Resumo de Banco de Dados do Kuskão (Os não relacionais, pq banco relacional é para maricas

Transações

* Transações é apenas uma maneira de chamar as operações que o banco de dados realiza
* De forma simples, uma transação pode ser encarada como um conjunto de operações de leitura e escrita de dados, como no exemplo abaixo:

Diagrama

Descrição gerada automaticamente

ACID

A 🡪 Atomicidade:

* Princípio do “Tudo ou Nada”
* Ou todas as operações de transação são efetivadas (commit) com sucesso no BD ou nenhuma delas se efetiva (roolback)

C 🡪 Consistência

* Uma transação sempre conduz o BD de um estado consistente para outro estado também consistente
* É responsabilidade do SGBD garantir que todas as transições de estados validos
* Caso se viole a integridade, o SGDB deve desfazer a transação que violou a integridade

I 🡪 Isolamento

* Quando duas transações estão ocorrendo de forma concorrente, a execução da transação T1 deve funcionar como se a transação T2 não existisse
* T1 não deve sofrer interferências de T2 ou outras transações ocorrendo concorrentemente
* É responsabilidade do scheduler do SGBD, garantir escalonamentos sem interferências

Tabela

Descrição gerada automaticamente

D 🡪 Durabilidade/Persistência

* Deve-se garantir que as modificações realizadas por uma transação que concluiu com sucesso persistam no BD
* Se ocorrer uma falha no BD, esses dados não devem ser perdidos
* É responsabilidade do subsistema de recovery refazer as transações que executaram com sucesso em caso de falha no BD

Tabela

Descrição gerada automaticamente com confiança média

Serialização

777

O SGDB controla por serialização a execução concorrente de transações para assegurar que o estado do banco de dados permaneça consistente, ela é assegurada garantindo-se que a escala de execução deve ser, de alguma forma, equivalente a uma escala sequencial

Existem dois tipos de serialização:

* Por conflito
* Por visão

Por conflito:

Considere duas instruções I1 e I2 das transações T1 e T2 respectivamente, se I1 e I2 referem-se a dados diferentes é permitido alternar sem afetar o resultado, se I1 e I2 referem-se aos memos dados então a ordem pode importar

* I1 = read(Q) e I2 = read(Q) 🡪 **a ordem não importa**
* I1 = read(Q) e I2 = write(Q) 🡪 **a ordem importa**, se I1 vier antes de I2, então T1 não lê o valor escrito por T2, se I2 vier antes de I1, então T1 lê o valor escrito por T2
* I1 = write(Q) e I2 = read(Q) 🡪 **a ordem importa**, por razões iguais a anterior
* I1 = write(Q) e I2 = write(Q) 🡪 **a ordem não importa**, porém o valor obtido pelo próximo read(Q) é afetado, ele irá ler o valor da última instrução write(Q)

Exemplo de serialização:

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Interface gráfica do usuário, Diagrama, Aplicativo

Descrição gerada automaticamente

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Controle de Concorrência

**Locks**

Obriga que o acesso de dados seja feito de forma exclusiva, preservando o ISOLAMENTO

Enquanto uma transação acessa um dado, nenhuma outra pode acessar o dado, a transação deve solicitar o bloqueio de forma apropriada

O gerenciador de concorrência conde o bloqueio para a transação, ele pode ter de esperar

**Tipos de Bloqueios:**

Compartilhado lock-s(Q) 🡪 Se uma transação obtiver o bloqueio compartilhado, então ela pode ler Q mas não escrever

Exclusivo lock-x(Q) 🡪 Se uma transação obteve bloqueio exclusivo, então só ela pode ler e escrever Q

Unlock(Q) 🡪 a transação desbloqueia Q

Protocolo 2PL (duas fases)

* É feito em duas fases:
  + Fase 1 (expansão) 🡪 a transação solicita bloqueios, mas não os liberas
  + Fase 2 (encolhimento) 🡪 a transação libera os bloqueios, mas não obtém nenhum novo bloqueio

Linha do tempo

Descrição gerada automaticamente com confiança média

Linha do tempo

Descrição gerada automaticamente

BASE

BA (Basicamente Disponivel) 🡪 a disponibilidade é a prioridade

S (Estado Leve) 🡪 não precisa ser consistente o tempo todo

E (Eventualmente Consistente) 🡪 consistente em algum momento determinado

ACID X BASE

Interface gráfica do usuário, Tabela

Descrição gerada automaticamente

Teorema CAP

C (consistência) 🡪 O sistema garante a leitura do dado mais atualizado

A (disponibilidade) 🡪 A aplicação continuará disponível independentemente de cair/offline/falha

