas porque o sistema operacional acha que os dados já devem ter expirado.

Replicação e alta disponibilidade

Um conjunto de réplicas no MongoDB é um grupo de processos mongod que mantém o mesmo conjunto de dados. Os conjuntos de réplicas fornecem redundância e alta disponibilidade e são a base para todas as implantações de produção. Essa seção apresenta a replicação no MongoDB, bem como os componentes e a arquitetura dos conjuntos de réplicas. Também fornece tutoriais para tarefas comuns relacionadas a conjuntos de réplicas.

Redundância e disponibilidade de dados

A replicação fornece redundância e aumenta a disponibilidade dos dados. Com várias cópias de dados em diferentes servidores de banco de dados, a replicação fornece um nível de tolerância a falhas contra a perda de um único servidor de banco de dados.

Em alguns casos, a replicação pode fornecer maior capacidade de leitura, pois os clientes podem enviar operações de leitura para servidores diferentes. Manter cópias de dados em diferentes data centers pode aumentar a localização e disponibilidade dos dados para aplicativos distribuídos. Você também pode manter cópias adicionais para fins dedicados, como recuperação de desastres, relatórios ou backup.

Replicação no MongoDB

O Replica Sets é um grupo de processos do mongod que mantém o mesmo conjunto de dados. Os conjuntos de réplicas fornecem redundância e alta disponibilidade. Quando o nó primário falha, o árbitro escolherá qual nó secundário será o novo primário. Quando o nó primário anterior retornar, o novo nó primário promove seu recovery.

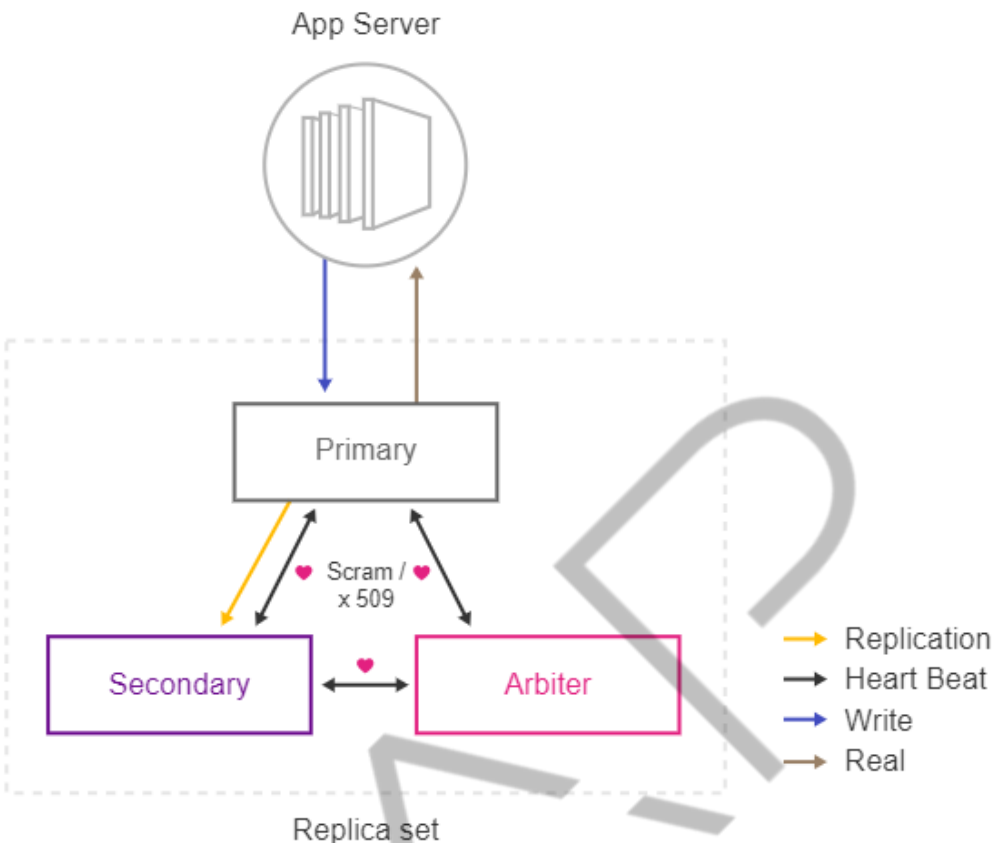


Figura 4.12 – MongoDB replica set
Fonte: MongoDB (2020)

Replica set define regras (rules) para escrita e para controlar onde os dados são escritos e lidos. Regras podem ser alteradas sem modificar a aplicação.

primary	Always read from the primary (default)
primaryPreferred	Always read from the primary, read from secondary if the primary is unavailable
secondary	Always read from a secondary
secondaryPreferred	Always read from a secondary, read from the primary if no secondary is available
nearest	Always read from the node with the lowest network latency

Figura 4.13 – Replica Set regras
Fonte: MongoDB (2020)

Um conjunto de réplicas é um grupo de instâncias mongod que mantém o mesmo conjunto de dados. Um conjunto de réplicas contém vários nós de suporte de dados e, opcionalmente, um nó de árbitro. Dos nós portadores de dados, um e apenas um membro é considerado o nó primário, enquanto os outros nós são considerados nós secundários.

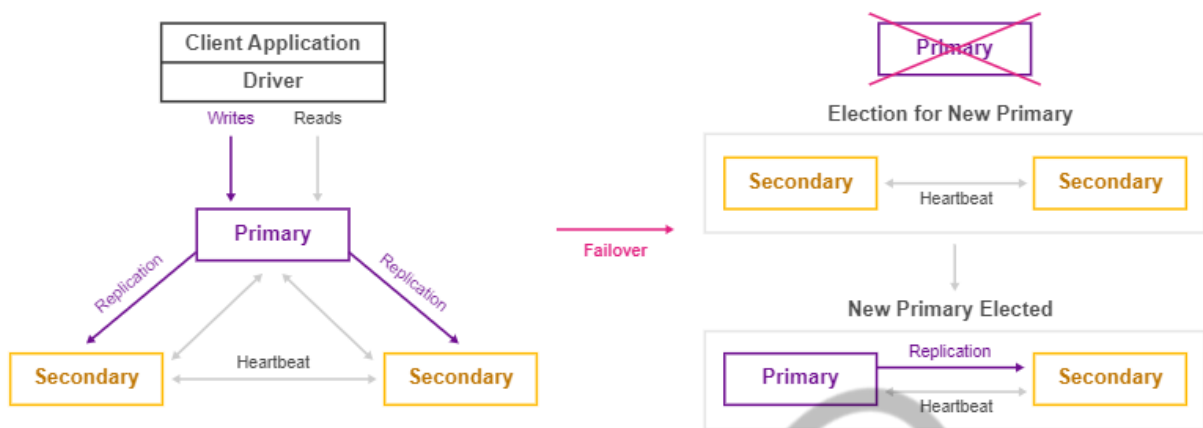


Figura 4.14 – Grupo de instâncias Mongod
Fonte: MongoDB (2020)

O nó primário recebe todas as operações de gravação. Um conjunto de réplicas pode ter apenas um primário capaz de confirmar gravações com preocupação de gravação {w: "majoritária"}; embora, em algumas circunstâncias, outra instância mongod possa temporariamente acreditar que também é primária. [1] O primário registra todas as alterações em seus conjuntos de dados em seu log de operação, ou seja, oplog. Para obter mais informações sobre a operação do nó primário, consulte Replica Set Primary.

Sharding

O MongoDB tem uma funcionalidade de autosharding que aumenta a escrita, permitindo scale-out. O MongoDB executa automaticamente sharding e quando os nós contêm uma quantidade diferente de dados, o MongoDB redistribui automaticamente os dados, para que a carga seja igualmente distribuída pelos nós. O mongod funciona como um roteador de consultas, uma interface entre a aplicação e o sharded cluster. O config servers possui parâmetros de configuração para o cluster e os metadados e não é um replica set. Replica Sets são para alta disponibilidade e leitura performática, não para escrita performática. Cada Partição/Sharding tem sua própria replica.

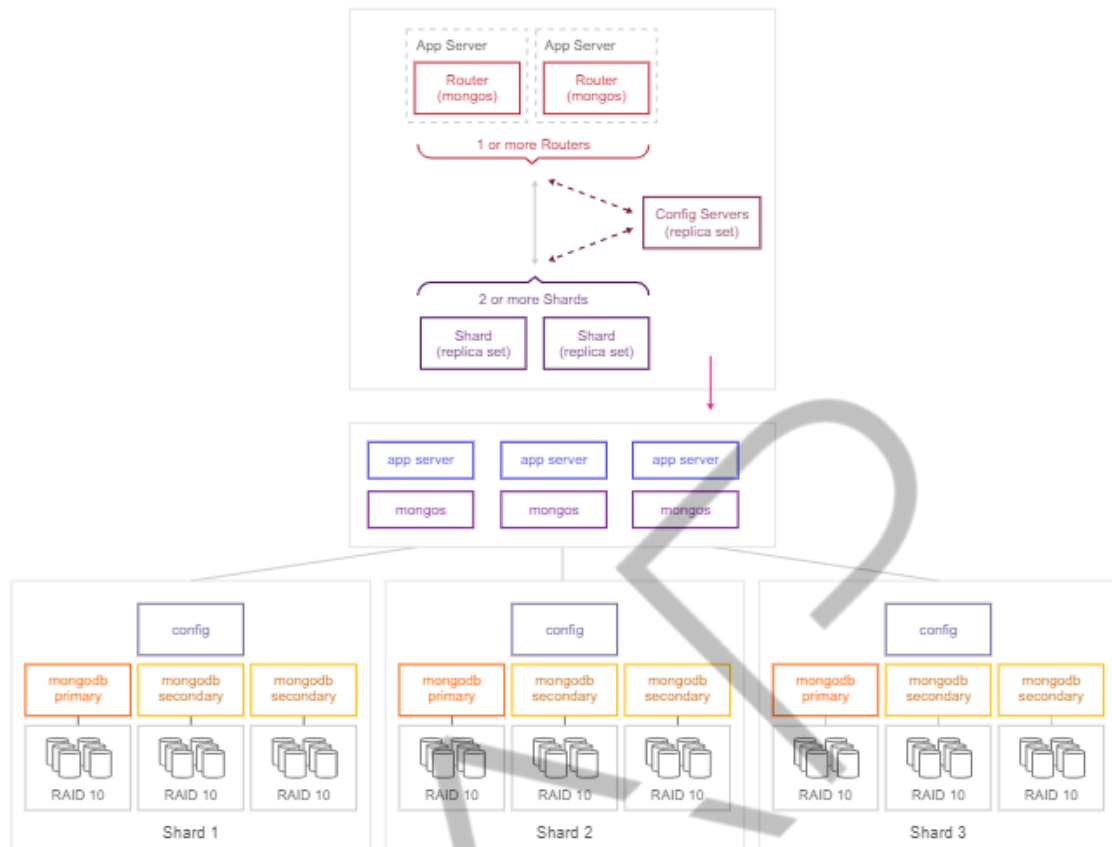


Figura 4.15 – Indexação Mongod
Fonte: MongoDB (2020)

MongoDB suporta a indexação de campos da mesma maneira que um banco de dados relacional; no entanto, não há junções entre coleções. Cada coleção pode ter até 64 índices. Ao criar uma coleção, um índice exclusivo é criado automaticamente para `_id`.

Documentos do MongoDB são armazenados em uma forma binária de JSON chamada formato BSON. O BSON suporta Boolean, float, string, número inteiro, data e tipos binários. Devido à estrutura do documento, o MongoDB é um banco de dados schema less sendo assim fácil adicionar novos campos num documento ou alterar a estrutura existente de um modelo.

4.12 Modelo de dados

Em termos gerais, o MongoDB segue uma mesma estrutura quando se trata de macro-objetos, porém, para que haja performance, outros objetos são referência de sua arquitetura:

SQL	MONGODB
DATABASE	DATABASE
TABLE	COLLECTION
ROW	DOCUMENT
INDEX	INDEX
JOIN	EMBEDDED DOCUMENT
FOREING KEY	REFERENCE
COLLUMN	KEY

Figura 4.16 – Modelo de dados
Fonte: MongoDB (2020)

Dentro da modelagem relacional, o ponto-chave são a não redundância e os relacionamentos, já no MongoDB isso seria um empecilho. Não há **JOINS**:

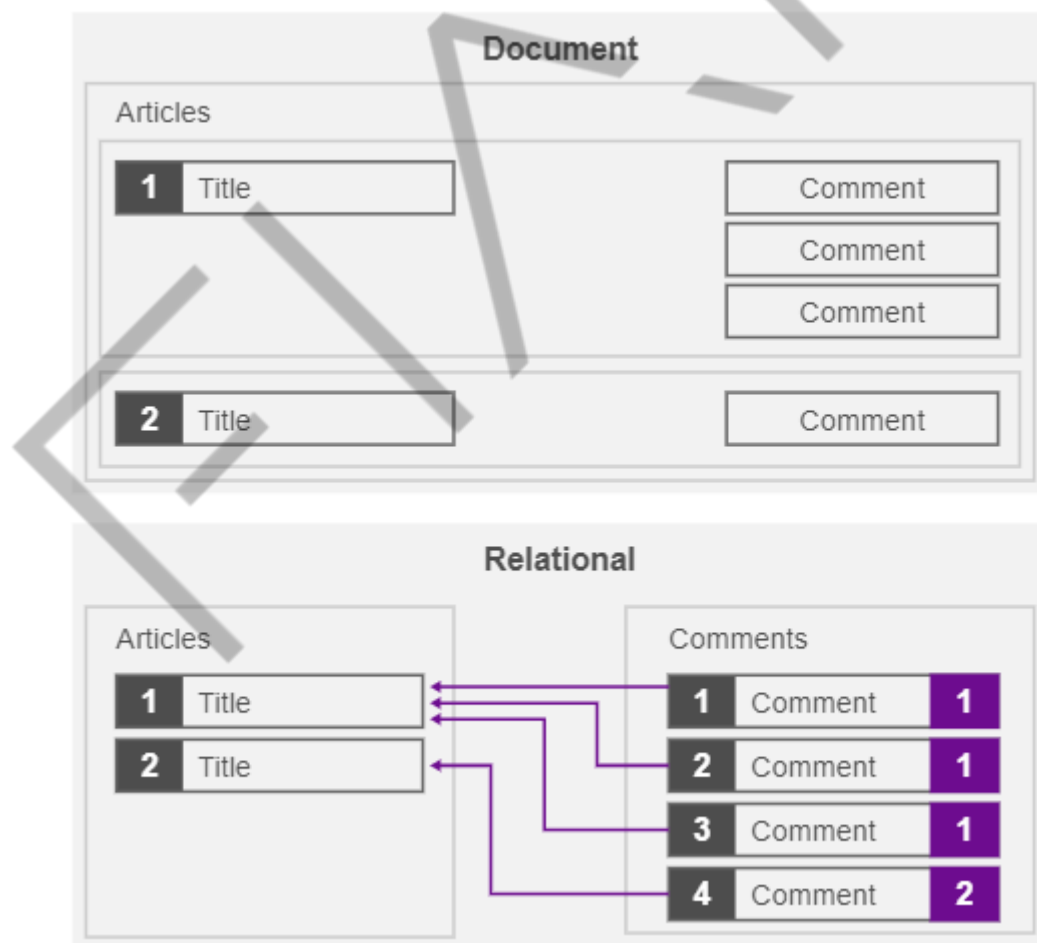


Figura 4.17 – Modelagem relacional
Fonte: MongoDB (2020)

O MongoDB não pode substituir bancos de dados relacionais, mas deve ser visto como uma alternativa a ele. O MongoDB pode ser instalado no Windows, Linux e MAC, portanto, é um banco de dados de plataforma cruzada. Ele não suporta junções, mas pode representar estruturas de dados hierárquicas avançadas. E o melhor recurso é o mais fácil: é facilmente escalável e pode proporcionar alto desempenho.

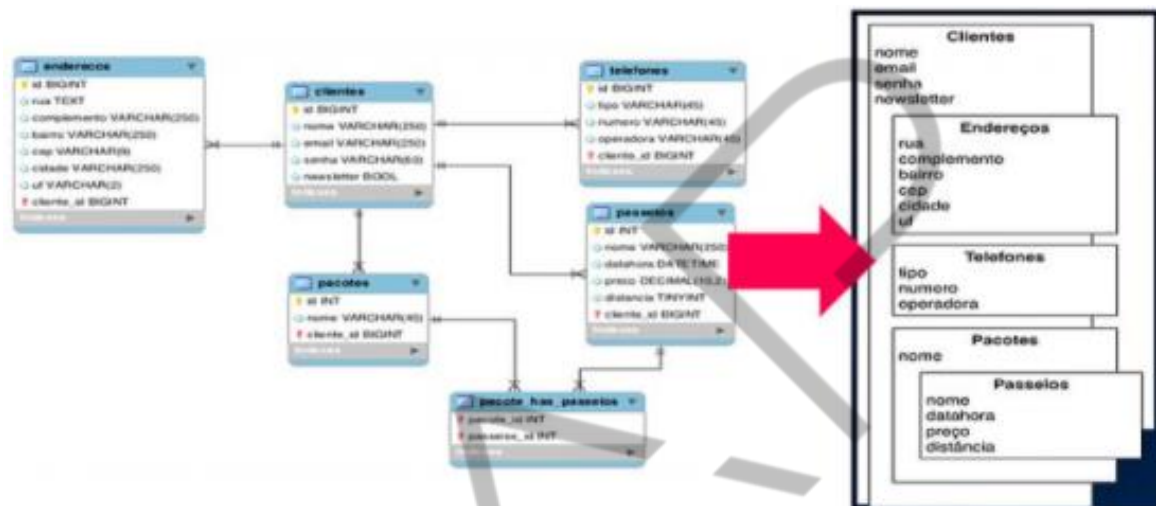


Figura 4.18 – MongoDB
Fonte: MongoDB (2020)

O banco de dados MongoDB consiste em um conjunto de bancos de dados no qual cada banco de dados contém várias coleções. O MongoDB é sem esquema, o que significa que cada coleção pode conter diferentes tipos de objetos. Todo objeto também é chamado de documento, que é representado como uma estrutura JSON (JavaScript Object Notation): uma lista de pares de valores-chave. O valor pode ser de três tipos: um valor primitivo, uma matriz de documentos ou novamente uma lista de pares de valores-chave.

