Conceitos de Processamento de Transações

Laboratório de Bancos de Dados

Prof. José de Jesús Pérez-Alcázar

Introdução

- Transação → mecanismo para descrição de unidades lógicas de processamento de BDs.
- Sistemas de processamento de transação → sistemas com grandes BDs e centenas de usuários executando transações concorrentes nos BDs: Sistemas de Reservas, Bancos, etc.

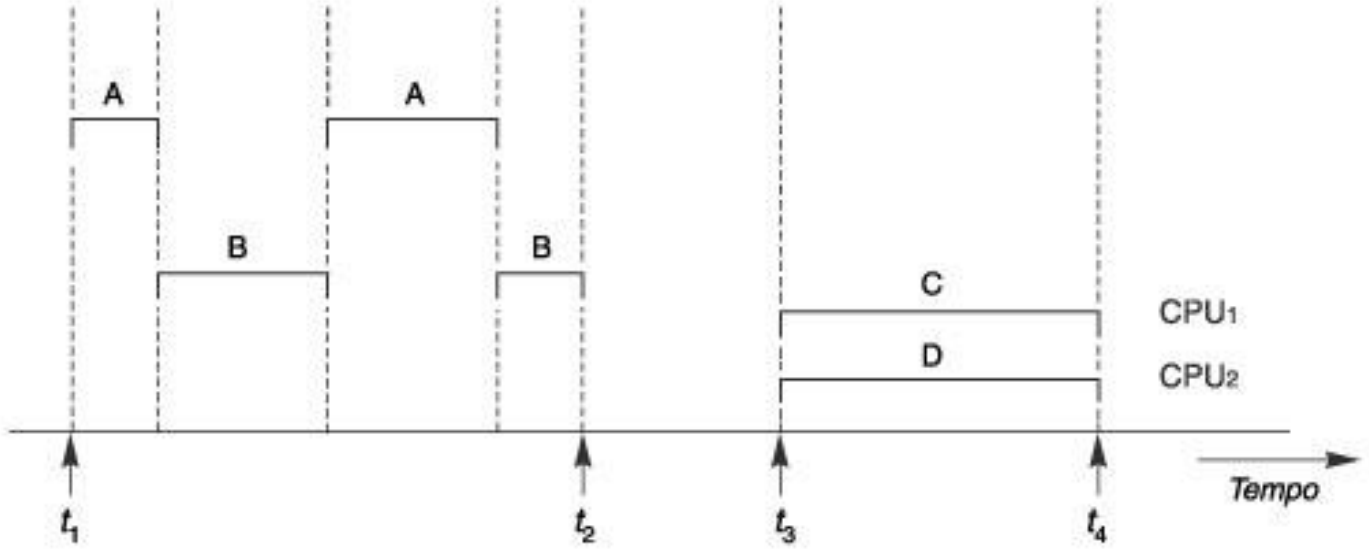
Agenda

- Porquê o controle de concorrência e a recuperação são necessários nos SBDs?
- Conceitos básicos
- Propriedades ACID.
- Planos de execução de transações (histórico)
- Serialidade
- Facilidades de suporte de transações em SQL

Introdução: Sistemas Monousuários vs. Multiusuários

- Sistemas de BDs classificados pelo número de usuários que podem usar o sistema concorrentemente.
- Monousuários: sistemas de computação pessoal; multiusuários: maioria dos SGBDs.
- Multiprogramação → execução diversos processos ao mesmo tempo, com 1 CPU (intercalada) ou mais (paralelo). Veja Fig.
- Maior parte da teoria de controle de concorrência foi desenvolvida em termos da concorrência intercalada.

