**Transações Concorrentes:** Concorrência entre transações é necessária para melhorar o desempenho; aproveita-se o tempo que uma transação permanece bloqueada para leitura e escrita no disco para processar outras transações; Evita que transações muito longas retardem a execução de transações mais curtas. Quando diversas transações são executadas de forma concorrente em um banco de dados, a propriedade do isolamento pode não ser preservada. É necessário que o sistema controle a interação entre transações concorrentes. Esse controle é alcançado por mecanismos chamados de esquemas de controle de concorrência. Subsistema que garante a seguinte propriedade das transações: **Isolamento**: a execução da transação não deve ser afetada pela execução concorrente de outras transações. Por quê fazer controle de concorrência? Transações submetidas por vários usuários podem ser executadas concorrentemente e podem acessar os mesmos itens do banco de dados. Se esta execução for descontrolada, pode ocorrer problemas. O subsistema de controle de concorrência precisa gerenciar o escalonamento das transações concorrentes. escalonamento é quando se pega um conjunto de transações e define a ordem em que elas vão ser executadas, para prover concorrência no BD. No escalonamento somente as operações de read\_item(X, xi) e write\_item(X,xi) são importantes. **Escalonamento Serializável**: Garante que qualquer escalonamento produzido ao se processar um conjunto de transações concorrentemente, seja computacionalmente equivalente a um escalonamento executando essas transações serialmente em alguma ordem. **Problemas Clássicos de Concorrência:** Na prática, são quatro os problemas de concorrência avaliados: Atualização Perdida (Lost Update). Leitura Suja (Dirty Read). Leitura Fantasma (Phatom Read): Novos registros “aparecem” em segunda leitura da tabela, ou seja, uma das transações faz INSERT. Leitura não-repetitiva (Nonrepeatable read): Os valores do registro são “alterados” em segunda leitura da tabela, ou seja, uma das transações faz UPDATE. **Controle de Concorrência:** Técnicas usadas para garantir a propriedade de não interferência ou isolamento das transações executadas concorrentemente. A maior parte dessas técnicas garante a serialização de schedules usando protocolos de controle de concorrência. Técnicas: 1- Bloqueio (tranca) de itens de dados: É utilizado para evitar que múltiplas transações acessem os itens de dados concorrentemente. **Pode ser Binário ou Compartilhados/Exclusivos**. 2- Pré-ordenação (timestamp): O sistema gera um identificador único (timestamp) para cada transação de forma que les fiquem ordenadas. Se Ti começa antes de Tj então Ts(Ti) < Ts(Tj). 3- Multiversão: Utiliza múltiplas versões dos itens de dados. 4- Otimista: Baseado no conceito de validação (certificação) de uma transação. **Bloqueio:** Um bloqueio é uma variável associada a um item de dados que descreve o status do item em relação a possíveis operações que podem ser aplicadas a ele. Em geral, existe um bloqueio para cada item de dados no banco de dados. Os bloqueios são utilizados como um meio de sincronizar o acesso por transações concorrentes aos itens do banco de dados. Bloqueio binário: Um bloqueio binário pode ter dois estados ou valores: bloqueado (1) e desbloqueado (0). Cada item do BD tem um valor associado. Se o valor for 1, o item não pode ser acessado por uma operação que o requisite. A transação bloqueia objeto (lock\_item()) antes de acessá-lo, liberando-o (unlock\_item()) antes de terminar. Operações: lock\_item(X): Bloqueia o item X, aplicada antes de qualquer operação read\_item e write\_item. unlock\_item(X): Desbloqueia o item X, aplicada depois de qualquer operação read\_item e write\_item. Observações: As operações lock e unlock devem ser incluídas nas transações quando se utiliza o protocolo do bloqueio binário. Uma transação T deve executar uma operação de lock\_item(X) antes de qualquer operação de read\_item(X) ou write\_item(X) ser executada em T. Uma transação T deve executar uma operação de unlock\_item(X) depois de qualquer operação de read\_item(X) ou write\_item(X) ter sido executada completamente em T. Uma transação T não vai executar a operação de lock\_item(X) se o item X já estiver bloqueado. Uma transação T não vai executar a operação de unlock\_item(X) se o item X já estiver desbloqueado. Bloqueios binários são simples de implementar, mas muito restritivos para fins de controle de concorrência porque, no máximo, uma transação pode manter um bloqueio em determinado item. Não são usados na prática. **Bloqueios compartilhados e exclusivos**: Operações de leitura no mesmo item por diferentes transações não estão em conflito. Assim, uma transação que utiliza bloqueio compartilhado/exclusivo utiliza bloqueios diferenciados para leitura e escrita de itens. Um bloqueio associado a um item X, lock\_item(X), agora tem três estados: Bloqueado para leitura, bloqueado para escrita e desbloqueado. Bloqueio conhecido também como multi-modo ou leitura/gravação. Bloqueio compartilhado: Bloqueia itens a serem lidos, mais de uma transação pode requerer um bloqueio compartilhado para ler um item X e Nenhum bloqueio de escrita pode ser requerido por outra transação. Bloqueio exclusivo: Para bloquear itens a serem escritos, pode existir somente um bloqueio de escrita de um item X e Nenhum bloqueio compartilhado pode ser requerido por outra transação. Operações: read\_lock(X): Bloqueia x na modalidade compartilhada. Aplicada antes de qualquer operação read\_item e write\_item: Abreviação-rl. write\_lock(X): Bloqueia x na modalidade exclusiva. Aplicada antes de qualquer operação read\_item e write\_item: Abreviação – wl. unlock\_item(X): Desbloqueia o item X. Aplicada depois de qualquer operação read\_item e write\_item. Abreviação – ul. Não é permitida a intercalação de read\_lock (X) e write\_lock (X) de diferentes transações deve usar fila de espera. Para uma transação T utilizar o bloqueio compartilhados e exclusivos: T deve