INSTITUTO DE GESTÃO E TECNOLOGIA DA INFORMAÇÃO

Armazenamento de Dados

Engenharia de Dados

Marcílio Andrade

2021

Armazenamento de Dados

Engenharia de Dados

Marcílio Andrade

Sumário

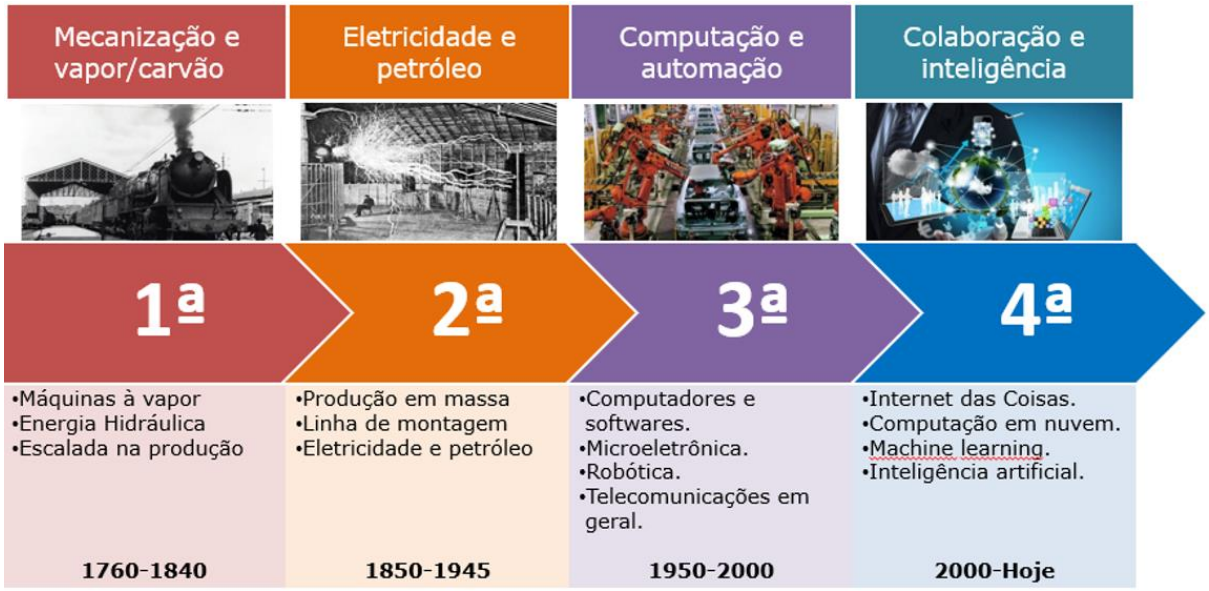
Capítulo 1. Introdução

Neste capítulo introdutório, busca-se uma contextualização geral no que diz respeito tanto a aspectos sociais quanto tecnológicos que, de alguma forma, influenciam nas demandas de armazenamento de dados nos dias de hoje. Para tanto, discute-se alguns marcos históricos da humanidade com sua influência na concepção da atual sociedade da informação. Além disso, apresenta-se a evolução tecnológica com alguns de seus mais importantes marcos, bem como discute-se a organização orientada a dados. Tudo isso em meio ao fenômeno de *Big Data* que também é descrito no texto que se segue.

1.1. Sociedade da Informação

Ao observar as revoluções industriais, fica mais fácil entender as diversas transformações pelas quais a humanidade passou ao longo dos últimos séculos. E, de alguma forma, como isso reflete na tecnologia que, por sua vez, volta a influenciar na própria sociedade.

Figura 1 – Revoluções industriais



Fonte: Fernanda Farinelli (2018)

O marco inicial foi a primeira revolução industrial. Antes dela, todo o processo de fabricação era basicamente artesanal. Com o uso da energia, sobretudo oriunda do carvão, e o advento da máquina a vapor, a humanidade começou a construir equipamentos, ainda simples e rudimentares, mas que já permitiam um certo grau de mecanização. Isso trouxe a capacidade de uma certa escala de produção que, até então, era difícil de ser alcançada somente com o emprego de trabalho manual. Etapa que vai de 1760 até por volta de 1840.

Em seguida, vem a segunda revolução industrial. Essa, marcada, sobretudo, pelo advento da energia elétrica e pelo uso do petróleo. Essas passam a ser as principais fontes de energia. Aparece, então, de forma efetiva, a produção em massa fortemente representada pelas linhas de montagem da indústria automobilística capitaneada por Henry Ford. Fase que se estende de 1850 a 1945.

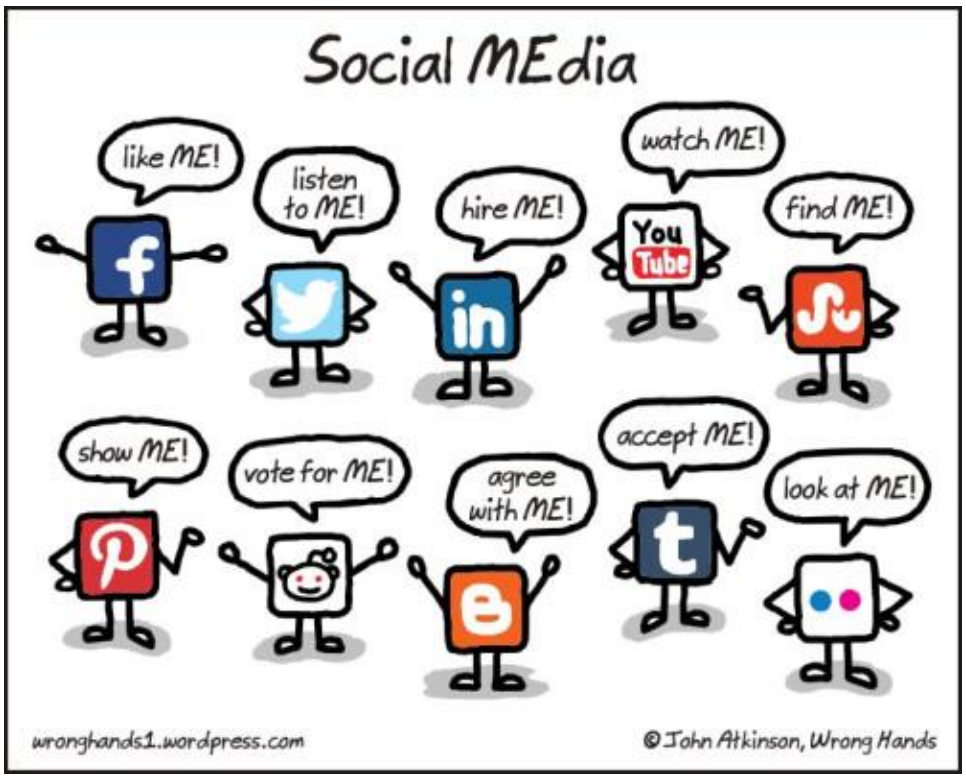
Chega-se, então, à terceira revolução. Essa, por sua vez, baseada principalmente em computação e automação. Como pontos chave, aparecem a microeletrônica e a robótica dando vida a linhas de produções altamente robotizadas. O avanço das telecomunicações e da computação, tanto no que diz respeito a *hardware* quanto *software*, também desempenham papel de destaque nessa fase que vai de 1950 a 2000.

Por fim, do ano 2000 em diante, tem-se a quarta revolução industrial. Essa, marcada sobretudo por colaboração e inteligência. As máquinas começam a ganhar capacidade além da mera repetição eficientemente de processos. Passam a ser capazes até de melhorar esses mesmos processos por meio de aprendizado baseado em inteligência artificial e *machine learning*. Esses equipamentos passam, também, a ser produtores massivos de dados que são transmitidos, armazenados e tratados no contexto da Internet das Coisas (IoT) e com suporte de computação em nuvem.

Ao observar o dia a dia, fica ainda mais fácil entender como são grandes as transformações vivenciadas pela sociedade nos últimos anos. A simples comparação entre videogames de primeira e última geração é capaz de mostrar isso. A forma como se consumia música anos atrás e como isso e feito hoje também é um outro aspecto bastante elucidativo. As redes sociais e sua influência na interação entre as

pessoas também demonstra com clareza as mudanças a que todos estão submetidos.

Figura 2 – Transformações da sociedade



Fonte: https://wronghands1.com/social-media-3/

Em meio a tudo isso, os especialistas apontam que, com o passar dos anos, vão formando-se algumas gerações que são bastante distintas entre si. De forma que é possível ver, dentro das empresas, a necessidade de que pessoas de gerações distintas, e, portanto, de comportamentos e vivências bastante heterogêneos, atuem de forma cooperativa.

Diante de todo esse contexto, tem-se uma coletividade que busca cada vez mais acesso a dados que permitam produzir informações úteis às rotineiras tomadas de decisões. Portanto, informação passa a ser algo fundamental, indispensável, o que passa a caracterizar uma sociedade da informação que impulsiona fortemente a evolução tecnológica.

1.2. Evolução Tecnológica

Quando se pensa na evolução tecnológica de forma geral, é possível perceber como isso influencia nas formas de se tratar o armazenamento de dados. Nesse sentido, tem papel fundamental os avanços na computação, na infraestrutura de tecnologia, computação em nuvem e nas metodologias e regulamentações corporativas (*compliance*).

Em termos computacionais, pode-se entender o *mainframe* como sendo o primeiro marco na rápida escalada que se seguiu a partir daí. Grandes máquinas que tinham como objetivo principal o processamento para resolução de problemas matemáticos que até então eram difíceis de serem resolvidos.

Na sequência, surgem os bancos de dados. Até então, toda aplicação que era desenvolvida era responsável por gerenciar a manipulação dos dados, ficando responsável por controlar aspectos como concorrência, segurança, recuperação, dentre diversos outros. A partir do uso dessa tecnologia, passa a não ser mais necessária a interação direta da aplicação com os meios físicos de armazenamento, o que representou uma grande evolução.

Outro importante salto vem com a evolução das redes de computadores. Isso permite a distribuição, e os sistemas passam a comunicar-se possibilitando uma interação bastante sofisticada.

Uma outra evolução bastante relevante surge na sequência. As máquinas que estavam, até então, à disposição somente de grandes centros de pesquisa e empresas, passam a estar também disponíveis dentro dos lares com a popularização dos computadores pessoais (PC). Há, a partir de então, capacidade computacional à disposição das famílias, seja para edição de texto, criação de planilhas ou para lazer.

Contudo, esses computadores pessoais tornam-se realmente relevantes no contexto da sociedade da informação quando passam a ser um instrumento de entrada ao novo mundo disponibilizado pela internet. Isso torna possível o acesso a uma quantidade de dados que, até então, era inimaginável.

Não bastasse o acesso a esse conteúdo a partir de casa ou de uma *lan house*, as tecnologias móveis vêm possibilitar o acesso mesmo fora desses locais, enquanto se está em trânsito com uso de um aparelho celular, por exemplo. Então, passa a ser possível ter capacidade computacional disponível em tempo integral e com acesso à rede mundial de computadores.

As redes sociais também representam um marco nessa história. O mais relevante aspecto, nesse sentido, é a forte influência sobre a forma das pessoas interagirem entre si. Conforme já dito anteriormente, hoje tem-se praticamente uma rede social para cada necessidade de convivência, seja ela pessoal, profissional etc.

Na sequência, a computação em nuvem (*cloud computing*) traz a possibilidade de acesso a recursos computacionais sofisticados e com grande capacidade. Isso sem a necessidade de um investimento total realizado de forma antecipada e possível de ser conduzido com um time de Tecnologia da Informação (TI) não tão grande ou especializado como era exigido anteriormente. Nessa modalidade, pode-se contratar tais recursos pagando-se como um serviço e livre de certas necessidades de implementação de *hardware* e *software*.

Mais recentemente, observa-se a contínua evolução da conectividade. Fala-se em tecnologias 5G com países já trabalho 6G e tudo isso trazendo uma capacidade de tráfego de dados até então desconhecida. O que possibilita o que tem se chamado de Internet das Coisas (IoT) onde cada dispositivo ligado à internet é capaz de gerar dados que, ao serem armazenados e analisados, geram diversas novas possibilidades.

Figura 3 – Evolução computacional

Fonte: Fernanda Farinelli (2019)

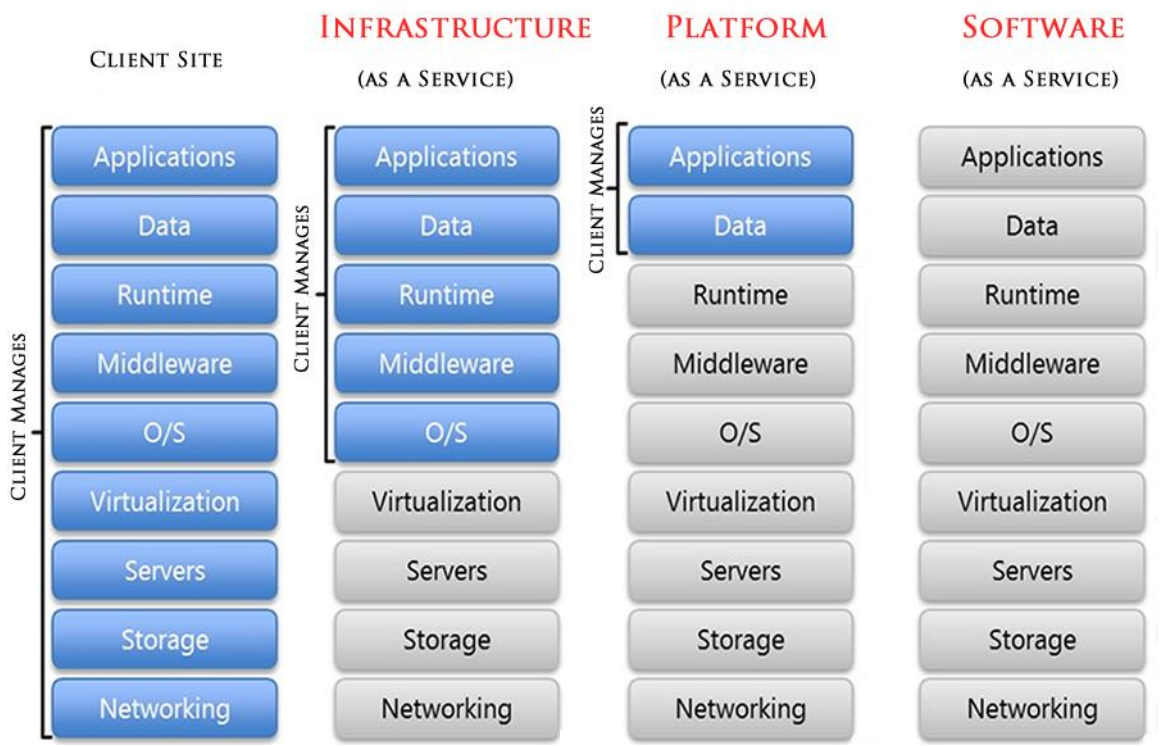
Outro aspecto relevante diz respeito à infraestrutura de tecnologia que também apresenta mudanças bastante significativas ao longo do tempo. Até algum tempo atrás, era necessário adquirir-se um novo *hardware* para cada novo sistema. Com isso, sendo necessário também toda a instalação e configuração desse equipamento. O resultado era um grande parque de máquinas heterogêneas a se administrar dentro das organizações. A primeira forma de tratar esse problema foi a virtualização onde, ao invés de se ter vários equipamentos, adquire-se uma quantidade menor de servidores de maior capacidade e passa-se a criar e administrar servidores virtuais.

Em seguida, passa-se a adotar toda a infraestrutura de tecnologia baseada em *software*. Agora, não somente os servidores, mas praticamente toda a infraestrutura fica passível de ser virtualizada e controlada por meio de *software*. Na necessidade de criar-se uma rede de comunicação, por exemplo, cria-se um roteador lógico sem a necessidade de aquisição de um novo roteador físico que traria, também, a necessidade de um novo cabeamento.

Ainda em termos de virtualização, as tecnologias de containers (Docker, Kubernets) possibilitaram que as funcionalidades da aplicação e seus serviços fossem também virtualizadas. O que minimiza a preocupação com questões relacionadas a sistemas operacionais dentre outras questões que simplificam e trazem escalabilidade a todo o ambiente.

A computação em nuvem também impacta fortemente a forma com que se consume e produz tecnologia nos dias de hoje. Em ambientes tradicionais, chamados também ditos *on premises*, todo parque estava instalado dentro da empresa ou em *data center*, mas sob total administração da organização. Com *cloud computing*, os recursos passam a estar disponíveis por meio de uma rede de comunicação com parte ou até toda a administração sob o controle de um fornecedor externo.

Figura 4 – Computação em nuvem



Fonte: media.licdn.com, 2017

Como IaaS, que é a infraestrutura como serviço, deixa de existir a necessidade de aquisição e configuração de recursos básicos, tais como servidores e roteadores. Na outra ponta, há o SaaS, software como serviço. Nesse caso, tem-se uma aplicação funcionando sem que haja qualquer tipo de administração com as camadas que a suportam. Como ponto intermediário entre esses dois extremos, tem-se PaaS, plataforma como serviço, onde além das camadas básicas, o fornecedor provê, também, a configuração e disponibilização da plataforma como um todo.

Há, por fim, um conjunto de novas metodologias e regras de conformidade (*Compliance*) que pressionam a forma com que se lida com as tecnologias utilizadas no desenvolvimento de aplicações. Metodologias ágeis, processos DevOps, LGPD e as disciplinas de gestão e governança de dados fazem com que o processo de concepção de *software,* e, por consequência, o armazenamento de dados para tais sistemas, sejam sensivelmente diferentes quando comparados com o que eram feitos anos atrás.

1.3. Organização Orientada a Dados

William Deming foi um matemático e estatístico norte-americano que teve grande importância na reconstrução da indústria japonesa após a Segunda Guerra Mundial. Ele já trazia, naquela época, a necessidade de que as organizações tomassem decisões baseadas em dados. Então, essa ideia antiga vem de encontro a já citada sociedade da informação.

Assim, pode-se dizer que a ideia de ser orientado a dados vem justamente da necessidade de embasar-se a sua tomada de decisão em dados previamente coletados, armazenados e analisados. Então, uma organização dita orientada a dados, é aquela que, em um primeiro momento, armazena digitalmente seus dados de forma organizada e coerente. Mas, não somente isso. Essa organização produz relatórios sobre esses dados e realiza análises sobre eles. A partir daí, promove ações baseadas nessas análises para trazer benefícios, agregando valor ao negócio.

Para tanto, é necessário desenvolver-se o que se chama de cultura orientada a dados. O que exige que cada setor e cada pessoa da organização seja conscientizada e capacitada a lidar de forma eficiente com os dados que são produzidos, armazenados, tratados e analisados em favor da empresa. Somente assim desenvolve-se uma cultura orientada a dados e chega-se à visão de que os dados representam realmente um ativo valioso e fundamental para realização de cada atividade. Evidentemente, esse é um processo gradual que pode, também, demandar recursos que sejam capazes de promover a mudança de hábitos e atitudes dentro das empresas.

Com foco nisso, há o Manifesto dos Líderes de Dados, documento lançado em 2017 durante uma conferência organizada pela *Data Management Association International* (DAMA), que registra e descreve a importância do dado no contexto atual das organizações. Em conformidade com a ideia de que o dado é o novo petróleo e o maior ativo da organização, esse manifesto tenta trazer conscientização aos líderes organizacionais de forma a promover a cultura orientada a dados.

1.4. Big Data

O termo *Big Data* surgiu em 1997 em um artigo de cientistas da Nasa, e a ideia era abordar desafios computacionais de captura, processamento, análise e armazenamento para grandes volumes de dados complexos. Contudo, o termo realmente foi popularizado de uma década para cá descrevendo enormes quantidades de dados no dia a dia das pessoas e das empresas. Para buscar-se a correta compreensão do assunto, talvez seja importante dizer que não se trata de um produto de *hardware*, tão pouco um *software* de prateleira ou uma metodologia.

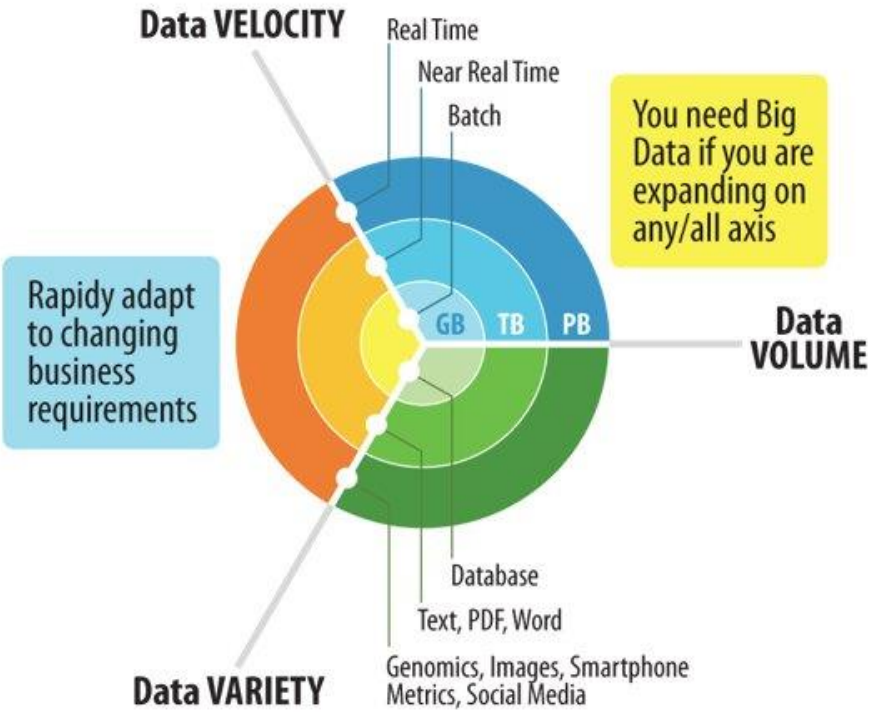
De forma prática, *Big Data* refere-se usualmente a *datasets* com tamanho além da capacidade de processamento por ferramentas usuais de software dentro de prazos que possibilitem o apoio à tomada de decisão. Portanto, engloba capacidade, tamanho e tempo para o tratamento dos dados. Nesse sentido, é importante destacar que o tamanho para ser considerado *Big Data* é transitório, sendo, em 2012, poucos

terabytes e, atualmente, talvez centenas de petabytes. Também pode ainda ser visto um conjunto de técnicas e tecnologias que requerem novas abordagens e métodos para se buscar o valor em conjuntos de dados diversificados, complexos e em escala massiva.

