Estratégia Concursos

Aula 10

Banco do Brasil (Escriturário - Agente de Tecnologia) Passo Estratégico de Tecnologia de Informação - 2023 (Pós-Edital)

Autor:
Thiago Rodrigues Cavalcanti

24 de Fevereiro de 2023

2. Banco de Dados: Banco de dados NoSQL (conceitos básicos, bancos orientados a grafos, colunas, chave/valor e documentos)

Sumário

ANÁLISE ESTATÍSTICA

Inicialmente, convém destacar os percentuais de incidência de todos os assuntos previstos no nosso curso – quanto maior o percentual de cobrança de um dado assunto, maior sua importância:

Assunto	Quantidade	Grau de incidência em concursos similares FGV
Linguagens de programação para ciência de dados: linguagem Python.	49	44,55%
Análise de dados. Agrupamentos. Tendências. Projeções. Conceitos de Analytics. Aprendizado de Máquina. Inteligência Artificial.	32	29,09%
Bancos de dados não relacionais: bancos de dados NoSQL; Modelos Nosql. Principais SGBD's. Soluções para Big Data.	9	8,18%
Processamento de Linguagem Natural.	8	7,27%
Linguagens de programação para ciência de dados: linguagem R.	5	4,55%
Ciência de dados: Importância da informação. Big Data. Big Data em relação a outras disciplinas. Ciência dos dados. Ciclo de vida do processo de ciência de dados. Papeis dos envolvidos em projetos de Ciência de dados e Big Data. Computação em nuvens. Arquitetura de Big Data.	4	3,64%
Modelos de entrega e distribuição de serviços de Big Data. Plataformas de computação em nuvem para Big Data.	3	2,73%
Fluência em dados: conceitos, atributos, métricas, transformação de Dados.	0	0,00%
Governança de Dados: conceito, tipos (centralizada, compartilhada e colegiada).	0	0,00%

ROTEIRO DE REVISÃO E PONTOS DO ASSUNTO QUE MERECEM DESTAQUE

A ideia desta seção é apresentar um roteiro para que você realize uma revisão completa do assunto e, ao mesmo tempo, destacar aspectos do conteúdo que merecem atenção.

Para revisar e ficar bem preparado no assunto, você precisa, basicamente, seguir os passos a seguir:

NoSQL

Começaremos analisando os conceitos e os modelos de dados que dão suporte a bases de dados NoSQL. Lembrem-se que o foco aqui é entender os conceitos e não a parte técnica do assunto. Alguns nomes técnicos de ferramentas e aplicações que usam esses tipos de modelos de dados serão apresentados, contudo, você não tem necessidade de conhecer nenhuma dessas ferramentas. Vamos em frente!?

Conceitos

Os **bancos de dados relacionais** foram bem-sucedidos porque trouxeram os benefícios de armazenamento de dados persistentes de forma mais organizada, com controle de concorrência entre as transações. As transações desempenham um papel importante na hora de lidar com os erros, pois é possível fazer uma alteração e, caso ocorra um erro durante seu processamento, pode-se desfazê-la e voltar ao estado anterior.

Embora haja diferenças de um banco de dados relacional para outro os mecanismos principais permanecem os mesmos: os dialetos SQL utilizados por diversos fornecedores são similares, e as transações são realizadas praticamente da mesma forma. Banco de dados relacionais fornecem muitas vantagens, mas não são, de forma alguma, perfeitos. Desde que surgiram, há muita frustração e críticas em relação a seu uso. Vejamos as palavras chaves de uma comparação entre SQL (Relacional) e NoSQL.

Para os desenvolvedores de aplicativos, a maior frustração tem sido a diferença entre o modelo relacional e as estruturas de dados na memória, comumente chamada de **incompatibilidade de impedância**. Os modelos de dados relacionais organizam os dados em uma estrutura de tabelas e linhas, ou, mais apropriadamente, de relações e tuplas. Uma **tupla** é um conjunto de pares nome-valor e uma relação é um conjunto de tuplas.

Ao longo dos anos, tornou-se mais fácil lidar com a incompatibilidade de impedância devido à ampla disponibilidade de frameworks de mapeamento objeto-relacional, como Hibernate e iBATIS, que

implementam padrões de mapeamento bastante conhecidos. Embora a questão do mapeamento ainda seja controversa dado que os frameworks poupam muito trabalho pesado às pessoas, mas podem se tornar um problema quando estas exageram ao ignorar o banco de dados, comprometendo o desempenho das operações de manipulação de dados sobre a base.

Outro ponto relevante dentro do contexto apareceu devido ao crescimento dos Sistemas Web. Lidar com o aumento da quantidade de dados e com o tráfego exigiu mais recursos computacionais. Para comportar esse crescimento, há duas opções: ir para cima (crescimento vertical) ou para fora (horizontal). Ir para cima significa adquirir máquinas maiores, mais processadores, ter maior capacidade de armazenamento em disco e memória. Maquinas maiores, todavia, tornam-se cada vez mais caras, sem mencionar que há limites físicos quanto ao aumento do seu tamanho ou para se escalar verticalmente.

A alternativa seria utilizar mais máquinas menores em um **cluster**. Um cluster de máquinas pequenas pode utilizar hardware mais acessível e acaba se tornando mais barato para a aplicação. Ele também pode ser mais **resiliente**. Embora falhas em máquinas individuais sejam comuns, o cluster, como um todo, pode ser criado para continuar funcionando apesar dessas falhas, fornecendo alta **confiabilidade**.

Duas empresas em particular – Google e Amazon – têm sido influentes no processo de desenvolvimento de rotas alternativas para armazenamento baseado na ideia de clusters. Ambas estiveram à frente na execução de grandes clusters. Além disso, obtiveram quantidades de dados relevantes para testarem e comprovarem seus modelos. Elas eram empresas bem-sucedidas e em crescimento com fortes componentes técnicos, proporcionando-lhes os meios e as oportunidades.

Não é surpresa o fato de que essas empresas tinham em mente acabar com seus bancos de dados relacionais. Quando a década de 2000 chegou, elas produziram artigos concisos, porém altamente influentes, a respeito de seus trabalhos: **BigTable** (Google) e **Dynamo** (Amazon).

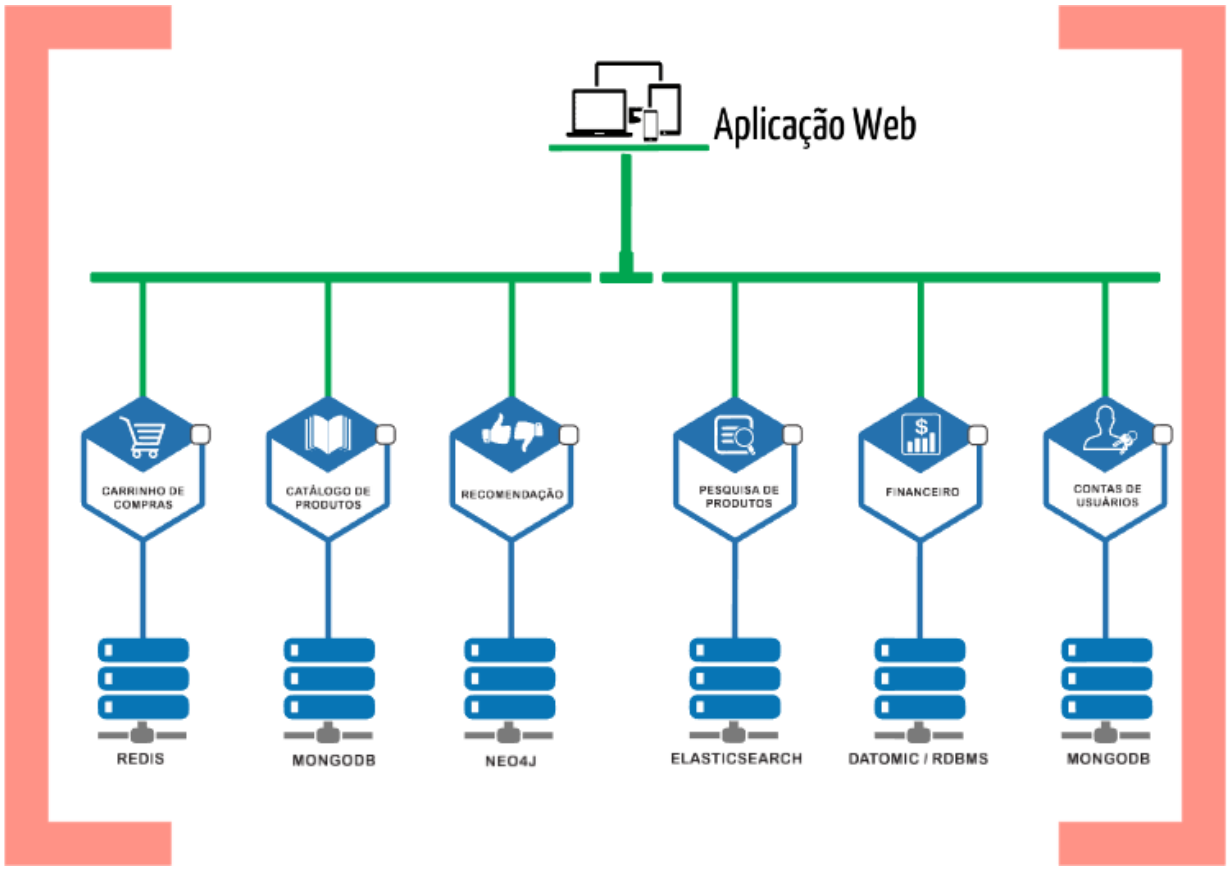
Os exemplos do BigTable e do Dynamo inspirou a criação de projetos, que faziam experimentações com armazenamentos alternativos de dados, e discussões sobre o assunto haviam se tornado uma parte essencial das melhores conferências sobre software daquela época. O termo “NoSQL” que conhecemos hoje é resultado de uma reunião realizada no dia 11 de junho de 2009, em São Francisco – Califórnia, organizada por Johan Oskarsson, um desenvolvedor de software de Londres.

