UERJ – Universidade do Estado do Rio de janeiro

Banco de Dados 2

2020-2

Avaliação 2 - *Entrega PDF*

Professora:

Ana Carolina Brito de Almeida

Alunos:

Breno Antunes

Carolina Carvalhosa Simões

Douglas Fernandes

Thalles Cotta Fontainha

“ Não há bem que sempre dure, nem mal que nunca se acabe…”

(Ditado Popular)

➢ Divisão do Trabalho

Como foi solicitado que deixássemos explícito a divisão dos tópicos-alunos, criamos essa página indicando através de tabelas nos próprios tópicos principais, quem foram os alunos responsáveis por cada assunto. Também deixaremos em cada tópico oficial (conforme PDF de instruções) com (➢) identificando assim os tópicos organizados pela disciplina e sub tópicos que julgamos complementar a explicação desses assuntos. *Reforçamos aqui este ser um trabalho em grupo e todos colaboraram para a melhor performance do mesmo.* A documentação oficial do MongoDB está toda em inglês e na conversão tentamos tomar o cuidado na tradução e adaptação às nossas palavras para melhor explicar os tópicos cobrados nesta avaliação.

(TEORIA)

➤**Teoria**: Descrição do BD e sua estrutura(relacional, orientado a colunas, documentos, XML etc);
Breno Antunes

O MongoDB, como os demais bancos NoSQL, foi desenvolvido para resolver problemas específicos e um deles é a capacidade de armazenar e recuperar grande volume de dados. Entretanto, para lidar com ele foi aberto mão de alguns recursos, sendo um deles a normalização. Muito se ouve falar da desnormalização para melhorar o desempenho em consultas em grandes volumes de dados, principalmente quando o assunto é Data Warehouse e Big Data. O fato de não trabalhar com dados normalizados é um dos fatores que colaboram com o desempenho de bancos NoSQL.

O primeiro passo para se modelar para MongoDB é pensar na sua arquitetura de forma única, como um único objeto, e se necessário sub-objetos relacionados a este. Todos os dados necessários devem ser retornados em uma única consulta. Como seu armazenamento é realizado através de um documento JSON, fica fácil adicionar qualquer estrutura em uma única formação.

➤ Descrição do BD

Banco de Dados Orientado a Documentos

A definição geral apresentada é que os Bancos de Dados orientados a Documentos utilizam o conceito de dados e documentos autocontidos e auto descritivos, e isso implica que o documento em si já define como ele deve ser apresentado e qual é o significado dos dados armazenados na sua estrutura.

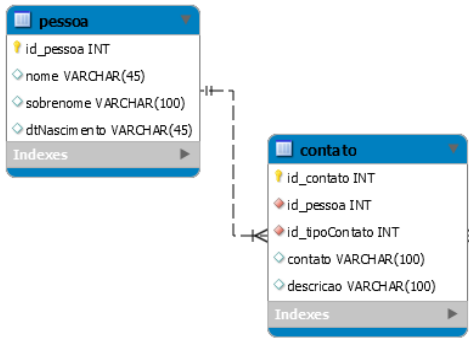
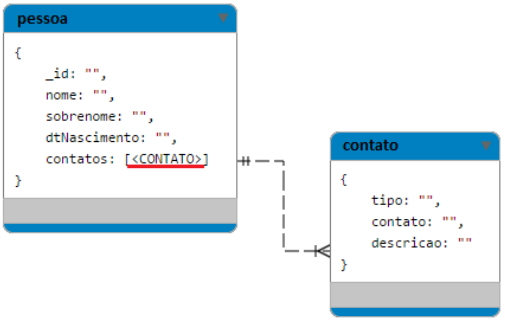
Banco de Dados Orientados a Documentos tem como característica conter todas as informações importantes em um único documento, ser livre de esquemas, possuir identificadores únicos universais (UUID), possibilitar a consulta de documentos através de métodos avançados de agrupamento e filtragem (MapReduce) e também permitir redundância e inconsistência.

➢ Sua estrutura (relacional, orientado a colunas, documentos, XML etc);

Conceito da modelagem para MongoDB

Modelar uma estrutura para o MongoDB é mais simples do que uma modelagem para um banco de dados relacional, porém esse trabalho pode ficar complicado se pensarmos de forma relacional. Um dos maiores problemas enfrentados pelas pessoas ou empresas que adotam o MongoDB é trocar de banco pelas vantagens oferecidas pela tecnologia, porém ainda trabalhar com as convicções e conhecimentos absorvidos ao longo do desenvolvimento com bancos de dados relacionais.

No MongoDB não é necessário se preocupar com os *"tipos de dados"* e o espaço para representar um atributo. Outra grande vantagem é que, por não existir um relacionamento, não existe a necessidade de informações pré-cadastradas. Com isso, podemos dizer que o MongoDB trabalha sob demanda, um atributo é sempre adicionado quando existir um real uso, um valor será preenchido quando for de fato utilizado. Isso poupa tempo na modelagem, visto que precisamos nos preocupar com o que de fato iremos utilizar, ou seja, o que a aplicação irá apresentar.

Figura 1. Modelagem banco relacional para armazenar contatos.	**Figura 2**. Modelagem no MongoDB para armazenar contatos.
	
https://arquivo.devmedia.com.br/REVISTAS/sql/imagens/147/modelagem/modelagem_mongo.png	

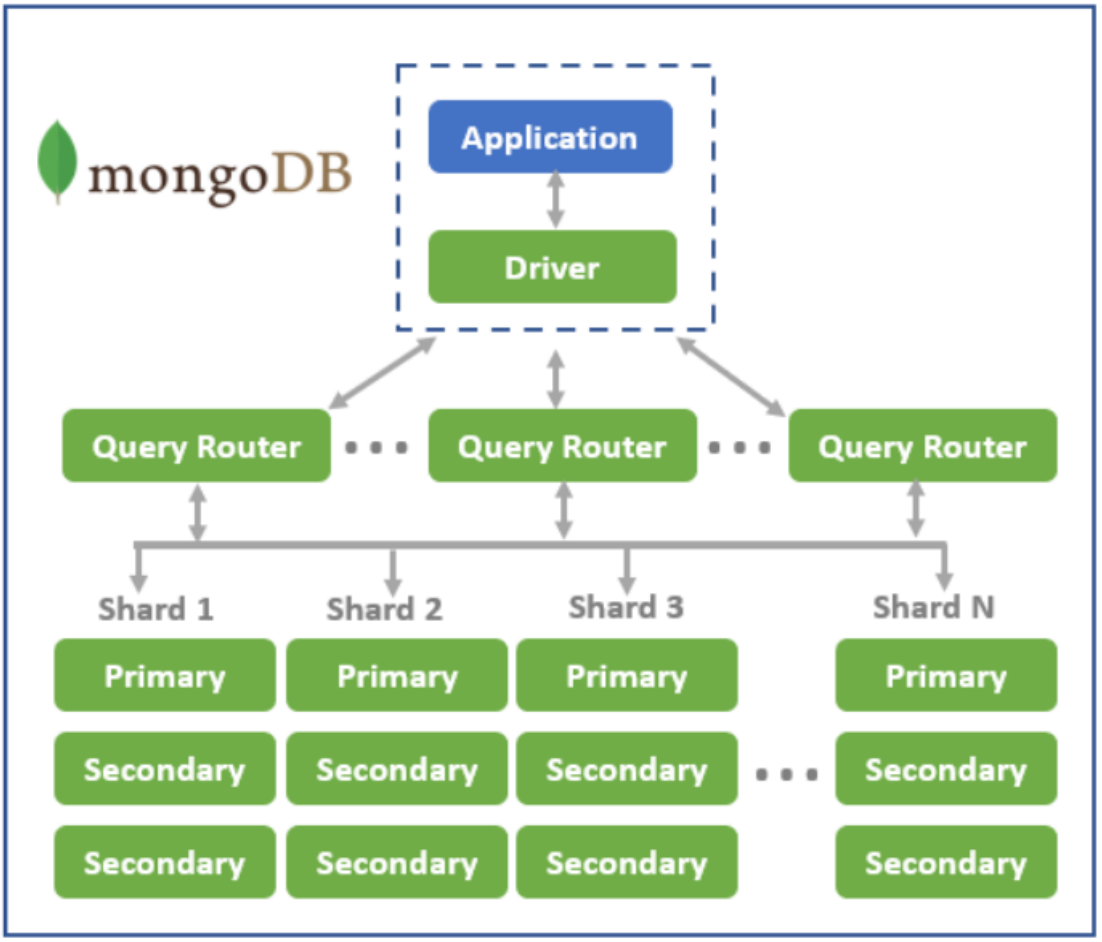
Acima, mostramos a diferença, num caso hipotético, na modelagem em um banco relacional (como MySQL, por exemplo) e como esses mesmos dados ficam representados no modelo MongoDB. Em um primeiro momento pode ser um pouco confuso, mas é muito mais simples que uma abordagem tradicional. Um detalhe é que o id é definido automaticamente pelo banco de dados, sendo um identificador único e não uma sequência. Observe na figura 2, que a estrutura é composta por dois objetos, um objeto principal que é o de "*pessoa*" com seus devidos atributos e um segundo objeto, "*contato"*, sendo esse objeto adicionado dentro do atributo contatos da pessoa em um array de objetos (uma pessoa pode ter vários contatos). Não pense na "*pessoa*" com relacionamento com o contato, mas sim que dentro da estrutura de pessoa contém 0 ou N estruturas que caracterizam o contato.

