CEFET/RJ

CONTROLE DE CONCORRÊNCIA

Eduardo Ogasawara
eogasawara@ieee.org
https://eic.cefet-rj.br/~eogasawara

Protocolos baseados em bloqueio

- Um bloqueio é um mecanismo para controlar o acesso simultâneo a um item de dados
- Os itens de dados podem ser bloqueados em dois modos:
  - Modo exclusivo (X). O item de dados pode ser lido e também escrito. O bloqueio X é solicitado pela instrução lock-X
  - Modo compartilhado (S). O item de dados só pode ser lido. O bloqueio S é solicitado pela instrução lock-S
- As solicitações de bloqueio são feitas ao gerenciador de controle de concorrência
- A transação só pode prosseguir após a concessão da solicitação

Protocolos baseados em bloqueio (cont.)

- Matriz de compatibilidade de bloqueio
- Uma transação pode receber um bloqueio sobre um item se o bloqueio solicitado for compatível com os bloqueios já mantidos sobre o item por outras transações
- Qualquer quantidade de transações pode manter bloqueios compartilhados sobre um item, mas se qualquer transação mantiver um bloqueio exclusivo sobre um item, nenhuma outra pode manter qualquer bloqueio sobre o item
- Se um bloqueio não puder ser concedido, a transação solicitante deve esperar até que todos os bloqueios incompatíveis mantidos por outras transações tenham sido liberados. O bloqueio é então concedido

	S	X
S	true	false
X	false	false

Protocolos baseados em bloqueio (cont.)

- Exemplo de uma transação realizando bloqueio:

T_2
lock-S(A)
read (A)
unlock(A)
lock-S(B)
read (B)
unlock(B)
display(A+B)

- O bloqueio ao lado não é suficiente para garantir a seriação - se A e B fossem atualizados entre a leitura de A e B, a soma exibida estaria errada
- Um protocolo de bloqueio precisa de uma política para solicitar e liberar bloqueios
- Os protocolos de bloqueio restringem o conjunto de schedules seriais possíveis

Armadilhas dos protocolos baseados em bloqueio (impasse)

- Considere o schedule parcial
- Nem T_3 nem T_4 podem ter progresso - a execução de lock-S(B) faz com que T_4 espere que T_3 libere seu bloqueio sobre B, enquanto a execução de lock-X(A) faz com que T_3 espere que T_4 libere seu bloqueio sobre A
- Essa situação é chamada de impasse
  - Para lidar com um impasse, um dentre T_3 ou T_4 precisa ser revertido e seus bloqueios liberados.

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

Armadilhas dos protocolos baseados em bloqueio (inanição)

- O potencial para impasse existe na maioria dos protocolos de bloqueio. Os impasses são um mal necessário
- Inanição também é possível se o gerenciador de controle de concorrência for mal projetado. Por exemplo:
  - Uma transação pode estar esperando por um bloqueio X sobre um item, enquanto uma sequência de outras transações solicita e recebe um bloqueio S sobre o mesmo item
  - A mesma transação é repetidamente revertida, devido aos impasses
- O gerenciador de controle de concorrência pode ser designado para impedir a inanição

O protocolo de bloqueio em duas fases

- Esse é um protocolo que garante schedules seriáveis por conflito
- Fase 1: Fase de crescimento
  - transação pode obter bloqueios
  - transação não pode liberar bloqueios
- Fase 2: Fase de encurtamento
  - transação pode liberar bloqueios
  - transação não pode obter bloqueios
- O protocolo garante a seriação
  - Pode ser provado que as transações podem ser seriadas na ordem de seus pontos de bloqueio (ou seja, o ponto onde uma transação adquiriu seu bloqueio final)

Conversões de bloqueio

- Bloqueio em duas fases com conversões de bloqueio:
- Primeira fase:
  - pode adquirir um bloqueio-S sobre o item
  - pode adquirir um bloqueio-X sobre o item
  - pode converter um bloqueio-S para um bloqueio-X (upgrade)
- Segunda fase:
  - pode converter um bloqueio-X para um bloqueio-S (downgrade)
  - pode liberar um bloqueio-X
  - pode liberar um bloqueio-S
- Esse protocolo garante a seriação
  - Do jeito que foi apresentado, demanda do programador para inserir as diversas instruções de bloqueio

Aquisição automática de bloqueios

- A operação read(D) é processada como:

```
if Ti tem um bloqueio sobre D then
     read(D)
else
begin
     se necessário, espera até que nenhuma outra transação
        tenha um bloqueio-X sobre D
          concede a Ti um bloqueio-S sobre D;
          read(D)
end
```

Uma transação Ti emite a instrução de leitura/escrita padrão, sem chamadas de bloqueio explícitas.