Johan estava interessado em descobrir mais sobre esses novos bancos de dados enquanto estava em São Francisco para um evento sobre Hadoop. Já que dispunha de pouco tempo, achou que não seria viável visitar todas as empresas, de modo que resolveu organizar uma reunião em que todos pudessem estar presentes e apresentar seu trabalho para quem estivesse interessado em conhecê-lo.

A chamada original, NoSQL Meetup, para a reunião pedia por “bancos de dados não relacionais, distribuídos e de código aberto”. As palestras lá realizadas versaram sobre Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB e MongoDB, mas o termo nunca ficou limitado a esse grupo original. Não há uma definição genericamente aceita nem uma autoridade para fornecer uma, de modo que tudo o que podemos fazer é discutir algumas características comuns em bancos de dados que tendem a ser chamados de “NoSQL”.

As **características comuns** dos bancos de dados NoSQl são: não utilizam o modelo relacional, tem uma boa execução em clusters, ter código aberto (open source), são criados para suportar propriedades da web do século XXI, e não tem um esquema definido (schema free).

O resultado mais importante do surgimento do NoSQL é a **persistência poliglota.** Em vez de escolher o banco de dados relacional mais utilizado por todos, precisamos entender a natureza dos dados que estamos armazenando e como queremos manipulá-los. O resultado é que a maioria das organizações terá uma mistura de tecnologias de armazenamento de dados para diferentes circunstâncias. Veja a figura abaixo:



Modelos de dados

Um modelo de dados é a forma pela qual percebemos e manipulamos nossos dados. Para as pessoas utilizarem um banco de dados precisam de um modelo que descreve a forma pela qual interagimos com os dados desse banco. Embora o termo formal para modelo esteja relacionado a um metamodelo ou uma abstração sobre os dados, quando tratamos de modelos dentro do contexto de NoSQL estamos nos referindo a forma ou o modo pelo qual o gerenciador do banco de dados organiza seus dados.

O modelo de dados dominante nas últimas décadas é o modelo relacional, já falamos bastante sobre ele em uma aula anterior, que pode ser entendido como um conjunto de tabelas. Cada tabela possui linhas e cada linha representa uma entidade de interesse. Descrevemos essa entidade por meio de colunas, cada uma tendo um único valor. Uma coluna pode se referir a outra linha da mesma tabela ou em uma tabela diferente, o que constitui um relacionamento entre essas entidades.

Uma das mudanças mais evidentes trazidas pelo NoSQL é o afastamento do modelo relacional. Cada solução NoSQL possui um modelo diferente, os quais dividimos em quatro categorias amplamente utilizadas no

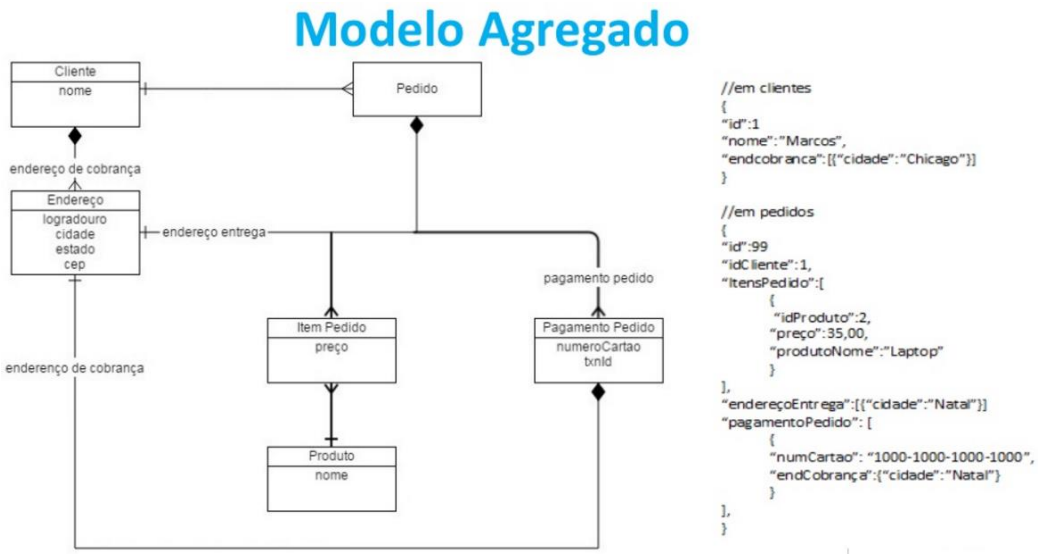
ecossistema NoSQL: **chave-valor, documento, família de colunas ou colunar, e grafos**. Dessas as três primeiras compartilham uma característica em comum conhecida como “orientação agregada”.

A orientação agregada reconhece que você, frequentemente, deseja trabalhar com dados na forma de unidades que tenham uma estrutura mais complexa do que um conjunto de tuplas. Pode ser útil pensar em termos de um registro complexo que permita que listas e outras estruturas de dados sejam aninhadas dentro dele. Partindo desta ideia é possível agora definir o conceito de **agregado**.

Um agregado é um conjunto de objetos relacionados que desejamos tratar como uma unidade. Em particular, é uma unidade de manipulação de dados e gerenciamento de consistência. Normalmente, preferimos atualizar agregados como operações atômicas e comunicarmo-nos com nosso armazenamento de dados em termos agregados.

Essa definição corresponde bem ao funcionamento dos bancos de dados chave-valor, documento e família de colunas. Lidar com agregados facilita muito a execução desses bancos de dados em um cluster, uma vez que o agregado constitui uma unidade natural para replicação e fragmentação. Agregados também são, frequentemente, mais simples de ser manipulados pelos programadores de aplicativos.

Vejam um exemplo de um modelo agregado na figura abaixo. Observem que nestes casos é possível criar uma estrutura hierárquica dentro dos atributos dos objetos. Para fazer isso usamos podemos usar JSON ou XML.

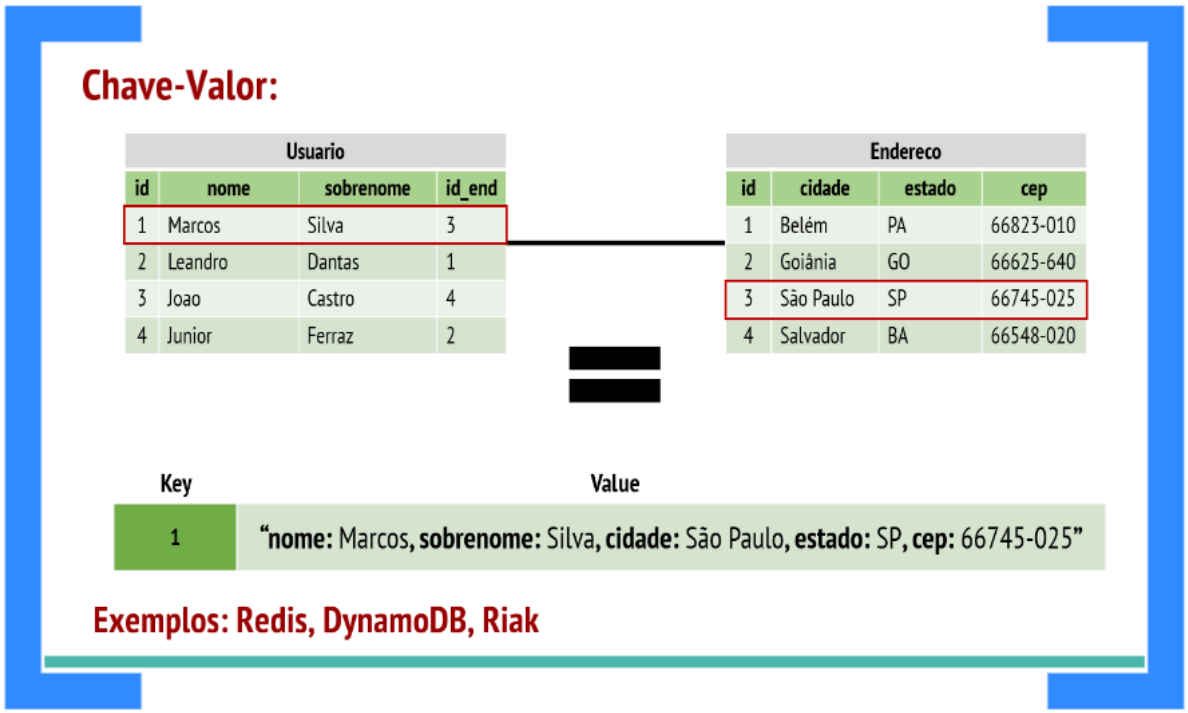


Vamos agora tratar com um pouco mais de detalhes cada uma das categorias de modelos que apresentamos anteriormente.