emitir um rl(X) ou wl(X) antes de qualquer read\_item(X) em T. T deve emitir um wl(X) antes de qualquer write\_item(X) em T. T deve emitir ul(X) depois de todo read\_item ou write\_item em T. T não emitirá um rl(X) se já apareceu um wl(X) em T. T não emitirá um wl(X) se já apareceu um rl(X) ou wl(X) em T. T não emitirá um ul(X) antes que apareça um rl(X) ou wl(X) em T. Protocolo de bloqueio em duas fases – 2PL: O uso de bloqueios binário ou de leitura/gravação não garante a serialização de escalonamentos por si só. Para garantir a serialização, temos que seguir um protocolo adicional em relação ao posicionamento das operações de bloqueio e desbloqueio em cada transação. O protocolo mais conhecido é o bloqueio em duas fases. Diz-se que uma transação segue o 2PL se todas as operações de bloqueio (read\_lock, write\_lock) precedem a primeira operação de desbloqueio da transação. Transação bloqueia objeto, na modalidade correta, antes de acessá-lo, liberando-o antes de terminar. Após liberar o primeiro objeto, a transação não bloqueia nenhum outro objeto. Ponto de Bloqueio de T - pb(T): instante em que a transação libera o primeiro objeto. Por quê duas fases? Fase de expansão ou crescimento: Novos bloqueios podem ser adquiridos, mas nenhum liberado e Fase de encolhimento: Bloqueios existentes podem ser liberados, mas nenhum novo bloqueio pode ser adquirido. Se toda transação em um escalonamento segue o protocolo 2PL, então a transação é serializável. Embora o protocolo 2PL garanta a serializabilidade, bloqueios podem gerar dois problemas: Rollback em cascata e Impasse (deadlock). **TRATAMENTO DE DEADLOCK:** Um sistema está em um estado de impasse se houver um conjunto de transações tal que cada transação no conjunto está esperando por outra transação no conjunto. Granularidade de Bloqueio: A granularidade de bloqueio indica o nível de utilização do bloqueio: Nível de banco de dados, Nível de tabela, Nível de linha e Nível de atributo. **O tratamento** de DeadLock pode ser feito de duas maneiras: 1- Prevenção de Impasse e 2-Detecção e Recuperação de Impasse. **Detecção e Recuperação de deadlock**: Não evita os deadlocks, mas os detecta e impede o bloqueio indefinido das transações envolvidas. Mais eficiente se ocorrerem poucos deadlocks. O Rollback pode ser necessário independentemente da técnica utilizada! **Prevenção de Impasse**: Evita os deadlocks antes que eles ocorram. Preferível se a probabilidade de ocorrerem deadlocks for muito alta. Existem duas técnicas**: 1. Assegura que nenhuma espera cíclica poderá ocorrer.** Exige que a transação bloqueie todos os itens antes de iniciar. Ordena os itens de dados e exige que uma transação bloqueie itens somente em uma sequência consistente com a ordenação. 2**. Usar preempção e rollbacks de transação**: Esquema esperar-morrer (wait-die): Apenas as transações mais velhas (timestamp) têm permissão para esperar outra transação desbloquear um item de dado. As demais são abortadas (rollback). Esquema ferir-esperar (wound-wait): Apenas as transações mais novas(timestamp) têm permissão para esperar outra transação desbloquear um item de dado. As demais são abortadas (rollback). Detecção e Recuperação de Deadlock: O sistema precisa processar as duas etapas: 1. Detectar se o sistema está em deadlock: Grafo de Espera. Se houver ciclos no grafo, o sistema está em deadlock. 2. Recuperar o Sistema: Selecionar a vítima: Livelock (O sistema tem dificuldades para selecionar a vítima). Fazer o Rollback: É preciso reverter aquelas transações que incorrerão em custo mínimo (Rollback parcial). Esse sistema pode levar uma transação a um estado chamado Estagnação (starvation), pois uma mesma transação sempre é escolhida como vítima, sendo sempre adiada, de forma que nunca é executada. **Controle de Concorrência Multiversão**: Multiversion Concurrency Control, MVCC. Cópias do mesmo registro para um conjunto de transações. As Transações divididas entre transações de leitura e de escrita: as Transações de escrita criam cópias do dado e as Transações de leitura só leem dados comitados. Operações de leitura NÃO bloqueiam operações de escrita e vice-versa. As Operações que alteram a estrutura da tabela precisarão bloquear qualquer operação de leitura/escrita concomitante. **Recuperação baseada em log:** Um sistema de computação, como qualquer outro dispositivo elétrico ou mecânico, está sujeito a falhas, como: quebra do disco, queda da energia elétrica, erros de software, fogo na sala da máquina, sabotagem, etc. Essas falhas podem levar a perda de informações no banco de dados ou torná-lo inconsistente. Como garantir a consistência do banco nessas situações? **Recuperação de Falhas x Restauração:** O subsistema de recuperação contra falhas (recovery) do SGBD age quando o sistema ‘cai’ e, ao voltar, o banco de dados continua íntegro, ou seja, falha APENAS no conteúdo de armazenamento volátil. Mas se houver danos no armazenamento não volátil será necessário restaurar o banco utilizando um backup que é uma Tarefa do DBA. Recuperação de Falhas: Restaurar o BD ao estado consistente mais recente antes do momento de falha. Garantir as propriedades de ATOMICIDADE e DURABILIDADE. COMO? O sistema deve manter informações sobre as mudanças que foram aplicadas aos itens de dados pelas diversas transações: Recuperação baseada em log ou Paginação com imagem. **Recuperação baseada em log:** Protocolo Write-Ahead Logging (WAL): Grava registro de operação no disco de log antes que a modificação do item seja gravada em disco, garante atomicidade. Todas as operações de uma transação são gravadas no disco de log antes do commit: garante durabilidade. Log de Transações: O log é um arquivo sequencial, apenas para inserção, que é mantido no disco, de modo que não é afetado por qualquer tipo de falha, exceto por falha no disco ou falhas catastróficas. recomenda-se um disco separado. O SGBD utiliza o log de transações para rastrear todas as transações que atualizam o banco de dados. OBJETIVO: Gravar informações