BANCOS DE DADOS SQL E NOSQL

UMA INTRODUÇÃO PARA DESENVOLVEDORES

Pedro Barossi

Ao iniciar seus estudos no campo do desenvolvimento de aplicações, você provavelmente se deparará logo com a dualidade *front end / back end*. Não será muito difícil construir uma noção básica após um pouco de leitura: o front é a fachada convidativa do restaurante, o garçom prestativo, as mesas bem dispostas, a atmosfera aconchegante. O back é a cozinha onde todo mundo corre, o chef grita e pelo menos um cozinheiro está com os cabelos pegando fogo. Cada um com sua responsabilidade, ambos são essenciais para que o cliente saia satisfeito.

Um aspecto que comumente fica de fora da discussão diz respeito aos bancos de dados. Imagine onde é feito o controle dos pratos disponíveis, reservas feitas, funcionários trabalhando, ingredientes em estoque, pagamentos efetuados, etc. Pois é, os bancos de dados são tão importantes que até na nossa metáfora do restaurante eles são… bancos de dados!

Conhecer o que são, como funcionam e suas diferenças é de grande importância para se ter uma boa compreensão do seu próprio papel no processo de desenvolvimento, independente de qual parte você eventualmente tome nele.

O que são bancos de dados?

Ora, é um lugar onde se guardam dados. Fim, próximo tópico. Mas espere aí, o que são dados mesmo?

Dados, no escopo da computação, dizem respeito a uma sequência de símbolos. Cada uma das palavras deste artigo é um dado. Uma imagem pode ser um dado, ou mesmo conter vários dados. Isso deixa de ser grosseiramente abstrato ao considerarmos que, interpretando os dados, eles se tornam

informação. “Sopa de cenoura” e “R$25,00” são dados a partir dos quais se obtém a informação de que sopa de cenoura custa vinte e cinco reais.

Partindo desse pensamento, fica claro que o banco de dados não pode ser simplesmente uma caixa onde jogamos pedaços de informação de qualquer jeito. Seria frustrante tentar recuperar algo específico de uma coleção caótica e desordenada. Basta pensar na sua área de trabalho.

Banco de dados, no escopo que nos interessa, é o nome que se dá para uma coleção organizada de dados armazenados e acessados eletronicamente. A palavra-chave é “organizada”: algum sentido estrutural deve haver para justificar a existência do banco de dados e servir de guia para seu uso e implementação.

Muitas vezes, se usa o termo banco de dados para referir-se a um SGBD, ou sistema gerenciador de banco de dados. Embora essa equivalência possa ser inofensiva em parte das situações, é importante ter em mente que o SGBD é o meio (software) pelo qual se cria, atualiza, mantém e controla o banco de dados, e não o banco em si (CONNOLLY & BEGG, 2014). Esses sistemas gerenciadores podem ser classificados de acordo com o modelo de banco de dados que suportam, o que nos leva à questão central deste artigo.

SQL - Rigidez e precisão

SQL, ou structured query language (linguagem estruturada de consulta), é o termo que comumente abarca SGBDs que suportam bancos de dados relacionais. Uma maneira simples de compreender esse conceito é imaginar duas tabelas: uma de pessoas e uma de bichos de estimação.

Na tabela de pessoas, temos os dados que descrevem seu nome e idade. Desses dados tiramos as informações de que Joãozinho tem 8 anos, Margarete tem 42, Diogo tem 31, etc. Na tabela de bichos de estimação estão relacionados os nomes, tipo de animal e donos de cada um deles. Por exemplo, Bob é um cachorro e seu dono é Joãozinho. Sim, o Joãozinho da primeira tabela! Apesar de cada uma guardar dados sobre entidades diferentes - pessoas e animais -, elas têm uma relação entre si. A partir dessa relação tiramos as informações de que Jiraya é a iguana de Diogo e que Margarete é dona dos gatos Fifi, Mel, Bolinha, Pequeno, Rony e Fofinha (não julgue).

Lembre-se, porém, que SQL diz respeito a uma linguagem, não aos bancos relacionais em si. A linguagem SQL funciona a partir de *statements*, ordens que são interpretadas e, quando possível, realizadas. Seria como dizer “mostre a tabela de pessoas”, “crie uma tabela de receitas”, “apague o Bob da tabela de bichos de estimação”. O último é só um exemplo, Bob está bem.