Processamento intercalado *versus* processamento paralelo transações concorrentes



Transações, Operações de Leitura e Escrita e Buffers de SGBD

- Transação → programa em execução que forma uma unidade lógica de processamento no BD. Inclui uma ou mais operações de acesso ao BD. Estas operações podem estar embutidas (Prog. Apl.) ou especificada interativamente.
- Especificação → **inicio de trans.** e **fim de trans.**
- Um programa de aplicação pode ter mais de uma transação
- Transação de leitura (read-only) : não contem operações de atualização

Transações, Operações de Leitura e Escrita e Buffers de SGBD

- Modelo de BDs para explicar conceitos: BD representado por itens de dados denominados. Tamanho do item (granularidade) – campo, registro, bloco, etc.
- Operações:
  - Ler_item(X): Lê um item do BD chamado X numa variável do programa.
  - Gravar_item(X): grava o valor de uma variável de programa X num item do BD chamado X.

Transações, Operações de Leitura e Escrita e Buffers de SGBD

- Ler_item(X):
  - Encontrar o endereço do bloco de disco que contem X
  - Copiar esse bloco para um buffer (se o bloco ainda não estiver em algum buffer)
  - Copiar item X do buffer para a variável de programa chamada X
- Gravar_item(X):
  - Encontrar o endereço do bloco de disco que contem X
  - Copiar esse bloco para um buffer (se o bloco ainda não estiver em algum buffer)
  - Copiar o item X de uma variável de programa chamada X para seu local correto no buffer
  - Armazenar o bloco atualizado no buffer de volta para o disco (pode ser posteriormente)
- Necessário políticas de administração do buffer.

Duas transações simples. (a) Transação T_1. (b) Transação T_2.

(a) T_1

```
ler_item (X);
X:=X-N;
escrever_item (X);
ler_item (Y);
Y:=Y+N;
escrever_item (Y);
```

(b) T_2

```
ler_item (X);
X:=X+M;
escrever_item (X);
```

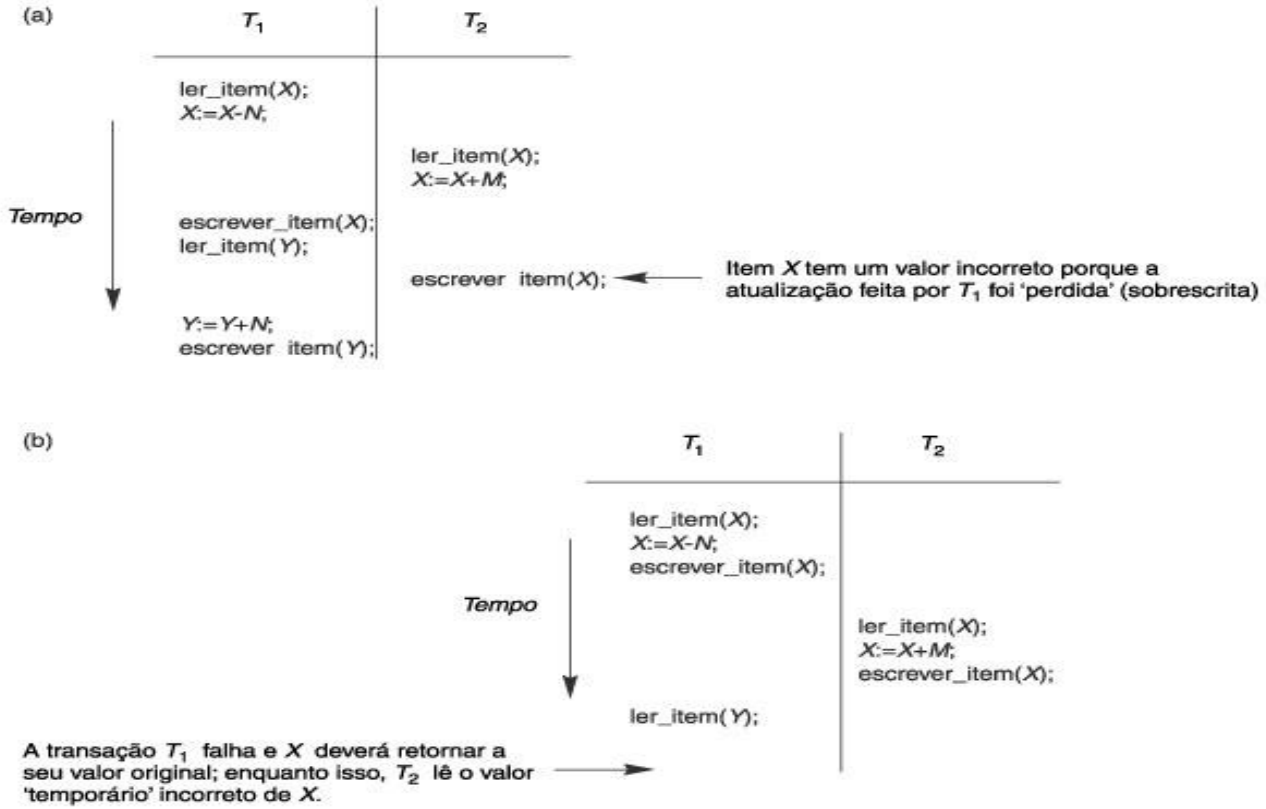
Conjunto_leitura de uma transação: todos os itens que a transação ler.

