FIAP ON

ARQUITETURA NOSQL

CONCEITOS E DISCUSSÕES SOBRE NOSQL

MILTON GOYA E REGINA CANTELE

1

LISTA DE FIGURAS

Figura 1.1 – Quadrante Mágico para Gerenciamento de Metadados....8
Figura 1.2 – Mapeamento de instruções entre linguagens....9
Figura 1.3 – Transação....11
Figura 1.4 – Teorema CAP....13
Figura 1.5 – Número de tecnologias por categoria....16
Figura 1.6 – Modelo Colunar....18
Figura 1.7 – Persistência Poliglota....24
Figura 1.8 – Banco de Dados mais populares....25
Figura 1.9 – Forrester Wave ™: Big Data NoSQL....26
Figura 1.10 – Critérios para escolha SQL e NoSQL....27
Figura 1.11 – Árvore de decisão para requisitos NoSQL....27

LISTA QUADROS

Quadro 1.1 – Comparação ACID x BASE 15
Quadro 1.2 – Modelo Chave-valor 17

SUMÁRIO

1 CONCEITOS E DISCUSSÕES SOBRE NOSQL 5
1.1 Contexto 5
1.2 Histórico 6
1.3 Características NoSQL 7
1.3.1 Schema less 7
1.3.2 Linguagens Independentes 9
1.3.3 Funcionalidade determina escolha 10
1.3.4 Gerenciamento de transações 11
1.4 Categorias NoSQL 16
1.4.1 Bancos de dados chave-valor 17
1.4.2 Bancos de dados colunares 18
1.4.3 Bancos de dados orientado a grafos 19
1.4.4 Bancos De Dados Orientados A Documentos 20
1.4.5 Search Engines 21
1.4.6 Time Series 22
1.5 Soluções Poliglotas 23
1.6 Critérios Para Escolha 27
REFERÊNCIAS 30

1 CONCEITOS E DISCUSSÕES SOBRE NOSQL

Um dos componentes essenciais na arquitetura de banco de dados é o armazenamento de diferentes tipos de dados, principalmente os semi e não estruturados, com o processamento de um grande volume de dados e com alta disponibilidade. Esta nova geração de banco de dados ficou conhecida como NoSQL (N*ot only SQL* ou não somente SQL) e trouxe novas técnicas e teorias para acessá-los. A conhecida linguagem padrão SQL (*Structured Query Language*) associada aos bancos de dados relacionais abre espaço para novas linguagens e formas de tratar os metadados associados a esta nova geração de banco de dados.

1.1 Contexto

Um dos desafios da área de computação é armazenar as informações da empresa de forma segura e de fácil acesso. Em busca desse objetivo, passamos por várias fases, saímos do armazenamento de relatórios impressos em papel, passamos pelo armazenamento em fita, pelos discos rígidos e, finalmente, pelo armazenamento de dados em memória de estado sólido e pelo armazenamento em nuvem. Mesmo com toda essa evolução nos sistemas de armazenamento, uma coisa permaneceu constante nas últimas décadas – os dados são armazenados em um banco de dados relacional.

Existem inúmeros motivos para o sucesso do banco de dados relacional: seu armazenamento é estável, seu modelo é bem compreendido e fácil de aprender, possui uma linguagem de consulta aos dados simples e padronizada e pode ser acessado a partir de diversas plataformas de programação (HOWS, 2015).

O modelo do banco de dados relacional atende perfeitamente as necessidades transacionais das empresas e corporações, e ainda deve perdurar por várias décadas. No entanto, por uma série de fatores que discutiremos mais à frente, o modelo relacional não atendeu as demandas de grandes empresas digitais, como Google, Facebook, eBay, LinkedIn, entre outras, para processar grandes volumes de dados na variedade e velocidade necessários para o negócio. O NoSQL surgiu para atender esse nicho de processamento (SADALAGE, 2013).

1.2 Histórico

O termo “NoSQL” foi usado pela primeira vez em 1998 – ironicamente – em um banco de dados relacional de código aberto, denominado Strozzi NoSQL. Esse banco relacional armazena seus dados em tabelas, em arquivos ASCII, e nele as colunas são separadas por tabulações. O nome desse banco é oriundo do fato de não usar a linguagem SQL como ferramenta de consulta. Os dados são manipulados por meio de comandos *shell scripts* combinados com *pipelines* do sistema operacional Unix. Apesar da coincidência de nomes, esse NoSQL não tem relação com o atual NoSQL que conhecemos (STROZZI, 2018).

O termo NoSQL, da forma como usado nos dias de hoje, foi proposto por um desenvolvedor da Rackspace, Erick Evans, em uma reunião informal realizada em 11 de junho de 2009, em São Francisco, organizada pelo desenvolvedor de software londrino, Johan Oskarsson, funcionário da Last.FM (SADALAGE, 2013).

Johan Oskarsson estava em São Francisco para um evento sobre Hadoop e se interessou em conhecer mais sobre os bancos de dados não relacionais. Como tinha pouco tempo, propôs esse encontro informal, o NoSQL Meetup, no qual pedia por “bancos de dados não relacionais, distribuídos e de código aberto”.

A partir desse evento, o termo NoSQL se popularizou. As palestras realizadas durante a reunião tiveram como tema os bancos não relacionais: Voldemort (LinkedIn), Cassandra (Facebook), DynamoDB (Amazon), HBase (Google), Hypertable, CouchDB e MongoDB. Exemplos de banco de dados desenvolvidos pelos engenheiros dos provedores como LinkedIn, Facebook, Amazon e Google para resolverem os problemas de armazenamento e acessos eficientes de grande quantidade de dados bem como problemas com escalabilidade e disponibilidade presentes nos bancos existentes na época. O significado mais aceito para NoSQL é *Not only Structured Query Language* ou, simplesmente, *Not only SQL*, em tradução livre, “não somente SQL”. Essa mudança de *NO SQL* para *Not Only SQL* surge da necessidade de enfatizar que esses sistemas podem, sim, suportar linguagens de consulta semelhantes a SQL relacionais como o Strozzi NoSQL citado anteriormente, daí a ironia do termo.

Uma definição mais abrangente foi dada por Mccreary e Keylly (2014) como NoSQL sendo um conjunto de conceitos e tecnologias relacionados a desempenho, confiabilidade e agilidade associado ao processamento rápido e eficiente de coleções de dados.?)

O termo é usado para definir um movimento e não uma tecnologia específica (HOWS, 2015). Não significa exclusão do uso de recursos de SGBDRs (Sistemas Gerenciadores de Banco de Dados Relacionais) e SQL. Forrester (2019) define NoSQL como:

> Um sistema de gerenciamento de banco de dados não relacional que fornece armazenamento, processamento e acesso de qualquer tipo de dados e que suporta uma arquitetura horizontal e escalável baseada em um modelo de dados flexível e sem esquemas. (FORRERSTER, ano, p. 2)

A tecnologia NoSQL passou a ser utilizada rapidamente pelas empresas com soluções na Web por agregar recursos de gerenciamento de dados que atendem às necessidades de aplicativos modernos: melhor produtividade no desenvolvimento de aplicativos por meio de modelo de dados mais flexíveis; maior capacidade de escalar dinamicamente para suportar crescimento no número de usuários e na quantidade de dados; desempenho aprimorado para satisfazer as expectativas dos usuários que desejam aplicativos altamente responsivos e para permitir um processamento mais complexo de dados. NoSQL passa a ser considerado nas soluções para sites interativos e aplicativos móveis.

1.3 Características Nosql

Algumas características são comuns aos bancos de dados NoSQL e aqui destacamos schema less, linguagens independentes, funcionalidade determina escolha e gerenciamento de transações.