Ao longo do tempo, surgiram novos desafios que demandaram técnicas de *Big Data* para serem superados. Em primeiro lugar, os dados passaram a ser gerados de maneira bastante aleatória e vindos de fonte de dados muitas vezes desconhecidas. O que também traz preocupações a respeito da confiabilidade desses dados. Além disso, começaram a ser produzidos em formatos não amigáveis que exigiram e continuam exigindo novas técnicas de tratamento e manipulação.

Ao analisar-se essas questões, percebeu-se que tais desafios podiam ser agrupados em basicamente três categorias: volume, variedade de tipos e velocidade de produção dos dados. O que deu origem e se convencionou chamar de “os três vês do *Big Data*”.

Figura 5 – Os vês do *Big Data*



Fonte: https://twitter.com/kdnuggets/status/632193542685859844

Então, fala-se muito em volume de dados. Contudo, além do volume, há também, uma maior variedade de tipos de dados que são produzidos a uma velocidade muito maior que observada há anos. O que nos leva à necessidade de novas técnicas e tecnologias para o tratamento apropriado de dados em tempo hábil para as organizações.

No que diz respeito ao volume de dados, há vários estudos que mostram a escalada exponencial que já vem ocorrendo e que deve continuar ao longo dos próximos anos. Quando se pensa em velocidade de produção, pode ser útil verificar o que ocorre na internet a cada minuto: 4,7 milhões de vídeos assistidos no YouTube, 194 mil pessoas postando no Twitter, 764 mil horas de vídeos assistidos no Netflix são apenas alguns exemplos de dados consolidados no ano de 2020. Sobre a variedade, além dos já conhecidos dados estruturados, representados por tabelas de bancos de dados e planilhas onde há uma clara definição da estrutura do que está armazenado, passa-se a lidar, também, com dados semiestruturados, arquivos XML e JSON, e os não estruturados, como fotos, vídeos, dentre outros.

Para viabilizar tudo isso, há perfis profissionais que passam a ser exigidos dentro das organizações. Três dos principais deles sendo o cientista de dados, o engenheiro de dados e o analista de *machine learning*. Cada um deles com características e capacidades distintas que precisam, muitas vezes, trabalhar em conjunto para fornecer um ambiente de *Big Data* adequado.

Capítulo 2. Banco de Dados

Considerando que os bancos de dados têm papel central quando o assunto é armazenamento de dados, faz-se necessário recapitular alguns de seus conceitos fundamentais. Além disso, a correta compreensão dos mecanismos de controle de transação e distribuição de dados coloca-se como ponto importante para a sequência dos estudos propostos. A compreensão da evolução dos sistemas gerenciadores de banco de dados e a correta visão de mercado dessas ferramentas complementam os assuntos abordados neste capítulo.

2.1. Conceitos de Banco de Dados

Para uma correta compreensão dos aspectos conceituais de banco de dados, é importante buscar e citar alguns dos principais estudiosos do assunto. Segundo Elmasri e Navathe (2016), um banco de dados é uma coleção de dados relacionados. Com dados referindo-se a fatos conhecidos que podem ser registrados e possuem significado implícito. Contudo, ainda de acordo com eles, o uso comum do termo banco de dados normalmente é mais restrito e tem as seguintes propriedades implícitas:

- Representa algum aspecto do mundo real. As mudanças no minimundo são refletidas no banco de dados.
- Coleção logicamente coerente de dados com algum significado inerente. Uma variedade aleatória de dados não pode ser corretamente chamada de banco de dados.
- Projetado, construído e populado com dados para uma finalidade específica. Ele possui um grupo definido de usuários e algumas aplicações previamente concebidas nas quais esses usuários estão interessados.

Apontam, ainda, que um sistema gerenciador de banco de dados (SGBD ou DBMS - *Database Management System*) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. O SGBD é um sistema de software

de uso geral que facilita o processo de definição, construção, manipulação e compartilhamento de bancos de dados entre diversos usuários e aplicações. Já a união do banco de dados com o software de SGBD é chamado de sistema de banco de dados.

Antes que existisse um sistema gerenciador de banco de dados, todos os aspectos de acesso aos dados, tais como manipulação e segurança, precisavam ser desenvolvidos para cada uma das aplicações criadas. Essa abordagem chama-se sistema de processamento de arquivos e nela cada analista desenvolvedor define e implementa os arquivos necessários para uma aplicação de software específico como parte da programação da aplicação. As dificuldades óbvias desse modelo são:

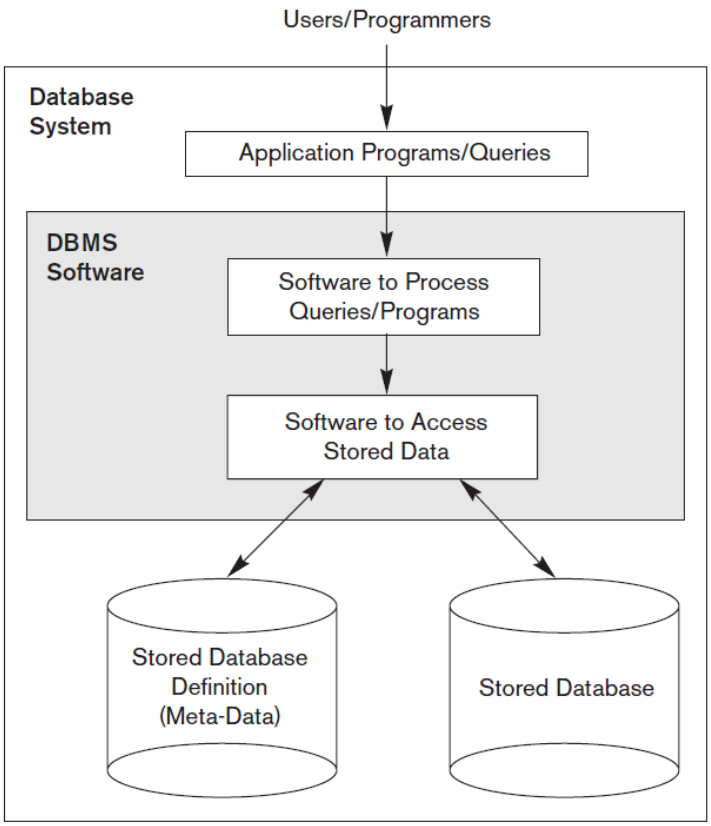
- Redundância e inconsistência de dados.
- Dificuldade no acesso aos dados.
- Isolamento de dados.
- Anomalias de acesso concorrente.
- Problemas de segurança.
- Problemas de integridade.
- Escalabilidade.

A Figura 6 mostra a arquitetura de alto nível de um SGBD. Nela, é possível observar que os usuários e os programadores interagem com o sistema de banco de dados que engloba o próprio banco de dados, o sistema gerenciador de banco de dados e as aplicações que se conectam a ele. Especificamente em relação ao SGBD, é possível identificar dois grandes componentes: o primeiro é responsável pelo processamento dos comandos que chegam, e o outro é dedicado ao acesso físico aos dados que estão armazenados.

Há dois tipos distintos de carga de trabalho que podem ser atendidos por um SGBD: transacionais e analíticos. Os bancos com foco no dia a dia das organizações dão suporte operacional ao negócio e precisam ter características específicas para lidarem com essa demanda. Esses sistemas chamados transacionais, ou *online transactional process* (OLTP), apresentam as seguintes características principais:

- Dados normalizados.
- Grande volume transacional (muitas atualizações).
- Diagrama de entidade relacionamento.
- Emissão de relatórios restritos e pouco flexíveis.
- Dificuldade na elaboração de consultas.
- Dificuldade na integração de ambientes heterogêneos.

Figura 6 – Arquitetura básica de um SGBD



Fonte: ELMASRI e NAVATHE (2016)

Já os sistemas que fornecem suporte à tomada de decisão são chamados analíticos, ou *online analytical process* (OLAP). Para essa categoria, as características necessárias são outras, conforme o seguinte:

- Dados orientados à análise de informação.
- Baixo volume transacional (poucas e controladas atualizações).
- Altamente consultado.

- Modelagem Dimensional.
- Integram dados originados de diversas fontes distintas.

2.2. Transação no SGBD

De acordo com Coronel e Morris (2018), uma transação é uma unidade lógica de trabalho que deve ser totalmente concluída ou totalmente cancelada; nenhum estado intermediário é aceitável. Em outras palavras, pode-se dizer que todas as instruções na transação devem ser concluídas com sucesso. Se alguma delas falhar, toda a transação será revertida para o estado original do banco de dados que existia antes do início da transação.

Importante ainda dizer que um SGBD deve suportar diversas transações sendo realizadas ao mesmo tempo (transações concorrentes) e que uma transação bem-sucedida altera o banco de dados de um estado consistente para outro. Nesse sentido, um estado de banco de dados consistente é aquele em que todas as restrições de integridade de dados são satisfeitas, sendo o SGBD projetado para recuperar um banco de dados para um estado consistente anterior quando uma interrupção impede a conclusão de uma transação.

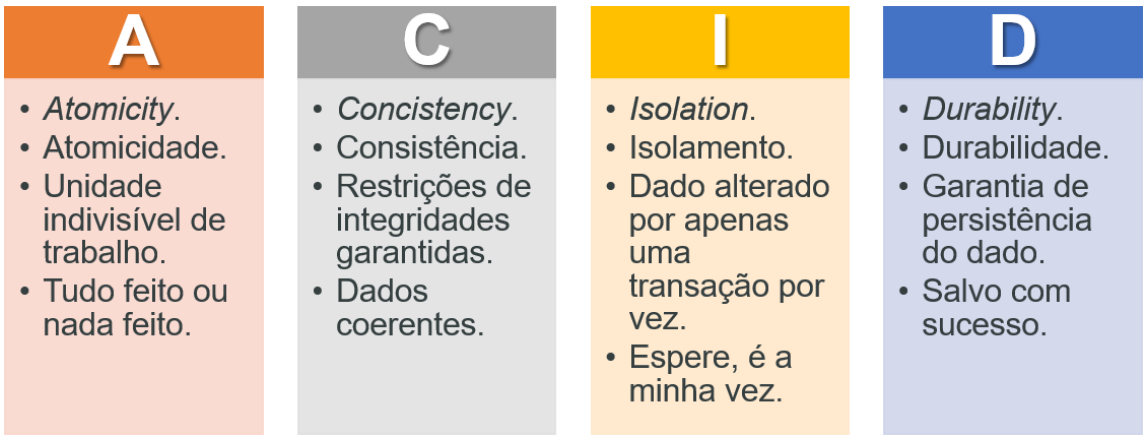
Em bancos de dados que suportam múltiplos usuários conectados ao mesmo tempo, o que geralmente é necessário, as diversas transações simultâneas devem criar o mesmo estado final do banco de dados que seria produzido com transações executadas em série. Para isso, é utilizado um log de transações para registrar todas as que atualizam o banco de dados, e uma parte da linguagem SQL que apresenta comandos próprios para controle dessas transações:

- BEGIN TRANSACTION: responsável por iniciar uma transação.
- COMMIT: comando para informar ao banco que a transação terminou com sucesso e deve ser confirmada.
- ROLLBACK: usado para indicar alguma falha e, portanto, comandar a reversão a um estado consistente anterior.

Para que sejam efetivas no controle de concorrência, as transações precisam apresentar algumas propriedades fundamentais: atomicidade, consistência,

isolamento e durabilidade. Essas propriedades formam o acrônimo ACID que é bastante conhecido e debatido na literatura de banco de dados e indicam o seguinte:

Figura 7 – Propriedades da transação



Fonte: Dados da pesquisa

Há algumas técnicas possíveis de serem utilizadas na implementação de SGBD para garantir o controle de concorrência. O uso de bloqueio é uma das mais comumente utilizadas, porque facilitam o isolamento de dados usados em transações simultâneas. Um bloqueio garante o uso exclusivo de um item de dados para uma transação que deve solicitar e adquirir um bloqueio antes de manipular dados. Tal bloqueio é liberado após a transação ser concluída, de forma que uma outra sessão possa solicitar aquele mesmo dado para seu uso exclusivo.

Existe um tipo especial de bloqueio chamado de impasse ou *deadlock*. Trata-se de uma situação em que, a menos que o SGBD intervenha, uma situação de bloqueio irá perdurar de forma indefinida. Para entendimento, basta imaginar um cenário onde duas transações, T1 e T2, são iniciadas:

- T1 solicita o bloqueio de um determinado dado D1 para atualização de seu valor.
- Enquanto isso, T2 bloqueia outro dado D2 com o objetivo de alterá-lo.
- Na sequência, T1 solicita acesso a D2 já bloqueado para T2. Até esse momento, nada demais. O SGBD faz com que T1 aguarde a liberação de D2 por parte de T2.

- Porém, a próxima etapa de T2 é justamente solicitar acesso a D1, bloqueado por T1.

Ou seja, temos T1 bloqueando D1 e querendo acesso a D2, enquanto T2 bloqueia D2 e deseja acesso a D1. Aí está formado o *deadlock*. Nesse momento, o SGBD, baseado em um algoritmo próprio, escolhe uma dessas transações para ser eliminada e, portanto, realizar o *rollback*. A outra então pode prosseguir até a finalização normal de sua execução com o *commit*.

Outro aspecto relevante relacionado ao controle de transações são os níveis de isolamento dos bancos de dados. Esses níveis indicam basicamente como o SGBD trata a exclusividade de acesso a um determinado dado que está bloqueado para uma determinada transação. Há isolamentos mais permissivos que permitem a leitura do dado bloqueado e outros mais restritivos impedem tanto escrita quanto leitura esse dado. Dependendo da escolha, passa a ser possível ou não a ocorrência de alguns fenômenos como leitura suja (*dirty read*), leitura não repetível (*nonrepeatable read*) e leitura fantasma (*phantom read*). Tudo conforme o seguinte:

Figura 8 – Níveis de isolamento

TABLE 10.15

TRANSACTION ISOLATION LEVELS

	ISOLATION LEVEL	ALLOWED			COMMENT
		DIRTY READ	NONREPEATABLE READ	PHANTOM READ	
Less restrictive	Read Uncommitted	Y	Y	Y	The transaction reads uncommitted data, allows nonrepeatable reads, and phantom reads.
	Read Committed	N	Y	Y	Does not allow uncommitted data reads but allows nonrepeatable reads and phantom reads.
	Repeatable Read	N	N	Y	Only allows phantom reads.
More restrictive	Serializable	N	N	N	Does not allow dirty reads, nonrepeatable reads, or phantom reads.
Oracle / SQL Server Only	Read Only / Snapshot	N	N	N	Supported by Oracle and SQL Server. The transaction can only see the changes that were committed at the time the transaction started.

Fonte: Coronel e Morris (2020)

2.3. Banco de Dados Distribuído

Para a melhor compreensão dos mecanismos utilizados por bancos de dados distribuídos, cujo conceito e características serão apresentados mais adiante, é importante a compreensão antecipada de alguns importantes requisitos de sistemas computacionais: escalabilidade, elasticidade e disponibilidade.

Escalabilidade pode ser compreendida como a capacidade do sistema em acomodar cargas maiores apenas adicionando recursos computacionais. Quando o caminho é por melhorar os recursos existentes, a escalabilidade é chamada de vertical (*scale up*). Por outro lado, ao adicionar-se novos recursos aos já existentes, diz que a escalabilidade foi alcançada de forma horizontal (*scale out*).

Quando um sistema é capaz de ajustar dinamicamente os recursos necessários para lidar com cargas de trabalho variáveis, diz tratar-se de um sistema elástico. Essa elasticidade pode ser alcançada tanto com escalabilidade vertical quanto horizontal. Contudo, na maioria das vezes, apenas a horizontal proporciona que isso seja realizado sem que o sistema precise ficar indisponível para os usuários.

O outro requisito muito importante para sistemas de computadores é justamente a disponibilidade. Ela pode ser entendida como a capacidade da aplicação permanecer em funcionamento mesmo em caso de falhas em seus componentes. É normalmente medida em função da quantidade anual de tempo em que a aplicação não pode ser acessada.

Realizada essa explanação, é possível avançar com o apoio de Coronel e Morris (2018) dizendo que um Sistema Gerenciado de Banco de Dados Distribuído (SGBDD) controla o armazenamento e o processamento de dados relacionados logicamente em sistemas de computador interconectados, nos quais os dados e o processamento são distribuídos entre vários locais.

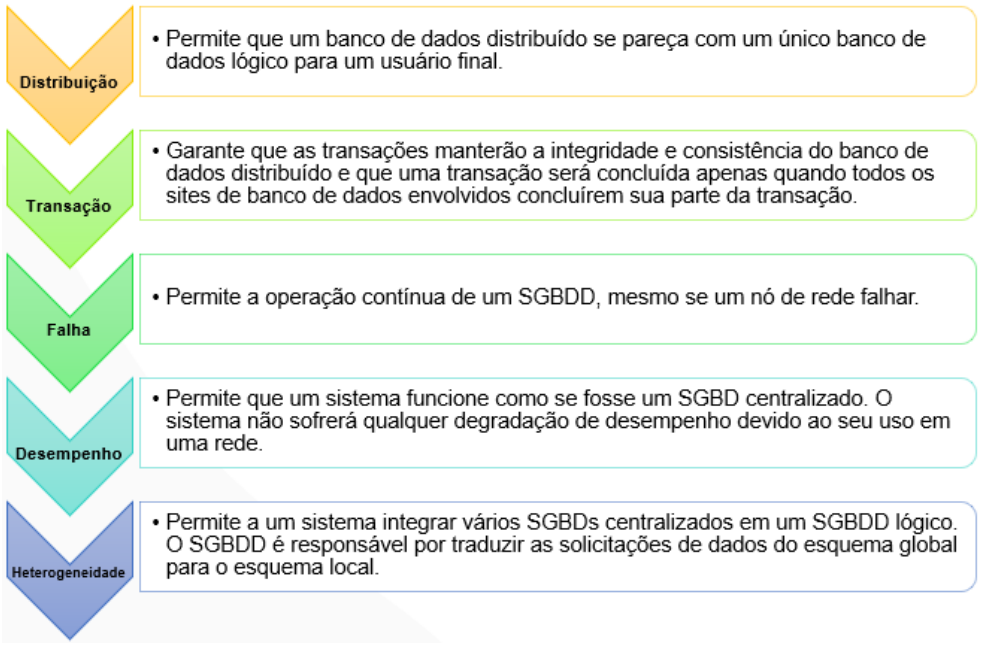
Considerando que o SGBDD trata um banco de dados distribuído como um único banco de dados lógico, fica claro perceber que a distribuição entre diferentes sites em uma rede de computadores adiciona complexidade ao sistema. Complexidade que deve ser transparente para o usuário final.

Os conceitos básicos de dados distribuídos aplicam-se tanto a bancos SQL (relacionais) quanto a bancos NoSQL e visam trazer alternativas a algumas dores apresentadas pelo banco de dados centralizado:

- Degradação de desempenho devido a um número crescente de locais remotos em distâncias maiores.
- Problemas de confiabilidade criados pela dependência de um site central (ponto único de falha).
- Problemas de escalabilidade associados aos limites físicos impostos por um único local, como espaço físico, refrigeração e consumo de energia.

A forma com que o SGBD lida com a distribuição deve ser como uma caixa preta. Ou seja, para a aplicação, não deve recair qualquer preocupação de como os mecanismos de banco de dados estão implementando essa distribuição. Em outras palavras, o SGBD deve prover os recursos de transparência destacados na Figura 9.

Figura 9 – Recursos de transparência do SGBDD



Fonte: Dados da pesquisa

Há duas principais técnicas de distribuição de dados: fragmentação (*sharding*) e replicação. Cada uma delas com suas características e casos de uso. A

fragmentação permite que um único objeto seja dividido em dois ou mais segmentos, cada qual armazenado em um nó, sendo de três tipos possíveis:

- Horizontal:
  - Tabela dividida em subconjuntos (fragmentos) de linhas.
  - Todas as linhas têm os mesmos atributos (colunas).
- Vertical:
  - Tabela dividida em subconjuntos de atributos (colunas).
  - Cada fragmento tem colunas exclusivas, com exceção da coluna-chave, que é comum a todos os fragmentos.
- Mista:
  - Combinação de estratégias horizontais e verticais.
  - A tabela pode ser dividida em vários subconjuntos horizontais (linhas), cada um tendo um subconjunto dos atributos (colunas).

Já a replicação refere-se ao armazenamento de cópias de dados em vários locais. Traz como benefícios a maior disponibilidade de dados, melhor distribuição de carga, maior tolerância a falhas de dados e custos de consulta reduzidos. Além disso, impõe sobrecarga adicional, porque cada cópia de dados deve ser mantida pelo sistema. E, como os dados são replicados em outro site, há custos de armazenamento associados e tempos de transação aumentados (dados atualizados em vários sites simultaneamente).