O ponto importante que queremos explicar, é que **deve-se evitar relacionamentos entre documentos diferentes.** Apesar dos avanços neste sentido nas últimas versões do MongoDB, este ainda é um banco não-relacional e, portanto, não faz sentido algum modelá-lo pensando em relacionamentos.

Os recursos comuns do MongoDB

- O design do modelo de dados reduz a necessidade de junções e fornece fácil evolução do esquema.
- Alto desempenho, pois não contém junção nem transações que fornecem acesso rápido e, portanto, o desempenho é aumentado.
- Alta disponibilidade devido à incorporação de conjuntos de réplicas que são capazes de fornecer backup durante falhas e também são altamente robustos.
- Facilidade de escalabilidade.
- A propriedade sharding do MongoDB permite um desempenho rápido e eficiente nas funções distribuídas. Isso também é possível, pois oferece suporte ao dimensionamento horizontal de dados.
- A linguagem é altamente rica na consulta. O MongoDB tem sua própria linguagem de consulta, chamada linguagem de consulta Mongo, que pode substituir as SQL. Da mesma forma, funções utilitárias e mapear ou reduzir podem substituir funções agregadas complicadas.

➢ **Teoria**: Arquitetura (Desenho da arquitetura dos componentes do BD);
Breno Antunes



(Imagem:https://severalnines.com/sites/default/files/blog/node_5989/image2.png)

Visão geral do MongoDB

A **arquitetura** do MongoDB fornece uma grande coleção de conjuntos de réplicas (*replica set*), onde cada conjunto pode conter mais de uma cópia de dados. Nos conjuntos de réplicas, todas as funções primárias (*leitura e gravação*) são executadas no conjunto primário, enquanto os conjuntos secundários são usados em caso de falha do anterior. O MongoDB incorpora *sharding*, que faz uso do processo de escalonamento horizontal. A propriedade de balanceamento de carga deste banco de dados de armazenamento de documentos é justificada pelo fato de ser executado em vários servidores, proporcionando a duplicação de dados e o balanceamento da carga. Em troca, ele também fornece backup durante a falha de hardware. Ele também faz uso de um sistema de arquivos em grade que divide o arquivo específico em diferentes partes e as armazena separadamente.

O MongoDB *sharded cluster* consiste nos seguintes componentes:

- ***Shard***: Cada *shard* contém um subconjunto dos dados fragmentados. Cada shard pode ser implementado como um conjunto de réplicas.
- ***Mongos***: o "*mongos"* atua como um roteador de consulta, fornecendo uma interface entre os aplicativos cliente e o cluster shard. A partir do MongoDB 4.4, os *mongos* podem oferecer suporte a "*Hedged Reads"* * para minimizar as latências.

  * A partir da versão 4.4, as instâncias do "*mongos*" podem evitar leituras que usam preferências de leitura não primárias. Com leituras protegidas, as instâncias *mongos* roteiam operações de leitura para dois membros do conjunto de réplicas por cada *shard* consultado e retornam resultados do primeiro *shard* que responde. *(https://docs.mongodb.com/manual/core/sharded-cluster-query-router/#std-label-mongos-hedged-reads)*

- ***Config Servers***: os servidores de configuração armazenam metadados e definições de configuração para o cluster.

Referências: (https://docs.mongodb.com/manual/sharding/)

Sharding (Fragmentação)

Sharding é um método para distribuir dados em várias máquinas. O MongoDB usa *sharding* para dar suporte a implantações com conjuntos de dados muito grandes e operações de alto rendimento.

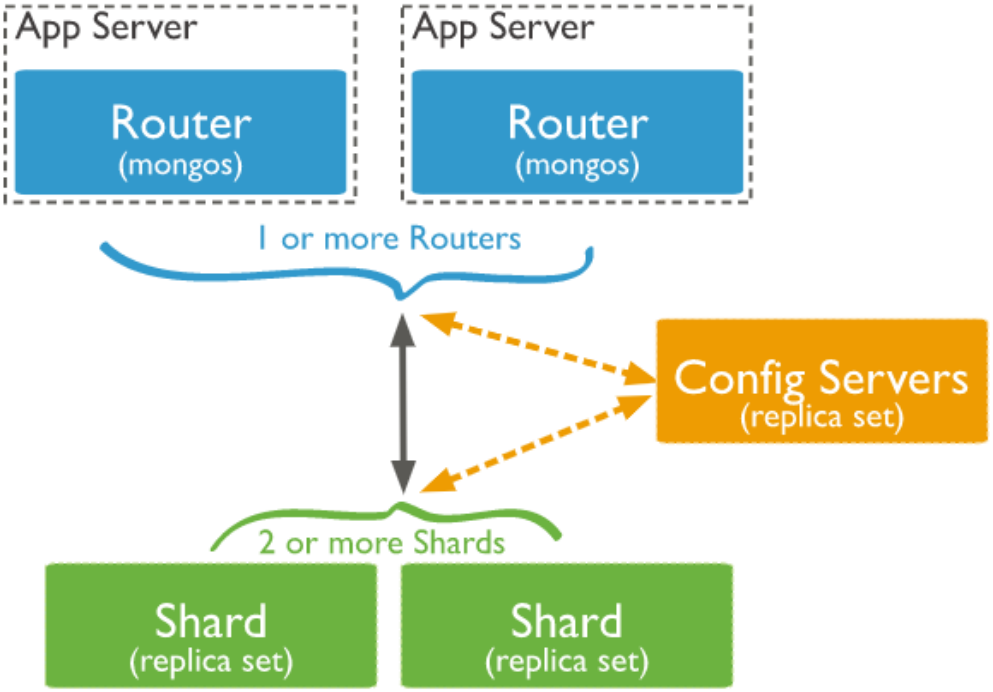
Os sistemas de banco de dados com grandes conjuntos de dados ou aplicativos de alto rendimento podem desafiar a capacidade de um único servidor. Por exemplo, altas taxas de consulta podem esgotar a capacidade da CPU do servidor. Os tamanhos de conjuntos de trabalho maiores que a RAM do sistema sobrecarregam a capacidade de I/O das unidades de disco.

O MongoDB oferece suporte ao ***escalonamento horizontal*** por ***sharding***. O escalonamento horizontal envolve a divisão do conjunto de dados do sistema e a carga em vários servidores, adicionando servidores adicionais para aumentar a capacidade conforme necessário. A desvantagem é o aumento da complexidade em infraestrutura e manutenção para a implantação.