Conjunto_gravação: todos os itens que a transação gravar.

Por que o Controle de Concorrência é Necessário

- Diversos problemas podem ocorrer quando transações concorrentes são executadas descontroladamente.
- BD muito simplificado de reservas de uma companhia aérea. Armazenado um registro para cada vôo.
- A figura acima apresenta Transação T1 que transfere N reservas de um vôo X para Y; T2 apenas reserva M assentos do X.

- Suponhamos que T1 e T2 são executadas concorrentemente. Podem acontecer vários tipos de problemas:
  - O problema da atualização perdida (Veja Fig. (a)).
  - O problema da Atualização Temporária ou leitura de sujeira (Veja Fig. (b)).
  - O problema do Sumário Incorreto (Veja Fig. (c)).T3 não contabilizará N.


(a)
T1
T2
ler_item(X);
X:=X-N;
ler_item(X);
X:=X+M;
Tempo
escrever_item(X);
ler_item(Y);
escrever item(X);
Item X tem um valor incorreto porque a atualização feita por T1 foi 'perdida' (sobrescrita)
Y:=Y+N;
escrever item(Y);
(b)
T1
T2
ler_item(X);
X:=X-N;
escrever_item(X);
Tempo
ler_item(X);
X:=X+M;
escrever_item(X);
ler_item(Y);
A transação T1 falha e X deverá retornar a seu valor original; enquanto isso, T2 lê o valor 'temporário' incorreto de X.
(c)
T1
T3
sum:=0;
ler_item(A);
sum:=sum+A;
ler_item(X);
X:=X-N;
escrever_item(X);
ler_item(X);
sum:=sum+X;
ler_item(Y);
sum:=sum+Y;
T3 lê X depois da subtração de N e lê Y antes da adição de N; o resultado é um sumário errado (sem N).
ler_item(Y);
Y:=Y+N;
escrever_item(Y);

Por que a Recuperação é Necessária

- Se uma transação for executada por um SGBD, o sistema deverá garantir:
  1. Todas as operações na transação foram completadas com sucesso e seu efeito será gravado permanentemente no BD ou
  2. A transação não terá nenhum efeito sobre o BD ou sobre quaisquer outras transações
- O SGBD não deverá permitir que algumas das operações sejam executadas enquanto que outras não. Por falhas.