1.3.1 Schema Less

Schema corresponde aos metadados associados às instâncias armazenadas nos bancos de dados. No SQL existe um conjunto de comandos denominados DDL (Data Definition Language) para definir estes metadados – criação dos elementos do

modelo relacional – tabelas, views, restrições, entre outros. Já nos bancos NoSQL, os metadados são definidos com os comandos de inserção. Assim, as instâncias dos dados podem sofrer modificações nos seus metadados e nem por isso deixam de ser armazenadas no banco de dados.

Com o crescimento do volume e diversidade de fontes de dados, é essencial ter ferramentas para democratização de dados dentro das organizações. Surgem ferramentas denominadas catálogos de dados para acompanhar estes metadados. O Gartner definiu um catálogo de dados como uma ferramenta que "cria e mantém um inventário de ativos de dados por meio da descoberta, descrição e organização de conjuntos de dados distribuídos".

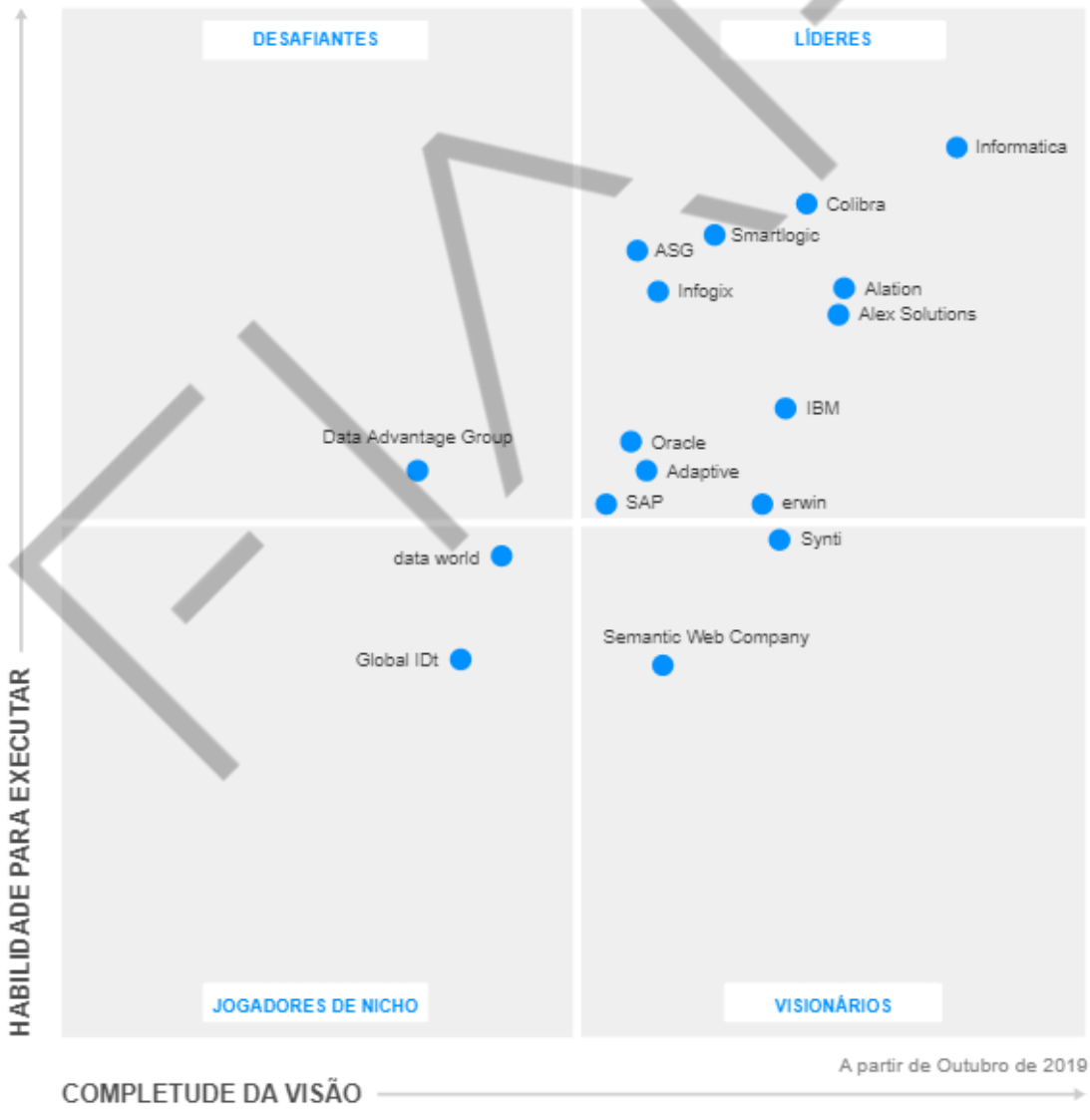

Figura 1.1 – Quadrante Mágico para Gerenciamento de Metadados
Fonte: Gartner (2019)

As ferramentas de linhagem de dados documentam o fluxo de dados para dentro e fora dos processos de uma organização e permitem realizar uma análise de impacto adequada em caso de problemas ou alterações nos ativos de dados à medida que eles se movem pelos pipelines. As principais ferramentas open source são Talend Open Studio, Apatar, CloverETL, Kylo, Dremio, Jaspersoft ETL, Octopai e ASG.

1.3.2 Linguagens Independentes

O padrão SQL tão aceito e utilizado para todos os bancos de dados relacionais ainda não encontra correspondente no universo NoSQL. A maioria dos sistemas NoSQL foi desenvolvida independentemente um do outro não existindo, deste modo, um padrão de acesso.

Com a popularização do uso dos NoSQL, logo surgiram movimentos para uma linguagem de consulta padronizada capaz de extrair dados de todos os tipos de banco de dados NoSQL. A Figura “Mapeamento de instruções entre linguagens” mostra algumas iniciativas de padronização para linguagens NoSQL com seus respectivos comandos.

Language	*SQL*	*SPARQL*	*Cypher*	*CQL*	*UnQL*	*Mongo QL*	*REDIS QL*
Model	Relational	Graph	Graph	Column family	Document	Document	Key–value
Insert data	*INSERT*	*INSERT*	*SET*	*INSERT*	*INSERT*	*insertOne()* *insertMany()*	*SET*
Update data	*UPDATE*	*MODIFY*	*SET* and *FOREACH*	*UPDATE*	*UPDATE*	*updateOne()* *updateMany()*	*SET*
Delete data	*DELETE*	*DELETE*	*REMOVE*	*DELETE*	*DELETE*	*deleteOne()* *deleteMany()*	*DEL*
Select data	*SELECT*	*SELECT*	*RETURN*	*SELECT*	*SELECT*	*find()*	*GET*
Source of data	*FROM*	*(FROM)* *URI*	*MATCH*	*FROM*	*FROM*	.collection	/
Filtering	*WHERE*	*WHERE* */ FILTER*	*WHERE*	*WHERE* (index column)	*WHERE*	*{field1:value1}*	/
Join operation	*JOIN*	Using pattern	*MATCH* (approximately)	/	/	*$lookup and “.”* *(for nested)*	/

Figura 1.2 – Mapeamento de instruções entre linguagens
Fonte: Bjeladinovic et al. (2020)

SPARQL (SPARQL Protocol and RDF Query Language) é um padrão W3C utilizado para representar e manipular dados semânticos construídos no modelo de grafos RDF (Resource Description Framework). Cypher linguagem associada a grafos com seus componentes: nós, arestas, caminhos. CQL (Cassandra Query Language) é propositalmente semelhante ao SQL usado em bancos de dados relacionais como MySQL e PostgreSQL. Muitas consultas são muito semelhantes entre as duas. De

fato, muitas coisas básicas são exatamente iguais. UnQL (Unstructured Query Language) é uma linguagem de consulta não estruturada para JSON, bases de dados semiestruturadas e de documentos. MongoQL(Mongo Query Language) e RedisQL (Redis Query Language) são as linguagens associadas ao MongoDB e Redis, respectivamente. Algumas tecnologias adequaram seus comandos para sintaxes mais próximas do SQL para facilitar seu uso entre os desenvolvedores.