Referências, adaptações e traduções: (https://laptrinhx.com/the-battle-of-the-nosql-databases-comparing-mongodb-and-couchdb-176648253/)

➢ **Teoria**: descrever como ocorre o processamento de consultas no BD, componente envolvidos;
Carolina Carvalhosa

O processamento de consultas no MongoDB funciona da seguinte forma:



(Imagem:https://docs.mongodb.com/manual/sharding/#sharded-cluster/)

Router (*mongos*): Instâncias *mongos* do MongoDB funcionam como uma interface e uma porta de entrada para clusters do MongoDB. As aplicações se conectam aos *mongos* que roteiam consultas e gravam operações em shards que ficam dentro dos clusters.

(Traduzido e apatado: https://docs.mongodb.com/manual/core/sharded-cluster-query-router/)

Como ocorre o processamento de consultas

Os *mongos* rastreiam quais dados estão em quais shards para determinar a lista de shards que irão receber a consulta e armazenam em cache os metadados dos servidores de configuração. Após isso, os *mongos* usam os metadados para estabelecer um cursor que passa por todos os shards e roteia operações de aplicativos e clientes para as instâncias *mongod*.

(Traduzido e adaptado: https://docs.mongodb.com/manual/core/sharded-cluster-query-router/)

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

(Traduzido: https://docs.mongodb.com/manual/reference/program/mongod/#mongodb-binary-bin.mongod)

Como os *mongos* manipulam modificadores de consulta:

- **Find**: Se nossa consulta incluir a chave de shard ou um prefixo da chave de shard, os *mongos* realizam uma operação direcionada, apenas consultando os shards que contêm as chaves que estamos procurando. Por exemplo, com uma chave de shard composta de *{_id, email, address}* em nossa coleção, podemos ter uma operação direcionada com qualquer uma das seguintes consultas:

```
db.User.find({_id: 1})
db.User.find({_id: 1, email: 'carolina.simoes@rainhadomongo.com.br'})
db.User.find({_id: 1, email: 'carolina.simoes@rainhadomongo.com.br',
address: 'Rua A'})
```

Todos os três são um prefixo (os dois primeiros) ou a chave de shard completa. Por outro lado, uma consulta em *{email, address}* ou *{address}* não será capaz de direcionar os shards corretos, resultando em uma operação de broadcast. Uma operação de broadcast é qualquer operação que não inclui a chave de shard ou um prefixo da chave de shard e resulta em *mongos* consultando cada shard e reunindo resultados deles. Também é conhecido como operação de dispersão e coleta ou consulta fanout.

- **Sort**: Se quisermos ordenar nossos resultados, existem duas opções:
    - Se estivermos usando a chave de shard em nossos critérios de ordenação, os *mongos* podem determinar a ordem em que ele deve consultar o(s) shard(s). Isso resulta em uma operação eficiente e direcionada.
    - Se não estivermos usando a chave de shard em nossos critérios de ordenação, então, como acontece com uma consulta sem ordenação, será uma consulta fanout. Para classificar os resultados quando não estamos usando a chave de shard, o shard primário executa uma classificação de mesclagem distribuída localmente antes de passar o conjunto de resultados classificado para *mongos*.

- **Limit:** O limite de consultas é aplicado em cada shard individual e novamente no nível dos *mongos*, pois pode haver resultados de vários shards.
- **Skip:** Não pode ser passado para shards individuais e será aplicado por *mongos* após recuperar todos os resultados localmente.
- **Update/Remove**: Em operações de modificador de documento, como atualizar e remover, temos uma situação semelhante a ser encontrada. Se tivermos a chave de shard na seção de procura do modificador, os *mongos* podem direcionar a consulta para o shard relevante. Se não tivermos a chave de shard na seção de procura do modificador, então será novamente uma operação fanout.

(Traduzido e adaptado: https://hub.packtpub.com/how-to-query-sharded-data-in-mongodb/)

➢ **Teoria**: descrever como ocorre a otimização de consultas no BD, componente envolvidos
Carolina Carvalhosa

Índices

Os índices melhoram a eficiência das operações de leitura, reduzindo a quantidade de dados que as operações de consulta precisam processar. Isso simplifica o trabalho associado às consultas no MongoDB.

Se consultarmos uma coleção em um determinado campo ou conjunto de campos, um índice no campo consultado ou um índice composto no conjunto de campos pode impedir a consulta de varrer toda a coleção, para localizar e retornar os resultados da consulta, melhorando assim, a performance.

Sintaxe de criação de índice:

```
db.collection.createIndex(<key and index type specification>, <options>)
```

(Traduzido e apatado: https://docs.mongodb.com/manual/indexes/)

Seletividade de consulta

A seletividade de consulta se refere a quão bem o predicado de consulta exclui ou filtra documentos em uma coleção. A seletividade da consulta pode determinar se as consultas podem ou não usar índices de forma eficaz. Um exemplo de uma consulta seletiva é uma correspondência de igualdade no campo *_id* exclusivo, pois pode-se corresponder a no máximo um documento:

```
db.users.find({ _id: 2 })
```

(Traduzido e apatado: https://docs.mongodb.com/manual/core/query-optimization/)

Consulta coberta

Uma consulta coberta é uma consulta que pode ser satisfeita inteiramente usando um índice. Um índice cobre uma consulta quando todos tópicos abaixo se aplicam:

- Todos os campos da consulta fazem parte de um índice;
- Todos os campos retornados nos resultados estão no mesmo índice;
- Nenhum campo na consulta é igual a nulo (ou seja, *{"field": null}* ou *{"field": {$ eq: null}}*);

Exemplo de uma criação e uso de um índice no MongoDB

Considerando a coleção de exemplo *records*:

```
{
  "_id": ObjectId("570c04a4ad233577f97dc459"),
  "score": 1034,
  "location": { state: "NY", city: "New York" }
}
```

A operação a seguir cria um índice crescente no campo de pontuação da coleção de registros:

```
db.records.createIndex( { score: 1 } )
```

O valor do campo na especificação do índice descreve o tipo de índice para esse campo. Por exemplo, um valor de 1 especifica um índice que ordena os itens em ordem crescente. Por exemplo, um valor -1 especificaria um índice que ordena os itens em ordem decrescente.

O índice criado suportará consultas que selecionam a pontuação do campo, como as seguintes:

```
db.records.find( { score: 2 } )
db.records.find( { score: { $gt: 10 } } )
```

(Traduzido e adaptado: https://docs.mongodb.com/manual/core/index-single/)

$gt: seleciona os documentos em que o valor do campo é maior que (ou seja,>) o valor especificado

(Traduzido: https://docs.mongodb.com/manual/reference/operator/query/gt/)

Analisando um plano de execução no MongoDB

Para retornar informações e estatísticas de execução dos planos de consulta e analisar se o index é realmente necessário, o MongoDB fornece:

- método *db.collection.explain()*,
- método *cursor.explain()* e
- comando *explain()*.

Os resultados do *explain* apresentam os planos de consulta como uma árvore de estágios.

Exemplo:

```
"winningPlan" : {
  "stage" : <STAGE1>,
  ...
  "inputStage" : {
     "stage" : <STAGE2>,
     ...
     "inputStage" : {
        "stage" : <STAGE3>,
        ...
     }
  }
},
```

(Traduzido e apatado: https://docs.mongodb.com/manual/reference/explain-results/)

winningPlan: Para uma consulta, o otimizador de consulta MongoDB escolhe e armazena em cache o plano de consulta mais eficiente de acordo com os índices disponíveis. A avaliação do plano de consulta mais eficiente é baseada no número de "unidades de trabalho" (trabalhos) realizadas pelo plano de execução da consulta quando o planejador de consulta avalia os planos candidatos.

(Traduzido e adaptado: https://docs.mongodb.com/manual/core/query-plans/)

Na parte prática do trabalho, iremos mostrar detalhadamente como funciona a análise do plano de execução vencedor e a decisão se o índice é realmente necessário para as consultas.