descrevendo as modificações no armazenamento estável sem modificar o banco de dados em si. POR QUÊ? Evitar operações de I/O no disco. **Hierarquia de Armazenamento:** As transações transferem blocos de informações do disco para a memória principal e depois os devolvem para o disco. Os blocos residentes no disco são chamados blocos físicos, e os residentes na memória, de blocos de buffer. Tipos de Armazenamento: Armazenamento volátil: Memória principal e cache. Armazenamento não-volátil: Discos e fitas e Armazenamento estável: Duplicar informações em diversos meios não-voláteis, com modos independentes de falhas, e atualizar a informação de uma maneira controlada. Informação armazenada em RAM é perdida se faltar energia ou se a máquina falhar. Informação armazenada em RAM é perdida se faltar energia ou se a máquina falhar. Informação em armazenamento estável sobrevive a tudo, exceto enchentes, terremotos, ... **Recuperação de Falha:** O processo de recuperação de falhas está estreitamente ligado a funções do sistema operacional. Cache do SGBD: Baseado em páginas de disco mantidas pelo SO. SGBD chama rotinas de baixo nível do SO. Essencial para desempenho, porém adiciona complexidade aos mecanismos de log. Recuperação de Falhas Gravação de Cache no Disco: **Uma página do cache atualizada pode ser gravada em disco ANTES do COMMIT da transação?** **Sim: Steal. Não: No Steal**. **Todas as páginas atualizadas por uma transação são IMEDIATAMENTE gravadas em disco quando a transação atinge seu ponto de confirmação? Sim: Force. Não: No Force**. Existem três algoritmos diferentes para recuperar o BD após uma falha: 1- Log com modificações adiadas: No Steal/Force. 2- Log com modificações imediatas: Steal/Force ou Steal/No Force e 3- Paginação com imagem. **Estrutura do log:** Cada registro do log descreve uma única gravação no banco de dados e tem os seguintes campos: Nome da transação: Só aquelas que executaram a operação write. Nome do item de dado: Atributo que será modificado. Valor antigo e Novo valor. O que aconteceria na transação de transferência bancária se o sistema caísse: 1-No meio da transação e 2- No final da transação, mas antes do output. Os SGBDs trabalham com dois algoritmos: **Modificação adiada e Modificação imediata**. Usando o log, o sistema pode manipular qualquer falha que resultar na perda de informações em dispositivo de armazenamento volátil. Dada uma falha, o sistema de recuperação consulta o log para determinar o que fazer com as transações. Modificação adiada: Refaz ou Ignora e Modificação imediata: Desfaz ou Ignora. Modificação do banco de dados **adiada**: Durante a execução de uma transação, todas as operações write são adiadas até que a transação seja parcialmente compromissada. Todas as atualizações são registradas no log, que precisa ser mantido em meio de armazenamento estável. Quando uma transação é parcialmente compromissada, a informação do log é usada na execução das gravações no disco. Na modificação adiada, o sistema varre o log e procura por transações onde: O log possui o registro <T, start> e o registro <T, commits> Ou seja, **transações terminadas e não comitadas** (Para comitar de fato no disco precisa finalizar o output). Então essas **transações parcialmente compromissadas são REFEITAS. As demais são IGNORADAS**, ou já terminaram e estão salvas no banco ou não terminaram e, por isso, não tem como refazer. Operação Redo: Redo(Ti ) Refaz Ti , ajustando o valor de todos os itens de **dados atualizados** pela transação ti, **para os novos valores**. Modificação **imediata** do banco de dados: Permite a gravação de modificações no banco de dados enquanto a transação ainda está no estado ativo. As modificações gravadas por transações ativas são chamadas modificações descompromissadas (uncommitted). Na modificação imediata, o sistema varre o log e procura por transações onde: O log possui o registro <T, start> e NÃO POSSUI o registro <T, commits>, ou seja, **transações não terminadas. Essas transações são DESFEITAS. As demais são IGNORADAS**, pois já terminaram e estão salvas no banco. Operação Undo: undo(Ti ): Se ocorrer uma falha, o campo com o **valor antigo** dos registros do log deve ser usado para **restaurar** os itens de dados modificados com **os valores que tinham antes do início da transação**. Pontos de Verificação: Quando uma falha no sistema ocorre, é necessário consultar o log para determinar aquelas transações que precisam ser refeitas e aquelas que precisam ser desfeitas. Em princípio, o log inteiro precisa ser varrido: Consumo de tempo e a maioria das transações já foram comitadas. SOLUÇÃO: Pontos de Verificação (checkpoints). O sistema mantém o log usando uma das duas técnicas descritas. Adiada ou Imediata. Adicionalmente, o sistema estabelece periodicamente pontos de verificação <checkpoint>. Abordagem mais comum de recovery: Abordagem mais comum de recovery. Utiliza técnica STEAL - imediata: Duas técnicas: UNDO/NO-REDO ou UNDO/REDO. Modificação imediata **UNDO/REDO**: Utiliza 2 listas de transações. Lista-REDO: Contém a identificação das transações que possuem o commit gravado no LOG e Lista-UNDO: Contém a identificação das transações ativas. Técnica mais trabalhosa de recovery: tanto UNDO quanto REDO devem ser realizados. Porém, o gerenciamento de buffer é mais simples. Os SGBDs criam estratégias: Transações curtas -> modificação adiada e Transações longas -> modificação imediata. Modificação Adiada: Demanda muito espaço de cache, costuma ser inviável na prática e Evita operações desnecessárias no disco. Modificação Imediata: Não há necessidade de fazer REDO. Mas Pode-se fazer UNDO de uma transação que foi gravada com sucesso no BD, porém não foi gravado a tempo o seu commit no Log. **Paginação com imagem:** Duas tabelas são mantidas durante a existência de uma transação: A tabela corrente e a tabela imagem. Quando a transação se inicia, ambas são idênticas. A tabela imagem nunca é modificada durante a transação e a corrente pode ser alterada quando uma