O problema de valores gravados temporariamente

Este problema ocorre quando uma transação altera o valor de um item de BD, e depois a transação falha por alguma razão. O item alterado pode ser acessado por outra transação antes de retornar ao valor anterior correto.

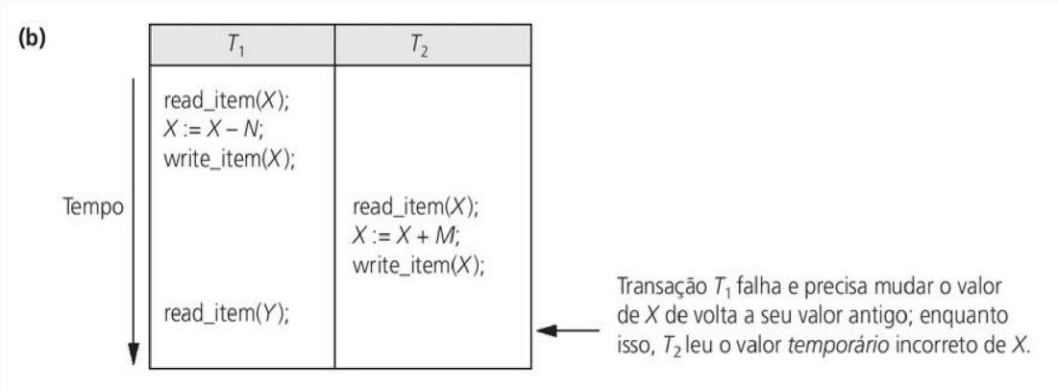

Figura (b): Problema da atualização temporária (ELMASRI e NAVATHE, 2018, p. 678).

Para solucionar esses e outros problemas, antes de qualquer ação reativa (depois que o problema ocorreu), o SGBD deve ter uma atitude precavida e gerar a forma correta de entrelaçar as transações. Para isso, existe um módulo que programa os **escalonamentos** entre as transações.

Segundo Elmasri e Navathe (2018), o escalonamento (schedule) de várias transações é a combinação de todas as operações das transações, tal que a ordem em que essas operações ocorrem no escalonamento é a mesma em que ocorrem para cada transação individualmente. Ou escalonamento é quando se pega um conjunto de transações e se define a ordem em que elas vão ser executadas, para prover concorrência no BD.

Depois de definida a forma ideal de realizar o entrelaçamento de transações concorrentes, a partir do que foi definido em seu escalonamento, as técnicas de controle de concorrência são usadas para assegurar isolamento (ou não interferência) durante a execução de transações coincidentes.

As principais técnicas que ao longo dos anos foram implementadas em diferentes SGBD são as seguintes (ELMASRI e NAVATHE, 2018):

- Bloqueio: é uma técnica considerada pessimista porque parte do pressuposto de que sempre poderá ocorrer um erro. Então, o item que será utilizado é bloqueado. É a técnica mais segura e por isso é a mais utilizada.
- Ordenação por marcadores de tempo: é uma técnica que divide a execução de transações por tempo, onde o mesmo tempo é dado para cada transação executar uma parte. Essa também é uma técnica considerada segura, porém, sua implementação é muito complexa e por isso é pouco utilizada.
- Uso de múltiplas versões: é uma técnica que cria diferentes versões dos itens de dados que serão compartilhados nas transações, altera as versões separadamente e posteriormente refaz as alterações no BD, ou faz a sincronização das diferentes versões dos itens de dados. É uma técnica que exige muito processamento e por isso torna-se lenta, se comparada com as demais. Por esse motivo, não é implementada.
- Validação: é uma técnica que parte do pressuposto de que não ocorrerão erros, por isso é considerada uma técnica otimista e não é segura. Praticamente, não há controle de concorrência na utilização dessa técnica e por isso ela não é considerada.

Como a técnica de bloqueio é a mais segura e mais utilizada vamos focar nossos estudos apenas nela. As técnicas baseadas em bloqueios podem ser implementadas com bloqueios binários ou bloqueios múltiplos.

Técnica **de bloqueio binário**

Nesta técnica de bloqueio binário, somente dois estados são permitidos para os itens do BD envolvidos nas transações: bloqueado (LOCK) ou desbloqueado (UNLOCK). Duas operações devem ser incluídas nas transações LOCK_ITEM (X) e UNLOCK_ITEM (X).

