SISTEMA GERENCIADOR DE BANCO DE DADOS

AULA 3

Profª Vívian Ariane Barausse de Moura

CONVERSA INICIAL

O objetivo da aula é prosseguir com os assuntos relacionados à segurança do banco de dados (BD), introduzindo os principais conceitos sobre a recuperação de um BD, bem como os principais conceitos sobre controle de concorrência e *performance*. Para isso, é preciso conhecer os aspectos gerais que estão relacionados à recuperação de um banco de dados e os conceitos sobre os processos de recuperação. Em controle de concorrência serão apresentados alguns protocolos utilizados para essa prática e, em *performance*, veremos algumas técnicas para otimizar as consultas.

TEMA 1 – RECUPERAÇÃO DE BANCO DE DADOS E SEGURANÇA

1.1 Aspectos gerais da recuperação de um banco de dados

Em relação às questões relacionadas à segurança, abordaremos os aspectos gerais relativos à recuperação de um banco de dados. Conforme a definição apresentada por Alves (2014, p. 148), a recuperação de um BD está relacionada à "forma de preservação e de recuperação de informações ou do banco de dados inteiro. Isso significa que o administrador deve fazer uso de recursos oferecidos pelo próprio servidor de banco de dados ou utilizar outro meio que possibilite ter cópias do banco de dados".

De acordo com Ramakrishnan e Gehrke (2008, p. 484), "o gerenciador de recuperação de um SGBD é responsável por garantir duas propriedades importantes das transações: atomicidade e durabilidade". Com tais garantias, desfazem-se as ações das transações que não foram efetivadas e se garante que todas as ações das transações efetivadas sobrevivam às falhas de sistema e de mídia. O autor complementa que o gerenciador de recuperação é um dos componentes do SGBD mais difíceis de projetar e implementar, pois ele precisa tratar uma variedade de estados do banco de dados (Ramakrishnan; Gehrke, 2008).

Alves (2014) destaca que a maneira mais óbvia é fazer *backups* (cópias de segurança) periódicos. Alguns sistemas operacionais e servidores oferecem ferramentas para essa tarefa, mas no caso de não estarem disponíveis, pode ser utilizado um dos vários aplicativos utilitários disponíveis no mercado, alguns até gratuitos. A maioria dos SGBDs relacionais padrão SQL oferece utilitários para

cópia e restauração (*backup/restore*) das bases de dados. A semântica da utilização de comandos vai depender de qual *software* está sendo utilizado.

Ainda em relação às cópias de segurança, é aconselhável que as operações de *backup* e *restore* sejam executadas quando não houver usuários acessando o banco de dados. Alves (2014) defende que não é possível deixar a critério de uma única pessoa a execução da cópia de segurança, mas utilizar o método de cópia automática quando for oferecido, e que nunca é demais ter duas ou três cópias. Por exemplo: uma que é a imagem atual do banco de dados e duas que são anteriores. Isso facilita no caso de haver necessidade de recuperação de dados.

Elsmari e Navathe (2011) defendem que em geral a recuperação de falhas significa que um banco de dados é restaurado ao estado consistente mais recente antes do momento da falha. Para que isso ocorra, é necessário que o sistema mantenha as informações sobre as mudanças que foram aplicadas aos itens de dados pelas diversas transações. Os autores propõem estratégias de recuperação conforme apresentada na Tabela 1.

Tabela 1 – Estratégias de recuperação

1	Se houver dano extensivo a uma grande parte do banco de dados devido a uma falha catastrófica, como uma falha de disco, o método de recuperação restaura uma cópia antiga do banco de dados que teve *backup* para o arquivamento (normalmente, fita ou outro meio de armazenamento *off-line* de grande capacidade) e reconstrói um estado mais recente, reaplicando ou refazendo as operações das transações confirmadas do *log* em *backup*, até o momento da falha.
2	Quando o banco de dados no disco não está danificado fisicamente, por conta de uma falha não catastrófica, a estratégia de recuperação é identificar quaisquer mudanças que possam causar uma inconsistência no banco de dados. Por exemplo, uma transação que atualizou alguns itens do banco de dados no disco, mas não confirmou as necessidades de ter suas mudanças revertidas ao desfazer suas operações de gravação. Pode também ser preciso refazer algumas operações a fim de restaurar um estado consistente do banco de dados; por exemplo, se uma transação tiver sido confirmada, mas algumas de suas operações de gravação ainda não tiverem sido gravadas em disco. Para a falha não catastrófica, o protocolo de recuperação não precisa de uma cópia de arquivamento completa do banco de dados. Em vez disso, as entradas mantidas no *log* do sistema *on-line* no disco são analisadas para determinar as ações apropriadas para recuperação.

Fonte: Elsmari; Navathe, 2011, p. 543.

Utilizando as estratégias de recuperação para distinguir as técnicas de recuperação de falhas não catastróficas, Elsmari e Navathe (2011) destacam duas principais: atualização adiada e atualização imediata, apresentadas no Quadro 1.

Quadro 1 – Técnicas de atualização

Atualização adiada	Não atualizam fisicamente o banco de dados no disco até que uma transação atinja seu ponto de confirmação; então as atualizações são registradas no banco de dados. Antes de atingir a confirmação, todas as atualizações de transação são registradas no espaço de trabalho de transação local ou nos *buffers* da memória principal que o SGBD mantém (o cachê da memória principal do SGBD). Antes da confirmação, as atualizações são gravadas persistentemente no *log* e, após a confirmação, elas são gravadas no banco de dados no disco. Se uma transação falhar antes de atingir seu ponto de confirmação, ela não terá alterado o banco de dados de forma alguma, de modo que o UNDO não é necessário. Pode ser preciso um REDO para desfazer o efeito das operações de uma transação confirmada com base no *log*, pois seu efeito pode ainda não ter sido registrado no banco de dados em disco. Assim, a atualização adiada também é conhecida como algoritmo NO-UNDO/REDO.
Atualização imediata	O banco de dados pode ser atualizado por algumas operações de uma transação antes que a transação alcance seu ponto de confirmação. Porém essas operações também precisam ser registradas no *log* no disco ao forçar a gravação antes que elas sejam aplicadas ao banco de dados no disco, tornando a recuperação ainda possível. Se uma transação falhar depois de gravadas algumas mudanças no disco, mas antes de atingir seu ponto de confirmação, o efeito de suas operações no banco de dados precisa ser desfeito; ou seja, a transação deve ser revertida. No caso geral da atualização imediata, tanto UNDO quanto REDO podem ser exigidos durante a recuperação. Essa técnica, conhecida como algoritmo UNDO/REDO, requer as duas operações durante a recuperação, e é usada na prática. Uma variação do algoritmo, em que todas as atualizações precisam ser registradas no banco de dados em disco antes que a transação se confirme, requer apenas UNDO, de modo que é conhecido como algoritmo UNDO/NO-REDO.