➢ **Teoria:** Descrever como ocorre o controle de transações no BD, confirmação, rollback, tipos de bloqueios, níveis de isolamento;
Thalles Cotta

Antes de mais nada, o termo "***Failover*"** (que irei comentar em alguns momentos seguintes), em computação, é a capacidade de alternar perfeita e automaticamente para um sistema de backup confiável. Ou seja, para servidores, a automação de *failover* inclui cabos de pulsação que conectam um par de servidores. O servidor secundário apenas descansa enquanto percebe que o pulso ou a pulsação contínua.

(Apadtado - https://pt.wikipedia.org/wiki/Toler%C3%A2ncia_a_falhas e https://www.infonova.com.br/gestao-de-ti/o-que-e-failover-importante/)

➢ *Controle de transações no MongoBD:*

Comparado com SGBD's onde se utiliza comandos de atualização da base como insert, update e delete para se gerar uma ou mais transações, a tecnologia NoSQL se faz semelhante no caso do MongoDB.

A diferença mais crucial desses modelos é o fato de os sistemas gerenciadores relacionais utilizarem os comandos commit e rollback, usados para confirmar ou desfazer uma ação realizada pelo usuário, respectivamente, o que não ocorre no NoSQL (FOWLER e SADALAGE, 2013).

O MongoDB não possui transações e por padrão os processos são retornados como bem-sucedidos. Existem duas alternativas para esse empecilho, o primeiro seria realizar as operações sendo atômicas, ou seja, de forma binária, a outra caso a primeira não tenha sido suficiente, pode ser utilizado método da gravação em duas fases (PIGA, 2013).

➢ *Confirmação:*

commitTransaction

Salva as alterações feitas pelas operações na transação de vários documentos e termina a transação. Para executar o *commitTransaction,* o comando deve ser executado no administrador do banco de dados e em um *Session()*. Em vez de executar o *commitTransaction* comando diretamente, a maioria dos usuários deve usar o método do driver ou o mongoshell *Session.commitTransaction( ) :*

O comando tem a seguinte sintaxe:

```
{
  commitTransaction: 1,
  txnNumber: <long>,
  writeConcern: <document>,
  autocommit: false,
  comment: <any>
 }
```

Comportamento

Ao confirmar a transação, a sessão usa a preocupação de gravação especificada no início da transação. Se você confirmar usando a "w: 1" preocupação de gravação, sua transação pode ser revertida se houver um *failover*.

Atomicidade

Quando uma transação é confirmada, todas as alterações de dados feitas na transação são salvas e ficam visíveis fora da transação. Ou seja, uma transação não irá confirmar algumas de suas alterações enquanto revertendo outras. Até que uma transação seja confirmada, as alterações de dados feitas na transação não são visíveis fora da transação. No entanto, quando uma transação grava em vários *shards*, nem todas as operações externas de leitura precisam esperar que o resultado da transação confirmada seja visível nos shards. Por exemplo, se uma transação é confirmada e a gravação 1 é visível no fragmento A, mas a gravação 2 ainda não é visível no fragmento B, uma leitura externa na preocupação da leitura "*local*" pode ler os resultados da gravação 1 sem ver a gravação 2.

(Aptado e traduzido manualmente - https://docs.mongodb.com/manual/reference/command/commitTransaction)

➤ *Rollback:*

Rollbacks durante o failover do conjunto de réplicas

Um rollback reverte as operações de gravação em um antigo primário quando o membro re-ingressa em seu conjunto de réplicas após um failover . O rollback é necessário apenas se o primário tinha operações de escrita aceite que os secundários *não tenha* replicado com sucesso antes das primárias desceu. Quando o primário reingressa no conjunto como secundário, ele reverte suas operações de gravação para manter a consistência do banco de dados com os outros membros.

Quando ocorre um rollback, geralmente é o resultado de uma partição de rede. Secundários que não conseguem acompanhar o rendimento das operações no antigo primário, aumentam o tamanho e o impacto da reversão.

Um *rollback* não ocorrerá se as operações de gravação replicarem para outro membro do conjunto de réplicas antes das etapas primárias para baixo *e* se esse membro permanecer disponível e acessível para a maioria do conjunto de réplicas.

(Aptado e traduzido manualmente - https://docs.mongodb.com/manual/core/replica-set-rollbacks/)

➤ *Tipos de bloqueios:*

Para garantir que as leituras nunca retornem dados que não estejam na mesma ordem causal do primário, o MongoDB bloqueia as leituras (no secundário), enquanto as entradas de *oplog** são aplicadas em lotes no secundário. [*O *oplog* (log de operações) é uma coleção limitada especial que mantém um registro contínuo de todas as operações que modificam os dados armazenados em seus bancos de dados.] *(*https://docs.mongodb.com/manual/core/replica-set-oplog/)*

Isso pode fazer com que leituras no secundário tenham latência variável, o que se torna mais evidente e perigoso, quando as cargas de gravação são mais intensas. *E por quê isso ocorre?*

Porque o MongoDB foi projetado para que cada um dos "*nós*" mostre as gravações na mesma ordem em que aconteceram no primário. Aproveitando as mudanças feitas para o suporte para transações, as leituras secundárias no MongoDB 4.0 não tem bloqueios! Tornando a latência de leitura previsível e mantendo uma visualização consistente dos dados. O maior benefício desta funcionalidade, é "*sentido*" nas cargas em lote e quando os clientes distribuídos estão acessando réplicas locais de baixa latência que são geograficamente remotas em relação à réplica primária.

(Referências: http://db4beginners.com/blog/mongodb-4-0-o-que-voce-precisa-saber/)

➤ *Níveis de isolamento:*

Quando uma única operação de gravação (por exemplo ***db.collection.updateMany()***) modifica vários documentos, a modificação de cada documento é atômica, mas a operação como um todo não é atômica. Ao realizar operações de gravação de vários documentos, seja por meio de uma única operação de gravação ou de várias operações de gravação, outras operações podem se intercalar.

Até que uma transação seja confirmada, as alterações de dados feitas na transação não são visíveis fora da transação.

No entanto, quando uma transação grava em vários *shards*, nem todas as operações externas de leitura precisam esperar que o resultado da transação confirmada seja visível nos *shards*.

Por exemplo, se uma transação for confirmada e a gravação 1 estiver visível no fragmento A, mas a gravação 2 ainda não estiver visível no fragmento B, uma leitura externa na preocupação de leitura "*local*" pode ler os resultados da gravação 1 sem ver a gravação 2.

Read uncommitted é o nível de isolamento padrão e se aplica a instâncias independentes do ***mongod***, bem como a conjuntos de réplicas e *clusters sharded.*

(Referências: https://docs.mongodb.com/manual/core/read-isolation-consistency-recency/**)**

O MongoDB não é um banco *ACID* (Atomicidade, Consistência, Isolamento, Durabilidade) deve usar um banco de dados compatível com o ACID.

Como para o isolamento real, atualmente MongoDB tem isolamento em um nível documento único com operações atômicas, como *$inc, $set, $unsete* todos os outros.

O isolamento não ocorre em vários documentos, há um operador $ *isolated* (http://docs.mongodb.org/manual/reference/operator/isolated/), mas é altamente recomendado não usá-lo, além de não ser compatível em *sharded collections*.

Há também uma página de documentação no MongoDB sobre como fornecer níveis de isolamento: (http://docs.mongodb.org/manual/tutorial/isolate-sequence-of-operations/), mas apenas a *findAndModify() e indexação* podem fornecer algum forma de isolamento pela qual outras consultas não interferirão.

Fundamentalmente, mesmo que tivesse operações atômicas em vários documentos, o MongoDB normalmente não pode suportar o isolamento em muitos documentos, isso se deve a um de seus principais recursos de simultaneidade (a capacidade de diminuir as operações fora da memória para as que estão na memória).