Quando o bloqueio binário é usado, toda transação deve obedecer às seguintes regras (ELMASRI e NAVATHE, 2018):

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.

Bloqueio múltiplo ou bloqueio compartilhado e exclusivo

Nesta técnica de bloqueio múltiplo dos itens do BD que estão sendo acessados nas transações, os bloqueios podem ser **compartilhados** para leitura, ou **exclusivos** para escrita. É considerado o modo múltiplo, pois diferenciam os tipos de bloqueios que serão realizados a depender da operação que estiver sendo executada. Três operações são necessárias para implementar o bloqueio múltiplo: READ_LOCK (X), WRITE_LOCK (X) e UNLOCK (X).

Um READ_LOCK é um bloqueio compartilhado, pois se uma transação está apenas lendo o valor de um item do BD, outras transações também podem compartilhar esse item para leitura; e um WRITE_LOCK é um bloqueio exclusivo, pois se um item está sendo alterado por uma transação, ele será bloqueado exclusivamente para essa transação, não podendo ser acessado pelas demais.

Quando o bloqueio de modo múltiplo é usado, o sistema deve seguir as seguintes regras (ELMASRI e NAVATHE, 2018):

1. Uma transação T deve executar uma operação de READ_LOCK (X) ou WRITE_LOCK (X) antes de qualquer operação de READ_ITEM (X) ser executada em T.
2. Uma transação T deve executar uma operação de WRITE_LOCK (X) antes de qualquer operação de WRITE_ITEM (X) ser executada em T.
3. Uma transação T deve executar uma operação de UNLOCK (X) depois de qualquer operação de READ_ITEM (X) e WRITE_ITEM (X) ter sido executada completamente em T.
4. Uma transação T não vai executar a operação de READ_LOCK (X) se o item X já estiver bloqueado compartilhado para um READ ou bloqueado exclusivo para um WRITE.

5. Uma transação T não vai executar a operação de WRITE_LOCK (X) se o item X já estiver bloqueado compartilhado para um READ ou bloqueado exclusivo para um WRITE.

6. Uma transação T não vai executar a operação de UNLOCK_ITEM (X) se o item X já estiver desbloqueado.

Algumas vezes, é possível fugir às regras 4 e 5. Por exemplo, se uma transação T estiver bloqueando um item X no modo compartilhado (READ) e for preciso fazer um bloqueio maior exclusivo (WRITE), nesse mesmo item X.

A Figura ilustra como ficariam duas transações T_1 e T_2 após aplicada a técnica de bloqueio múltiplo em ambas.

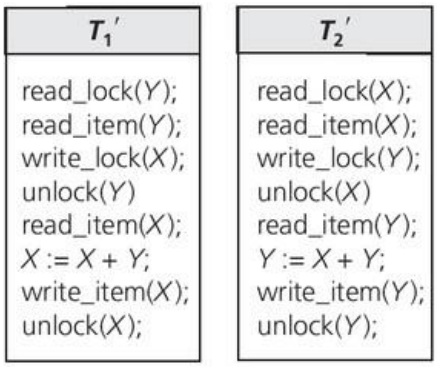

Figura: Exemplos de transações com a utilização da técnica de bloqueio múltiplo.

Fonte: Elmasri e Navathe (2018, p. 709).

A técnica de bloqueio binário é menos utilizada que a de bloqueio múltiplo, porque a primeira praticamente não permite o compartilhamento de itens do BD, ou seja, não permite a concorrência, enquanto a segunda, apesar de ser um pouco mais complexa, permite o compartilhamento para leitura e com isso aumenta a concorrência no BD. De qualquer forma, para utilização dessas técnicas alguns protocolos devem ser seguidos para garantir que a execução das transações seja realizada de forma segura. Ainda assim, alguns problemas podem ocorrer, tais como: impasse (**deadlock**) e inanição (**livelock**). O deadlock ocorre quando existem duas transações bloqueando o mesmo item do banco de dados e cada uma das duas está esperando que a outra libere esse bloqueio. A Figura ilustra como pode ocorrer um

deadlock entre duas transações T_1' e T_2'. Observe que T_1' vai utilizar o item Y do BD e o bloqueia, depois disso a transação T_2' vai utilizar o item X do BD e o bloqueia. Quando a execução volta para T_1', esta deveria utilizar o item X, mas está bloqueado para T_2'. Sendo assim, ela fica esperando a sua liberação. Depois a transação T_2' volta a executar, mas esta precisaria utilizar o item Y que está bloqueado por T_1' e também entra em estado de espera. Ou seja, nesse momento ocorre um impasse, pois T_1' espera por T_2' e vice-versa.