O NewSQL surge a partir da necessidade de ter a consistência dos dados e de poder escalonar mais facilmente o sistema. O termo NewSQL foi utilizado pelo analista Matt Aslett, para descrever um novo grupo de bancos de dados, em que o NewSQL é uma classe de sistemas de gerenciamento de bancos de dados relacionais, que procura oferecer o mesmo desempenho e escalabilidade do modelo NoSQL, para cargas de trabalho de leitura e gravação no processamento de transações on-line, mantendo as garantias ACID do modelo relacional e o padrão SQL como linguagem de manipulação. NewSQL é essencialmente um rótulo para um conjunto altamente variado de ofertas de banco de dados que preenche a lacuna entre os bancos de dados SGDBRs convencionais e NoSQL. Plataformas NewSQL mais conhecidas incluem Google Spanner, Clustrix, VoltDB, MemSQL, GemFire XD, NuoDB e Trafodion da Pivotal.

Não raro, engenheiros de dados propõem uma camada de integração entre os diferentes bancos de dados. Uma das primeiras surgidas é o Akzaban proposta pelo LinkedIn. Outro exemplo é a ferramenta Dremio que fica entre diversas fontes de dados, do MongoDB ao Oracle, do CSV ao Parquet e as mais diversas ferramentas de visualização como Tableau, Power BI e até Python com suas inúmeras bibliotecas, via ODBC. A Paypal processa terabytes de dados todos os dias, com mais de mil nós utilizando este mix de tecnologias.

1.3.3 Funcionalidade Determina Escolha

Encontrar a solução para um problema é fácil, encontrar a melhor solução para um problema é que é difícil. O psicólogo humanista Abrahan Maslow disse, certa vez, em citação livre, que “se a única ferramenta que você possui é um martelo, todo problema parecer-se-á um prego”.

Muito mais do que permitir recuperações de dados baseadas em projeções e uniões presentes nos modelos relacionais, nos NoSQL buscamos recuperar dados de acordo com seu significado e operações inerentes a ele. Assim, se estivermos tratando com dados sobre uma localização geográfica como longitude e latitude, vamos querer operações georreferenciadas como cálculo de distâncias, pontos dentro de um polígono ou de um raio, entre outras. Da mesma forma, se estivermos tratando com nós e arestas vamos querer saber o menor caminho entre dois nós ou com maior número de conexões no grafo.

1.3.4 Gerenciamento De Transações

Uma transação é uma coleção de operações que desempenha uma função lógica única dentro de uma aplicação do sistema de banco de dados. Ou todas operações são executas com sucesso ou nenhuma delas é considerada.

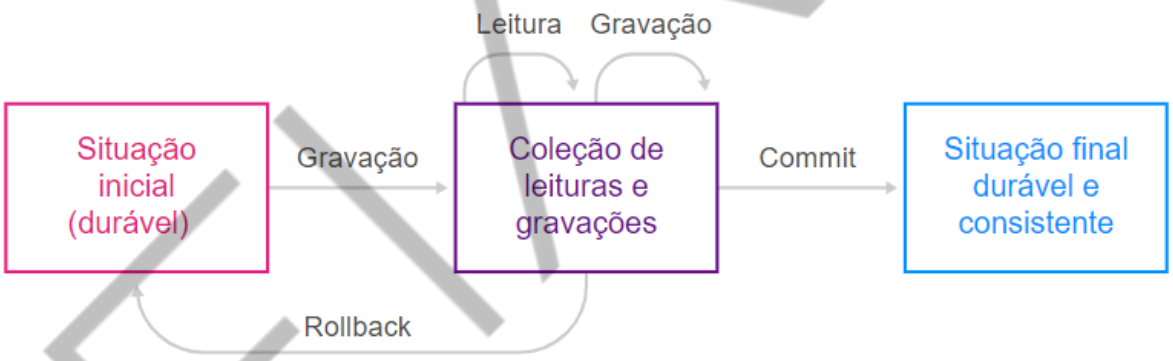

Figura 1.3 – Transação
Fonte: Elaborada pelo autor (2020)

O controle de transações é um aspecto importante a considerar, quando se pensa no desempenho e consistência de bases de dados implementadas num ambiente de computação distribuída (MCCREARY; KELLY, 2014).

Os SGDBRs garantem as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Todavia, com o surgimento dos NoSQL, ocorreu uma mudança no paradigma de controle de transações passando para BASE (Basically Available, Soft-State, Eventually Consistent) com o teorema CAP. As características das transações ACID são:

- **Atomicidade:** ou uma transação operacional ocorre com sucesso ou falha. Se uma parte da transação falha, toda a transação falha; só se todos os comandos da transação forem bem-sucedidos é que a transação é considerada bem-sucedida. Se algum comando da transação falhar, todas as alterações efetuadas até aquele ponto devem ser revertidas e a base de dados tem de voltar ao estado em que se encontrava antes de iniciar a transação.
- **Consistência:** assegura que os dados inseridos numa transação têm de respeitar restrições impostas pelo esquema da base de dados, tipos de dados e integridade referencial de tabelas ou linhas.
- **Isolamento:** quando os mesmos dados são acessados por duas transações concorrentes, cada transação tem de ser executada em total isolamento. Dependendo do nível de isolamento imposto, isso pode significar que o gerenciador de banco de dados pode necessitar bloquear temporariamente a execução de uma transação até que outra tenha terminado.
- **Durabilidade:** assegura que, depois de uma transação ser executada com sucesso, o seu resultado será representado no banco de dados, mesmo na eventualidade de erros ou falhas de sistema. Isso implica em mecanismos do gerenciador de banco de dados para garantir sua consistência como os Logs.

O teorema CAP tem três pilares:

- **Consistência:** diz respeito à atomicidade e isolamento. De uma forma simples, isso implica que todos os processos executados de forma concorrente visualizem a mesma versão dos dados.
- **Disponibilidade (Availability):** significa que o sistema está disponível quando é solicitado. No contexto de um servidor web, refere que cada pedido eventualmente receberá uma resposta.
- **Tolerância a partição (Partition tolerance):** implica que o sistema seja capaz de funcionar corretamente, mesmo no caso de falha por parte de algum dos componentes.

O teorema CAP (MCCREARY; KELLY, 2014) indica que qualquer sistema que suporte bases de dados distribuídas, apenas consegue em qualquer momento responder a dois destes pilares simultaneamente. Existem então apenas 3 combinações possíveis: CP, CA e AP.