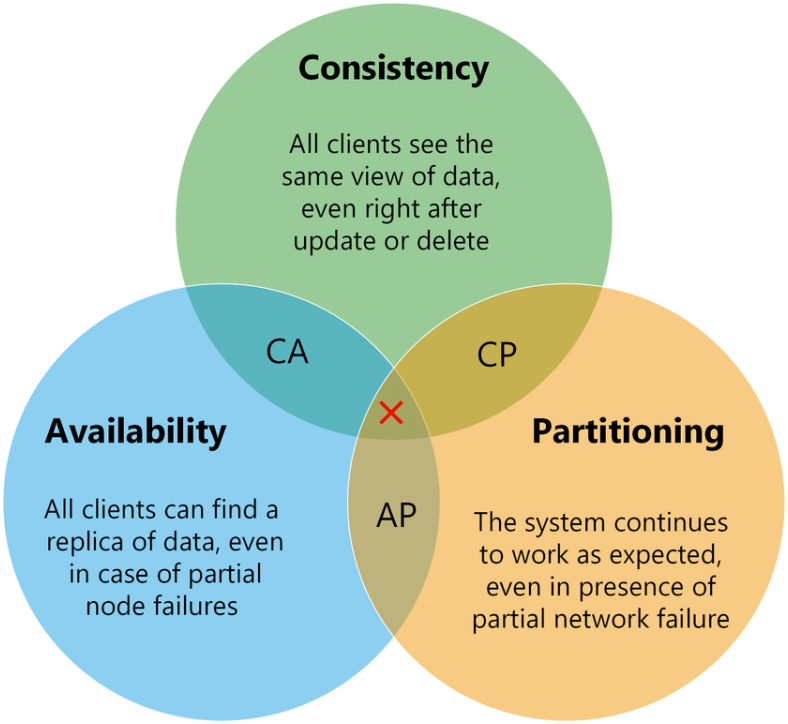
Seguindo para a conclusão do tópico, é preciso, ainda, dizer que, em qualquer sistema de dados altamente distribuído, existem três propriedades comumente desejáveis: consistência (*consistency*), disponibilidade (*availability*) e tolerância à partição (*partitioning tolerance*). A seguir, um breve descritivo sobre cada uma dessas propriedades:

- Consistência: Todos os nós devem ver os mesmos dados ao mesmo tempo, o que significa que as réplicas devem ser atualizadas imediatamente.
- Disponibilidade: Uma solicitação é sempre atendida pelo sistema. Nenhuma solicitação recebida é perdida.

- Tolerância a partição: O sistema continua a operar mesmo em caso de falha de um nó. O sistema falhará apenas se todos os nós falharem.

Em relação a isso, o cientista da computação Eric Brewer afirmou, em 2000, que é impossível para um sistema fornecer todas as três propriedades ao mesmo tempo. O que ficou conhecido como Teorema de Brewer ou Teorema CAP (acrônimo de *consistency*, *availability* e *partitioning tolerance*. A Figura 10 demonstra as implicações decorrentes desse teorema.

Figura 10 – Implicações do Teorema CAP



Fonte: https://www.researchgate.net/figure/Visualization-of-CAP-theorem_fig2_282679529

2.4. Evolução do SGBD

Interessante observar que a evolução dos sistemas gerenciadores de banco de dados é gradativa e ocorre aproveitando tecnologias anteriores que são aprimoradas pelos fornecedores originais ou por concorrentes. Com isso, os vários sistemas existentes podem, de certa forma, ser colocados em uma linha evolutiva que aponta a linhagem daquele produto.

Além disso, a quantidade de produtos relacionados aos desafios de armazenamento de dados é realmente muito grande nos dias de hoje. Há várias referências que buscam apresentar o panorama (*landscape*) de mercado indicando os principais fornecedores de cada uma das categorias dessas aplicações. Uma boa forma de compreender tudo isso é lançando um olhar sobre a evolução cronológica do SGBD.

Na década de 60, surgiu o *Integrated Data Store* (IDS), que foi desenvolvido pela General Eletric (GE), sendo considerado um dos primeiros SGBD comercialmente disponíveis. Teve, em Charles Bachman, seu grande nome e tornou-se base para o *Committee on Data Systems Languages* (CODASYL), que contribuiu para o desenvolvimento do COBOL e Modelo de Dados de Rede.

Na sequência disso e ainda na década de 60, surge o *Information Management System* (IMS). Esse, por sua vez, utilizando um modelo hierárquico desenvolvido pela IBM e construído para apoiar no gerenciamento de componentes de construção da Saturno 5 dentro da missão espacial Apollo.

Então, por volta de 1970, surge um dos maiores nomes da história dos bancos de dados: Ted Codd. Enquanto trabalhava como pesquisador na IBM, ele desenvolveu o modelo relacional que é amplamente utilizado até os dias de hoje. A formação matemática desse inglês, que chegou a ser piloto da *Royal Air Force* (RAF) durante a Segunda Guerra Mundial, certamente foi fundamental para a proposição desse modelo, e da Álgebra Relacional que deu base para a linguagem SQL.

Tendo como base o modelo relacional, surgem os primeiros SGBDs relacionais com algum caráter comercial, conforme o seguinte:

- System R desenvolvido pela IBM.
- Ingres:
    - University of California, Berkley.
    - Origem do Postgres (Post Ingres), posteriormente renomeado para PostgreSQL.
    - Jim Gray e Michael Stonebraker (fundador da Vertica e VoltDB).
- Oracle:
    - Projeto de banco de dados da Ampex para a CIA.
    - IBM demora a perceber o potencial do Modelo Relacional.
    - Larry Ellison.

A década de 80 marca realmente a prevalência dos modelos relacionais e o surgimento, a consolidação e a evolução da linguagem estruturada de consulta (*Structured Query Language* - SQL) como padrão de mercado para acesso a dados. Nesse período, surgem novos concorrentes:

- IBM DB2.
- Informix (IBM, em 2001).
- Sybase (Microsoft, em 1988, e SAP, em 2010).
- Teradata (1 TB de dados no Walmart, em 1992, e adquirido pela AT&T, em 1991).

Ao longo desse período, surgiu, ainda, o SGBD Orientado a Objetos numa tentativa de aproximar o banco de dados da linguagem de programação utilizando o mesmo modelo de representação. Essa abordagem não chegou a ter grande êxito mesmo após a adaptação para um conceito intermediário, chamado de SGBD Objeto Relacional.

Nos anos de 90, surgiram ainda mais concorrentes no mundo relacional:

- PostgreSQL (Postgres com suporte ao SQL).
- MySQL (adquirido pela Sun Microsystems, em 2008, posteriormente adquirida pelo Oracle, em 2009, e deu origem ao MariaDB, em 2010).
- Microsoft SQL Server (originado do Sybase).

Em 2000, houve o surgimento de bancos de dados especializados em cargas OLAP. Tais bancos adotavam uma arquitetura colunar voltada para desempenho de consultas. Alguns exemplos são os seguintes:

- Vertica (fundada por Stonebraker, em 2005, e adquirida pela HP, em 2011).
- Netezza (baseado no PostgreSQL e adquirida pela IBM, em 2010).
- Greenplum (baseado no PostgreSQL e adquirido pela EMC, em 2010, e pela Vmware, em 2020).

Todavia, a grande novidade dos anos 2000 ocorreu com o surgimento do banco NoSQL. Com foco em alta disponibilidade e escalabilidade, esses SGBDs trouxeram algumas importantes mudanças:

- Dados sem estrutura fixa pré-definida.
- Modelo não relacional (documento, chave-valor etc.).
- Sem grandes requisitos transacionais.
- *Application Programming Interface* (API) em lugar de SQL.
- Geralmente código aberto.
- Alguns já nativos para nuvem (*cloud-native*).

Alguns exemplos dessa categoria são: Neo4j, MongoDB, Couchbase, Cassandra, Redis, Azure Cosmos DB, Amazon DynamoDB e Google Cloud BigTable.

Já em 2010, surge o termo NewSQL. Nessa categoria, estão sistemas que buscam, de alguma forma, prover as melhores características de bancos SQL e NoSQL, tais como:

- Alta disponibilidade e escalabilidade.
- Modelo relacional.
- Capacidade transacional.
- Linguagem SQL.
- Geralmente código fechado.
- A maioria nativos para nuvem (*cloud-native*).

Nessa categoria, os maiores exemplos são: Amazon Aurora, Google Cloud Spanner, VoltDB, NuoDB, CockroachDB, YugabyteDB e SingleStore.

Houve, ainda, uma grande evolução em plataformas que vem para suportar as necessidades analíticas das organizações. Em lugar de utilizar um gerenciador de armazenamento próprio, essas tecnologias aproveitam o armazenamento distribuído disponibilizado pelos fornecedores de nuvem e utilizam a virtualização de dados impulsionado por ambientes de *Data Lake*. Como principais exemplos, pode-se citar: Druid, Amazon Redshift, Azure Synapse Analytics, Google BigQuery, Snowflake, Pinot e Presto (recentemente renomeado para Trino).

2.5. Mercado de Banco de Dados

Entender o mercado de banco de dados pode trazer uma melhor compreensão acerca das várias ferramentas e fornecedores que disputam a preferência das empresas ao redor do mundo. Para isso, há organizações que dedicam recursos para o levantamento de indicadores que podem ser relevantes para tal compreensão.

Nesse sentido, DB-Engines é uma iniciativa para coletar e apresentar informações sobre esse tema. Com foco em medir a popularidade das soluções, adota os seguintes critérios e seus levantamentos:

- Consultas nos motores de busca (Google e Bing).
- Frequência de pesquisas no Google Trends.
- Volume de perguntas relacionadas e o número de usuários interessados nos principais fóruns de discussão (Stack Overflow e DBA Stack Exchange etc.).
- Quantidade de ofertas nos principais motores de busca de empregos (Even e Simply Hired).
- Quantidade de perfis em redes profissionais em que o sistema é mencionado (LinkedIn).
- Relevância nas redes sociais (Twitter).

As figuras a seguir apresentam alguns dos principais indicadores apresentados recentemente pela DB-Engines:

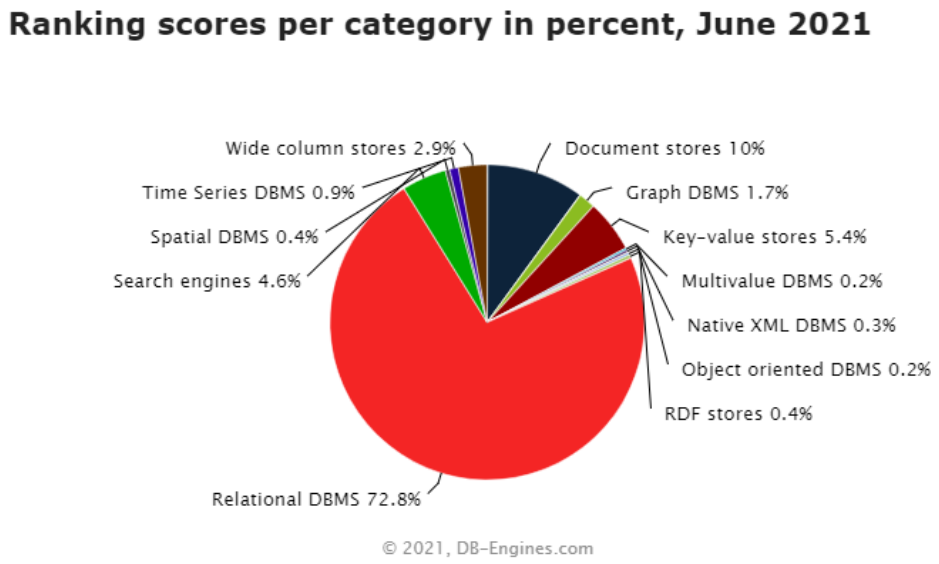
Figura 11 – DB-Engines: Ranking geral de popularidade

371 systems in ranking, June 2021

Rank Jun 2021	Rank May 2021	Rank Jun 2020	DBMS	Database Model	Score Jun 2021	Score May 2021	Score Jun 2020
1.	1.	1.	Oracle	Relational, Multi-model	1270.94	+1.00	-72.65
2.	2.	2.	MySQL	Relational, Multi-model	1227.86	-8.52	-50.03
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	991.07	-1.59	-76.24
4.	4.	4.	PostgreSQL	Relational, Multi-model	568.51	+9.26	+45.53
5.	5.	5.	MongoDB	Document, Multi-model	488.22	+7.20	+51.14
6.	6.	6.	IBM Db2	Relational, Multi-model	167.03	+0.37	+5.23
7.	7.	↑ 8.	Redis	Key-value, Multi-model	165.25	+3.08	+19.61
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	154.71	-0.65	+5.02
9.	9.	9.	SQLite	Relational	130.54	+3.84	+5.72
10.	10.	↑ 11.	Microsoft Access	Relational	114.94	-0.46	-2.24
11.	11.	↓ 10.	Cassandra	Wide column	114.11	+3.18	-4.90
12.	12.	12.	MariaDB	Relational, Multi-model	96.79	+0.10	+7.00
13.	13.	13.	Splunk	Search engine	90.27	-1.84	+2.19
14.	14.	14.	Hive	Relational	79.69	+3.51	+1.04
15.	15.	↑ 23.	Microsoft Azure SQL Database	Relational, Multi-model	74.79	+4.33	+27.01
16.	16.	16.	Amazon DynamoDB	Multi-model	73.76	+3.69	+8.90
17.	17.	↓ 15.	Teradata	Relational, Multi-model	69.34	-0.64	-3.95
18.	↑ 19.	↑ 22.	Neo4j	Graph	55.75	+3.52	+7.48
19.	↓ 18.	19.	SAP HANA	Relational, Multi-model	54.11	+1.36	+3.29
20.	20.	↓ 18.	Solr	Search engine, Multi-model	52.10	+0.91	+0.84

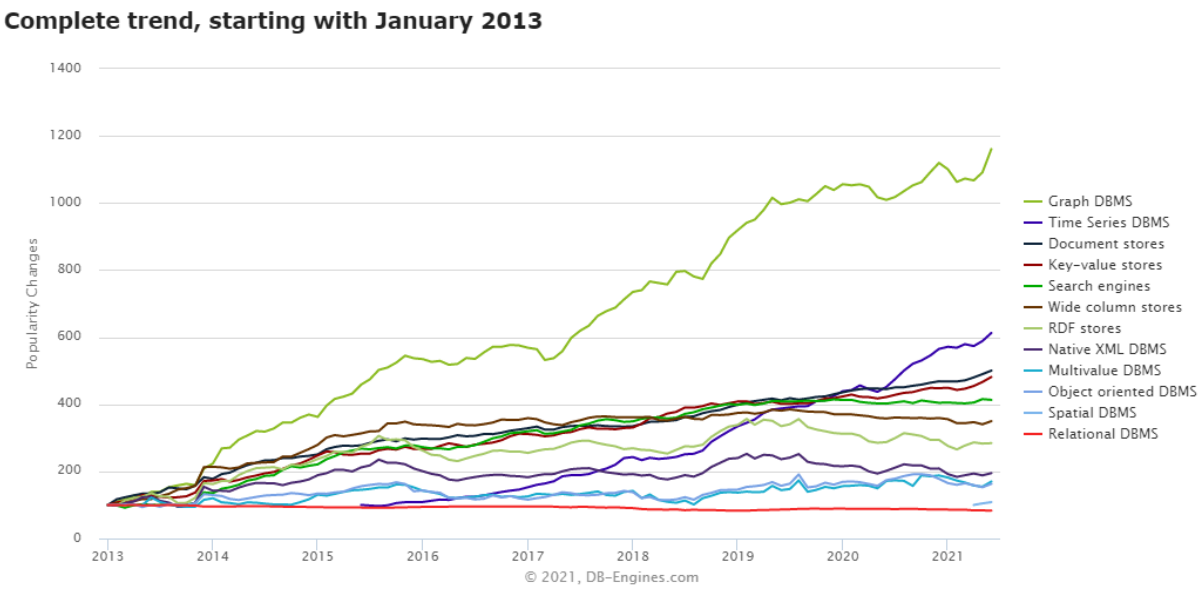
Fonte: https://db-engines.com/en/ranking

Figura 12 – DB_Engines: Ranking percentual por categoria



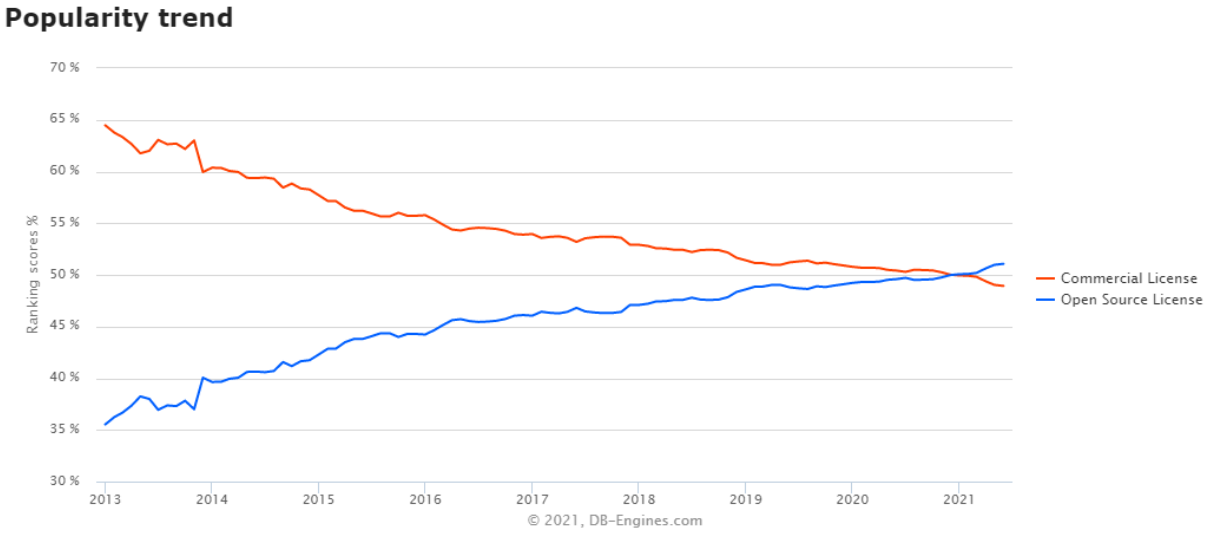
Fonte: https://db-engines.com/en/ranking_categories

Figura 13 – DB-Engines: Tendência por categoria



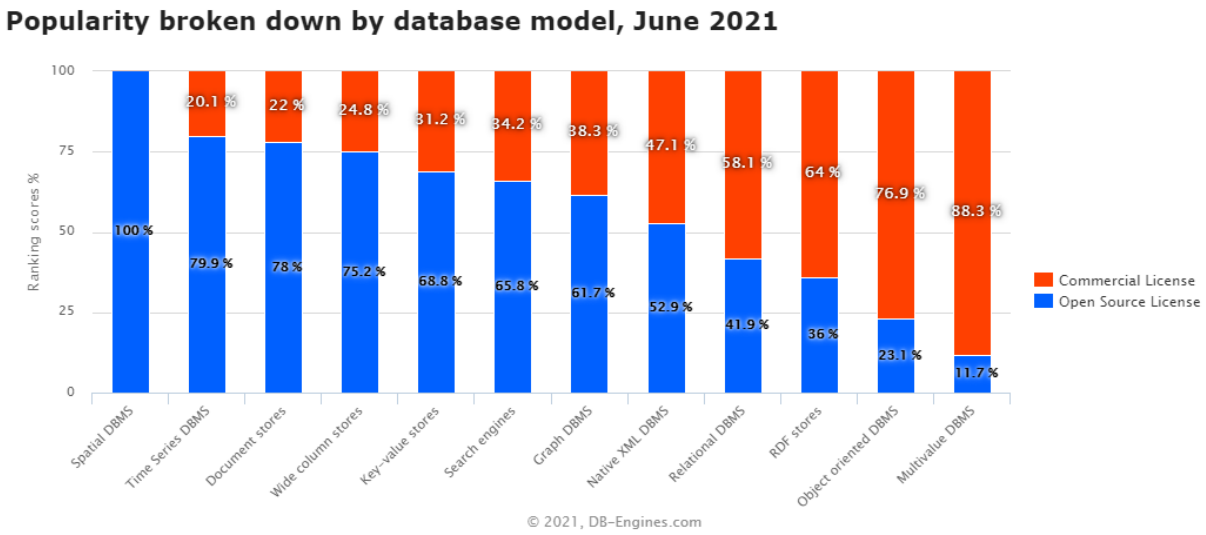
Fonte: https://db-engines.com/en/ranking_categories

Figura 14 – DB-Engines: Popularidade por tipo de licenciamento



Fonte: https://db-engines.com/en/ranking_osvsc

Figura 15 – DB-Engines: Popularidade por tipo de licença e categoria



Fonte: https://db-engines.com/en/ranking_osvsc

Figura 16 – DB-Engines: Top 5 por tipo de licença

The top 5 commercial systems, June 2021

Rank	System	Score	Overall Rank
1.	Oracle	1271	1.
2.	Microsoft SQL Server	991	3.
3.	IBM Db2	167	6.
4.	Microsoft Access	115	10.
5.	Splunk	90	13.