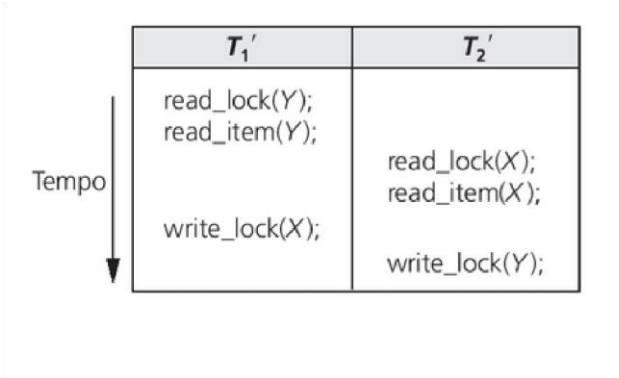

Figura: Exemplo de um deadlock ocorrendo entre duas transações (ELMASRI e NAVATHE, 2018, p. 711).

A solução mais comum para solucionar esse problema de impasse é abortar uma das transações envolvidas. Existem diversas implementações diferentes para decidir qual transação deve ser interrompida no caso de um impasse. Geralmente, é escolhida a transação mais nova. Isso pode causar o livelock (quando o sistema não consegue decidir qual das transações deve ser abortada). Então, é introduzida uma prioridade para as transações (por exemplo, maior prioridade para transações que já foram abortadas anteriormente).

Técnicas **de recuperação de falhas**

Segundo Elmasri e Navathe (2018), quando uma transação é submetida para execução num SGBD, o sistema é responsável por garantir que todas as operações na transação serão completadas com sucesso e seu resultado será gravado permanentemente no BD, ou a transação não terá efeito algum no BD ou em outra transação.

O SGBD não pode permitir que algumas operações de uma transação T sejam aplicadas ao BD, enquanto outras operações de T não são. Isso poderia acontecer no caso da ocorrência de uma falha durante a execução da transação.

Existem diversos tipos de falhas que podem ocorrer enquanto transações estão sendo executadas no SGBD, como as listadas a seguir (ELMASRI e NAVATHE, 2018):

- Falha no computador (system crash): um erro de hardware ou software, que pode causar a perda do conteúdo na memória.
- Erro de sistema ou de transação: alguma operação na transação pode causar uma falha, tal como sobrecarga de valor inteiro ou divisão por zero.
- Erros locais ou detecção de condições de exceção na transação: durante a execução, algumas condições podem ocorrer que necessitem do cancelamento da transação, por exemplo, um dado para uma operação pode não ser encontrado.
- Obrigação de controle de concorrência: o método de controle de concorrência pode decidir abortar a transação, para ser recomeçada depois, por diversos motivos, como por exemplo, deadlock.
- Falha em disco: alguns blocos do disco podem perder seus dados por causa de um defeito no cabeçote de leitura e escrita.
- Problemas físicos e sinistros: uma lista infinita de problemas que podem acontecer, tais como falta de energia, fogo, sabotagem, erros de entrada de dados do operador etc.

Se esses problemas existem e causam falhas, o SGBD deve se prevenir para evitar que os dados sejam danificados durante a execução de transações. A estratégia típica de ação dos SGBD no caso de ocorrerem falhas é a seguinte (ELMASRI e NAVATHE, 2018):

- se houver dano extenso a uma grande parte do BD devido a alguma falha catastrófica, o método de recuperação recarrega uma cópia de uma versão antiga do BD e restaura um estado mais recente refazendo transações comprometidas a partir do log. Usa um backup e geralmente interrompe o sistema para refazer;

- quando o BD apenas se tornar inconsistente devido a alguma falha não catastrófica, a estratégia é reverter às alterações que causaram inconsistência desfazendo e, algumas vezes, refazendo determinadas operações de modo a retornar o BD a um estado consistente. Um deadlock, por exemplo, pode causar uma falha desse tipo.