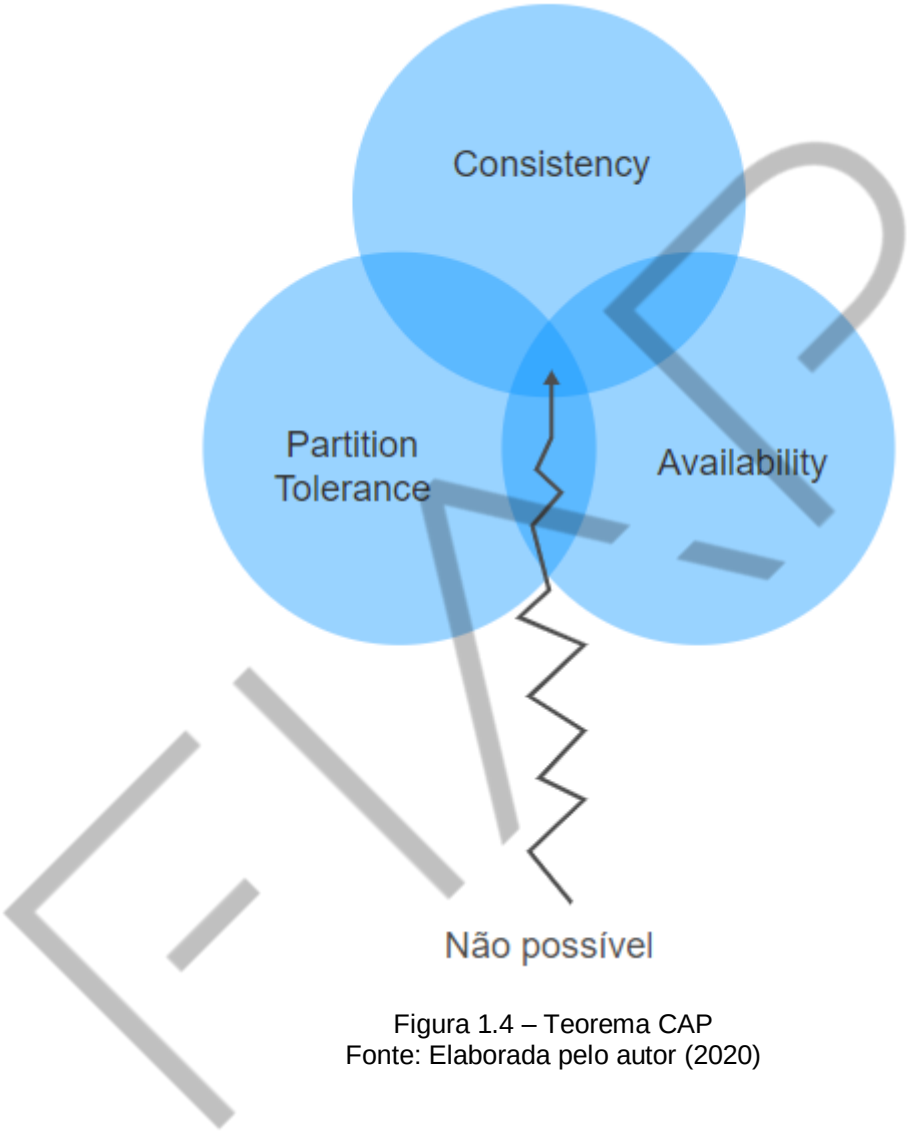

Figura 1.4 – Teorema CAP
Fonte: Elaborada pelo autor (2020)

Um exemplo simples:

Linha X é replicada nos nós M e N

Cliente A grava linha X no nó N

Após um tempo t

Cliente B lê a linha X do nó M

O cliente B vê a linha gravada pelo cliente A?

Para gerenciadores NoSQL, a resposta pode ser: talvez

Consistência e tolerância à partição, comprometendo a disponibilidade (CP). Na situação em que um nó falha, o sistema pode ficar indisponível por muito tempo, até que o sistema seja restaurado a um estado consistente. Este é um sistema típico em que transações monetárias e o tempo têm um papel importante. Em muitas situações, os sistemas conseguem recuperar e replicar rapidamente a informação, levando a que o sistema esteja indisponível por um curto espaço de tempo. Na maioria dos casos, uma pequena quebra na disponibilidade não é catastrófica e revela-se uma opção viável.

Consistência e disponibilidade, comprometendo a tolerância à partição (CA). Parte da base de dados não apresenta a preocupação de ser tolerante à falha e recorre à técnica de replicação para garantir a disponibilidade e consistência da informação. Normalmente, esta situação é encontrada nas tradicionais bases de dados relacionais.

Tolerância à partição e disponibilidade, comprometendo a consistência (AP). Em algumas situações, a disponibilidade não pode ser comprometida e o sistema é tão largamente distribuído que também não é possível comprometer a tolerância à partição. Nestes sistemas, no entanto, é possível abdicar de uma forte consistência. O termo fraca consistência pode variar entre nenhuma consistência e uma eventual consistência. Uma eventual consistência significa que, após a atualização de um atributo, “eventualmente” todos os nós do sistema sincronizarão essa informação. Esta é uma das bases para o modelo BASE (Basically Available, Soft-state, Eventually consistente):

Basicamente disponível (Basically Available) significa que o sistema garante a disponibilidade dos dados de acordo com as propriedades do teorema CAP. No entanto, a resposta obtida pode ser “Falha” ou “Inconsistência”, se os dados requeridos se encontrarem inconsistentes ou num estado de mudança.

Estado flexível (soft state) significa que os dados acedidos podem estar incorretos ou imprecisos durante um período de tempo variável, e que esses dados podem mudar ao mesmo tempo em que são utilizados. Mesmo após grandes períodos de tempo sem novas inserções de dados, não está garantida a sua total consistência, isto deve-se às implicações relativas ao ponto da “eventual consistência”.

Eventual consistência (Eventual consistency) significa que o sistema irá, eventualmente, tornar-se consistente e é um dos modelos de consistência utilizados no âmbito da programação paralela. Este modelo indica que, após um longo período de tempo sem alterações, todas as atualizações efetuadas sobre os dados de um nó do sistema se propagarão a todos os outros pontos do sistema, e que todas as réplicas da base de dados, eventualmente, atingirão um estado de consistência. Ao contrário dos sistemas ACID, cujo foco está na consistência, os sistemas BASE têm seu foco na disponibilidade.

As propriedades BASE garantem que novos dados chegam a cada momento e que eles necessitam de armazenamento imediato, mesmo que isso implique o risco de o sistema ficar dessincronizado por um breve período de tempo (MCCREARY; KELLY, 2014). Estes sistemas “relaxam” as regras para assim possibilitarem que queries sejam executadas, mesmo que nem todas as partes da base de dados em causa se encontrem devidamente sincronizadas.

Os sistemas BASE são normalmente mais rápidos e apresentam uma estrutura mais simples; não há a necessidade de escrever código relativo a bloqueios e desbloqueios de recursos. O objetivo a atingir com esta filosofia passa por manter todos os processos em movimento e tratar das partes incompletas ou que falham, num momento posterior.

ACID	BASE
Consistência forte	Fraca consistência
Isolamento	Foco em disponibilidade
Concentra-se em **“commit”**	Melhor esforço
Disponibilidade	Mais simples e mais rápido
Conservador (pessimista)	Agressivo (otimista)
Evolução difícil	Evolução mais fácil

Quadro 1.1 – Comparação ACID x BASE
Fonte: Elaborado pelo autor (2020)

1.4 Categorias NoSQL

Db-Engines é um site organizador dos diferentes bancos de dados - 358 sistemas diferentes de gerenciamento de banco de dados - e realiza um ranking das categorias mais utilizadas. Segundo esta classificação temos os tradicionais bancos de dados relacionais (141), search engines (21), time series (34), orientado a documentos (48), grafos (32), chave-valor (65), entre outros.