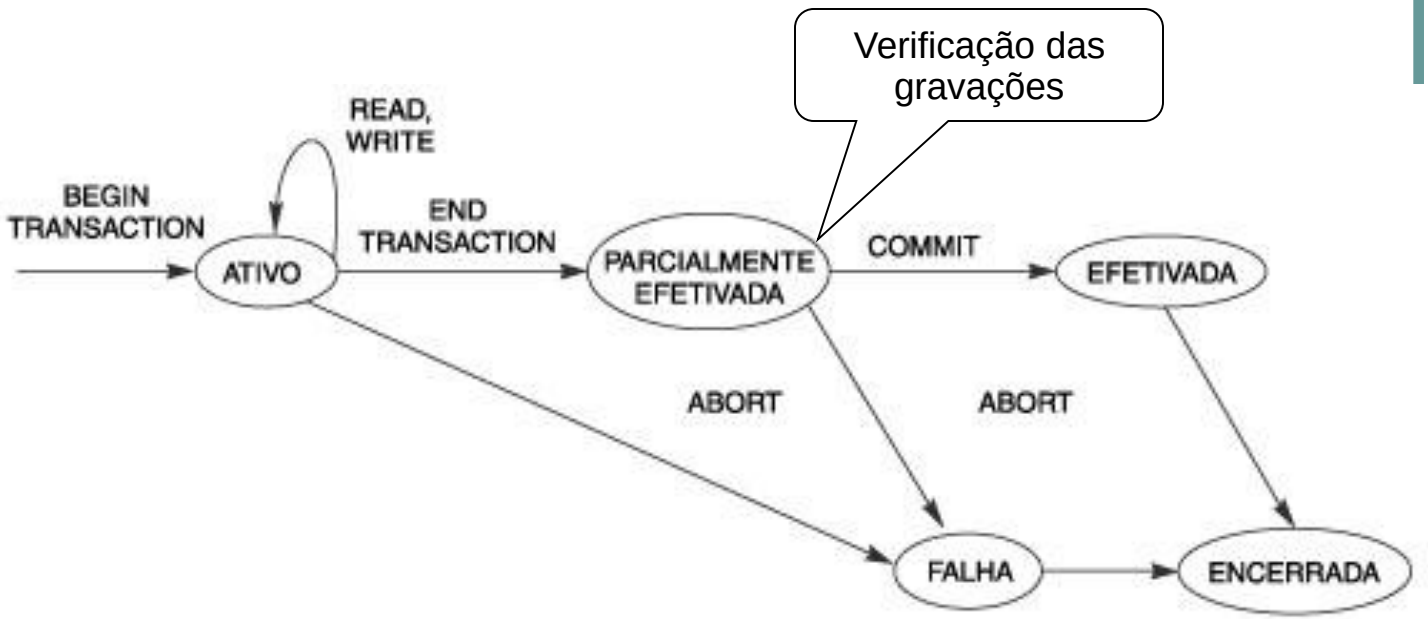
Por que a Recuperação é Necessária

- Tipos de falhas: de transação, sistema ou mídia. Razões para falha:
  1. O computador falhar: erro de hardware, software ou rede.
  2. Erro de transação ou sistema: operação de transação pode causar falha – divisão por zero e erro de lógica.
  3. Erros locais ou condições de exceção detectadas pela transação
  4. Imposição de controle de concorrência
  5. Falha de disco
  6. Problemas físicos e catástrofes
- Falhas dos tipos 1,2, 3 e 4 são mais freqüentes que as dos tipos 5 ou 6. Falhas do tipo 1 a 4, o sistema deverá manter informações suficientes para se recuperar dessa falha. 5 e 6, precisa restauração (“backup”)

Conceitos de Transação e Sistema: Estados e operações adicionais

- Uma transação é uma unidade atômica de trabalho (estará completa ou não foi realizada). Para a recuperação, o sistema precisa de manter o controle de quando a transação inicia, termina e de sua efetivação ou interrupção. O administrador de recuperações controla:
  - BEGIN_TRANSACTION
  - READ ou WRITE
  - END_TRANSACTION
  - COMMIT_TRANSACTION
  - ROLLBACK (ou ABORT)

Diagrama de transição de estado ilustrando os estados de execução de uma transação.



O Log (Registro de Ocorrências) do Sistema

- Útil para recuperação de falhas que afetam as transações. É mantido em disco para que não seja afetado por nenhum tipo de falha com exceção das catastróficas e de disco.
- O log deve ser periodicamente gravado em disco.