Fonte: Elsmari; Navathe, 2011, p. 544.

Elsmari e Navathe (2011) destacam que as operações UNDO e REDO precisam ser idempotentes. Isso quer dizer que a execução de uma operação várias vezes é equivalente a executá-la apenas uma vez. Nesse sentido, o processo de recuperação inteiro deve ser idempotente, pois se o sistema falhasse durante o processo de recuperação, a próxima tentativa de recuperação poderia realizar um UNDO e um REDO de certas operações *write_item* que já tinham sido executadas durante o primeiro processo de recuperação. Elsmari e Navathe (2011, p. 544) afirmam que “O resultado da recuperação de uma falha do sistema durante a recuperação deve ser igual ao resultado da recuperação quando não há falha durante esse processo”.

2.1 Algoritmo de recuperação ARIES

Uma das propostas apresentadas por Ramakrishnan e Gehrke (2008, p. 485) é o algoritmo de recuperação ARIES, que, segundo o autor, está sendo usado em um número cada vez maior de sistemas de banco de dados, pois funciona bem com uma ampla variedade de mecanismos de controle de concorrência.

Ramakrishnan e Gehrke (2008, p. 485) definem que "o ARIES é um algoritmo de recuperação projetado para trabalhar com uma estratégia de roubo, sem imposição. Quando o gerenciador de recuperação é ativado após uma falha, o reinício ocorre em três fases", as quais estão representadas no Quadro 2:

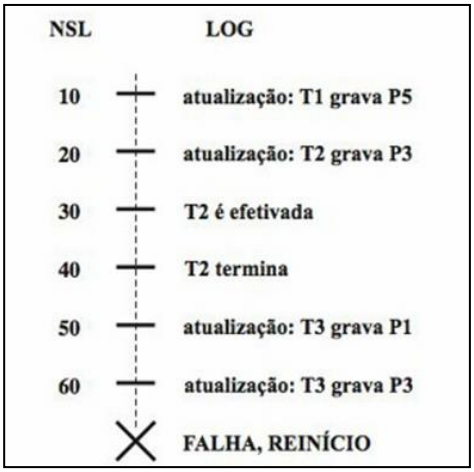
Quadro 2 – Fases de reinício do ARIES

1. Análise	Identifica as páginas sujas no *pool* de *buffers* (isto é, alterações que não foram gravadas no disco) e as transações ativas no momento da falha.
2. Refazer	Repete todas as ações, partindo de um ponto apropriado no *log*, e restaura o banco de dados para o estado em que ele estava no momento da falha.
3. Desfazer	Desfaz as ações das transações que não foram efetivadas, para que o banco de dados reflita apenas as ações das transações efetivadas.

Fonte: Ramakrishnan; Gehrke, 2008, p. 485.

Essas fases estão representadas no histórico de execução simples ilustrado na Figura 1 e na sequência explicativa apresentada no Quadro 3.

Figura 1 – Histórico de execução com uma falha



Fonte: Ramakrishnan; Gehrke, 2008, p. 486.

Quadro 3 – Processos da execução com uma falha

1. Análise	Quando o sistema é reiniciado, a fase Análise identifica **T1** e **T3** como transações ativas no momento da falha e, portanto, a serem desfeitas; identifica **T2** como uma transação efetivada e, portanto, todas as suas ações a serem gravadas no disco; e **P1, P3** e **P5** como páginas possivelmente sujas.
2. Refazer	Todas as atualizações (incluindo as de **T1** e **T3**) são reaplicadas na ordem mostrada durante a fase Refazer
3. Desfazer	As ações de **T1** e **T3** são desfeitas na ordem inversa, durante a fase Desfazer; ou seja, a gravação de **P3** realizada por **T3** é desfeita, a gravação de **P1** realizada por **T3** é desfeita e, então, a gravação de **P5** realizada por **T1** é desfeita.

Fonte: Ramakrishnan; Gehrke, 2008, p. 485.

De acordo com Ramakrishnan e Gehrke (2008), existem três princípios fundamentais no algoritmo de recuperação ARIES, representados na Tabela 2.

Tabela 2 – Princípios do algoritmo de recuperação ARIES

1. Gravação antecipada do *Log (Write-Ahead Logging, WAL)*	Qualquer alteração em um objeto de banco de dados é primeiramente gravada no *log*; o registro que está no *log* deve ser gravado em um meio de armazenamento estável, antes que a alteração no objeto de banco de dados seja gravada no disco.
2. Repetição do Histórico durante a fase Refazer	Na inicialização após uma falha, o ARIES refaz todas as ações do SGBD antes da falha e coloca o sistema de volta no estado exato em que se encontrava no momento da falha. Ele então desfaz as ações das transações que ainda estavam ativas no momento da falha (cancelando-as efetivamente).
3. Registro das alterações durante a fase Desfazer:	As alterações feitas no banco de dados, enquanto uma transação é desfeita, são registradas para garantir que essa ação não seja repetida no caso de reinícios (causando falhas) repetidos.

Fonte: Ramakrishnan; Gehrke, 2008, p. 486.

O segundo ponto distingue o ARIES dos outros algoritmos de recuperação é a base de grande parte de sua simplicidade e flexibilidade. Em particular, o ARIES pode aceitar protocolos de controle de concorrência que envolvem bloqueios de granularidade mais precisa do que uma página (por exemplo, bloqueios em nível de registro). O segundo e o terceiro pontos também são importantes ao se tratar com operações em que refazer e desfazer a operação não são as inversas exatas uma da outra.

