Repositório Github
Apresentação

(Breve história dos Bancos de Dados Relacionais)

O advento dos computadores na década de 50 trouxe consigo, uma nova problemática: como é possível armazenar os dados computados por aquelas máquinas? Foi justamente esse questionamento que deu início ao que conhecemos como Bancos de Dados.

(Cartões Perfurados)

Na década de 50, uma das primeiras formas de armazenamento fora utilizada, os cartões perfurados. Cada um deles continha uma série de furos que representavam dados binários, os quais poderiam ser “lidos” e interpretados pelas máquinas. Seu sucesso se deu, principalmente, por serem instrumentos fáceis de se manipular e produzir, além de proporcionar uma forma de entrada e saída de dados. Entretanto, houveram alguns desafios a serem vencidos, como o caso de se lidar com um volume maior de informações. Nesses casos, os cartões não performaram tão bem, uma vez que exigia uma organização muito maior, e nenhuma utilidade para o acesso de dados de forma sequencial.

(Fitas Magnéticas)

Esses fatores foram vencidos com o uso das fitas magnéticas. Elas detinham essa vantagem de comportar uma grande quantidade de dados, entretanto, sua gravação era feita de forma sequencial. Isso implicava que para acessar um dado específico, era necessário percorrê-la toda até que o fosse encontrado, e assim, sua performance era extremamente baixa em questão de tempo. A título de exemplo, o UNIVAC 1, ou um dos primeiros computadores comerciais da história, criado em 1951, utilizava desta ferramenta para acesso a dados.

(Modelo Hierárquico)

Na década de 60, havia a utilização de duas modalidades principais de armazenamento de dados: o Modelo Hierárquico e o Modelo de Rede.

O Modelo Hierárquico foi um dos primeiros modelos de bancos de dados, no qual sua principal característica foi o armazenamento em formato de árvore. Cada registro tem um único pai, e cada pai pode ter vários registros. Esse era o princípio da relação 1 para muitos, que atualmente, estudamos nesta disciplina. Todos eles podem ser acessados mediante ao acesso pelo nó raiz, ou *root*. Em função desta especificidade, este modelo trouxe uma melhora considerável em relação à performance, uma vez que, conforme aprendemos na disciplina de Algoritmos e Estruturas de Dados, o algoritmo de árvore é excelente para acesso. Além disso, podia ser utilizado através do SGBD IBM IMS, aquele que foi utilizado pelo programa Apollo para levar a humanidade à lua, pela primeira vez.

(Modelo de Rede)

Além dele, era também utilizado o Modelo de Rede, que por sua vez tinha a proposta de oferecer relações mais complexas, nas quais um pai pode ter vários filhos, e os filhos, vários pais. Esse era o antecessor ideológico das relações muitos para muitos, que novamente, vemos nesta disciplina. Por causa de sua característica, sua estrutura era vista como uma variedade de grafos. Ou seja, enquanto o foco do primeiro era performance, o deste era flexibilidade.

(Banco de Dados Relacional)

Entretanto, tudo havia de mudar na década de 70, quando, em 1971, foi postulado o conceito de Banco de Dados Relacional, através do pesquisador Edgar F. Codd em seu artigo seminal “A Relational Model of Data for Large Shared Data Banks”. Três anos depois, Chamberlin e Boyce, também da IBM, desenvolveram uma linguagem de consulta chamada SEQUEL, voltada para a implementação do Banco de Dados postulado por Codd. Em 1979, a Relational Software Inc, mais tarde Oracle, desenvolveu o Oracle v2, aquele que seria o primeiro SGBD comercial a utilizar SQL.

(Padronização do SQL)

Na década de 80, houveram padronizações da linguagem SQL, como o SQL-86, e 87, bem como o reconhecimento da linguagem como um padrão internacional, pela Organização Internacional de Padronização.

(Avanços nas décadas de 90 e 2000)

Na década de 90 e 2000, houve o lançamento de versões mais avançadas da linguagem, que introduziram novos recursos, como triggers, tipos de dados estendidos e consultas recursivas. Além disso, houve o advento de estruturas como *big data*, *data warehousing* e *olap*.

(Popularização da Internet e NoSQL)

Entretanto, nos anos 2000, houve um grande aumento na popularização do acesso à internet, e com isso, os sistemas web necessitaram de uma evolução para atender a esta crescente demanda. Um grande exemplo disso era, na época, a demora para acesso a dados em websites congestionados. Assim, necessitava-se de sistemas de bancos de dados que pudessem lidar com demandas de forma flexível e confiável, e dessa forma, nasceram os Bancos de Dados Não Relacionais, NoSQL.