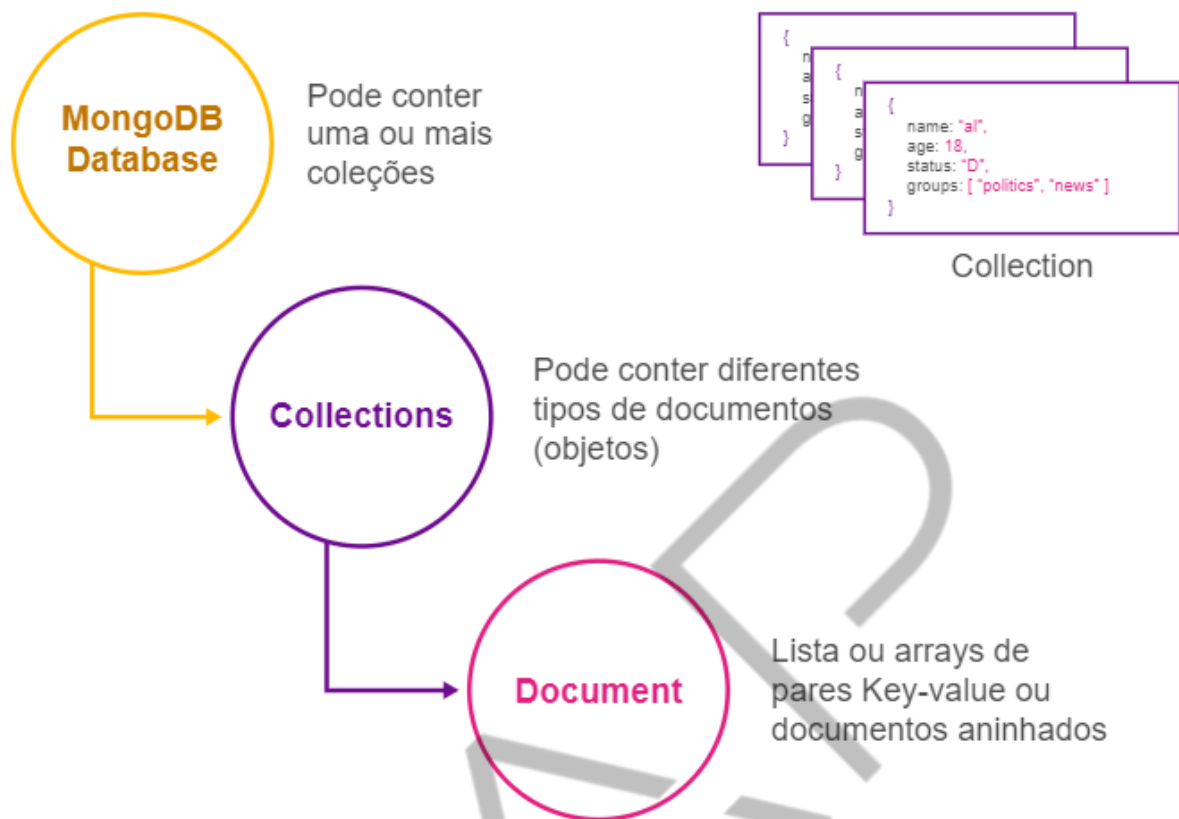


Figura 4.19 – MongoDB
Fonte: MongoDB (2020)

4.13 MongoDB Atlas

O Atlas é o núcleo do MongoDB Cloud unificando diferentes serviços de dados e totalmente gerenciado. Possui vários módulos com distintas funcionalidades. Atlas Auto-Scale é uma funcionalidade do Atlas para monitorar as métricas em tempo real e ajustar os recursos para processamento e armazenamento de dados no cluster, de acordo com as necessidades da carga de trabalho.

No Atlas Performance Advisor e no Data Explore podem-se obter proativamente dicas sobre como modelar seus dados para obter o melhor desempenho. As sugestões de esquema vêm dos metadados e logs do banco de dados para sinalizar anti-patterns comuns quando se trabalha com o modelo de dados do documento.

O Atlas Search é uma tecnologia de pesquisa para criar índices de pesquisa diretamente no Atlas e acessá-los com a mesma estrutura de agregação do MongoDB.

O Atlas Data Lake permite consultar mais rapidamente dados presentes no data lake consultando-os em qualquer formato no Amazon S3 usando a MongoDB Query Language (MQL).

O Atlas Online Archive move automaticamente os dados mais antigos para o Data Lake, preservando o acesso às consultas nas duas camadas com as consultas federadas.

O Realm Mobile Database, compatível com iOS e Android, permite aos aplicativos móveis acesso local aos dados e junto com o Realm Sync automatiza o processo de sincronização bidirecional com um cluster Atlas no backend. Possui um serviço GraphQL que facilita a exposição de uma API GraphQL para o MongoDB Atlas.

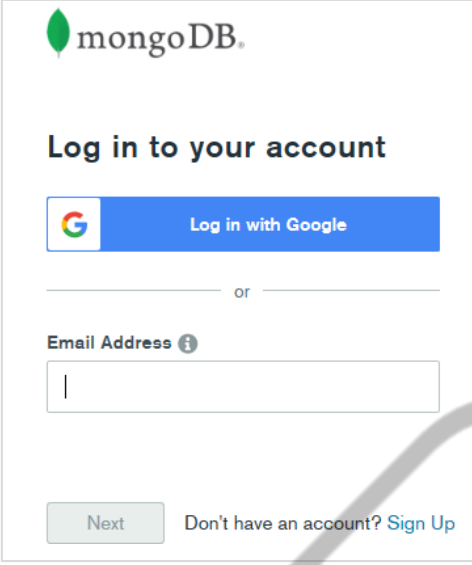
O MongoDB Charts é a melhor maneira de criar visualizações analíticas de dados e facilitar o acesso e o compartilhamento dessas visualizações. Após essa rápida descrição dos componentes do MongoDB Cloud, vamos acessar a plataforma e construir alguns charts!

4.14 Construir o ambiente

Todo ambiente no sandbox do MongoDB Atlas na nuvem será preparado bem como as conexões necessárias.

4.15 Acessar o ambiente

O MongoDB permite criarmos um ambiente na nuvem free até 500Mb de dados. Para isso, precisamos criar uma conta MongoDB-as-a-service, acessando em <https://cloud.mongodb.com>.

The image shows the MongoDB login interface. At the top is the MongoDB logo. Below it is the heading "Log in to your account". There is a "Log in with Google" button with the Google logo. Below this is a horizontal line with "or" in the center. Underneath is a label "Email Address" with an information icon, followed by a text input field. At the bottom, there is a "Next" button and a link "Don't have an account? Sign Up".

mongoDB.

Log in to your account

Log in with Google

or

Email Address ⓘ

Next Don't have an account? [Sign Up](#)

Figura 4.20 – Criação da conta
Fonte: Elaborado pelo autor (2020)

4.16 Criar um cluster

Após criar o usuário e acessar a plataforma, o primeiro passo é criar um cluster para nossos dados. Para isso, clique em 'Projects', no canto superior esquerdo. Nomeie seu projeto e clique em 'Next'.

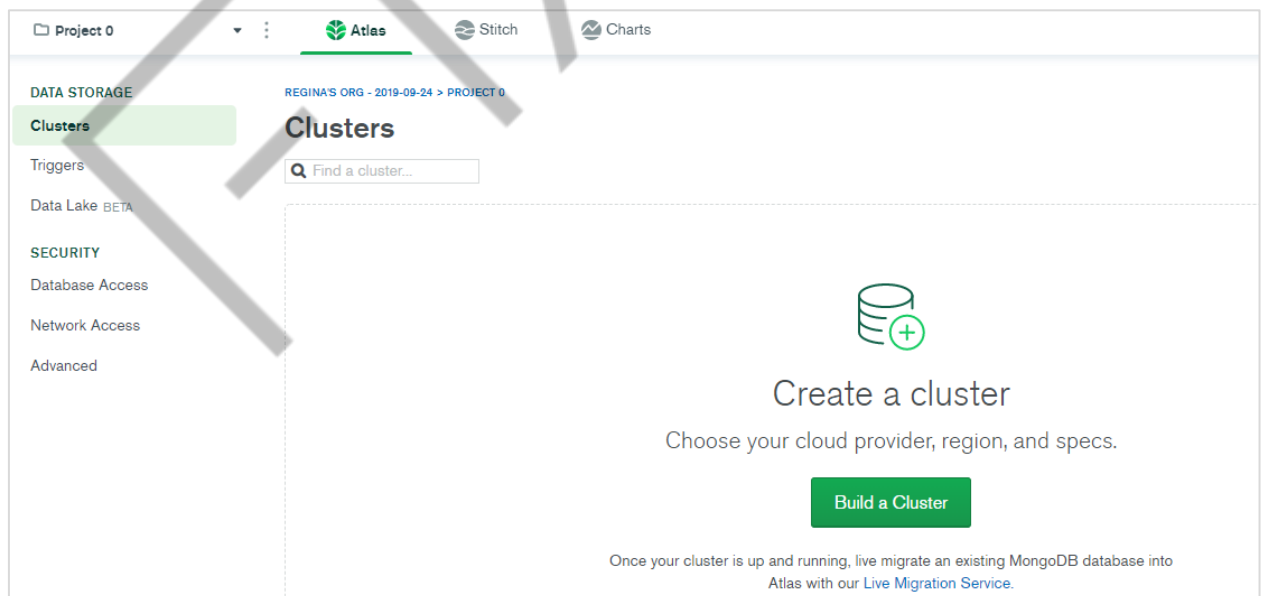


Figura 4.21 – Criar um cluster
Fonte: Elaborado pelo autor (2020)

Agora vamos escolher um provedor para nosso cluster: AWS, Google Cloud Plataforma ou Azure. Escolha a opção de cluster 'free'. Um cluster com nome Cluster0 será criado sem custo após escolher o provedor.

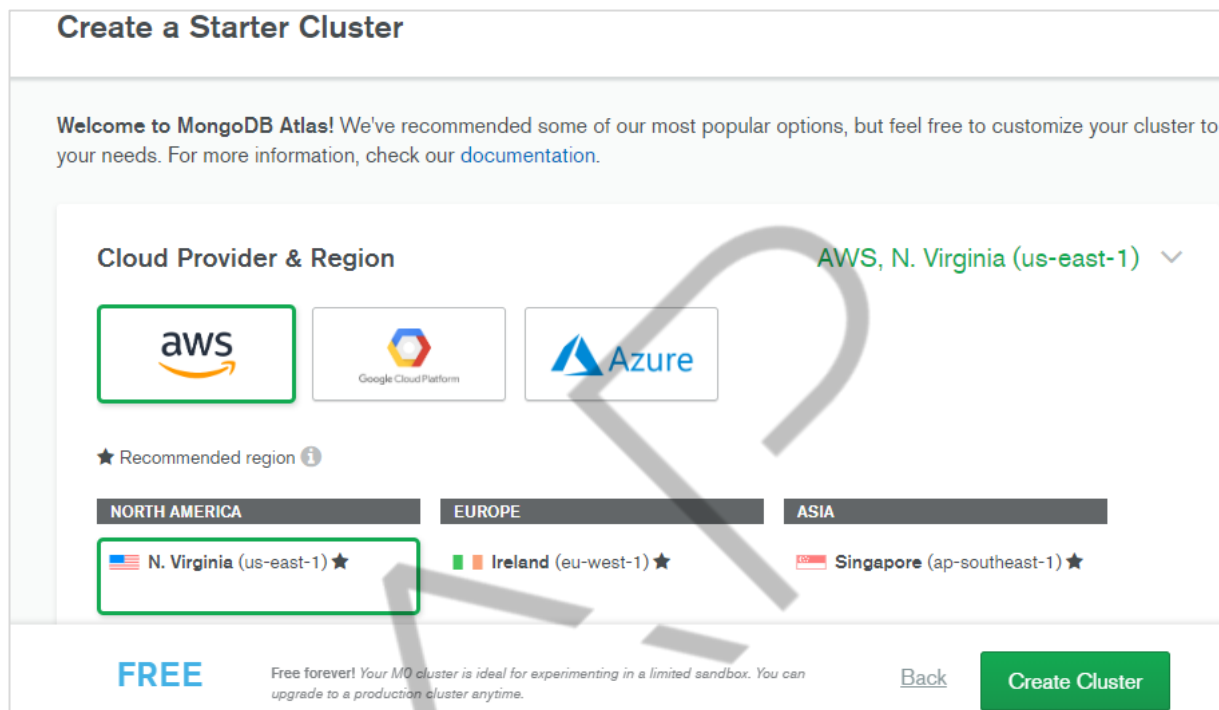


Figura 4.22 – Escolher provedor do cluster
Fonte: Elaborado pelo autor (2020)

Essa criação pode demorar uns minutinhos. Cada cluster será construído com os Replica Sets: um nó primário e dois nós secundários. Ao final da criação, todo ambiente de gerenciamento será disponibilizado.

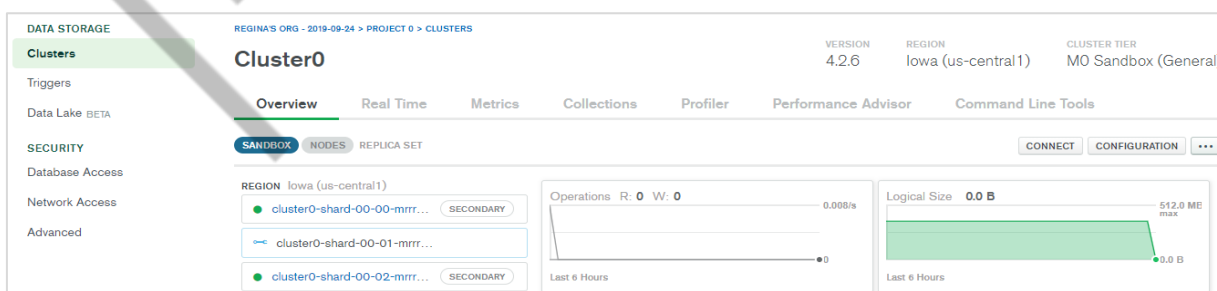


Figura 4.23 – Cluster provisionado
Fonte: Elaborado pelo autor (2020)

4.17 Segurança de conexão de configuração

Para acessar o cluster com ferramentas client como o Compass, precisaremos garantir a segurança. Então, vamos criar um usuário no nosso cluster. Para isso,

criamos um acesso na rede e depois o usuário. No menu do lado esquerdo da plataforma, clicaremos em Network Access e Add IP Address.

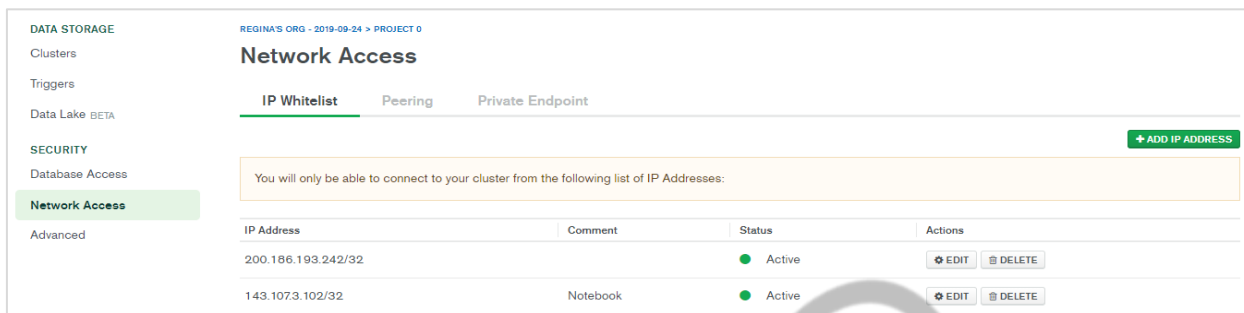


Figura 4.24 – Criação da conexão
Fonte: Elaborado pelo autor (2020)

No menu do lado esquerdo da plataforma, clicaremos em Database Access e Add New Database User.

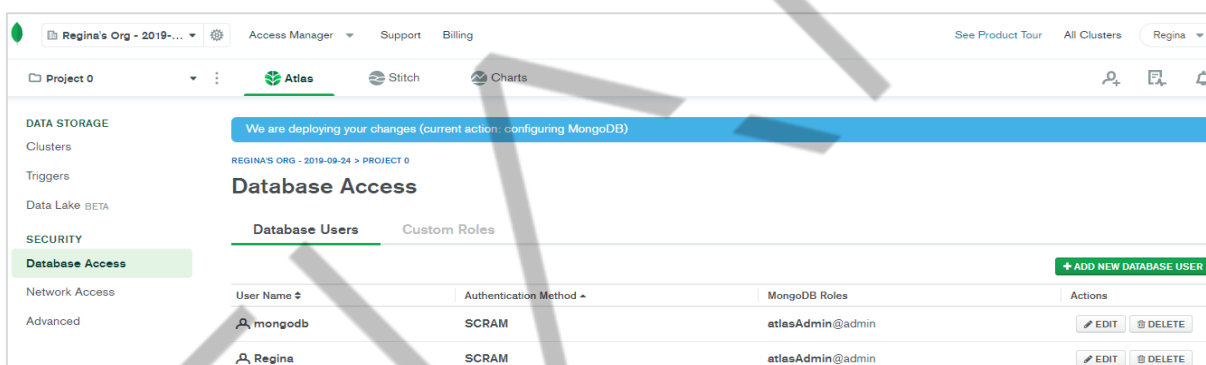


Figura 4.25 – Criar um usuário para conexão
Fonte: Elaborado pelo autor (2020)

Neste passo, observar os privilégios do usuário e a senha (password).

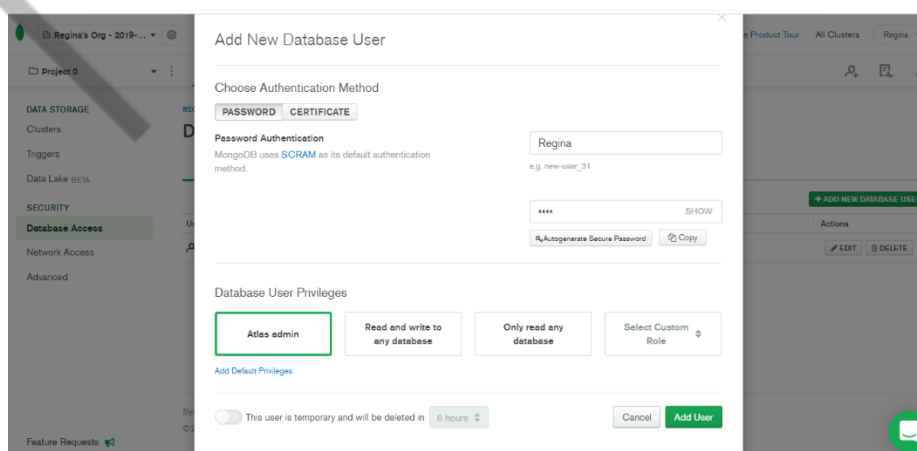


Figura 4.26 – Acesso com MongoDB Compass
Fonte: Elaborado pelo autor (2020)

Clicar em CONNECT para configurar a segurança da conexão e assinalar o MongoDB Compass como ferramenta a ser utilizada e copiar a string para ser utilizada no Compass.