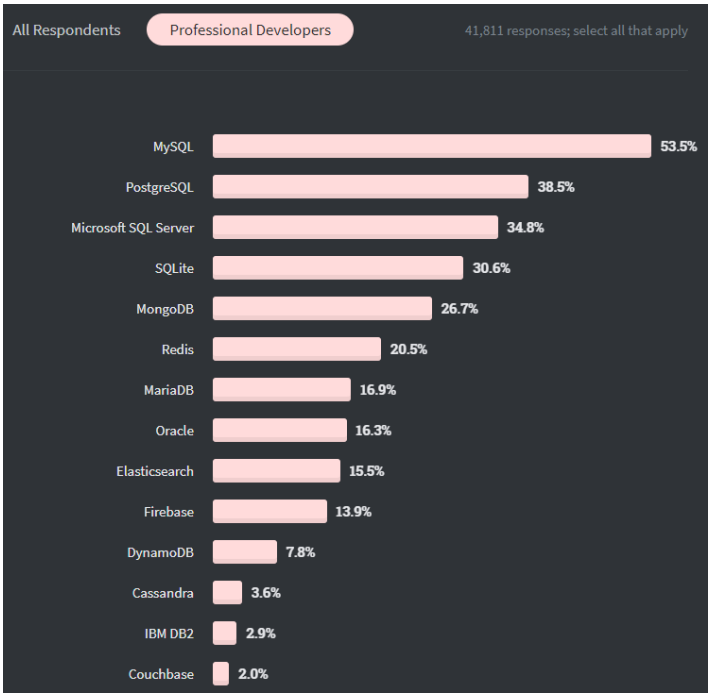
The top 5 open source systems, June 2021

Rank	System	Score	Overall Rank
1.	MySQL	1228	2.
2.	PostgreSQL	569	4.
3.	MongoDB	488	5.
4.	Redis	165	7.
5.	Elasticsearch	155	8.

Fonte: https://db-engines.com/en/ranking_osvsc

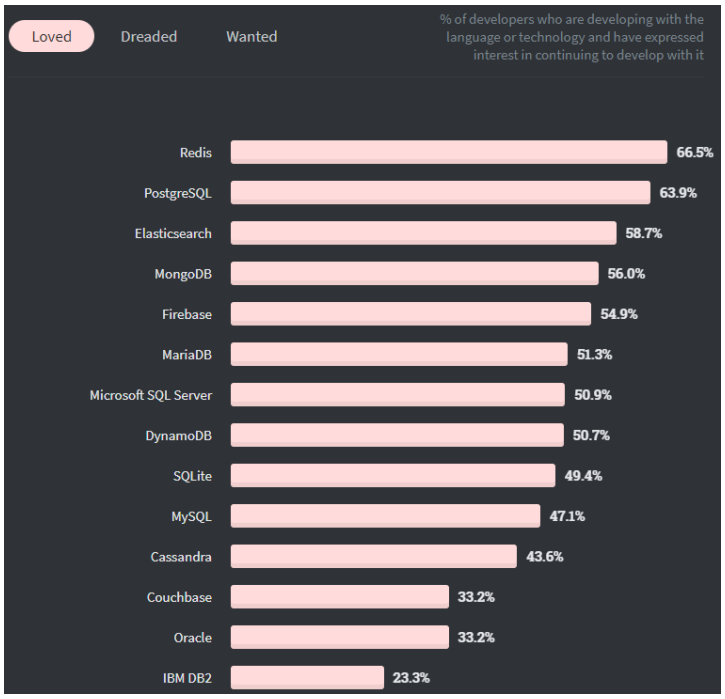
Fundada em 2008, a plataforma pública de Stack Overflow é usada por cerca de 100 milhões de pessoas que buscam aprender, compartilhar seu conhecimento, colaborar e construir suas carreiras na área de TI. Anualmente, realizam uma pesquisa chamada *Developer Survey,* buscando entender aspectos relacionados ao desenvolvimento de *software*. Em 2020, essa pesquisa originou um relatório baseado em uma pesquisa com 65.000 desenvolvedores de software de 186 países. As figuras a seguir mostram alguns dos principais resultados relacionados a banco de dados:

Figura 17 – Stack Overflow: Popularidade junto aos desenvolvedores



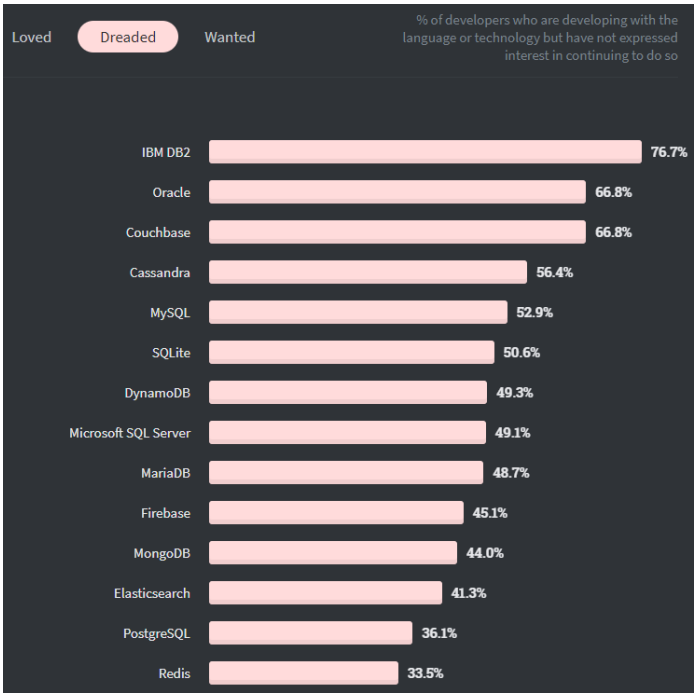
Fonte: https://insights.stackoverflow.com/survey/2020#technology

Figura 16 – Stack Overflow: SGBDs já utilizados e bem avaliados



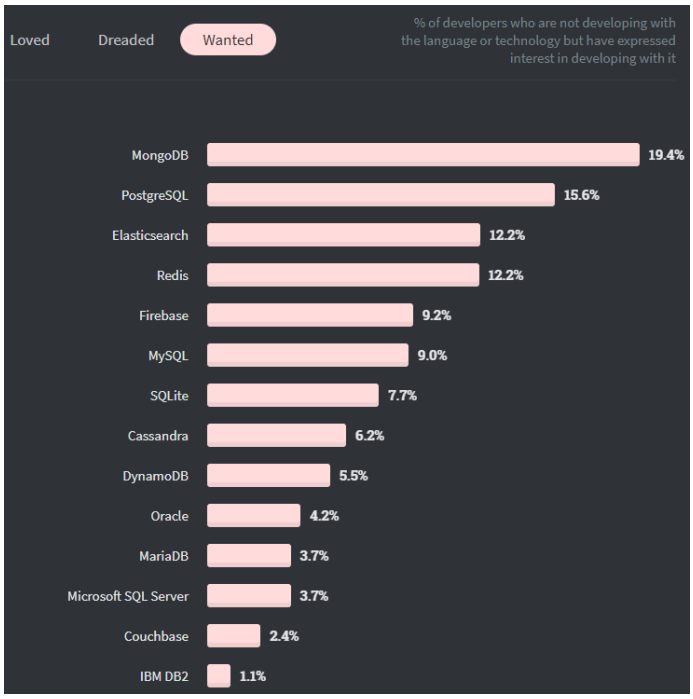
Fonte: https://insights.stackoverflow.com/survey/2020#technology

Figura 17 – Stack Overflow: SGBDs já utilizados e mal avaliados



Fonte: https://insights.stackoverflow.com/survey/2020#technology

Figura 18 - Stack Overflow: SGBDs ainda utilizados, mas desejados

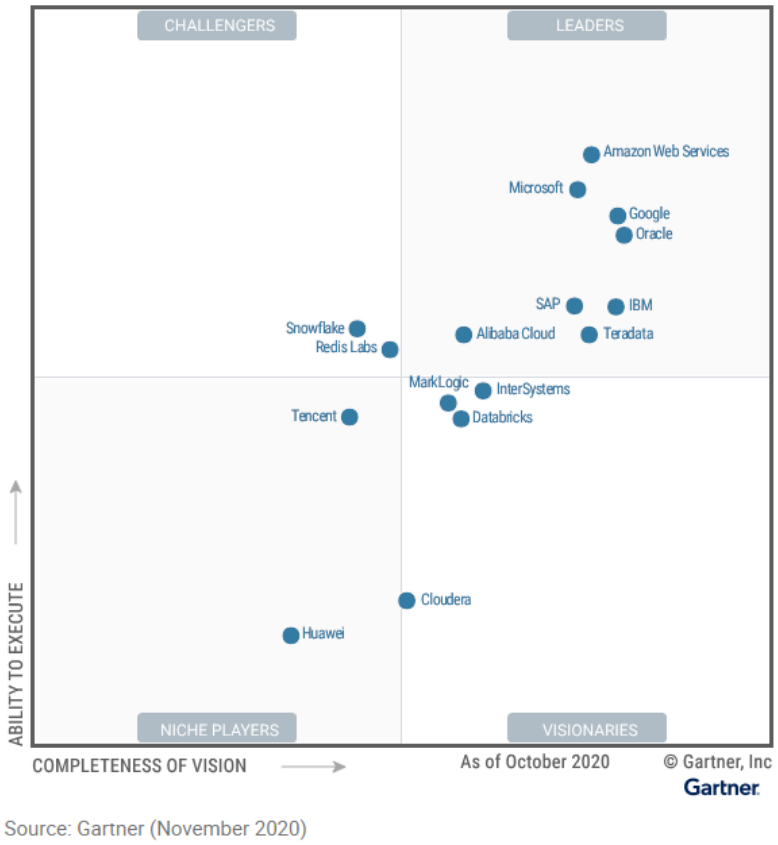


Fonte: https://insights.stackoverflow.com/survey/2020#technology

Por sua vez, a Gartner, conceituada empresa líder em pesquisa e consultoria, acompanha o mercado de banco de dados com diversas análises. Dentre elas, o *Magic Quadrant for Cloud Database Management Systems*, que, em 2020, foi atualizado. Nesse relatório, define o mercado de sistema de gerenciamento de banco de dados em nuvem como sendo aquele fornecido em nuvem pública ou privada totalmente gerenciado e cujos dados residem em uma camada de armazenamento em nuvem. Não inclui fornecedores que possuem apenas SGBDs hospedados em infraestrutura como serviço (IaaS), como em uma máquina virtual ou contêiner, e gerenciados pelo cliente. A Figura 19 apresenta o quadrante mágico resultante desse último levantamento.

Desse relatório, é possível destacar a classificação dos três principais fornecedores de computação em nuvem:

Figura 19 – Gartner: Quadrante mágico de SGBD em nuvem



Source: Gartner (November 2020)

Fonte: https://www.gartner.com/doc/reprints?id=1-24NM3UP3&ct=201125&st=sb

- AWS:
  - A Amazon Web Services (AWS) é o maior provedor de serviços de nuvem do mundo em receita, com presença internacional e uma base de clientes global em todos os principais setores.
  - A AWS oferece uma variedade de serviços de gerenciamento de banco de dados.
    - Operacionais: Amazon Relational Database Service (RDS), Amazon Aurora e Amazon DynamoDB.
    - Analíticos: Amazon Redshift, Amazon Athena e Amazon EMR.
    - Especializados: Amazon Neptune (grafos), Amazon DocumentDB (documentos), Amazon Timestream (séries temporais), Amazon Keyspaces (Apache Cassandra) e Amazon Quantum Ledger Database (blockchain).

- Microsoft Azure:
  - As operações da Microsoft são diversificadas geograficamente e seus clientes cobrem uma ampla gama de setores e tamanhos de implantação.
  - A Microsoft está focada em fornecer um ecossistema de gerenciamento de dados em nuvem coeso que abrange todos os casos de uso avaliados.
  - Fornece uma ampla variedade de ofertas de SGBD em nuvem, incluindo:
    - Azure Synapse Analytics.
    - Azure SQL Database.
    - Azure SQL Managed Instance.
    - Azure Cosmos DB.
    - Azure HDInsight.
    - Azure Database for PostgreSQL, MySQL and MariaDB.
- Google Cloud Platform (GCP):
  - As operações do Google são diversificadas geograficamente. Tendo clientes de todos os tamanhos em todos os setores.
  - O Google possui uma ampla variedade de produtos relacionais e não relacionais gerenciados.
  - A Google Cloud Platform (GCP) oferece suporte a muitos produtos de plataforma de banco de dados como serviço (dbPaaS), desde versões totalmente gerenciadas de produtos de fornecedores terceirizados até seus próprios produtos:
    - Google Cloud SQL (PostgreSQL, MySQL e SQL Server).
    - Cloud Spanner.
    - Cloud Bigtable.
    - BigQuery.
    - Dataproc.
    - Cloud Firestore.
    - Firebase Realtime Database.

Capítulo 3. Banco de Dados Relacional

Os bancos de dados relacionais, baseados na teoria relacional de Ted Codd idealizada na década de 70, são os principais mecanismos de armazenamento de dados utilizados ao longo da história. E, mesmo atualmente, permanecem firmes no atendimento das demandas por sistemas transacionais dentro das organizações. Por isso mesmo, dedica-se este capítulo ao entendimento dos conceitos de bases relacionais, bem como na linguagem SQL, estrutura arquitetural do SGBDR e no mercado consumidor dessa tecnologia.

3.1. Conceitos de Banco de Dados Relacional

Conforme já mencionado, o modelo de dados relacional foi introduzido por Codd, em 1970, e baseia-se no uso de relação, atributos e tuplas. Com o avanço dos SGBDs comerciais, a nomenclatura original acabou perdendo força, prevalecendo o uso dos termos tabela, colunas e linhas, respectivamente.

A tabela, unidade fundamental de armazenamento de dados no SGBDR, nada mais é que o conjunto de linhas e colunas. Sendo que cada linha representa um registro, cada coluna representa um tipo de dado e a intersecção entre linha e coluna determina um campo. Todos os valores de uma mesma coluna são do mesmo tipo de dados, e, em uma mesma tabela, as colunas terão que possuir nomes distintos.

Por sua vez, o relacionamento é a associação entre duas ou mais tabelas distintas por meio do uso de chave primária e chave estrangeira. A chave primária (PK) identifica unicamente uma linha em uma tabela. Já a chave estrangeira (FK) é o campo que se relaciona com a chave primária de outra tabela.

Há, ainda, alguns outros objetos bastante úteis no SGBDR, sendo os principais:

- View:
  - Query armazenada como um objeto no banco de dados.

  - Subconjunto de colunas de uma ou mais tabelas.
- Procedimento Armazenado (*Stored Procedure*):
  - Coleção de comandos SQL armazenados no banco de dados com um determinado nome.
  - Passível de controle de segurança.
  - Pode ou não exigir parâmetros de entrada/saída.
- Gatilho (*Trigger*):
  - Procedures disparadas automaticamente.
  - Disparada em casos de UPDATE, DELETE ou INSERT.
  - Utilizada quando a FK não é suficiente para prover a integridade (em casos envolvendo manipulação de dados) ou quando deseja-se manipular regras de negócios.
- Índice:
  - Estruturas auxiliares de acesso a dados.
  - Proveem caminhos secundários para atingir-se determinada linha.
  - Utilizado para agilizar pesquisas baseadas em certas condições de pesquisa.

Há, ainda, no modelo relacional, o conceito de restrições. Vários tipos de restrições podem ser especificados em um esquema de banco de dados relacional, tais como:

- Restrições de domínio:
  - Obedecer a definição dos tipos de dados.
  - Aceitação ou não de valores nulos.
- Restrições de chave:
  - Duas linhas não podem ter a mesma combinação de valores para todos os seus atributos.
  - A chave primária define o conjunto de colunas que identificam unicamente as linhas da tabela.
  - A tabela pode ter no máximo uma *primary key*.
- Restrições de integridade referencial:

- Especificada entre duas tabelas e utilizada para manter a consistência entre elas.
- Utiliza-se o conceito de chave estrangeira (*foreign key*) para definir as restrições de integridade referencial.
- Garantem que o conteúdo de um banco de dados esteja sempre íntegro e coerente.
- A integridade referencial estabelece que todo valor de chave estrangeira numa tabela deve corresponder a um valor de chave primária de uma segunda tabela ou deve ser nulo.
- A FK é definida na tabela filha e a tabela contendo a coluna referenciada é a tabela pai.
- Deve incluir sempre colunas cujos tipos correspondem exatamente aos tipos das colunas da restrição de chave primária ou de unicidade referenciada.

3.2. Linguagem, Estrutura e Mercado Relacional

Como base para descrever os mecanismos de manipulação dos dados em um modelo relacional, foi utilizada a álgebra relacional. Ela define as operações fundamentais para buscar e manipular tuplas (linhas) em uma relação (tabela). Nela, cada operador assume uma ou mais relações (tabelas) como entrada e produz uma nova relação (tabela) como saída. Além disso, conforme destacado na Figura 20, operadores podem ser encadeados para gerar operações mais complexas.

Figura 20 – Operadores da álgebra relacional e exemplo de uso

Fonte: Andy Pavlo (2020)

A partir daí, surge a linguagem SQL. Originalmente chamada SEQUEL, foi desenvolvida inicialmente pela IBM, em 1974, e adotada pela Oracle, em 1979. Posteriormente, padronizada pela *American National Standars Institute* (ANSI), em 1986, e pela *Internation Organization for Starndardization* (ISO), em 1987.

Apesar dessa padronização, cada SGBD apresenta sua própria linguagem que possui variações. Como exemplos, é possível citar a Transact SQL do SQL Server e a PL/SQL do Oracle.

Independentemente dessas variações, na linguagem SQL, o usuário concentra-se na resposta que deseja e não como chegar até ela, pois é o SGBD quem fica responsável pela avaliação da consulta por meio de seu otimizador que busca por estratégias ótimas de execução.

Para melhor organização e entendimento, a linguagem SQL pode ser dividida em subcategorias, sendo relacionadas, abaixo, as principais delas com suas atribuições:

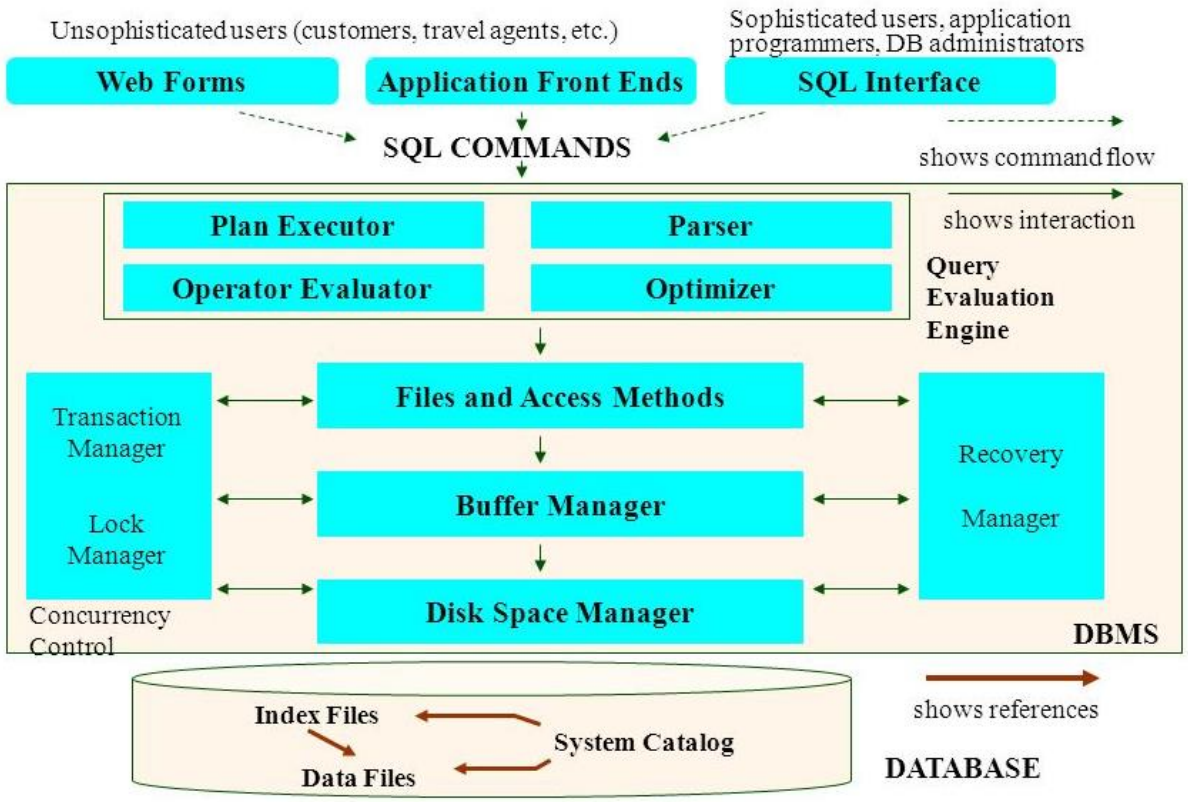
- DCL - Data Control Language (Controle): manipular permissões de usuários:
  - Grant – Permite acesso.
  - Deny – Nega acesso.
- DDL - Data Definition Language (Definição): manipular estruturas de dados:

- Ex.: create table.

- DML - Data Manipulation Language (Manipulação): manipular dados:
  - Busca (select).
  - Inserção (insert).
  - Remoção/Eliminação (delete).
  - Alteração/Modificação (update).

Em termos arquiteturais, mesmo sabendo que cada fornecedor tem a liberdade para implementar seus próprios mecanismos internos, é possível dizer que a estrutura de um SGBD segue o que está apresentado na Figura 21.

Figura 21 – Estrutura de um SGBD relacional



Fonte: RAMAKRISHNAN; GEHRKE, 2003

Nesse diagrama, é possível observar um conjunto de aplicações que precisa fazer uso de sistema gerenciador de banco de dados, sendo isso realizado por meio de comandos na linguagem SQL. A partir do momento que isso chega para o SGBDR,

inicia-se uma série de operações necessárias para dar suporte ao atendimento das requisições.

Os dados fisicamente alocados no sistema de arquivos são controlados diretamente pelo sistema operacional. Então, para fazer a interface com o SO, existe, no SGBD, um componente que vai ficar o tempo inteiro conversando com esse SO, entregando e recebendo blocos de dados. Na figura é representado pelo Gerenciador de Espaço em Disco (*Disk Space Manager*).