Para solução de problemas causados no caso dos danos extensos, existem módulos específicos que auxiliam os usuários a realizar cópia de segurança dos dados (backup) e executar posteriormente a restauração dos dados (restore). Esses módulos geralmente são disponíveis em interfaces gráficas, muito fáceis de utilizar e são da responsabilidade do Administrador do Banco de dados (DBA).

Já os problemas causados por falhas leves e danos que muitas vezes passam despercebidos pelos usuários, o módulo de recuperação após falhas desenvolve as seguintes técnicas principais: técnica de atualização imediata (Algoritmo UNDO / REDO) ou técnica de atualização adiada (Algoritmo NO UNDO / REDO).

Técnica de atualização imediata (Algoritmo UNDO / REDO)

Quando o SGBD utiliza a técnica da atualização imediata, significa que as alterações nos dados ocorrem no BD no momento exato em que a transação foi executada. Nesse caso, então, se ocorre uma falha enquanto a transação está sendo executada, todas as alterações feitas terão que ser desfeitas e posteriormente tudo terá que ser refeito. Ou seja, como a atualização é imediata no BD, se ocorre à falha, tem que desfazer (UNDO) tudo e refazer (REDO).

Na prática, a técnica de atualização imediata é muito pouco utilizada. Sendo assim, estudaremos somente a técnica de atualização adiada.

Técnica de atualização adiada (Algoritmo NO UNDO / REDO)

Quando o SGBD utiliza a técnica da atualização adiada, significa que a alteração não foi feita “fisicamente” no BD, porém já foi feita numa determinada área de trabalho e as operações foram gravadas no log de operações. Se ocorrer uma falha durante a execução de uma transação com essa técnica, os dados ainda não

foram gravados de fato. Então, não precisa desfazer nada (NO UNDO), só refazer (REDO) o que já tinha sido comprometido antes.

A ideia básica dessa técnica é adiar a execução de qualquer atualização sobre o BD até que a transação seja concluída com sucesso e atinja o seu ponto de comprometimento. Durante a execução de uma transação, as operações são apenas registradas no log e na área de trabalho da transação. Depois que a transação atinge seu ponto de comprometimento e o log é escrito em disco, as atualizações são registradas no BD.

Para garantir a integridade dos dados usando essa técnica, é aplicado um protocolo:

- Uma transação não pode alterar o BD até que atinja o seu ponto de comprometimento.
- Uma transação não atinge o seu ponto de comprometimento até que suas operações de atualização estejam gravadas no log e o log esteja escrito em disco.

No caso de ocorrer uma falha durante a execução de transações, o procedimento básico de recuperação deverá executar a operação REDO para todas as operações WRITE das transações já comprometidas na ordem em que elas aparecem no log e reiniciar as transações que estavam ativas no momento da falha. A operação de REDO (refaz) irá reexecutar a operação de WRITE, basicamente irá buscar no log de operações, para cada entrada [W IT _IT M, T, X, valor_antigo, valor_novo] no log, substitui o valor de “X” no BD pelo “valor_novo”.

Transações ativas não têm efeito sobre o BD, uma vez que a alteração de fato ainda não ocorreu, mas elas não são totalmente ignoradas durante o processo de recuperação, pois elas precisam ser reiniciadas.

Geralmente, é enviada uma mensagem ao usuário que está executando essas transações informando que ocorreu uma falha e que sua transação foi reiniciada. É o chamado ROLLBACK implícito.

Os Principais SGBD do Mercado

Atualmente, existem diversos SGBD distintos no mercado. Na verdade, são algumas centenas de produtos diferentes, mas apenas algumas dezenas são os mais utilizados pela maioria das empresas. Esses SGBD se diferenciam pelo modelo que utilizam como base para sua implementação (relacional, orientado a objeto, objeto relacional etc.), por sua popularidade (quantidade de pessoas que utilizam o produto), pela forma como distribuem suas licenças (proprietário ou software livre), pela quantidade de usuários que irá acessar o BD (multiusuário ou monousuário), dentre outros critérios.

Cada SGBD possui suas particularidades e é mais adequado à realidade de cada empresa. Geralmente, cabe ao DBA a decisão pela escolha por um ou outro produto, mas de fato o que é mais importante nessa área é compreender pelo menos um pouco da teoria de bancos de dados para que, posteriormente, seja possível tomar as melhores decisões com segurança.