O Log (Registro de Ocorrências) do Sistema

- Entradas de um log (Registros de log)

  (T → id_transação)

  1. [start_transaction, T]
  2. [escrever_item, T, X, valor_antigo, novo_valor]
  3. [ler_item, T, X]
  4. [commit, T]: a transação T foi completada com sucesso e seus efeitos devem ser efetivados no BD.
  5. [abort, T]

O Log (Registro de Ocorrências) do Sistema

- Protocolos de recuperação (restauração) que evitam reversões em cascata (quase todos os protocolos práticos) → log sem ler_item. Mas se o log for para auditoria → este tipo de entrada é necessária.
- Para fazer a recuperação do sistema a partir do log são necessárias operações de desfazer (undo) e refazer (redo).
- Refazer → necessário se todas as atualizações de uma T estiverem registradas no log, mas pode ocorrer uma falha antes que se possa garantir que todos os novos_valores foram gravados permanentemente.

Ponto de efetivação (Commit point) de uma Transação

- Quando todas as operações de uma transação T que acessam o BD foram executadas com sucesso e o efeito delas foi gravado no log a transação alcança seu ponto de efetivação → após esse ponto seus efeitos serão gravadas de modo permanentemente no BD, e é gravado no log um registro de efetivação [commit, T]
- Uma falha no sistema → desfaz transações sem registro de efetivação no log e refaz transações com registro de efetivação.
- As informações do log são gravados no disco mas antes são gravadas em buffers. No caso de falhas, apenas as entradas do log gravadas no disco serão consideradas no processo de recuperação. Então, antes de uma transação alcançar seu ponto de efetivação → gravação forçada

Propriedades Desejáveis das Transações

- Propriedades ACID → impostas pelo controle de concorrência e métodos de recuperação do SGBD
  - **A**tomicidade
  - Preservação da **C**onsistência: a execução de uma transação deve preservar a consistência do BD.
  - **I**solamento: Uma transação deve ser executada como se estiver isolada das demais
  - **D**urabilidade ou permanência: as mudanças aplicadas ao BD por uma transação efetivada devem persistir sem importar as falhas.
- Atomicidade → responsabilidade do módulo de recuperação

Propriedades Desejáveis das Transações

- Assumindo que não exista nenhuma interferência de outras transações, a preservação de consistência deve ser mantida desde que o programador tenha feito bem seu trabalho.
- O Isolamento é imposto pelo subsistema de controle de concorrência
- A propriedade de durabilidade é responsabilidade do susbsistema de recuperação.

Definindo Plano de Execução (Schedule) Baseado na Recuperação

- Quando transações são executadas concorrentemente de maneira intercalada, a ordem de execução das operações das várias transações → plano de execução (ou histórico)
- Um histórico S de n transações T1, T2, ...,Tn é a ordenação das operações das transações tal que para cada Ti as operações de Ti em S deverão aparecer na mesma ordem em que elas ocorrem em Ti

Definindo Plano de Execução (Schedule) Baseado na Recuperação

- Para a recuperação e controle de concorrência o interesse é nas operações ler_item (r) , escrever_item (w), commit (c) e abort (a).
- Exemplo o plano da fig 17(a) → S_a pode ser escrito: r1(X); r2(X); w1(X); r1(Y); w2(X); w1(Y)
- E S_b: r1(X); w1(X); r2(X); w2(X); r1(Y); a

Definindo Plano de Execução (Schedule) Baseado na Recuperação

- Duas operações num plano são ditas em conflito se elas satisfazerem:
  - Pertencerem a diferentes transações;
  - Acessarem o mesmo item X; e
  - Pelo menos uma das operações ser um escrever_item(X).
- Ex: No plano S_a as operações r1(X) e w2(X)

Definindo Plano de Execução (Schedule) Baseado na Recuperação

- Um plano S, com n transações T1, T2, ...,Tn é chamado um plano completo se forem garantidas as seguintes condições:
  1. As operações em S são exatamente as operações em T1,...,Tn, tendo um commit ou um abort como última operação de cada transação no plano.
  2. Para quaisquer pares de operações da mesma transação Ti, sua ordem de aparecimento em S será a mesma que em Ti.
  3. Para quaisquer duas operações conflitantes, uma das duas precisa aparecer antes da outra no plano.
- Operações não conflitantes – ordenação parcial