(Definição e Características dos NoSQL)

A sigla NoSQL é uma sigla que é erroneamente compreendida, uma vez que se subentende que não é utilizado SQL, por causa do “No”. Entretanto, “No” significa *Not Only*, ou seja, “Não Somente SQL”.

Dentro do mundo da gestão de dados, este possui várias qualidades que serão abordadas posteriormente.

O design dos NoSQL é voltado para um importante conceito chamado escalabilidade horizontal. Para entender isso, tome a seguinte analogia: imagine que o banco de dados principal de uma grande empresa esteja hospedado em um único servidor, dentro de suas instalações. Durante as manhãs, ele consegue atender aos usuários visitantes de forma satisfatória, entretanto, entre meio-dia e quatro da tarde, enfrenta horários de pico, e por conseguinte, acessos mais demorados. Agora, considere que o banco de dados da empresa seja um NoSQL. Se assim o fosse, haveria a possibilidade de subdividir os dados em vários servidores, automaticamente determinados pelo gerenciador do banco de dados, com o intuito de estabelecer uma proporcionalidade de infraestrutura adequada, e diminuir tempo de obtenção de informações aos usuários.

Em segundo plano, podemos falar sobre eles serem altamente disponíveis e tolerantes a falhas. Considere o seguinte caso: ao utilizar um servidor com SQL, a disponibilidade é dependente de condições adequadas, como acesso suficiente à internet e disponibilidade de energia elétrica, dentre outros fatores como manutenção e etc. Entretanto, quando se utiliza de bancos de dados NoSQL, empresas específicas como Amazon, IBM, Microsoft, dentre outras, são as que ofertam tais servidores, que são localizados em *Data Centers* ao redor do mundo. Dessa forma, elas se tornam as responsáveis por ofertar todas as condições necessárias para que a infraestrutura funcione corretamente. Assim, supondo que seu banco de dados esteja nos domínios da AWS, no Rio de Janeiro (us-east-1-rio-1a), e por algum motivo, seja de perigo cibernético ou de infraestrutura, esteja indisponível, seu acesso seria redirecionado para outro mais próximo, podendo ser o data center do Chile ou da Argentina, ao invés de estar indisponível para 100% dos usuários, como ocorria em infraestruturas antigas de SQL, que consistiam em locais centralizados, que dependiam de times de Tecnologia da Informação para fazer a manutenção e desenvolvimento.

Por conseguinte, a união dos dois atributos anteriores refletem diretamente na forma de uma performance melhorada.

Quando falamos sobre a alta disponibilidade, é importante falar também sobre replicação. Dentro da área de DevOps atual, é possível fazer réplica de softwares em variados servidores através de ferramentas como Docker (para a criação de containers), e o Kubernetes (para a orquestração dos mesmos, de forma automática). Mediante a utilização destes softwares, é possível disponibilizá-los de forma operante nas mínimas unidades de funcionamento (containers), e aumentar a escalabilidade horizontal automaticamente através do Kubernetes, que será responsável por coordená-los.

Por fim, pode-se falar sobre sua fácil integração com frameworks de computação distribuída, como por exemplo o Apache Hadoob e Apache Spark, para o processamento de dados em larga escala e distribuída (*big data*).

Por fim, pode-se falar também sobre a sua flexibilidade. Eles são consideravelmente superiores aos SQL, uma vez que ao invés de seguirem um rígido modelo de tabelas, podem seguir outros modelos, como chave-valor, colunas compostas, entre outros, que serão vistos posteriormente. De uma forma geral, cada um oferece uma especificidade diferente, que pode ser escolhida sob medida para o propósito do projeto, desde aplicativos de troca de mensagens a armazenamento de dados de forma temporal, como interfaces web de valor de ações e câmbio

(Comparação com bancos de dados relacionais)

Dessa forma, nesse primeiro momento, vamos analisar os pontos fortes a serem ressaltados sobre os Bancos de Dados Relacionais:

- Integridade e Consistência de Dados
- Complexidade de consultas
- Escalabilidade vertical

No que tange à Integridade e Consistência de Dados, é consideravelmente melhor que se utilize os relacionais, uma vez que são rígidos em relação à escrita de seus dados, não permitindo tipos diferentes dos pré-definidos para determinados campos, ou mesmo registros direcionados para outros que não existam.