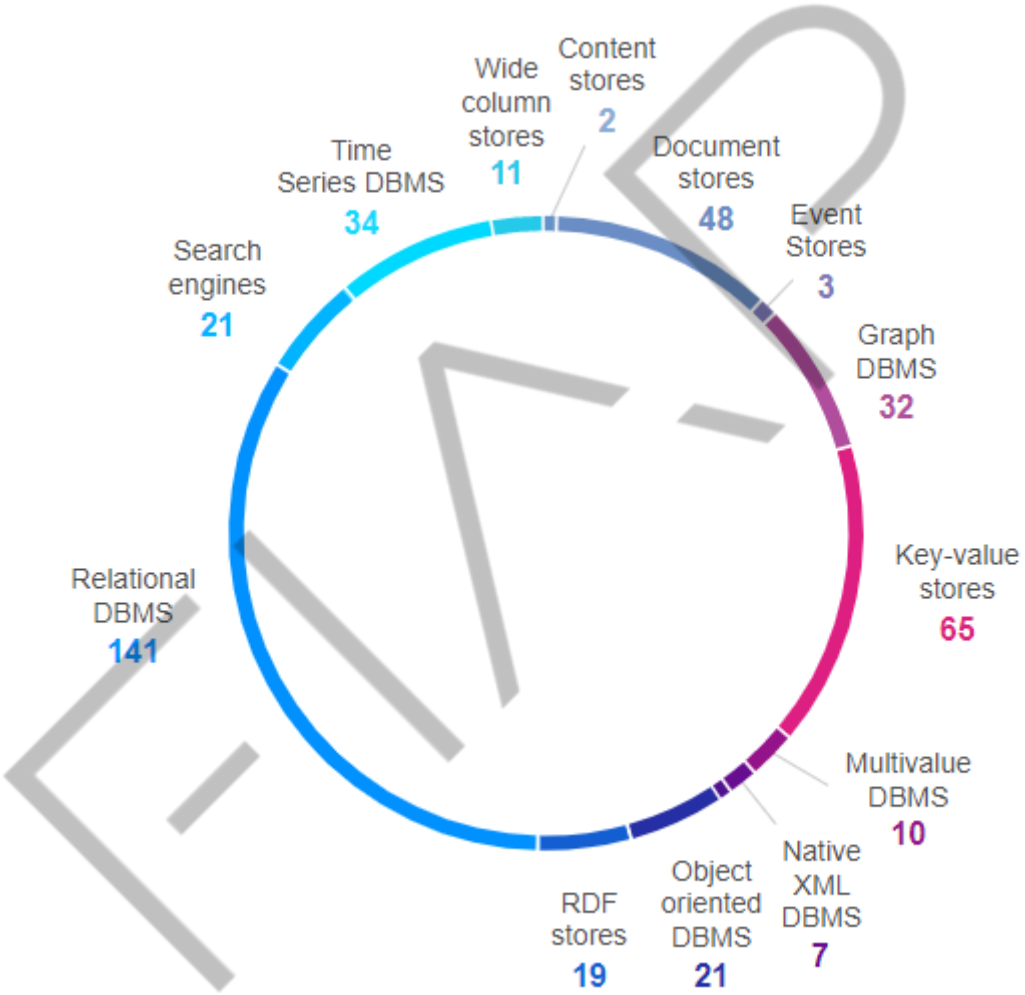

Figura 1.5 – Número de tecnologias por categoria
Fonte: Db-Engines (2020)

Quando uma tecnologia implementa mais de um modelo é dita multimodel.

1.4.1 Bancos De Dados Chave-Valor

Os bancos de chave-valor armazenam dados em uma tabela hash simples. É utilizada quando o acesso ao banco de dados é feito, principalmente, por meio de uma chave primária. O usuário pode efetuar buscas, inserir, ou, até mesmo, apagar um valor de uma chave em um depósito de dados chave-valor. O valor é armazenado pelo banco de dados sem preocupação com o que ele representa, é a aplicação que faz o tratamento e se preocupa com o entendimento do valor.

Os acessos aos depósitos chave-valor são feitos, sempre, pela chave primária, por conseguinte, têm ótimo desempenho e podem ser facilmente escaláveis. É importante destacar que, entre suas limitações, a consulta só pode ser feita pela chave, que retorna o valor; o valor não pode ser consultado pelo atributo (SADALAGE, 2013).

Chave	Valor
Imagem123.jpg	Arquivo binário contendo a imagem
www.fiap.com.br	HTML de uma página web
C:/Documentos/LivroKDD.pdf	Documento PDF

Quadro 1.2 – Modelo Chave-valor
Fonte: Elaborado pelo autor (2019)

Exemplos de bancos chave-valor:

- Amazon DynamoDB.
- Microsoft Azure Cosmos DB.
- Aerospike.
- Redis.
- ArangoDB.
- Memcached.

Os bancos de dados chave-valor são indicados para:

- Armazenamento de informações de sessão.
- Perfil de usuário e preferências.
- Dados de carrinho de compra.

Não são indicados para:

- Relacionamento entre dados.
- Transações com múltiplas operações.
- Consultas por dados e atributos.
- Operações por conjuntos.

1.4.2 Bancos de Dados Colunares

Os bancos de dados colunares armazenam dados em famílias de colunas. Essas famílias de colunas agem como linhas com muitas colunas associadas e que são acessadas por uma chave de linha. De maneira geral, famílias de colunas são grupos de dados relacionados que, frequentemente, são acessados juntos. Os dados são indexados por uma tripla, composta por linha, coluna e timestamp. Uma das vantagens de usar uma tripla é que podemos identificar diferentes versões do mesmo dado (SADALAGE, 2013).

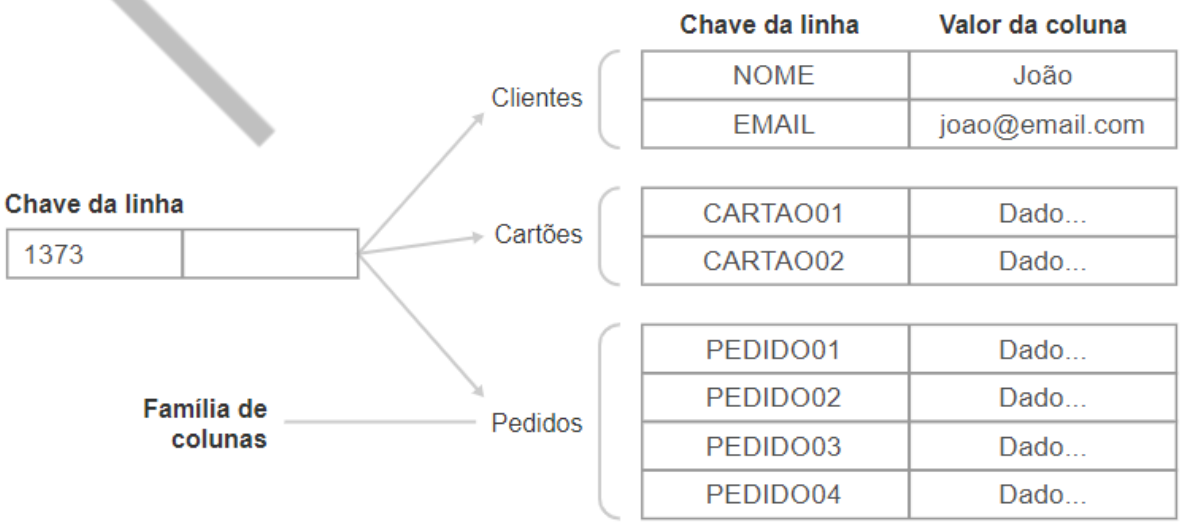

Figura 1.6 – Modelo Colunar
Fonte: Adaptado por FIAP (2020)

Exemplos de bancos colunares:

- Cassandra.
- HBase.
- Hypertable.
- Google Cloud Bigtable.
- ScyllaDB.
- Datastax Enterprise.

Os bancos de dados colunares são indicados para:

- Registro de eventos (log).
- Sistema de Gerenciamento de Conteúdo (CMS).
- Contadores.

Não são indicados para:

- Sistemas que requeiram ACID para leituras e gravações.

1.4.3 Bancos De Dados Orientado A Grafos

Os bancos de dados orientados a grafos são baseados na teoria dos grafos, permitem armazenar nós e arestas entre esses nós. Neste modelo, o banco pode ser comparado com um multigrafo rotulado e direcionado, no qual cada nó pode ser conectado por mais de uma aresta. Possui três componentes básicos: os nós (são os vértices do grafo), os relacionamentos (são as arestas) e as propriedades (ou atributos) dos nós e dos relacionamentos. (SADALAGE, 2013).

Exemplos de bancos orientados a grafos:

- Neo4J.

- Amazon Neptune.
- Stardog.
- Giraph.

Os bancos de dados orientados a grafos são indicados para:

- Dados conectados.
- Roteamentos, envio e serviços baseados em localização.
- Sistemas de recomendação.

Não são indicados para:

- Sistemas com atualização em lote (nos quais várias entidades são atualizadas em uma operação).