Modelo de dados chave-valor

O modelo de dados chave-valor trata o agregado como um todo opaco, o que significa que somente será possível fazer uma pesquisa por chave para o agregado como um todo, não sendo possível executar uma consulta nem recuperar apenas parte do agregado.

Esse é o tipo de banco de dados NoSQL mais simples e permite a visualização do banco de dados como uma grande tabela *hash*. Conforme falamos acima, o banco de dados é composto por um conjunto de chaves, as quais estão associadas um único valor. A figura abaixo apresenta um exemplo de um banco de dados que armazena informações pessoais no formato chave-valor. A chave representa um campo como o nome, enquanto o valor representa a instância do correspondente.



Este modelo, por ser de fácil compreensão e implementação, permite que os dados sejam rapidamente acessados pela chave, principalmente em sistemas que possuem alta escalabilidade, contribuindo para aumentar a disponibilidade de acesso aos dados. As operações disponíveis para manipulação de dados são bem simples, como *get() e set()*, que permitem retornar e capturar valores, respectivamente. A desvantagem deste modelo é que não permite a recuperação de objetos por meio de consultas mais complexas.

Como exemplo de banco de dados NoSQL que adota o modelo chave-valor podemos destacar o **DynamoDB**, o qual foi desenvolvido pela Amazon. Dentre as principais funcionalidades do Dynamo temos a possibilidade de realizar particionamento, replicação e versionamento dos dados.

Além do Dynamo, outras soluções NoSQL seguem o mesmo conceito de chave-valor: **Redis**, **Riak** e **GenieDB**.

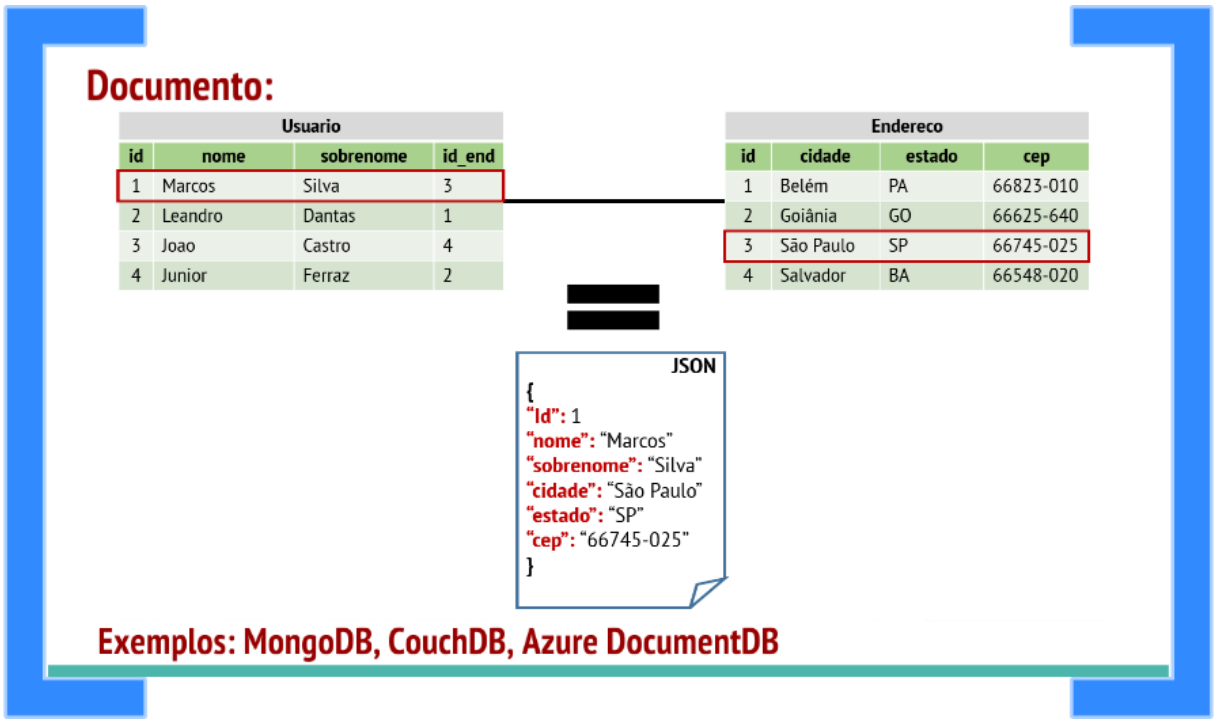
Modelo de dados de documento

O modelo de documentos torna o agregado transparente para o banco de dados, permitindo que sejam executadas consultas e recuperações parciais. Entretanto, pelo fato de o documento não possuir esquema, o banco de dados não pode atuar muito na estrutura desse documento para otimizar o armazenamento e a recuperação de partes do agregado.

Um documento, em geral, é um objeto com um identificador único e um conjunto de campos, que podem ser strings, listas ou documentos aninhados. Estes campos se assemelham a estrutura chave-valor, que cria uma única tabela hash para todo o banco de dados. No modelo orientado a documentos temos um conjunto de documentos e em cada documento temos um conjunto de campos (chaves) e o valor deste campo.

Outra característica importante é que este modelo não depende de um esquema rígido, ou seja, não exige uma estrutura fixa como ocorre nos bancos de dados relacionais. Assim, é possível que ocorra uma atualização na estrutura do documento, com a adição de novos campos, por exemplo, sem causar problemas no banco de dados.

Na figura a seguir temos um exemplo de documento representado por um banco de dados de fornecedor (*supplier*) que tem os campos ID, Name, Address e Order. Para cada um desses campos temos os valores associados. Vejam que o atributo order aponta para outro documento.

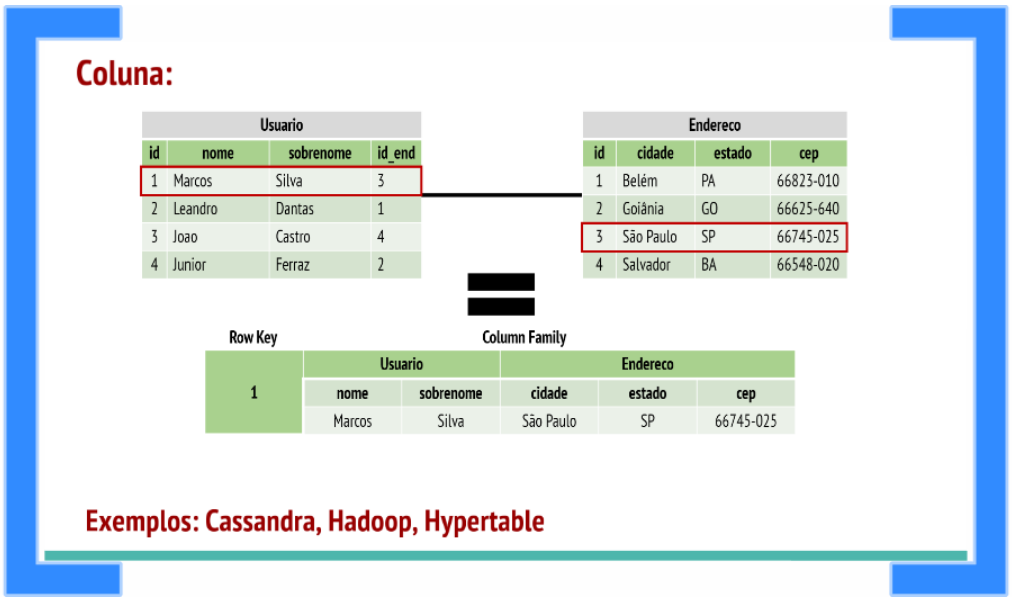


Como principais soluções que adotam o modelo orientado a documentos destacamos o **CouchDB** e o **MongoDB**. CouchDB utiliza o formato JSON e é implementado em Java, além disso permite replicação e consistência. O MongoDB foi implementado em C++ e permite tanto concorrência como replicação.

Modelo colunar

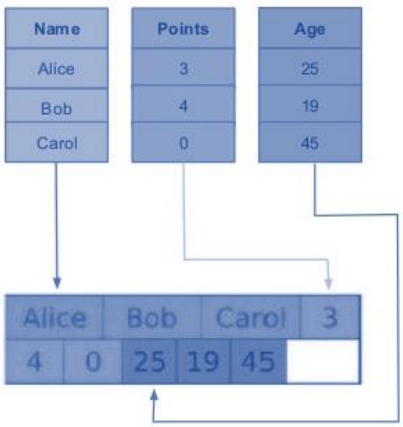
Modelos de famílias de colunas dividem o agregado em famílias de colunas, permitindo ao banco de dados tratá-las como unidades de dados dentro do agregado da linha. Isso impõe alguma estrutura ao agregado, mas também permite que o banco de dados aproveite a estrutura para melhorar sua acessibilidade.