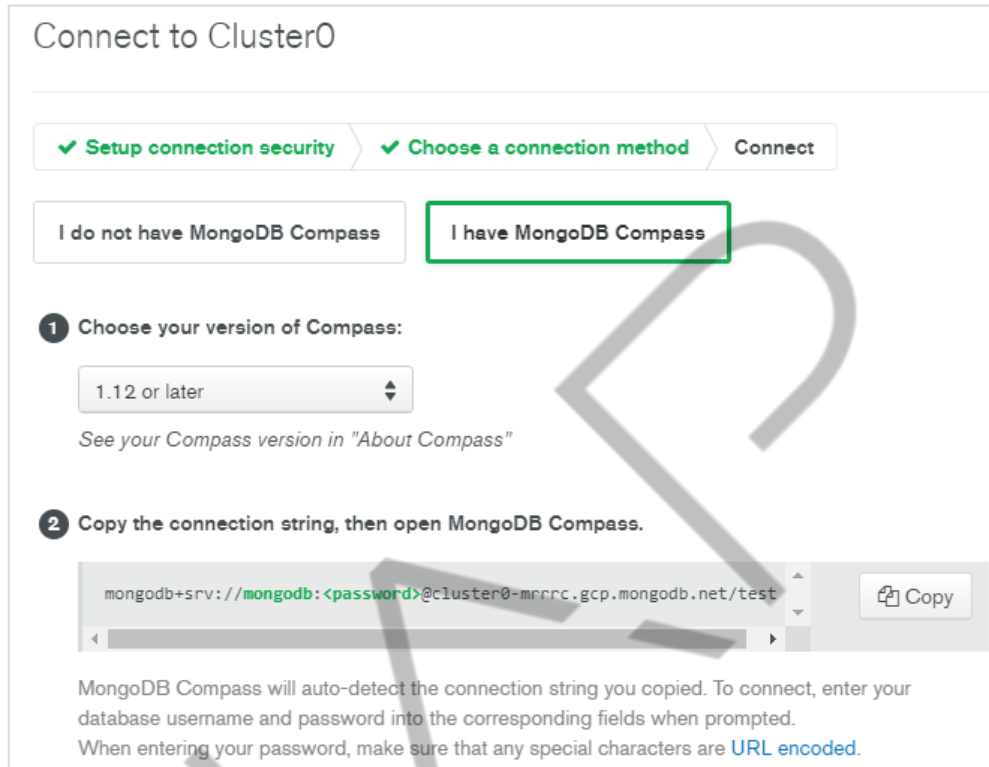


Figura 4.27 – String conexão
Fonte: Elaborado pelo autor (2020)

4.18 Acessar cluster com MongoDB Compass

Compass é a interface gráfica (GUI) para o MongoDB, já instalada no capítulo anterior. Ao acioná-la na primeira tela já solicita o string de conexão do usuário. Utilizaremos a string copiada no passo anterior, porém observando o usuário e senha.

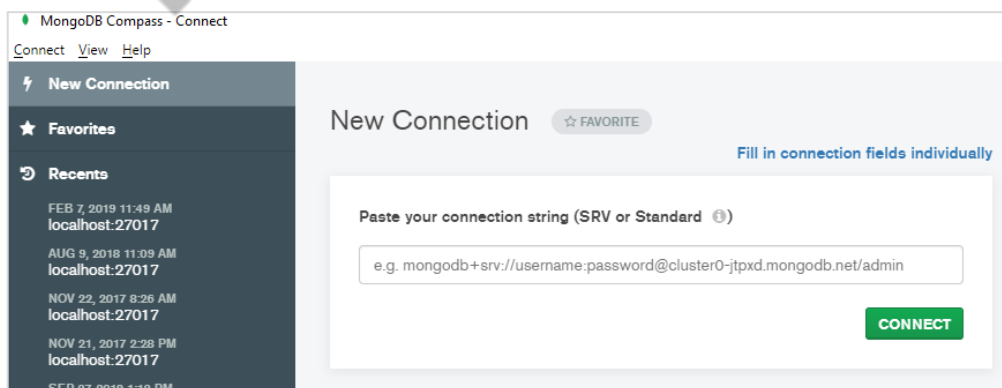


Figura 4.28 – MongoDB Compass
Fonte: Elaborado pelo autor (2020)

Se a conexão for bem-sucedida, podemos observar o cluster criado com seu Replica Set e shards associados.

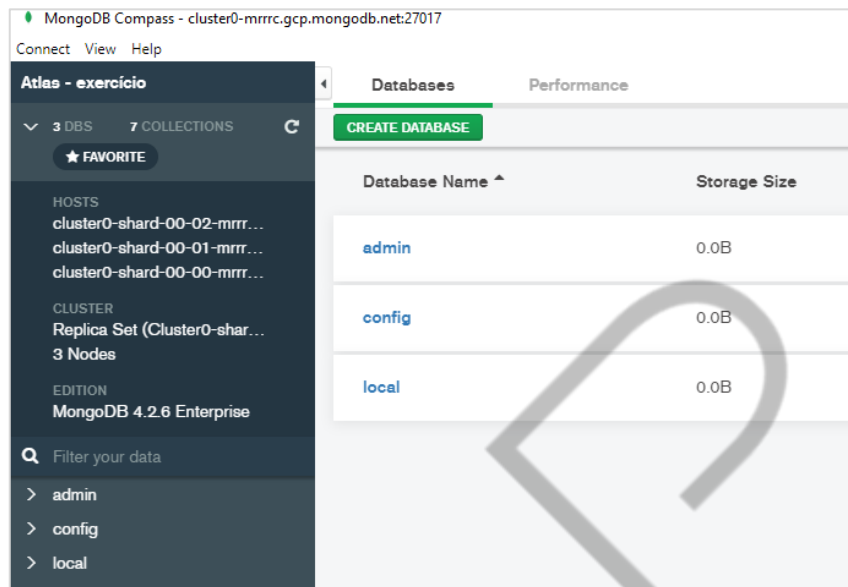


Figura 4.29 – MongoDB Compass acessando Atlas
Fonte: Elaborado pelo autor (2020)

Tudo pronto para criarmos a base de dados e utilizarmos os recursos tanto no MongoDB Atlas como no MongoDB Compass.

4.19 On-Premises

De forma a ser possível reproduzir todos os exemplos presentes neste capítulo, faz-se necessário providenciar uma infraestrutura para a utilização do MongoDB. Ao acessar o site <www.mongodb.com>, podemos observar a possibilidade de se utilizar na *cloud* denominado MongoDB Atlas com opção de uso nos provedores Google Cloud Platform, AWS ou Microsoft Azure.

Entretanto, você pode optar por uma instalação local - on-premises, clicando no botão “Try Free” e, posteriormente, em “Server”, para baixar o instalador do *MongoDB Community Server* <<https://www.mongodb.com/try/download/community>>. Existem versões para Windows, MacOS e principais distribuições Linux. Escolha versão, plataforma e Package (nesse caso, 'msi'). Faça o download e instale em seguida.

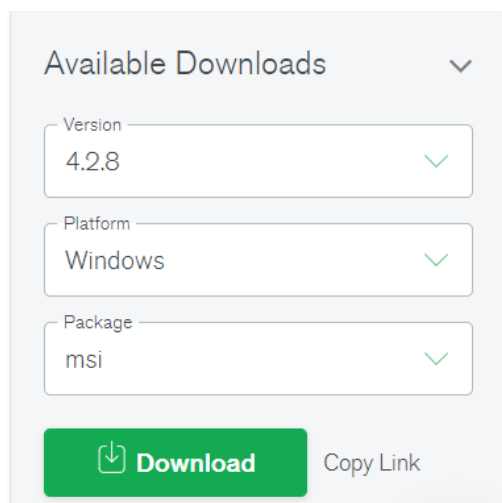


Figura 4.30 – Tela para download do instalador do MongoDB Community Server
Fonte: MongoDB (2020)

Após o download na pasta de sua preferência, e instalado o MongoDB com todas as opções default, um serviço é criado MongoDB Database Server. Pode-se verificar acessando o Gerenciador de Tarefas no caso da plataforma Windows.

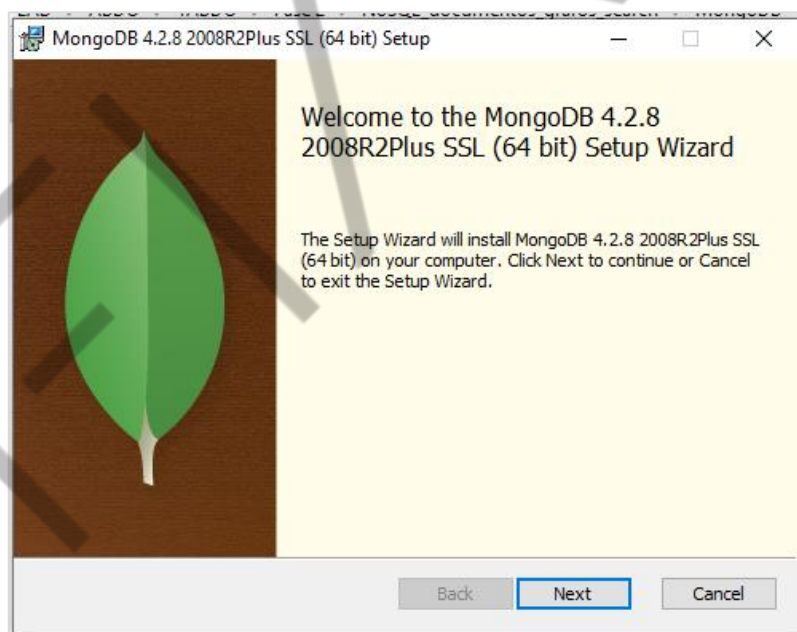


Figura 4.31 – Instalador MongoDB Windows
Fonte: Elaborado pelo autor (2020)

É igualmente recomendável a instalação do MongoDB Compass para interface gráfica com o banco de dados (marcar a caixa 'Install MongoDB Compass durante a instalação do MongoDB server'). Assim como o instalador do MongoDB Community Server, existem versões para os principais sistemas operacionais do mercado.



Figura 4.32 – Tela para download do instalador do MongoDB Compass
Fonte: MongoDB (2020)

No final da instalação, o Compass é inicializado automaticamente e sua conexão pode ser realizada com o banco local ou na nuvem.

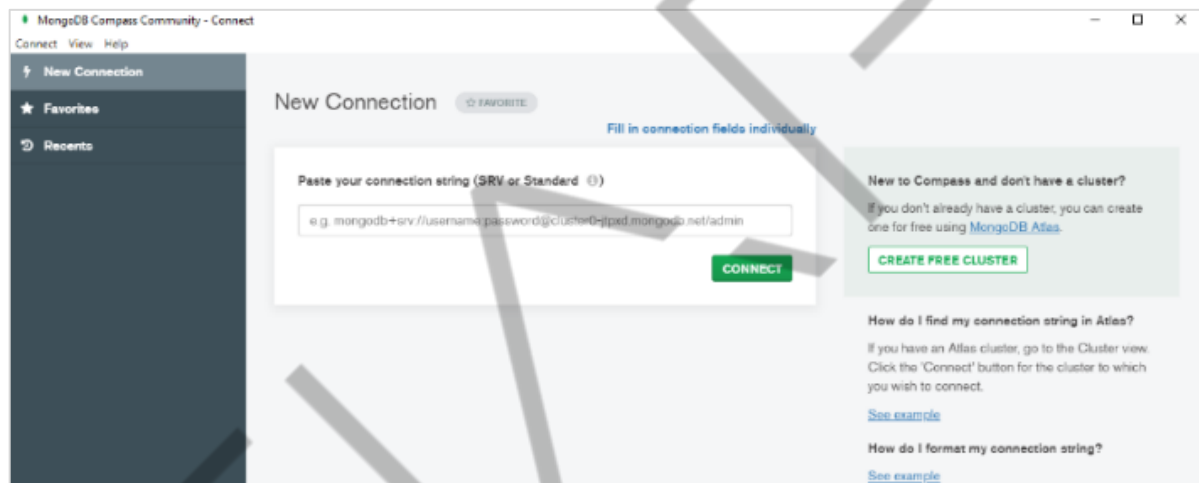


Figura 4.33 – MongoDB Compass
Fonte: Elaborado pelo autor (2020)

A instalação de ambos é bem simples e intuitiva, e recomenda-se manter a instalação em suas configurações padrões. O objetivo é instalar os softwares servidor e cliente no mesmo equipamento, assim sendo, quando for fazer uso do MongoDB Compass e ele solicitar informações do servidor, basta informar o IP 127.0.0.1 (localhost) na porta padrão de servidor do MongoDB (27017). Digitar o caminho da conexão, neste caso localhost: `mongodb://127.0.0.1:27017`. Pronto, está criada a conexão.

Para realizarmos os comandos de linha diretamente no banco de dados, podemos acionar o mongo no diretório bin da instalação realizada. Para a versão 4.2.8 instalada acima para plataforma Windows fica no diretório “C:\Program Files\MongoDB\Server\4.2\bin”. Pode ser necessário criar uma pasta denominada

'data' em 'C: ' e, dentro dela, uma pasta 'db', onde serão armazenadas as informações das collections.

```
C:\Program Files\MongoDB\Server\4.2\bin>mongo
MongoDB shell version v4.2.8
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("0258d211-545f-47b4-b75c-9ac3d152e8bd") }
MongoDB server version: 4.2.8
Server has startup warnings:
2020-07-16T18:55:11.502-0300 I CONTROL [initandlisten]
2020-07-16T18:55:11.502-0300 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-07-16T18:55:11.502-0300 I CONTROL [initandlisten] ** Read and write access to data and configuration is
unrestricted.
2020-07-16T18:55:11.503-0300 I CONTROL [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
```

Figura 4.34 – MongoDB interface comando de linha
Fonte: Elaborado pelo autor (2020)

4.20 Criando um banco de dados

O MongoDB armazena dados na forma de coleções. Cada coleção contém documentos que possuem um identificador exclusivo. Os documentos são agrupados em coleções e cada documento em uma coleção pode ter um número arbitrário de campos.

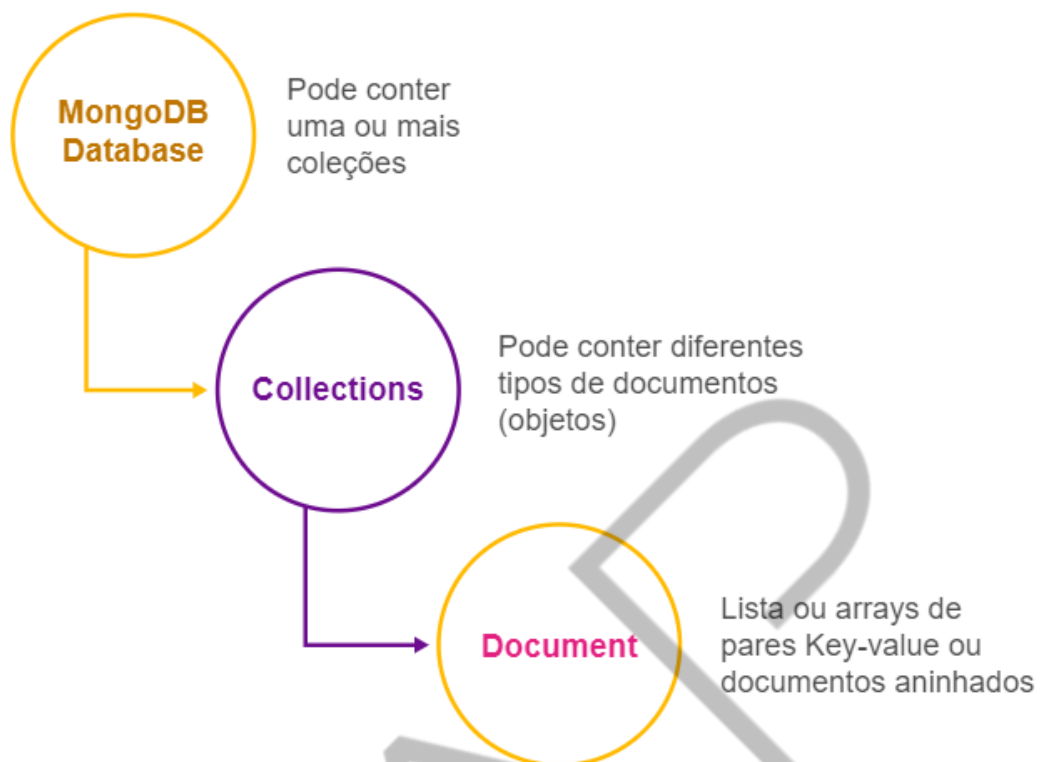


Figura 4.35 – MongoDB
Fonte: Adaptado por FIAP (2020)

Ao contrário dos bancos de dados relacionais, não existe um comando `create database`. O banco é fisicamente criado no momento em que uma coleção é criada (MONGODB, 2020). O banco de dados default é o `test`. Para mudarmos de um banco para outro, usamos o comando `use`.

```
use (<db>) ;
```

Código-fonte 4.1 – Sintaxe do comando `use`
Fonte: MongoDB (2020)

Em que:

`db` – indica o nome do banco de dados a ser usado.

A variável `shell db` é alterada para o nome do banco de dados que será usado.

A figura mostra o uso do comando `use`.

```
> use test
switched to db test
> db
test
> use biblioteca
switched to db biblioteca
> db
biblioteca
```

Figura 4.36 – Exemplo de uso dos comandos `use`

Fonte: Elaborado pelo autor (2020)

Na imagem, usamos o comando `use test` e, em seguida, exibimos o conteúdo da variável `shell db`, nesse primeiro caso, ela está com o valor **test**. A seguir, usamos o comando `use biblioteca` e, novamente, exibimos o conteúdo da variável `shell db`, dessa vez, ela está com o valor **biblioteca**.

4.21 Coleção

Uma coleção ou *collection* é o local em que os documentos são armazenados no MongoDB. Uma coleção pode ser comparada com as tabelas de um banco de dados relacional, mas sua estrutura é muito menos rígida do que em um RDBMS.

Uma coleção pertence a um único banco de dados MongoDB e não possui um esquema. Cada documento, em uma coleção, pode ter campos diferentes. Por uma questão de praticidade, procuramos criar coleções com documentos semelhantes. (MONGODB, 2018). As coleções podem ser criadas no momento de inserção de um documento que a referencie ou por meio do comando `createCollection` (MONGODB, 2018).