Para podermos destacar alguns dos principais SGBD do mercado, utilizamos como fonte de dados um conhecido ranking de destaque internacional, o DB-Engines Ranking, que mensalmente cria uma lista dos principais e mais populares SGBD utilizados no mundo. Ao todo, a lista apresenta 345 diferentes sistemas.

Dados de abril de 2019 apresentam os seguintes SGBD como os 10 mais populares:

Oracle
MySQL
Microsoft SQL Server
PostreSQL
MongoDB
IBM DB2
Redis
Elasticsearch
Microsoft Access
SQLite

SISTEMAS AVANÇADOS EM BANCOS DE DADOS

Bancos de Dados Distribuídos

Segundo Elmasri e Navathe (2018), um Banco de Dados Distribuído (BDD) é um conjunto de múltiplos bancos de dados logicamente inter-relacionados, distribuídos por uma rede de computadores. Ou seja, num BDD as partes do banco de dados espalhadas por uma rede de computadores possuem uma unidade lógica.

Os BDD geralmente acompanham a estrutura descentralizada da organização para a qual o sistema foi desenvolvido. Então, se a organização se encontra distribuída em diferentes cidades ou estados de um país, o banco de dados estará da mesma forma distribuído.

O BDD permite que os dados sejam acessados localmente e gerenciados globalmente, podendo fornecer informações confiáveis em qualquer lugar da rede de computadores onde está implementado. A **transparência** é a característica que torna possível sua utilização. Nem os usuários, nem os programadores precisam saber onde ou como os dados estão armazenados. Para ambos, as operações parecem ser efetuadas sobre um único BD contíguo. O Sistema Gerenciador de Banco de Dados Distribuído (SGBDD) é que administra a distribuição dos dados.

Um SGBDD é usado para gerenciar e facilitar a distribuição dos dados, tornando-a transparente aos usuários. Em outras palavras, quem utiliza os BDD não percebe que ele está distribuído numa rede de computadores. Isso é transparente, porque o gerenciamento da distribuição fica por conta do SGBDD. A Figura ilustra uma arquitetura de um banco de dados distribuído em 5 diferentes locais de uma rede de computadores.

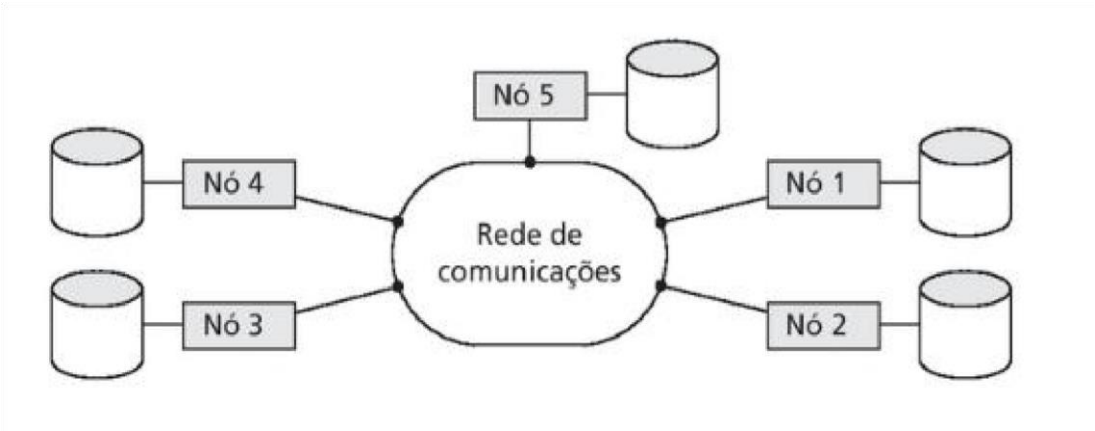

Figura: Arquitetura de um banco de dados verdadeiramente distribuído.
Fonte: Adaptada de Elmasri e Navathe (2018, p. 783).

Características de BD Distribuídos vs. Centralizados

Duas características importantes a serem consideradas em um BDD são: a distribuição (o fato de que os dados não estão no mesmo local ou processador, podendo assim ser feita a distinção entre um BDD de um BD centralizado) e a correlação lógica (o fato de que os dados possuem algumas propriedades que os relacionam, tornando possível ser feita a distinção entre um BDD de um conjunto de diferentes bancos de dados locais ou arquivos que são residentes em diferentes locais de uma rede de computador).