Os statements podem ser subdivididos de acordo com sua função:

- DDL (*Data Definition Language*) - cria e modifica estruturas dentro do banco de dados, ou até os bancos em si. São comandos como CREATE, ALTER e DROP (criar, alterar e descartar);
- DML (*Data Manipulation Language*) - diz respeito a manipular os dados que serão inseridos, alterados ou deletados dentro das estruturas definidas no DDL. Respectivamente, comandos INSERT INTO, UPDATE e DELETE FROM;
- DQL (*Data Query Language*) - usada para fazer as consultas de dados. Apesar de muitas vezes considerada parte da DML, seu versátil comando SELECT não acarreta em si uma manipulação dos dados;
- DCL (*Data Control Language*) - cuida do acesso aos dados, aspecto de altíssima importância no aspecto de segurança da informação. Seus statements gerenciam as permissões que cada usuário tem em um determinado banco de dados através dos comandos GRANT e REVOKE (conceder e revogar).

Uma das maiores forças dos bancos de dados construídos em cima de SQL é sua confiabilidade: os dados se manterão válidos independente de eventuais problemas durante as execuções dos statements. Isso acontece porque as transações no SQL seguem os princípios do acrônimo **ACID**. A saber:

- **A**tomicidade - nenhuma transação ocorre pela metade: ou tudo que foi pedido é executado, ou nada é. Nenhuma foto no Instagram tem meio like por mais feia que seja;
- **C**onsistência - uma transação só pode levar o banco de dados de um estado válido até outro, nenhuma operação que viole as regras do banco será executada. O campo Data de Nascimento tem que ser preenchido com uma data. “Juventude é um estado de espírito” não é uma data, sinto muito;
- **I**solamento - no caso de múltiplas transações ocorrerem ao mesmo tempo, o estado intermediário de uma delas não é visível para as outras. Ou o dinheiro está na sua conta ou não está, “chegando” não é um estado possível, por mais que seu cunhado que emprestou uma grana queira te convencer do contrário;
- **D**urabilidade: uma vez realizada a transação, seu resultado está guardado definitivamente e não volta atrás por qualquer falha de sistema. A pizza está vindo mesmo que seu celular caia na privada depois do pedido efetuado.

Porém, essa robustez do SQL vem com um custo: sua estrutura é rígida, engessada. Não há espaço para a flexibilidade e rápida transformação e geração das informações no mundo moderno.

NoSQL - Adaptando-se às demandas

Imagine um conceito tão flexível que até mesmo a interpretação de seu nome varia de um contexto para outro. É possível que autores o descrevam como uma antítese, literalmente “não”-SQL: um banco de dados não relacional, que rejeita o rigor estrutural e arcaico do SQL. Também se encontram referências a “Not Only SQL”, indicando uma extensão na forma de pensar que não exclui completamente o paradigma anterior.

Qualquer que seja o contexto, a primeira coisa a se deixar clara é que bancos NoSQL não são bagunçados. Ainda é preciso dar sentido aos dados para que eles sejam informação em vez de desperdício de espaço e recursos. Ou seja, alguma organização precisa existir. Existem diversas metodologias pelas quais isso pode ser alcançado saindo do padrão de tabelas do SQL e veremos as quatro mais comuns:

- **Colunas**: os dados são organizados em colunas extensíveis que podem ou não ser agrupadas numa mesma família. Se você acha este conceito muito parecido com tabelas, é porque é exatamente isso que ele é. Diferente dos bancos relacionais, porém, um NoSQL orientado a colunas não possui uma estrutura pré definida e imutável. No primeiro caso, se uma informação extra precisasse ser adicionada a uma linha, uma nova coluna seria criada para toda a tabela. Já num banco não-relacional apenas aquela parte seria afetada. Da mesma forma, se um certo dado não se aplica àquela linha, pode ser ignorado sem necessidade de inserir um valor nulo;
- **Grafos**: grafos são representações em que elementos são conectados uns aos outros por um número finito de relacionamentos. Pense nas pessoas que você conhece: você é uma pessoa que está ligada às outras pessoas da sua família por uma relação de parentesco, aos seus amigos por uma ligação de amizade, aos vizinhos por uma relação de proximidade (e, talvez, fofoca). Extrapolando um pouco o conceito, pense naquele colega de escola. Vocês dois estão ligados à instituição escola, que está ligada a vocês e aos professores, mas por relacionamentos diferentes;