1.4.4 Bancos De Dados Orientados A Documentos

Os bancos de dados orientados a documentos armazenam e recuperam documentos. Esses documentos podem ser XML, JSON ou BSON, entre outros. Os documentos, nesse caso, são objetos com um código único e um conjunto de campos, que podem ser strings, listas, dados escalares, coleções ou documentos aninhados. (SADALAGE, 2013).

Exemplos de bancos orientados a documentos:

- MongoDB.
- CouchDB.
- Azure CosmosDB.
- Couchbase.
- Firebase Realtime Database.

Os bancos de dados orientados a documentos são indicados para:

- Registro de eventos (log).
- Sistema de Gerenciamento de Conteúdo (CMS).
- Análise web ou em tempo real (*analytics*).
- Aplicativos de comércio eletrônico.

Não são indicados para:

- Transações complexas com diferentes operações.
- Consultas em estruturas de agregados variáveis.
- Atualizações em uma operação.

1.4.5 Search Engines

Um banco de dados search engines (mecanismo de pesquisa) pode ser utilizado para indexar grandes volumes de dados e fornecer acesso a esses índices quase em tempo real.

Você envia blobs de documentos semiestruturados para eles, mas em vez de armazená-los como estão e usando os analisadores XML ou JSON para extrair informações, o mecanismo de pesquisa divide o conteúdo do documento em um novo formato otimizado para pesquisa com base em substrings de campos do formato texto.

Exemplos de bancos search engine:

- Elasticsearch.
- Splunk.
- Solr.
- Microsoft Azure Search.
- Amazon CloudSearch.

Os bancos de dados search engine são indicados para:

- Filtragem e classificação rápida.
- Facetamento instantâneo.
- Buscas em tempo real e auto complete.
- Tratamento de logs.

Não são indicados para:

- Como única solução de armazenamento.

1.4.6 Time Series

Os dados da série temporal são um conjunto de valores organizados no tempo e um banco de dados de séries temporais é um banco de dados otimizado para esse tipo de dados. Os bancos de dados de série temporal devem fornecer suporte para um número alto de gravações, pois geralmente coletam grandes quantidades de dados em tempo real a partir de diversas fontes de dados. As atualizações são raras e as exclusões geralmente são feitas como operações em massa. Embora os registros gravados em um banco de dados de série temporal sejam geralmente pequenos, muitas vezes há um grande número de registros e o tamanho total dos dados pode crescer rapidamente.

Exemplos de bancos de séries temporais:

- InfluxDB.
- Prometheus.
- Graphite.
- Druid.
- Amazon Timestream.

Os bancos de dados de séries temporais são indicados para:

- Telemetria.
- Sensores IoT.
- Contadores

Não são indicados para:

- Transações complexas com diferentes operações.
- Atualizações.

1.5 Soluções Poliglotas

Neste contexto de banco de dados relacionais e NoSQL surgem soluções poliglotas, em que mais do que um banco de dados compõe a solução de acordo com suas funcionalidades, como para plataforma e-commerce: para armazenar dados da sessão e do carrinho pode ser um chave-valor, para acessar todos os dados do pedido pode ser um orientado a documento, para garantir as transações de atualização de preço e estoque pode ser os banco relacionais e para recomendar produtos para os clientes pode ser o de grafos.

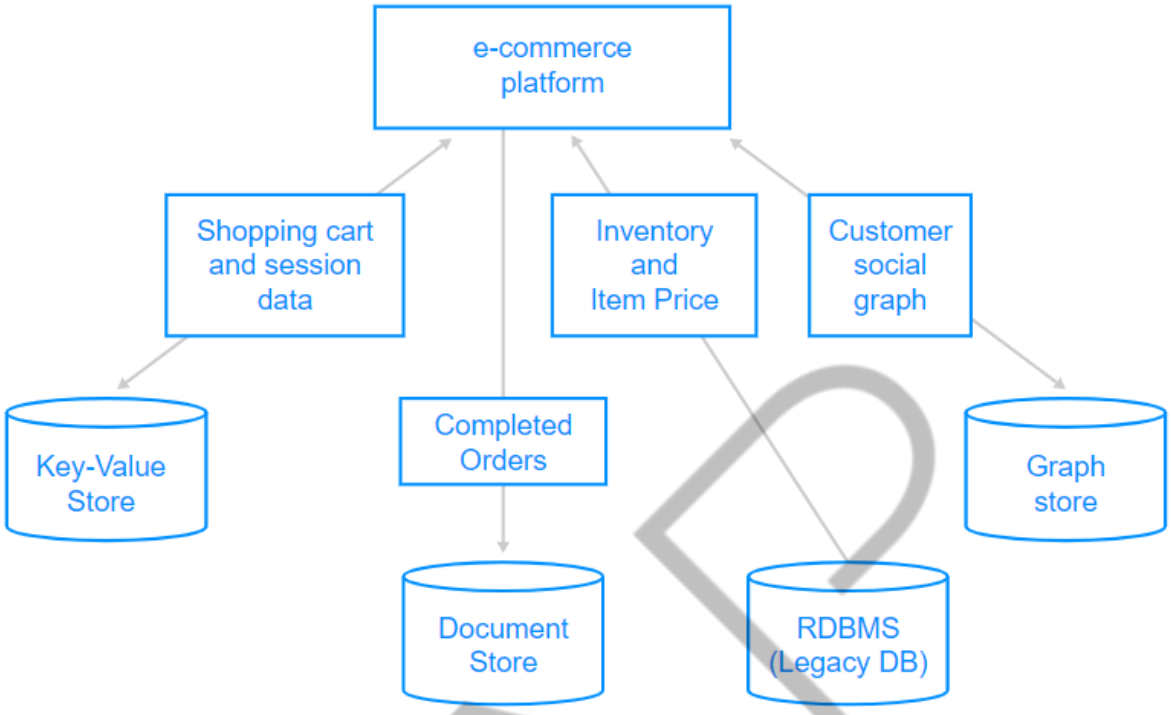

Figura 1.7 – Persistência Poliglota
Fonte: Sadalage e Fowler (2013)

Em 2020 os bancos de dados mais populares segundo Db-Engines por categoria são os relacionais, orientado a documentos, chave valor e search engines.