Contudo, toda a manipulação de dados realizada pelo SGBD deve ocorrer em memória. Portanto, tudo que é retornado pelo *Disk Space Manager* precisa ser alocado e gerenciado em memória para ser trabalhado. Para isso há o gerenciador de memória (*buffer manager*) que fica responsável por esse trâmite de receber e entregar blocos de dados ao componente de disco, gerenciando a alocação em memória. Importante destacar que esse gerenciamento passa também por invalidar áreas de memória e entregar, ao componente de disco, os objetos alterados e que precisam ser persistidos.

Há, ainda, os componentes responsáveis pelo controle de concorrência. Um deles é o módulo que vai implementar fisicamente o processo de bloqueios (*Lock Manager*) que, em conjunto com o Gerenciador de Transações (*Transaction Manager*), permite o controle transacional do que está sendo executado pelos diversos usuários simultaneamente.

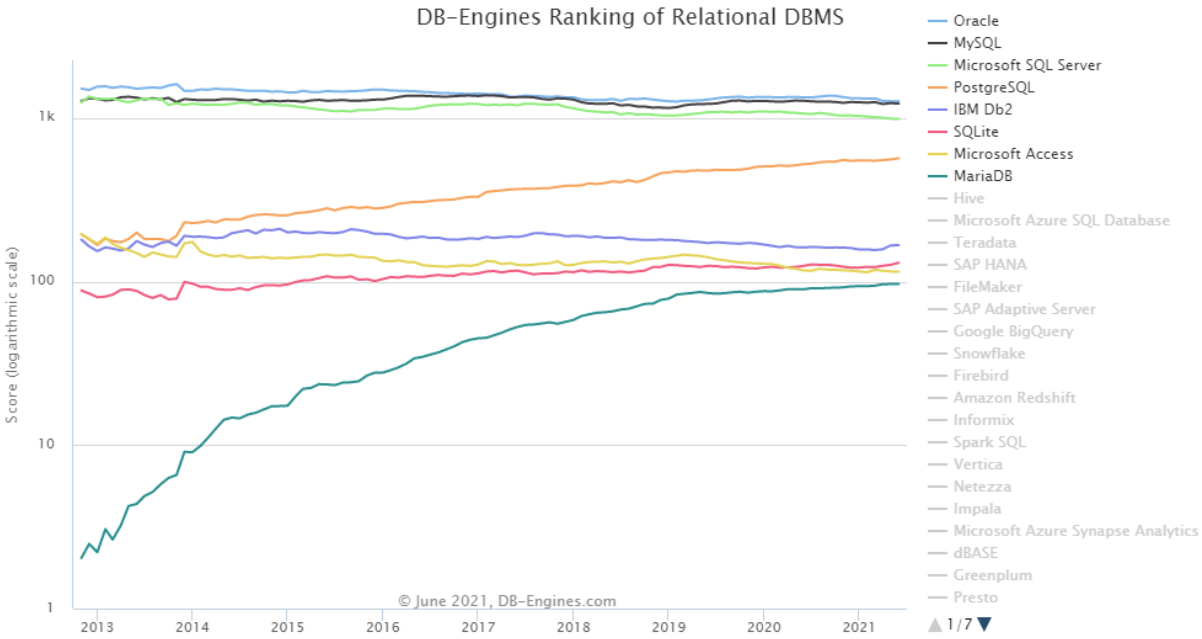
Existe, também, um componente que vai fazer toda a parte de gerenciamento de recuperação (*Recovery Manager*) em caso de falha. Isso inclui não somente recuperação em si, mas também os mecanismos necessários para possibilitar a realização de cópias de segurança (*backup*) do banco de dados. É essa estrutura que, no caso de uma falha, será a responsável por tentar fazer com que o banco volte à situação coerente original existente antes da ocorrência do problema.

Finalmente, temos as estruturas responsáveis por receber e executar os comandos SQL que chegam ao banco, realizando, para tanto, toda a parte de avaliação da consulta. Para isso, há o *Parser*, que fica responsável pela validação da

sintaxe do comando submetido, e o otimizador (Optimizer) que, baseado em diversas estatísticas e métricas do banco de dados, busca o caminho ótimo para atendimento da requisição.

No que diz respeito ao mercado de sistemas gerenciadores de banco de dados relacional, a DB-Engines coloca Oracle, MySQL, SQL Server, PostrgreSQL e DB2 como as cinco plataformas mais populares. A Figura 21 apresenta mais detalhes a esse respeito:

Figura 22 – DB-Engines: SGBDR com maior popularidade



Fonte: https://db-engines.com/en/ranking_trend/relational+dbms

3.3. Demonstração de SGBDR

Além de toda a abordagem teórica, a disciplina traz demonstrações de sistemas gerenciadores de banco de dados relacionais, disponibilizadas em formato de vídeo para melhor compreensão e acompanhamento das práticas propostas.

Capítulo 4. Banco de Dados NoSQL e NewSQL

Bancos de Dados NoSQL trouxeram uma significativa contribuição na medida em que acrescentaram novos e bons modelos de dados que podem ser utilizados em demandas de armazenamento de dados. Portanto, faz-se necessário entender os conceitos de bancos NoSQL bem como detalhar os modelos chave-valor, documento, família de colunas e grafo. Além disso, importante trazer clareza sobre o termo NewSQL que surgiu mais recentemente e que ainda causa certa confusão no mercado de sistemas gerenciadores de banco de dados.

4.1. Conceitos de Banco de Dados NoSQL

O termo NoSQL está relacionado a uma ampla gama de tecnologias de banco de dados não relacionais que foram desenvolvidas para lidar com os desafios representados pelo *Big Data*. Inicialmente, esse nome foi utilizado como uma *hashtag* do Twitter para simplificar a comunicação entre desenvolvedores que discutiam ideias sobre as tecnologias de banco de dados não relacionais.

O nome é questionável, porque não descreve o que é ao mesmo tempo em que deixa de mencionar o que não é um Banco NoSQL. Apesar de trazer a percepção de negação da linguagem SQL, alguns defendem que o termo não significou que os produtos dessa categoria nunca deveriam incluir suporte para SQL. Mais recentemente, alguns observadores da indústria tentaram sugerir que NoSQL poderia significar “Não apenas SQL”. Sobretudo quando alguns produtos começaram a oferecer suporte a linguagens de consulta que imitam o SQL de maneiras importantes.

Isso também não ajuda muito, já que muitos bancos relacionais também suportam outras linguagens. Então, na prática, em lugar de fazerem frente à linguagem SQL, essa nova classe de SGBD vem contrapor os tradicionais bancos de dados relacionais que dominavam sozinhos o mercado até então. Dessa forma, fica a pergunta: será que o mais correto teria sido chamá-los de NoRel?

Independentemente da nomenclatura, vale destacar que o desenvolvimento dessa nova categoria de banco de dados surgiu da iniciativa de organizações, como Google, Amazon e Facebook, que buscavam alternativas para lidarem com os problemas que eles estavam encontrando quando seus conjuntos de dados alcançaram enormes tamanhos. Atualmente, existem dezenas de produtos que podem ser considerados NoSQL.

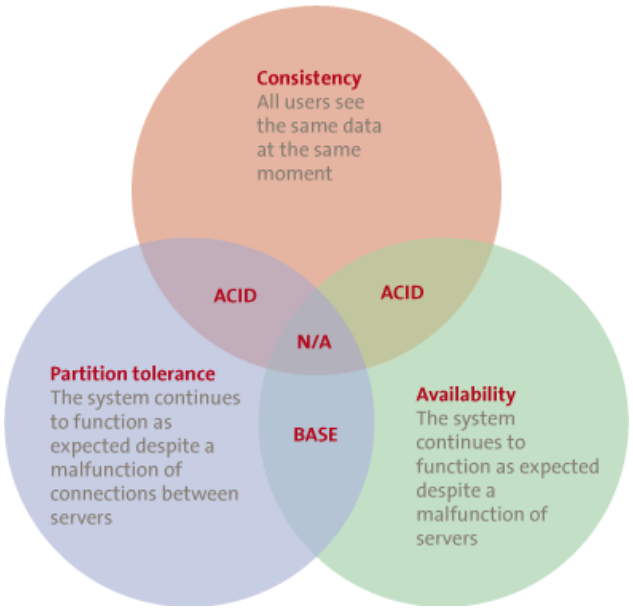
A maioria dos bancos de dados NoSQL foi produzida como software de código aberto e, por isso mesmo, associados ao sistema operacional Linux. Até porque, ao se pensar em um *cluster* contendo dezenas de nós, as questões de licenciamento Windows, por exemplo, podem ser impeditivas, dando ainda mais força à adoção de Linux que pode ser gratuito e altamente personalizável. Portanto, a maioria dos produtos NoSQL é executada apenas em um ambiente Linux ou Unix.

Há diversas características típicas de bancos NoSQL. Dentre elas, destacam-se:

- Simplificação dos requisitos do SGBD:
  - Função mais restrita de armazenamento de dados.
  - Camada de aplicação deve assumir a complexidade do sistema.
  - Estrutura de dados flexível sem necessidade de ser previamente construída ou de atender requisitos complexos de restrições.
- Capacidade de manipulação de grandes volumes de dados. Contudo, um SGBDR também é capaz de suportar isso. A aplicabilidade aumenta quando, além de volume, há velocidade e variedade de dados.
- Custo reduzido quando comparado a alguns bancos relacionais de mercado.
- Comunicação muitas vezes realizada por meio de *Application Programming Interface* (API) para simplificar a recuperação e a manipulação de dados.
- Pelos requisitos a serem atendidos, costumam trazer distribuição nativa de dados: fragmentação (*sharding*) e replicação.
- Foco em escalabilidade horizontal. Sobretudo aproveitando as características de elasticidade providas pela Computação em Nuvem (*Cloud Computing*).

Considerando o Teorema CAP já apresentado anteriormente, que coloca a impossibilidade de atendimento simultâneo de mais de dois dos requisitos, a saber consistência, disponibilidade e tolerância à partição, fica fácil perceber que essa classe de SGBD renuncia a consistência em prol das duas outras características. Isso fica mais bem demonstrado na Figura 23:

Figura 23 – Requisitos de distribuição em relação às categorias de SGBD



Fonte: https://www.compact.nl/wordpress/wp-content/uploads/2016/02/C-2013-0-Keur-02.png

Ainda em relação à Figura 23, é possível perceber que, para atendimento de sistemas AP (*Availability e Partition Tolerance*), deixa-se de lado as propriedades ACID e surge algo novo, chamado BASE. Esse novo termo é, também, um acrônimo que tenta contrapor, inclusive na palavra final formada, aquilo que vinha sendo utilizado até o momento. Perceba que ACID (ácido) e BASE (base) tem significados químicos antagônicos e bastante conhecidos. Esses acrônimos e seus significados podem ser melhor entendidos ao se observar a Figura 24:

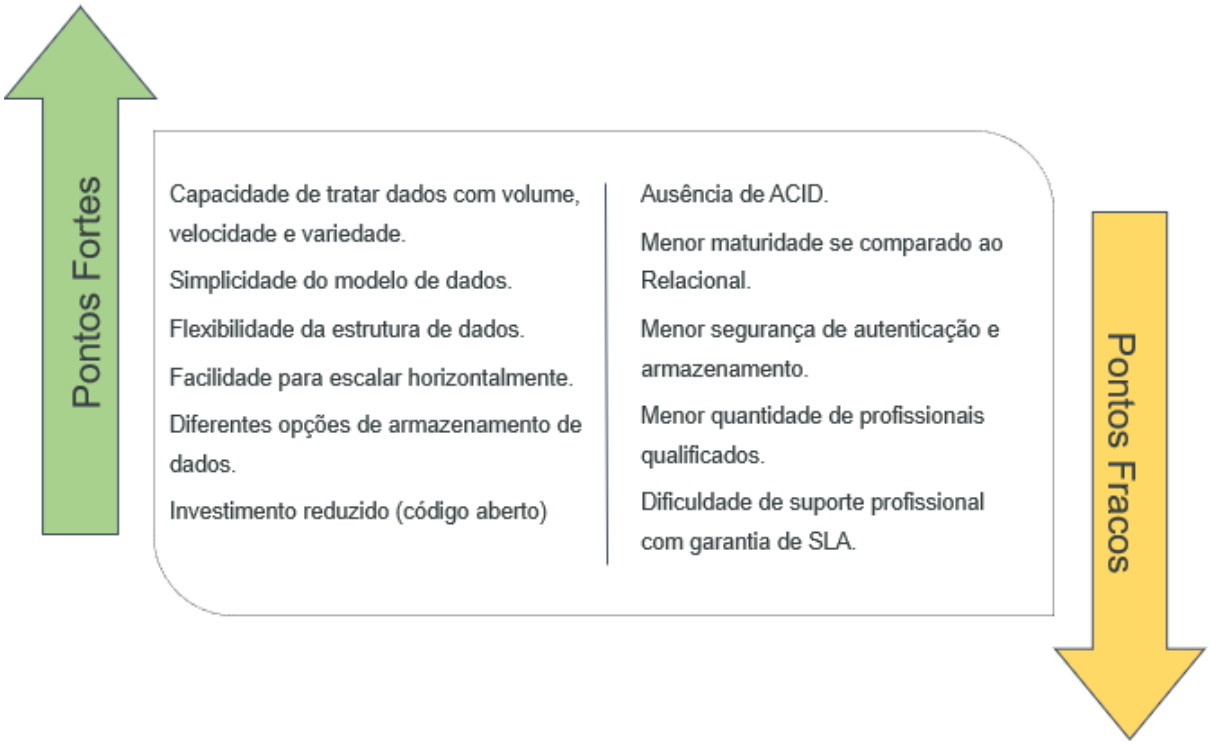
Figura 24 – ACID x BASE

Fonte: Dados da pesquisa

Como seria de se esperar, essa nova classe de banco de dados traz consigo pontos fortes, mas também pontos fracos. A Figura 25 consolida esses pontos para uma melhor compreensão.

Diferentemente dos bancos relacionais que são todos equivalentes em termo do modelo de dados adotado, em bancos NoSQL há diferentes categorias de produtos sendo cada uma delas apropriada para o atendimento de certas demandas específicas. Então, bancos NoSQL podem ser do tipo chave-valor, documento, família de colunas e grafo. Esses tipos serão detalhados nas sessões seguintes.

Figura 25 – Pontos fortes e fracos do NoSQL



Fonte: Dados da pesquisa

4.2. NoSQL Tipo Chave-Valor

Os bancos de dados chave-valor (*key-value database*) são, conceitualmente, os mais simples dos modelos de dados NoSQL. Eles armazenam dados como uma coleção de pares de chave-valor, sendo que a chave atua como um identificador para o respectivo valor.

O valor armazenado pode ser qualquer coisa, como texto, um documento XML ou uma imagem, e o banco de dados não tenta entender o conteúdo ou o significado desse componente. Ao invés disso, simplesmente armazena qualquer valor fornecido para a chave.

Sem suporte a relacionamentos, também não há o conceito de chaves estrangeiras. Isso simplifica muito o trabalho que o SGBD deve executar, tornando os

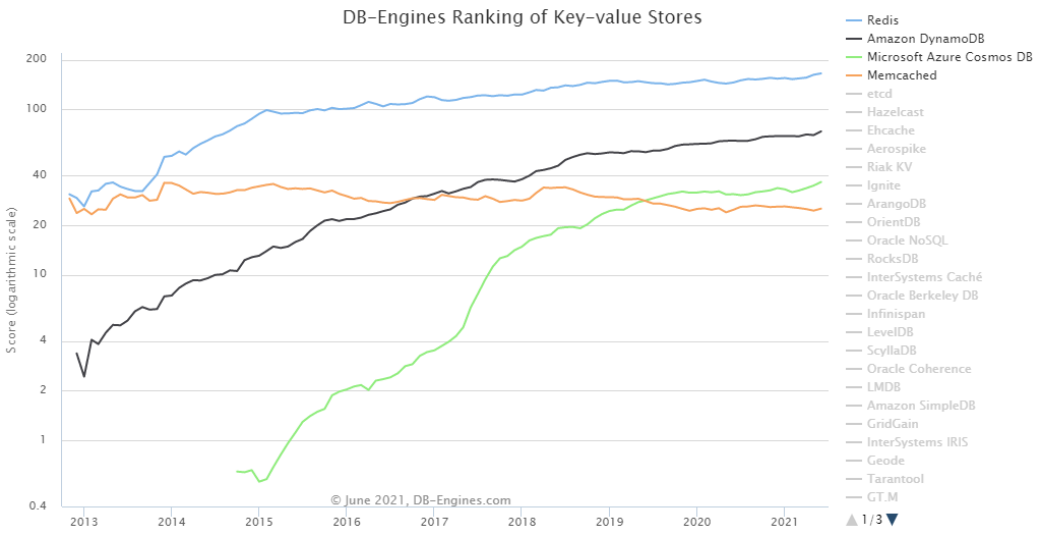
bancos de dados chave-valor extremamente rápidos e escalonáveis para processamento básico.

Além disso, os pares de chave-valor são normalmente organizados em *buckets* que seria o equivalente a uma tabela no relacional. Contudo, trata-se de uma estrutura de armazenamento físico bastante distinta. Importante ainda destacar que:

- Os valores da chave devem ser exclusivos em um *bucket.*
- Todas as operações de dados são realizadas especificando-se o *bucket* e a chave.
- Não há filtro com base no componente valor.
- As operações em bancos de dados chave-valor são bastante simples:
  - Get/Fetch: recuperar o componente valor do par.
  - Store: atribuir o componente valor a uma chave existente ou criando um par caso a chave não exista.
  - Delete: remover o par existente.

A Figura 26 traz o ranking de popularidade de bases NoSQL do tipo Chave-Valor apresentado pela DB-Engines:

Figura 26 – DB-Engines: popularidade de NoSQL Chave-Valor



Fonte: https://db-engines.com/en/ranking_trend/key-value+store

4.3. NoSQL Tipo Documento

Para um melhor entendimento de bancos NoSQL do tipo Documento, é preciso, antes, compreender a Notação de Objetos JavaScript (*JavaScript Object Notation* - JSON).

O JSON é um formato de texto legível para intercâmbio de dados que define atributos e valores em um documento. Ele é mais compacto do que XML, o que torna mais rápida a análise do documento. Apresenta três tipos básicos de dados: numérico, lógico (*boolean*) e caracteres (*string*). E, utilizando-se esses tipos básicos, é possível construir arranjos (*arrays*) e matrizes. A Figura 27 mostra o fragmento de um arquivo JSON bastante simples.

Figura 27 – Exemplo de arquivo JSON

```
[
   {
     “titulo”: “JSON x XML”,
     “resumo”: “o duelo de dois modelos de representação de informações”,
     “ano”: 2012,
     “genero”: [“aventura”, “ação”, “ficção”]
    },
   {
     “titulo”: “JSON James”,
     “resumo”: “a história de uma lenda do velho oeste”,
     “ano”: 2012,
     “genero”: [“western”]
    }
]
```

Fonte: https://www.devmedia.com.br/json-tutorial/25275

Os bancos de dados documento (*document database*) são conceitualmente semelhantes aos bancos de dados Chave-Valor. Contudo, armazenam dados no formato de documentos com marcadores (tags) em pares de chave-valor e esses documentos podem estar em qualquer formato codificado, tais como XML e JSON.

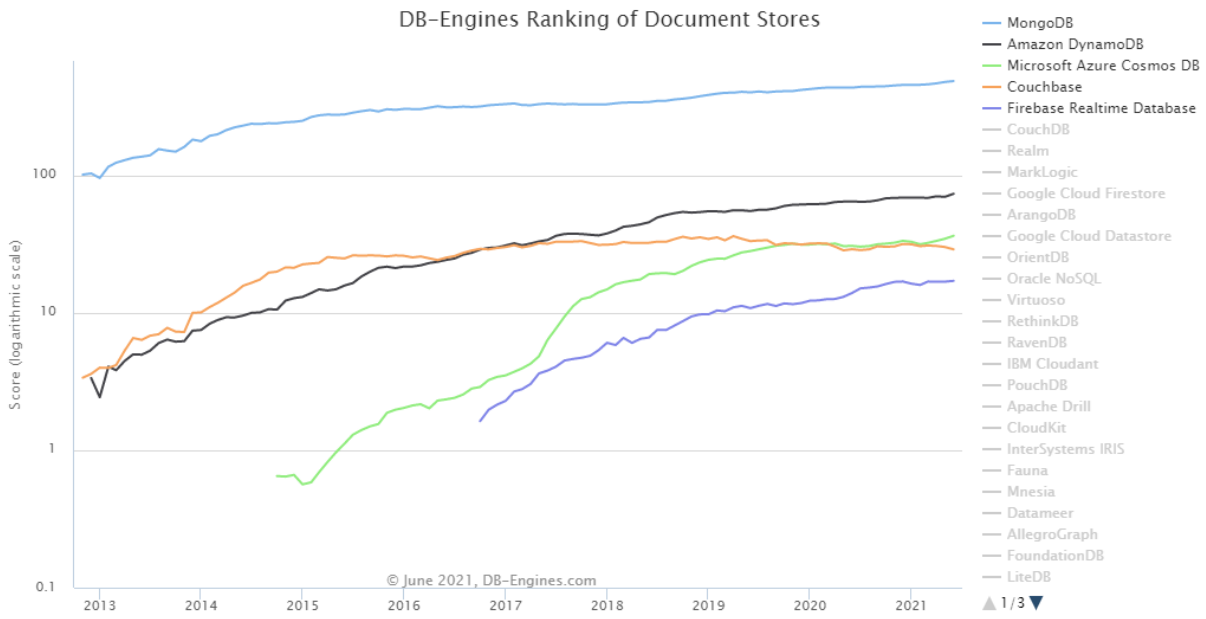
O banco de dados documento pode compreender o conteúdo armazenado no componente de valor. As *tags* dentro do documento são acessíveis ao SGBD, o que torna possível uma consulta mais elaborada. Inclusive, bancos do tipo documento suportam algumas funções agregadas, como somar ou calcular a média.