Para exemplificar as fases de reinício do ARIES foi utilizada a Figura1. Nela estão contidas as representações de *log* e NSL. De acordo com Ramakrishnan e Gehrke (2008, p. 486), “o *log*, às vezes chamado de trilha ou periódico, é um histórico das ações executadas pelo SGBD”. Fisicamente, o *log* é um arquivo de registros armazenado em um meio de armazenamento estável

que supostamente deve sobreviver a falhas; essa durabilidade pode ser obtida por meio da manutenção de duas ou mais cópias do *log* em discos diferentes (talvez em locais diferentes), para que a chance de que todas as cópias do *log* sejam perdidas ao mesmo tempo torne-se insignificantemente pequena. (Ramakrishnan; Gehrke, 2008)

A parte mais recente do *log*, chamada de *cauda do log*, é mantida na memória principal, e é periodicamente gravada no armazenamento estável. Desse modo, os registros do *log* e os registros dos dados são gravados no disco com a mesma granularidade (páginas ou conjuntos de páginas). Todo registro de *log* recebe um *id* exclusivo, chamado de *número de sequência de log (NSL).* (Ramakrishnan; Gehrke, 2008)

Assim como acontece com qualquer *id* de registro, dado o NSL, podemos buscar um *log* gravado com um único acesso a disco. Além disso, os NSLs devem ser atribuídos em ordem monotonicamente crescente; essa propriedade é obrigatória para o algoritmo de recuperação ARIES. Se o *log* é um arquivo sequencial, em princípio crescendo indefinidamente, o NSL pode simplesmente ser o endereço do primeiro *byte* do registro de *log*.1 Para propósitos de recuperação, cada página no banco de dados contém o NSL do registro de *log* mais recente que descreve uma alteração nessa página. Esse NSL é chamado de *NSLpágina* (NSL de página) (Ramakrishnan; Gehrke, 2008).

Além do *log*, Ramakrishnan e Gehrke (2008, p. 489) defendem outras estruturas que contêm importantes informações relacionadas à recuperação, representadas no Quadro 4.

Quadro 4 – Tabelas relacionadas à recuperação

Tabela de transações	**Tabela de páginas sujas**
Essa tabela contém uma entrada para cada transação ativa. A entrada contém (dentre outras coisas) o *id* da transação, o *status* e um campo chamado último NSL, que é o NSL do registro de *log* mais recente dessa transação. O *status* de uma transação pode indicar que ela está em andamento, foi efetivada ou foi cancelada. (Nos dois últimos casos, a transação será removida da tabela quando certas etapas de 'limpeza' forem completadas.)	Essa tabela contém uma entrada para cada página suja no *pool* de *buffers*; isto é, cada página com alterações ainda não refletidas no disco. A entrada contém um campo NSLreg, que é o NSL do primeiro registro de *log* que fez a página tornar-se suja. Note que esse NSL identifica o registro de *log* mais recente que talvez tenha que ser refeito para essa página durante o reinício a partir de uma falha.

Fonte: Ramakrishnan; Gehrke, 2008, p. 489.

Durante a operação normal, essas tabelas são mantidas, respectivamente, pelo gerenciador de transação e pelo gerenciador de *buffer*; durante o reinício, após uma falha, elas são reconstruídas na fase Análise.

Ramakrishnan e Gehrke (2008, p. 490) salientam que, antes de gravar uma página no disco, todo registro de *log* de atualização que descreve uma alteração nessa página precisa ser gravado em armazenamento estável. Isso é feito gravando-se em armazenamento estável todos os registros de *log* até (e incluindo) aquele com o NSL igual a NSLpágina, antes de gravar a página no disco.

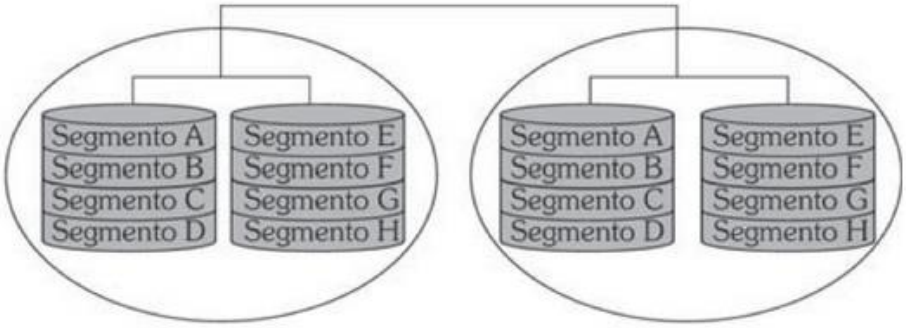
Neste sentido, os autores destacam a importância do protocolo de gravação antecipada do *log*, *write-ahead log – WAL*. O WAL é a regra fundamental que garante que um registro de cada alteração no banco de dados esteja disponível enquanto se tenta recuperar de uma falha. Se uma transação fez alterações e foi efetivada, a estratégia sem imposição significa que algumas dessas alterações podem não ter sido gravadas no disco no momento de uma falha subsequente. Sem um registro delas, não haveria como garantir que as alterações de uma transação efetivada sobrevivessem às falhas.

2.2 Discos rígidos espelhados – arquitetura RAID

Outra forma, segundo Alves (2014), um pouco mais dispendiosa, porém mais segura e que não depende de terceiros, é a utilização de discos rígidos espelhados, como na arquitetura RAID (*Redundant Array Inexpensive/Independent Disk*, ou “Conjunto Redundante de Discos Econômicos/Independentes”). De acordo com Alves (2014, p. 149), em ambientes cliente/servidor que rodam aplicações de missão crítica, é possível encontrar máquinas inteiras espelhadas, cuja arquitetura (*hardware* e *software*) permite que, se uma delas falhar, a outra entre em substituição automaticamente, sem que os usuários percebam qualquer anomalia.

A arquitetura RAID compreende um agrupamento de discos rígidos que funcionam de forma concomitante ou paralela, com o objetivo de reduzir os riscos de danos causados a arquivos e aumentar o desempenho no acesso aos dados. Os discos podem trabalhar independentemente ou de maneira sincronizada, com os dados espalhados entre eles. O RAID pode ser implementado via *software* ou por *hardware*. Existem vários tipos de configurações disponíveis, denominados de *níveis*.

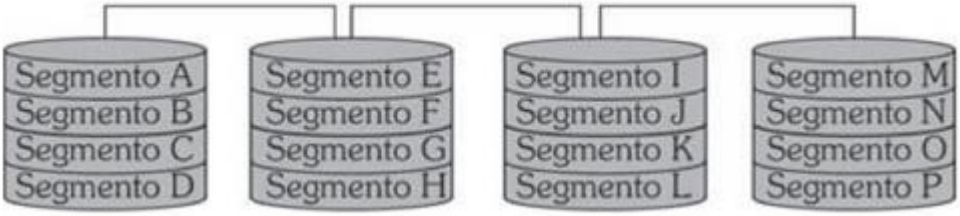
Figura 2 – Arquitetura RAID 0+1



Fonte: Alves, 2014, p. 150.