- **Chave-valor**: os dados contêm um valor e são identificados por uma chave, a qual é usada nas consultas. Por exemplo, eu pergunto ao banco sobre a chave "*estoque_de_cafe*" e ele me retorna o valor "0". Basicamente, este tipo de banco de dados funciona como um vetor associativo, também conhecido como mapa ou dicionário. Apesar de simples, é um modelo que se torna poderoso quando as chaves são organizadas em ordem alfabética, por exemplo;
- **Documentos**: compreendendo um documento como um algo que encapsula e codifica informação de acordo com um padrão, este banco de dados trabalha organizando-os em coleções. Uma forma de entender este conceito é imaginar o próprio sistema de pastas do seu computador. Ou do computador de alguém realmente organizado.

A natureza mais fluida desses bancos de dados os tornam essencialmente diferentes do SQL. Ao contrário das regras transacionais ACID, o conceito geralmente associado aos SGBDS NoSQL é o **BaSE**: ***B****asically* ***a****vailable,* ***S****oft state,* ***E****ventually consistent*:

- Várias fontes terão acesso para alterar um dado ao mesmo tempo, em diferentes nós de um sistema distribuído, o que pode resultar em inconsistências;
- Não é possível afirmar sempre com certeza em que estado se encontra uma transação, pois esta pode ainda não ter sido replicada corretamente ao longo de todo o sistema;
- Sua consistência é bastante diferente, descrita como *eventual*: se não houver atualização de um determinado dado, eventualmente todos os acessos a ele resultarão na sua versão mais atual.

Qual escolher?

Os bancos NoSQL não vieram para substituir os SQL, o que quer dizer que eles coexistem pacificamente no mundo, muitas vezes no mesmo contexto de aplicação. Como saber, então, qual modelo será utilizado? As próprias características de cada um nos guiam na escolha.

Um sistema de pagamentos, por exemplo, não pode se dar ao luxo de um banco "soft state": o rigor formal dos bancos SQL é necessário para garantir a segurança das transações. Já uma rede social, com suas relações sempre em transformação, não poderia operar com um banco de dados de estrutura estática, imutável.

Outra questão importante é a *escalabilidade*, ou seja, a capacidade que um sistema tem de lidar com uma crescente de demandas somando recursos a si (BONDI, 2000). A escalabilidade *vertical* dos SGBDs SQL, de maneira resumida, se dá explorando ao máximo a configuração de uma única máquina, muitas vezes um supercomputador. Apesar desta maneira ser relativamente

mais simples de administrar, toda máquina tem um limite. Uma alternativa, para a qual os sistemas NoSQL são mais orientados, é a escalabilidade *horizontal*, na qual mais de um servidor trabalhará na mesma carga de requisições. É imprescindível deixar claro que ambos esses grandes paradigmas podem trabalhar em conjunto, o que é comum em aplicações de grande porte.

Em conclusão, espero com este artigo ter apresentado uma pequena base de conhecimento que lhe permita, ao menos, escolher por onde seguir com suas próprias pesquisas sobre este assunto tão importante para o desenvolvimento de software. Bons estudos!

Referências bibliográficas

CONNOLLY, Thomas M.; BEGG, Carolyn E. (2014). *Database Systems – A Practical Approach to Design Implementation and Management* (6th ed.). Pearson. ISBN 978-1292061184.

LANG, Niklas (2022). *Database Basics: ACID Transactions*. https://towardsdatascience.com/database-basics-acid-transactions-bf4d38bd8e26?gi=1a5f9f867709. Acesso em 25/09/2022.

SEEGER, Mark (2009). *Key-value Stores: A Practical Overview*. http://blog.marc-seeger.de/assets/papers/Ultra_Large_Sites_SS09-Seeger_Key_Value_Stores.pdf. Acesso em 25/09/2022.

BONDI, André B. (2000). *Characteristics of scalability and their impact on performance*. Proceedings of the second international workshop on Software and performance – WOSP ISBN 158113195X.