Vejam que neste caso, mudamos o paradigma em relação ao modelo chave-valor. A orientação deixa de ser por registros ou tuplas para orientação por colunas. Neste modelo os dados são indexados por uma trilha (linha, coluna e *timestamp*), onde linhas e colunas são identificadas por chaves e o *timestamp* permite diferenciar múltiplas versões de um mesmo dado.



Vale ressaltar que operações de leitura e escrita são atômicas, ou seja, todos os valores associados a uma linha são considerados na execução destas operações, independentemente das colunas que estão sendo lidas ou escritas. Outro conceito associado ao modelo é o de família de colunas (*column family*), que é usado com o intuito de agrupar colunas que armazenam o mesmo tipo de dados.

Observem abaixo o que geralmente acontece na prática em banco de dados colunar. Neste caso as colunas das tabelas são serializadas e armazenadas em disco.



Este modelo de dados surgiu com o **BigTable** do Google, por isso é comum falar sobre o modelo de dados BigTable. Dentre as características deste modelo temos a possibilidade de particionamento dos dados, além de oferecer forte consistência, mas não garante alta disponibilidade. Outras soluções sugiram após o BigTable, dentre elas o **Cassandra**, desenvolvido pelo Facebook. Temos também o **Hbase**, que é um banco de dados open source semelhante ao BigTable, que utiliza o Hadoop.

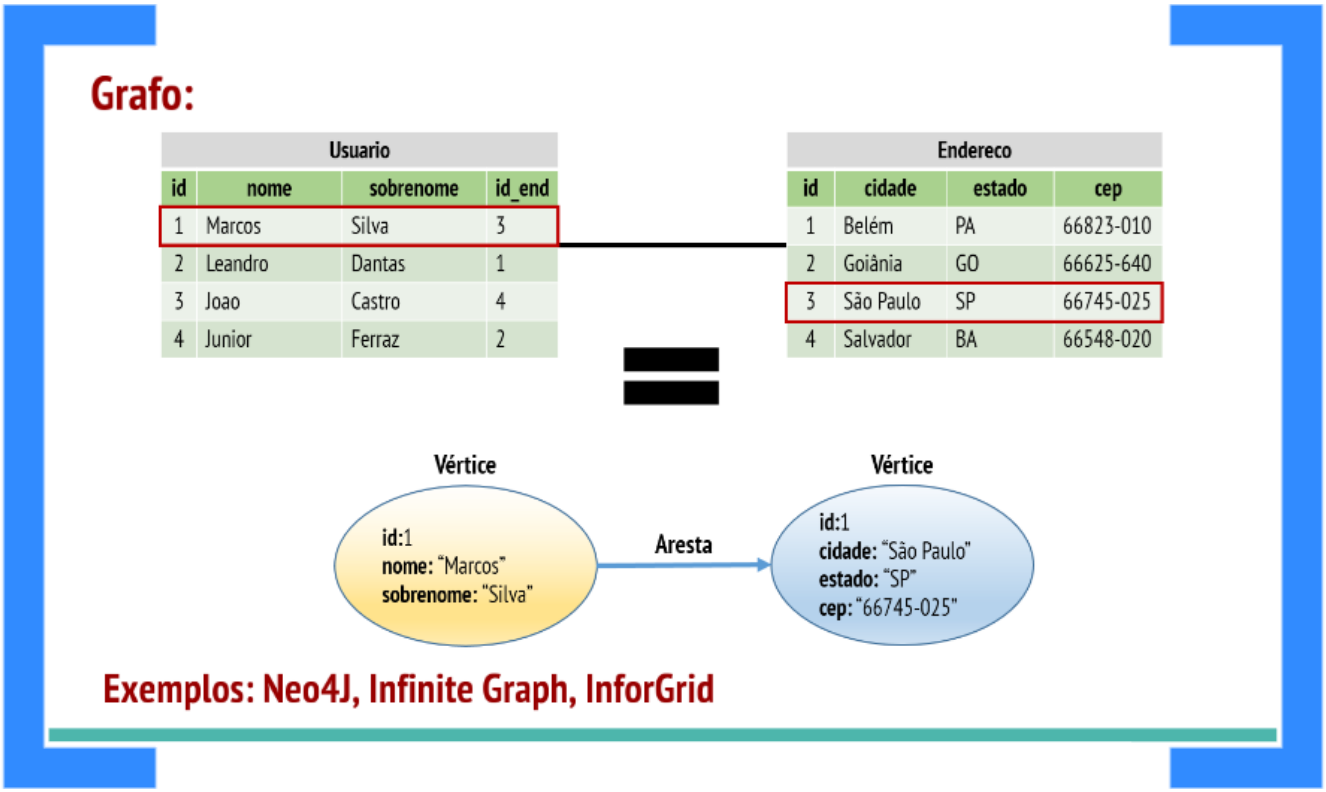
Modelo de grafos

Bancos de dados de grafos são motivados por uma frustração diferente com banco de dados relacionais e, por isso, têm um modelo oposto – registros pequenos com interconexões complexas. A ideia desse modelo é representar os dados e/ou o esquema dos dados como grafos dirigidos, ou como estruturas que generalizem a noção de grafos.

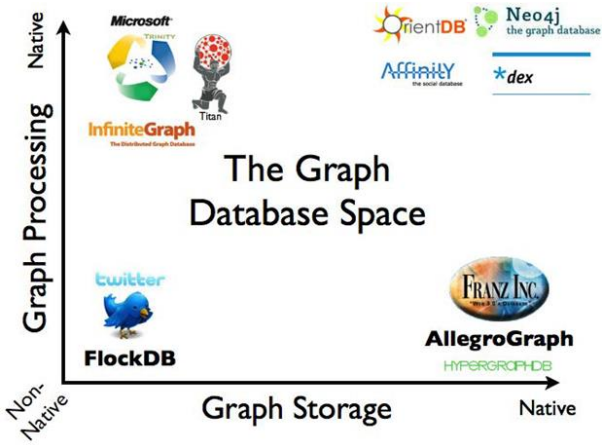
O modelo de grafos é mais interessante que outros quando informações sobre a interconectividade ou a topologia dos dados são mais ou tão importantes quantos os dados propriamente ditos. O modelo orientado a grafos possui três componentes básicos: os **nós** (são os vértices do grafo), os **relacionamentos** (são as arestas) e as **propriedades** (ou atributos) dos nós e relacionamentos.

Neste caso, o banco de dados pode ser visto como um multigrafo rotulado e direcionado, onde cada par de nós pode ser conectado por mais de uma aresta. Um exemplo pode ser: "Quais cidades foram visitadas anteriormente (seja residindo ou viajando) por pessoas que viajaram para o Rio de Janeiro?".

No modelo relacional esta consulta poderia ser muito complexa devido a necessidade de múltiplas junções, o que poderia acarretar uma diminuição no desempenho da aplicação. Porém, por meio dos relacionamentos inerentes aos grafos, estas consultas tornam-se mais simples e diretas.



Alguns bancos que utilizam esse padrão são: Neo4J, Infinite Graph, InfoGrid, HyperGraphDB. Vejam a figura abaixo que apresenta duas características relacionadas aos bancos de dados de grafos: o processamento e o armazenamento nativo.



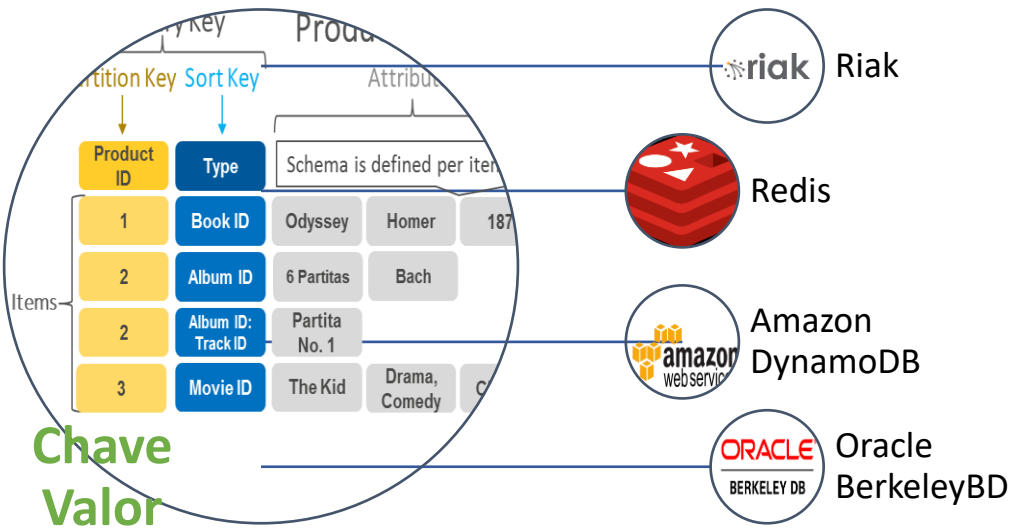
Considerações finais sobre modelos

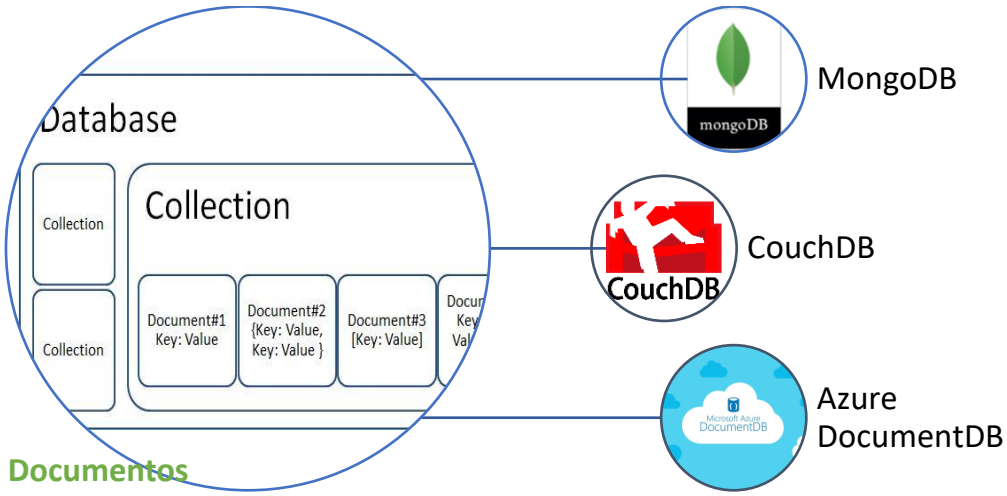
Bancos de dados orientados a agregados tornam os relacionamentos entre agregados mais difíceis de lidar do que relacionamentos intra-agregados.