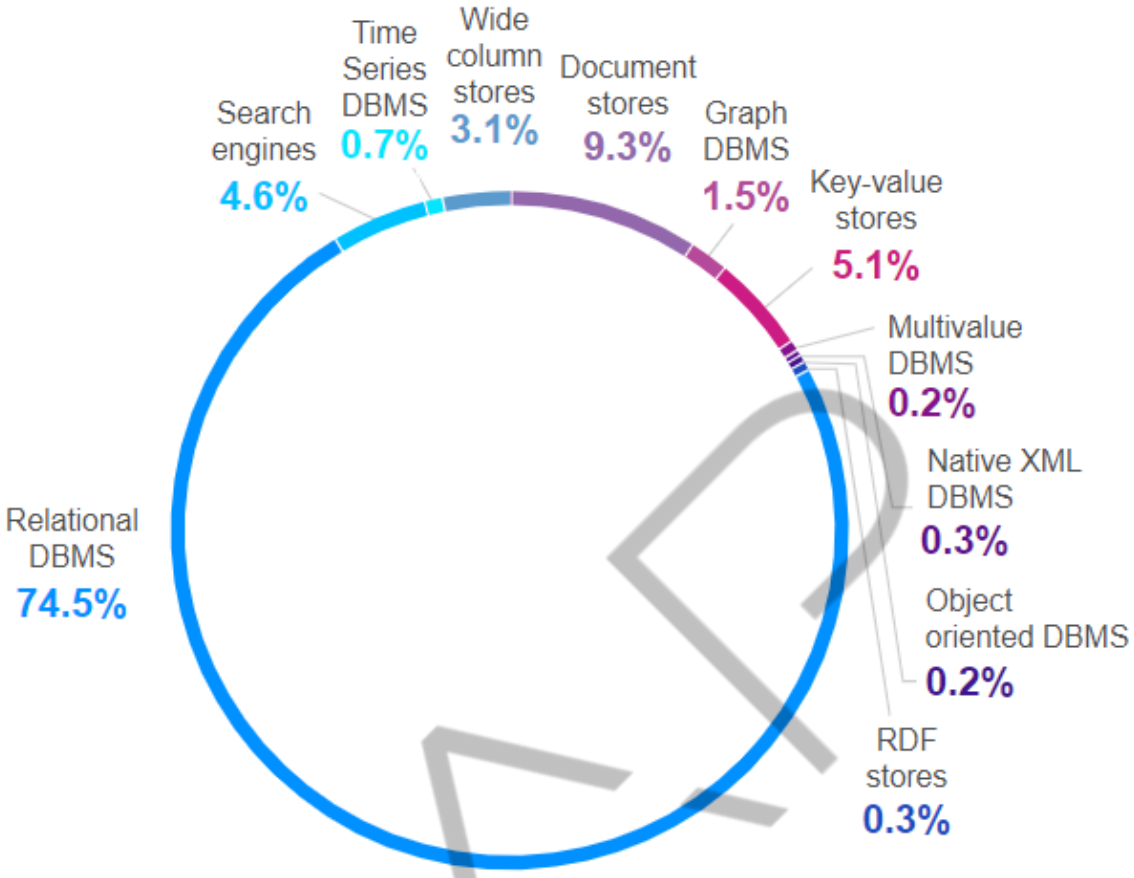

Figura 1.8 – Banco de Dados mais populares
Fonte: DB-Engines (2020)

O relatório Forrester Wave™: Big Data NoSQL, do primeiro trimestre de 2019, avaliou 15 fornecedores com base em 26 critérios, incluindo consistência, desempenho e escalabilidade de dados, multimodelo e segurança de dados.

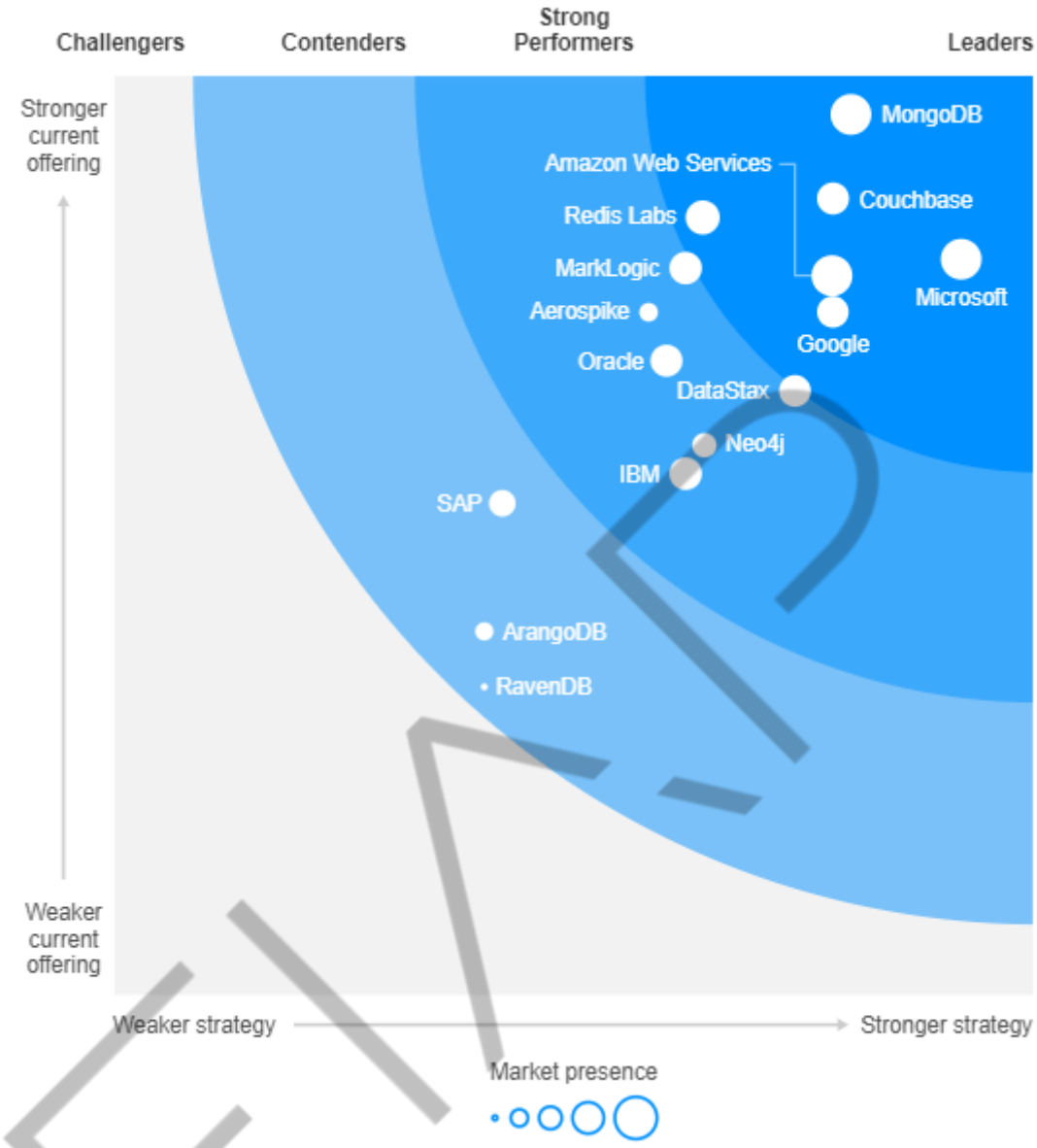

Figura 1.9 – Forrester Wave ™: Big Data NoSQL
Fonte: Adaptado por FIAP (2020)

1.6 Critérios Para Escolha

Figura 1.10 – Critérios para escolha SQL e NoSQL
Fonte: Adaptado por FIAP (2020)

Os critérios para escolha dos bancos de dados SQL e NoSQL são projetados para atender a necessidades muito diferentes. Gessert (2017) criou uma árvore de decisão para mapear os requisitos para os bancos de dados NoSQL.

