Atividade 1

- Universidade Federal de Sergipe - CCET - Departamento de Computação
- COMP0455 - BANCO DE DADOS I (2023.1 - T01)
- Prof. Dr. André Britto de Carvalho
- Pedro Vinícius de Araújo Barreto
- João Filipe de Araújo Santos Rezende

O código fonte desse documento está disponível aqui.

Sistemas de Gerenciamento de Banco de Dados (SGBD) e modelos de dados

No começo do uso dos computadores, foi enxergado o potencial dessas máquinas de armazenarem e processarem dados. As primeiras abordagens de uso do computador como tecnologia fundamental no manuseio de dados consistia na utilização de arquivos e programas não-estruturados, porém logo percebeu-se que estas levavam à grande complexidade cognitiva, de tempo e de espaço, além de riscos de segurança, ou seja, preocupações de extrema prioridade para empresas de todo porte.

Era preciso então uma nova abordagem, que fosse eficiente, segura, simples. A partir dessa necessidade, nasceram os SGBDs.

Um SGBD consiste em um conjunto de dados armazenados, isto é, o banco de dados em si, e em um conjunto de programas úteis para lidar com esses dados. Um sistema capaz de encapsular os dados, trazendo as características citadas anteriormente e que funcionasse como uma plataforma de gerenciamento que possibilitasse tanto o armazenamento quanto a análise dos dados, esta última tendo mais importância com as ferramentas estatístico-computacionais do campo de aprendizagem de máquina, que trazem *insights* sobre os dados coletados para quem estiver interessado: indivíduos, empresas, universidades e governos.

Os programas para lidar com os dados podem cumprir as mais variadas funções, como:

- Implementação de baixo nível das estruturas de dados (Árvores B, Grafos, etc) no sistema de arquivos do Sistema Operacional do computador, buscando performance e confiabilidade.
- Definição de estruturas lógicas (tabelas, hierarquias, etc) sobre as quais os dados serão representados
- Manipulação e atualização das estruturas já definidas
- Garantia de restrições de segurança ou consistência
- Estabelecer uma interface básica de conexão entre os programas supracitados com as aplicações interessadas no consumo dos dados

SGBDs modernos buscam um fundamento, ou modelo, sob o qual o banco será abstraído e, a partir da abstração, modelado da maneira apropriada. Essas

ferramentas de abtração são conhecidas como **modelos de dados**. Podemos citar alguns modelos, como:

Hierárquico e Rede

São modelos legados, que não passaram no teste do tempo, mas serviram de base e inspiração para modelos mais modernos.

Eram baseados em noções de relações primitivas para a atribuição de conexões entre o que os dados representavam no mundo real, ou "minimundo". Enquanto o hierárquico era organizado ao redor de árvores onde os nós representavam dados estruturados, o de rede baseava-se numa estrutura de grafo generalizada.

Alguns exemplos de SGBDs baseados nesses modelos são:

- Rede: IMAGE, IDS, CODASYL
- Hierárquico: IMS e RDM Mobile

Objeto-relacional

Esse modelo é o resultado da evolução de dois modelos anteriores:

- Relacional, criado por Edgar Codd, baseado na teoria matemática de relações, explicitando a estrutura (ou esquema) de cada relação. Esse modelo teve um fortíssimo sucesso, eventualmente superados os modelos hierárquico e de rede. Junto com o advento do modelo relacional, uma linguagem de dados declarativa foi criada prover as utilidades de definição e manipulação dos SGBDs que seguiram esse modelo, a **SQL**.
- Orientado a objetos: Com o advento de linguagens de programação orientadas a objetos, que permitiam a criação de componentes de software que encapsulavam informações desncessárias e expunham métodos de comunicação entre si, não demorou muito para que acadêmicos e profissionais buscassem trazer esse paradigma para o campo dos banco de dados, provendo suporte para classes, objetos e herança.

O modelo relacional passou a ser fortemente criticado devido ao seu baixo suporte pelo paradigma Orientado a Objetos, levando à uma fusão dos paradigmas existentes em um novo que trazia tanto a possibilidade de criação de relações e de seus esquemas, tendo o uso do SQL para sua definição e manipulação, mas também permitia a modelagem das relações como classes, de tal forma que os atributos dos objetos passaram a ser equivalente às colunas nas tabelas relacionais, e cada instância da classe é equivalente a uma tupla da relação. Hoje, podemos dizer que essa evolução do modelo Relacional virou o padrão da indústria, a ponto de passarem a surgir técnicas e ferramentas para fazer o mapeamento objeto-relacional, os *ORM*s.