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

Código-fonte 4.2 – Sintaxe do comando `createCollection`

Fonte: MongoDB (2020)

Em que:

Name – indica o nome da coleção a ser criada.

Options – podem assumir os seguintes valores:

capped – coleção ordenada, circular, de tamanho fixo. Default “false”.

autoIndexID – cria índice nos campos `_id`. Default “false”.

size – define o tamanho, em bytes, de uma coleção capped.

max – define o número máximo de documentos de uma coleção capped.

O comando “`creatCollection`” cria uma coleção denominada **biblioteca**.

```
db.createCollection("biblioteca")
```

Código-fonte 4.3 – Exemplo do uso do comando `createCollection`

Fonte: Elaborado pelo autor (2020)

A figura “Resultado da execução do comando `createCollection`” mostra o resultado da execução do comando:

```
> db.createCollection("biblioteca")
{ "ok" : 1 }
```

Figura 4.37 – Resultado da execução do comando `createCollection`

Fonte: Elaborado pelo autor (2020)

Podemos ver que a execução foi bem-sucedida, porque o comando retornou a chave valor { "ok" : 1 }. Os nomes de coleções devem obedecer aos seguintes critérios:

- O nome de uma coleção deve começar por uma letra ou sublinhado (“_”).
- Pode conter números.
- Não pode utilizar o caractere cifrão (“\$”).
- Não pode exceder 128 caracteres.

É interessante notar que é possível criar grupos nomeados usando um ponto (“.”). Esses grupos são nomeados de *namespace collection*. Podemos, por exemplo, criar uma coleção para os livros de arte e outra para os romances:

- livros.romance
- livros.artes
- O comando `show collections;` pode ser usado para exibir todas as coleções do banco de dados corrente.

```
show collections;
```

Código-fonte 4.4 – Exemplo do uso do comando `show collections`

Fonte: MongoDB (2020)

- A imagem mostra o uso do comando `show collection;`, seguido da criação de uma coleção e da reexecução do comando `show collection;`.

```
> show collections;
> db.createCollection("biblioteca");
{ "ok" : 1 }
> show collections;
biblioteca
```

Figura 4.38 – Exemplo de uso dos comandos `show collections` e `createCollection`
Fonte: Elaborado pelo autor (2020)

Nessa figura, utilizamos o comando `show collections`. Como não havia nenhuma coleção criada, nenhum valor foi retornado, depois, criamos a coleção biblioteca, podemos ver que a execução foi bem sucedida porque o comando retornou a chave valor `{ "ok" : 1 }`. Em seguida reexecutamos o comando `show collections`, dessa vez, o comando retornou o nome da coleção que acabamos de criar: biblioteca.

4.22 Documento

Em um banco MongoDB, um documento, ou *document*, é o equivalente a uma linha ou tupla, em uma tabela de um banco relacional. O documento é representado por um conjunto de pares de chave-valor (MONGODB, 2018). O exemplo “documentos com dois campos” mostra um documento com um nome e uma idade.

```
{
  Nome: 'Rita',
  Idade: 26
}
```

Código-fonte 4.5 – Exemplo de documentos com dois campos
Fonte: Elaborado pelo autor (2020)

Note que o documento é aberto com uma chave e é encerrado por outra, tudo que está entre as duas chaves faz parte desse documento. Existem dois pares de chave-valor, observe que cada par é separado por uma vírgula. O primeiro campo tem o nome de **Nome** e tem o valor **'Rita'**, perceba que entre a chave e o valor existe o símbolo de dois pontos. O segundo campo tem o nome de **Idade** e tem o valor **26**.

Documentos na mesma coleção não precisam ter a mesma estrutura ou conjunto de campos. Documentos de uma mesma coleção podem conter diferentes tipos de dados (MONGODB, 2018).

4.23 Tipos de dados

O MongoDB possui suporte aos principais tipos de dados primitivos utilizados em qualquer banco de dados (MONGODB, 2018). Vejamos uma lista com os principais tipos de dados:

- 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 (*true* ou *false*).
- 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.
- Arrays: este tipo é usado para armazenar *arrays*, listas ou múltiplos valores dentro de uma chave (*key*).
- Timestamp: *timestamp*. 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.
- 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* por meio 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.

Dentre os diversos tipos de dados, os mais utilizados são os:

- Numéricos;
- String (use " ou "" para identificá-los);
- Date;
- Boolean (*true* ou *false*);
- Array.

Como você já deve ter percebido, o banco de dados MongoDB pode conter uma ou mais coleções; uma coleção pode conter diferentes tipos de documentos e um documento pode conter um conjunto de chave-valor, *arrays* e documentos aninhados.

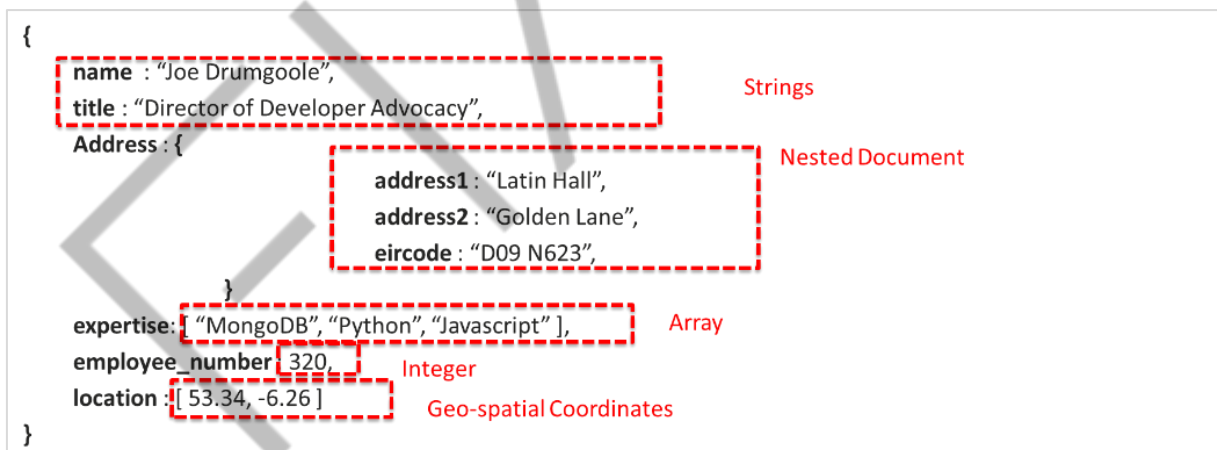


Figura 4.39 – MongoDB - Exemplo de um documento e tipos de dados
Fonte: Adaptado por FIAP (2020)

O nome das variáveis `shell` no MongoDB deve iniciar por uma letra minúscula [a-z], os caracteres seguintes podem ser letras minúsculas, maiúsculas ou números [a-zA-Z0-9] (MONGODB, 2018). É possível visualizar o conteúdo de uma variável, simplesmente digitando o nome da variável no terminal (MONGODB, 2018). A figura “Exibindo o conteúdo da constante db” mostra um exemplo de uso.

```
connecting to: test
> db
test
```

Figura 4.40 – Exibindo o conteúdo da constante `db`
Fonte: Elaborado pelo autor (2020)

4.24 Operações CRUD

As quatro operações básicas em banco de dados são:

- Incluir dados (*c**reate*).
- Consultar dados (*r**ead*).
- Alterar dados (*u**ppdate*).
- Apagar dados (*d**elete*).

O acrônimo das quatro palavras em inglês (*create*, *read*, *update* e *delete*) forma a palavra CRUD. Convencionou-se que o termo CRUD indica essas quatro operações. A estrutura básica de um comando no banco de dados MongoDB é:

```
database.coleção.função()
```

Código-fonte 4.6 – Estrutura básica de um comando no banco de dados MongoDB
Fonte: MongoDB (2020)

No qual:

Database – indica o banco de dados. A variável `db` contém o nome do banco de dados.

Coleção – indica o nome da coleção.

Função – indica a função que será aplicada à coleção.

Os comandos usados pelas operações CRUD, normalmente, começarão por `db`, indicando que as operações serão realizadas no banco corrente (MONGODB, 2018).

4.25 Incluindo dados

Adicionamos **documentos** a uma **coleção**, por meio de uma operação de inclusão.

IMPORTANTE: Caso a coleção ainda não tenha sido criada, a operação de inclusão a criará.

Há várias formas de incluirmos dados em uma coleção. Nesse primeiro momento, nos concentraremos na função `insert`.

```
db.coleção.insert(  
  <documento ou array de documentos>,  
  {  
    writeConcern: <documento>,  
    ordered: <boolean>  
  }  
)
```

Código-fonte 4.7 – Sintaxe do comando insert
Fonte: MongoDB (2020)

Na qual:

- Documento ou array de documentos – indica o documento, ou array de documentos, a ser inserido em uma coleção.
- writeConcern – define se a persistência dos dados será acknowledged (w) ou unacknowledged (j).
- ordered – insere os documentos na sequência dos que estão no array. Se um erro ocorrer, os demais documentos do array não serão inseridos.

Vamos a um exemplo de uso. Imagine que você queira incluir dados de um funcionário em uma coleção denominada “emp”. Os dados do funcionário são:

Nome – Bianka

Idade – 22

Cargo – Desenvolvedor

Salário – 12.000,00

Perceba que são quatro pares de chave-valor. A chave “Nome” tem o valor “Bianka”, a chave “Idade” tem o valor “22”, a chave “Cargo” tem o valor “Gerente” e a chave “Salario” tem o valor “12.000,00”. Vamos converter esses dados para uma estrutura de chave valor mais apropriada:

```
{  
  
  Nome: “Bianka”,
```



```
Idade: 22,  
Cargo: 'Desenvolvedor',  
Salario: 12000.00  
}
```

Perceba que os campos que contêm texto são indicados por aspas simples ou aspas duplas, qualquer uma delas é aceita. Isso indica, para o banco de dados, que esse campo conterá um texto. Os campos numéricos não contêm aspas. As vírgulas são usadas para separar os pares de chave-valor. Os valores decimais são indicados por um ponto. Vamos usar a sintaxe para incluir os dados na coleção **emp**.

```
db.emp.insert({ Nome: "Bianka",  
               Idade: 22,  
               Cargo: 'Desenvolvedor',  
               Salario: 12000.00  
})
```

Código-fonte 4.8 – Exemplo de inclusão de dados na coleção **emp**

Fonte: Elaborado pelo autor (2020)

A figura mostra o resultado da operação:

```
> db.emp.insert({ Nome: "Bianka", Idade: 22, Cargo: 'Desenvolvedor', Salario: 12000.00 })  
writeResult({ "nInserted" : 1 })  
> _
```

Figura 4.41 – Resultado da operação de inclusão na coleção **emp**

Fonte: Elaborado pelo autor (2018)

Note que o resultado da operação foi a mensagem de `writeResult({ "nInserted" : 1 })`. Isso indica que o comando foi bem-sucedido e que um documento foi incluído na coleção **emp**. Vamos a outro exemplo. Dessa vez, os dados são os seguintes:

Nome – Neusa

Idade – 23

Cargo – Desenvolvedor

Salário – 12.830,26

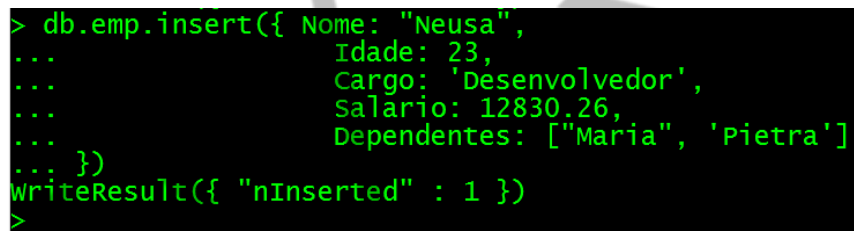
Dependentes – Maria, Pietra

Note que acrescentamos a chave “Dependentes”, que possui dois valores: “Maria” e “Pietra”. Como essa chave possui dois valores, usaremos um *array* para representá-los. Lembrando que campos do tipo texto podem ser indicados usando aspas simples ou aspas duplas.

```
db.emp.insert(
  {
    Nome: "Neusa",
    Idade: 23,
    Cargo: 'Desenvolvedor',
    Salario: 12830.26,
    Dependentes: ["Maria", 'Pietra']
  }
)
```

Código-fonte 4.9 – Exemplo de inclusão de dados com *array* na coleção **emp**
Fonte: Elaborado pelo autor (2020)

A figura “resultado da operação de inclusão com *array* na coleção **emp**” mostra o resultado da operação.



```
> db.emp.insert({ Nome: "Neusa",
...               Idade: 23,
...               Cargo: 'Desenvolvedor',
...               Salario: 12830.26,
...               Dependentes: ["Maria", 'Pietra']
... })
WriteResult({ "nInserted" : 1 })
>
```

Figura 4.42 – Resultado da operação de inclusão com *array* na coleção **emp**
Fonte: Elaborado pelo autor (2020)

Vamos a mais um exemplo um pouco mais complexo. Dessa vez, vamos incluir informações sobre o departamento em que o empregado trabalha.

Os dados do funcionário são:

Nome – Thais

Idade – 19

Cargo – Analista

Salario – 14.530,77

Dependentes – Angelina, Dora, Hugo

Os dados do departamento são:

Nome – Pesquisa

Local – São Paulo

Vamos tratar os dados do departamento como um documento dentro do documento.

```
db.emp.insert(
  { Nome: "Thais",
    Idade: 19,
    Cargo: "Analista",
    Salario: 14530.77,
    Dependentes: ["Angela", "Dora", "Hugo"],
    Departamento: { Nome: "Pesquisa",
                     Local: "São Paulo"
                   }
  }
)
```

Código-fonte 4.10 – Exemplo de inclusão de dados com a junção de outro documento na coleção **emp**

Fonte: Elaborado pelo autor (2020)

Note que, ao contrário dos bancos de dados relacionais, procuramos evitar a junção de duas coleções. Sempre que possível, já efetuamos a operação de junção durante a inclusão dos dados. A imagem mostra o resultado da operação.

```
> db.emp.insert({ Nome: "Thais",
...               Idade: 19,
...               Cargo: "Analista",
...               Salario: 14530.77,
...               Dependentes: ["Angela", "Dora", "Hugo"],
...               Departamento: { Nome: "Pesquisa",
...                               Local: "São Paulo"
...                             }
...             })
WriteResult({ "nInserted" : 1 })
>
```

Figura 4.43 – Resultado da operação de inclusão com a junção de outro documento na coleção **emp**

Fonte: Elaborado pelo autor (2020)

Podemos ter vários documentos dentro do mesmo documento. Também podemos ter vetores de documentos. No documento do funcionário, vamos incluir as informações sobre as suas promoções dentro da empresa.

Os dados do funcionário são:

Nome – Fátima

Idade – 29

Cargo – Analista

Salario – 12.345,67

Dependentes – Gohan

Os dados do departamento são:

Nome – Vendas

Local – Campinas

O histórico de promoções é:

Ano – 2001

Cargo – "Estagiário"

Valor – 180,00

Ano – 2002

Cargo - "Desenvolvedor"

Valor – 1.700,00

Vamos tratar os dados das promoções como um *array* de documentos dentro do documento.

```
db.emp.insert (
  { Nome: "Fátima",
    Idade: 29,
    Cargo: "Analista",
    Salario: 12345.67,
    Dependentes: ["Gohan"],
    Departamento: { Nome: "Vendas",
                     Local: "Campinas"
                   },
    Promoções: [{Ano: 2001,
                  Cargo: "Estagiário",
                  Valor: 180.00},
                {Ano: 2002,
                  Cargo: "Analista",
                  Valor: 1700.00}]
  }
)
```

Código-fonte 4.11 – Exemplo de inclusão de dados com um *array* de documentos na coleção **emp**
Fonte: Elaborado pelo autor (2020)

Note que temos um *array* de documentos em promoções. A figura “Resultado da operação de inclusão de um *array* de documentos na coleção **emp**” mostra o resultado dessa inclusão.

```
> db.emp.insert({ Nome: "Fátima",
...               Idade: 29,
...               Cargo: "Analista",
...               Salario: 12345.67,
...               Dependentes:["Gohan"],
...               Departamento: { Nome: "Vendas",
...                               Local: "Campinas"
...                             },
...               Promoções: [{Ano: 2001,
...                             Cargo: "Estagiário",
...                             Valor: 180.00},
...                           {Ano: 2002,
...                             Cargo: "Analista",
...                             Valor: 1700.00}]
...             })
writeResult({ "nInserted" : 1 })
>
```

Figura 4.44 – Resultado da operação de inclusão de um *array* de documentos na coleção **emp**
Fonte: Elaborado pelo autor (2020)

4.26 Inclusão de documentos por meio de variáveis

O uso de variáveis permite que incluamos vários documentos em uma única operação. Imagine que você queira incluir os seguintes dados em uma coleção denominada *grade*:

Nome	Autor	Professor	Novo	Cód	Mês	ISBN	Valor
Estruturas de dados	Rossetti		FALSE			12345678	171,13
Engenharia de software		Rita		FES	2		
Inglês instrumental	Ana		FALSE			45678901	122,99
Interfaces com usuário		André		IU	2		
Programação orientada a objeto	Gatti		TRUE			23456789	161,16
Modelagem orientada a objeto		Rodrigo		MOO1	2		
Banco de dados	Angélica		TRUE			34567890	133,29
Gestão empresarial	Silva	Silva	TRUE	GE	2	56789012	156,55

Tabela 4.1 – Dados a incluir na coleção *grade*
Fonte: Adaptado pelo autor (2020)

É possível criar uma variável usando a instrução **var**, que receberá esses documentos.

```
var dados = [  
  {Nome: 'Estruturas de dados',  
    Autor: 'Rossetti',  
    Novo: false,  
    ISBN: 12345678,  
    Valor: 171.13},  
  
  {Nome: 'Engenharia de software',  
    Professor: 'Rita',  
    Cod: 'FES', Mes: 2},  
  
  {Nome: 'Inglês instrumental',  
    Autor: 'Ana',  
    Novo: false,  
    ISBN: 45678901,  
    Valor: 122.99},  
  
  {Nome: 'Interfaces com usuário',  
    Professor: 'André',  
    Cod: 'IU',  
    Mes: 2},  
  
  {Nome: 'Programação orientada a objeto',  
    Autor: 'Gatti',  
    Novo: true,  
    ISBN: 23456789,  
    Valor: 161.16},  
  
  {Nome: 'Modelagem orientada a objeto',  
    Professor: 'Rodrigo',  
    Cod: 'MOO1',  
    Mes: 2},  
  
  {Nome: 'Banco de dados',  
    Autor: 'Angélica',  
    Novo: true,  
    ISBN: 34567890,  
    Valor: 133.29},  
  
  {Nome: 'Gestão empresarial',  
    Autor: 'Silva',  
    Professor: 'Silva',  
    Novo: true,  
    Cod: 'GE',  
    Mes: 2,  
    ISBN: 56789012, Valor: 156.55}  
]
```

Código-fonte 4.12 – Exemplo de criação de variável com oito documentos
Fonte: Elaborado pelo autor (2020)

Essa variável pode ser usada junto com o método `insert()` para incluir os documentos na coleção `grade`.

```
db.grade.insert(dados);
```

Código-fonte 4.13 – Exemplo de inclusão de dados através de variável

Fonte: Elaborado pelo autor (2020)

4.27 Consultando dados

Os documentos que foram incluídos nas coleções podem ser consultados usando o método `find()`. Esse método retorna um *array* com os objetos da coleção, mesmo que a coleção tenha apenas um objeto (MONGODB, 2018).

```
db.coleção.find(consulta, projeção)
```

Código-fonte 4.14 – Sintaxe do método `find()`

Fonte: MongoDB (2020)

Em que:

- **Coleção:** Indica o nome da coleção.
- **Consulta:** Indica quais serão os filtros aplicados na pesquisa. Para retornar todos os documentos de uma coleção, basta omitir este parâmetro ou informar um documento vazio (`{}`). Este parâmetro é opcional.
- **Projeção:** Indica quais são os campos específicos do documento que serão retornados pela pesquisa. Se omitido, todos os campos do documento serão retornados. Este parâmetro é opcional.