Bancos de dados **sem esquema** permitem que campos sejam adicionados livremente aos registros, mas geralmente há um esquema implícito esperado pelos usuários dos dados.

Banco de dados orientados a agregados, muitas vezes, criam **visões materializadas** para fornecer dados organizados de um modo diferente de seus agregados primários. Isso, muitas vezes, é realizado com computação **MapReduce**.

Outro ponto importante é saber associar cada categoria dos modelos de dados aos seus respectivos representantes ou principais referências. Veja nas figuras a seguir essa lista de forma organizada.




Database
Collection
Collection
Collection
Document#1
Key: Value
Document#2
{Key: Value,
Key: Value }
Document#3
[Key: Value]
Documentos
MongoDB
CouchDB
Azure
DocumentDB

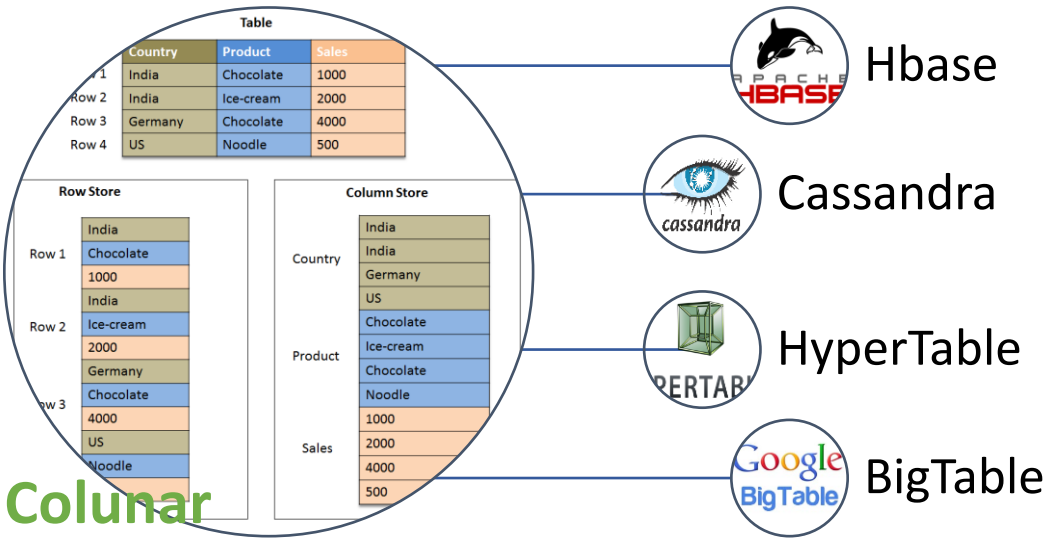
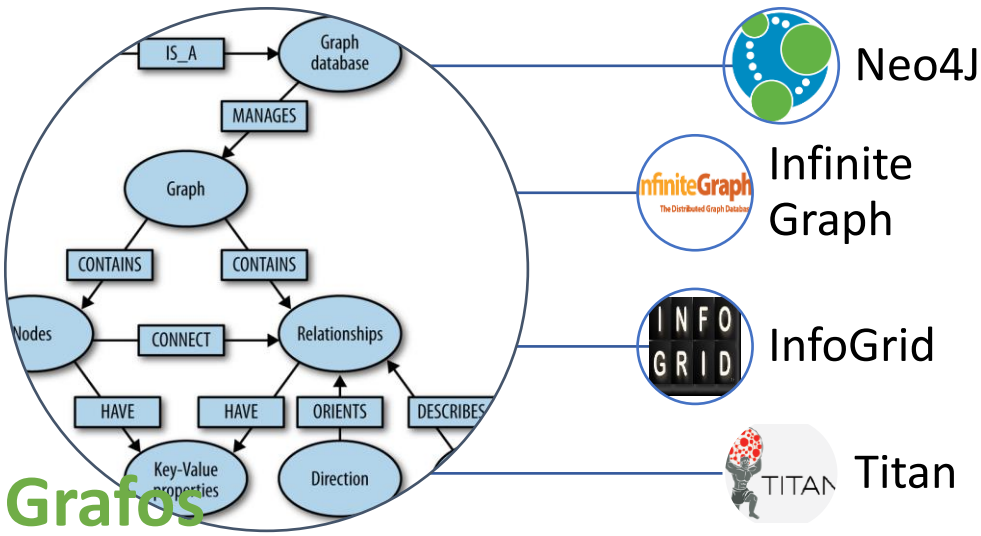


Table
Country
Product
Sales
India
Chocolate
1000
India
Ice-cream
2000
Germany
Chocolate
4000
US
Noodle
500
Row Store
Column Store
Colunar
Hbase
Cassandra
HyperTable
BigTable


IS_A
Graph database
MANAGES
Graph
CONTAINS
CONTAINS
Nodes
CONNECT
Relationships
HAVE
HAVE
ORIENTS
DESCRIBES
Key-Value
Direction
Grafos
Neo4J
Infinite
Graph
InfoGrid
Titan

Formas de distribuição

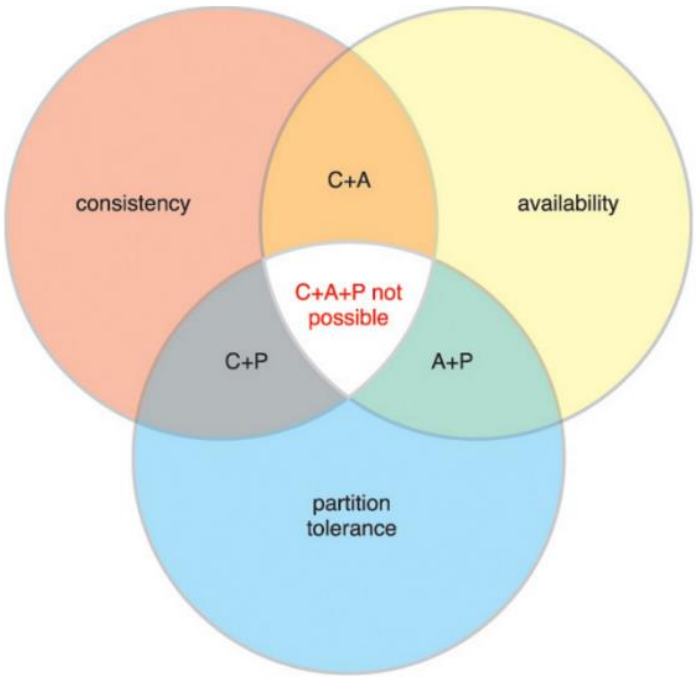
Há dois estilos de distribuição de dados: a **replicação** e a **fragmentação**. A fragmentação distribui dados diferentes em múltiplos servidores, de modo que cada servidor atue como a única fonte de um subconjunto de dados. A replicação copia os dados para servidores distintos, de modo que cada parte dos dados pode ser encontrada em múltiplos lugares. Um sistema pode utilizar ambas as técnicas.

A replicação mestre-escravo torna um nodo a cópia oficial, a qual lida com gravações, enquanto os escravos sincronizam-se com o mestre e podem lidar com as leituras. A replicação ponto a ponto permite gravações em qualquer nodo; os nodos são coordenados para sincronizar suas cópias de dados. A replicação mestre-escravo reduz a chance de conflitos de atualização, mas a ponto a ponto evita carregar todas as gravações em um único ponto de falha.