(Referências : Slide 20 do link:
https://pt.slideshare.net/imasters/sql-e-nosql-trabalhando-juntos-uma-comparao-para-obter-o-melhor-de-ambos-adamo-tonete
e https://stackoverflow.com/questions/18713392/how-efficient-is-mongo-db-isolation **)**

➢ **Teoria**: descrever como ocorre o controle de concorrência no BD;
Thalles Cotta

O MongoDB não possui controle de concorrência e gerenciamento de transações. Então, se um usuário lê um documento, escreve uma modificação e devolve ao banco de dados, pode acontecer de outro usuário escrever uma nova versão do documento entre o processo de leitura e escrita do primeiro. (POLITOWSKI; MARAN, 2014, p. 4)

MongoDB persiste documentos no formato BSON (*Binary JSON*), que são objetos JSON binários. BSON, por sua vez, suporta estruturas como *arrays* e *embedded objects* assim como JSON. MongoDB permite que usuários realizem modificações de apenas um atributo em um documento sem a necessidade de interação com o restante da estrutura.

Documentos podem ser armazenados em coleções (*collections*), onde serão efetuadas operações de busca (*queries*) e indexação (*indexing*). *Queries* são expressadas na sintaxe JSON e enviadas ao MongoDB como objetos BSON pelo *driver* de conexão ao banco.

Para persistência, MongoDB usa arquivos mapeados em memória, deixando o gerenciador de memória virtual do sistema operacional decidir quais partes vão para o disco e quais ficam na memória. Isto faz com que o MongoDB não tenha controle sobre o momento onde os dados são escritos no disco [Matthes and Orend 2010].

(Referências e citações, após cada parágrafo e

http://repositorioinstitucional.uea.edu.br/bitstream/riuea/2536/1/Desenvolvimento%20de%20dispositivo%20para%20o%20monitoramento%20e%20controle%20de%20estoque%20no%20processo%20de%20movimenta%C3%A7%C3%A3o%20de%20mat%C3%A9ria-prima%20em%20uma%20ind%C3%BAstria%20utilizando%20a%20plataforma%20arduino.pdf)

➢ **Teoria:** descrever sobre a segurança no BD, controle de acesso, concessão e revogação de privilégio, existência ou não de criptografia de dados;
Douglas Fernandes

➢ ***Descrever sobre a segurança no MongoBD:***

Autenticação

No MongoDB “*Autenticação*” e “*Autorização*” são dois processos diferentes.
Podemos descrever “Autenticação” como uma pergunta que o banco de dados nos faz: quem é você?
Por outro lado “Autorização” a pergunta é: o que você pode fazer?

O processo de autenticação verifica a identidade do cliente que está tentando se conectar com a instância do MongoDB. Uma vez habilitada, todos os clientes deverão passar por esse processo para determinar seu nível de acesso.

Métodos de autenticação

Para se autenticar no MongoDB, deve-se passar: usuário, senha e a base de dados de autenticação. Esse último parâmetro vale uma atenção especial:

Os usuários do MongoDB precisam "*residir*" em uma base de dados. Não existe uma obrigação para isso, porém, pode-se ter usuários que estão em bases de dados específicas, por isso sempre deve-se informar qual é a base de dados de autenticação. É possível utilizar duas maneiras diferentes através do shell do MongoDB:

- Utilizando a linha de comando da seguinte forma:

```
mongo --username <nome_de_usuario> --password <senha_do_usuario>
--authenticationDatabase <nome_da_base_de_dados_de_autenticacao>
```

Também é possível nos conectar numa instância mongod ou mongos, e utilizar o comando de autenticação *db.auth( )* contra a base de dados de autenticação. Os parâmetros desse comando são usuário e senha: *db.auth("usuario","senha")* Mecanismos de autenticação MongoDB suporta alguns mecanismos de autenticação:

- SCRAM (default)
- x.509 Certificate Authentication

Além desses, a versão Enterprise suporta os seguintes mecanismos:

- LDAP, e
- Kerberos

(Referências: https://laptrinhx.com/seguranca-no-mongodb-parte-1-354585455/)

➢ ***Controle de acesso:***

Como na maioria dos bancos de dados, o MongoDB utiliza *Role-Based Access Control* (RBAC) para gerenciar o seu acesso. Para um usuário podemos atribuir uma ou mais *roles* para determinar quais os recursos e operações ele pode acessar e realizar dentro do banco de dados. Uma vez habilitado, nada que estiver fora dessas *roles* será permitido ao usuário.

➢ ***Concessão e revogação de privilégio:***

Roles

Uma *role* dá privilégios para executar ações específicas num recurso. É considerado como recurso um banco de dados, uma coleção, um conjunto de coleções ou um cluster.

Cada privilégio pode ser especificado explicitamente nas *roles* do usuário, herdado de outra *role* ou os dois.

Herdando Privilégios

Um usuário pode incluir uma ou mais *roles* em sua definição, nesse caso o usuário herda todos os privilégios das *roles* incluídas. Ou seja, se estou criando uma *role* que tenha um privilégio específico como *read* por exemplo, e incluir uma *role* que tenha *write*, a nova *role* passa a ter acesso de *readWrite* por herança. Exemplo: Uma *role* criada no banco de dados **admin** pode herdar privilégios das *roles* de qualquer banco de dados.

➢ ***Existência ou não de criptografia de dados:***

Criptografia de ponta a ponta

Todo o tráfego da rede é criptografado usando a camada de segurança de transporte (TLS), com flexibilidade para configurar a versão mínima do protocolo TLS. A criptografia dos dados em repouso é automatizada usando volumes de armazenamento criptografados. A criptografia pode ser dividida em *"Em repouso"* e *"Em trânsito"*. A criptografia é fundamental para proteger o MongoDB.

Criptografia de disco

Os dados estão "em trânsito" ou "em repouso". Podem se criptografar um ou ambos no MongoDB. “Dados em trânsito” usam a criptografia SSL. “Dados em repouso” são dados armazenados no disco. A criptografia de dados em repouso normalmente se refere aos dados salvos em um local de armazenamento criptografado. Isso é para evitar o roubo por meios físicos e criar backups que são armazenados de maneira que não sejam facilmente lidos por terceiros.

(Apatado: https://dev.to/delbussoweb/segurana-no-mongodb)

➢ **Teoria**: descrever formas de backup e recuperação do BD, exemplificando com os comandos necessários;
Douglas Fernandes

➢ Descrever formas de backup:

Backup com Atlas

MongoDB Atlas, a opção de serviço MongoDB hospedado na nuvem, oferece dois métodos totalmente gerenciados para backups:

1. ***Backups em nuvem***, que utilizam a funcionalidade de instantâneo nativo do provedor de serviços em nuvem da implantação para oferecer opções robustas de backup. Os backups em nuvem fornecem: Instantâneos sob demanda , Backups contínuos em nuvem.
2. ***Backups Legados*** *(descontinuado)*.

Backup com MongoDB Cloud Manager ou Ops Manager

MongoDB Cloud Manager é um serviço hospedado de backup, monitoramento e automação para o MongoDB. O MongoDB Cloud Manager oferece suporte para backup e restauração de conjuntos de réplicas e clusters fragmentados do MongoDB a partir de uma interface gráfica de usuário.

(Adpatado e traduzido: https://docs.mongodb.com/manual/core/backups/)

Backup com mongodump

Uma das maneiras mais convencionais de fazer backup no mongo é usando o comando mongodump

Gramática

Para salvar todo o bando em um único arquivo .archive :

```
$ mongodump --archive="mongodump-test-db" --db=test
```

- **--archive:**
  nome do arquivo .archive que será gerado ao fim da execução do comando.
- **--db:**
  nome do banco a sofrer o backup.

Para salvar collections separadamente no diretório -o (output) :

```
$ mongodump -h sample.mongodbhost.com:27017 -d DATABASE_NAME -u USER_NAME -p
SAMPLE_PASSWORD -o ~/Desktop
```

- **-h:**
  endereço do servidor MongoDB
- **-d:**
  nome do banco a sofrer o backup
- **-u:**
  nome de usuário para autenticação e averiguação dos privilégios (somente um usuário com permissões suficientes poderá fazer o backup)
- **-p:**
  senha do usuário fornecido.
- **-o:**
  caminho na máquina local onde serão salvos os dados do backup.