Aquisição automática de bloqueios (cont.)

- write(D) é processado como:

```
if Ti tem um bloqueio-X sobre D
       then
         write(D)
      else
        begin
           se for preciso, espera até que nenhuma outra
              transação tenha um bloqueio sobre D,
           if Ti tem um bloqueio-S sobre D
                then
                   upgrade do bloqueio sobre D para bloqueio-X
               else
                   concede a Ti um bloqueio-X sobre D
               write(D)
        end;
```

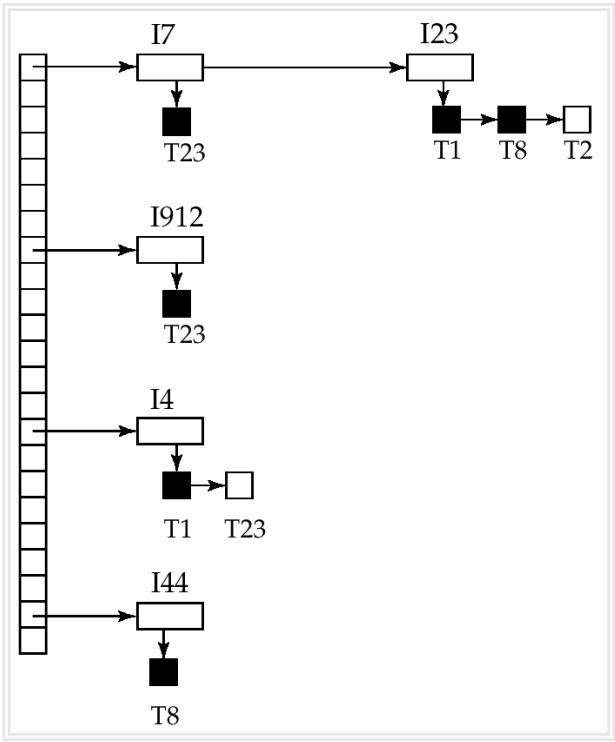
Todos os bloqueios são liberados após o commit ou abort

Implementação do bloqueio

- Um gerenciador de bloqueio pode ser implementado como um processo separado para o qual as transações enviam solicitações de bloqueio e desbloqueio
- O gerenciador de bloqueio responde a uma solicitação de bloqueio enviando uma mensagem de concessão de bloqueio (ou uma mensagem pedindo à transação para reverter, no caso de um impasse)
- A transação solicitante espera até que sua solicitação seja respondida
- O gerenciador de bloqueio mantém uma estrutura de dados chamada tabela de bloqueio para registrar bloqueios concedidos e solicitações pendentes
- A tabela de bloqueio normalmente é implementada como uma tabela de hash na memória indexada sobre o nome do item de dados sendo bloqueado

Tabela de bloqueio

- Retângulos pretos indicam bloqueios concedidos, brancos indicam solicitações aguardando
- A tabela de bloqueio também registra o tipo de bloqueio concedido ou solicitado
- A nova solicitação é acrescentada ao final da fila de solicitações para o item de dados, e concedida se for compatível com todos os bloqueios anteriores
- As solicitações de desbloqueio resultam na solicitação sendo excluída e solicitações posteriores são verificadas para saber se agora podem ser concedidas
- Se a transação abortar, todas as solicitações aguardando ou concedidas da transação são excluídas
  - o gerenciador de bloqueio pode manter uma lista de bloqueios mantidos por cada transação, para implementar isso de forma eficiente



Protocolos baseados em estampa de tempo

- Cada transação tem uma estampa de tempo emitida quando entra no sistema. Se uma transação antiga $\boldsymbol{T_i}$ tem a estampa de tempo TS($\boldsymbol{T_i}$), uma nova transação $\boldsymbol{T_j}$ recebe a estampa de tempo TS(Tj) de modo que TS($\boldsymbol{T_i}$) < TS($\boldsymbol{T_j}$).
- O protocolo gerencia a execução concorrente tal que as estampas de tempo determinam a ordem de seriação
- Para garantir esse comportamento, o protocolo mantém para cada dado Q dois valores de estampa de tempo:
  - W-timestamp(Q) é a maior estampa de tempo de qualquer transação que executou write(Q) com sucesso
  - R-timestamp(Q) é a maior estampa de tempo de qualquer transação que executou read(Q) com sucesso

Protocolos baseados em estampa de tempo (cont.)