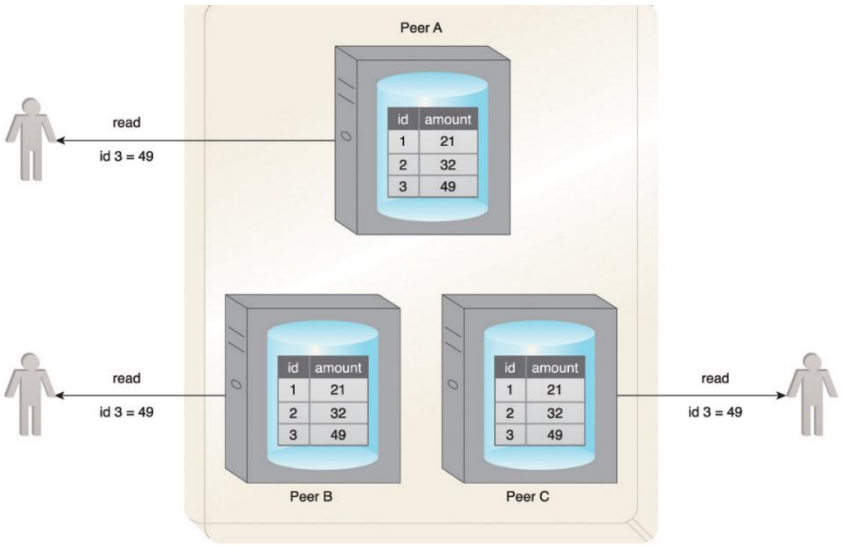
Teorema CAP

No mundo NoSQL, é comum referimo-nos ao teorema CAP como o motivo pelo qual pode-se precisar relaxar a consistência. Ele foi proposto originalmente por Eric Brewer em 2000, e recebeu uma prova formal de Seth Gilbert e Nancy Lynch alguns anos depois. É possível ouvir referências aos termos como conjectura de Brewer.

A declaração do teorema CAP é que, dadas as três propriedades de **Consistência**, **Disponibilidade** e de **Tolerância a partições**, somente é possível obter duas delas. A sigla vem do nome dessas propriedades em inglês: Consistency, Availability, e Partition tolerance. Logicamente, a existência do teorema vai depende de como são definidas cada uma dessas propriedades. Vejamos então a definição de cada uma delas. Antes vejam a figura abaixo o diagrama de Van que representa o relacionamento entre as três propriedades.



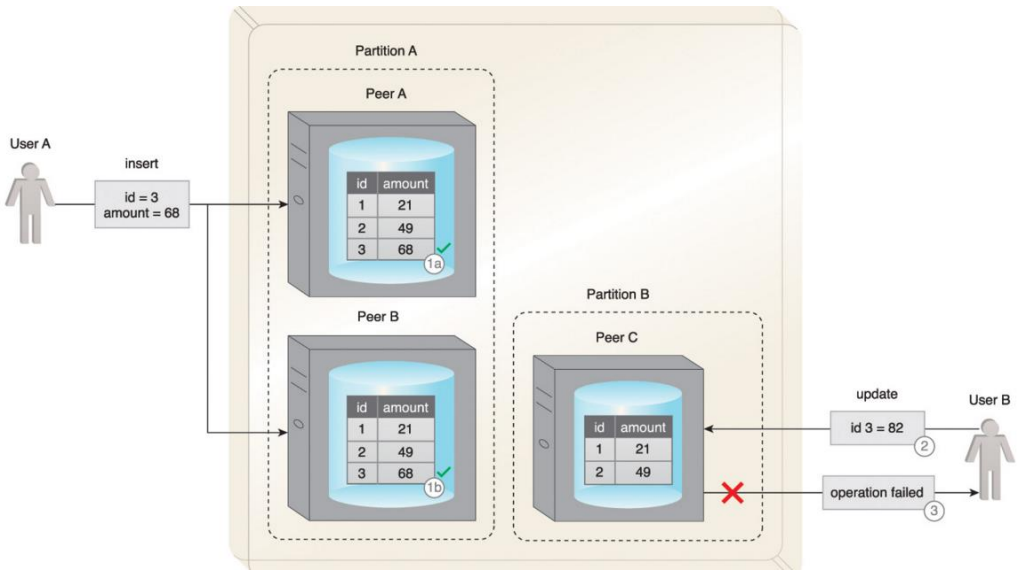
A **consistência** refere-se ao fato de que uma leitura em qualquer um dos nodos de um sistema retorna como resultado a mesma informação. Vejam a figura abaixo:



A **disponibilidade** trata do fato das requisições de leitura e escrita sempre serão reconhecidas e respondidas sobre a forma de sucesso ou falha. Desta forma, toda solicitação recebida por um nodo que não esteja no estado de falha deve resultar em uma resposta.

A **tolerância a partições** significa que o cluster pode suportar falhas na comunicação que o dividam em múltiplas partições incapazes de se comunicar entre si, essa situação é conhecida como divisão cerebral (split brain).

Observe na figura a seguir um exemplo de disponibilidade e tolerância a partições. Veja que a partição B não possui os dados referentes ao id 3 e recebe uma mensagem de erros no caso de uma partição.



ACID x BASE

ACID é um princípio de design de banco de dados relacionado ao gerenciamento de transações. É um acrônimo que significa: Atomicidade, Consistência, Isolamento e Durabilidade.

ACID é um estilo de gerenciamento de transações que utiliza controles de simultaneidade pessimista para garantir que a consistência seja mantida através da aplicação de bloqueios de registo. ACID é a abordagem tradicional de gerenciamento de transações de banco de dados, uma vez que é aproveitada pelos sistemas de gerenciamento de banco de dados relacionais.

Quando olhamos para o teorema CAP, os bancos de dados relacionais estão associados à priorização das propriedades de Consistência e Disponibilidade.

O Modelo BASE (Basically Avaliable, Soft State, Eventual Consistency), foi sugerido como contraposição ao ACID. O ACID é pessimista e força a consistência no final de cada operação, enquanto o modelo BASE é otimista e aceita que a consistência no banco de dados estará em um estado de fluxo, ou seja, não ocorrerá no mesmo instante, gerando uma “fila” de consistência de posterior execução.

A disponibilidade do Modelo BASE é garantida tolerando falhas parciais no sistema, sem que o sistema todo falhe. Por exemplo, se um banco de dados está particionado em cinco nós e um deles falha, apenas os clientes que acessam aquele nó serão prejudicados, pois o sistema como todo não cessará seu funcionamento.

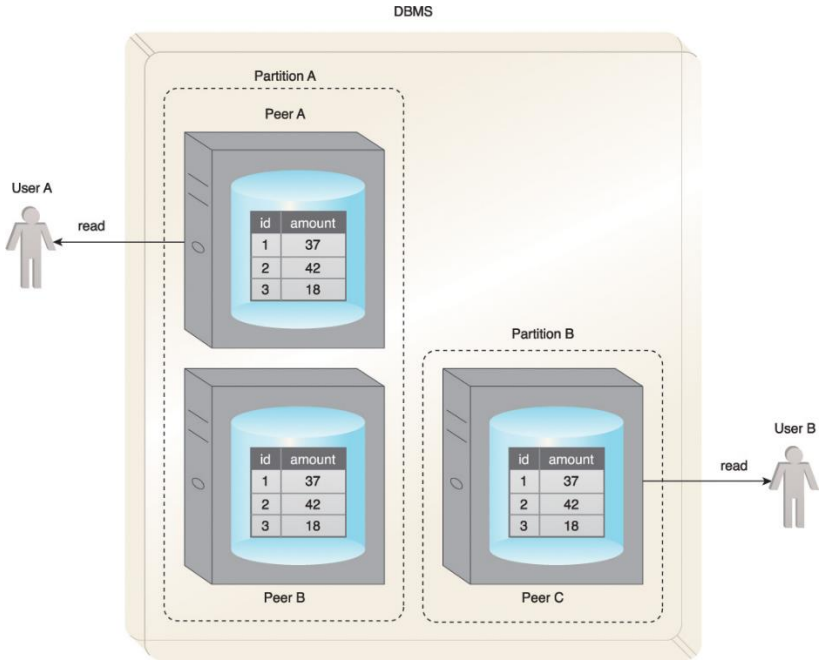
A consistência pode ser relaxada permitindo que a persistência no banco de dados não seja efetivada em tempo real (ou seja, logo depois de realizada uma operação sobre o banco). Pelo ACID, quando uma operação é realizada no SGBD, ela só será finalizada se houver a certeza de que a persistência dos dados foi realizada no mesmo momento, fato que é derivado da propriedade da durabilidade.

Já no BASE isso não se confirma. Para garantir a disponibilidade, algumas etapas são dissociadas à operação requisitada, sendo executadas posteriormente. O cliente realiza uma operação no banco de dados e, não necessariamente, a persistência será efetivada naquele instante. O Modelo BASE pode elevar o sistema a níveis de escalabilidade que não podem ser obtidos com ACID.

No entanto, algumas aplicações necessitam que a consistência seja precisamente empregada. Nenhuma aplicação bancária poderá por em risco operações de saque, depósito, transferência, etc. O projetista do banco de dados deverá estar ciente de que se utilizar o Modelo BASE estará ganhando disponibilidade em troca de consistência, o que pode afetar os usuários da aplicação referente ao banco de dados.