transação executa uma operação write. Todas as operações input e output usam a tabela corrente. Quando uma transação é parcialmente compromissada, a tabela imagem é descartada e a corrente torna-se a nova tabela no disco. Se a transação é abortada, a tabela corrente é simplesmente descartada. **Bancos de Dados Objeto-Relacionais:** Os BDOR modelam objetos armazenados em tabelas. Utilizam as tabelas do modelo relacional, mas nelas são armazenados objetos, com seus atributos e comportamentos, unindo assim os paradigmas. Os dados são armazenados em relações, mas pode-se armazenar dados complexos abstraindo seu comportamento da mesma forma como é feito na orientação a objetos. Utiliza os conceitos de supertabelas, supertipos, herança, reutilização de código, encapsulamento, controle de identidade de objetos (OID), referência a objetos, consultas avançadas e alta proteção dos dados. A área de atuação dos SGBDs ObjetoRelacionais tenta suprir a dificuldade dos sistemas relacionais convencionais, que é o de representar e manipular dados complexos. SGBDOR PostgreSQL: No PostgreSQL, tabelas, relacionamentos, restrições e triggers são considerados objetos. Mas não são os “objetos” tais quais os das linguagens de programação. É possível utilizar os conceitos de herança, polimorfismo e object ids. No caso do PostgreSQL, o “objeto” de “objeto-relacional” está ligado a todos os aspectos citados anteriormente, mais a organização dos “recursos”, como na utilização de schemas, e na criação de tipos e manipulação de dados complexos. Tipo Composto: Os dados complexos são implementados no PostgreSQL por meio do chamado tipo composto; O tipo composto descreve a estrutura de uma linha ou registro; essencialmente, é apenas uma lista de nomes de campos com seus tipos de dado. Array: Bancos objeto-relacionais também implementam o conceito de array para enriquecer a semântica do dado. No PostgreSQL inclusive arrays multidimensionais são permitidos. Herança: é um conceito da orientação à objetos onde, uma determinada classe herda características (no caso das classes - propriedades ou atributos e métodos ou funções) de uma outra "super" classe. A relação de herança pode ser lida da seguinte forma: "a classe que herda É UM TIPO DA super classe (classe herdada)". Ex: Capitais herda tabela cidade, assim Sempre que houver uma atualização na tabela capitais (insert, update e delete), o mesmo registro é atualizado também na tabela cidades. O contrário não é verdadeiro. Em select \* From Only cidade O termo "ONLY" antes de cidades indica que a consulta deve ser executada apenas na tabela cidades, sem incluir as tabelas descendentes de cidades na hierarquia de herança ( SELECT, UPDATE, DELETE). Limitações: A PRIMARY KEY não impede que a tabela ‘capitais’ tenha linhas com identificadores idênticos aos da tabela cidades e, por padrão, estas linhas duplicadas aparecem nas consultas à tabela cidades. Uma FOREIGN KEY definida na tabela cidades não se propagará automaticamente para a tabela capitais e nem vice-versa. Especificar para uma coluna de outra tabela REFERENCES cidades(idCidade) permite à outra tabela conter os nomes das cidades, mas não os nomes das capitais. Não existe uma maneira boa para contornar este problema. **Big Data**: Big data é um termo que se refere a conjuntos de dados extremamente grandes e complexos, que não podem ser facilmente tratados com métodos tradicionais de processamento. Esses conjuntos de dados geralmente apresentam alta variedade, velocidade e volume, exigindo ferramentas e técnicas especiais para armazenamento, análise e extração de informações significativas. O objetivo do big data é obter insights e conhecimentos valiosos a partir desses dados, a fim de tomar decisões mais informadas e direcionadas. 3V- Volume, velocidade e variedade. 5v: acrescentado veracidade e valor. Os "5 V's" do big data são um conjunto de características que descrevem os principais aspectos dos conjuntos de dados volumosos e complexos: Volume: refere-se à quantidade massiva de dados gerados e armazenados, que ultrapassa a capacidade dos métodos tradicionais de processamento. Velocidade: diz respeito à taxa em que os dados são gerados e precisam ser processados em tempo real ou próximo a ele. Variedade: abrange a diversidade dos tipos de dados, incluindo estruturados, não estruturados e semiestruturados, como texto, imagens, vídeos, áudios, redes sociais e sensores. Veracidade: refere-se à qualidade e confiabilidade dos dados, garantindo que sejam precisos, consistentes e livres de erros. Valor: está relacionado à capacidade de extrair insights significativos e informações úteis a partir dos dados para tomar decisões estratégicas e obter vantagens competitivas. Esses 5 V's são essenciais para compreender a natureza e os desafios do big data e para implementar estratégias eficazes de armazenamento, processamento e análise desses conjuntos de dados. O big data suporta uma ampla variedade de tipos de dados, incluindo: Dados estruturados: São dados organizados em um formato tabular com campos pré-definidos e relacionamentos claros. Exemplos incluem bancos de dados relacionais, planilhas e arquivos CSV. Dados não estruturados: São dados sem um formato definido ou organização predefinida. Isso inclui texto livre, documentos, e-mails, arquivos de mídia (como imagens, vídeos e áudios), dados de redes sociais, logs de servidores, entre outros. Dados semiestruturados: São dados que possuem algum nível de estrutura, mas não seguem um esquema rígido. Exemplos comuns incluem arquivos JSON, XML, arquivos de registro (logs) e dados de sensores. Dados de streaming: São dados que chegam em tempo real, como feeds de mídia social, transmissões de sensores, dados de dispositivos conectados à Internet das Coisas (IoT), transações financeiras em tempo real, entre outros. O big data é capaz de lidar com esses diferentes tipos de dados e extrair informações relevantes deles, permitindo análises abrangentes e insights significativos.Parte superior do formulário

