CONSISTÊNCIA DE DADOS

O que é Consistência de Dados?

A consistência de dados é um conceito fundamental no campo da gestão de dados e se refere à qualidade dos dados em um sistema ou banco de dados. É a garantia de que os dados estão corretos, precisos e atualizados, de acordo com as regras e restrições definidas. A consistência de dados é essencial para garantir a confiabilidade e a integridade dos dados, bem como para evitar erros e inconsistências que possam afetar negativamente as operações de uma organização.

Sem consistência, a precisão das análises, relatórios e funcionalidades do sistema pode ser comprometida, levando a falhas operacionais, perda de confiança dos usuários e impactos negativos nos resultados organizacionais. Portanto, implementar e manter a consistência de dados não é apenas uma prática recomendada, mas uma necessidade crítica para qualquer sistema de software que busca operar de forma eficiente e confiável em um ambiente dinâmico e complexo.

Na programação, a consistência de dados refere-se à garantia de que os dados mantidos e manipulados por um programa ou sistema estejam corretos, válidos e livres de contradições em todos os momentos. Isso é crucial para garantir a integridade dos dados e a correta funcionalidade do software. Alguns aspectos importantes da consistência de dados na programação são:

Consistência Semântica, Consistência de Formato, Consistência de Estado, Consistência de Integridade, Consistência de Transação

Importância da Consistência de Dados

A consistência de dados desempenha um papel crucial em várias áreas, como finanças, saúde, comércio eletrônico e muito mais. Quando os dados não são consistentes, podem ocorrer problemas graves, como transações financeiras incorretas, erros médicos, envio de produtos errados aos clientes, entre outros. Além disso, a falta de consistência de dados pode levar a decisões erradas e prejudicar a reputação de uma empresa.

Garantir a consistência de dados é especialmente importante em sistemas distribuídos, nos quais os dados são armazenados em vários locais e podem ser acessados por diferentes usuários simultaneamente. Nesses casos, é necessário implementar mecanismos de controle e sincronização para garantir que todos os dados estejam consistentes em todos os locais.

Para manter a consistência de dados na programação, é comum usar técnicas como validação de entrada de dados, uso de estruturas de dados apropriadas, implementação de regras de negócio para verificar a integridade dos dados, e transações para agrupar operações relacionadas que devem ser tratadas de forma atômica. Essas práticas ajudam a prevenir erros de programação, melhoram a confiabilidade do software e garantem a precisão dos resultados produzidos pelo sistema.

Tipos de Consistência

Consistência forte (ou strong consistency) refere-se a um modelo no qual todas as operações de leitura refletem imediatamente todas as operações de escrita anteriores. Em outras palavras, após uma operação de escrita ser concluída com sucesso, todas as operações de leitura subsequentes retornarão imediatamente o valor mais recente gravado. Isso significa que todos os nós no sistema veem a mesma versão mais recente dos dados no mesmo momento.

Consistência eventual (ou eventual consistency), por outro lado, é um modelo onde as leituras podem retornar valores desatualizados por um período de tempo limitado, até que todas as réplicas de dados sejam atualizadas. Em sistemas com consistência eventual, após uma operação de escrita, pode levar algum tempo até que todas as réplicas de dados sejam atualizadas para refletir a mudança. Durante esse intervalo de tempo, diferentes nós podem retornar valores diferentes para a mesma consulta de leitura.

Consistência transacional (ACID):

ACID é um acrônimo que descreve as propriedades fundamentais das transações em bancos de dados

Sistemas que seguem o modelo ACID são geralmente consistentes e oferecem garantias fortes sobre a integridade dos dados, mas podem ser mais complexos de escalar horizontalmente em ambientes distribuídos devido à necessidade de coordenar todas essas propriedades entre nós diferentes.

Exemplos de uso de transações em bancos de dados relacionais.