De acordo com Alves (2014), a Figura 2 demonstra que o funcionamento exige pelo menos quatro discos na implementação, com espelhamento de cada par de discos e os pares representando o RAID nível 0, que pode ser visto na Figura 3.

Figura 3 – Arquitetura RAID 0



Fonte: Alves, 2014, p. 150.

Alves (2014) indica que existem níveis da aplicação do RAID. O autor cita do nível 1 ao nível 4. O que é alterado nos níveis são as questões estruturais e de implementação.

2.3 Recuperação de mídia

De acordo com Ramakrishnan e Gehrke (2008, p. 497), a recuperação de mídia é baseada em fazer periodicamente uma cópia do banco de dados. “Como copiar um objeto de banco de dados grande, por exemplo, um arquivo, pode demorar bastante e o SGBD precisa continuar com suas operações nesse meio tempo, a criação de uma cópia é tratada de maneira semelhante a usar um ponto de verificação *fuzzy*”.

O autor salienta que, quando um objeto de banco de dados, como um arquivo ou uma página é corrompido, a cópia desse objeto é atualizada usando-se o *log* para identificar e reaplicar as alterações das transações efetivadas e

desfazer as alterações das transações não efetivadas (no momento da operação de recuperação da mídia) (Ramakrishnan; Gehrke, 2008).

Neste caso, o NSL *início_ponto-de-verificação* do ponto de verificação mais recente concluído é registrado junto com a cópia do objeto de banco de dados para minimizar o trabalho na reaplicação das alterações das transações efetivadas. Vamos comparar o menor NSLreg de uma página suja no registro *final_ponto-de-verificação* com o NSL do registro *início_ponto-de-verificação* e chamar de I o menor desses dois NSLs. Observamos que as ações registradas em todos os registros de *log* com NSLs menores do que I devem ser refletidas na cópia. Assim, apenas os registros de *log* com NSLs maiores do que I precisam ser reaplicados na cópia.

Segundo Ramakrishnan e Gehrke (2008), ao realizar esse processo, as atualizações das transações que estão incompletas no momento da recuperação da mídia ou que foram canceladas depois que a cópia *fuzzy* foi concluída precisam ser desfeitas para garantir que a página reflita apenas as ações das transações efetivadas.

TEMA 3 – CONTROLE DE CONCORRÊNCIA

O controle de concorrência é abordado por Elsmari e Navathe (2011, p. 523) como a “aplicação de diversas técnicas que são usadas para garantir a propriedade de não interferência ou isolamento das transações executadas simultaneamente”. Para isso, são utilizadas o que os autores definem como *protocolos de controle de concorrência*, que são um conjunto de regras que garantem a serialização.

De acordo com Ramakrishnan e Gehrke (2008, p. 444) “um SGBD deve ser capaz de garantir que apenas planos de execução serializáveis e recuperáveis sejam permitidos e que nenhuma ação de transações efetivadas seja perdida ao desfazer transações canceladas”. Para conseguir isso, o SGBD utiliza um protocolo de bloqueio.

Um conjunto de protocolos importante é o protocolo de bloqueio em duas fases, que, segundo Elsmari e Navathe (2011, p. 523), “emprega a técnica de bloqueio de itens de dados para impedir que múltiplas transações acessem os itens ao mesmo tempo”. De acordo com esses mesmos autores, os protocolos de bloqueio são utilizados na maioria dos SGBDs comerciais.

Ramakrishnan e Gehrke (2008, p. 444) defendem que “bloqueio é um pequeno objeto de controle associado a um objeto do banco de dados” e o protocolo de bloqueio “é um conjunto de regras a serem seguidas por transação (e impostas pelo SGBD) para garantir que, mesmo intercalando as ações de várias transações, o resultado seja idêntico à execução de todas as transações em alguma ordem serial”. Esses autores especificam que diferentes protocolos de bloqueio usam diferentes tipos de bloqueios, como bloqueios compartilhados ou exclusivos.

O protocolo de bloqueio mais amplamente usado, de acordo com Ramakrishnan e Gehrke (2008), é o bloqueio de duas fases restrito, ou Strict 2PL (do inglês, *Strict Two-Phase Locking*), que tem duas regras, dispostas na Tabela 3.

Tabela 3 – Regras do bloqueio de duas fases restrito

1. Se uma transação T quer ler (respectivamente, modificar) um objeto, ela primeiro solicita um bloqueio compartilhado (respectivamente, exclusivo) sobre o objeto.	1. Uma transação que tenha um bloqueio exclusivo também pode ler o objeto; não é exigido um bloqueio compartilhado adicional. Uma transação que solicita um bloqueio é suspensa até que o SGBD seja capaz de garantir-lhe o bloqueio solicitado. O SGBD monitora os bloqueios que concedeu e garante que, se uma transação mantiver um bloqueio exclusivo sobre um objeto, nenhuma outra transação manterá um bloqueio compartilhado ou exclusivo sobre o mesmo objeto.
2. Todos os bloqueios mantidos por uma transação são liberados quando a transação termina.	2. Os pedidos para adquirir e liberar bloqueios podem ser inseridos automaticamente nas transações pelo SGBD; os usuários não precisam se preocupar com esses detalhes. Na verdade, o protocolo de bloqueio só permite intercalações “seguras” de transações. Se duas transações acessam partes completamente independentes do banco de dados, elas obtêm concorrentemente os bloqueios que precisam e seguem seu caminho normalmente. Por outro lado, se duas transações acessam o mesmo objeto e uma quer modificá-lo, suas ações são efetivamente ordenadas em série – todas as ações de uma dessas transações (aquela que recebe primeiro o bloqueio sobre o objeto comum) são concluídas antes (que esse bloqueio seja liberado) e que a outra transação possa prosseguir.

Fonte: Ramakrishnan; Gehrke, 2008, p. 444.