- O protocolo de ordenação de estampa de tempo garante que quaisquer operações read e write em conflito sejam executadas na ordem de estampa de tempo
- Suponha que uma transação $\boldsymbol{T_i}$ emita um read(Q)
  - Se TS($\boldsymbol{T_i}$) ≤ W-timestamp(Q), então $\boldsymbol{T_i}$ precisa ler um valor de Q que já foi modificado. Logo, a operação read é rejeitada, e $\boldsymbol{T_i}$ é revertida
  - Se TS($\boldsymbol{T_i}$) ≥ W-timestamp(Q), então a operação read é executada, e R-timestamp(Q) é definido como o máximo de R-timestamp(Q) e TS($\boldsymbol{T_i}$)

Protocolos baseados em estampa de tempo (cont.)

- Suponha que a transação T_i emita write(Q)
- Se TS(T_i) < R-timestamp(Q), então o valor de Q que T_i está produzindo foi necessário anteriormente, e o sistema considerou que esse valor nunca seria produzido.
  - Logo, a operação write é rejeitada, e T_i é revertido
- Se TS(T_i) < W-timestamp(Q), então T_i está tentando escrever um valor obsoleto de Q.
  - Logo, essa operação write é rejeitada, e T_i é revertida
- Caso contrário, a operação write é executada, e W-timestamp(Q) é definida como TS(T_i)

Exemplo de uso do protocolo

- Um schedule parcial para transações com estampas de tempo 1, 2, 3, 4

T_1	T_2	T_3	T_4
			read(x)
	read(y)		
read(y)			
		write(y)	
		write(z)	
			read(z)
	write(x)		
	abort		
read(x)			
		write(z)	
		abort	
			write(y)
			write(z)

Exatidão do protocolo de ordenação de estampa de tempo

- O protocolo de ordenação de estampa de tempo garante a seriação, pois todos os arcos no grafo de precedência são da forma:
- Assim, não haverá ciclos no grafo de precedência
- O protocolo de estampa de tempo garante liberdade de impasse, pois nenhuma transação precisa esperar

Regra do write de Thomas

- Versão modificada do protocolo de ordenação de estampa de tempo em que as operações write obsoletas podem ser ignoradas sob certas circunstâncias
- Quando $\boldsymbol{T_i}$ tenta escrever o item de dados Q, se TS($\boldsymbol{T_i}$) < W-timestamp(Q), então $\boldsymbol{T_i}$ está tentando escrever um valor obsoleto de {Q}. Logo, em vez de reverter $\boldsymbol{T_i}$ como o protocolo de ordenação de estampa de tempo teria feito, essa operação {write} pode ser ignorada
- Caso contrário, esse protocolo é igual ao protocolo de ordenação de estampa de tempo
- A regra do write de Thomas permite maior concorrência em potencial. Diferente dos protocolos anteriores, ela permite alguns schedules de seriação por view que não são seriáveis por conflito

Protocolo baseado em validação

- A execução da transação T_i é feita em três fases.
  - Fase de leitura e execução: A transação T_i escreve apenas em variáveis locais temporárias
  - Fase de validação: A transação T_i realiza um "teste de validação" para determinar se as variáveis locais podem ser escritas sem violar a seriação
  - Fase de escrita: Se T_i for validada, as atualizações são aplicadas ao banco de dados; caso contrário, T_i é revertida
- As três fases da execução simultânea de transações podem ser intercaladas, mas cada transação precisa passar pelas três fases nessa ordem
- Também chamado controle de concorrência otimista, pois a transação é executada totalmente na esperança de que tudo correrá bem durante a validação

Protocolo baseado em validação (cont.)