Apesar do uso de *tags* nos documentos, os bancos de dados de documentos são considerados sem esquema, ou seja, não impõem uma estrutura predefinida aos dados armazenados. Além do mais, cada documento pode ter sua própria estrutura e são agrupados em grupos chamados coleções (*collections*) e que seriam o equivalente às tabelas do relacional.

Contudo, diferentemente do relacional, nesse tipo de banco, os dados tendem a ser agrupados na mesma coleção. Por exemplo, cada documento de pedido em uma coleção de Pedidos conteria dados sobre o cliente, o próprio pedido e os produtos adquiridos naquele pedido, tudo como um único documento independente. Deve-se destacar, ainda, que os bancos de dados de documentos não armazenam relacionamentos conforme percebidos no modelo relacional e geralmente não têm suporte para operações de junção.

Em termos de popularidade, os bancos NoSQL do tipo Documento são classificados da seguinte forma pela DB-Engines:

Figura 28 – DB-Engines: Popularidade de NoSQL do tipo Documento



Fonte: https://db-engines.com/en/ranking_trend/document+store

4.4. NoSQL Tipo Orientado a Colunas

O termo Banco de Dados Orientado a Colunas (*column-oriented database*), também chamado colunar (*columnar*), pode referir-se a dois conjuntos diferentes de tecnologias que costumam ser confundidos.

Em certo sentido, o banco de dados colunar pode indicar tecnologias tradicionais de banco de dados relacional que usam armazenamento centrado em coluna em vez de armazenamento centralizado em linha.

Bancos de dados relacionais apresentam dados em tabelas lógicas. Entretanto, eles são realmente armazenados em blocos de dados contendo linhas de dados. Todos os de uma determinada linha são armazenados juntos em sequência com muitas linhas no mesmo bloco de dados. Se uma tabela tiver muitas linhas de dados, as linhas serão distribuídas por muitos blocos de dados.

O armazenamento centrado em linha (orientado por linha) minimiza o número de leituras de disco necessárias para recuperar uma linha de dados, já que a recuperação de uma linha requer o acesso a apenas um bloco de dados. Isso funciona muito bem em sistemas transacionais, onde a normalização é usada para decompor dados complexos em tabelas relacionadas, de forma a reduzir a redundância e a melhorar a velocidade de manipulação rápida de pequenos conjuntos de dados.

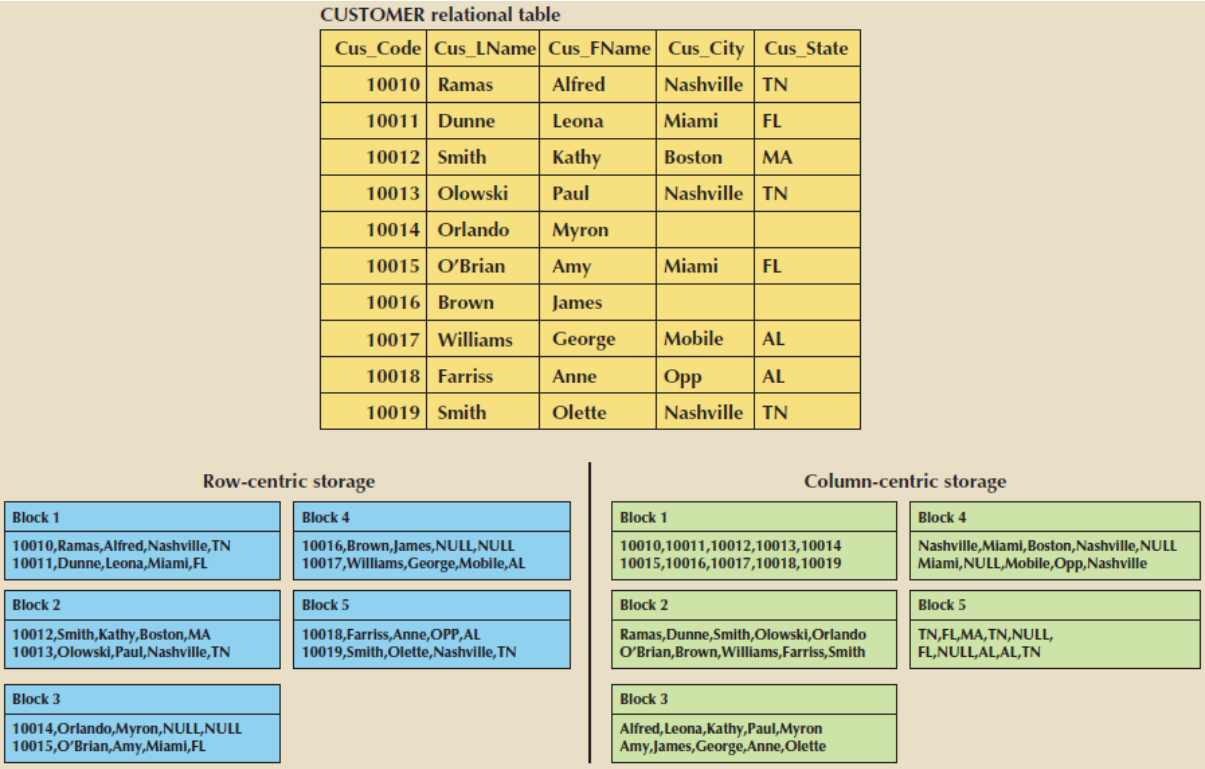
No entanto, em consultas que recuperam um pequeno conjunto de colunas em um grande conjunto de linhas, muitos acessos ao disco serão necessários. Por exemplo, uma consulta que deseja recuperar apenas a cidade e o estado de cada cliente necessitará acessar cada bloco de dados que contém uma linha de cliente para recuperá-los.

Um banco de dados orientado à coluna ou colunar armazena os dados em blocos por coluna em vez de por linha. Os dados de um único cliente serão espalhados por vários blocos, mas todos os dados de uma única coluna estarão em apenas alguns blocos. Esse tipo de armazenamento centrado em colunas funciona muito bem para bancos de dados que são usados principalmente para executar consultas em poucas colunas, mas muitas linhas, como em sistemas analíticos.

Ao mesmo tempo, o armazenamento centrado em coluna seria muito ineficiente para o processamento de transações, pois as atividades de inserção, atualização e exclusão consumiriam muito o disco. O que pode ficar ainda mais claro se observar o comparativo entre o armazenamento centrado em linhas e o centrado em colunas, apresentado pela Figura 29.

Até aqui, foi apresentado algo que já era adotado mesmo para as tecnologias relacionais. Porém, o outro uso do termo Banco de Dados Orientado a Colunas, também chamado de Banco de Dados de Família de Colunas (*column-family*), vem para descrever um tipo de banco de dados NoSQL que leva o conceito de armazenamento centrado em coluna além dos limites do modelo relacional.

Figura 29 – Armazenamento baseado em linha e baseado em colunas



Fonte: Coronel; Morris (2020)

Esse tipo de SGBD, até pela sua natureza NoSQL, não exige que os dados estejam em conformidade com estruturas predefinidas nem oferecem suporte ao SQL para consultas. Além disso, ele organiza dados em pares de chave-valor com chaves mapeadas para um conjunto de colunas no componente de valor.

Embora os bancos de dados de família de colunas usem muitos dos mesmos termos que os bancos de dados relacionais, os termos não significam exatamente as mesmas coisas. Uma coluna é um par de chave-valor semelhante a uma célula de dados em um banco de dados relacional. A chave é o nome da coluna, e o componente de valor são os dados armazenados nessa coluna. E, apesar de estruturalmente ser muito diferente, uma família de colunas é conceitualmente semelhante a uma tabela no modelo relacional.

À medida que mais colunas são adicionadas, fica claro que algumas colunas formam grupos naturais e tais agrupamentos são usados para criar super colunas.

Uma super coluna é um grupo de colunas logicamente relacionadas. A Figura 30 traz a ideia geral da estrutura de um NoSQL do tipo família de colunas:

Figura 30 – Estrutura do NoSQL do tipo Família de Colunas

Column Family Name	CUSTOMERS
Key	Rowkey 1
Columns	City: Nashville Fname: Alfred Lname: Ramas State: TN
Key	Rowkey 2
Columns	Balance: 345.86 Fname: Kathy Lname: Smith
Key	Rowkey 3
Columns	Company: Local Markets, Inc. Lname: Dunne

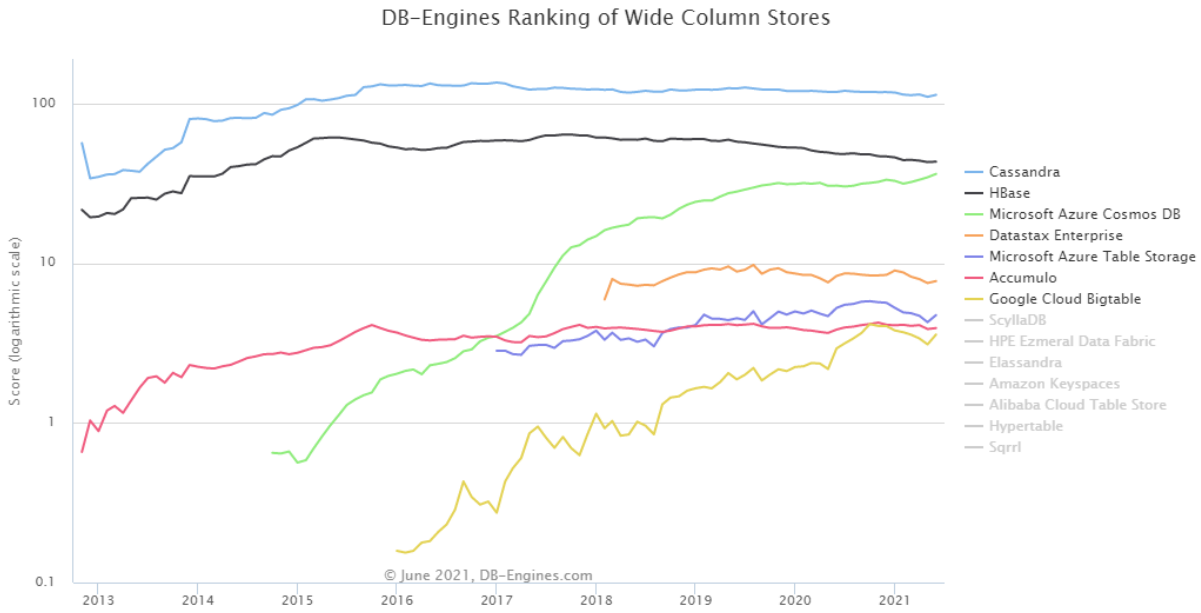
Fonte: Coronel; Morris (2020)

Esse Modelo de banco de dados originou-se do SGBD BigTable do Google. A partir daí, outros produtos de banco de dados orientado a colunas incluem:

- Hbase.
- Hypertable.
- Cassandra. Iniciado como um projeto no Facebook, mas o Facebook o lançou para a comunidade de código aberto, que continuou a desenvolver o Cassandra em um dos bancos de dados orientados a colunas mais populares.
- Mesmo que os bancos de dados de família de colunas (ainda) não suportem SQL padrão, os desenvolvedores do Cassandra criaram uma linguagem de consulta Cassandra (CQL). É semelhante ao SQL em muitos aspectos e é uma das razões mais convincentes para adotar o Cassandra.

Quanto ao ranking de popularidade da DB-Engines, bancos NoSQL do tipo família de colunas estão conforme o seguinte:

Figura 31 – D-Engines: popularidade de bancos NoSQL Família de Colunas



Fonte: https://db-engines.com/en/ranking_trend/wide+column+store

4.5. NoSQL Tipo Grafo

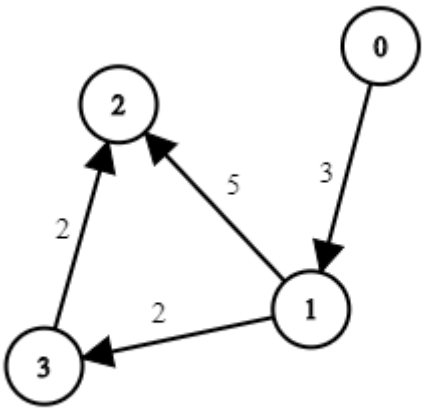
Assim como os bancos NoSQL tipo Documento exigiram um entendimento prévio de JSON, os do tipo Grafo também necessitam que seja introduzido um conceito antes de ser possível caminhar com o entendimento. Nesse caso, faz-se necessário ao menos uma rápida passada pela teoria de Grafos.

A teoria de Grafos é o ramo da matemática que estuda as relações entre os objetos de um determinado conjunto. Foi introduzida no século XVIII pelo matemático suíço Leonhard Euler, que utilizou grafos para resolver o problema conhecido como “As sete pontes de Königsberg”.

Conforme pode ser observado na Figura 32, Grafos possuem uma representação gráfica de fácil entendimento, podendo traduzir problemas mais

complexos em visualizações triviais. Sua estrutura é composta por um conjunto (não vazio) de pontos (vértices) e um conjunto de linhas que ligam esses pontos (arestas).

Figura 32 – Representação gráfica de Grafos



Grafo com custo em suas arestas

Fonte: https://medium.com/xp-inc/grafos-teoria-e-aplica-2a87444df855

Banco de dados NoSQL do tipo grafo (*graph database*) é baseado na teoria de grafos, sendo útil em contextos que sejam ricos em relacionamentos. O foco é modelar e armazenar dados sobre relacionamentos e, assim, fornece uma fonte rica para algoritmos e aplicativos que precisam manipular dados com essa estrutura.

O interesse pelas bases de dados grafo teve origem na área das redes sociais, mas vão muito além de Facebook, Twitter e Instagram, abrangendo questões que dependem do rastreamento de relacionamentos complexos entre objetos:

- Gerenciamento de conhecimento.
- Logística e roteamento.
- Gerenciamento de dados mestre.
- Gerenciamento de identidade e acesso.

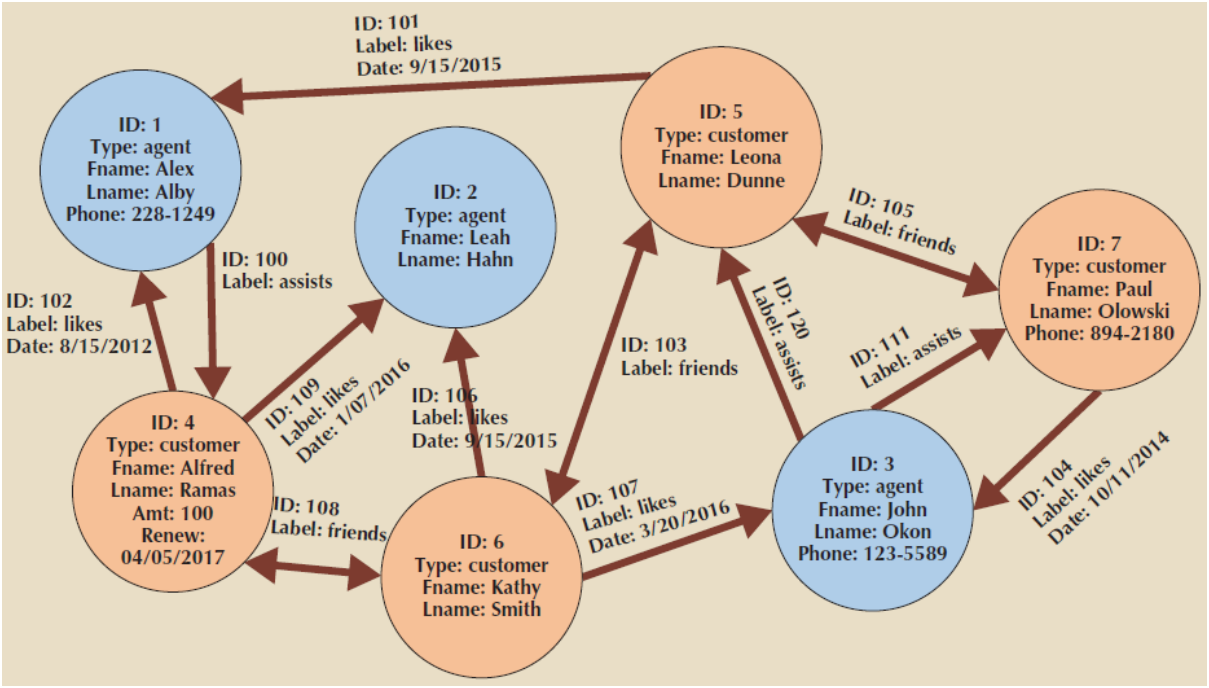
Os componentes primários dos bancos de dados de grafos são nós, arestas e propriedades. Um nó corresponde à ideia de uma instância de entidade relacional. O nó é uma instância específica de algo sobre o qual queremos manter os dados. As

propriedades são como atributos; eles são os dados que precisamos armazenar sobre o nó. Todos os nós podem ter propriedades, mas nem todos os nós precisam ter as mesmas propriedades. Uma aresta é um relacionamento entre nós e podem estar em uma direção ou ser bidirecionais.

Assim como as demais categorias de bancos NoSQL, os do tipo Grafo não requerem estruturas de dados predefinidas. Além disso, não suportam linguagem SQL e são otimizados para fornecer velocidade de processamento para dados intensivos em relacionamento. Destaca-se, ainda, que não apresentam escalabilidade tão aprimorada quanto os outros tipos de bancos NoSQL.

A Figura 33 demonstra uma estrutura de NoSQL do tipo Grafo:

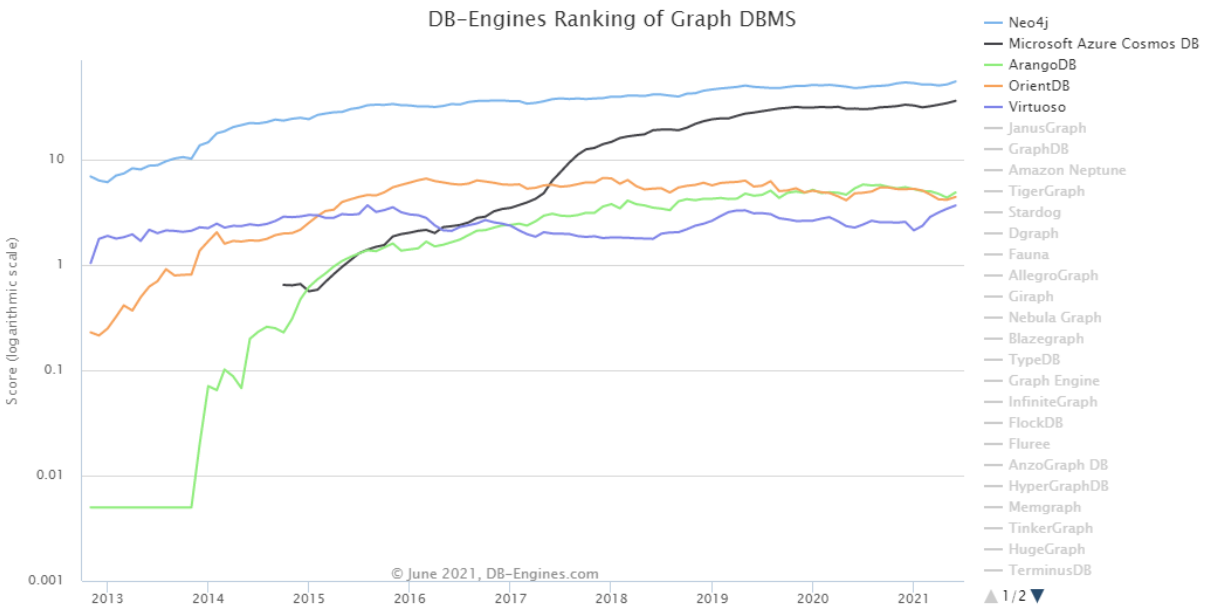
Figura 33 – Estrutura do NoSQL tipo Grafo



Fonte: Coronel; Morris (2020)

No que diz respeito ao mercado, a DB-Engines apresenta o seguinte *ranking*:

Figura 34 – DB-Engines: Ranking de NoSQL tipo Grafo



Fonte: https://db-engines.com/en/ranking_trend/graph+dbms

4.6. Bancos de Dados NewSQL

O termo NewSQL foi cunhado, em 2011, por Matt Aslett, um analista do 451 Group, para descrever uma nova geração de fornecedores de banco de dados relacional. Segundo ele, publicado em *Ten years of NewSQL: Back to the future of distributed relational databases* (2021):

> Quando começamos a usar o termo NewSQL no início de 2011, não era nossa intenção definir uma categoria de mercado. Estávamos simplesmente usando o termo para nos referirmos coletivamente a um grupo de produtos e fornecedores de banco de dados emergentes. O que os vários produtos incluídos neste grupo vagamente afiliado tinham em comum era combinar os benefícios do modelo relacional e arquiteturas distribuídas.
>
> Mal sabíamos que o termo seria usado pela indústria em geral para descrever uma nova categoria de produtos, desencadeando inúmeras campanhas de

marketing e eventos da indústria, sua própria entrada na Wikipedia e trabalhos de pesquisa acadêmica dedicados (ASLETT, 2021).

Na prática, os bancos de dados relacionais suportam os sistemas transacionais nas organizações, e as tecnologias NoSQL não conseguem substituí-los nesse tipo de cenário. Esses sistemas que suportam as operações diárias de negócios dependem de transações compatíveis com ACID e controle de concorrência. Por outro lado, bancos NoSQL, exceto grafos que tem aplicação mais específica, tem foco na distribuição e disponibilidade.

O NewSQL tenta preencher a lacuna entre bancos relacionais e NoSQL, fornecendo transações compatíveis com ACID em uma infraestrutura altamente distribuída.

Assim como os relacionais, bancos de dados NewSQL suportam SQL como interface primária e transações compatíveis com ACID. Em contrapartida, semelhante ao NoSQL, os bancos de dados NewSQL também suportam clusters altamente distribuídos. Contudo, como esperado, nenhuma tecnologia pode fornecer perfeitamente as vantagens de SGBDR e NoSQL, e o teorema CAP ainda se aplica.