Parte inferior do formulário

O predomínio do big data representa três mudanças na forma como analisamos informações que transformam a maneira como entendemos e organizamos a sociedade: Usar o conjunto total de dados, e não amostragem. Maior % de erros e Afastamento da busca pela causalidade. No mundo do big data, por sua vez, não temos de nos fixar na causalidade: podemos descobrir padrões e correlações nos dados que podem não nos dizer com exatidão por que algo está acontecendo, mas nos alertam que algo está acontecendo. Tecnologias Associadas ao Big Data: Sistemas Distribuídos, Computação em Nuvem , Virtualização, Armazenamento. **MapReduce:** Paradigma de programação para gerenciar grandes quantidades de dados (mais que 1TB), popularizado pelo Google em 2004. O principal interesse desse paradigma é que as duas primitivas Map e Reduce, sejam facilmente paralelizadas e capazes de lidar com a grande quantidade de dados. Todos os programas desenvolvidos sobre esse paradigma realizam o processamento paralelo de conjuntos de dados e podem, portanto, ser executados em servidores simples, sem muito esforço. A primitiva Map consiste no processamento de uma lista de dados, a fim de criar pares chave/valor. A primitiva Reduce irá processar cada par, a fim de criar um novo agrupamento chave/valor. Em geral, a primitiva Reduce utiliza alguma função de agregação (SUM, AVG, etc). o **MapReduce** é um modelo de programação e processamento distribuído que permite lidar com grandes volumes de dados de forma eficiente. Ele divide tarefas complexas em etapas simples de mapeamento e redução. O mapeamento envolve a transformação dos dados em pares chave-valor, enquanto a redução agrega e combina esses pares para produzir resultados significativos. O MapReduce é usado principalmente em sistemas de processamento de big data para paralelizar o processamento e aumentar a velocidade de análise de grandes conjuntos de dados, aproveitando clusters de computadores. Diferenças: **Big data**: Refere-se ao conjunto de dados extremamente grandes e complexos que requerem técnicas especiais para armazenamento, processamento e análise, a fim de extrair insights e conhecimentos valiosos. **Data science:** É um campo multidisciplinar que combina habilidades de estatística, programação e conhecimento de domínio para extrair insights, padrões e conhecimentos dos dados, usando métodos estatísticos, algoritmos de machine learning e técnicas de visualização. **Business intelligence (BI**): É um processo de coleta, análise e apresentação de informações relevantes para auxiliar na tomada de decisões empresariais. Ele usa dados históricos e atuais para fornecer insights sobre o desempenho da organização e identificar tendências e oportunidades de negócios. **Data warehouse**: É um repositório centralizado que armazena dados de várias fontes diferentes em um formato estruturado e otimizado para análise. Ele é projetado para suportar consultas complexas e fornecer acesso rápido aos dados para suportar a tomada de decisões estratégicas. O data warehouse geralmente é alimentado por dados de várias fontes e é usado como base para análises de business intelligence. Resumidamente, big data refere-se aos grandes e complexos conjuntos de dados, data science é a disciplina que utiliza métodos e técnicas para extrair insights desses dados, business intelligence é o processo de análise e interpretação dos dados para auxiliar na tomada de decisões empresariais e data warehouse é um repositório centralizado otimizado para análise de dados de diferentes fontes. O Apache Spark quanto o Hadoop são softwares de código aberto. Eles são considerados frameworks ou plataformas para processamento e análise de big data. Essas tecnologias fornecem um conjunto de ferramentas, bibliotecas e APIs para facilitar o processamento distribuído, escalável e eficiente de grandes volumes de dados em clusters de computadores.