- Cada transação T_i possui três estampas de tempo
  - Start(T_i): a hora em que T_i iniciou sua execução
  - Validation(T_i): a hora em que T_i entrou em sua fase de validação
  - Finish(T_i): a hora em que T_i concluir sua fase de escrita
- A ordem de seriação é determinada pela estampa de tempo dada na hora da validação, para aumentar a concorrência. Assim, TS(T_i) recebe o valor de Validation(T_i)
- Esse protocolo é útil e oferece maior grau de concorrência se a probabilidade de conflitos for baixa. Isso porque a ordem de seriação não é decidida previamente e relativamente menos transações terão que ser revertidas

Teste de validação para a transação Tj

- Se para toda T_i com TS (T_i) < TS(T_j) qualquer uma destas condições é mantida:
  - finish(T_i) < start(T_j)
  - start(T_j) < finish(T_i) < validation(T_j) e o conjunto de itens de dados escritos por T_i não coincidir com o conjunto de itens de dados lidos por T_j.
- então a validação tem sucesso e T_j pode ser confirmada. Caso contrário, a validação falha e T_j é abortada.
- Justificativa: Ou a primeira condição é satisfeita, e não existe execução sobreposta, ou a segunda condição é satisfeita e
  - as escritas de T_i não afetam as leituras de T_j, pois ocorrem após T_i ter concluído suas escritas.
  - as escritas de T_i não afetam as leituras de T_j, pois T_j não lê qualquer item escrito por T_i

Schedule produzido por validação

- Exemplo de schedule produzido usando validação

T_1	T_2
read(b)	
	read(b)
	b := b – 50
	read(a)
	a := a + 50
read(a)	
(validate)	
display(a+b)	
	(validate)
	write(b)
	write(a)

Esquemas multiversão

- Esquemas multiversão mantêm versões antigas do item de dados para aumentar a concorrência
  - ordenação de estampa de tempo multiversão
  - Bloqueio em duas fases multiversão
- Cada write bem sucedido resulta na criação de uma nova versão do item de dados escrito
- Use estampas de tempo para rotular versões
- Quando uma operação read(Q) for emitida, selecione uma versão apropriada de Q com base na estampa de tempo da transação, e retorne o valor da versão selecionada
- reads nunca têm que esperar que uma versão apropriada seja retornada imediatamente

Ordenação de estampa de tempo multiversão

- Cada item de dados Q tem uma sequência de versões < $\boldsymbol{Q_1}$, $\boldsymbol{Q_2}$,…., $\boldsymbol{Q_m}$>. Cada versão $\boldsymbol{Q_k}$ contém três campos de dados:
  - Conteúdo - o valor da versão $\boldsymbol{Q_k}$
  - W-timestamp($\boldsymbol{Q_k}$) - estampa de tempo da transação que criou (escreveu) a versão $\boldsymbol{Q_k}$
  - R-timestamp($\boldsymbol{Q_k}$) - maior estampa de tempo de uma transação que leu com sucesso a versão $\boldsymbol{Q_k}$
- quando uma transação $\boldsymbol{T_i}$ cria uma nova versão $\boldsymbol{Q_k}$ de Q, W-timestamp e R-timestamp de $\boldsymbol{Q_k}$ são inicializados como TS($\boldsymbol{T_i}$)
- R-timestamp de $\boldsymbol{Q_k}$ é atualizado sempre que uma transação $\boldsymbol{T_j}$ lê $\boldsymbol{Q_k}$, e TS($\boldsymbol{T_j}$) > R-timestamp($\boldsymbol{Q_k}$)

Ordenação de estampa de tempo multiversão (cont)

- O esquema de estampa de tempo multiversão apresentado em seguida garante a seriação
- Suponha que a transação T_i emita uma operação read(Q) ou write(Q). Considere que Q_k indique a versão de Q cuja estampa de tempo de escrita é a maior estampa de tempo de escrita menor ou igual a TS(T_i)
  - Se a transação T_i emitir um read(Q), então o valor retornado é o conteúdo da versão Q_k
  - Se a transação T_i emitir um write(Q), e se TS(T_i) < R-timestamp(Q_k), então a transação T_i é revertida. Caso contrário, se TS(T_i) = W-timestamp(Q_k), o conteúdo de Q_k é modificado, caso contrário uma nova versão de Q é criada
- Leituras sempre têm sucesso; uma escrita por T_i é rejeitada se alguma outra transação T_j que (na ordem de seriação definida pelos valores de estampa de tempo) tiver que ler a escrita de T_i, já tiver lido uma versão criada por uma transação mais antiga que T_i