Diferentemente dos bancos NoSQL que tiveram rápida aceitação no mercado, o NewSQL apresentou lenta adoção desde o seu surgimento. Como principais motivos para isso, pode-se destacar:

- Desafio computacional de combinar as vantagens de escalabilidade do NoSQL com a estrutura, consistência, desempenho e suporte transacional do modelo relacional.
- Competição em um setor do mercado dominado por gigantes, como Oracle, IBM, Microsoft, Amazon Web Services e Google.
- Surgimento anterior à demanda comercial por processamento de dados distribuídos e serviços gerenciados de banco de dados.
- Evolução do NoSQL que tornou-se alternativa realista para algumas cargas de trabalho potencialmente atendidas por NewSQL.

Matt Aslett separa as tecnologias NewSQL em duas gerações. A primeira, que ainda é da publicação de seu artigo original. E uma outra mais recente, que mostra algumas diferenças importantes na abordagem do mercado.

Dos bancos NewSQL mencionados na publicação de 2011, ditos da primeira geração, poucos sobreviveram:

- VoltDB é o único que continua com a mesma aparência hoje, embora tenha estreitado seu foco e agora esteja voltado principalmente para telecom.
- SingleStore, anteriormente conhecido como MemSQL, continua a visar a oportunidade para cargas de trabalho híbridas - lidando com processamento OLTP e OLAP.
- CodeFutures tornou-se AgilData, em 2015, e agora é um provedor de serviços centrado em nuvem e dados.
- JustOne Database foi rebatizado como Edge Intelligence, em 2017, com foco em *edge computing*.

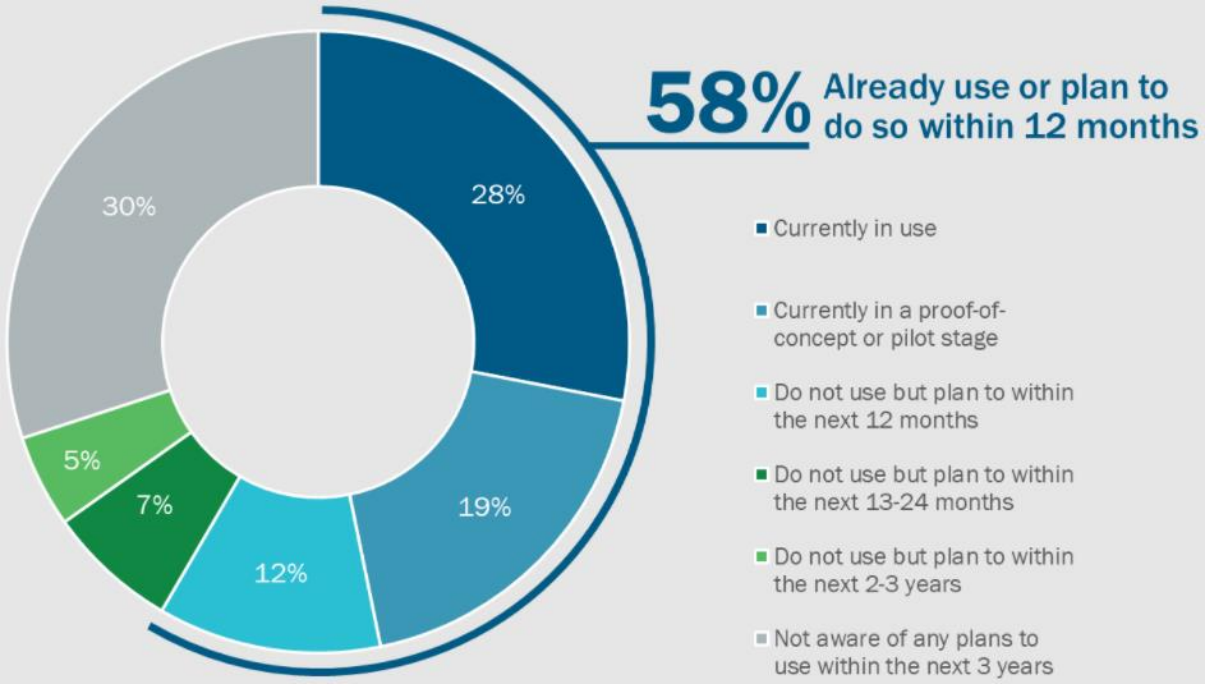
Já a segunda geração parece seguir um caminho mais promissor. A seguir, são apresentados alguns indicadores nesse sentido:

- Cockroach Labs tem mais de 300 clientes e está avaliada em 2 bilhões de dólares.
- PlanetScale anunciou recentemente o lançamento de seu serviço de banco de dados focado no desenvolvedor.
- PingCAP lançou a prévia pública de seu serviço gerenciado *TiDB Cloud*, tendo levantado uma rodada de financiamento de 270 milhões de dólares.
- YugaByte recentemente levantou uma rodada de 48 milhões de dólares para trazer seu financiamento total para 103 milhões de dólares.
- Fauna se estabeleceu como fornecedora de um banco de dados gerenciado sem servidor, consumido por clientes por meio de uma API SQL.
- SGBDs em nuvem horizontalmente escaláveis: AWS Amazon Aurora, Google Cloud Spanner e Banco de Dados Azure para PostgreSQL - Hyperscale.

As ofertas de última geração podem ser consideradas como os exemplos mais verdadeiros de NewSQL, visto que reformularam o conceito de banco de dados relacional para uma arquitetura distribuída globalmente. De modo geral, pararam de usar o termo NewSQL, preferindo SQL distribuído, que é um termo mais descritivo e que demonstra melhor seu valor.

Os fornecedores de SQL distribuído também estão altamente focados no envolvimento do desenvolvedor e estão destacando seus benefícios potenciais em termos de facilidade de adoção, facilidade de uso e melhorias de produtividade. A Figura 35 traz o resultado de uma pesquisa de adoção dessa tecnologia:

Figura 35 – Expectativa de adoção de SQL Distribuído



Fonte: https://www.spglobal.com/marketintelligence/en/news-insights/blog/ten-years-of-newsql-back-to-the-future-of-distributed-relational-databases

O 451 Research estima que, em 2020, os vários provedores de banco de dados NewSQL geraram receita de 587 milhões de dólares. Pequena proporção do mercado de banco de dados relacional que chegou a 35,6 bilhões de dólares e substancialmente menor que os 4,9 bilhões de dólares gerados pelos NoSQL.

Em relação às plataformas disponíveis, o Google Spanner parece liderar esse mercado, sendo o primeiro banco de dados compatível com ACID escalonável de leitura/gravação horizontal. O Amazon Aurora também se destaca como um banco de dados compatível com ACID horizontalmente escalável e que oferece quase todos os recursos SQL. Contudo, seu escalonamento horizontal ainda é apenas para leitura. O CockroachDB também é um SQL distribuído muito proeminente e que avança no mercado.

Capítulo 5. Demonstração de NoSQL e NewSQL

Além de toda a abordagem teórica, a disciplina traz demonstrações de sistemas gerenciadores de banco de dados NoSQL e NewSQL disponibilizadas em formato de vídeo para melhor compreensão e acompanhamento das práticas propostas.

Capítulo 6. Plataformas Analíticas

Apesar das demandas por sistemas que apoiem a tomada de decisão dentro das organizações sejam um assunto já de longa data, uma série de novos conceitos surgiu desde o advento do *Data Warehouse*. Embasadas na computação em nuvem, as novas plataformas analíticas trazem diversos benefícios para o armazenamento de dados de aplicações OLAP. Diante disso, importante compreender o armazenamento de objetos em nuvem, bem como fazer um paralelo entre a abordagem tradicional e moderna de *Data Warehouse*.

6.1. Armazenamento de Objetos em Nuvem

Durante os últimos anos, houve grandes avanços relacionados à computação em nuvem, arquiteturas distribuídas para bancos de dados e desenvolvimento de software baseado na Web. Esses avanços impulsionaram mudanças fundamentais na infraestrutura de armazenamento.

Os sistemas baseados em arquivos apresentaram evoluções para novas arquiteturas abertas de armazenamento. Nesse contexto, surgiu o armazenamento baseado em objeto (*object-based storage*).

Nesse esquema, os dados são gerenciados na forma de objetos, em vez de arquivos organizados em blocos. Esses objetos carregam metadados que contêm propriedades que podem ser usadas para gerenciá-los. Além disso, cada objeto carrega um identificador global exclusivo que é usado para localizá-lo.

O armazenamento de objetos também permite flexibilidade adicional em termos de interfaces. Dá controle a aplicativos que podem controlar os objetos diretamente e permite que os objetos sejam endereçáveis em um amplo espaço de nomes (*namespace*) que abrange vários dispositivos. Adicionalmente, fornece suporte à replicação e distribuição de objetos.

Em geral, o armazenamento baseado em objetos é ideal para a persistência escalonável de grandes quantidades de dados não estruturados. Como exemplos de uso, pode-se mencionar:

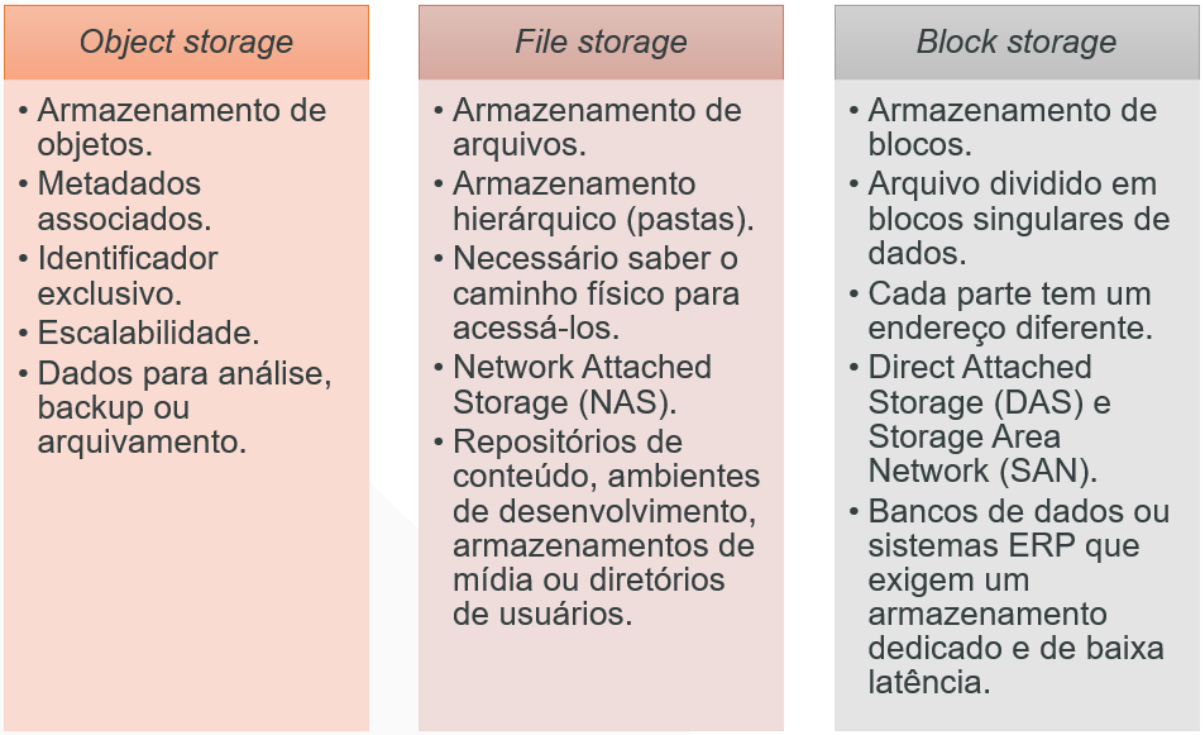
- Facebook: fotos.
- Spotify: músicas.
- Dropbox: arquivos diversos.

Em relação aos principais fornecedores de nuvem, as principais ofertas são as seguintes:

- Amazon Web Service S3.
- Microsoft Azure Blob Storage.
- Google Cloud Storage.

A Figura 36 traz um comparativo entre as abordagens tradicionais de armazenamento baseado em arquivos e blocos em contraste ao baseado em objetos:

Figura 36 – Object, file e block storage



Fonte: Dados da pesquisa

Até pela maturidade da solução, o AWS S3 aparece como referência de *object storage*. Entretanto, Microsoft e Google vem rapidamente evoluindo suas ofertas. A Figura 37 mostra um comparativo entre esses três principais fornecedores:

Figura 37 – Comparativo de armazenamento entre fornecedores de nuvem

	AWS	Azure	Google Cloud
Archival storage	S3 Glacier, S3 Glacier Deep Archive	Archive Storage	Archive Storage
Backup	AWS Backup	Azure Backup	N/A
Block storage	Amazon Block Store (EBS)	Disk Storage	Persistent Disk, Local SSD
File storage	Amazon Elastic File Service (EFS), Amazon FSx for Windows File Server, Amazon FSx for Lustre	File Storage, Azure NetApp Files	Filestore
Object storage	Amazon S3	Blob storage	Cloud Storage, Cloud Storage for Firebase

Fonte: https://searchcloudcomputing.techtarget.com/feature/A-cloud-services-cheat-sheet-for-AWS-Azure-and-Google-Cloud

Quando se fala em armazenamento escalável, impossível não pensar em Apache Hadoop. Por isso mesmo, importante entender em que aspecto essa tecnologia concorre com o armazenamento baseado em objetos apresentados até aqui neste capítulo.

O Apache Hadoop é um *framework open source*, escalável e tolerante a falhas escrito em Java. Ele processa eficientemente grandes volumes de dados em um *cluster* de hardware de baixo custo. O Hadoop é uma plataforma de armazenamento e processamento de dados com três componentes principais:

- Hadoop Distributed File System (HDFS): armazenamento.
- Map-Reduce: processamento.

- YARN: gerenciamento.

Especificamente falando de HDFS, trata-se do sistema de arquivos utilizado para armazenamento de dados na plataforma Hadoop e que se caracteriza por ser:

- Altamente tolerante a falhas.
- Distribuído.
- Confiável.
- Escalável.

O HDFS foi desenvolvido para lidar com grandes volumes de dados e pode ser usado com arquivos de grandes tamanhos (TB). Um arquivo é dividido em blocos e armazenado de forma distribuída em várias máquinas de acordo com o fator de replicação configurado.

Pode-se afirmar que armazenamento em nuvem fornece elasticidade, com melhor disponibilidade e durabilidade, maior desempenho e menor custo que *clusters* HDFS tradicionais. O HDFS transformou em *commodity* o armazenamento de *Big Data*, tornando mais barato armazenar e distribuir uma grande quantidade de dados. No entanto, em uma arquitetura nativa da nuvem, o benefício do HDFS é mínimo e não compensa a complexidade operacional. A Figura 38 traz um comparativo entre HDFS e AWS S3:

Figura 38 – Comparativo entre HDFS e AWS S3

	S3	HDFS	S3 vs HDFS
Elasticidade	sim	Não	S3 é mais elástico
Custo / TB / mês	$ 23	$ 206	10X
Disponibilidade	99,99%	99,9% (estimado)	10X
Durabilidade	99,999999999%	99,9999% (estimado)	10X +
Gravações transacionais	Sim com DBIO	sim	Comparável

Fonte: https://databricks.com/blog/2017/05/31/top-5-reasons-for-choosing-s3-over-hdfs.html

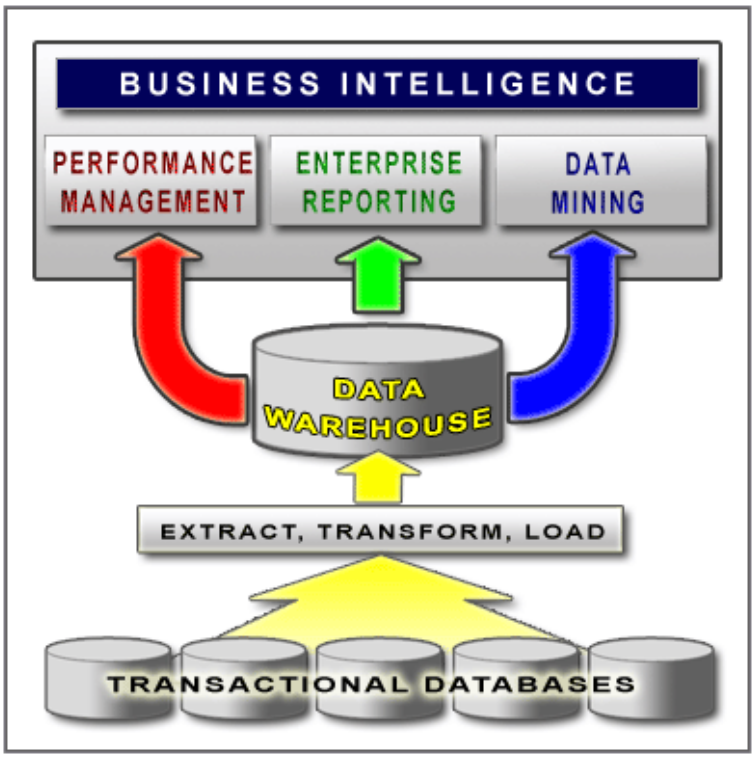
6.2. Demonstração de Cloud Storage

Além de toda a abordagem teórica, a disciplina traz demonstração de *cloud storage* disponibilizada em formato de vídeo para melhor compreensão e acompanhamento das práticas propostas.

6.3. Business Intelligence e *Data Warehouse*

De acordo com Barbieri (2001), *Business Intelligence* é a utilização de variadas fontes de informação para se definir estratégias de competitividade nos negócios da empresa. Nesse mesmo caminho, Alcantara (2006) diz que é o conjunto de ferramentas/aplicações que tem por objetivo fornecer as informações necessárias para que os tomadores de decisões possam administrar de forma mais consciente a organização. Na Figura 39 é possível observar a estrutura de um BI tradicional:

Figura 39 – Estrutura de BI tradicional



Fonte: http://sxcma.cn/uploads/allimg/181120/1J1204a6-0.gif

Nesse ambiente tradicional, é possível observar bancos transacionais que dão suporte ao dia a dia da organização, tais como sistema de RH, venda e estoque, servindo de fonte de dados. Para isso, existe um processo de ETL, extração, transformação e carga (*Extract, Transform and Load*) que se conecta a essas várias fontes transacionais, extrai, dali, os dados necessários e, após as transformações devidas, carrega para um banco de dados especial chamado *Data Warehouse* (DW). A essa base são conectadas ferramentas OLAP que permitirão diversas visualizações por meio de cubos de dados.

Nesse contexto, o DW tem papel de destaque. Segundo Turban (2009), um *Data Warehouse* (DW) é um conjunto de dados produzido para oferecer suporte à tomada de decisões; é também um repositório de dados atuais e históricos de possível interesse aos gerentes de toda a organização. Os dados normalmente são estruturados de modo a estarem disponíveis em um formato pronto para as atividades de processamento analítico (ex.: processamento analítico online - OLAP, *Data Mining*, consultas, geração de relatórios, outras aplicações de suporte à decisão). Portanto, um DW é uma coleção de dados orientada por assunto, integrada, variável no tempo e não volátil, que proporciona suporte ao processo de tomada de decisões da gerência.

O que há de especial no DW é justamente o modelo dimensional utilizado para o armazenamento de dados. A modelagem dimensional é a técnica utilizada para se ter uma visão multidimensional dos dados, sendo utilizada para sumarizar e reestruturar os dados e apresentá-los em visões que suportem a análise de seus valores. Esse modelo possui três elementos básicos: fatos, dimensão e medidas (métricas).

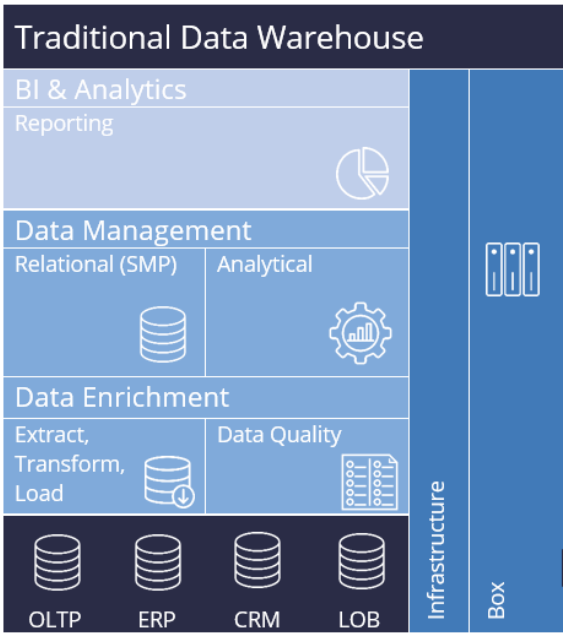
O processo de ETL, como já foi dito, é o responsável pelo transporte dos dados das bases transacionais para o DW. Não sem antes promover os tratamentos necessários para a limpeza e consolidação desses dados. Por ser uma atividade trabalhosa, costuma demandar boa parte do tempo destinada à execução do projeto de BI.

Relatórios OLAP, *dashboards* e mesmo *Data Mining* surgem como possibilidades distintas de uso para os dados consolidados no DW. Indo desde a simples exploração dos dados até a busca automatizada de padrões presentes na base analítica.