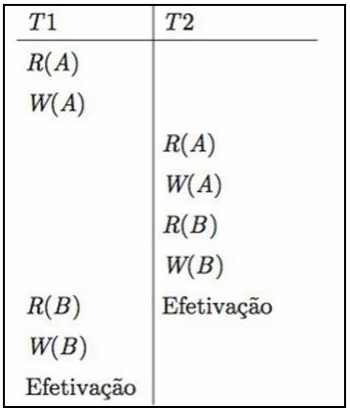
Ramakrishnan e Gehrke (2008, p. 440) ilustram as três maneiras principais pelas quais um plano de execução envolvendo duas transações efetivadas e que preservam a consistência poderia ser executado em um banco de dados consistente, e deixá-lo em um estado inconsistente. Considere o exemplo representado no Quadro 5.

Quadro 5 – Exemplo para explicação

Situação	Duas ações sobre o mesmo objeto de dados entram em conflito se pelo menos uma delas é uma gravação. As três situações anômalas podem ser descritas em termos de quando as ações de duas transações (**T1** e **T2)** entram em conflito entre si: em um conflito de gravação-leitura (**WR**), **T2** lê um objeto de dados gravado anteriormente por **T1**; definimos os conflitos de leitura-gravação (RW) e gravação-gravação (WW) analogamente.
Lendo dados não efetivados (conflitos de WR)	A primeira fonte de anomalias é que uma transação **T2** poderia ler um objeto de banco de dados **A** que foi modificado por outra transação **T1**, que ainda não foi efetivada. Essa leitura é chamada de *leitura suja*.
	Um exemplo simples ilustra como um plano de execução poderia levar um banco de dados a um estado inconsistente: Considere duas transações **T1** e **T2**, sendo que cada uma destas, quando executada sozinha, preserva a consistência do banco de dados: **T1** transfere US$ 100 de **A** para **B** e **T2** acresce **A** e **B** em 6% (por exemplo, os juros anuais são depositados nessas duas contas). Suponha que as ações sejam intercaladas de modo que (1) o programa de transferência entre contas **T1** subtraia US$ 100 da conta A e, então, (2) o programa de depósito de juros **T2** leia os valores correntes das contas **A** e **B**, adicione juros de 6% em cada uma e, em seguida, (3) o programa de transferência entre contas credite US$ 100 na conta **B**. O plano de execução correspondente, que é a visão que o SGBD tem dessa série de eventos, está ilustrado na Figura 4. O resultado desse plano é diferente de qualquer resultado que obteríamos executando primeiro uma das duas transações e depois a outra. O problema pode ser descoberto pelo fato de que o valor de **A** gravado por **T1** é lido por **T2** antes que **T1** tenha concluído todas as suas alterações.
Problema	O problema geral ilustrado aqui é que **T1** pode gravar algum valor em **A** que torne o banco de dados inconsistente. Contanto que **T1** sobrescreva seu valor com um valor “correto” de A, antes de efetivar, nenhum dano é causado se **T1** e **T2** forem executadas em alguma ordem serial, pois **T2** não veria a inconsistência (temporária). Por outro lado, a execução intercalada pode expor essa inconsistência e levar o banco de dados a um estado final inconsistente. Note que, embora uma transação deva deixar um banco de dados em um estado consistente depois de ser concluída, ela não é obrigada a mantê-lo consistente enquanto ainda está em andamento. Tal requisito seria restritivo demais: para transferir dinheiro de uma conta para outra, uma transação deveria debitar uma conta, deixando o banco de dados temporariamente inconsistente e, depois, creditar na segunda conta, restaurando a consistência.

Fonte: Ramakrishnan; Gehrke, 2008, p. 441.

Figura 4 – Lendo dados não efetivados



Fonte: Ramakrishnan; Gehrke, 2008, p. 441.

Considerando o plano de execução apresentado na Figura 4, denotamos a ação de uma transação **T** solicitando um bloqueio compartilhado (respectivamente, exclusivo) sobre o objeto O, como $S_T(O)$ [respectivamente, $X_T(O)$], e omitimos o subscrito que denota a transação quando isso está claro segundo o contexto, apresentado no Quadro 6.

Quadro 6 – Exemplo para representação protocolo Script 2PL

Intercalação	Essa intercalação poderia resultar em um estado que não acontece por meio de nenhuma execução serial das três transações: Por exemplo, **T1** poderia mudar **A** de 10 para 20, assim **T2** (que lê o valor 20 para **A**) poderia mudar **B** de 100 para 200, e **T1** poderia ler o valor 200 para **B**. Se fosse executada em série, **T1** ou **T2** executaria primeiro e leria os valores 10 para **A** e 100 para **B**. Claramente, a execução intercalada não é equivalente a nenhuma execução serial.
Se o protocolo Strict 2PL for usado, tal intercalação é proibida	Supondo que as transações ocorram na mesma velocidade relativa como antes, **T1** obteria um bloqueio exclusivo sobre **A** primeiro e, então, leria e gravaria **A** (IMAGEM 4); depois **T2** solicitaria um bloqueio sobre **A**. Entretanto, esse pedido não poderia ser garantido até que **T2** liberasse seu bloqueio exclusivo sobre **A** e, portanto, o SGBD suspenderia **T2**. Agora, **T1** obtém um bloqueio exclusivo sobre **B**, lê e grava **B** e finalmente é efetivada, momento esse em que seus bloqueios são liberados. Agora, o pedido de bloqueio de **T2** é garantido e ela continua.

Fonte: Ramakrishnan; Gehrke, 2008, p. 443.

Figura 5 – Plano de execução ilustrando o protocolo Script 2PL

$T1$	$T2$
$X(A)$	
$R(A)$	
$W(A)$	

Fonte: Ramakrishnan; Gehrke 2008, p. 445.

No exemplo ilustrado, o protocolo de bloqueio resulta em uma execução serial das duas transações, mostradas na Figura 6.

Figura 6 – Plano de execução ilustrando o protocolo Strict 2PL com execução serial

$T1$	$T2$
$X(A)$	
$R(A)$	
$W(A)$	
$X(B)$	
$R(B)$	
$W(B)$	
Efetivação	
	$X(A)$
	$R(A)$
	$W(A)$
	$X(B)$
	$R(B)$
	$W(B)$
	Efetivação

Fonte: Ramakrishnan; Gehrke, 2008, p. 445.

Em geral, entretanto, Ramakrishnan e Gehrke (2008, p. 446) afirmam que “as ações de diferentes transações poderiam ser intercaladas”. O autor exemplifica a intercalação de duas transações, mostrada na Figura 7, que é permitida pelo protocolo Strict 2PL.