Bloqueio em duas fases multiversão

- Diferencia entre transações somente leitura e transações de atualização
- Transações de atualização adquirem bloqueios de leitura e escrita, e mantêm todos os bloqueios ativos até o final da transação. Ou seja, transações de atualização seguem o bloqueio rigoroso em duas fases
  - Cada write bem-sucedido resulta na criação de uma nova versão do item de dados escrito
  - cada versão de um item de dados tem uma única estampa de tempo cujo valor é obtido de um contador ts-counter que é incrementado durante o processamento do commit
- Transações somente de leitura recebem uma estampa de tempo lendo o valor atual de ts-counter antes de começar a execução; elas seguem o protocolo de ordenação de estampa de tempo multiversão para realizar leituras

Bloqueio em duas fases multiversão (cont.)

- Quando uma transação de atualização quiser ler um item de dados, ela obtém um bloqueio compartilhado nele e lê a versão mais recente
- Quando quiser escrever um item, ela obtém o bloqueio X; depois, cria uma nova versão do item e define a estampa de tempo dessa versão como ∞
- Quando a transação de atualização T_i terminar, ocorre o processamento de confirmação:
  - T_i define estampa de tempo nas versões que criou como ts-counter + 1
  - T_i incrementa ts-counter em 1
- Transações somente de leitura que começam após T_i incrementar ts-counter verão os valores atualizados por T_i
- Transações somente de leitura que começam antes que T_i incremente o ts-counter verão o valor antes das atualizações por T_i
- Somente schedules seriáveis são produzidos

Tratamento de impasse

- Considere as duas transações a seguir:

T_1	T_2
write(x)	
	write(y)
	write(x)
write(y)	

- Schedule com impasse

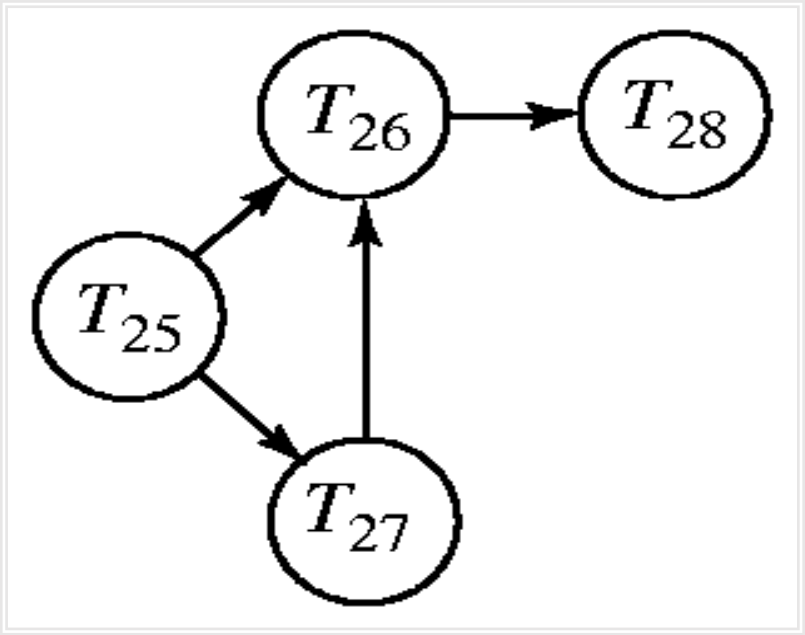
T_1	T_2
lock-X on X	
write(x)	
	lock-X on Y
	Write(Y)
	wait for lock-X on X
wait for lock-X on Y	