O setor varejista depende fortemente de um gerenciamento de estoque preciso e eficiente para garantir a satisfação do cliente e minimizar rupturas ou excesso de estoque. Os bancos de dados relacionais desempenham um papel crucial no gerenciamento e organização de grandes quantidades de dados de estoque para empresas de varejo. Os varejistas usam sistemas de gerenciamento de banco de dados relacional (RDBMS) para armazenar e gerenciar informações de produtos, categorias, fornecedores, níveis de estoque e transações de vendas. Esses bancos de dados ajudam os varejistas a rastrear os níveis de estoque em vários locais, gerenciar o reabastecimento e avaliar o desempenho do produto ao longo do tempo. Por exemplo, um banco de dados pode incluir as seguintes tabelas:

Produtos: ID do produto (chave primária), nome, descrição, categoria, preço, imagem e outros atributos específicos do produto

Categorias: ID da categoria (chave primária), nome e descrição

Fornecedores: ID do fornecedor (chave primária), nome, endereço e informações de contato

Inventário: ID do inventário (chave primária), ID do produto (chave estrangeira), ID do local da loja (chave estrangeira), nível de estoque e última atualização

Transações de vendas: ID da transação (chave primária), ID do cliente (chave estrangeira), ID do produto (chave estrangeira), quantidade e data

Ao estabelecer relações entre essas tabelas, os varejistas podem gerar rapidamente relatórios e analisar dados sobre níveis de estoque, desempenho de vendas e necessidades de reposição de estoque. Essas

informações ajudam a tomar decisões informadas em relação ao gerenciamento de estoque, reduzir custos e garantir a satisfação do cliente.

Controle de concorrência

Controle de concorrência é um conjunto de controles para evitar possíveis inconsistências nos dados quando se trabalha em ambientes com múltiplos usuários. Se faz necessário bloquear as informações.

- **Bloqueios**: A técnica de bloqueio em duas fases para controle de concorrência é baseado no bloqueio de itens de dados. Este bloqueio pode ser binário (possui dois valores: 1 e 0), assim, o item de dados pode estar, ou não, bloqueado, o que permite que o item de dado só esteja acessível para uma transação apenas se a variável não estiver bloqueada (ou estiver com valor 0). Em ambientes com aplicações multiusuários, onde uma situação comum é a concorrência de dados, vários usuários pegam um mesmo dado simultaneamente, e a atualização do mesmo pode ser trabalha de duas maneiras, otimista ou pessimista.

Controle otimista:

O controle otimista não bloqueia registro quando os lê. Quando um usuário deseja alterar uma senha, o programa verifica se outro usuário fez alguma alteração desde que foi lido. Esse tipo de controle geralmente é utilizado quando se tem baixo fluxo de dados, o que possibilita aliviar o servidor de custo adicional em manter vários bloqueios.

Controle pessimista:

Controle pessimista trabalha com o conceito de bloqueio do registro na fonte de dados, impedindo que os usuários alterem os dados de uma forma que afeta outros usuários. Dessa forma, se um usuário efetuar um bloqueio, os outros estão impedidos de realizar alguma ação até que o proprietário do bloqueio finalize.

Consistência eventual

O termo foi criado no Bayou Paper no ano de 1995 por Douglas Terry e significa que, para cada objeto em uma coleção de objetos, todas as réplicas de cada um terão eventualmente o mesmo valor. A consistência eventual é um modelo de consistência de dados que permite que os armazenamentos de dados estejam altamente disponíveis.
Também é conhecido como replicação otimista e é fundamental para sistemas distribuídos.

Princípio BASE:

- **Basically Available :**

O conceito de "**Basically Available**" tem relação com a durabilidade e disponibilidade de uma alteração, em um banco de dados com dados distribuídos, todos os nós são considerados disponíveis e é possível que duas aplicações alterem uma determinada informação, simultaneamente, cada uma em um nó diferente.

- **Soft State:**

O segundo conceito envolvido no modelo BASE é o conceito de "Soft State", se imaginarmos os estados pelos quais uma transação passa, podemos ter períodos em que está ativa, em processo de finalização, sendo abortada ou concluída, seguindo o mesmo princípio uma transação distribuída só pode ser considerada "concluída" quando de fato a alteração foi replicada para todos os nós.

Antes que isso ocorra, a leitura da informação a partir de todos os nós pode retornar dados inconsistentes e não temos como saber se a alteração falhou no segundo nó ou devido à alta latência, o dado ainda não tenha sido replicado, como não podermos afirmar em que estado a transação se encontra, temos o "Soft State".