No que tange à Complexidade de Consultas, é consideravelmente melhor que se utilize os relacionais, uma vez que utilizam do SQL, uma linguagem de consulta poderosa que é capaz de trabalhar com joins e funções de agregação, que em variados contextos, pode trazer informações importantes sobre o contexto em que os usuários da aplicação estão inseridos.

No que tange à escalabilidade, na maior parte das vezes, é necessário somente que se escale verticalmente, ou seja, que faça o upgrade dos recursos do servidor onde se hospeda o banco de dados. De uma forma geral, é mais fácil e barato do que escalar horizontalmente. Esta última modalidade também é possível, mas é consideravelmente mais complexa de se implementar.

De uma forma geral, podemos subdividir os pontos fortes do NoSQL em:

- Escalabilidade
- Flexibilidade
- Desempenho

No que tange à escalabilidade, os bancos de dados não-relacionais são muito importantes, uma vez que são projetados para escalar horizontalmente. Escalabilidade

horizontal nada mais é do que a capacidade de se escalar um sistema entre múltiplos servidores com o intuito de atender a cargas de trabalho. Dessa forma, bancos de dados NoSQL são projetados para fazer isso facilmente, dentro do Sistema Gerenciador de Bancos de Dados em que operem.

No que tange à flexibilidade, os bancos de dados dessa natureza são consideravelmente mais vantajosos, uma vez que permitem que variados tipos de dados sejam armazenados, sem a necessidade de alguma grande alteração nas suas configurações.

No que tange ao desempenho, eles também são superiores, uma vez que foram desenvolvidos com o intuito de ler e armazenar rapidamente, especialmente quando em um sistema de gerenciamento de grandes volumes de dados.

(Tipos de bancos de dados não-relacionais)

Em relação aos bancos de dados relacionais, há algumas diferenças chave entre eles. Primeiro, devemos relembrar os tipos de bancos de dados não relacionais, abaixo:

- Chave-valor
- Documento
- Coluna Ampla
- Grafos

(Chave-valor)

Considerando, agora, os **não-relacionais do tipo chave-valor**, tem-se pares chave-valor, nos quais as chaves são identificadores únicos e que não podem ser repetidos, e o valor, que é um objeto JSON. Um exemplo disso pode ser visto abaixo:

```
Key: "session:12345"
Value: {
  "userId": "67890",
  "username": "kauanpecanha",
  "loginTime": "2024-05-27T10:15:00Z",
  "lastActivityTime": "2024-05-27T10:45:00Z",
  "sessionData": {
    "cartItems": ["item1", "item2", "item3"],
    "preferences": {
      "theme": "dark",
      "language": "en"
    }
  }
}
```

Como pode ser observado, o valor session: 12345 é a chave que identifica unicamente uma sessão de usuário em um e-commerce. Enquanto isso, seus dados estão salvos em um objeto JSON, contendo o identificador de usuário, nome de usuário, momento em que foi realizado o login, último momento de atividade, e os dados de sua sessão, como os itens adicionados ao seu carrinho, tema de cor do site, e linguagem do mesmo. É por causa dessa estruturação de dados que os não-relacionais de tipo “chave-valor” são mais adequados para sessões de usuário e armazenamento em cache. Um grande exemplo deste tipo de banco de dados é o Redis.

(Tipo Documento)

Agora, os não-relacionais de tipo documento. Um dos exemplos mais conhecidos desse tipo é o MongoDB, um que pode ser utilizado tanto na sua modalidade local através do MongoDB Atlas, como também in-cloud através do seu website. Seus dados são armazenados em BSON, ou Binary JSON, contendo todas as informações relevantes sobre aquela determinada ocorrência. O Binary JSON é um JSON otimizado, binário, que ocupa menor espaço de armazenamento e é mais eficiente, e foi criado pela MongoDB. Considere, por exemplo, o script abaixo, de um website que contém posts e comentários:

```
{
  "_id": ObjectId("60d5ec49f9fd0001bcd6c11d"),
  "title": "Introdução aos Bancos de Dados NoSQL",
  "author": "Kauan Pecanha",
  "content": "Neste artigo, discutimos os conceitos básicos de bancos de dados NoSQL...",
  "tags": ["NoSQL", "Bancos de Dados", "Tecnologia"],
  "comments": [
    {
      "author": "João Vinícius Vitral",
      "content": "Excelente artigo! Muito informativo."
    },
    {
      "author": "Guilherme Pensabem",
      "content": "Gostei da explicação sobre diferentes tipos de NoSQL."
    }
  ],
  "createdAt": "2024-05-27T10:00:00Z",
  "updatedAt": "2024-05-27T12:00:00Z"
}
```

Como pode-se notar, o _id é um identificador único daquele post, e tem atributos como título, autor, conteúdo, tags e comentários, voltados para outros usuários. Nota-se como, através de uma funcionalidade implícita, é possível inter-relacionar autores com posts através dos comentários. Ou seja, assim, mesmo sendo não-relacional, é possível fazer relações.