6.4. Data Warehouse Moderno

Já discutimos o ambiente de BI tradicional. Nele, foi possível entender o papel central do *Data Warehouse* que, de uma outra forma, aparece também na Figura 40:

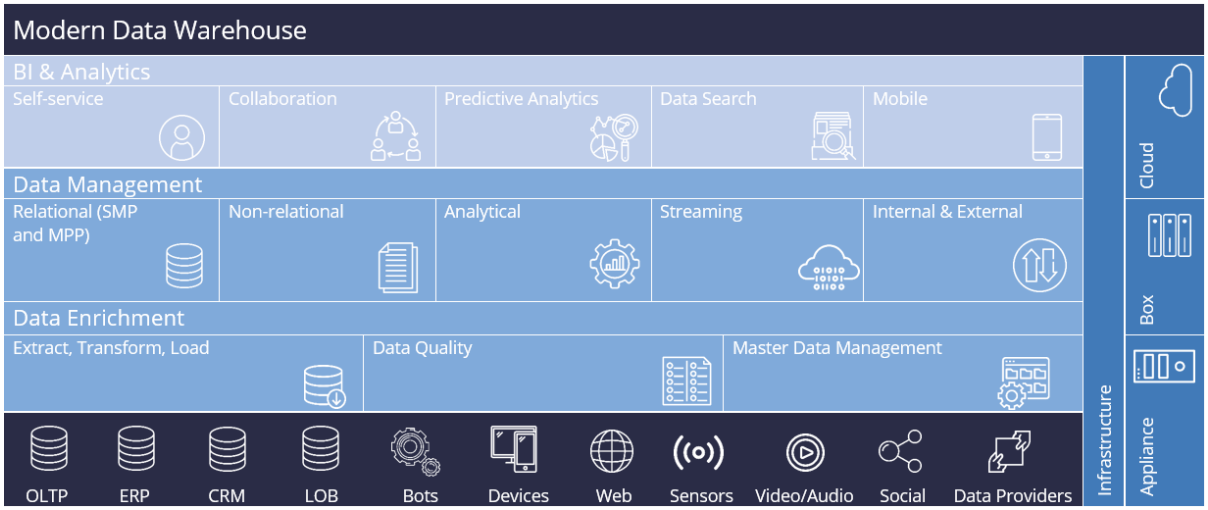
Figura 40 – Date Warehouse tradicional



Fonte: https://www.predicagroup.com/app/uploads/2020/08/Modern-Data-Warehouse-model.png

Ocorre que, conforme já discutido, um novo cenário corporativo surgiu nos últimos anos e os desafios de *Big Data* se fizeram presentes. O que reflete, também, na abordagem de DW. A Figura 41 mostra esse aumento de complexidade:

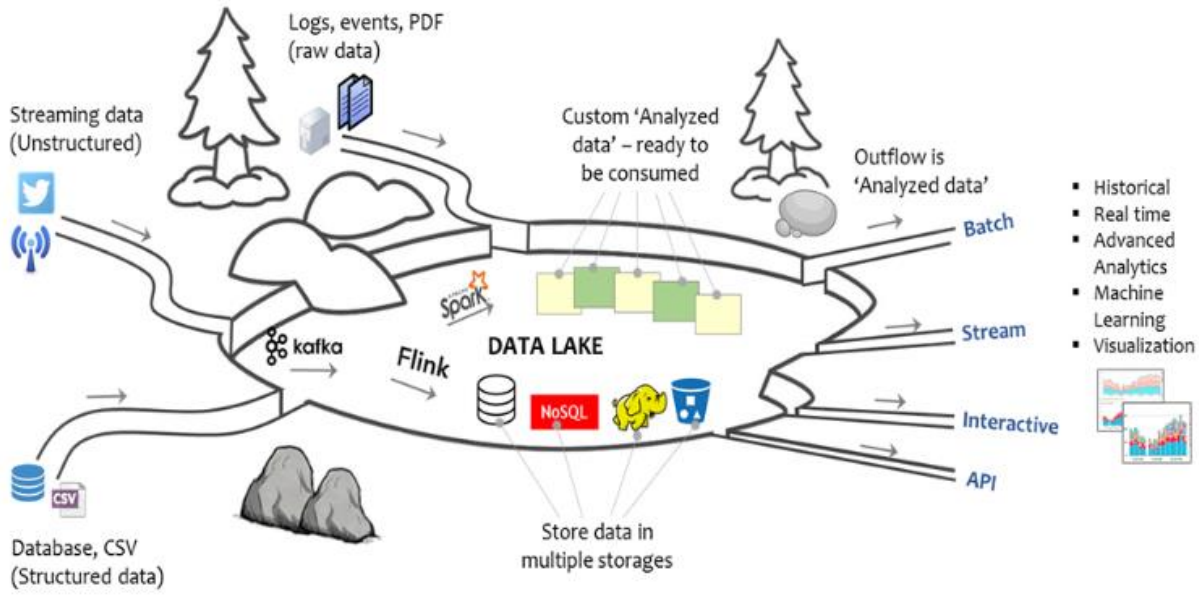
Figura 41 – Data Warehouse moderno



Fonte: https://www.predicagroup.com/app/uploads/2020/08/Modern-Data-Warehouse-model.png

Em face dessa maior complexidade e por conta do grande volume de dados gerado em formatos diversos e a uma velocidade muito grande, passa-se a adotar o que se convencionou chamar de *Data Lake*. Esse sendo um repositório de armazenamento que mantém uma grande quantidade de dados brutos em seu formato nativo até que sejam necessários. Utilizado para investigar, explorar, experimentar, refinar e arquivar dados e que permite a integração de dados estruturados, semiestruturados (JSON, XML), não estruturados (texto, áudio, vídeo) e dados gerados por máquina (IoT). De forma simplificada, não passa de uma maneira de descrever qualquer grande *pool* de dados em que o esquema e os requisitos de dados não são definidos até que os dados sejam consultados (*schema on read*). A Figura 42 mostra a estrutura geral de um *Data Lake*:

Figura 42 – Estrutura do *Data Lake*



Fonte: https://www.gslab.com/blogs/shifting-to-data-lakes?hcb=1

Já a Figura 43 traz um comparativo entre a abordagem tradicional, utilizando somente *Data Warehouse* e com o uso de *Data Lake*:

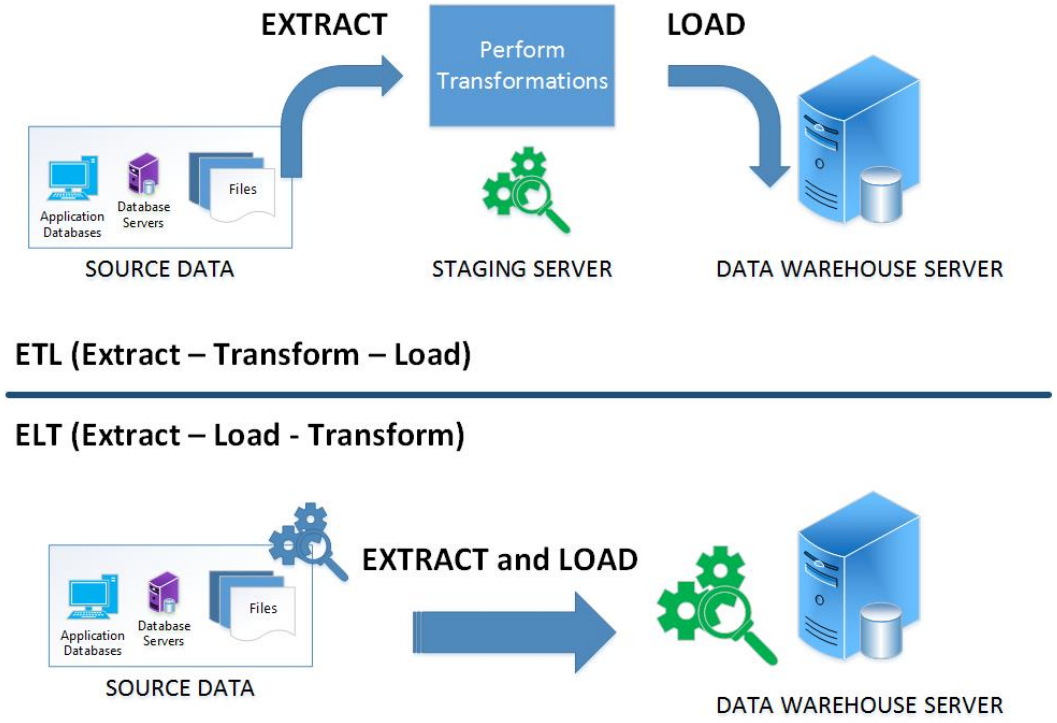
Figura 43 – Comparativo de Data Warehouse e Data Lake

DATA WAREHOUSE	vs.	DATA LAKE
structured, processed	DATA	structured / semi-structured / unstructured, raw
schema-on-write	PROCESSING	schema-on-read
expensive for large data volumes	STORAGE	designed for low-cost storage
less agile, fixed configuration	AGILITY	highly agile, configure and reconfigure as needed
mature	SECURITY	maturing
business professionals	USERS	data scientists et. al.

Fonte: https://blog.sysfore.com/data-lake-is-it-the-future-for-big-data/

Outra importante mudança ocorre com o processo de ETL. Com o aumento da variedade, do volume e da velocidade dos dados chegando aos repositórios analíticos, muitas vezes torna-se inviável promover as transformações necessárias durante as cargas de dados. Por isso, surgiu, como alternativa, o deslocamento da transformação dos dados para depois da carga. Ou seja, extração, carga e transformação. Tudo conforme destacado na Figura 44:

Figura 44 – ETL x ELT



Fonte: https://www.cartelis.com/blog/comparatif-logiciels-etl/

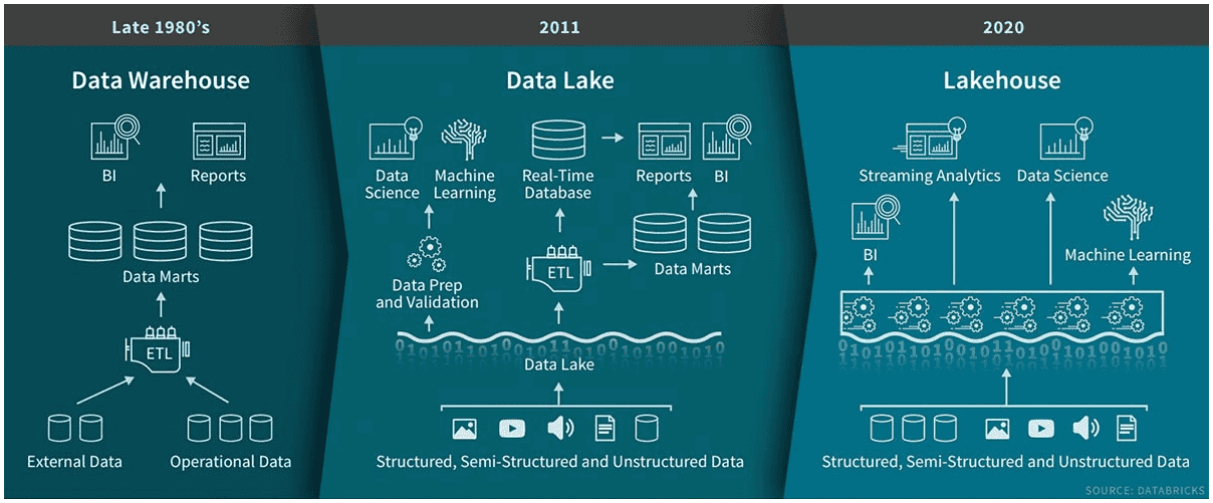
Os desafios da correta implementação e uso de *Data Lakes* são enormes. O que dá espaço para o surgimento de uma situação conhecida como *Data Swamp*. Um pântano de dados (*data swamp*) é um lago de dados (*data lake*) deteriorado e não gerenciado, inacessível aos usuários pretendidos ou que fornece pouco valor. Pela liberdade de entrada de dados, o *Data Lake* pode sofrer com uma falta de estruturação. Para evitar-se que um *Data Lake* vire um *Data Swamp*, é preciso existir governança de dados que vem para estruturar e gerenciar o conteúdo do lago, deixando-o mais acessível e valioso.

Muitas empresas operam seus *Data Warehouses* independentemente de seus *Data Lakes*, aproveitando o DW para obter valiosos *insights* de negócios e os *Data Lakes* para armazenamento e ciência de dados. Já outras combinam seu *Data Lake* com seu DW em uma única plataforma de dados dentre uma das seguintes opções:

- *Data Warehouse* trabalhando em paralelo com o *Data Lake*.
- *Data Warehouse* embutido no *Data Lake*.

Daí surge a ideia de um *Data Lakehouse*, servindo como uma plataforma única para *Data Warehousing* e *Data Lake*. Inclusive, algumas empresas adotam uma arquitetura *Lakehouse* que vai além do *Data Lake* e do *Data Warehouse*. Eles fazem isso usando consultas federadas para integrar dados através do *Data Lake*, *Data Warehouse* e quaisquer outros serviços de dados. Essa arquitetura facilita a consulta de dados em várias fontes, criando uma camada semântica e usando um mecanismo de consulta distribuída. A Figura 45 mostra um comparativo entre essas possíveis abordagens analíticas:

Figura 45 – DW, Data Lake e Lakehouse



Fonte: https://databricks.com/wp-content/uploads/2020/01/data-lakehouse.png

Como chave para o sucesso de arquiteturas *Lakehouse* aparece, conforme já mencionado, a federação ou virtualização de dados. A virtualização de dados é

baseada na execução de processamento distribuído sobre várias fontes heterogêneas de gerenciamento de dados. As consultas às diversas fontes são combinadas (federação) como uma visualização (*view*) centralizada e virtual. Essas visualizações ficam, então, disponíveis para os aplicativos, ferramentas de relatório, dentre outros componentes de infraestrutura de gerenciamento de dados. A virtualização faz com que não seja necessário executar a movimentação de dados, e seu armazenamento físico em estruturas criadas apenas para consolidação dos dados, fornecendo uma camada de abstração acima da implementação física dos dados para simplificar a lógica de consulta.

A virtualização de dados é um componente da arquitetura de integração de dados e pode ser observada sob diferentes nomenclaturas, dentre eles *Data Warehouse* lógico, federação de dados, banco de dados virtual, *Data Warehouse* descentralizado. A virtualização de dados permite a integração de dados de várias fontes analíticas distintas, mantendo-os no seu local original. Alguns autores chegam a mencionar que é uma alternativa à construção de um *Data Warehouse* ou pode, ainda, ser utilizada em conjunto, sendo essa última opção a que parece mais adequada para a maioria dos casos. A Figura 46 traz um resumo das abordagens de virtualização de dados com suas principais características e ferramentas:

Figura 46 – Abordagens de virtualização de dados

APPROACH	DESCRIPTION	PROS	CONS	EXAMPLE VENDORS
Core Data Virtualization Platforms	A stand alone virtual data (semantic) layer that abstracts the physical data platform and location and allows queries across disparate data platforms.	Platform independent True semantic layer Caching for performance	Separate service to manage Data needs to be modeled	AtScale Denodo Dremio TIBCO DV
SQL-On-Anything	A query federation engine that allows a single SQL query to combine data from more than one data platform.	Good for file-based access Scale out with clustering	Users need to understand remote schemas Unpredictable performance	Amazon Athena Apache Drill Presto
Remote Data Source Bridges	A database extension that is embedded in a RDBMS that allows SQL access to remote databases using "external" tables.	Good when one primary warehouse is used No separate service to manage	Users need to understand remote schemas Unpredictable performance	Amazon Redshift Spectrum IBM Db2 Big SQL Microsoft SQL Server Polybase Oracle Big Data SQL Teradata QueryGrid
Autonomous Data Warehouses	A platform that automates the modeling, integration, and connectivity of source data which is then loaded into a target data platform.	Data stays in one place Performance is easier to predict	Requires data movement (data copies) Data latency or staleness	Data Virtuality Infoworks.io Incorta

Fonte: https://www.atscale.com/wp-content/uploads/2020/01/Complete_Buyers_January21.pdf

6.5. Demonstração de Plataforma Analítica

Além de toda a abordagem teórica, a disciplina traz demonstração de plataforma analítica disponibilizada em formato de vídeo para melhor compreensão e acompanhamento das práticas propostas.

Referências

ARMBRUST, Michael; GHODSI, Ali; XIN, Reynold; ZAHARIA, Matei. **Lakehouse:** A New Generation of Open Platforms that Unify DataWarehousing and Advanced Analytics. *In***:** 11th Annual Conference on Innovative Data Systems Research (CIDR '21), 2021, Online. Disponível em: http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf. Acesso em: 12 nov. 2021.

ATSACALE. **The Complete Buyer's Guide For Intelligent Data Virtualization**. EUA: Boston, 2020. 17 p. Disponível em: https://www.atscale.com/complete-buyers-guide-for-intelligent-data-virtualization/. Acesso em: 12 nov. 2021.

BARBIERI, Carlos. **BI2 Business Intelligence**: Modelagem e Qualidade. 1. ed. Rio de Janeiro: Elsevier, 2011. 416p.

BDQ – BIG DATA QUARTERLY. **Data Sourcebook**. EUA: New Providence, 2020. 40 p. Disponível em: https://www.dbta.com/DBTA-Downloads/SourceBook/Data-Sourcebook-2020-10511.aspx. Acesso em: 12 nov. 2021.

CORONEL, Carlos; MORRIS, Steven. **Database Systems**: Design, Implementation, and Management. 13. ed. Boston: Cengage Learning, 2018. 816p.

DZONE. **Data Persistence**. EUA: Durham, 2021. 40 p. Disponível em: https://dzone.com/trendreports/data-persistence-3. Acesso em: 12 nov. 2021.

DZONE. **The Database Evolution:** SQL, NoSQL in the Age of Big Data. EUA: Durham, 2020. 39 p. Disponível em: https://dzone.com/trendreports/database-evolution. Acesso em: 12 nov. 2021.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Fundamentals of Database Systems**. 7. ed. Hoboken: Pearson, 2015. 1272p.

FORRESTER. **The Forrester Wave:** Cloud Data Warehouse, Q1 2021. EUA: Cambridge, 2021. Disponível em: https://cloud.google.com/forrester-wave-cloud-data-warehouse-2021. Acesso em: 12 nov. 2021.

FULLER, Matt; MOSER, Manfred; TRAVERSO, Martin. **Trino:** The Definitive Guide. 1. ed. Sebastopol: O'Reilly, 2021. 288p.

GARTNER. **Magic Quadrant for Analytics and Business Intelligence Platforms**. EUA: Stamford, 2021. Disponível em: https://info.microsoft.com/ww-Landing-2021-Gartner-MQ-for-Analytics-and-Business-Intelligence-Power-BI.html?LCID=EN-US. Acesso em: 12 nov. 2021.

GARTNER. **Magic Quadrant for Cloud Database Management Systems**. EUA: Stamford, 2020. Disponível em: https://pages.awscloud.com/GLOBAL-multi-DL-gartner-mq-cloud-dbms-2020-learn.html?trk=ar_card. Acesso em: 12 nov. 2021.

GARTNER. **Magic Quadrant for Data Management Solutions for Analytics**. EUA: Stamford, 2019. Disponível em: https://b2bsalescafe.files.wordpress.com/2019/09/gartner-magic-quadrant-for-data-management-solutions-for-analytics-january-2019.pdf. Acesso em: 12 nov. 2021.

GARTNER. **Predicts 2021:** Data Management Solutions - Operational Efficiency Rises to the Top. EUA: Stamford, 2020. Disponível em: https://www.datastax.com/resources/ report/gartner-report-predicts-2021-data-management-solutions. Acesso em: 12 nov. 2021.

HARRISON, Guy. **Next Generation Databases**: NoSQL, NewSQL, and Big Data. 1. ed. Berkeley: Apress, 2015. 235p.

INMON, William H. **Como construir o data warehouse**. 1. ed. Rio de Janeiro, Campus, 1997. 388p.

KIMBALL, Ralph; ROSS, Margy. **The data warehouse toolkit**: guia completo para modelagem dimensional. 1 ed. Rio de Janeiro: Campus, 2002. 494p.

KNOB, Ronan; SCHREINER, Geomar; FROZZA, Angelo; MELLO, Ronaldo. Uma Análise de Soluções NewSQL. *In:* ESCOLA REGIONAL DE BANCO DE DADOS (ERBD), 2019, Chapecó. **Anais**. Porto Alegre: Sociedade Brasileira de Computação, 2019. p. 21-30.

PAVLO, Andrew. **Advanced Database Systems:** History of Databases. *In:* Carnegie Mellon University, Course 15-721, Spring, 2020. Disponível em: https://15721.courses.cs.cmu.edu/spring2020/slides/01-history.pdf. Acesso em: 12 nov. 2021.

PAVLO, Andrew. **Intro to Database Systems:** Course Intro & Relational Model. *In:* Carnegie Mellon University, Course 15-445/15-645, Fall, 2020. Disponível em: https://15445.courses.cs.cmu.edu/fall2020/slides/01-introduction.pdf. Acesso em: 12 nov. 2021.

PAVLO, Andrew. **Intro to Database Systems:** Intermediate SQL. *In:* Carnegie Mellon University, Course 15-445/15-645, Fall, 2020. Disponível em: https://15445.courses.cs.cmu.edu/fall2020/slides/02-advancedsql.pdf. Acesso em: 12 nov. 2021.

PAVLO, Andrew; ASLETT, Matthew. **What's Really New with NewSQL?**. *In:* SIGMOD Record, 2016, Vol. 45, No. 2. Disponível em: https://db.cs.cmu.edu/papers/2016/pavlo-newsql-sigmodrec2016.pdf. Acesso em: 12 nov. 2021.

RAMAKRISHNAN, Raghu; GEHRKE, Johannes. **Database Management Systems**. 3. ed. New York: Mc Graw Hill, 2002. 1104p.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Database System Concepts**. 6. ed. New York: Mc Graw Hill, 2010. 1376p.

TURBAN, Efraim *et al*. **Business Intelligence**: um enfoque gerencial para a inteligência do negócio. 1. ed. Porto Alegre: Bookman, 2009. 253p.

TURCK, Matt. **Resilience and Vibrancy:** The 2020 Data & AI Landscape. Mattturck, 2020. Disponível em: https://mattturck.com/data2020/. Acesso em: 12 nov. 2021.