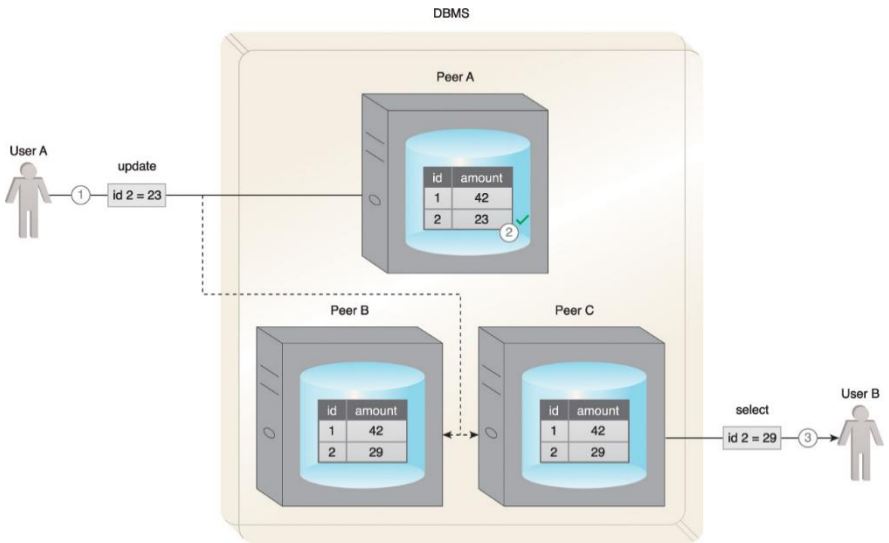
Quando um banco de dados suporta BASE, favorece a disponibilidade sobre a consistência. Em outras palavras, a base de dados é A + P a partir de uma perspectiva de teorema CAP. Em essência, BASE aproveita concorrência otimista, relaxando restrições fortes de consistência determinadas pelas propriedades ACID.

Se um banco de dados é **basicamente disponível**, esse banco de dados sempre dará conhecimento ao pedido de um cliente, quer sob a forma dos dados solicitados ou de uma notificação de sucesso/fracasso. Na figura abaixo, o banco de dados é basicamente disponível, mesmo que tenha sido particionado como um resultado de uma falha de rede.



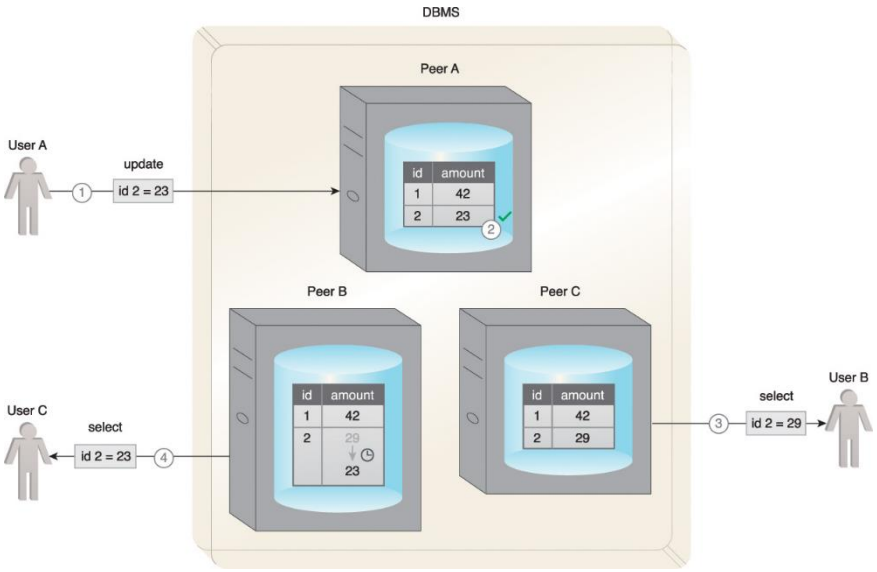
Soft state significa que um banco de dados pode estar em um estado inconsistente quando os dados são lidos. Assim, os resultados podem mudar se os mesmos dados forem solicitados novamente. Isso ocorre porque os dados podem ser atualizados para a consistência, mesmo que nenhum usuário tenha escrito no banco de dados entre duas leituras. Esta propriedade está intimamente relacionada com a consistência eventual.

Na figura abaixo, 1. Um usuário atualiza um registro no peer A. 2. Antes dos outros peers serem atualizados, o usuário B solicita o mesmo registro do peer C. 3. A base de dados está agora em um estado soft, e dados obsoletos são devolvidos ao usuário B.



Consistência eventual é o estado em que leituras feitas por diferentes clientes, imediatamente após a gravação para o banco de dados, podem não retornar resultados consistentes. O banco de dados só alcança a consistência uma vez que as mudanças foram propagadas para todos os nós. Embora a base de dados esteja em processo para atingir o estado de consistência, estaremos em um estado soft.

Na figura a seguir: 1. Um usuário atualiza um registro. 2. O registro só fica atualizado no peer A, mas antes que os outros pares possam ser atualizados, o usuário B solicita o mesmo registro. 3. A base de dados está agora em um estado soft, dados obsoletos são retornados para o usuário B do Peer C. 4. No entanto, a consistência é eventualmente atingida, e usuário C recebe o valor correto.



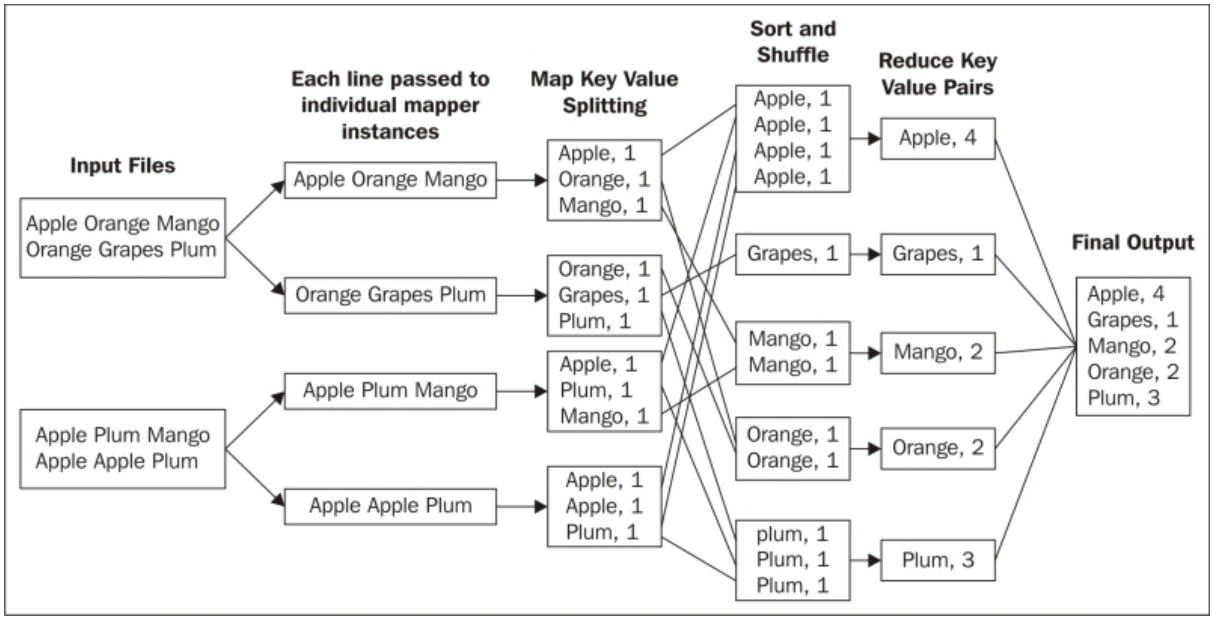
BASE enfatiza disponibilidade imediata em relação à consistência, em contraste com o ACID, que garante a consistência imediata à custa de disponibilidade devido a bloqueio de registo. Esta abordagem *soft* para a consistência **permite que as bases de dados compatíveis com BASE possam servir a múltiplos clientes sem qualquer latência embora servindo resultados inconsistentes**. No entanto, as bases de dados compatíveis com BASE não são úteis para sistemas transacionais onde a falta de consistência é uma preocupação.

MapReduce

MapReduce é um padrão que permite que operações computacionais sejam realizadas em um cluster. A tarefa de mapeamento lê dados de um agregado e os agrupa em partes chave-valor relevante. Mapeamentos somente leem um único registro de cada vez e podem, assim, ser paralelizados e executados no nodo que armazena o registro.

A tarefa de redução recebe muitos valores de uma única chave de saída, a partir da tarefa de mapeamento, e os resume em uma única saída. Cada redutora trabalha sobre o resultado de uma única chave, de modo que podem ser paralelizados por chave.