**NoSQL:** os SGBDR não dão correto suporte aos 3Vs do Big Data. Volume e Velocidade (para escabilidade) e Variedade (esquema rígido do R). Limitações dos bancos relacionais para armazenamento de Big Data: Leitura e escrita lenta com o aumento do tamanho dos dados, é possível provocar bloqueios e outros problemas de concorrência, o que leva a um rápido declínio na eficiência da leitura e escrita. Capacidade limitada: bancos de dados relacionais atuais não suportam um volume grande de dados no motor de busca. Dificuldade de expansão: mecanismos de correlação entre múltiplas tabelas, devido ao uso de chaves estrangeiras existentes em bancos de dados relacionais, não oferecem um bom mecanismo para a escalabilidade, sendo este um fator crucial para atender aos requisitos da Big Data. Dificuldade de indexação: devido a grande quantidade de dados, indexá-los torna-se impraticável, pois o espaço ocupado pelos índices pode chegar a tamanhos críticos, o que leva a uma perda de desempenho. Escalabilidade: Escalar significa quão bem uma solução particular se encaixa em um problema enquanto o escopo do problema aumenta. A medida que o banco cresce, o servidor precisa aumentar sua capacidade de processamento, armazenamento, memória, etc. Problema: O aumento de utilização de recursos computacionais causado pelo aumento de acessos e/ou aumento do tamanho físico do banco. Soluções: 1-Escalabilidade Verticale 2-Escalabilidade Horizontal. Escalar verticalmente refere-se a aumentar a capacidade de processamento, armazenamento ou recursos de um único servidor ou máquina, geralmente adicionando mais recursos, como CPU, RAM ou disco rígido, a um servidor existente. Isso é feito para melhorar o desempenho e lidar com cargas de trabalho maiores. Escalar horizontalmente, por outro lado, envolve adicionar mais servidores ou máquinas ao ambiente de computação, em vez de aumentar os recursos de uma única máquina. Isso permite distribuir a carga de trabalho entre várias máquinas, permitindo um processamento paralelo e lidando com volumes de dados maiores. O escalonamento horizontal é usado para melhorar o desempenho e a capacidade de lidar com demandas crescentes, adicionando mais nós de processamento em um cluster ou infraestrutura distribuída. Qual é o problema? Apesar da riqueza de recursos, os SGBDs relacionais tendem a aumentar a complexidade de utilização com o aumento do fluxo de dados. Eles não foram criados para trabalhar em ambientes distribuídos e o escalonamento, em geral, se dá pelo aumento da capacidade da máquina servidora (escalonamento vertical). Acontece que ao aumentar muito o fluxo de dados, o uso de clusters torna-se inevitável e o desempenho dos SGBDs relacionais cai. Um banco de dados relacional baseado em ACID não consegue facilmente escalar de forma horizontal devido sua arquitetura baseada em tabelas. Esquema Rígido: Um banco de dados relacional possui esquema definido, rígido, cuja alterações apresentam forte impacto na solução implementada. Aplicações ficam ‘presas’ ao esquema que o modelo relacional pode representar. São criadas entidades no modelo que não existem no mundo real. Mapeamento objeto-relacional é caro. Hoje, a programação orientada a objetos é o paradigma de programação mais prevalente. Uma aplicação persistindo objetos em um SGBDR requer um mapeamento entre o modelo do objeto e do modelo relacional. Definir isso toma tempo e processamento. O pensamento "One Size Fits All" é falho. Este pensamento quer dizer que os SGBDRs são vistos como um instrumento geral que pode lidar com todos os requisitos que uma aplicação pode ter no gerenciamento de dados. Uma vez que diferentes aplicações podem ter diferentes exigências sobre a consistência, desempenho e assim por diante, este pensamento pode ser falho. ACID não é sempre necessário. Em algumas aplicações, transações ACID não são necessárias, o que significa que pode se perder alguns trade-offs como desempenho. **NoSQL:** Não há definição rígida para o termo NoSQL - Not Only SQL. Surgiu em 1998 com a criação de uma aplicação de banco de dados que não oferecia uma interface SQL, mas que ainda era baseado no modelo relacional. Mais tarde o termo passou a representar soluções que promoviam uma alternativa ao Modelo Relacional. NoSQL, que significa "Not Only SQL" (Não Apenas SQL, em tradução livre), é um termo que se refere a um tipo de banco de dados não relacional. Diferentemente dos bancos de dados relacionais tradicionais, o NoSQL é projetado para armazenar e recuperar grandes volumes de dados não estruturados ou semiestruturados. Ele oferece maior flexibilidade e escalabilidade horizontal, permitindo o processamento eficiente de grandes quantidades de dados distribuídos em clusters de servidores. O NoSQL é usado em cenários onde é necessária alta velocidade de acesso, escalabilidade e flexibilidade de esquema, como em aplicações web, análises em tempo real e sistemas de gerenciamento de conteúdo**. Principais Características:** Não utilizam o modelo relacional e nem a linguagem SQL; são projetados para rodar em clusters (sistemas distribuídos); tendem a ser open source; não possuem um esquema fixo, permitindo a persistência em qualquer registro. O propósito das soluções NoSQL não é substituir o modelo relacional, mas sim serem utiizadas quando houver necessidade de **maior flexiblidade na estruturação do banco de dados.** Não é uma nova tecnologia, é uma nova abordagem. Propõe um modelo alternativo de banco de dados. Não é relacional. Não respeita as propriedades ACID. **Persistência Poliglota**: **Utilizar diferentes armazenamentos de dados em diferentes circunstâncias**. Em vez de escolher o banco de dados relacional mais utilizado por todos, precisamos entender a natureza dos dados que estamos armazenando e como queremos manipulá-los. Persistência poliglota é uma abordagem de desenvolvimento de software que **envolve a utilização de diferentes tecnologias e modelos de armazenamento de dados para atender aos requisitos específicos de uma aplicação.** **Em vez de utilizar um único sistema de banco de dados para todas as necessidades de persistência de dados, a persistência poliglota permite que diferentes tipos de dados sejam armazenados em sistemas especializados, como bancos de dados relacionais, bancos de dados NoSQL, armazenamento em memória ou até mesmo serviços de terceiros. Essa abordagem permite que as empresas aproveitem as características e vantagens de cada tecnologia para otimizar o armazenamento e o acesso aos dados, escolhendo a melhor opção para cada caso de uso específico.** Soluções NoSQL costumam buscar atingir objetivos como baixa latência, alta performance e escalabilidade. Para tanto seguem um conjunto de propriedades chamado BASE: Basically Available; Soft-state e eventual consistency. **Basically Available:** Basicamente Disponível ou Disponibilidade Básica. O sistema estará disponível basicamente todo o tempo. Determina que o sistema deve estar disponível na maior parte do tempo, mas que subsistemas podem estar temporariamente inoperantes. **Soft-state:** Estado leve e consistente. O estado do sistema não precisa estar sempre consistente. Os desenvolvedores devem criar mecanismos para consistências de dados. **Eventual consistency:** Consistência Eventual- O sistema torna-se consistente em um determinado momento. Garante que se certo registro não sofre atualizações, eventualmente estará consistente e todas as leituras retornarão a última atualização do valor. O NoSQL Armazena e replica dados em sistemas distribuídos, muitas vezes através de data centers, para alcançar escalabilidade e confiabilidade. Por ser distribuído, podem haver conflitos: De gravação: 2 clientes tentam gravar os mesmos dados ao mesmo tempo. De leitura-gravação: um cliente lê dados inconsistentes durante a gravação de outro cliente. Abordagens: Pessimista: bloqueiam registros de dados para evitar conflitos. Otimistas: detectam conflitos e os resolvem. Consistência: Às vezes é necessário sacrificar a consistência. Sempre é possível projetar um sistema para evitar inconsistências, mas, muitas vezes, é impossível fazê-lo sem sacrificar outras características. Consequentemente, muitas vezes é necessário balancear a consistência com algo diferente. Diferentes domínios têm diferentes tolerâncias à inconsistência, logo, precisamos levar essa tolerância em consideração ao tomarmos nossas decisões. Teorema CAP: Soluções: Designar um nó como mestre; Permitir que ambos os nós continuassem aceitando reservas, mesmo que a conexão caísse. Questões. Quão tolerante sua aplicação pode ser quanto a leituras desatualizadas? Quanto tempo sua aplicação tolera inconsistência? **Modelagem NoSQL**: Modelagem representa uma abstração da realidade. Representar o mundo real em um determinado modelo conceitual e físico (de armazenamento). Problemas do Modelo Relacional: 1-Normalização: Evita redundância e Espalha os dados pelo disco rígido -> maior tempo para realizar consultas e realizar alterações.2- Esquema Rígido: Utilização de valores Nulos e Grande impacto no caso de alteração do esquema .3-Problemas de semântica. Modelagem NoSQL: Schema-less. Não há esquema pré-definido. Modelagem baseada nas consultas da aplicação. Dados agregados: Redundância e Incorporação. A modelagem é focada nas consultas que serão executadas nos dados. Em contraste com o banco relacional, o NoSQL procura agrupar dados agregados juntos no disco. Reduz o número de requisições ao disco e Melhora tempo de acesso. (Modelo Relacional:A modelagem é decidida pela estrutura de dados). Embora esta nova geração de bancos de dados seja livre de esquema, a importância do modelo de dados será sempre entender e demonstrar o armazenamento de dados.