- **Eventual Consistency:**

Para entender como o "Eventual Consistency" funciona vamos imaginar que um database possui 2 nós e que todo dado gravado em um dos nós é replicado para o outro. Com esta configuração, passamos a ter duas opções de leitura da informação, dobrando o desempenho da leitura.

Porém, para não prejudicar o desempenho da escrita, a aplicação faz alterações em apenas um dos nós e, através de um processo independente, estas alterações são replicadas para o outro nó.

Como esta replicação não ocorre instantaneamente (não atômica), existe a possibilidade de que uma aplicação lendo a mesma informação a partir dos dois nós obtenha valores distintos, sendo considerados inconsistentes.

Entretanto, se uma informação não sofre alterações por algum tempo, podemos esperar que ao término da replicação, os dados fiquem iguais, voltando a ser consistentes.

Exemplos de Consistência Eventual:

- **Banco de dados:**

Tecnologias **NoSQL** costumam ser ótimas para obter **escalabilidade**.
Já a consistência é problemática. A NoSQL basicamente aceita a tese da **consistência eventual**. Ou seja, em algum momento o banco de dados estará consistente. Inclusive em algum momento não definido estará durável, ou disponível.
Nem todos os problemas exigem que a consistência seja absoluta. Então NoSQL troca uma capacidade por outra.

- **Microoserviços:**

Permite mais flexibilidade aos microsserviços. Pode ocorrer, em um sistema distribuído, que um nó seja atualizado e o novo valor exibido, mesmo que os demais nós ainda estejam exibindo um valor desatualizado. Ou seja, a consistência eventual permite que exista um delay entre os nós do sistema.

Ferramentas de Bancos de Dados

- **Ferramentas que suportam transações ACID:**

Transações ACID são um conjunto de propriedades essenciais em sistemas de banco de dados:

Atomicidade: Garante que uma transação seja realizada completamente ou não seja realizada de forma alguma, evitando estados intermediários inconsistentes.

Consistência: Assegura que apenas transações válidas e que respeitam as regras do banco de dados sejam aceitas, mantendo a integridade dos dados.

Isolamento: Garante que o resultado de uma transação seja independente de outras transações em execução simultaneamente, evitando interferências entre elas.

Durabilidade: Garante que as alterações realizadas por uma transação confirmada sejam permanentemente salvas no sistema, mesmo em caso de falhas.

Essas propriedades garantem que as transações sejam executadas de maneira segura, confiável e consistente, mantendo a integridade dos dados em ambientes de múltiplas operações concorrentes.

As principais ferramentas de bancos de dados que suportam transações ACID são:

- Oracle Database
- SQL Server (Microsoft)
- SQLite
- MariaDB
- PostgreSQL
- MySQL

- **Ferramentas que suportam consistência eventual:**

A consistência eventual permite que diferentes partes de um sistema distribuído possam temporariamente divergir na visão dos dados atualizados, com a promessa de que eventualmente todas as réplicas alcançarão um estado consistente.

Além do Cassandra e MongoDB, outras ferramentas de banco de dados que suportam consistência eventual são:

- Amazon DynamoDB
- Couchbase
- Riak

Integração de Dados

São técnicas para garantir consistência em sistemas integrados.

As técnicas são essenciais para manter a integridade e a confiabilidade dos dados em sistemas integrados, assegurando que todas as partes do sistema tenham uma visão precisa e atualizada dos dados em tempo real. As principais técnicas são:

- **Modelos de Consistência de Dados:** Definir e aplicar modelos que regulam como os dados são atualizados e acessados para manter uma visão coerente entre todas as partes do sistema.

- **Transações Distribuídas:** Utilizar transações que coordenam operações entre diferentes partes do sistema, assegurando que todas sejam concluídas com sucesso ou revertidas de forma consistente.

- **Protocolos de Comunicação e Sincronização:** Implementar protocolos robustos para garantir que as operações ocorram na sequência correta e que os dados sejam atualizados de maneira consistente entre os componentes.

- **Resolução de Conflitos:** Desenvolver métodos para resolver divergências nos dados entre diferentes partes do sistema, incluindo detecção e resolução automática ou manual de conflitos.