MapReduce é complicado de entender no início, por isso vamos tentar compreendê-lo com um exemplo simples. Vamos supor que temos um arquivo que tem algumas palavras e o arquivo é dividido em blocos, e temos que contar o número de ocorrências de uma palavra no arquivo. Iremos passo a passo para alcançar o resultado usando a funcionalidade MapReduce. Todo o processo é ilustrado no diagrama a seguir:



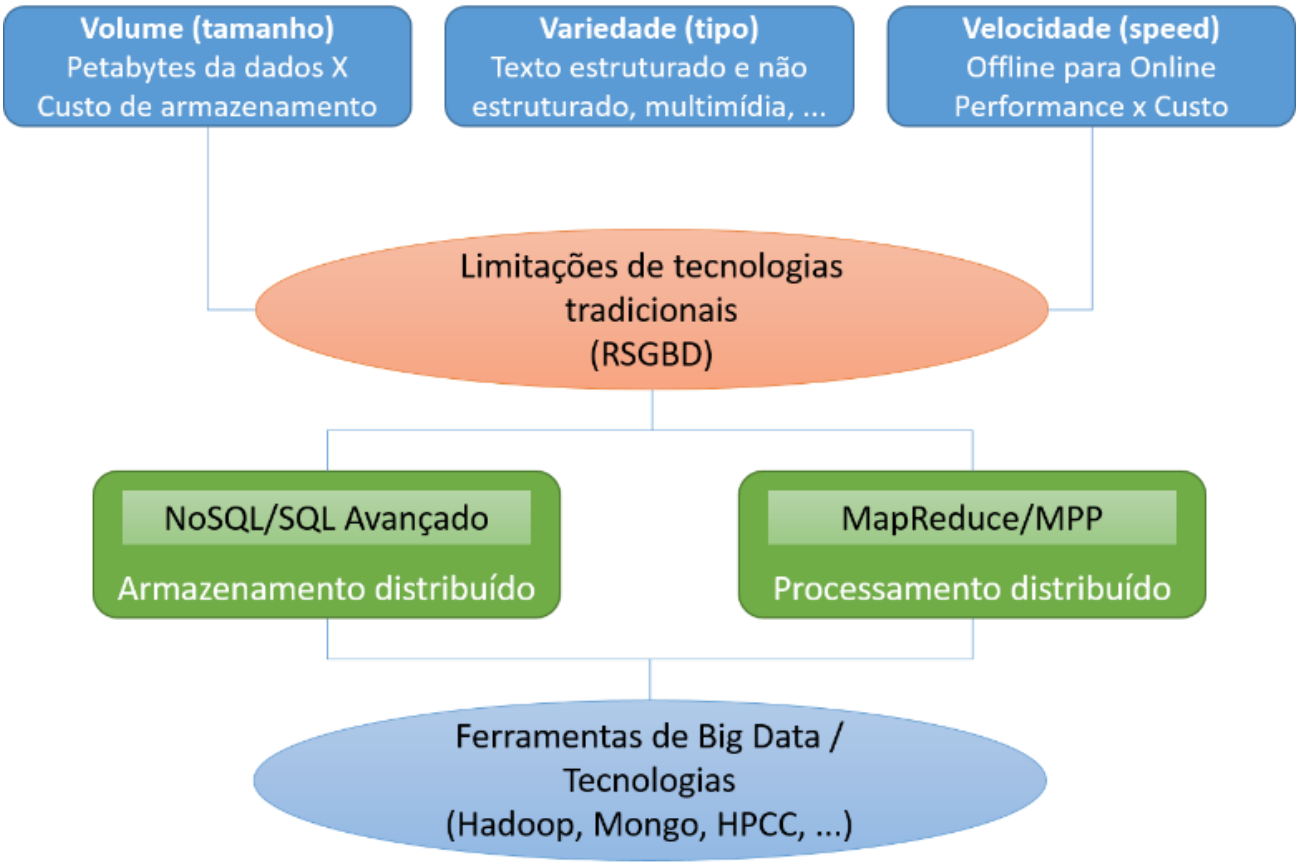
Elementos que tenham o mesmo formato de entrada e saída podem ser combinadas em pipelines. Isso melhora o paralelismo e reduz a quantidade de dados a serem transferidos. Operações de MapReduce podem ser compostas em pipelines, nos quais a saída de uma redução é a entrada do mapeamento de outra operação.

Se o resultado de uma computação MapReduce for amplamente utilizado, pode ser armazenado como uma visão materializada. Visões materializadas podem ser atualizadas por meio de operações MapReduce que apenas computem alterações na visão, em vez de computar novamente tudo desde o início.

APOSTA ESTRATÉGICA

A ideia desta seção é apresentar os pontos do conteúdo que mais possuem chances de serem cobrados em prova, considerando o histórico de questões da banca em provas de nível semelhante à nossa, bem como as inovações no conteúdo, na legislação e nos entendimentos doutrinários e jurisprudenciais[1].

[1] Vale deixar claro que nem sempre será possível realizar uma aposta estratégica para um determinado assunto, considerando que às vezes não é viável identificar os pontos mais prováveis de serem cobrados a partir de critérios objetivos ou minimamente razoáveis.



Imprima o capítulo Aposta Estratégica separadamente e dedique um tempo para absolver tudo o que está destacado nessas duas páginas. Caso tenha alguma dúvida, volte ao Roteiro de Revisão e Pontos do Assunto que Merecem Destaque. Se ainda assim restar alguma dúvida, não hesite em me perguntar no fórum.

QUESTÕES ESTRATÉGICAS

Nesta seção, apresentamos e comentamos uma amostra de questões objetivas selecionadas estrategicamente: são questões com nível de dificuldade semelhante ao que você deve esperar para a sua prova e que, em conjunto, abordam os principais pontos do assunto.

A ideia, aqui, não é que você fixe o conteúdo por meio de uma bateria extensa de questões, mas que você faça uma boa revisão global do assunto a partir de, relativamente, poucas questões.

No último concurso do Banco do Brasil, a CESGRANRIO anulou as duas questões sobre esse tema.

1. CESGRANRIO - Escriturário (BB)/Agente de Tecnologia/2021

Um banco comercial deseja obter um tipo de banco de dados NoSQL que trate os dados extraídos de redes sociais, de modo a formar uma coleção (collection) interconectada. Nessa coleção (collection), os dados são organizados em vértices ou objetos (O) e em relacionamentos, que são relações (R) ou arestas.

Nesse modelo de banco de dados NoSQL, os dados seriam apresentados da seguinte forma:

O:Usuario{u1:Joao, u2:Jose, u3:Maria, u4:Claudio}
O:Escola{e1:UFRJ, e2:URGS, e3:IFB}
R:Estudaem{re1=u1:e2;re2=u2:e2;re3=u3:e1;re4=u4:e3}
R:Amigode{ra1=u1:u2;ra2=u1:u3;ra3=u2:u3}

O banco de dados NoSQL que representa essa situação deve ter uma estrutura do tipo

A) Distribuided Hashing
B) Consistent Hashing
C) Document Oriented
D) Graph Oriented
E) Vector Clock

Comentários:

Quando há essa definição de vértices e arestas, estamos comentando de um SGBD orientado a grafo.

Esse tipo de SGBD é muito usado em redes sociais. Um exemplo clássico de SGBD do tipo grafo é o Neo4J.

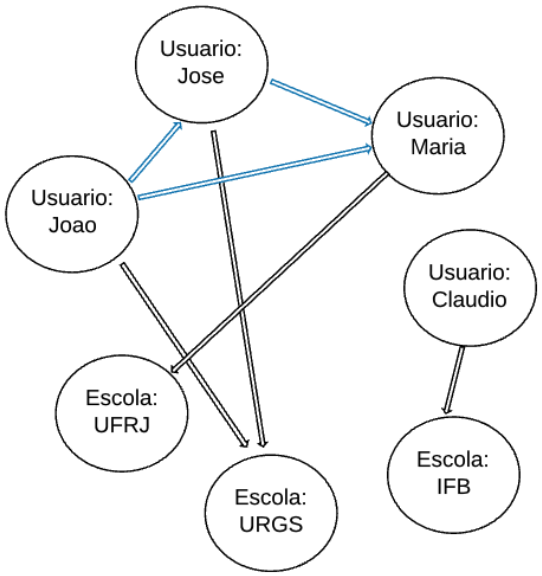
Na questão, temos:

O:Usuario{u1:Joao, u2:Jose, u3:Maria, u4:Claudio}

O:Escola{e1:UFRJ, e2:URGS, e3:IFB}

R:Estuda em{re1=u1:e2;re2=u2:e2;re3=u3:e1;re4=u4:e3}

R:Amigo de{ra1=u1:u2;ra2=u1:u3;ra3=u2:u3}



Gabarito alternativa D.

---

2. CESGRANRIO - Escriturário (BB)/Agente de Tecnologia/2021

Um administrador de um banco de dados construído por meio do MongoDB inseriu dados em uma coleção (collection) de dados da seguinte forma:

```
db.fornecedores.insert( {
    codigo: “thx1138“,
    nome: “Roupas Syfy ltda“,
    pais: “Arabia Saudita“ } )
```

Posteriormente, esse administrador construiu uma consulta que retornou apenas o nome, sem repetição, de todos os países que fazem parte dessa coleção (collection).

O comando utilizado para tal consulta foi

A) db.fornecedores.find("pais")

B) db.fornecedores.find().pretty({"pais":1})

C) db.fornecedores.find().sort({"pais":1})

D) db.fornecedores.distinct({"pais":0})

E) db.fornecedores.distinct("pais")

Comentários:

De acordo com a documentação do MongoDB, temos:

- db.collection.distinct()

Defnição:

Encontra os valores distintos para um campo especificado em uma única coleção ou visualização e retorna os resultados em um array.

Gabarito: alternativa E.

...

Forte abraço e bons estudos.

"Hoje, o 'Eu não sei', se tornou o 'Eu ainda não sei'"

(Bill Gates)

Thiago Cavalcanti

Face: www.facebook.com/profthiagocavalcanti
Insta: www.instagram.com/prof.thiago.cavalcanti
YouTube: youtube.com/profthiagocavalcanti

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.