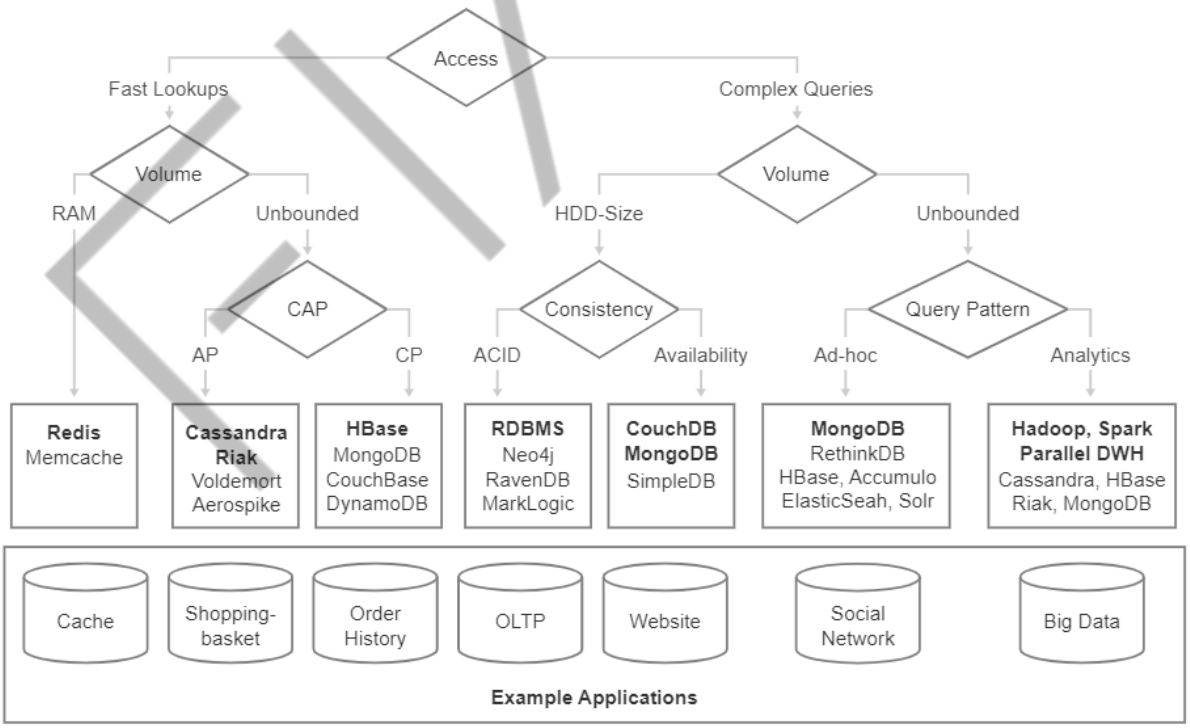

Figura 1.11 – Árvore de decisão para requisitos NoSQL
Fonte: Gessert (2017)

Os aplicativos/projetos podem variar bastante, dependendo de cada organização, portanto a transição dependerá do seu caso de uso. Aqui estão algumas diretrizes gerais para a transição:

Entender os principais requisitos para sua aplicação:

- Desenvolvimento rápido de aplicativos: mudanças nas necessidades do mercado e modificação contínua dos dados.
- Escalabilidade.
- Desempenho constante: baixo tempo de resposta para uma melhor experiência do usuário.
- Confiabilidade operacional: alta disponibilidade para gerenciar erros com impacto mínimo no aplicativo e APIs de monitoramento integradas para melhor manutenção.

Entender os diferentes tipos de ofertas NoSQL para cada modelo de dados:

- Por exemplo, os bancos de dados NoSQL orientados a documentos - como Couchbase e MongoDB, são os dois exemplos mais conhecidos e mais amplamente adotados. Além disso, o Cassandra pode ser uma solução que você pode usar para análise de dados, devido ao seu modelo colunar. O Neo4j, um banco de dados de gráficos, pode ser o banco de dados perfeito para aplicativos que precisam armazenar relacionamentos entre entidades.

Criar um protótipo:

- Depois de restringir as opções possíveis para o tipo de banco de dados, tente desenvolver um protótipo integrando as principais características do seu aplicativo. Este protótipo o ajudará a avaliar os tempos de resposta, o desempenho em termos de taxa de transferência e a capacidade de escalar facilmente.

Modelar e desenvolver documentos e/ou grafos:

- Para bancos de dados orientados a documentos e/ou grafos, gaste algum tempo modelando seus dados a partir de diagramas para chegar a modelos de dados flexíveis.

Implantação e produção:

- A estabilidade operacional é um aspecto muito importante para aplicativos da web interativos. Teste mais de uma vez a sua implantação, como faria em aplicativos que normalmente usam sistemas RDBMS tradicionais.

Acompanhar as últimas tendências:

- A melhor maneira de garantir uma implementação bem-sucedida do NoSQL é estar atualizado nas versões mais recentes da tecnologia.

A escolha de um sistema de gerenciamento de banco de dados (DBMS) é, portanto, um momento importante e estruturante para qualquer projeto de dados. Obviamente, é sempre possível escolher uma opção e depois mudar para outra mais tarde. Mas um pouco de análise conceitual e pensamento no início de um projeto farão com que você economize tempo e dinheiro.

O mercado hoje está cheio de bancos de dados NoSQL. Somos praticamente confrontados com dois ou três deles todos os dias, porque há muitas vantagens para um desenvolvedor mudar para o NoSQL. Um modelo de dados mais flexível e livre de esquemas rígidos é uma grande vantagem.

Mas a maioria dos produtos NoSQL ainda está nos estágios iniciais do ciclo do produto. Para recursos como junções complexas, os desenvolvedores podem preferir usar um RDBMS tradicional. E para alguns projetos, uma abordagem híbrida pode ser a melhor escolha.

Para concluir, cada empresa terá suas próprias preferências, dependendo dos requisitos do projeto. Portanto, determine suas necessidades e o banco de dados que criteriosamente fornece suporte integrado para o desenvolvimento do seu projeto.

REFERÊNCIAS

BINANI, S.; GUTTI, A.; UPADHYAY, S. SQL vs. NoSQL vs. NewSQL - A Comparative Study. **Communications On Applied Electronics**. v. 6, n. 1, out. 2016, p. 43-46. Disponível em: < https://pdfs.semanticscholar.org/9d92/d600ff1a4bec346b6ca3fe6d8bf9677294b2.pdf?_ga=2.198626991.128243254.1601058557-1027294064.1601058557>. Acesso em: 25 set. 2019.

BJELADINOVIC, S.; MARJANOVIC, Z.; BABAROGIC, S. A proposal of architecture for integration and uniform use of hybrid SQL/NoSQL database components. **Journal of Systems and Software**, v. 168, 2020.

BUCKLER, C. **SQL vs NoSQL: The Differences**. 2015. Disponível em: <https://www.sitepoint.com/sql-vs-nosql-differences/>. Acesso em: 25 set. 2020.

EVANS, E. **NoSQL: What's in a name?** 2009. Disponível em: http://blog.sym-link.com/posts/2009/30/nosql_whats_in_a_name/ >. Acesso em: 25 set. 2020.

FORRESTER. **The Forrester Wave™: Big Data NoSQL, Q1 2019**. Disponível em: https://www.forrester.com/report/The+Forrester+Wave+Big+Data+NoSQL+Q1+2019/-/E-RES136481 Acesso em: 04 out 2020.

GEORGIEV, G. **What is Vertical scaling and Horizontal scaling – Vertical and Horizontal hardware / services scaling**. 2014. Disponível em: <http://www.pc-freak.net/blog/vertical-horizontal-server-services-scaling-vertical-horizontal-hardware-scaling/>. Acesso em: 25 set. 2020.

GESSERTY, F. **NoSQL Databases: a Survey and Decision Guidance.** 2017. Disponível em: <https://medium.baqend.com/nosql-databases-a-survey-and-decision-guidance-ea7823a822d>. Acesso em: 25 set. 2020.

GROLINGER, K. et al. Data management in cloud environments: NoSQL and NewSQL data stores. **Journal Of Cloud Computing: Advances, Systems And Applications,** v. 2, p. 22, 2013. Disponível em: <https://link.springer.com/content/pdf/10.1186%2F2192-113X-2-22.pdf>. Acesso em: 20 jul. 2020.

HOWS, D., PLUGGE, E., MEMBREY, P., AND HAWKINS, T. The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB, 3rd ed. Apress, 2015.

HOWS, D.; MEMBREY, P., PLUGGE, E. **Introdução ao MongoDB**. São Paulo:Novatec, 2015. 167 p.

MCCREARY, D.; KELLY, A. **Making Sense of NoSQL:** A Guide for Managers and the Rest of Us. 1. ed. Nova York: Manning Publications, 2014.

SADALAGE, P. J.; FOWLER, M.; RIVER, U. S. **NoSQL distilled:** a brief guide to the emerging world of polyglot persistence. São Paulo: Pearson, 2013.

SADALAGE, P. J.; FOWLER, M. **NoSQL Essencial.** Brasil: Novatec, 2013.

STROZZI, C. **NoSQL Strozzi**. [s.d] Disponível em: <http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/Home%20Page>. Acesso em: 25 set. 2020.

YEGULALP, S. **What is NoSQL?** NoSQL databases explained. 2017. Disponível em: <https://www.infoworld.com/article/3240644/what-is-nosql-databases-for-a-cloud-scale-future.html>. Acesso em: 20 jul. 2020.