Caracterizando um plano baseado na recuperação

- Em alguns planos é fácil a restauração de transações em outros não → quais são as características
- Deve ser garantida que uma vez efetivada uma transação T, nunca mais ela seja revertida. Planos que buscam esses critérios são chamados de recuperáveis → um plano S é restaurável se nenhuma transação T de S for efetivada até que todas as transações T´, que tiverem gravado um item lido por T, tenham sido efetivadas. Além disso, T´não deve ser abortada antes que T leia o item X, e não deve haver transações que gravem X depois da gravação de T´e antes da leitura de T (não abortadas antes que T tivesse lido X).

Caracterizando um plano baseado na recuperação

- Planos restauráveis exigem um processo de restauração complexo → necessário informação suficiente no log.
- Os planos S_a e S_b, da fig 17(a) são ambos restauráveis.
- Seja S_a´: r1(X); r2(X); w1(X); r1(Y); w2(X); c2; w1(Y); c1;
- S_a´ é restaurável, ainda que sofra do problema da perda de atualização.

Caracterizando um plano baseado na recuperação

- Considere os planos parciais:
  - S_c: r1(X); w1(X); r2(X); r1(Y); w2(X); c2; a1;
  - S_d: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); c1; c2;
  - S_e: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); a1; a2;
- S_c não é restaurável porque T_2 lê o item X de T_1, e T_2 é efetivada antes que T_1 o seja. Para que o plano o seja, a operação c_2 deverá ser adiada até que T_1 seja efetivada como em S_d; se T_1 abortar, então T_2 deverá abortar também, como em S_e.

Caracterizando um plano baseado na recuperação

- Num plano restaurável nenhuma transação efetivada jamais deveria precisar ser revertida. Mas, pode ocorrer uma reversão em cascata, quando uma transação não efetivada tem de ser revertida porque leu um item de uma transação que falhou. Veja plano Se. T2 foi revertida porque leu um item de T1 e T1 abortou.
- Em um plano livre de cascata as transações só lêem itens que foram gravados por transações efetivadas.
- Existe o plano restrito, no qual as transações não podem nem ler nem gravar um item X até que a última transação que grave X tenha sido efetivada (ou abortada). Restauração mais simples.