(Tipo Coluna Ampla)

Agora, tendo os de **coluna ampla** como foco, um dos mais conhecidos é o Apache Cassandra. Ele é uma mistura da arquitetura já existente à época, proveniente do DynamoDB, e o modelo de dados do BigData, do Google. A sintaxe do CQL (Cassandra Query Language) é um híbrido entre o SQL, com seu formato tabular, e o par chave-valor já discutido anteriormente. Sua organização, dentro do contexto de um e-commerce, voltando para o carrinho de um determinado cliente, pode ser visualizado como o abaixo:

Row Key	Column Name	Value
order12345	customer_id	cust67890
order12345	order_date	2024-05-27
order12345	total_amount	150.00
order12345	itens	{“itemId:”: “001”, “qtd”: “2”, “price”:”50”}
order12345	itens	{“itemId:”: “002”, “qtd”: “1”, “price”:”50”}

(Tipo Grafos)

Por fim, podemos falar do **banco de dados não-relacional em grafos**. Este tipo de banco de dados pode ser utilizado para expressar relações entre determinadas entidades, a título de exemplo, em uma rede social, por exemplo. O Facebook utiliza o TAO para a relação de amizades de usuários. Entretanto, o mais conhecido hoje em dia é o Neo4j, um banco de dados em grafos que utiliza de uma linguagem de consulta denominada Cypher. O exemplo de sua aplicação pode ser visto abaixo:

```
CREATE (alice:Pessoa {name: 'Alice', age: 30})
CREATE (bob:Pessoa {name: 'Bob', age: 25})
CREATE (carol:Pessoa {name: 'Carol', age: 35})
CREATE (alice)-[:AMIGO]->(bob)
CREATE (alice)-[:AMIGO]->(carol)
CREATE (bob)-[:AMIGO]->(carol);
```

Pode-se perceber que as três primeiras linhas de comando consistem na criação de três instâncias da entidade Pessoa, e as três seguintes, relações de amizade entre elas.

(Desvantagens do Banco de Dados Não Relacional)

Entretanto, os mesmos bancos de dados NoSQL tem algumas desvantagens.

Desvantagens:

1. Consistência Eventual: Alguns bancos de dados não relacionais oferecem consistência eventual, o que significa que pode haver atrasos na propagação de atualizações entre os nós do sistema, levando a resultados inconsistentes em certos cenários. Ou seja, não vai de acordo com 100% das transações **ACID**, que serão melhor explicadas posteriormente.
2. Complexidade na Consulta: Em geral, os bancos de dados não relacionais não oferecem a mesma complexidade de consulta que os relacionais, o que pode dificultar a análise de dados em certos casos.
3. Menos Ferramentas e Suporte para problemas não documentados: Em comparação com os bancos de dados relacionais, os bancos de dados não relacionais podem ter menos ferramentas e suporte disponíveis, o que pode dificultar o desenvolvimento e a manutenção de aplicativos.

(Transações ACID)

ACID significa Atomicidade, Consistência, Isolamento e Durabilidade.

(Exemplos de aplicações possíveis com MongoDB)

O banco de dados não-relacional que falaremos, nesse primeiro momento, é o MongoDB, um dos gerenciadores de bancos de dados não-relacionais mais famosos atualmente.

- Exemplo do ODS Quiz

Com ele, é possível fazer o armazenamento de registros de usuários com login em uma determinada plataforma de mapeamento de Objetivos de Desenvolvimento Sustentável, as ODSs, como foi feito no ODS Quiz. Além disso, também é possível saber o quanto cada uma foi atendida em uma determinada localidade.

- Leitura de dados vindos de dispositivos IoT

Os dispositivos de Internet of Things como Arduino e RaspberryPi são especialmente úteis no que tange a automação de tarefas repetitivas. Alguns exemplos disso são a coleta de valores de umidade através da resistividade do solo, luminosidade de um ambiente, som, entre muitos outros. Esses dados podem ser coletados e armazenados em tempo real em um banco de dados como o MongoDB para posteriormente fazer sua análise.