- **Monitoramento e Auditoria:** Estabelecer sistemas de monitoramento para verificar e garantir a conformidade das operações com as políticas estabelecidas, identificando e corrigindo inconsistências rapidamente.

- **Padrões de Projeto e Arquitetura:** Utilizar práticas de projeto e arquitetura que promovam a modularidade e coesão entre os componentes, facilitando a manutenção da consistência ao longo do ciclo de vida do sistema.

Exemplos de uso de ETL (Extract, Transform, Load).

ETL (Extract, Transform, Load) é um processo utilizado para extrair, transformar e carregar dados de diferentes fontes para um destino específico. Exemplos de aplicações:

- **Migração de Dados:** Transferência de dados de um sistema legado para um novo sistema, usando ETL para extrair, transformar e carregar os dados.
- **Data Warehousing:** Coleta de dados de várias fontes operacionais para um data Warehouse centralizado, utilizando ETL para extrair, transformar e carregar os dados para análise.
- **Integração de Dados:** Integração de sistemas após aquisições corporativas, com ETL para extrair, transformar e carregar dados de múltiplas fontes.
- **Consolidação de Dados**: Unificação de dados de diferentes unidades ou filiais em um sistema central, usando ETL para extrair, transformar e carregar os dados consolidados.
- **Business Intelligence (BI):** Preparação de dados para análise de BI, onde ETL é usado para extrair, transformar e carregar dados para um data warehouse de BI.

Estudos de Caso e Exemplos Práticos

O que é um sistema distribuído?

Um sistema distribuído é uma coleção de computadores independentes que se comportam como um único computador para os usuários. Nós, ou esses computadores, colaboramos para realizar tarefas e fornecer serviços. Os principais atributos dos sistemas distribuídos são:

Multiplicidade de Máquinas: é composto por múltiplos computadores que estão juntos.

Comunicação através de Rede: Máquinas se comunicam e planejam ações por meio de uma rede, como a Internet ou a intranet.

Transparência: Para os usuários finais, o sistema parece ser um único computador, escondendo a complexidade da distribuição de tarefas.

Tolerância a Falhas: Mesmo que alguns computadores falhem, o sistema pode continuar funcionando, embora com menor desempenho.

Alguns exemplos de sistemas distribuídos são:

- Redes de Computadores: uma rede composta por muitos computadores com acesso a internet
- Sistemas de arquivos distribuídos: como os sistemas de arquivos Google (GFS) e Hadoop Distributed File System (HDFS), permitem acesso e armazenamento de arquivos em vários computadores.

- Serviços de nuvem: plataformas como Amazon Web Services (AWS), Google Cloud Platform (GCP) e Microsoft Azure.
- Aplicativos Web e Bancos de Dados Distribuídos: Fornecem serviços escaláveis e resilientes, como o Facebook ou o Google, usando vários servidores e bancos de dados distribuídos.

Hoje em dia na área de TI esses sistemas distribuidos são muito utilizados para aumentar a eficiência, a capacidade de processamento e a disponibilidade de serviços.

Principais Desafios na Garantia da Consistência de Dados

- **Coordenação de Réplicas:** Em sistemas distribuídos, os dados são frequentemente replicados em múltiplos nós para melhorar a disponibilidade e a tolerância a falhas. Um dos principais desafios é garantir que todas as réplicas de um dado específico estejam consistentes entre si. Isso envolve sincronização adequada para que todas as atualizações sejam replicadas corretamente e na ordem adequada.

- **Consistência vs. Disponibilidade:** O teorema CAP (Consistency, Availability, Partition Tolerance) postula que em um sistema distribuído é impossível garantir simultaneamente consistência forte, alta disponibilidade e tolerância a partições de rede. Assim, equilibrar entre essas três propriedades é um desafio constante. Escolher o nível apropriado de consistência é crucial e geralmente envolve compromissos com disponibilidade e desempenho.