P (Tolerância ao particionamento) 🡪 Continua operando mesme que aconteça alguma falha de rede exemplo: um servidor de DB nos EUA fica sem comunicação com um servidor de DB na Índia

Diagrama

Descrição gerada automaticamente

Teorema CAP: é impossível garantir essas três propriedades ao mesmo tempo, mas é possível garantir quaisquer duas propriedades ao mesmo tempo

Diagrama, Diagrama de Venn

Descrição gerada automaticamente

Sistemas CAP

**Sistemas CA (Consistência e Disponibilidade):**

* Consistência forte e alta disponibilidade
* Não sabem lidar com a possível falha de uma partição
* Caso ocorra a falha o sistema fica indisponível até o membro do cluster voltar
* Exemplos: BD Relacionais Centralizados

Sistemas CP (Consistência e Particionamento):

* Consistência forte e tolerância ao particionamento
* O sistema fica operante se houver particionamento, mas demora para responder às requisições, ou nunca as receber
* Exemplos: Google BigTable, HBase ou MongoDB
* Consistência por meio de Quóruns (quantidade de votos que uma operação precisa receber para poder ser aplicada)
* Quórum de Escrita 🡪 O número de nós que confirmam uma escrita precisa ser maior que a metade do número de réplicas do item a ser escrito, ou seja, maioria
* Para a escrita ser com consistência forte 🡪 W > N/2, onde W = quórum de escrita e N = fator de replicação dos dados
* Quórum de Leitura 🡪 Para garantir leituras altamente consistentes, a seguinte fórmula precisa ser respeita 🡪 R+W > N onde R = quórum de leitura, W = quórum de escrita e N = fator de replicação dos dados

Sistemas AP (Disponibilidade e Particionamento):

* Alta disponibilidade, com tolerância a particionamento, nunca ficam offline
* Sempre disponível para escrita e depois sincroniza os dados

Diagrama

Descrição gerada automaticamente

Tipos de Banco de Dados não relacionais

* **Documentos**
  + Armazenam os dados em formato de documentos, geralmente JSON, BSON ou XML
  + Exemplos: MongoDB, Couchbase, CouchDB
  + Quando usar: Útil para cenários em que os dados são representados em formato de documento, como informações de produtos, perfis de usuários e configurações
  + Vantagens:
    - Flexibilidade de esquema: Cada documento pode ter uma estrutura diferente
    - Escalabilidade: Pode lidar com grandes volumes de dados e distribuição em várias máquinas
    - Consultas eficientes: Suporta consultas complexas usando índices
* **Colunas**
  + Armazenam dados em colunas, em vez de linhas, otimizando consultas que envolvem operações em colunas específicas
  + Exemplos: Apache Cassandra, HBase
  + Quando Usar: Adequado para aplicações que exigem consultas eficientes em colunas específicas de um grande conjunto de dados, como análise de dados e series temporais
  + Vantagens:
    - Alta performance em leitura e consulta de colunas específicas
    - Escalabilidade horizontal: Pode lidar com grande volume de dados e distribuição
    - Compressão eficiente: Armazena os dados de forma compacta
* **Grafos**
  + Modelam as relações entre os dados na forma de grafos, sendo úteis para representar redes complexas de interconexões
  + Exemplos: Noe4j, Amazom Neptune, OrientDB
  + Quando usar: Ótimo para representar e analisar relações complexas entre entidades, como redes sociais, rotas de transporte e interações de sistemas
  + Vantagens:
    - Modelagem natural de relações: Representa conexões entre dados de forma intuitiva
    - Desempenho em consultas de relacionamentos: Eficiência em recuperar e explorar relações
    - Flexibilidade: Adapta-se bem a estruturas de dados variadas
* Chave-valor
  + Armazenam dados em pares de chave-valor, sendo extremamente eficientes para operações simples de busca e escrita
  + Exemplos: InfluxDB, Prometheus, OpenTSDB
  + Quando usar: Ideal para aplicativos que necessitam de operações de busca e gravação rápidas e simples, como armazenamento de sessões, caches e metadados
  + Vantagens:
    - Alta performance: Rápido acesso por meio de chaves
    - Escalabilidade: Distribuição eficiente dos dados
    - Simplicidade: Estrutura básica de chave-valor é fácil de entender e usar