➤Formas de recuperação do BD, exemplificando com os comandos necessários:

Para restaurar a partir de um arquivo .archive :

```
$ mongorestore --archive="mongodump-test-db" --nsFrom='test.*' --nsTo='ex.*'
```

- **--archive:**
  endereço onde o arquivo .archive foi armazenado (endereço do arquivo de backup)
- **--nsFrom:**
  recebe o nome de um banco e
- **--nsTo:**
  salva com um novo nome

Para restaurar a partir de uma pasta com arquivos de collections individuais :

```
$ mongorestore -h dbhost -d dbname --directoryperdb dbdirectory
```

- **-h:**

  endereço do servidor MongoDB onde

- **-d:**

  necessidade de restaurar a instância de banco de dados

- **--directoryperdb:**

  diretório onde estão localizadas as *collections* geradas pelo mongodump.

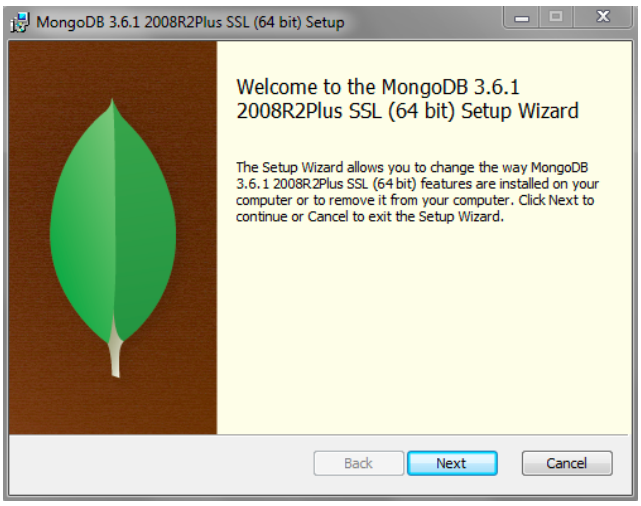
(Referência: http://www.w3big.com/pt/mongodb/mongodb-mongodump-mongorestore.html)

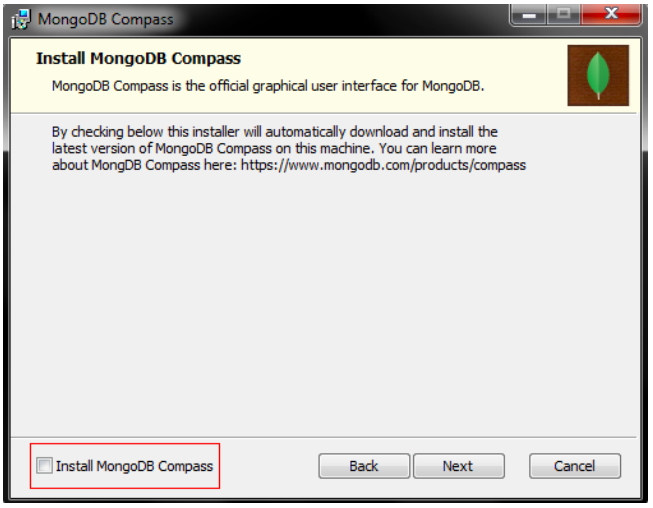
(PRÁTICA)

➢ Instalação do BD localmente;
Breno Antunes

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

Primeiramente é preciso baixar e instalar o instalador do MongoDB na máquina, no endereço acima.

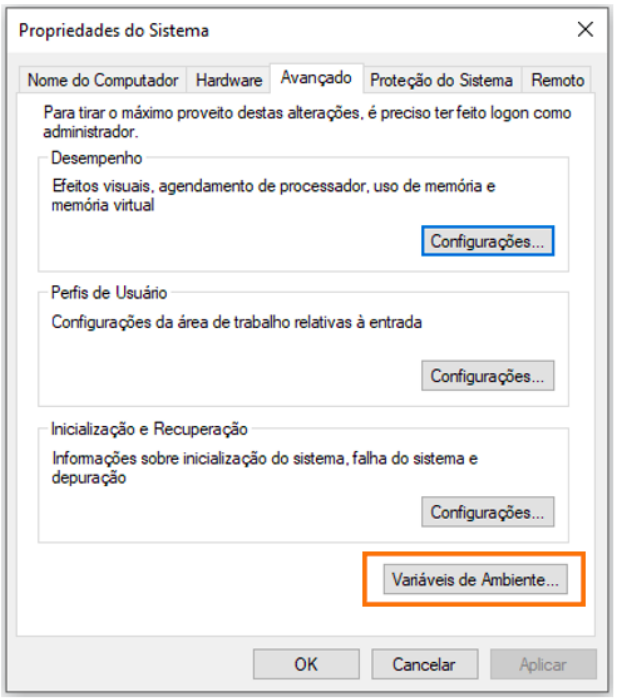


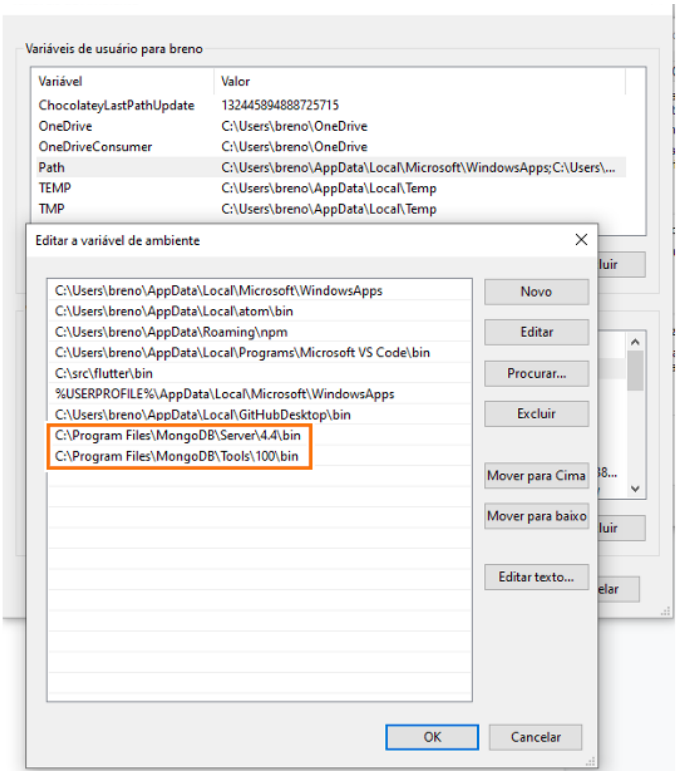


MongoDB Community Download | MongoDB

Após a instalação tive problemas nas funções do “mongod”, então tive que instalar as ferramentas completas do mongodb, acessadas no link acima.

Agora é preciso adicionar o diretório “bin” das pastas tanto do mongo, quanto das ferramentas do mongo.





Com os caminhos adicionados às variáveis de ambiente é preciso também adicionar uma pasta no mesmo drive onde o mongodb está instalado com o nome “*data*”, e dentro desta pasta uma outra com o nome “*db*”. Em seguida é só abrir o “*cmd*”, no caso do windows, e digitar mongod. Com isso o banco local está criado, para acessá-lo, basta abrir outro “*cmd*” e digitar “*mongo*”.