Figura 7 – Plano de execução seguindo o protocolo Strict 2PL com ações intercaladas

$T1$	$T2$
$S(A)$	
$R(A)$	
	$S(A)$
	$R(A)$
	$X(B)$
	$R(B)$
	$W(B)$
	Efetivação
$X(C)$	
$R(C)$	
$W(C)$	
Efetivação	

Fonte: Ramakrishnan; Gehrke, 2008, p. 446.

Analisando o exposto, o algoritmo Strict 2PL só permite planos de execução serializáveis, e as anomalias já identificadas dentro das pesquisas não podem surgir se o SGBD implementar o protocolo Strict 2PL (Ramakrishnan; Gehrke, 2008).

Outro conjunto de protocolos de controle de concorrência, de acordo com Elsmari e Navathe (2011), utiliza rótulos de tempo (*timestamp*). Um rótulo de tempo é um identificador exclusivo para cada transação gerado pelo sistema. Os valores de rótulo de tempo são gerados na mesma ordem que os tempos de início de transação.

Ramakrishnan e Gehrke (2008) mostram que o controle de concorrência *timestamp* é traduzido como “marca de tempo” e é feito um comparativo com o controle de concorrência baseado em bloqueio, pois, segundo os autores, nestes casos, as ações conflitantes de diferentes transações são ordenadas pela disposição na qual os bloqueios são obtidos, e o protocolo de bloqueio estende essa ordem para as ações das transações, garantindo com isso a serialidade. No controle de concorrência otimista, uma ordem de marca de tempo (*timestamp*) é imposta às transações e a validação verifica se todas as ações conflitantes ocorreram na mesma ordem.

Elsmari e Navathe (2011) chamam a atenção para outro fator que afeta o controle de concorrência: a granularidade dos itens de dados, ou seja, que parte do banco de dados de um item de dados representa. Todas as técnicas de controle de concorrência consideram que o banco de dados é formado por uma série de itens de dados nomeados. Um item de banco de dados é variável, pois pode ser pequeno como um único valor de atributo (campo) ou tão grande quanto um bloco de disco e pode ser escolhido como sendo um dos seguintes itens (Elsmari; Navathe, 2011, p. 536):

- Um registro de banco de dados;
- Um valor de campo de um registro de banco de dados;
- Um bloco de disco;
- Um arquivo inteiro;
- Um banco de dados inteiro.

Considerando o exposto, a granularidade pode afetar o desempenho do controle de concorrência e recuperação, o que ocasiona alguns dilemas com

relação à escolha do nível de granularidade usando bloqueios. Existem esquemas de bloqueio com granularidade múltipla, em que o nível de granularidade (tamanho do item de dados) pode ser alterado dinamicamente (Elsmari; Navathe, 2011).

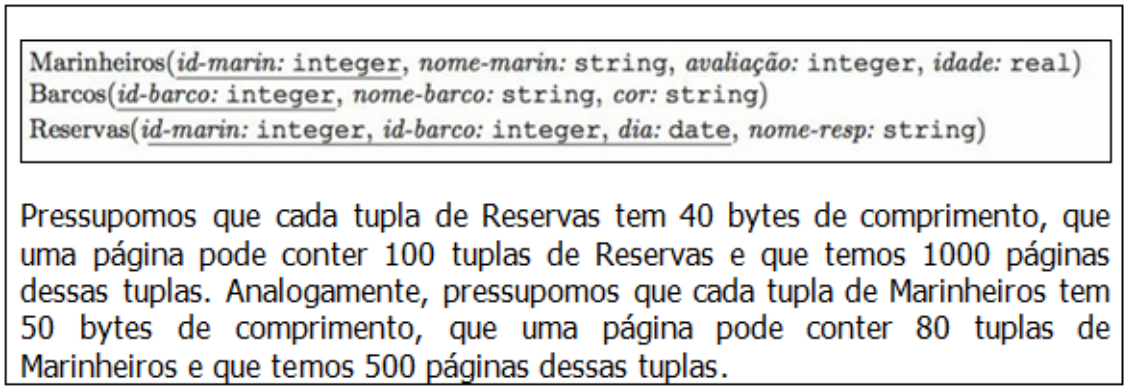
TEMA 4 – *PERFORMANCE* – CONHECER TÉCNICAS DE OTIMIZAÇÃO DE CONSULTAS (CONSULTAS, ÍNDICES)

De acordo com Ramakrishnan e Gehrke (2008, p. 390), “a principal tarefa de um otimizador é encontrar um bom plano para avaliar essas expressões”. O autor defende que a otimização de uma expressão da álgebra relacional envolve duas etapas:

- Enumerar planos alternativos para avaliar a expressão. Normalmente, o otimizador considera um subconjunto de todos os planos possíveis, pois o número deles é muito grande.
- Estimar o custo de cada plano enumerado e escolher o plano com o custo estimado mais baixo.

Para explicitar os exemplos de consulta, Ramakrishnan e Gehrke (2008) utilizam o esquema representado na Figura 8:

Figura 8 – Esquema de consulta para os exemplos



Fonte: Ramakrishnan; Gehrke, 2008, p. 399.

4.1 Consultas

Ramakrishnan e Gehrke (2008, p. 391) indicam que, para serem realizadas, as consultas SQL são otimizadas pela sua decomposição em um conjunto de unidades menores chamadas *blocos*. Um otimizador de consultas

relacional típico se concentra na otimização de um único bloco por vez. Para isso, uma consulta é decomposta em blocos, e a otimização de um único bloco pode ser compreendida em termos de planos constituídos de operadores da álgebra relacional.

A decomposição de uma consulta em blocos ocorre, conforme Ramakrishnan e Gehrke (2008), da seguinte forma: quando um usuário envia uma consulta SQL, ela é decomposta em um conjunto de blocos e depois enviada para o otimizador de consultas. Um bloco de consulta (ou simplesmente bloco) é uma consulta SQL sem nenhum aninhamento e com exatamente uma cláusula SELECT, uma cláusula FROM e, no máximo, uma cláusula WHERE, uma cláusula GROUP BY e uma cláusula HAVING. Supõe-se que a cláusula WHERE esteja na forma normal conjuntiva. Ramakrishnan e Gehrke (2008) usam a consulta da Figura 9 como exemplo de funcionamento:

Figura 9 – Marinheiros reservando barcos vermelhos

```
SELECT      M .id-marin, MIN (R.dia)
FROM        Marinheiros M, Reservas R, Barcos B
WHERE       S.id-marin = R.id-marin AND R.id-barco = B.id-barco AND
            B.cor = 'vermelho' AND
            S.avaliação = ( SELECT MAX (M2.avaliação)
                            FROM Marinheiros M2)
GROUP BY    S.id-marin
HAVING      COUNT (*) > 1
```

Para cada marinheiro com a classificação mais alta (em relação a todos os marinheiros) e com pelo menos duas reservas para barcos vermelhos, encontre o id do marinheiro e a data mais próxima na qual ele tem uma reserva para um barco vermelho.