A seguir, serão considerados alguns aspectos associados a Bancos de Dados Centralizados (BDC) e como eles são enfocados sobre o ponto de vista dos BDD:

Controle e Administração do Banco de Dados: um dos motivos para se introduzir o conceito banco de dados foi à possibilidade de se ter controle centralizado sobre as informações de uma organização. A principal função do Administrador de Banco de Dados (DBA) era garantir a segurança e integridade dos dados; os dados eram reconhecidos como um importante investimento de uma organização e se fazia necessária uma responsabilidade centralizada. Nos BDD, a ideia de controle centralizado é muito menos enfatizada. Em BDD, é possível identificar uma estrutura

de controle hierárquica baseada num DBA global, que tem controle do BD como um todo, e dos DBAs locais, que têm responsabilidade sobre os BD locais. Contudo, devemos enfatizar que os DBAs locais devem ter um alto grau de autonomia. Essa característica é usualmente chamada de autonomia local. Esse grau pode variar em BDD: desde uma completa autonomia local, sem nenhum DBA global, até um controle centralizado completo.

- **Independência de Dados**: uma das características da independência de dados é garantir imunidade das aplicações à estrutura de armazenamento e a estratégia de acesso aos dados propriamente ditos. Programas são escritos tendo uma visão “conceitual” dos dados, o chamado esquema conceitual. A maior vantagem da independência de dados é que os programas não são afetados por mudanças na organização física dos dados. Nos BDD, a independência de dados tem a mesma importância que em BD centralizados; contudo, um novo aspecto é adicionado à noção usual de independência de dados – a transparência de distribuição (programas podem ser escritos como se o BD não fosse distribuído). Dessa forma, apenas a velocidade de execução de um programa é afetada ao movimentar dados de um local para o outro.

- **Redução de Redundância:** nos BD centralizados, a redundância é reduzida ao máximo, porque inconsistência entre muitas cópias do mesmo dado lógico são automaticamente evitadas tendo somente uma cópia e o espaço de armazenamento é economizado. Redução de redundância é obtida por compartilhamento de dados, isto é, permitindo que várias aplicações acessem as mesmas tabelas ao mesmo tempo. Nos BDD, existem muitas razões para se considerar a redundância de dados como uma característica desejável. Primeiro, a localidade das aplicações pode ser aumentada se o dado for replicado em todos os locais onde as aplicações precisam dele e, segundo, a disponibilidade do sistema pode ser aumentada, porque a queda em um local não para a execução das aplicações em outros locais onde os dados estejam replicados.

Componentes de uma Arquitetura de Bancos de Dados Distribuídos

São dois os principais componentes da arquitetura de um BDD: um esquema global e um esquema de fragmentação. O esquema global define todos os dados que estão contidos no BDD como se o BD não fosse totalmente distribuído. Por essa razão, o esquema global pode ser definido exatamente do mesmo modo que é definido num BD centralizado. Usando o modelo relacional, o esquema global consiste de um conjunto de relações globais. É a esse esquema global que os usuários do sistema terão acesso.

Cada relação global pode ser dividida em porções chamadas de fragmentos ou segmentos, localizados em um ou muitos locais (sites) de uma rede. Existem muitas maneiras diferentes de realizarmos uma operação de segmentação, como veremos a seguir. O mapeamento entre as relações globais e os segmentos é definido como esquema de fragmentação ou segmentação.

Enquanto o esquema de segmentação é o mapeamento entre as relações globais e os segmentos, o esquema de alocação define em que local o segmento está fisicamente localizado. Note que o tipo de mapeamento definido no esquema de alocação determina o quanto o BDD é redundante ou não. Do esquema de fragmentação surgem os esquemas internos locais de cada site da rede, diretamente ligados aos bancos de dados propriamente armazenados em cada localidade. A Figura representa uma arquitetura de esquemas de um BDD.