Exercicios:

Considera as duas transações a seguir: T31: read(A); read(B); if A = 0 then B := B + 1; write(B); T32: read(B); read(A); if B = 0 then A := A + 1; write(A);

1. Considerando a técnica de bloqueio compartilhado/exclusivo, adicione instruções de lock e unlock para as transações T31 e T32 de forma que obedeça ao protocolo de bloqueio em duas fases. A execução dessas transações de forma concorrente pode resultar em deadlock? Caso afirmativo, ilustre e justifique sua resposta.

|  |  |  |  |
| --- | --- | --- | --- |
| T31:  read\_lock(A);  read\_item(A);  write\_lock(B);  unlock(A);  read\_item(B);  if A = 0 then B := B + 1;  write\_item(B);  unlock(B); | T32:  read\_lock(B);  read\_item(B);  write\_lock(A);  unlock(B);  read\_item(A);  if B = 0 then A := A + 1; write\_item(A);  unlock(A); | T31:  L(A) - Bloqueia o item A em modo compartilhado (MC)  read(A);  L(B) - Bloqueia B em MC  read(B);  if A = 0 then B := B + 1;  write(B);  U(B) -Libera o bloqueio de B  U(A) -Libera o bloqueio de A | T32:  L(B) – Bloqueia B em modo compartilhado (MC)  read(B);  L(A) -- Bloqueia A em MC  read(A);  if B = 0 then A := A + 1;  write(A);  U(A) - Libera o bloqueio de A  U(B) - Libera o bloqueio de B |

Essas instruções de bloqueio garantem que os itens A e B sejam bloqueados corretamente durante a leitura e a escrita.

No entanto, a execução dessas transações de forma concorrente pode resultar em um deadlock. Isso ocorreria se, por exemplo, a transação T31 bloqueasse o item A e a transação T32 bloqueasse o item B. Em seguida, se T31 tentar bloquear o item B e T32 tentar bloquear o item A, ocorrerá um impasse (deadlock), já que cada transação está esperando que o item bloqueado pela outra transação seja liberado. Portanto, sim, é possível ocorrer um deadlock nessa situação, dependendo da ordem de execução das transações e dos bloqueios adquiridos por cada uma delas. ou

Caso as duas operações anteriores forem executadas concorrentemente poderá resultar em deadlock. Como as duas transações estão trabalhando com os mesmos itens (A e B), isso faz com que a T31 espera a T32 liberar algo que ela precise e vice versa (formando uma espera circular e causando deadlock). Por exemplo: a T31 bloquear A e esperar que a T32 desbloqueie B, e a T32 esperar que T31 desbloqueie A (com o B bloqueado).

1. Faça uma relação entre controle de concorrência e recuperação de falhas. O controle de concorrência refere-se ao gerenciamento de múltiplas transações concorrentes que acessam e modificam os mesmos dados simultaneamente. O objetivo principal do controle de concorrência é garantir que as transações sejam executadas corretamente e de forma consistente, evitando problemas como leituras sujas, escritas perdidas, anomalias de atualização e deadlocks. Ele envolve técnicas como bloqueios, controle de versões, serialização de transações e isolamento de transações. Por outro lado, a recuperação de falhas diz respeito à capacidade de um sistema de banco de dados de se recuperar de falhas, como panes de hardware, erros de software ou interrupções de energia. O objetivo principal da recuperação de falhas é garantir a durabilidade e a consistência dos dados, mesmo após a ocorrência de falhas. Isso é geralmente alcançado por meio da aplicação de técnicas de registro de transações, como log de transações, checkpoints e técnicas de replay de log, que permitem a recuperação do estado consistente do banco de dados antes da falha. Ambos os aspectos são fundamentais para garantir a integridade e a confiabilidade dos sistemas de banco de dados. O controle de concorrência ajuda a evitar inconsistências e conflitos durante a execução simultânea de transações, enquanto a recuperação de falhas garante que o sistema seja capaz de se recuperar e restaurar um estado consistente após uma falha. Juntos, eles contribuem para a confiabilidade e a disponibilidade dos dados em um ambiente de banco de dados. Ou...O controle de concorrência é responsável por controlar a interação entre as transações concorrentes a fim de manter a propriedade do isolamento. A recuperação de falhas (usando modificações imediata ou adiada) é responsável por restaurar o banco de dados ao estado de consistência mais recente quando ocorre uma falha. Usando a resposta da questão anterior, o controle de concorrência irá controlar a interação das transações T31 e T32 para que elas trabalhem de forma concorrente. Como ocorre uma falha (deadlock), a restauração irá realizar a detecção e a recuperação do deadlock (para que o banco volte a ficar consistente).