- Aplicativo de mensagens em chat

Um aplicativo do tipo chat pode ser feito utilizando MongoDB, uma vez que as mensagens nada mais são do que registros, e eles por sua vez podem ser adequadamente organizados dentro de clusters no MongoDB.

- Sistema de gerenciamento de conteúdo do tipo blog

Eles também são úteis para a execução de webpages do tipo blogs. Um caso de uso real é o TheGuardian. Entretanto, vamos estudar seu caso melhor.

(Estudo de Caso: The Guardian)

O TheGuardian, a partir de 2011, começou a usar o MongoDB em função de sua maior demanda por dados em larga escala. Assim, artigos, comentários, e interações entre usuários começaram a ser processados nas dependências dele. Dessa forma, foi possível utilizar de todas as vantagens como flexibilidade, escalabilidade horizontal, entre outros. Entretanto, naquela década de 2010, o MongoDB não atendia a determinadas necessidades do jornal inglês, como suporte a problemas de infraestrutura local, bem como consultas de complexidade superior, e por isso, consideraram uma solução que tivesse os seguintes atributos:

- Gerenciamento mínimo de banco de dados
- criptografia em repouso suportada
- fácil migração a partir do mongodb

Um ponto importante de se falar sobre essa migração é o banco de dados PostgreSQL. Apesar de ele ser relacional, é um dos, se não a principal exceção às características mencionadas. É como se fosse um banco de dados que reúne o melhor dos dois mundos: a complexidade de consulta e consistência de dados dos relacionais, e os atributos de escalabilidade e alta disponibilidade dos não-relacionais.

(Estudo de Caso: Pokémon Go)

O Pokémon Go é um jogo eletrônico para mobile que se destaca de todos os outros, principalmente, por ser geolocalizado. Sua estrutura pode ser melhor visualizada conforme o esquema. Aqui, podemos notar a presença de três bancos de dados diferentes, sendo eles: Spanner, Bigtable e Cloud Storage. O Spanner é o principal responsável por processar as ações do usuário dentro do jogo, como captura de pokémons e outras atividades. O Bigtable é voltado para a análise de dados, armazenando entre 5Tb a 10Tb por dia, de informação que será analisada pelos analistas e engenheiros de dados, enquanto que o Cloud Storage é responsável pelo armazenamento de objetos não estruturados e estáticos, como por exemplo mídias de imagem e animações. Um importante ponto a ser ressaltado é

sobre o Spanner, que anteriormente era o não-relacional Datastore, mas por uma questão de atendimento à transações ACID, foi preferido o Spanner.

(Exemplos de grandes softwares que utilizam Bancos de Dados Não Relacionais)

Grandes empresas e aplicativos têm adotado cada vez mais bancos de dados não relacionais para atender às suas necessidades de armazenamento e processamento de dados. Alguns exemplos notáveis incluem:

1. Tao (Facebook): O Facebook utiliza o banco de dados em grafos TAO para gerenciar as relações de amizade entre seus usuários. O TAO aproveita a estrutura flexível de um banco de dados em grafos para representar eficientemente essas relações complexas.
2. Espresso (LinkedIn): O LinkedIn utiliza o banco de dados não relacional Espresso para armazenar e consultar dados em tempo real. O Espresso oferece um desempenho consistente e uma experiência rápida para os usuários, mesmo em ambientes de alto tráfego.
3. MongoDB (Volvo): A Volvo utiliza o MongoDB para gerenciar e armazenar dados de telemetria de seus veículos. O MongoDB oferece uma alta disponibilidade e capacidade de lidar com grandes volumes de dados, garantindo a integridade dos dados em tempo real.

Esses exemplos demonstram a diversidade de aplicativos e empresas que se beneficiam das vantagens dos bancos de dados não relacionais em diferentes cenários de uso, destacando a importância e a relevância dessa tecnologia no cenário atual de armazenamento de dados.

(Conclusão)

Por fim, pode-se concluir que os bancos de dados não relacionais são soluções extremamente eficazes quando se necessita de armazenamento com flexibilidade e alto desempenho. E aqueles pontos que costumavam ser fracos, hoje, foram ultrapassados através da incorporação dos mesmos, como por exemplo o PostgreSQL, que reúne os pontos fortes dos dois mundos.