Os principais SGBDs objeto-relacionais da atualidade são Microsoft Server, Oracle Database, MySQL, PostgreSQL e MariaDB.

NoSQL

Uma das principais críticas aos SGBDs relacionais é que eles são muito rígidos no formato dos dados o qual eles aceitam. Isso significa que sempre existiu um gasto de recursos com a necessidade de garantir a compatibilidade com o banco. O advento da Web trouxe inovações no campo de armazenamento e representações de dados, como o JSON, levando a um confronto com a rigidez relacional e uma busca por alternativas.

O NoSQL (*Not Only SQL*) não é um modelo de dados em si, mas uma categoria de modelos de dados que quebraram com a homogenia do modelo relacional e trazem (ainda são um fenômeno relativamente recente) inovações que permitem a criação de aplicações de maneira rápida e escalável, ao mesmo tempo que eles sacrificam consistência dos dados armazenados.

Os principais modelos NoSQL são aqueles baseados em:

- Documentos: Esse modelo assume que os dados são encapsulados por documentos, que podem estar em formatos como XML, YAML, JSON ou formatos binários como BSON, com cada documento associado à uma chave identificadora no banco. Diferentes implementações utilizam diferentes métodos de organizar conjuntos de documentos. Exemplos de bancos baseados em documentos são DynamoDB, MongoDB e Firestore.
- Grafos: Grafos são capazes de modelar relações do mundo real e alguns bancos de dados fazem uso dos nós, arestas e outras propriedades dos grafos para armazenar e manipular os dados de maneira escalável, fazendo uso de mais de 60 anos de algoritmos de grafos desenvolvidos na academia e na indústria. Os melhores exemplos de bancos baseados em grafos são o Neo4j e o Azure Cosmos DB (parte da plataforma cloud da Microsoft, Azure)
- Chave-valor: Bancos de dados baseados nessa abordagem utilizam um vetor associativo de chaves-valores para a represenção dos dados, de tal maneira que as chaves são únicas. Os principais bancos baseados em chave-valor são Couchbase, Azure Cosmos DB, Dynamo DB e Redis. Este último faz uso de um sistema de cache eficiente.
- Tabular: É possível modelar um banco de chave-valor em duas dimensões. Isto é feito por bancos de coluna ampla (*wide-column*), ou tabular, que usam tabelas, linhas e colunas (assim como o relacional), mas os nomes e o formatos das colunas podem variar linha-a-linha numa mesma tabela. Exemplos de bancos tabulares são Cassandra, Bigtable e DynamoDB

SGBDs particulares

Apresentamos as principais características, modelos de dados e aplicações de alguns SGBDs particulares.

PostgreSQL

Oracle 23c

Cache Database

Neo4j

Um SGBD do paradigma NoSQL, adotando o modelo baseado em grafos, onde a unidade fundamental de dados armazenados são nós, arestas e atributos de ambos. O Neo4j é escrito em Java (daí o 4j), com o objetivo de ser o principal SGBD baseado em grafos. Neo4j vem com uma linguagem de consulta adaptada para seu modelo de dados, a Cypher.

Um nó, ou vértice, pode conter uma quantidade um número arbitrário de propriedades (pares chave-valor), além de terem 0 ou mais rótulos, que ajudam a identificar papéis específicos no minimundo modelado pelo banco. Toda aresta num grafo do Neo4j é direcionada e tem um "tipo de relação". Arestas também podem ter propriedades.