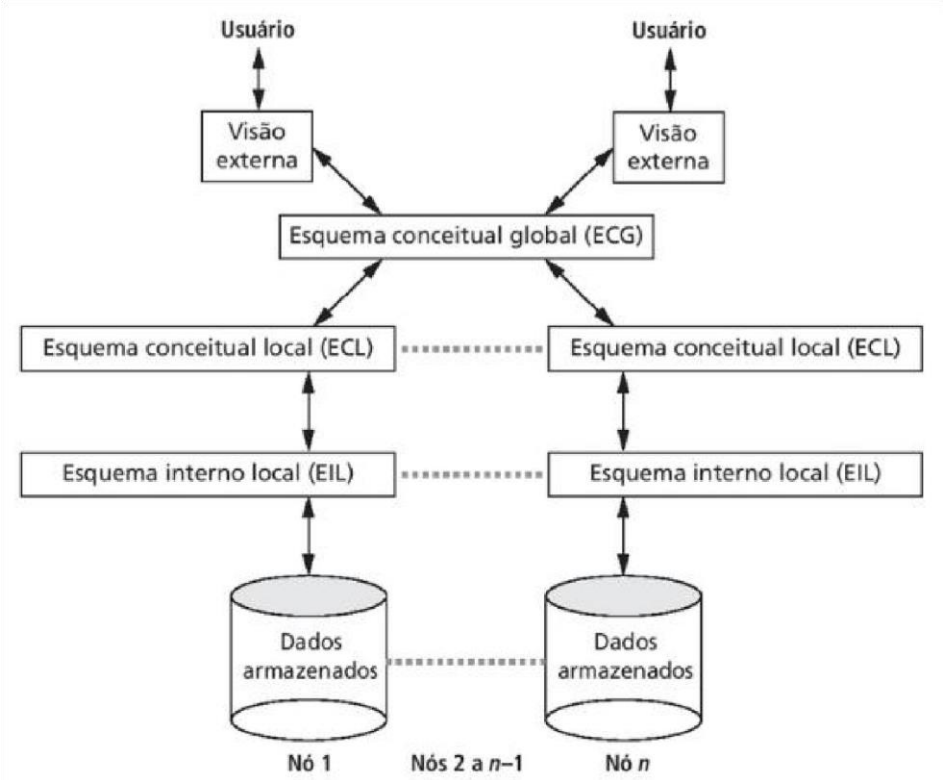

Figura: Arquitetura do esquema para bancos de dados distribuídos.

Fonte: Adaptada de Elmasri e Navathe (2018, p. 784).

Essa arquitetura proporciona uma organização conceitual bem generalizada para o entendimento dos BDD.

Os três objetivos mais importantes que motivam as características dessa arquitetura são:

- Separação do Conceito de Segmentação de Dados do Conceito de Alocação de Dados: essa separação nos leva a distinguir dois diferentes níveis de transparência de distribuição: transparência de segmentação e transparência de alocação. A separação entre os conceitos de segmentação e alocação é muito conveniente no projeto de BDD.
- Controle de Redundância: a referência de arquitetura proporciona controle de redundância no nível de segmento.
- Independência do SGBD local: essa característica, chamada transparência de mapeamento local, leva ao estudo de muitos problemas do SGBDD sem ter que se preocupar com modelos específicos de dados de um SGBD local. Certamente, num sistema homogêneo, é possível que a independência do local seja

definida usando o mesmo modelo de dados do SGBD, reduzindo assim a complexidade do mapeamento.

Outro tipo de transparência, que é estritamente relacionado à transparência de localização, é a transparência de replicação. Transparência de replicação significa que o usuário não tem conhecimento da replicação dos segmentos.

A decomposição das relações globais em segmentos pode ser realizada aplicando dois tipos diferentes de segmentação: segmentação horizontal e/ou segmentação vertical.

Em todos os tipos de segmentação, um segmento pode ser definido por uma expressão numa linguagem relacional, que toma relações globais como operandos e produz segmentos como resultados. Existem, contudo, algumas regras que devem ser seguidas quando definimos segmentos:

- Condição de Completeza: todos os dados de uma relação global devem ser mapeados em segmentos; isto é, não pode existir um item de dado que pertença a uma relação global que não pertença a algum segmento;
- Condição de Reconstrução: deve ser sempre possível reconstruir cada relação global a partir dos seus segmentos. A necessidade desta condição é óbvia: de fato, somente segmentos são armazenados em BDD, e uma relação global tem que ser construída através de uma operação de reconstrução, se necessário.