A consulta básica aos dados de uma coleção MongoDB é bem simples. Vejamos um exemplo de uso:

```
db.emp.find();
```

Código-fonte 4.15 – Exemplo de consulta aos documentos na coleção **emp**

Fonte: Elaborado pelo autor (2020)

A imagem mostra o resultado parcial da execução do comando.

```

db.emp.find();
{ "_id" : ObjectId("5b3bee4ba8a46b85015e70ed"), "Nome" : "Bianka", "Idade" : 22, "Cargo" : "Desenvolvedor",
  "_id" : ObjectId("5b3bfa3da8a46b85015e70ee"), "Nome" : "Neusa", "Idade" : 23, "Cargo" : "Desenvolvedor",
  "_id" : ObjectId("5b3c09f2a8a46b85015e70ef"), "Nome" : "Thais", "Idade" : 19, "Cargo" : "Analista", "Salari
  "_id" : ObjectId("5b3c1673a8a46b85015e70f0"), "Nome" : "Fátima", "Idade" : 29, "Cargo" : "Analista", "Salari

```

Figura 4.45 – Resultado parcial da consulta aos dados da coleção **emp**

Fonte: Elaborado pelo autor (2020)

O método `find()` retornou uma lista com todos os documentos da coleção. Podemos usar o método `findOne()` para retornar o primeiro objeto encontrado na coleção que atenda aos critérios de busca da consulta (MONGODB, 2018).

```
db.coleção.findOne(consulta, projeção)
```

Código-fonte 4.16 – Sintaxe do método `findOne()`

Fonte: MongoDB (2020)

Em que:

- Coleção – Indica o nome da coleção.
- Consulta – Indica quais serão os filtros aplicados na pesquisa. Para retornar todos os documentos de uma coleção, basta omitir este parâmetro ou informar um documento vazio (`{}`). Este parâmetro é opcional.
- Projeção – Indica quais são os campos específicos do documento que serão retornados pela pesquisa. Se omitido, todos os campos do documento serão retornados. Este parâmetro é opcional.

A consulta básica usando o `findOne()` é bem simples. Vejamos um exemplo de uso:

```
db.emp.findOne();
```

Código-fonte 4.17 – Exemplo de consulta ao primeiro documento na coleção **emp**

Fonte: Elaborado pelo autor (2018)

A imagem mostra o resultado da execução do comando.

```

> db.emp.findOne();
{
  "_id" : ObjectId("5b3bee4ba8a46b85015e70ed"),
  "Nome" : "Bianka",
  "Idade" : 22,
  "Cargo" : "Desenvolvedor",
  "salario" : 12000
}
>

```

Figura 4.46 – Resultado de uso do método `findOne()` em consulta aos dados da coleção **emp**

Fonte: Elaborado pelo autor (2020)

Note que apenas os dados do primeiro objeto da coleção foram exibidos. Perceba que há um atributo novo em nosso documento, **_id**.

4.28 Atributo **_id**

O atributo **_id** tem o comportamento similar a uma chave primária em bancos de dados relacionais. Cada documento possui seu próprio **_id** e o seu número é único. Um índice único (*unique index*) é criado para o atributo **_id** quando a coleção é criada.

Ele sempre é o primeiro campo de um documento. Se não for informado durante a inclusão de dados, um tipo de dados **ObjectId** do **BSON** com 12 *bytes* será atribuído a ele. Se o valor do **_id** for informado e ele não for o primeiro campo, o servidor deslocará o atributo para a primeira posição. Normalmente, a estrutura do **_id** segue a seguinte composição:

- Os quatro primeiros bytes indicam o *timestamp* da criação do objeto.
- 3 bytes identificam a máquina.
- 2 bytes mostram o identificador (id) do processo.
- 3 bytes contador, iniciado com um valor aleatório.

Uma coleção não requer que documentos armazenados nela tenham todos os mesmos campos. Todos documentos da coleção possuem um **_id** único.



Figura 4.47 – MongoDB – ObjectId
Fonte: MongoDB(2020)

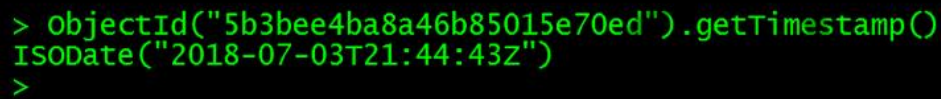
Ordenar os documentos pelo `_id` é o equivalente a ordená-los pela data de criação. Podemos usar o método `getTimestamp()` para verificarmos a data de criação de um objeto. Exemplo de uso do método `getTimestamp()`.

```
ObjectId("5b3bee4ba8a46b85015e70ed").getTimestamp();
```

Código-fonte 4.18 – Exemplo de uso do método `getTimestamp()`

Fonte: Elaborado pelo autor (2020)

A imagem exibe o resultado da execução do comando.



```
> ObjectId("5b3bee4ba8a46b85015e70ed").getTimestamp()  
ISODate("2018-07-03T21:44:43Z")  
>
```

Figura 4.48 – Resultado de uso do método `getTimestamp()`

Fonte: Elaborado pelo autor (2020)

Note que o objeto foi criado no dia 03 de julho de 2018, às 21h44min43s do horário local.

4.29 Pesquisa por igualdade

Uma consulta vazia (`{}`) retorna todos os documentos de uma coleção, não especificar uma consulta é o equivalente a uma consulta vazia. Isso quer dizer que o comando `db.emp.find()` é equivalente a `db.emp.find({})`

Listar todos os dados de uma coleção é muito útil, mas, algumas vezes, temos a necessidade de fazer consultas por igualdade. Para isso, basta usarmos o método `find()` com uma chave-valor. Por exemplo:

```
db.emp.find({Idade: 19});
```

Código-fonte 4.19 – Exemplo de consulta por igualdade

Fonte: Elaborado pelo autor (2020)

Nesse exemplo, serão listados todos os documentos cujo atributo **Idade** seja igual a 19. O método `find()` pode encadear vários métodos para melhorar a busca de documentos. Por exemplo, podemos encadear o método `pretty()` para formatar a saída de nossa consulta e deixá-la mais legível. O exemplo de código ilustra isso.

```
db.emp.find({Idade: 19}).pretty();
```

Código-fonte 4.20 – Exemplo de consulta por igualdade com saída formatada pelo método `pretty()`

Fonte: Elaborado pelo autor (2020)

A imagem ilustra a execução do comando.

```
> db.emp.find({Idade: 19}).pretty();
{
  "_id" : ObjectId("5b3c09f2a8a46b85015e70ef"),
  "Nome" : "Thais",
  "Idade" : 19,
  "Cargo" : "Analista",
  "Salario" : 14530.77,
  "Dependentes" : [
    "Angela",
    "Dora",
    "Hugo"
  ],
  "Departamento" : {
    "Nome" : "Pesquisa",
    "Local" : "São Paulo"
  }
}
```

Figura 4.49 – Resultado da busca por igualdade e do método `pretty()`

Fonte: Elaborado pelo autor (2020)

4.30 Classificando a saída dos dados

Vários outros métodos podem ser concatenados ao método `find()`, o método `sort()`, por exemplo, classifica a saída em ordem crescente ou decrescente.

- O valor 1 no parâmetro indica que os dados serão classificados em ordem crescente.
- O valor -1 no parâmetro indica que os dados serão classificados em ordem decrescente

Vejamos alguns exemplos de utilização:

```
db.emp.find().sort({Nome:1});
db.emp.find().sort({Idade:-1});
db.emp.find().sort({Cargo:1, Idade:-1});
```

Código-fonte 4.21 – Exemplo de uso do método `sort()`

Fonte: Elaborado pelo autor (2020)

No exemplo, o primeiro comando retorna todos os documentos da coleção **emp** classificados pelo nome, em ordem crescente; o segundo comando retorna todos os documentos da coleção classificados pela idade, em ordem decrescente e o último comando retorna todos os documentos classificados pelo cargo, em ordem crescente, e pela idade, em ordem decrescente.

4.31 Operadores de comparação

Os operadores de comparação retornam um valor booleano. Comparam as duas expressões informadas nos argumentos e retornam o valor *true* ou *false*.

- `$lt` (abreviação de *less than*) usado para verificar se um valor é menor que outro. Retorna *true* se o primeiro valor for menor que o segundo.

```
db.emp.find({Idade:{ $lt: 22}});
```

Código-fonte 4.22 – Exemplo de uso do operador `$lt`
Fonte: Elaborado pelo autor (2020)

- `$lte` (abreviação de *less than or equal*) usado para verificar se um valor é menor ou igual a outro. Retorna *true* se o primeiro valor for menor ou igual ao segundo.

```
db.emp.find({Idade:{ $lte: 22}});
```

Código-fonte 4.23 – Exemplo de uso do operador `$lte`
Fonte: Elaborado pelo autor (2020)

- `$gt` (abreviação de *greater than*) usado para verificar se um valor é maior que outro. Retorna *true* se o primeiro valor for maior que o segundo.

```
db.emp.find({Idade:{ $gt: 22}});
```

Código-fonte 4.24 – Exemplo de uso do operador `$gt`
Fonte: Elaborado pelo autor (2020)

- `$gte` (abreviação de *greater than or equal*) usado para verificar se um valor é maior ou igual a outro. Retorna *true* se o primeiro valor for maior ou igual ao segundo.

```
db.emp.find({Idade:{ $gte: 22}});
```

Código-fonte 4.25 – Exemplo de uso do operador `$gte`
Fonte: Elaborado pelo autor (2020)

O quadro seguinte resume esses operadores e os exemplifica.

Operador	Significado	Exemplo de Uso
\$lt	Menor que	db.emp.find({Idade:{\$lt: 23}});
\$lte	Menor que ou igual a	db.emp.find({Idade:{\$lte: 23}});
\$gt	Maior que	db.emp.find({Idade:{\$gt: 23}});
\$gte	Maior que ou igual a	db.emp.find({Idade:{\$gte: 23}});

Quadro 4.1 – Quadro com os principais operadores de comparação
Fonte: Adaptado de MongoDB (2020)

O exemplo do operador `$lt`, exibido no quadro, retorna todos os documentos nos quais o campo **Idade** tenha o valor menor do que 23.

O exemplo do operador `$lte`, exibido no quadro, retorna todos os documentos nos quais o campo **Idade** tenha o valor menor ou igual a 23.

O exemplo do operador `$gt`, exibido no quadro, retorna todos os documentos nos quais o campo **Idade** tenha o valor maior do que 23.

O exemplo do operador `$gte`, exibido no quadro, retorna todos os documentos nos quais o campo **Idade** tenha o valor maior ou igual a 23.

O quadro a seguir compara o uso de comandos SQL dos bancos de dados relacionais e o MongoDB.

Exemplo Em SQL	Exemplo em MongoDB
SELECT * FROM emp	db.emp.find()
SELECT * FROM emp WHERE idade = 33	db.emp.find({idade: 33})
SELECT * FROM emp WHERE idade > 33	db.emp.find({idade: {\$gt: 33}})
SELECT * FROM emp WHERE idade >= 33	db.emp.find({idade: {\$gte: 33}})
SELECT * FROM emp WHERE idade < 33	db.emp.find({idade: {\$lt: 33}})
SELECT * FROM emp WHERE idade <= 33	db.emp.find({idade: {\$lte: 33}})

Quadro 4.2 – Quadro de equivalência de operadores SQL versus MongoDB
Fonte: Adaptado de MongoDB (2020)

4.32 Operadores lógicos

Além dos operadores de comparação, podemos usar os operadores lógicos:

- `$or` retorna documentos pesquisados quando pelo menos uma das cláusulas de comparação for verdadeira.

```
db.emp.find({$or:[{Idade:{$gt:21}},{Salario:{$lt:12500}}]});
```

Código-fonte 4.26 – Exemplo de uso do operador `$or`

Fonte: Elaborado pelo autor (2020)

- `$and` retorna documentos pesquisados quando todas as cláusulas de comparação forem verdadeiras.

```
db.emp.find({$and:[{Idade:{$gt:21}},{Salario:{$lt:12500}}]});  
ou  
db.emp.find({Idade: {$gt: 21},Salario:{$lt: 12500}});
```

Código-fonte 4.27 – Exemplo de uso do operador `$and`

Fonte: Elaborado pelo autor (2020)

- `$nor` retorna documentos pesquisados que não satisfaçam as cláusulas de comparação.

```
db.emp.find({$nor:[{Idade:{$gt:22}},{Salario:{$gt:12500}}]});
```

Código-fonte 4.28 – Exemplo de uso do operador `$nor`

Fonte: Elaborado pelo autor (2020)

- `$ne` retorna documentos pesquisados cujos valores sejam diferentes do valor informado.

```
db.emp.find({Idade:{$ne:22}});
```

Código-fonte 4.29 – Exemplo de uso do operador `$ne`

Fonte: Elaborado pelo autor (2020)

- `$not` efetua uma operação NOT lógica.

```
db.emp.find({Idade:{$not:{$gt:21}}});
```

Código-fonte 4.30 – Exemplo de uso do operador `$not`

Fonte: Elaborado pelo autor (2020)

- O quadro seguinte resume esses operadores e os exemplifica.

Operador	Significado	Exemplo de Uso
\$or	Ou lógico	db.emp.find({\$or:[{Idade:{\$lt:21}}, {Salario:{\$gt:12500}}]});
\$and	E lógico	db.emp.find({Idade: {\$lt: 21},Salario:{\$gt: 12500}});
\$nor	NOR lógico	db.emp.find({\$nor:[{Idade:{\$lt:21}}, {Salario:{\$gt:12500}}]});
\$ne	Diferente	db.emp.find({Idade:{\$ne:21}});

Quadro 4.3 – Quadro com os principais operadores lógicos
Fonte: Adaptado de MongoDB (2020)

O exemplo do operador `$or`, exibido no quadro, retorna todos os documentos nos quais o campo **Idade** tenha o valor menor do que 21 ou o campo **Salario** seja maior que 12.500,00.

O exemplo do operador `$and`, exibido no quadro, retorna todos os documentos nos quais o campo **Idade** tenha o valor menor que 21 e o campo **Salario** seja maior que 12.500,00.

O exemplo do operador `$nor`, exibido no quadro, retorna todos os documentos nos quais o campo **Idade** tenha o valor maior do que 21 e o campo **Salario** seja menor que 12.500,00.

O quadro compara o uso de comandos SQL dos bancos de dados relacional e o MongoDB.

Exemplo Em SQL	Exemplo em MongoDB
SELECT * FROM emp WHERE Idade > 33 AND Idade < 40	db.emp.find({Idade: {\$gt: 33, \$lt: 40}})
SELECT * FROM emp WHERE Idade = 32 AND Nome = 'Thais'	db.emp.find({Idade: 32, Nome: "Thais"})
SELECT * FROM emp WHERE Idade = 33 OR Nome = 'Thais'	db.emp.find({\$or:[{Idade:33}, {Nome: "Thais"}]}))
SELECT * FROM emp WHERE Idade >= 32 AND (Nome = 'Thais' OR Cargo='Analista')	db.emp.find(Idade: {\$gte:32}, \$or: [{Nome: "Thais"}, {Cargo: "Analista"}]))
SELECT * FROM emp WHERE Idade > 33 AND Idade < 40	db.emp.find({Idade: {\$gt: 33, \$lt: 40}})
SELECT * FROM emp WHERE Idade = 32 AND Nome = 'Thais'	db.emp.find({Idade: 32, Nome: "Thais"})

Quadro 4.4 – Quadro de equivalência de operadores lógicos SQL versus MongoDB
Fonte: Adaptado de MongoDB (2020)

4.33 Operadores para tratamento arrays

O MongoDB também fornece operadores para tratarmos *arrays*:

- `$in` retorna documentos nos quais o valor pesquisado esteja na lista de valores de um *array*.

```
db.emp.find({Dependentes:{$in:["Rosa", "Hugo"]}});
```

Código-fonte 4.31 – Exemplo de uso do operador `$in`

Fonte: Elaborado pelo autor (2020)

- `$nin` retorna documentos nos quais o valor pesquisado não esteja na lista de valores de um *array* ou o campo pesquisado não exista no documento.

```
db.emp.find({Dependentes:{$nin:["Rosa", "Rita"]}});
```

Código-fonte 4.32 – Exemplo de uso do operador `$nin`

Fonte: Elaborado pelo autor (2020)

- `$all` retorna documentos nos quais todos valores pesquisados estejam na lista de valores de um *array*.

```
db.emp.find({Dependentes:{$all:["Rita", "Ana"]}});
```

Código-fonte 4.33 – Exemplo de uso do operador `$all`

Fonte: Elaborado pelo autor (2020)

- `$exists` não específico para o uso em *arrays*, mas pode ser usado para complementar os operadores `$in` e `$nin`. É usado para testar se um campo existe em um documento ou não. No exemplo, usamos o método `count()` que efetua a contagem do número de elementos retornados.

```
db.emp.find({Dependentes:{$exists: true}}).count()  
ou  
db.emp.find({Dependentes:{$exists: false}}).count()
```

Código-fonte 4.34 – Exemplo de uso do operador `$exists`

Fonte: Elaborado pelo autor (2020)

- O quadro “os principais operadores de arrays” mostra esses operadores e os exemplifica.