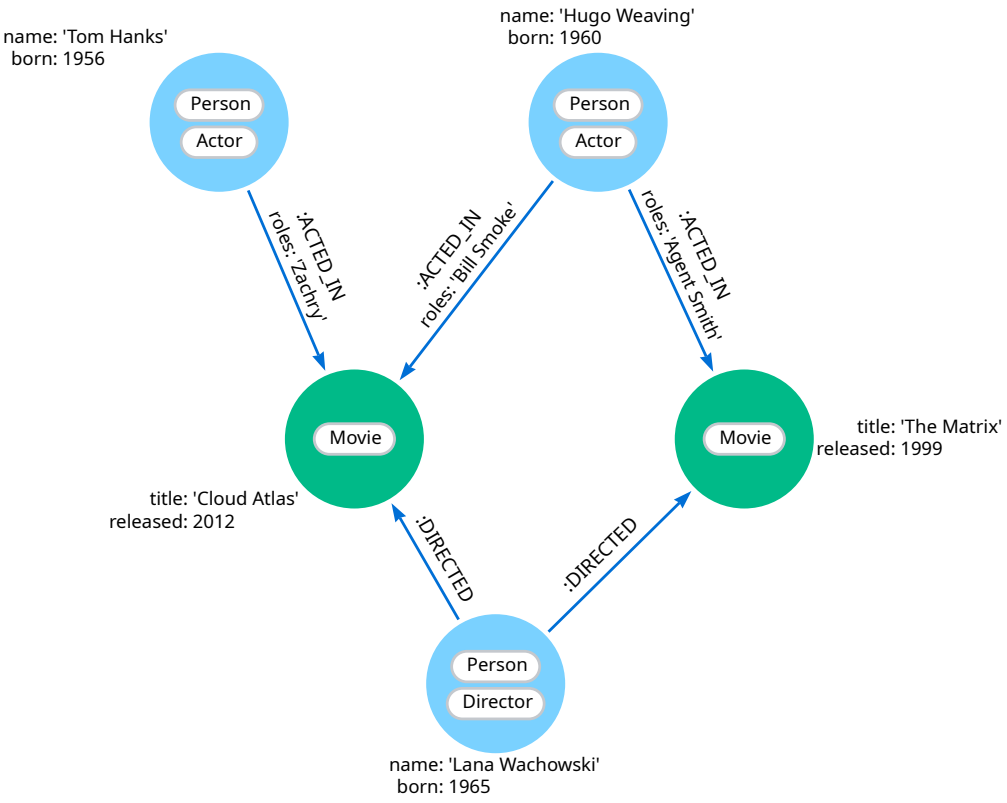

Figure 1: Um grafo que modela o minimundo da indústria de cinema

Perceba que no exemplo da figura, dois vértices com o mesmo rótulo - ator - podem ter diferentes propriedades, ou de diferentes tipos, etc. A quebra da rigidez estrutura do modelo relacional permite um esquema mais flexível para representar os dados.

Usar o grafo como modelo de dados permite eliminar joins – muito comuns no contexto relacional – atravessando o grafo coletando as informações necessárias para determinada consulta.

A linguagem de consulta, Cypher, é inspirada em SQL e adaptada para trabalhar com o modelo baseado em grafos. É uma linguagem única no sentido de que as consultas são feitas usando um apelo visual muito forte, e se baseiam fundamental na abstração de **padrão**, que são representação de relacionamentos entre nós. De maneira geral, um padrão em Cypher é da forma `(nós-[se_conectam_com]->(outrosNós)`. Para o exemplo de atores e filmes acima, o padrão "Tom Hanks atuou em Cloud Atlas" poderia ser escrito como:

```
hanks_movie = (Person {name: "Tom Hanks"})-[ACTED_IN]->(m:Movie {title: "Cloud Atlas"})
// cria essa relação no banco
CREATE hanks_movie
// encontra todos os títulos de filmes que o Tom Hanks atuou
// e o papel que ele fez em cada
MATCH (Person {name: "Tom Hanks"})-[r:ACTED_IN]->(m:Movie)
RETURN m.title, r.roles
```

Algumas das empresas e aplicações que usam Neo4j são eBay, Comcast e NASA.

SQL Server

Firebase

O sucesso da computação em nuvem permitiu a oferta de vários serviços, sejam eles de computação, armazenamento ou monitoramento. Um das categorias de serviços oferecidos é o BaaS (*Backend as a Service*), que permite delegar à plataforma nuvem muitas das atividades associadas com a lógica backend de serviços web.