Fonte: Ramakrishnan; Gehrke, 2008, p. 400.

A versão SQL dessa consulta aparece na Figura 9. A consulta tem dois blocos, sendo que o bloco aninhado está representado na Figura 10:

Figura 10 – Representação bloco aninhado

```
SELECT  MAX (M2.avaliação)
FROM    Marinheiros M2
```

Fonte: Ramakrishnan; Gehrke, 2008, p. 401.

Segundo Ramakrishnan e Gehrke (2008), o bloco aninhado calcula a classificação de marinheiro mais alta. O bloco externo aparece na Figura 11, em

que toda consulta SQL pode ser decomposta em um conjunto de blocos sem aninhamento.

Figura 11 – Bloco externo da consulta de barcos vermelhos

```
SELECT    S.id-marin, MIN (R.dia)
FROM      Marinheiros S, Reservas R, Barcos B
WHERE     S.id-marin = R.id-marin AND R.id-barco = B.id-barco AND
          B.cor = 'vermelho' AND
          S.avaliação = Referência ao bloco aninhado
GROUP BY  S.id-marin
HAVING    COUNT (*) > 1
```

Fonte: Ramakrishnan; Gehrke, 2008, p. 401.

Dessa forma, o otimizador examina os catálogos do sistema para recuperar informações sobre os tipos e comprimentos dos campos, estatísticas sobre as relações referenciadas e os caminhos de acesso (índices) disponíveis para elas. Depois, o otimizador considera cada bloco de consulta e escolhe um plano de avaliação de consulta para esse bloco (Ramakrishnan; Gehrke, 2008).

Para cada plano enumerado, precisamos estimar seu custo. Ramakrishnan e Gehrke (2008) defendem que existem duas partes para estimar o custo de um plano de avaliação de um bloco de consulta, conforme representado na Tabela 4:

Tabela 4 – Partes para estimar o custo de um plano de avaliação de um bloco de consulta

Para cada nó na árvore, devemos estimar o custo da execução da operação correspondente. Os custos são afetados significativamente pelo fato de se usar *pipelining* ou se relações temporárias são criadas para passar a saída de um operador para seu ascendente.	Para cada nó na árvore, devemos estimar o tamanho do resultado e verificar se ele está ordenado. Esse resultado é a entrada para a operação que corresponde ao ascendente do nó corrente e, por sua vez, o tamanho e a ordem afetam a estimativa de tamanho, o custo e a ordem do ascendente.

Fonte: Ramakrishnan; Gehrke, 2008, p. 403.

Ramakrishnan e Gehrke (2008) destacam que estimar custos exige o conhecimento de vários parâmetros das relações de entrada, como o número de páginas e os índices disponíveis. Tais estatísticas são mantidas nos catálogos de sistema do SGBD. As estimativas usadas por um SGBD para tamanhos e custos do resultado são, na melhor das hipóteses, aproximações dos tamanhos

e custos reais. Não é realista esperar que um otimizador encontre o melhor plano possível; é mais importante evitar os piores planos e encontrar um bom plano.

TEMA 5 – *PERFORMANCE* – CONHECER TÉCNICAS DE OTIMIZAÇÃO DE CONSULTAS (CONSULTAS, ÍNDICES)

5.1 Índices

De acordo com Ramakrishnan e Gehrke (2008), todos os principais SGBDRs reconhecem a importância dos planos somente de índice e, quando possível, procuram esses planos. A utilização ocorre de acordo com o autor, conforme apresentado na Tabela 5.

Tabela 5 – Utilização de índices nos SGBDRs

IBM DB2	Ao criar um índice, o usuário pode especificar um conjunto de colunas 'de inclusão' que devem ser mantidas no índice, mas que não fazem parte da chave do índice. Isso possibilita que um conjunto mais rico de consultas somente de índice seja manipulado, pois as colunas acessadas frequentemente são incluídas no índice, mesmo que não façam parte da chave.
Microsoft SQL Server	Considerada uma classe interessante de planos somente de índice, suponha uma consulta que seleciona atributos salário e idade a partir de uma tabela, dado um índice em salário e outro em idade.
SQL Server	Associando as entradas por junção no *rid* dos registros de dados para identificar os pares (salário, idade) que aparecem na tabela.

Fonte: Ramakrishnan; Gehrke, 2008, p. 414.

Ramakrishnan e Gehrke (2008) defendem que os índices podem ser utilizados de várias maneiras e podem levar a planos significativamente mais rápidos do que qualquer outro que não os utilize, conforme representado no Quadro 7.

Quadro 7 – Representação de índice

1. Caminho de acesso com um único índice	Se vários índices correspondem às condições de seleção na cláusula WHERE, cada índice correspondente oferece um caminho de acesso alternativo. Um otimizador pode escolher o caminho de acesso que, segundo sua estimativa, vai resultar na recuperação do menor número de páginas, e pode aplicar projeções e termos de seleção não primários (isto é, partes da condição de seleção que não correspondem ao índice), e pode passar a calcular as operações de agrupamento e agregação (ordenando nos atributos de GROUP BY).