- **Transações Distribuídas:** Coordenar transações que envolvem múltiplos nós em um sistema distribuído é complexo. Garantir atomicidade, consistência, isolamento e durabilidade (ACID) em operações que abrangem diferentes partes do sistema requer protocolos de transação distribuída robustos, como o Two-Phase Commit (2PC) ou protocolos mais modernos baseados em transações distribuídas.

- **Concorrência e Conflitos:** Em ambientes distribuídos, várias operações concorrentes podem modificar os mesmos dados ao mesmo tempo. Gerenciar conflitos de maneira eficiente e garantir que as operações sejam aplicadas corretamente, mantendo a consistência, é um desafio crítico. Técnicas como controle de concorrência, travamentos distribuídos e resolução de conflitos são utilizadas para lidar com essas situações.

- **Escalabilidade Horizontal:** À medida que o número de usuários e transações aumenta, os sistemas precisam escalar horizontalmente adicionando mais nós. Garantir que a consistência dos dados seja mantida durante esse processo de escalabilidade é desafiador. Técnicas como particionamento de dados, sharding e técnicas de consistência eventual são frequentemente utilizadas para lidar com grandes volumes de dados e alta concorrência.

- **Latência e Desempenho:** Manter a consistência dos dados sem comprometer o desempenho é essencial em cenários de alta concorrência. Protocolos de consistência que introduzem menor latência, como consistência eventual, são preferidos em alguns casos, enquanto outros exigem protocolos mais rígidos, como consistência forte, que podem aumentar a latência.

- **Concorrência Extrema:** Em casos de picos de carga ou eventos simultâneos intensos, a concorrência nos acessos aos dados pode ser extrema. Isso pode levar a bloqueios, gargalos de desempenho e aumento do risco de inconsistências. Estratégias como escalonamento de transações, otimizações de acesso aos dados e distribuição inteligente de carga são importantes para mitigar esses problemas.

- **Resiliência a Falhas:** Aumentar a resiliência do sistema contra falhas de rede, falhas de nó ou erros de software é fundamental para manter a consistência dos dados em cenários de alta concorrência. Mecanismos de detecção de falhas, recuperação automática e replicação de dados são algumas das técnicas utilizadas para garantir a disponibilidade e a integridade dos dados.

Em resumo, garantir a consistência de dados em sistemas distribuídos e em cenários de alta concorrência requer um planejamento cuidadoso, escolha de tecnologias apropriadas e implementação de estratégias de gerenciamento de dados robustas. O equilíbrio entre consistência, disponibilidade e desempenho é um dos desafios centrais enfrentados pelos arquitetos de sistemas distribuídos modernos.

Melhores Práticas

Design de banco de dados e arquitetura de sistemas para suportar consistência:

- **Modelagem de Dados**: Começa com um sólido entendimento dos requisitos de negócio e dos padrões de acesso aos dados. Utilizar modelos de dados que minimizem a necessidade de atualizações concorrentes em um mesmo registro pode reduzir conflitos e melhorar a consistência. O uso de técnicas como particionamento de dados, sharding e replicação controlada pode ajudar a distribuir a carga e melhorar a escalabilidade sem comprometer a consistência.

- **Escolha do Modelo de Consistência:** Decidir entre modelos de consistência forte, eventual ou transacional depende das necessidades específicas do sistema. Em sistemas distribuídos, muitas vezes é necessário abrir mão de consistência forte em troca de disponibilidade e

tolerância a partições. Avaliar os compromissos entre consistência, disponibilidade e tolerância a falhas é crucial durante o design.

- **Protocolos de Sincronização e Consenso:** Utilizar protocolos de sincronização como Paxos, Raft ou algoritmos customizados para coordenar atualizações entre nós distribuídos. Esses protocolos ajudam a garantir que todas as réplicas de dados estejam consistentes após uma operação de escrita.

- **Gerenciamento de Transações Distribuídas:** Implementar mecanismos de controle de transações distribuídas (como Two-Phase Commit) para garantir atomicidade e consistência em operações que envolvem múltiplos nós.

Monitoramento e Auditoria de Dados para Detectar Inconsistências:

- **Monitoramento Contínuo:** Estabelecer um sistema robusto de monitoramento que acompanhe o estado de todas as réplicas de dados em tempo real. Isso inclui verificar a integridade dos dados, identificar divergências entre as réplicas e alertar quando inconsistências são detectadas.