Uma dessas plataformas, a GCP (*Google Cloud Platform*) oferece a Firebase, um conjunto de serviços BaaS, com um foco particular em desenvolvimento mobile. Existem 2 produtos na Firebase dedicados especificamente a, ordenados em ordem de lançamento:

- Realtime Database: O primeiro produto da Firebase, consiste num banco de dados NoSQL pouquíssimo estruturado (é literalmente uma grande árvore JSON), hospedado na nuvem da Google, e com foco na sincronização dos dados entre diferentes clientes interessados em uma determinada parte dos dados. Permite obter informações de quando os clientes estão ou não onlines, quando modificam dados (com ou sem conexão, fazendo uso de

cache local para posterior sincronização) e também quando fazem escritas e leituras frequentes de maneira síncrona com baixa tolerância para latência.

- Cloud Firestore: Segue o modelo NoSQL baseado em documentos e em agrupamentos de documentos chamados coleções, porém com uma melhor estruturação dos dados. Portanto, consultas podem ser feitas de maneira mais fácil e rápida do que no Realtime Database, fazendo uso de uma linguagem declarativa similar à SQL, que permite técnicas de consulta, ordenação e transações avançadas. A Firestore também se assemelha ao modelo hierárquico legado, pois as relações entre diferentes documentos é feita utilizando uma estrutura de árvore. Isso permite a realização de consultas superficiais, ou seja, se temos um nó que contêm um documento que aponta para um milhão de outros documentos, podemos escrever uma consulta para retornar apenas as informações do nó raiz.

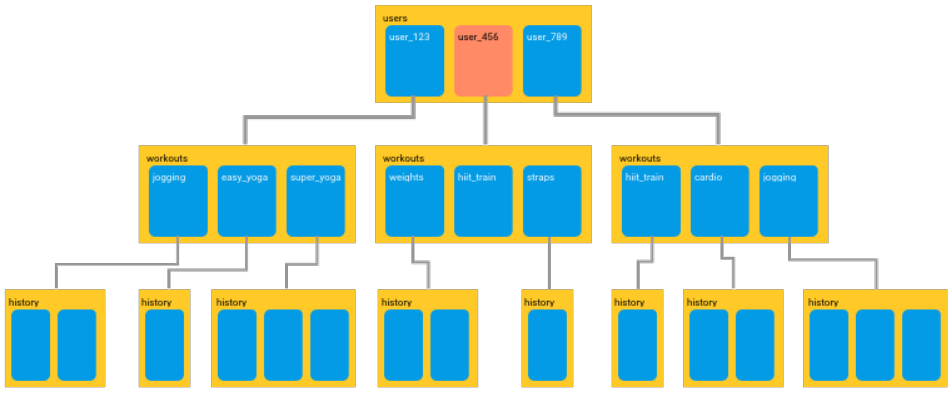

Figure 2: Organização Hierárquica de Coleções na Firestore

Assim como seu "produto irmão", a Firestore também hospeda os dados na nuvem e tem capabilidades de sincronização entre diferentes clientes.

Firebase tem sido a plataforma de escolha de muitos desenvolvedores que querem abstrair a construção do backend de sua aplicação. É usada por aplicações como Duolingo, The New York Times, Lyft, entre outros.

É importante dizer que a Google já sofreu contestações na justiça por não respeitar a privacidade de usuários da plataforma.

MongoDB

Um dos melhores representantes da categoria NoSQL é o MongoDB, um dos bancos de dados baseados em documentos mais avançados da atualidade. Foi

desenvolvido visando ser fácil de usar, escalável (de maneira distribuída), com muitas features mas sem sacrificar velocidade.

O modelo de dados usado pelo MongoDB tem como elemento fundamental o documento, que difere do elemento fundamental do modelo relacional, a tupla. O documento é um conjunto ordenado de chaves que mapeiam para certos valores (incluindo outros documentos). O resultado é maior expressividade (e escalabilidade) dos dados armazenados. Uma coleção de documentos então pode ser vista como uma tabela com um esquema dinâmico, pois não precisam seguir uma estrutura rígida. Todo documento é identificado unicamente numa coleção por uma chave `_id`.

Documentos podem ser consultados usando métodos clássicos que relembram SQL, mas também podem fazer uso de pipelines de agregações, que são uma sequência de funções chamadas estágios. A entrada e a saída de um estágio quase sempre é uma coleção. Isso permite que a saída de uma função seja a entrada de outra.