Selecionando / Criando banco de dados:

```
use rpg-db
```

➤ Executar UMA inserção de dado;
Breno Antunes

Inserção usuário:

```
MongoDB Enterprise atlas-e451mu-shard-0:PRIMARY> db.Users.insert ({email : "breno.cantunes@hotmail.com"
,username : "Breno626",password : "123456",cash : 100})
```

Print consulta ao dado inserido:

```
MongoDB Enterprise atlas-e451mu-shard-0:PRIMARY> db.Users.find({username: "Breno626"})
{ "_id" : ObjectId("609db078e584d5234e277691"), "email" : "breno.cantunes@hotmail.com", "username" : "B
reno626", "password" : "123456", "cash" : 100 }
```

➤ Executar UMA consulta aos dados do BD;
Thalles Cotta

Executar UMA consulta aos dados do BD:

```
> db.Chars.find(ObjectId("609dd9bf1f7c53014de64ee0"))
< { _id: ObjectId("609dd9bf1f7c53014de64ee0"),
    nickname: 'Leon Wong',
    stats:
     { char_level: 100,
       base_hp: 651,
       base_res: 821,
       base_mana: 282,
       max_strength: 97,
       max_weight: 10 },
    last_region_id: ObjectId("609dbb7be584d5234e277693"),
    user_id: ObjectId("609db62ee584d5234e277692"),
    coins: 4159,
    race_types: 'Demon',
    demociac_breath: 100,
    items_id:
     [ ObjectId("609dc7be0c8b2945b1f71b5d"),
       ObjectId("609dc7be0c8b2945b1f71b5a") ] }
Enterprise atlas-e451mu-shard-0 [primary] >
```

➤Exemplo de criação de um usuário;

```
> use admin
switched to db admin
> db.createUser({user: "thalles", pwd: "123", roles: ["root"]})
Successfully added user: { "user" : "thalles", "roles" : [ "root" ] }
>
```

➤ Como gerar plano de execução no BD, caso tenha a possibilidade e um exemplo de 1 cenário de otimização de consulta;
Carolina Carvalhosa

Para a consulta `db.Items.find({item_cost: {$gte: 1000, $lte: 3000}})` que procura os itens que tem um custo entre 1000 e 3000, vamos analisar um plano de execução:

Rodando o comando `db.Items.find({item_cost: {$gte: 1000, $lte: 3000}}).explain("executionStats")` para extrair o plano de execução:

```
MongoDB Enterprise atlas-e451mu-shard-0:PRIMARY> db.Items.find({item_cost: {$gte: 1000, $lte:
 3000}}).explain("executionStats")
```

Output:

```
{
	"queryPlanner" : {
		"plannerVersion" : 1,
		"namespace" : "rpg-db.Items",
		"indexFilterSet" : false,
		"parsedQuery" : {
			"$and" : [
				{
					"item_cost" : {
						"$lte" : 3000
					}
				},
				{
					"item_cost" : {
						"$gte" : 1000
					}
				}
			]
		},
```

```
        "winningPlan" : {
                "stage" : "COLLSCAN",
                "filter" : {
                        "$and" : [
                                {
                                        "item_cost" : {
                                                "$lte" : 3000
                                        }
                                },
                                {
                                        "item_cost" : {
                                                "$gte" : 1000
                                        }
                                }
                        ]
                },
                "direction" : "forward"
        },
        "rejectedPlans" : [ ]
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 23,
        "executionTimeMillis" : 0,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 50,
        "executionStages" : {
                "stage" : "COLLSCAN",
                "filter" : {
                        "$and" : [
                                {
                                        "item_cost" : {
                                                "$lte" : 3000
                                        }
                                },
                                {
                                        "item_cost" : {
                                                "$gte" : 1000
                                        }
                                }
                        ]
                },
                "nReturned" : 23,
                "executionTimeMillisEstimate" : 0,
                "works" : 52,
                "advanced" : 23,
                "needTime" : 28,
                "needYield" : 0,
```

```
                    "saveState" : 0,
                    "restoreState" : 0,
                    "isEOF" : 1,
                    "direction" : "forward",
                    "docsExamined" : 50
              }
        },
        "serverInfo" : {
              "host" : "av2-shard-00-01.rhw3n.mongodb.net",
              "port" : 27017,
              "version" : "4.4.6",
              "gitVersion" : "72e66213c2c3eab37d9358d5e78ad7f5c1d0d0d7"
        },
        "ok" : 1,
        "$clusterTime" : {
              "clusterTime" : Timestamp(1621041979, 10),
              "signature" : {
                    "hash" : BinData(0,"tccTZHNgzmaAfZmZhk5lX9Z+AGw="),
                    "keyId" : NumberLong("6948801960128544771")
              }
        },
        "operationTime" : Timestamp(1621041979, 10)
}
```


Analisando os resultados do explain():

- `queryPlanner.winningPlan.stage` mostrando `COLLSCAN` indica que que o *mongos* teve que verificar a coleção inteira (documento por documento) para identificar os resultados. Essa operação é bem custosa e pode resultar em consultas mais lentas;
- `executionStats.nReturned` indica que `23` resultados foram retornados (correspondências);
- `executionStats.totalKeysExamined` mostrando `0` indica que essa consulta não está usando índice.
- `executionStats.totalDocsExamined` mostrando `50` indica que o MongoDB teve que analisar `50` documentos (todos os documentos da coleção) para achar as `23` correspondências;

Criando o índice:

```
MongoDB Enterprise atlas-e451mu-shard-0:PRIMARY> db.Items.createIndex({item_cost: 1})
```

Output:

```
{
	"createdCollectionAutomatically" : false,
	"numIndexesBefore" : 1,
	"numIndexesAfter" : 2,
	"commitQuorum" : "votingMembers",
	"ok" : 1,
	"$clusterTime" : {
		"clusterTime" : Timestamp(1621043167, 8),
		"signature" : {
			"hash" : BinData(0,"Y/sy51nS3km5h+xQ+pLKXFoVDhk="),
			"keyId" : NumberLong("6948801960128544771")
		}
	},
	"operationTime" : Timestamp(1621043167, 8)
}
```

Output da nova execução do comando explain( ):

```
{
	"queryPlanner" : {
		"plannerVersion" : 1,
		"namespace" : "rpg-db.Items",
		"indexFilterSet" : false,
		"parsedQuery" : {
			"$and" : [
				{
					"item_cost" : {
						"$lte" : 3000
					}
				},
				{
					"item_cost" : {
						"$gte" : 1000
					}
				}
			]
		},
		"winningPlan" : {
			"stage" : "FETCH",
			"inputStage" : {
				"stage" : "IXSCAN",
				"keyPattern" : {
					"item_cost" : 1
				},
				"indexName" : "item_cost_1",
```

```
                        "isMultiKey" : false,
                        "multiKeyPaths" : {
                              "item_cost" : [ ]
                        },
                        "isUnique" : false,
                        "isSparse" : false,
                        "isPartial" : false,
                        "indexVersion" : 2,
                        "direction" : "forward",
                        "indexBounds" : {
                              "item_cost" : [
                                    "[1000.0, 3000.0]"
                              ]
                        }
                  }
            },
            "rejectedPlans" : [ ]
      },
      "executionStats" : {
            "executionSuccess" : true,
            "nReturned" : 23,
            "executionTimeMillis" : 1,
            "totalKeysExamined" : 23,
            "totalDocsExamined" : 23,
            "executionStages" : {
                  "stage" : "FETCH",
                  "nReturned" : 23,
                  "executionTimeMillisEstimate" : 0,
                  "works" : 24,
                  "advanced" : 23,
                  "needTime" : 0,
                  "needYield" : 0,
                  "saveState" : 0,
                  "restoreState" : 0,
                  "isEOF" : 1,
                  "docsExamined" : 23,
                  "alreadyHasObj" : 0,
                  "inputStage" : {
                        "stage" : "IXSCAN",
                        "nReturned" : 23,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 24,
                        "advanced" : 23,
                        "needTime" : 0,
                        "needYield" : 0,
                        "saveState" : 0,
                        "restoreState" : 0,
                        "isEOF" : 1,
```

```
                        "keyPattern" : {
                              "item_cost" : 1
                        },
                        "indexName" : "item_cost_1",
                        "isMultiKey" : false,
                        "multiKeyPaths" : {
                              "item_cost" : [ ]
                        },
                        "isUnique" : false,
                        "isSparse" : false,
                        "isPartial" : false,
                        "indexVersion" : 2,
                        "direction" : "forward",
                        "indexBounds" : {
                              "item_cost" : [
                                    "[1000.0, 3000.0]"
                              ]
                        },
                        "keysExamined" : 23,
                        "seeks" : 1,
                        "dupsTested" : 0,
                        "dupsDropped" : 0
                  }
            }
      },
      "serverInfo" : {
            "host" : "av2-shard-00-01.rhw3n.mongodb.net",
            "port" : 27017,
            "version" : "4.4.6",
            "gitVersion" : "72e66213c2c3eab37d9358d5e78ad7f5c1d0d0d7"
      },
      "ok" : 1,
      "$clusterTime" : {
            "clusterTime" : Timestamp(1621043419, 6),
            "signature" : {
                  "hash" : BinData(0,"FozoyVgaBzlBSoYPaV/8FnNWV5M="),
                  "keyId" : NumberLong("6948801960128544771")
            }
      },
      "operationTime" : Timestamp(1621043419, 6)
}
```