Operador	Exemplo de Uso
\$in	db.emp.find({Dependentes:{ \$in:["Dora", "Pietra"]}});
\$nin	db.emp.find({Dependentes:{ \$nin:["Dora", "Pietra"]}});
\$all	db.emp.find({Dependentes:{ \$all:["Dora", "Hugo"]}});
\$exists	db.emp.find({Dependentes:{ \$exists:true, \$nin:["Dora", "Pietra"]}});

Quadro 4.5 – Quadro com os principais operadores de *arrays*
 Fonte: Adaptado de MongoDB (2020)

O exemplo do operador `$in`, exibido no quadro, retorna todos os documentos nos quais o campo **Dependentes** tenha o valor “Dora” ou o valor “Pietra”.

O exemplo do operador `$nin`, exibido no quadro, retorna todos os documentos nos quais o campo **Dependentes** não exista ou exista, mas não contenha o valor “Dora” ou o valor “Pietra”.

O exemplo do operador `$all`, exibido no quadro, retorna todos os documentos nos quais o campo **Dependentes** contenha tanto o valor “Dora” quanto o valor “Hugo”.

O exemplo do operador `$exists`, exibido no quadro, retorna todos os documentos nos quais o campo **Dependentes** exista e não contenha o valor “Dora” ou o valor “Pietra”.

4.34 Projeção

O uso do opcional de projeção permite indicar quais campos serão retornados pelo método `find()`.

```
db.emp.find({Idade:22},{_id:0, Idade:0, Cargo:0});
```

Código-fonte 4.35 – Exemplo de projeção
 Fonte: Elaborado pelo autor (2020)

No exemplo, serão exibidos todos os campos dos funcionários com idade igual a 22 anos, exceto os campos **_id**, **Idade** e **Cargo**. O quadro “Equivalência de projeção SQL versus MongoDB” compara o uso de comandos SQL dos bancos de dados relacional e o MongoDB.

Exemplo Em SQL	Exemplo em MongoDB
SELECT * FROM emp	db.emp.find()
SELECT Nome, Salario FROM emp WHERE Idade = 22	db.emp.find({Idade:22},{_id:0, Idade:0, Cargo:0})
SELECT Nome, Salario FROM emp WHERE Idade = 22	db.emp.find({Idade:22},{Nome:1, Salario:1})

Quadro 4.6 – Quadro de equivalência de projeção SQL versus MongoDB
Fonte: Adaptado de MongoDB (2020)

4.35 Expressões regulares

Em MongoDB, expressões regulares permitem pesquisar por padrões em campos texto. Há duas formas para a sintaxe de expressões regulares (MONGODB, 2020).

```
{<campo>: { $regex: /padrão/, $options: '<opções>' } }
{<campo>: { $regex: 'padrão', $options: '<opções>' } }
{<campo>: { $regex: /padrão/<opções> } }
```

OU

```
{ <campo>: /padrão/<opções> }
```

Código-fonte 4.36 – Sintaxe do método find()

Fonte: MongoDB (2020)

Em que:

Campo – Indica o campo no qual será pesquisado o padrão.

Padrão – Indica o padrão a ser pesquisado.

Opções – Indica as opções a serem usadas com a expressão regular. Algumas das opções são:

- *i* – Indica *case insensitive*, isto é, o texto pode estar tanto em letras maiúsculas quanto em letras minúsculas.
- *m* – Usado em padrões que incluem âncoras, isto é, *^* para o início de um texto e *\$* para o final do texto.
- *x* – Usado para ignorar todos os espaços em branco no padrão.

Alguns exemplos:

```
db.emp.find({Nome:{ $regex:"^B" }});
```

Código-fonte 4.37 – Exemplo de expressão regular
 Fonte: Elaborado pelo autor (2020)

- No exemplo, estamos pesquisando por todos os nomes iniciados pela letra “B” maiúscula.

```
db.emp.find({Nome:{$regex:"^B", $options: 'i'}});
```

Código-fonte 4.38 – Exemplo de expressão regular
 Fonte: Elaborado pelo autor (2020)

- No exemplo, estamos pesquisando por todos os nomes iniciados pela letra “B”, independentemente de ter sido digitado em letra maiúscula ou minúscula.

```
db.emp.find({Nome:{$regex:"i"}});
```

Código-fonte 4.39 – Exemplo de expressão regular
 Fonte: Elaborado pelo autor (2020)

- No exemplo, estamos pesquisando por todos os nomes que possuem a letra “i”, minúscula, em qualquer parte do nome.

```
db.emp.find({Nome:{$regex:"i", $options: 'i'}});
```

Código-fonte 4.40 – Exemplo de expressão regular
 Fonte: Elaborado pelo autor (2020)

- No exemplo, estamos pesquisando por todos os nomes que possuem a letra “i”, independentemente de ter sido digitada em letra maiúscula ou minúscula, em qualquer parte do nome.

```
db.emp.find({Nome:{$regex:"s$"}});
```

Código-fonte 4.41 – Exemplo de expressão regular
 Fonte: Elaborado pelo autor (2020)

- No exemplo, estamos pesquisando por todos os nomes terminados pela letra “s”, minúscula.

O quadro “equivalência de expressões regulares entre SQL versus MongoDB” compara o uso de comandos SQL dos bancos de dados relacionais e o MongoDB.

Exemplo Em SQL	Exemplo em MongoDB
SELECT * FROM emp WHERE nomeLIKE '%h%'	db.emp.find({Nome:{\$regex:"h"}});

Quadro 4.7 – Quadro de equivalência de expressões regulares entre SQL versus MongoDB
 Fonte: Adaptado de MongoDB (2020)

4.36 Alterando dados

O método `update()` permite alterar um ou mais documentos em uma coleção. É possível alterar um ou mais campos de um documento ou, até mesmo, o documento inteiro. Por padrão, o método `update()` altera apenas um documento por vez (MONGODB, 2020). Sua sintaxe é:

```
db.coleção.update(consulta, atualização, opções)
```

Código-fonte 4.42 – Sintaxe do método `update()`

Fonte: MongoDB (2020)

Em que:

Coleção – Indica o nome da coleção.

Consulta – Indica quais serão os critérios para pesquisar o documento que será alterado.

Atualização – Indica a modificação que será feita.

Opções – Indica quais são as opções para o comando. Entre as opções estão:

- `upsert` – Opcional. Se definido como *true*, cria um novo documento, caso nenhum documento atenda os critérios de busca. O padrão é *false*.
- `multi` - Opcional. Se definido como *true*, atualiza todos os documentos que atendam os critérios de busca. O padrão é *false*.

O método `update()` pode alterar um documento inteiro. O exemplo abaixo ilustra o caso.

```
db.países.find();
db.países.insert({_id:504, Nome:"Mauritânia", Sobrenome:
"Nouakchott"});
db.países.find();
db.países.update({_id:504}, {Pais:"Honduras", Capital:"Tegucigalpa"});
```

Código-fonte 4.43 – Exemplo de alteração completa de um documento

Fonte: Elaborado pelo autor (2020)

Nesse caso, o documento inteiro foi substituído.

O método `update()` tem operadores para tratar os campos individualmente. Entre eles, destacamos:

- `$inc` incrementa o valor de um campo pelo valor informado no operador.

```
db.emp.insert({Nome: "Ana", idade: 22});
db.emp.find({Nome: "Ana"});
db.emp.update({Nome: "Ana"}, {$inc: { idade: 5}});
db.emp.find({Nome: "Ana"});
```

Código-fonte 4.44 – Exemplo de uso do operador `$inc`

Fonte: Elaborado pelo autor (2020)

- `$set` modifica o conteúdo de um campo específico. Caso o campo não exista, um novo campo será acrescentado ao documento.

```
db.emp.update({Nome: "Ana"}, {$set: { idade: 22}});
db.emp.find({Nome: "Ana"});
db.emp.update({Nome: "Ana"}, {$set: { salario: 15000}});
db.emp.find({Nome: "Ana"});
```

Código-fonte 4.45 – Exemplo de uso do operador `$set`

Fonte: Elaborado pelo autor (2020)

- `$rename` renomeia um campo.

```
db.emp.find({Nome: "Ana"});
db.emp.update({Nome: "Ana"}, {$rename: {"slaario": "salario"}});
db.emp.find({Nome: "Ana"});
```

Código-fonte 4.46 – Exemplo de uso do operador `$rename`

Fonte: Elaborado pelo autor (2020)

- `$unset` remove um campo específico de um documento.

```
db.emp.find({Nome: "Ana"});
db.emp.update({Nome: "Ana"}, {$unset: {salario: ""}});
db.emp.find({Nome: "Ana"});
```

Código-fonte 4.47 – Exemplo de uso do operador `$unset`

Fonte: Elaborado pelo autor (2020)

O quadro “equivalência de operadores de atualização entre SQL versus MongoDB” compara o uso de comandos SQL dos bancos de dados relacionais e o MongoDB.

Exemplo Em SQL	Exemplo em MongoDB
UPDATE emp SET Idade = Idade + 2 WHERE Nome = 'Ana'	db.emp.update({Nome: "Ana"}, {\$inc: {Idade: 2}}, {multi: true})
UPDATE emp SET Idade = 33 WHERE Nome = 'Ana'	db.emp.update({Nome: "Ana"}, {\$set: {Idade: 33}}, {multi: true})

Quadro 4.8 – Equivalência de operadores de atualização entre SQL versus MongoDB

Fonte: Adaptado de MongoDB (2020)

4.37 Eliminando dados

O método `remove()` remove todos os documentos que atendam as condições de pesquisa especificadas. A opção `justOne` limita a operação de remoção a um único documento (MONGODB, 2020). Sua sintaxe é:

```
db.coleção.remove(consulta, {justOne:<boolean>})
```

Código-fonte 4.48 – Sintaxe do método `remove()`

Fonte: MongoDB (2020)

Em que:

Coleção – Indica o nome da coleção.

Consulta – Indica quais serão os critérios para pesquisar o documento que será removido.

`justOne` – Indica que apenas um documento será removido.

Se todos os documentos forem eliminados, mesmo assim, a coleção continuará existindo. Caso queira eliminar a coleção use o método `drop()`;

Exemplo de uso:

```
db.apaga.insert({cod:7369,nome:"Maria",cargo:"DBA"});
db.apaga.insert({cod:7499,nome:"Rosa",cargo:"DBA"});
db.apaga.insert({cod:7521,nome:"Ana",cargo:"DBA"});
db.apaga.insert({cod:7566,nome:"Rita",cargo:"DBA"});
db.apaga.find();
db.apaga.remove({cod:{$gt:7521}});
db.apaga.find();
db.apaga.remove({cargo:"DBA"},true);
db.apaga.find();
db.apaga.remove({});
db.apaga.find();
show collections;
db.apaga.drop();
show collections;
```

Código-fonte 4.49 – Exemplo de remoção documentos de uma coleção

Fonte: Elaborado pelo autor (2020)

O quadro “equivalência de operações de remoção entre SQL versus MongoDB” compara o uso de comandos SQL dos bancos de dados relacionais e o MongoDB.

Exemplo Em SQL	Exemplo em MongoDB
DELETE FROM apaga WHERE nome = 'Ana'	db.users.remove({name: "Ana"})

Quadro 4.9 – Quadro de equivalência de operações de remoção entre SQL versus MongoDB
Fonte: Adaptado de MongoDB (2020)

Agora que você já sabe realizar todas as operações com MongoDB, imagine as possibilidades para dar aquele “up” em suas aplicações.

4.38 Help

O MongoDB possui vários comandos que fornecem ajuda ao desenvolvedor. Observe alguns desses comandos:

Comando	Função	Exemplo de Uso
help;	Exibe esta lista de funções.	help;
db.help();	Fornece uma lista das funções de ajuda para um determinado método do banco de dados.	db.help('insert');
db. <coleção>.help();	Fornece uma lista dos métodos aplicáveis a uma coleção. <coleção> indica o nome de uma coleção a ser criada ou previamente existente.	db.biblioteca.help();

Quadro 4.10 – Quadro com as principais funções de ajuda do banco de dados MongoDB
Fonte: MongoDB (2020)

O comando `show` também nos ajuda muito,. Com ele, podemos obter informações sobre quais coleções existem em nosso banco, quais são os bancos de dados existentes no servidor, quais são os usuários do banco de dados corrente. Veja um quadro com os principais usos do comando `show`.

Comando	Função
show dbs;	Fornece uma lista de todos os bancos de dados do servidor.
show collections;	Fornece uma lista de todas as coleções do banco de dados corrente.
show users;	Fornece uma lista de todos os usuários do banco de dados corrente.
show roles;	Fornece uma lista de todas as roles, pré-definidas ou definidas pelo usuário, do banco de dados corrente.
show profile;	Fornece uma lista das últimas cinco operações que levaram um milissegundo ou mais para serem executadas.
show databases;	Fornece uma lista de todos os bancos de dados disponíveis.

Quadro 4.11 – Principais usos do comando `show` do banco de dados MongoDB
Fonte: MongoDB (2020)

Também nos ajuda saber quais são os símbolos e operadores que usamos para trabalhar com os documentos. Observe os principais símbolos e operadores:

Operador / Símbolo	Significado
() (parênteses)	Indica o uso de um método
[] (colchetes)	Usado em <i>arrays</i>
{ } (chaves)	Usado em documentos
. (ponto)	Separador de métodos
= (igual)	Atribui valor para variáveis
, (vírgula)	Separa atributos. Não deve ser usada com números
: (dois pontos)	Especifica o valor de um atributo
' (aspas simples)	Indica que o valor de um atributo é texto
" (aspas duplas)	Indica que o valor de um atributo é texto
; (ponto-e-vírgula)	Encerra um comando

Quadro 4.12 – Símbolos e operadores usados em conjunto com os documentos MongoDB
Fonte: MongoDB (2020)

4.39 Correspondência com o relacional

Para aqueles que conhecem bancos de dados relacionais, o quadro “Comparação entre os termos” apresenta uma comparação entre os termos usados em um banco de dados relacional Oracle e um banco de dados orientado a documentos MongoDB.

Oracle (RDBMS)	MongoDB (documento)
Database	Database
Instância de Banco de Dados	Instância MongoDB
Esquema	Banco de Dados
Tabela, <i>View</i>	Coleção
Tupla	Documento (JSON, BSON)
Coluna	Campo ou <i>Field</i>
ROWID / Primary Key	<code>_id</code>
Junção	DBRef (Embedded Document)
Foreign Key	Reference
Partição	Shard
Select	Método <i>find</i>
Insert	Método <i>insert</i>
Update	Método <i>update</i>
Delete	Método <i>remove</i>

Quadro 4.13 – Comparação entre os termos usados pelo banco de dados relacional Oracle e o banco de dados orientado a documentos MongoDB

Fonte: MongoDB (2020)

4.40 Exemplo Airbnb Collections

Algumas coleções já são disponibilizadas no MongoDB Atlas para explorar os recursos da plataforma. Realizaremos sua carga e construiremos visualizações a partir da coleção Airbnb.

4.41 Collections

No Atlas, carregar as coleções exemplo. Clicar em Load Sample Dataset associado aos pontinhos ao lado da aba Collections. Na sequência, confirmar no botão Load Sample Dataset. Serão carregados aproximadamente 350Mb. Esse processo pode demorar alguns minutos. Após a carga uma mensagem de sucesso será apresentada.

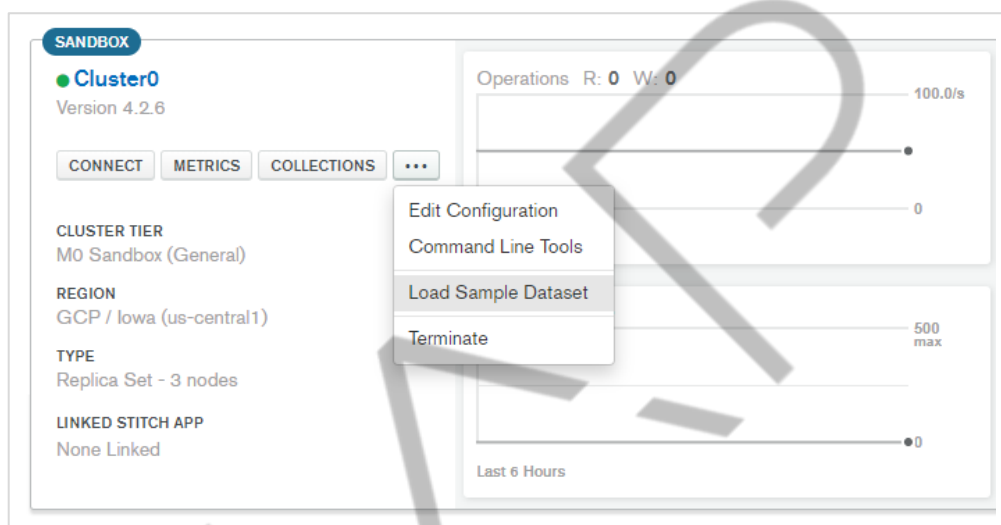


Figura 4.50 – Load Sample Dataset
Fonte: Elaborado pelo autor (2020)

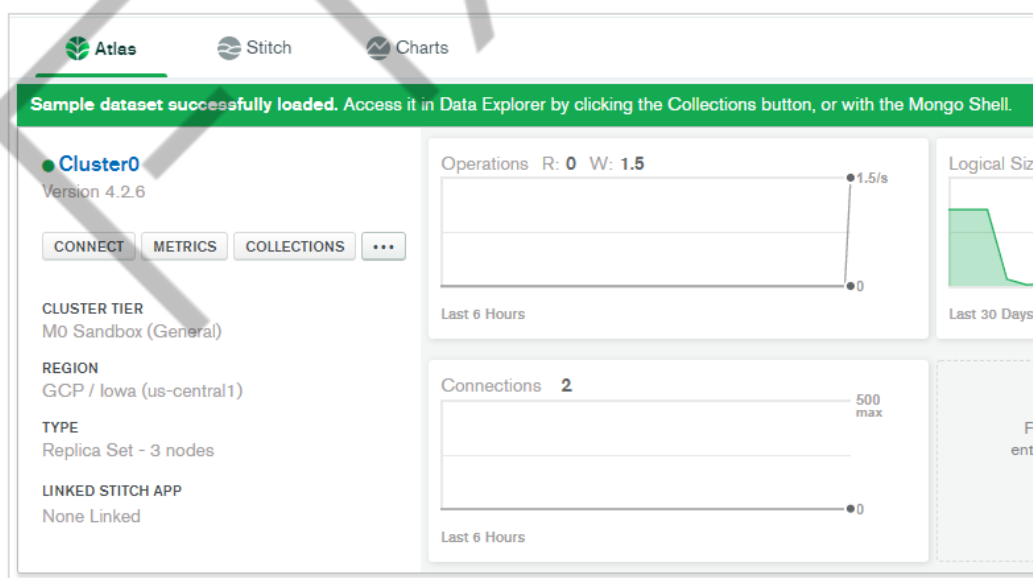


Figura 4.51 – Carga com sucesso
Fonte: Elaborado pelo autor (2020)

4.42 Airbnb – Construir visualizações

No Atlas, vamos utilizar o MongoDB Charts para construir algumas visualizações a partir da coleção exemplo do Airbnb. O primeiro passo é clicar em Charts e verificar as coleções disponíveis (Data Sources). No nosso exemplo utilizaremos `sample_airbnb.listingsAndReviews`.