(continua)

(continuação do Quadro 7)

2. Caminho de acesso com vários índices	Se vários índices usando as alternativas (2) ou (3) para entradas de dados corresponderem à condição de seleção, cada índice poderá ser usado para recuperar um conjunto de *rids*. Podemos fazer a intersecção desses conjuntos de *rids* e, depois, ordenar o resultado pelo *id* da página (supondo que a representação de *rid* inclua o *id* da página) e recuperar as tuplas que satisfazem os termos da seleção primária de todos os índices correspondentes. Todas as projeções e termos de seleção não primários podem, então, ser aplicados, seguidos das operações de agrupamento e agregação.
3. Caminho de acesso com índice ordenado	Se a lista de atributos de agrupamento for um prefixo de um índice de árvore, o índice pode ser usado para recuperar as tuplas na ordem exigida pela cláusula GROUP BY. Todas as condições de seleção podem ser aplicadas em cada tupla recuperada, os campos indesejados podem ser removidos e as operações agregadas podem ser calculadas para cada grupo. Essa estratégia funciona bem para índices agrupados.
4. Caminho de acesso somente de índice	Se todos os atributos mencionados na consulta (nas cláusulas SELECT, WHERE, GROUP BY ou HAVING) estiverem incluídos na chave de pesquisa para algum índice denso na relação da cláusula FROM, uma varredura somente de índice poderá ser usada para calcular as respostas. Como as entradas de dados no índice contêm todos os atributos de uma tupla necessários para essa consulta e há apenas uma entrada de índice por tupla, nunca precisamos recuperar tuplas reais da relação. Usando apenas as entradas de dados do índice, podemos executar os passos a seguir, conforme for necessário, em determinada consulta: aplicar as condições de seleção, remover os atributos indesejados, ordenar o resultado para obter agrupamento e calcular as funções agregadas dentro de cada grupo. Essa estratégia somente de índice funciona mesmo que o índice não corresponda às seleções da cláusula WHERE. Se o índice corresponder à seleção, precisamos examinar apenas um subconjunto das entradas de índice; caso contrário, devemos percorrer todas as entradas de índice. Em qualquer caso, podemos evitar a recuperação de registros de dados reais; portanto, o custo desta estratégia não depende de o índice ser agrupado. Além disso, se é um índice de árvore e a lista de atributos na cláusula GROUP BY forma um prefixo da chave do índice, podemos recuperar as entradas de dados na ordem necessária para a cláusula GROUP BY e, com isso, evitar a ordenação.

Fonte: Ramakrishnan; Gehrke, 2008, p. 413.

Ramakrishnan (2008) ilustra esses casos usando a consulta representada na Figura 12 como exemplo de funcionamento, supondo que estão disponíveis os índices a seguir, todos usando a Alternativa (2) para entradas de dados: um índice de árvore B+ em *avaliação*, um índice de *hashing* em *idade* e um índice de árvore B+ em *{avaliação, nome-marin, idade*}. As etapas desses planos são varreduras (uma varredura de arquivo, uma varredura recuperando tuplas usando um índice ou uma varredura apenas de entradas de índice), ordenação e gravação de relações temporárias.

Figura 12 – Uma consulta de relação única

```
SELECT      S.avaliação, COUNT (*)
FROM        Marinheiros M
WHERE       S.avaliação > 5 AND S.idade = 20
GROUP BY    S.avaliação
HAVING      COUNT DISTINCT (S.nome-marin) > 2
```

Fonte: Ramakrishnan; Gehrke, 2008, p. 411.

Quadro 8 – Aplicação utilizando índice

1. Caminho de acesso com um único índice	Poderá optar por recuperar tuplas de marinheiros tais que *M.idade=20*, usando o índice de *hashing* em idade. O custo dessa etapa é o da recuperação das entradas de índice mais o custo da recuperação das tuplas de marinheiros correspondentes, que depende de o índice ser agrupado ou não. Pode então aplicar a condição *S.avaliação > 5* em cada tupla recuperada, remover os campos não mencionados nas cláusulas SELECT, GROUP BY e HAVING, e gravar o resultado em uma relação temporária. No exemplo, apenas os campos *avaliaçã*o e *nome-marin* precisam ser mantidos. Assim, a relação temporária é ordenada no campo avaliação para identificar os grupos, e alguns grupos são eliminados pela aplicação da condição de HAVING.
2. Caminho de acesso com vários índices	Poderá recuperar *rids* de tuplas que satisfazem *avaliação>5* usando o índice em *avaliação*, recuperar *rids* de tuplas que satisfazem *idade=20* usando o índice em *idade*, ordenar os *rids* recuperados pelo número de página e recuperar as tuplas de marinheiros correspondentes. Pode manter apenas os campos *avaliação* e *nome* e gravar o resultado em uma relação temporária, a qual ordenaremos em avaliação para implementar a cláusula GROUP BY. (Um bom otimizador poderia usar *pipeline* nas tuplas projetadas para o operador de ordenação, sem criar uma relação temporária.) A cláusula HAVING é manipulada como antes.
3. Caminho de acesso com índice ordenado	Poderá recuperar tuplas de marinheiros nas quais *S.avaliação > 5*, ordenadas por *avaliação*, usando o índice de árvore B+ em *avaliação*. Pode calcular dinamicamente as funções agregadas nas cláusulas HAVING e SE-LECT, pois as tuplas são recuperadas na ordem de avaliação.
4. Caminho de acesso somente de índice	Pode recuperar *entradas de dados* do índice de *{avaliação, nome-marin, idade}* nas quais *avaliação > 5*. Essas entradas são ordenadas por *avaliação* (e, depois, por *nome-marin* e *idade*, embora essa ordenação adicional não seja relevante para essa consulta). Pode escolher entradas com *idade=20* e calcular dinamicamente as funções agregadas nas cláusulas HAVING e SELECT, pois as entradas de dados são recuperadas na ordem de *avaliação*. Nesse caso, em contraste com o caso anterior, não recupera quaisquer tuplas de marinheiros. Essa propriedade de não recuperar registros de dados torna a estratégia de usar somente índice particularmente útil com índices não agrupados.

Fonte: Ramakrishnan; Gehrke, 2008, p. 414.

FINALIZANDO

Nesta aula apresentamos os conceitos sobre recuperação de um banco de dados, assim como discutimos algumas técnicas que podem ser usadas para a recuperação do banco de dados contra eventuais falhas. Foram apresentados conceitos utilizados pelos processos de recuperação, como *log* do sistema e pontos de confirmação, além de conceitos sobre os protocolos de recuperação e uma visão geral sobre alguns algoritmos de recuperação de banco de dados. Aprendemos algumas técnicas que são usadas para garantir o controle de concorrência e, em *performance*, algumas técnicas de otimização de consultas e índices.

REFERÊNCIAS

ALVES, W. P. **Banco de dados**. São Paulo: Érica, 2014.

ELSMARI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson Education do Brasil, 2011.

RAMAKRISHNAN, R.; GEHRKE, J. **Sistemas de gerenciamento de bancos de dados**. 3. ed. Porto Alegre: McGraw Hill, 2008.