- **Auditoria de Logs e Eventos:** Manter registros detalhados de todas as operações realizadas nos dados. Isso não só facilita a detecção de inconsistências, mas também ajuda na investigação de falhas e na reconstrução de estados de sistema.

- **Utilização de Métricas de Saúde do Sistema:** Monitorar métricas como latência de replicação de dados, taxa de erros em operações de sincronização e tempo médio para resolver inconsistências pode ajudar a identificar problemas antes que eles afetem a operação do sistema.

Ferramentas de Testes e Validação de Consistência:

- **Testes de Unidade e Integração:** Incluir testes automatizados que validem a consistência dos dados em diferentes cenários de uso. Isso pode incluir testes de concorrência, testes de recuperação de falhas e testes de carga para avaliar como o sistema se comporta sob pressão.

- **Testes de Performance:** Avaliar o desempenho do sistema sob diferentes níveis de carga para garantir que a consistência dos dados seja mantida sem comprometer o tempo de resposta e a escalabilidade.

- **Simulação de Falhas:** Realizar testes de simulação de falhas para verificar como o sistema se comporta em situações de falha de rede, falha de nó ou perda temporária de conectividade entre nós distribuídos.

- **Validação de Backups e Recuperação:** Testar regularmente procedimentos de backup e recuperação para garantir que dados recuperados estejam consistentes e íntegros.

Em resumo, adotar práticas sólidas de design de banco de dados, implementar monitoramento rigoroso e utilizar ferramentas adequadas de testes e validação são essenciais para garantir a consistência de dados em sistemas distribuídos. A combinação dessas práticas ajuda a mitigar riscos de inconsistência e a manter a integridade dos dados ao longo do tempo.

CONCLUSÃO

A consistência de dados é crucial para assegurar que as decisões empresariais sejam baseadas em informações confiáveis e precisas. Quando os dados são consistentes, a precisão e a confiabilidade das análises e operações são substancialmente melhoradas, proporcionando uma base sólida para o crescimento e sucesso organizacional.

Entre as principais técnicas discutidas para garantir a consistência de dados estão a aplicação de padrões rigorosos na entrada de dados, a validação automática para detecção precoce de erros, a normalização para uniformizar formatos, o controle de versão para rastreamento de mudanças ao longo do tempo, e o uso de ferramentas de ETL para integração e transformação de dados de maneira consistente.

Além disso, enfatizamos a importância da governança de dados e do monitoramento contínuo para garantir que os dados mantenham sua integridade ao longo do ciclo de vida da informação. Essas práticas não apenas melhoram a qualidade dos dados, mas também contribuem para a eficiência operacional e a confiança nas análises estratégicas.

Em resumo, a consistência de dados não é apenas um objetivo técnico, mas sim um pilar fundamental para a tomada de decisões informadas e bem-sucedidas em qualquer organização moderna. Ao implementar as técnicas e ferramentas discutidas, as empresas estarão melhores posicionadas para aproveitar ao máximo o potencial dos dados, impulsionando inovação, competitividade e crescimento sustentável.

REFERÊNCIAS

Database System Concepts - Abraham Silberschatz, Henry F. Korth, S. Sudarshan.

Elmasri, Ramez, e Shamkant B. Navathe. "Sistemas de Banco de Dados." Pearson, 2011

GUILHERME TAVARES DE ASSIS. [s.l: s.n.]. Disponível em: <http://www.decom.ufop.br/guilherme/BCC441/geral/bd2_controle-de-concorrencia.pdf>.

CORDEIRO, D. ach 2147 -desenvolvimento de sistemas de informação distribuídos modelos de consistência. [s.l: s.n.]. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/3609790/mod_resource/content/1/13.pdf>.

Bancos de dados ACID versus BASE — Diferença entre bancos de dados — AWS. Disponível em: <https://aws.amazon.com/pt/compare/the-difference-between-acid-and-base-database/>.

ZANLUCAS, L. Tratamento de dados: Transações ACID. Disponível em: <https://www.dio.me/articles/tratamento-de-dados-transacoes-acid>. Acesso em: 28 jun. 2024.

.