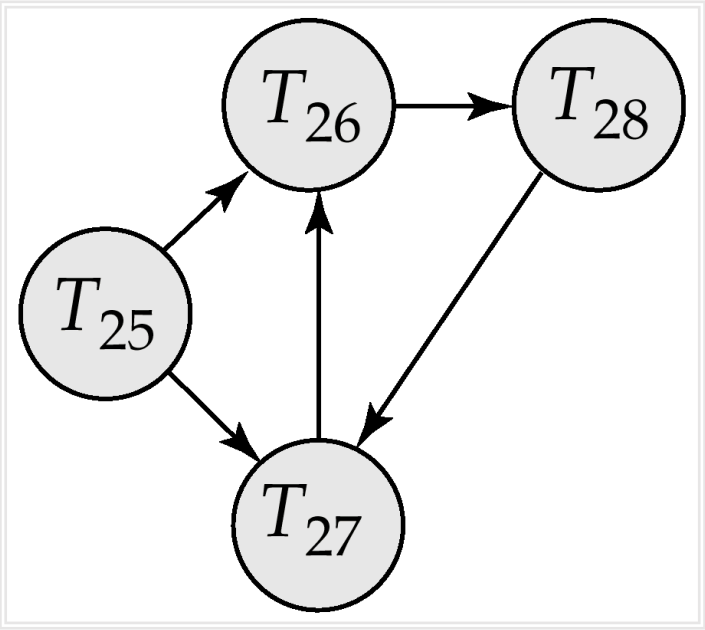
Detecção de impasse

- Os impasses podem ser descritos como um grafo de espera, que consiste em um par G = (V,E),
  - V é um conjunto de vértices (todas as transações no sistema)
  - E é um conjunto d arestas; cada elemento é um par ordenado $T_i \rightarrow T_j$
- Se $T_i \rightarrow T_j$ está em E, então existe uma aresta dedicada de T_i para T_j, implicando que T_i está esperando que T_j libere um item de dados
- Quando T_i solicita um item de dados atualmente mantido por T_j, então a aresta T_i, T_j é inserida no grafo de espera. Essa aresta só é removida quando T_j não está mais mantendo um item de dados necessário por T_i
- O sistema está em um estado de impasse se e somente se o grafo de espera tiver um ciclo. Precisa invocar um algoritmo de detecção de impasse periodicamente para procurar ciclos

Detecção de impasse (cont.)

- grafo de espera sem um ciclo grafo de espera com um ciclo

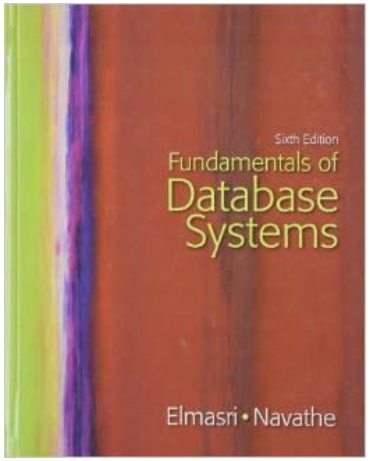


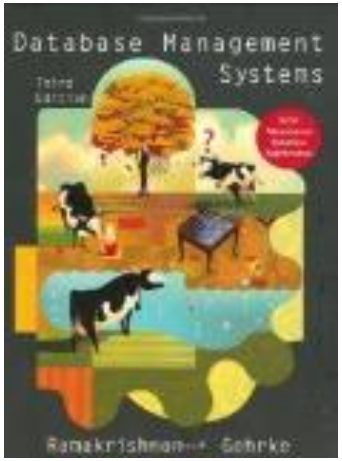


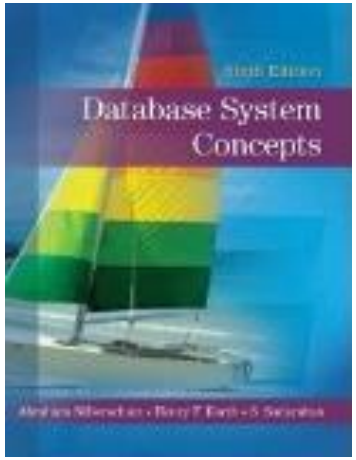
Recuperação de impasse

- Quando o impasse for detectado:
  - Alguma transação terá que ser revertida (uma vítima) para romper o impasse Selecione essa transação como a vítima que terá o menor custo
  - Rollback - determine até onde reverter a transação
    - Rollback total: Aborte a transação e depois reinicie-a
    - Mais eficiente reverter a transação somente até o ponto necessário para romper o impasse
  - A inanição acontece se a mesma transação sempre for escolhida como vítima Inclua o número de rollbacks no fator de custo para evitar inanição

Referências







A maioria dos slides apresenta exemplos do livro Database System Concepts