Podemos perceber agora as seguintes mudanças:

- `queryPlanner.winningPlan.stage` mostrando `IXSCAN` indica que houve o uso de index;
- `executionStats.totalKeysExamined` mostrando `23` indica que o MongoDB analisou 23

entradas de índice, que bate com o número de documentos retornados. Isso indica que o *mongos* só teve que examinar chaves com índice para retornar o resultado, ou seja, não teve que percorrer todos os documentos da coleção. Isso significa que a consulta foi super eficiente;

- `executionStats.totalDocsExamined` mostrando `23` indica que o MongoDB só teve que analisar `23` documentos para achar as `23` correspondências;

Conclusão da análise do plano de execução

Sem o índice, essa consulta percorreria todos os documentos da coleção. Porém, como são poucos documentos, podemos avaliar pelo `executionStats.executionTimeMillisEstimate` que o tempo estimado para execução da consulta não mudou, portanto, neste caso o índice não seria necessário.

Cenário em que essa otimização seria necessária

Essa operação poderia ser bem lenta se tivéssemos uma quantidade imensa de documentos e documentos que possuem outros objetos JSON dentro deles. Se nossa aplicação tivesse que fazer muitas buscas com um *range* parecido ou menor, seria muito custoso executar esse tipo de consulta, já que percorre a coleção inteira sem necessidade e poderíamos comparar o `executionTimeMillisEstimate` para decidir se devemos criar o índice ou não.

(Referência: https://docs.mongodb.com/manual/reference/explain-results/)

➤ Exemplo de uma concessão de privilégio com execução de um comando que demonstre esse privilégio com sucesso;
Douglas Fernandes

Executar o comando mongod para iniciar o banco sem autenticação

```
$ mongod
```

Iniciar o mongosh

```
$ mongo --port 27017
```

Criar usuários com privilégios de leitura, apenas.

```
> use rpg-db-local
> db.createUser({ user:"dummy",  pwd:"123", roles:["read"] });
```

Reiniciar o mongod e mongo com o parâmetro de autenticação

```
$ mongod --auth
$ mongo --port 27017 -u “dummy” --authenticationDatabase “rpg-db-local”
```

obs.: esse último comando exigirá a senha do usuário “dummy”

Selecionar o database onde o usuário não possui permissões

```
> use rpg-db-local
> db.users.find()
> db.users.insert ({email : "exemplo@email.com", username : "dummy",
password:"123", cash:0});
```

Será retornado a mensagem:

```
WriteCommandError({
      "ok" : 0,
      "errmsg" : "not authorized on test to execute command { insert: \"users\",
ordered: true, lsid: { id:
UUID(\"f6cd8dbe-25ca-4a1b-9b42-ab18d9436546\") }, $db: \"test\" }",
      "code" : 13,
      "codeName" : "Unauthorized"
})
```

➤ Exemplo de uma revogação de privilégio com tentativa de execução de comando proibido e gerando erro;

Primeiramente, deve haver um usuário com privilégios root no banco e um usuário comum que possa escrever e ler. Para isso iremos iniciar o banco sem autenticação (mongod) e criar ambos usuários:

```
$ mongod
$ mongo --port 27017
> use admin
> db.createUser({user:"root", pwd:"123", roles: ["root"]});
> use rpg-db-local
> db.createUser({user:"commonUser", pwd:"123", roles:["readWrite"]});
```

Com ambos criado, reiniciaremos o banco com a flag de autenticação e faremos login com o usuário que possui permissões de leitura e escrita:

```
$ mongod --auth
$ mongo --port 27017 -u "commonUser" --authenticationDatabase "rpg-db-local"
```

Executar uma inserção que comprove os direitos de escrita do usuário:

```
> db.users.insert ({email : "exemplo@email.com", username : "commonUser",
password:"123", cash:0});
WriteResult({ "nInserted" : 1 }) //retorna sucesso na inserção
```

Com o usuário *root* logado no banco, iremos revogar a permissão de escrita do usuário *commonUser*:

```
$ mongo --port 27017 -u "root" --authenticationDatabase "admin"
> use rpg-db-local
> db.updateUser("commonUser", {roles:["read"]});
```

Agora vamos autenticar com o usuário *commonUser* e tentar fazer novamente a inserção de um novo documento. Essa inserção retornará erro devido aos direitos revogados.

```
$ mongo --port 27017 -u "commonUser" --authenticationDatabase "rpg-db-local"
> use rpg-db-local
> db.users.insert ({email : "exemplo2@email.com", username : "commonUser2",
password:"123", cash:0});
WriteCommandError({
      "ok" : 0,
      "errmsg" : "not authorized on rpg-db-local to execute command { insert:
\"users\", ordered: true, lsid: { id:
UUID(\"d234cba9-6d83-49a8-93d4-bfbc38d6043a\") }, $db: \"rpg-db-local\" }",
      "code" : 13,
      "codeName" : "Unauthorized"
}) // retorna erro na inserção
```

(***Referências para os exemplos práticos de permissões e criação de usuário:***
https://docs.mongodb.com/manual/reference/method/db.updateUser/
https://docs.mongodb.com/manual/reference/program/mongod/
https://docs.mongodb.com/manual/tutorial/manage-users-and-roles/)

Referências Bibliográficas:

https://arquivo.devmedia.com.br/REVISTAS/sql/imagens/147/modelagem/modelagem_mongo.png

https://severalnines.com/sites/default/files/blog/node_5989/image2.png

https://docs.mongodb.com/manual/sharding/

https://docs.mongodb.com/manual/sharding/#sharded-cluster/

https://docs.mongodb.com/manual/core/sharded-cluster-query-router/

https://docs.mongodb.com/manual/indexes/

https://docs.mongodb.com/manual/core/query-optimization/

https://docs.mongodb.com/manual/reference/explain-results/

https://pt.wikipedia.org/wiki/Toler%C3%A2ncia_a_falhas e

https://www.infonova.com.br/gestao-de-ti/o-que-e-failover-importante/

https://docs.mongodb.com/manual/core/replica-set-rollbacks/

https://docs.mongodb.com/manual/core/replica-set-rollbacks/

https://docs.mongodb.com/manual/core/replica-set-oplog/

http://db4beginners.com/blog/mongodb-4-0-o-que-voce-precisa-saber/

https://docs.mongodb.com/manual/core/read-isolation-consistency-recency/

(POLITOWSKI; MARAN, 2014, p. 4)

[Matthes and Orend 2010].

http://repositorioinstitucional.uea.edu.br/bitstream/riuea/2536/1/Desenvolvimento%20de%20dispositivo%20para%20o%20monitoramento%20e%20controle%20de%20estoque%20no%20processo%20de%20movimenta%C3%A7%C3%A3o%20de%20mat%C3%A9ria-prima%20em%20uma%20ind%C3%BAstria%20utilizando%20a%20plataforma%20arduino.pdf

https://pt.stackoverflow.com/questions/49851/revogar-privil%C3%A9gios

https://docs.mongodb.com/manual/core/backups/

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

https://docs.mongodb.com/manual/reference/method/db.updateUser/

https://docs.mongodb.com/manual/reference/program/mongod/

https://docs.mongodb.com/manual/tutorial/manage-users-and-roles