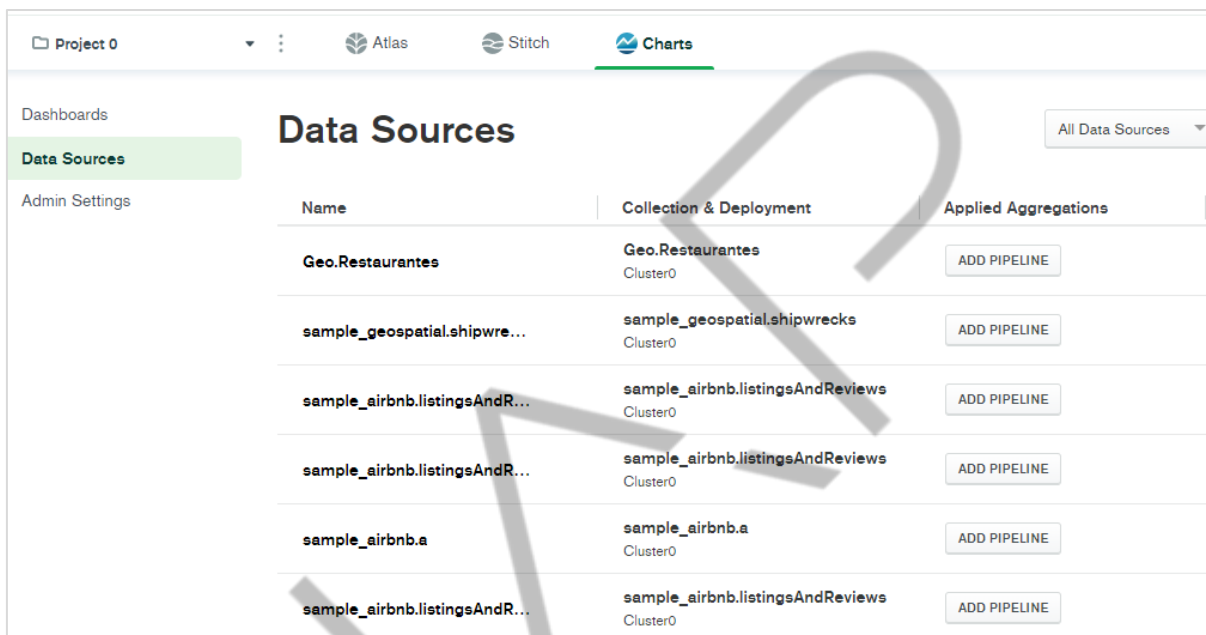


Figura 4.52 – Associar Data Source
Fonte: Elaborado pelo autor (2020)

Na sequência, vamos criar um novo dashboard (New Dashboard), no qual informamos o nome (Title) e uma breve descrição (Description).

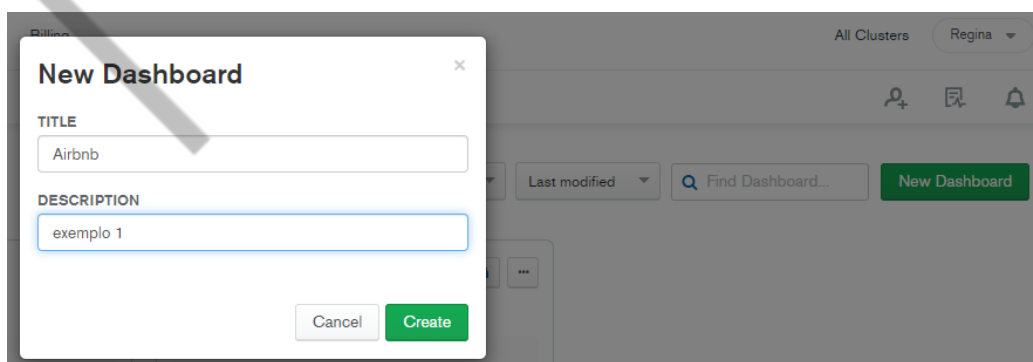


Figura 4.53 – Criar um dashboard
Fonte: Elaborado pelo autor (2020)

Um novo ambiente é apresentado e no canto superior direito escolhemos/ assinalamos/confirmamos o Data Source `sample_airbnb.listingsAndReviews`.

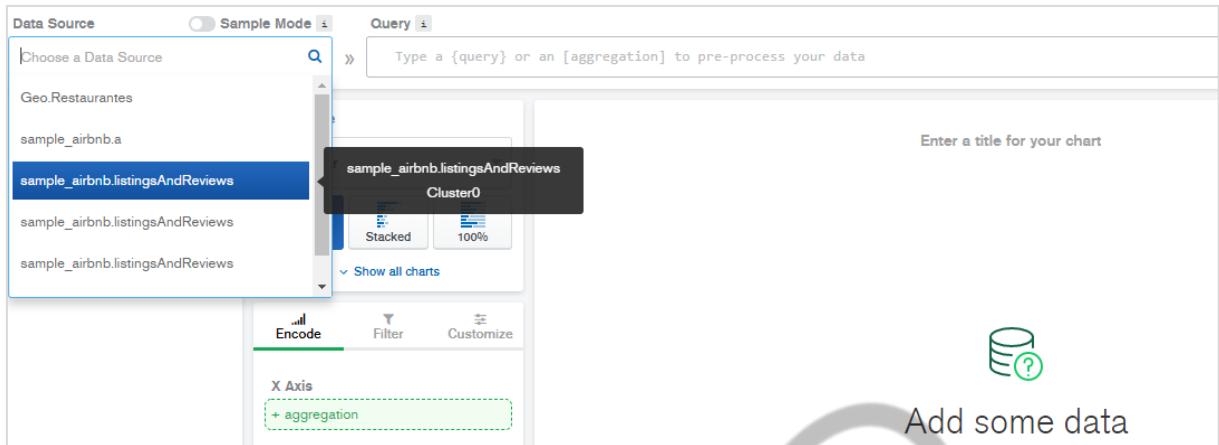


Figura 4.54 – Chart x Data Source
Fonte: Elaborado pelo autor (2020)

Agora vamos construir um Chart para mostrar o total de locações por bairro no Brasil. Podemos verificar os metadados da coleção no lado esquerdo, os diversos tipos de chart no centro e uma área à esquerda para mostrar o resultado de nossa exploração. Para construirmos a visualização, realizamos a seguinte sequência de comandos:

- Add Chart e selecionar Airbnb dataset como datasource.
- Chart Type selecionar Bar/Stacked.
- Na barra Filter, arrastar o metadado country presente no address . Ao fazer isso, uma lista de países existentes na coleção será mostrada. Assinalar Brazil.

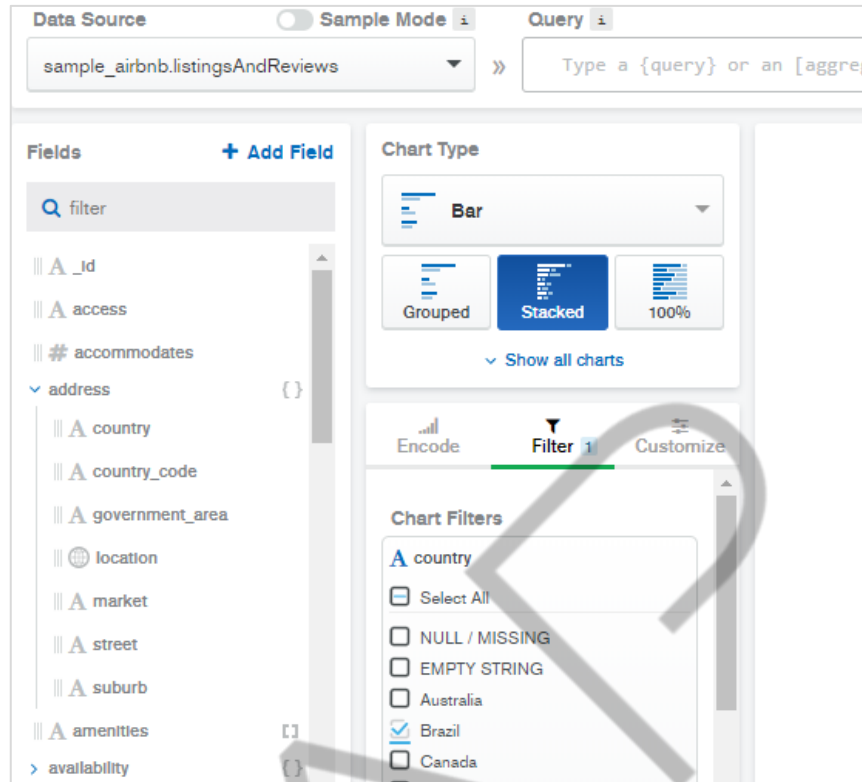


Figura 4.55 – Explorando Airbnb
Fonte: Elaborado pelo autor (2020)

- Na barra Encode, arrastamos os metadados a serem visualizados no eixo X e Y, a operação a ser realizada sobre eles:
 - X Axis: _id, Count aggregation.
 - Y Axis: address.suburb.
- Sort By: Aggregated Value, Descending.
- Limit: 30.
- Series: property_type.
- Alterar nomes dos eixos e legenda.

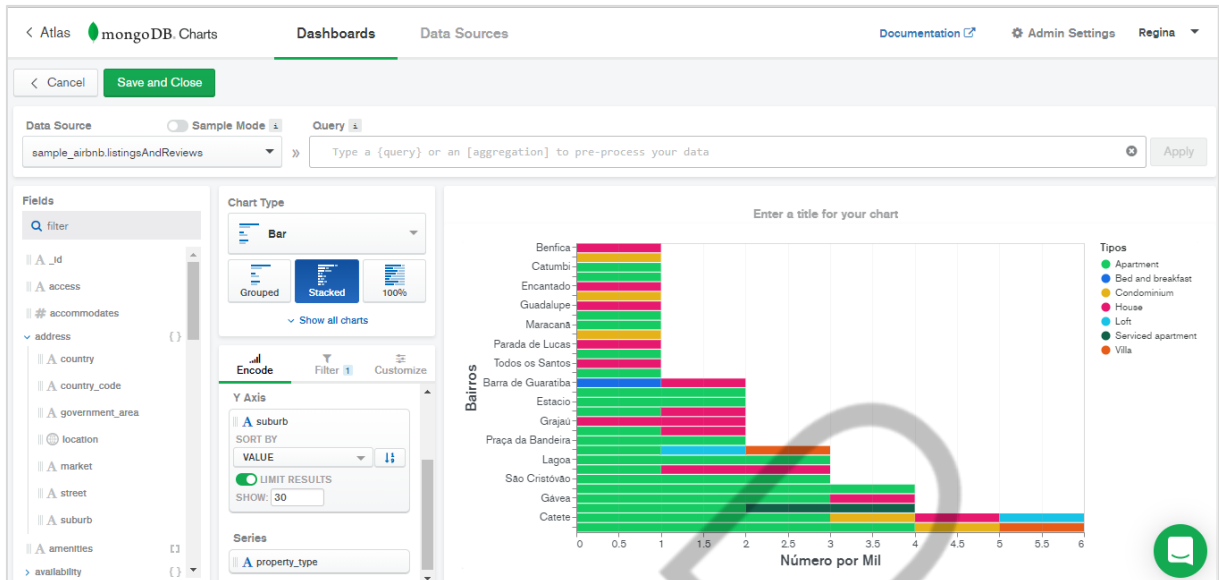


Figura 4.56 – Airbnb Total de locações no Brasil
Fonte: Elaborado pelo autor (2020)

Da mesma forma, podemos construir um chat para visualizar os bairros com as propriedades com valor de locação maior e correlacionar com as avaliações realizadas. Será que as locações mais caras recebem as melhores avaliações?

- Add Chart e selecionar Airbnb dataset como datasource.
- Chart Type select Bar/Stacked
- Filter Brazil
- Encode:
 - X Axis: _id, Count aggregation.
 - Y Axis: address.suburb.
 - Sort By: Aggregated Value, Descending.
 - Limit: 30.
 - Series: property_type.
 - Alterar nomes dos eixos e legenda.

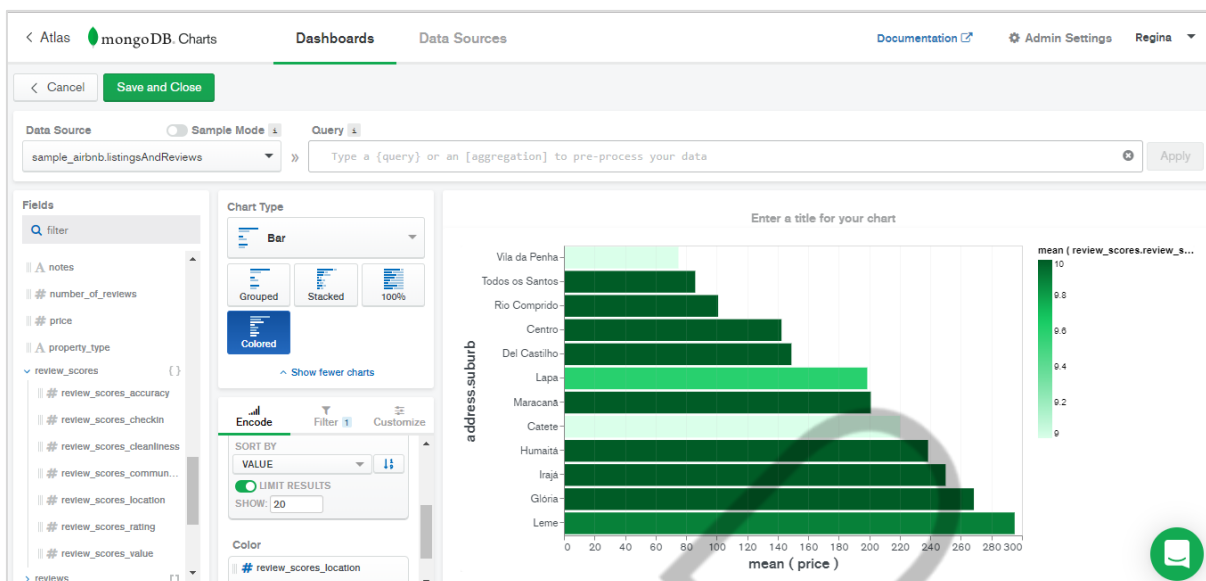


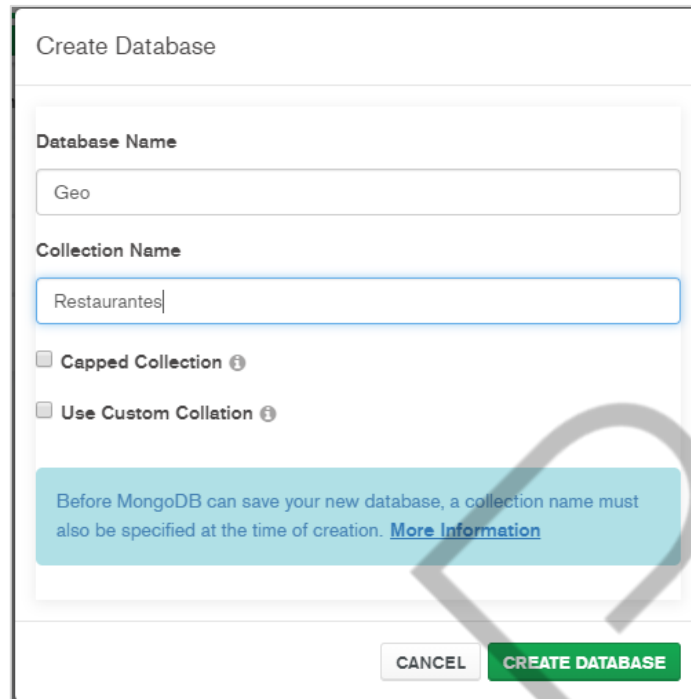
Figura 4.57 – Airbnb Valor Locação x Avaliação
Fonte: Elaborado pelo autor (2020)

4.43 Restaurantes de Nova York

Neste exemplo, vamos explorar o MongoDB Compass para criar uma nova base de dados no nosso cluster criado no MongoDB Atlas, carregar uma coleção a partir de um arquivo Json, explorar seus metadados e criar visualizações com o MongoDB Chart.

4.43.1 Criar um banco de dados e coleção

No Compass conectado com no cluster Atlas, clicar no botão CREATE DATABASE e informar o nome do banco de dados (Database Name) e o nome da coleção (collection name). Neste exemplo, GEO e Restaurantes respectivamente.



Create Database

Database Name

Geo

Collection Name

Restaurantes

☐ Capped Collection ⓘ

☐ Use Custom Collation ⓘ

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

CANCEL CREATE DATABASE

Figura 4.58 – Create Database
Fonte: Elaborado pelo autor (2020)

4.43.2 Importar arquivo

O arquivo Json com restaurantes a ser utilizado está disponível em <https://github.com/OpenKitten/Mongo-Assets/blob/master/primer-dataset.json> (File > Save Page As no Chrome). O conjunto de dados é de 11,3 MB e tem cerca de 25 mil restaurantes.

Navegar na coleção de Restaurantes e selecionar Import To Collection no menu. Na sequência, selecionar o arquivo json baixado, selecionar o tipo e observar os metadados encontrados com seus respectivos tipos. No final, escolher importar. Ao final, aparece uma mensagem de 25.359 documentos importados na coleção Restaurantes.