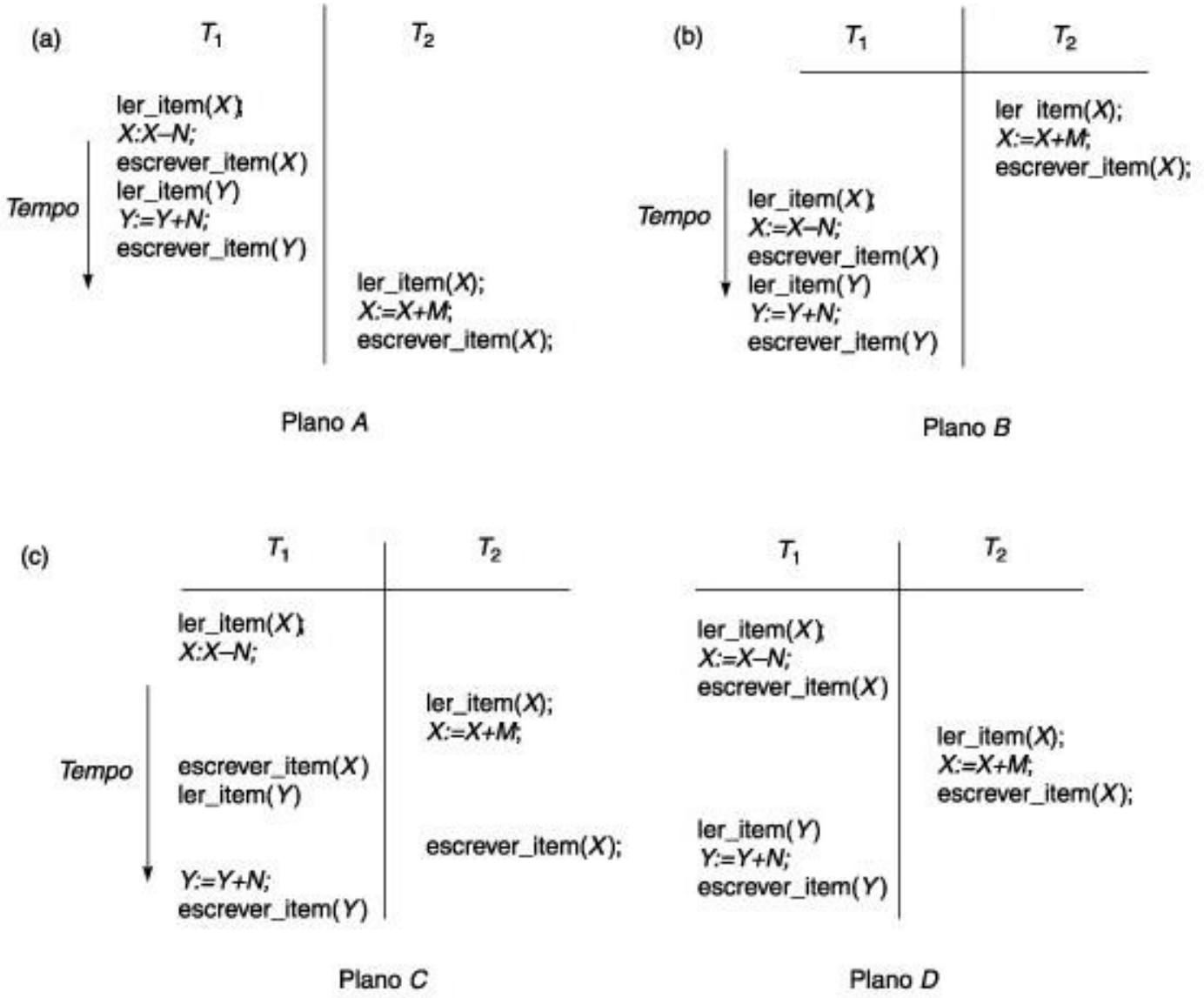
Caracterizando um plano baseado na recuperação

- Considere o plano S_f:$w_1(X,5)$; $w_2(X,8)$; a_1;
- Suponha valor de X fosse 9 (imagem anterior de w1. Se T1 abortar, o procedimento de restauração simples tentará voltar o valor de X para 9 sem importar que T2 colocou o valor em 8.
- Sf não é restrito mas é livre de reversão em cascata.
- Todos os planos restritos são livres de reversão em cascata, e todos os planos livres de RC são restauráveis.

Definindo Planos de Execução (Schedule) Baseados em Serialidade

- Caracterizaremos agora os tipos de planos que são considerados corretos quando transações concorrentes estiverem em execução.
- A figura seguinte apresenta as formas de combinar T1 e T2. a) e b) estão em seqüência. Quando a intercalação é possível há muitas maneiras, mas quais são corretas?

Figura 17.5 Exemplos de planos seriais e não-seriais envolvendo as transações T_1 e T_2. (a) Plano serial *A*: T_2, seguido por T_2. (b) Plano serial *B*: T_2 seguido por T_1. (c) Dois planos não-seriais, *C* e *D*, com intercalação de operações.



Planos Seriais, Não-Seriais e de Conflitos Serializáveis

- Os planos a) e b) da figura são seriais: são executadas consecutivamente, sem intercalação. Os planos C e D são não-seriais: intercalação de operações.
- Todo plano serial é correto.
- Problema dos planos seriais → limitam a concorrência ou a intercalação de operações → E/S e transações longas. Por isso, inaceitáveis.
- Consideremos os planos da fig. com valores iniciais X=90, Y=90 e que N=3 e M=2. Depois da execução dos planos seriais a) e b), X=89 e Y=93 → resultados corretos.