O código SQL abaixo ilustrado inicializa uma transação, faz uma consulta na tabela T, entra em um estado de espera por 1 minuto e, posteriormente, realiza uma nova consulta na tabela T.

BEGIN TRANSACTION;

SELECT \* FROM T;

WAITFOR DELAY ‘00:01:00’

SELECT \* FROM T; COMMIT;

Considerando possíveis transações concorrentes a esta acima ilustrada, que modifiquem registros na tabela T durante o minuto de espera, indique o resultado do segundo SELECT, para os seguintes níveis de isolamento no SGBD PostgreSQL: a) Read committed b) Repeatable read c) Serializable

Para o cenário descrito, onde outras transações podem modificar registros na tabela T durante o minuto de espera, os resultados do segundo SELECT podem variar de acordo com o nível de isolamento definido no PostgreSQL. Vamos analisar os resultados para cada um dos níveis de isolamento mencionados: a) **Read committed:** Nesse nível de isolamento, cada instrução **SELECT enxerga apenas as alterações já confirmadas** no banco de dados até aquele momento. **Portanto, o segundo SELECT retornará as modificações realizadas por outras transações que foram confirmadas antes do início do segundo SELECT**. b) **Repeatable read**: Nesse nível de isolamento, o PostgreSQL garante que as leituras feitas durante uma transação verão uma imagem **consistente dos dados.** Isso significa que, para cada instrução SELECT dentro da mesma transação, **os dados retornados serão consistentes com o momento em que a transação foi iniciada.** **Portanto, mesmo que outras transações tenham modificado registros na tabela T durante o minuto de espera, o segundo SELECT retornará os dados consistentes com o início da transação, ignorando as alterações feitas por outras transações**. c) Serializable: No nível de isolamento Serializable, o PostgreSQL garante que todas as transações se comportem como se estivessem sendo executadas em série, **evitando possíveis anomalias de concorrência**. Portanto, se outras transações modificarem registros na tabela T durante o minuto de espera, **o segundo SELECT aguardará até que essas transações sejam confirmadas ou revertidas antes de retornar os resultados. Isso significa que o resultado do segundo SELECT será consistente com o momento em que a transação atual começou, sem incluir as modificações feitas por outras transações que estão em andamento ou foram confirmadas**. (a - No nível de isolamento Read committed (no SGBD PostgreSQL) é possível ocorrer os seguintes problemas: leitura não repetitiva e leitura fantasma. Caso uma transação concorrente insira um registro novo na tabela T, no segundo select irá aparecer esse registro “fantasma”. Caso uma transação concorrente modifique algum registro na tabela T, no segundo select pode acontecer de os registros aparecem de maneira diferente comparado ao primeiro select. b- Como não é possível acontecer leitura suja, leitura fantasma e leitura não repetitiva no nível de isolamento Repeatable read (no SGBD PostgreSQL), então o segundo select será o mesmo que o primeiro. C- Como não é possível acontecer leitura suja, leitura fantasma e leitura não repetitiva no nível de isolamento Serializable (no SGBD PostgreSQL), então o segundo select será o mesmo que o primeiro.)

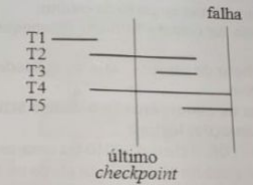
Escreva as transações acima em termos de READ e WRITE.

|  |  |
| --- | --- |
| READ(INVOICE, 10983, ‘10010’));  READ(CUSTOMER, ‘10010’);  WRITE(CUSTOMER, CUS\_BALANCE + 108.80);  READ(PRODUCT, ‘11QER/31’);  WRITE(PRODUCT, P\_QYTCH - 1); | READ(PAYMENTS, 3428, ‘10010’);  READ(CUSTOMER, ‘10010’);  WRITE(CUSTOMER, CUS\_BALANCE - 100); |

Crie um log de transações para representar as ações de duas transações anteriores.

|  |  |
| --- | --- |
| <T1 START>  <T1, Customer, CUS\_BALANCE, CUS\_BALANCE + 108.80>  <T1, Product, PY\_QYTCH, PY\_QYTCH - 1>  <T1 COMMIT> | <T2 START>  <T2, Customer, CUS\_BALANCE, CUS\_BALANCE – 100>  <T2 COMMIT> |

Considere a ilustração de um sistema com transações concorrentes antes da falha abaixo. Considerando as técnicas de recuperação baseadas em log vistas em sala de aula, o que acontecerá quando o sistema for reiniciado?



Existem duas formas de fazer a restauração baseadas em log: a modificação adiada e a imediata.

**Imediata:** T1 - Não irá acontecer nada, pois já foi commitado antes do checkpoint. T2 - Não irá acontecer nada, pois já foi commitado e o dado já está persistido no banco. T3 - Não irá acontecer nada, pois já foi commitado e o dado já está persistido no banco. T4 - Deverá ser realizada a operação UNDO, pois não terá o commit da transação (no checkpoint). Ou seja, ela não foi concluída. T5 - Deverá ser realizada a operação UNDO, pois não terá o commit da transação (no checkpoint). Ou seja, ela não foi concluída.

**Adiada:** T1 - Não irá precisar fazer nada, pois ela deu commit antes do último checkpoint. T2 - Deverá ser feita a operação REDO, já que possui o ínicio da transação e o commit antes da falha acontecer (porém é depois do checkpoint). T3 - Deverá ser feita a operação REDO, já que possui o ínicio da transação e o commit antes da falha acontecer (porém é depois do checkpoint). T4 - Será abortada, pois não há o commit antes da falha (apenas o início da transação). T5 - Será abortada, pois não há o commit antes da falha (apenas o início da transação).