Import To Collection Geo.Restaurantes

Select File

C:\bkpgina\fiap\ABD\9ABD\primer-dataset.json [BROWSE](#)

Select Input File Type

JSON CSV

Options

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	address.building	address.coord	address.street	address.zipcode
1	1007	[-73.856077, 40.848447]	Morris Park Ave	10462
2	469	[-73.961704, 40.662942]	Flatbush Avenue	11225
3	351	[-73.98513559999999, 40.7676919]	West 57 Street	10019
4	2780	[-73.98241999999999, 40.579505]	Stillwell Avenue	11224
5	97-22	[-73.8601152, 40.7311739]	63 Road	11374

Figura 4.59 – Importar coleção restaurantes
Fonte: Elaborado pelo autor (2020)

4.43.3 Explorar atributos

No Compass observar como os documentos da coleção Restaurantes têm subdocumentos aninhados como address.

```

_id: ObjectId("5eb31f3e494c514528cbd862")
address: Object
  building: "1007"
  coord: Array
    0: -73.856077
    1: 40.848447
  street: "Morris Park Ave"
  zipcode: "10462"
  borough: "Bronx"
  cuisine: "Bakery"
grades: Array
  0: Object
  1: Object
  2: Object
  3: Object
  4: Object
name: "Morris Park Bake Shop"
restaurant_id: "30075445"

```

Figura 4.60 – Explorando metadados
Fonte: Elaborado pelo autor (2020)

Na aba Schema, selecionar Análise. Além de fornecer nomes e tipos de campo, o Compass também fornecerá um resumo dos valores dos dados. Por exemplo, para a culinária, podemos ver que o chinês é o terceiro mais comum em 7%.

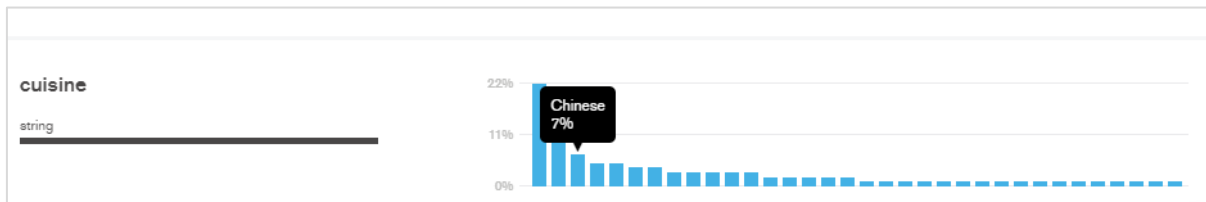


Figura 4.61 – Compass explorando metadados

Fonte: Elaborado pelo autor (2020)

4.43.4 Explorar graficamente dados de geolocalização

Expandir o campo de endereços para visualizar o mapa e dar um zoom em Nova York (NYC).

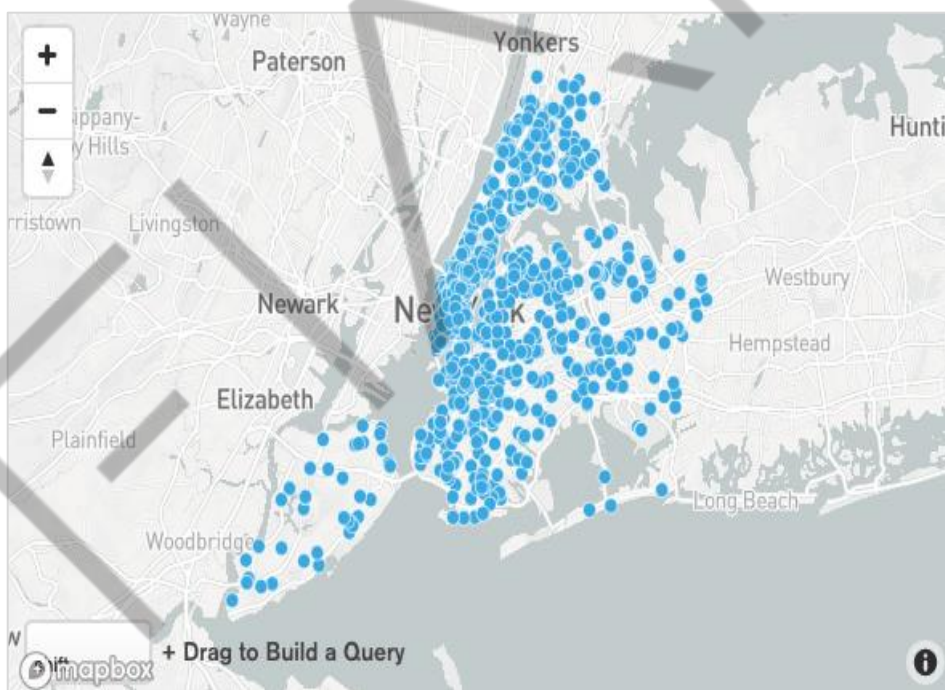


Figura 4.62 – Compass com coordenadas geolocalizadas

Fonte: Elaborado pelo autor (2020)

4.43.5 Utilizar MQL

O Analisador Schema no Compass fornece uma maneira fácil de aprender e utiliza MongoDB Query Language (MQL). Selecionar um bairro específico (borough) como “Staten Island” e os restaurantes chineses presentes nele (cuisine: chinês).

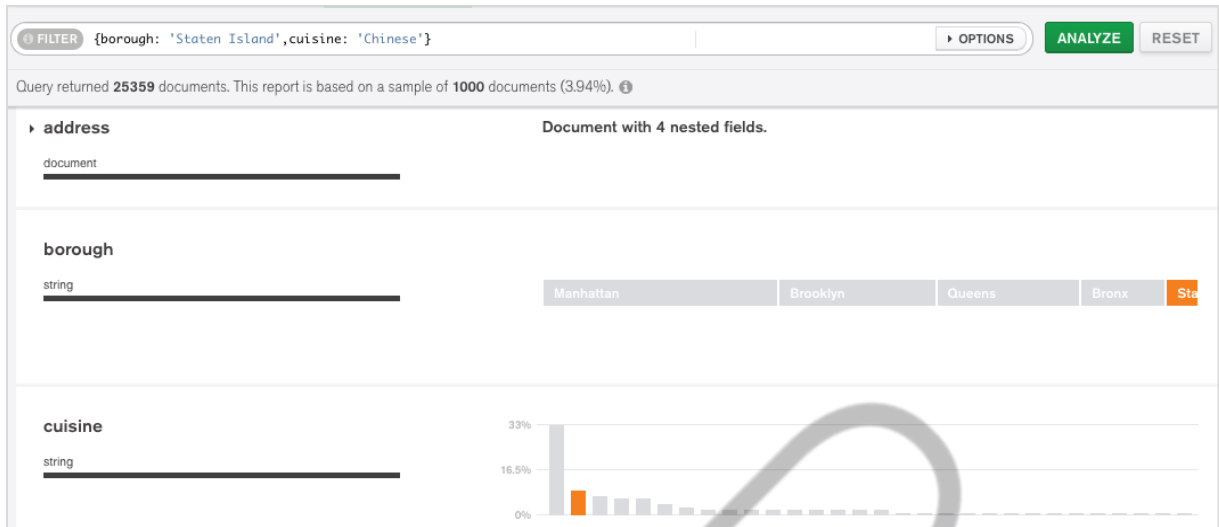


Figura 4.63 – MQL
Fonte: Elaborado pelo autor (2020)

Clicar no botão ANALYZE para filtrar para restaurantes chineses em Staten Island. E agora você pode ver isso refletido no mapa com os 88 restaurantes resultantes.



Figura 4.64 – Compass visualização no mapa
Fonte: Elaborado pelo autor (2020)

Para realizar uma consulta geoespacial, deslocar, clicar e arrastar o círculo no mapa – canto superior um círculo ou um polígono. Uma vez que o círculo esteja no lugar, ele pode ser redimensionado:

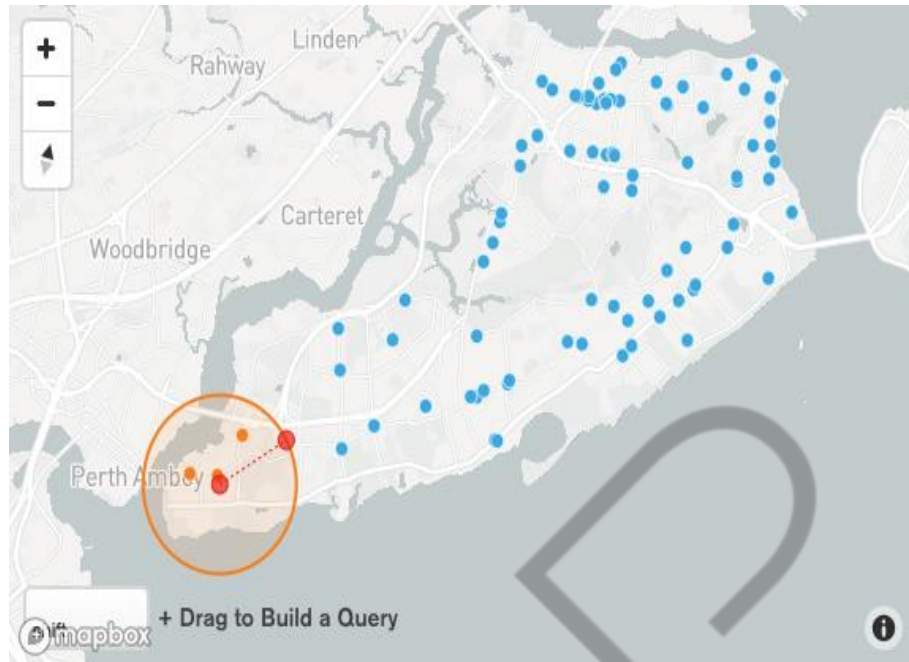


Figura 4.65 – Explorando mapas
Fonte: Elaborado pelo autor (2020)

E observar o filtro `$geoWithin` adicionado na consulta:



Figura 4.66 – MQL resultante
Fonte: Elaborado pelo autor (2020)

Clicar em ANALYZE novamente e visualizar os restaurantes chineses no raio selecionado em Staten Island.

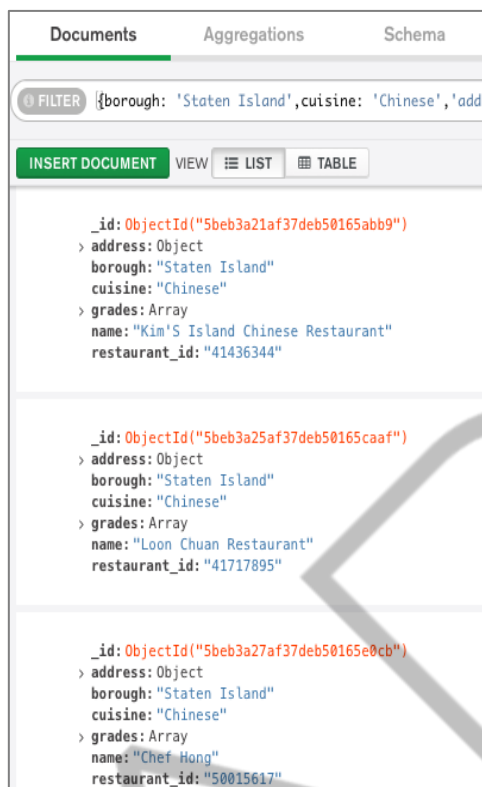


Figura 4.67 – MQL documentos resultantes
Fonte: Elaborado pelo autor (2020)

4.43.6 Visualizar dados no MongoDB Charts

Como realizado anteriormente, vamos criar um Dashboard, associar o data source (Restaurantes), conectar e em seguida Set Permissions / Publish Data Source.

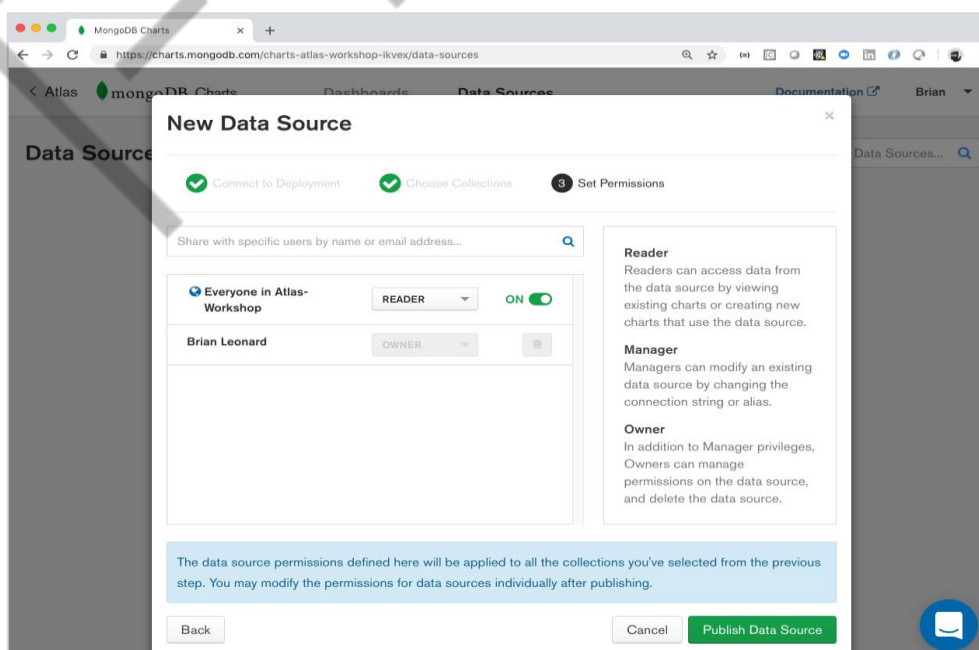


Figura 4.68 – Dashboard no Chart

Fonte: Elaborado pelo autor (2020)

Adicionar um gráfico simples. Definir o tipo de chart como texto. Em seguida, arrastar bairro e culinária para a categoria Grupos.

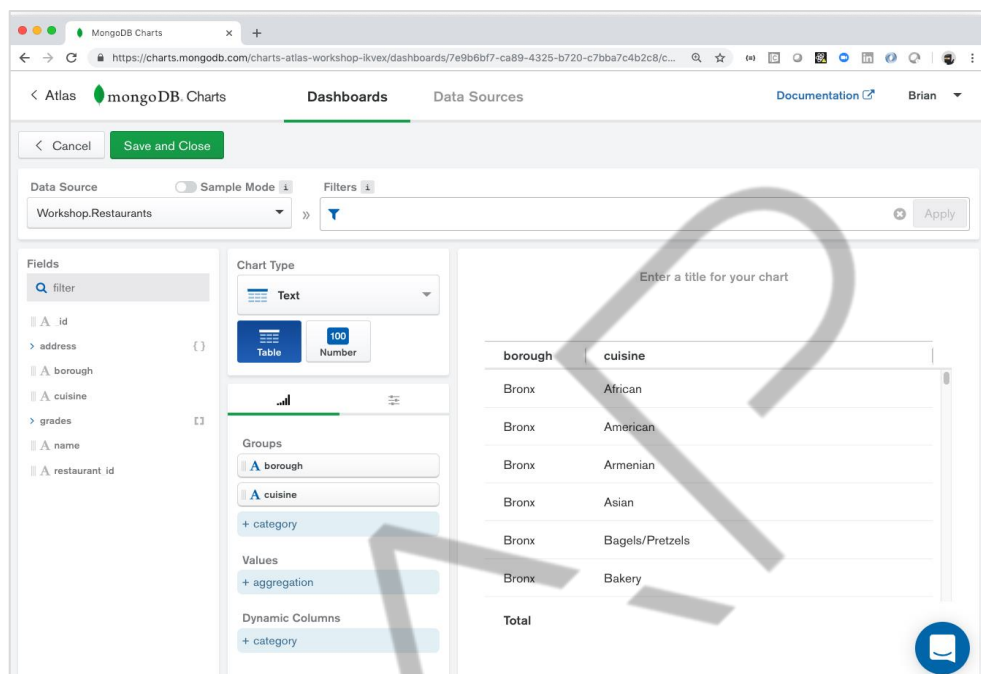


Figura 4.69 -- Restaurantes por bairro e cozinha

Fonte: Elaborado pelo autor (2020)

Em seguida, arrastar a culinária para a agregação valores.

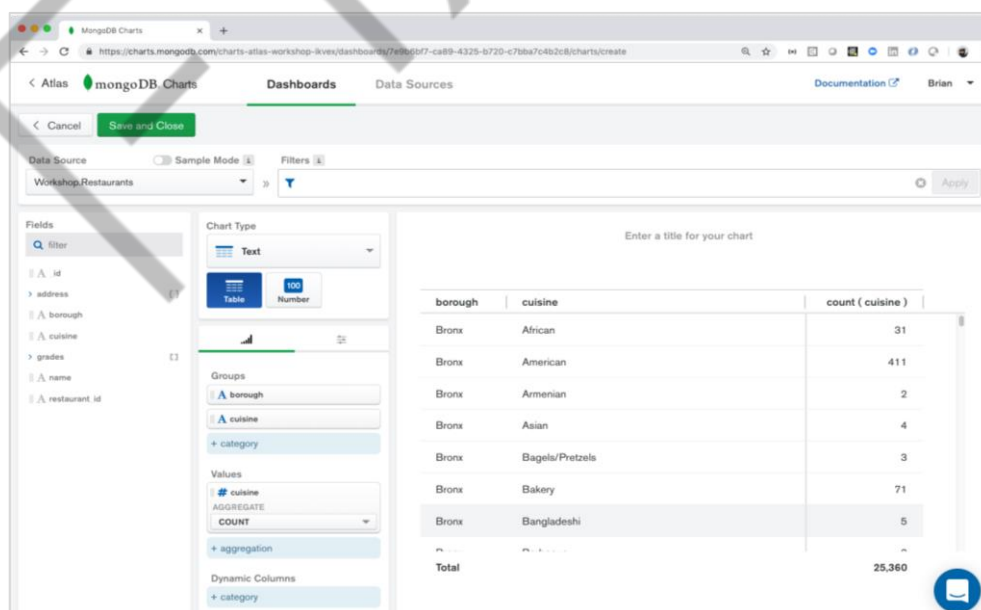


Figura 4.70 – Total de restaurantes por bairro e cozinha

Fonte: Elaborado pelo autor (2020)

Alterar o título do gráfico Total por Culinária e Bairro, em seguida, Salvar e Fechar.

EMSE

REFERÊNCIAS

HOWS, D.; MEMBREY, P.; PLUGGE, E. **Introdução ao MongoDB**. São Paulo: Novatec, 2015.

MONGODB. **The MongoDB 4.4 Manual**. 2020 (a). Disponível em: <<https://docs.mongodb.com/manual/>>. Acesso em: 01 jul. 2020.

MONGODB. **MongoDB Atlas**. 2020 (b). Disponível em: <<https://www.mongodb.com/cloud/atlas/>>. Acesso em: 01 jul. 2020.