Por exemplo, uma pipeline de agregação para encontrar estatísticas de consumo e detalhes dos maiores clientes de uma coleção de pedidos seria:

```
db.orders.aggregate([
  {
    $match: {
      orderDate: { $gte: ISODate("2023-01-01T00:00:00Z") }
    }
  },
  {
    $group: {
      _id: "$customerId",
      totalOrders: { $sum: 1 },
      totalAmount: { $sum: "$totalAmount" },
      lastOrderDate: { $max: "$orderDate" }
    }
  },
  {
    $sort: {
      totalAmount: -1
    }
  },
  {
    $limit: 10
  },
  {
    $lookup: {
      from: "customers",
      localField: "_id",
      foreignField: "_id",
```

```
      as: "customerDetails"
    }
  },
  {
    $project: {
      _id: 0,
      customerId: "$_id",
      totalOrders: 1,
      totalAmount: 1,
      lastOrderDate: 1,
      customerName: { $arrayElemAt: ["$customerDetails.name", 0] },
      customerEmail: { $arrayElemAt: ["$customerDetails.email", 0] }
    }
  }
])
```

Finalmente, o método que o MongoDB utiliza para ser escalável horizontalmente é a utilização de *sharding*, ou fragmentos. Basicamente, uma coleção de muito grande pode ser dividida em fragmentos, e aplicações podem realizar consultas através de interfaceamento com um componente chamado roteador, que sincroniza entre os diferentes fragmentos e os documentos que eles possuem.

Algumas aplicações que utilizam MongoDB são Forbes, Thermo Fisher e Toyota.

Redis

Uma das alternativas de armazenamento mais populares entre desenvolvedores, o banco Redis faz parte da categoria NoSQL, usando o modelo de dados Chave-Valor, porém com um sistema de cache eficiente.

Primeiramente, é importante entender que o "valor" no modelo chave-valor adotado por Redis não está limitado a tipos primitivos que programadores estão acostumados (inteiros, floats, chars), mas embarca estruturas de dados como listas, hashes, conjuntos, hyperlogslogs, entre outros. Os dados que essas estruturas armazenam estão sempre na memória RAM e no cache. Isso difere da maioria dos SGBDs tradicionais, que fazem uso de árvores-B representadas no disco rígido, que é mais lento que a memória principal em ordens de magnitude.

Redis foi construído usando uma arquitetura cliente-servidor. O servidor interno do Redis faz algumas operações para persistir os dados no disco, de tal maneira que a eficiência não seja muito prejudicada. Veja que ainda existe um pequeno risco de não-persistência dos dados. Aplicações onde tal característica seja crítica não são apropriadas ao banco Redis.

Veja que como Redis mapeia chaves para estruturas de dados, é completamente possível que esse mapeamento também ocorra para outras estruturas que servem de suporte para outros modelos NoSQL: Grafos (RedisGraph), documentos (RedisJSON), séries temporais, etc. O conjunto dessas e outras extensões sobre

Redis é chamado de *Redis Stack*.

Inicialmente, Redis foi muito utilizado em aplicações onde a escalabilidade devido ao seu sistema de cache podia ser aproveitada: Instagram, Twitter, entre outros. Além disso tudo, é software livre e de código aberto.

Referências

- **Database System Concepts - 7th ed** Abraham Silberschatz, Henry F. Korth, S. Sudarshan. McGraw-Hill Education, 2020.
- **MongoDB: The Definitive Guide - 3rd ed**. Shannon Bradshaw, Eoin Brazil, Kristina Chodorow. O'Reilly Media, Inc, 2019.
- **NoSQL**. Wikipedia, acessado em junho de 2023. Disponível aqui
- **Redis Stack**. Redis.io, acessado em junho de 2023. Disponível aqui
- **The Infrastructure Behind Twitter: Scale**. Mazdak Hashemi. Twitter Engineering, 2017. Disponível aqui
- **Storing hundreds of millions of simple key-value pairs in Redis**. Mike Krieger. Instagram Engineering, Medium, 2011. Disponível aqui
- **Firebase website**. Acessado em Junho de 2023. Disponível aqui
- **Cloud Firestore vs the Realtime Database: Which one do I use?**. Todd Kerpelman. The Firebase Blog, 2017. Disponível aqui
- **Neo4j Documentation**. Acessado em Junho de 2023. Disponível aqui