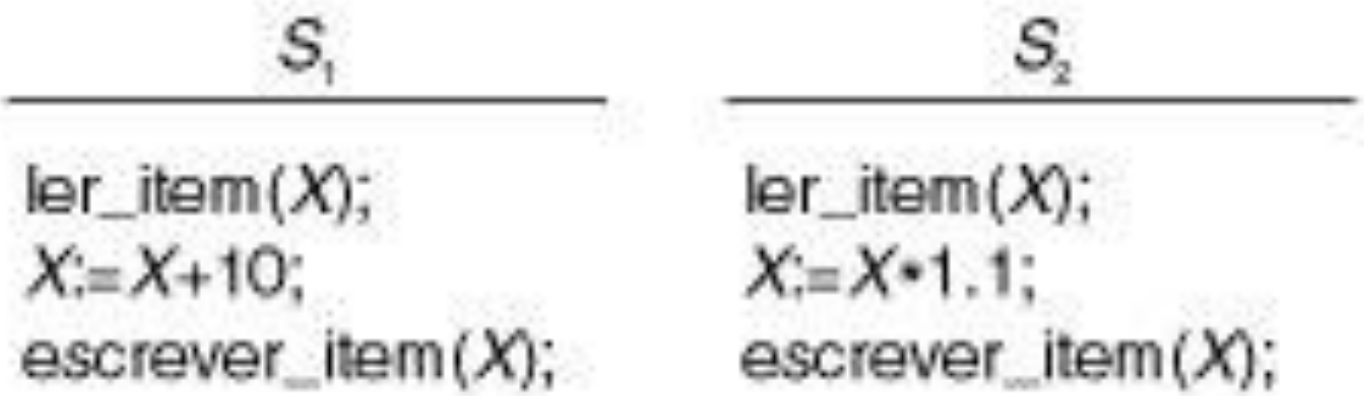
Planos Seriais, Não-Seriais e de Conflitos Serializáveis

- Agora o plano c) resulta em X=92 e Y=93; o plano d) fornece resultados corretos. O plano c) fornece um resultado errôneo por causa do problema das atualizações perdidas.
- Porque alguns planos não-seriais fornecem resultados corretos e outros não?
- O conceito de serialidade vai nos ajudar a caracterizar esses planos.
- Um plano S com n transações é serializável se ele for equivalente a algum plano serial com as mesmas n transações.
- O que é equivalência de planos?

Planos Seriais, Não-Seriais e de Conflitos Serializáveis

- Dizer que um plano S não-serial é serializável é o mesmo que dizer que ele é correto porque ele é equivalente a um plano serial que é considerado correto.
- Definição de equivalência mais simples mas menos satisfatória → produzem resultados iguais.
- Exemplo da fig. 17.6 – os planos S1 e S2 para X=100 produzirão os mesmos resultados.

Figura 17.6 **Dois planos que têm resultados equivalentes quando o valor inicial de *X* = 100, mas que, no geral, não são resultados equivalentes.**



Planos Seriais, Não-Seriais e de Conflitos Serializáveis

- Dois planos são conflito equivalentes se a ordem de quaisquer duas operações conflitantes for a mesma em ambos os planos.
- Operações conflitantes: pertencem a diferentes transações usam o mesmo item do BD e pelo menos uma das duas é de gravação do item.
- Exemplo se em S1 ocorrer r1(X); w2(X); e w2(X); r1(X) em S2

Planos Seriais, Não-Seriais e de Conflitos Serializáveis

- Um plano S é conflito serializável se ele for (conflito) equivalente a um plano serial S´.
- Então o plano d) da figura 17.5 será equivalente ao plano serial a) da mesma figura.
- O plano c) não é equivalente a nenhum dos planos seriais possíveis a) e b)

Testando o Conflito Serialidade de um Plano

- Há um algoritmo simples para determinar o conflito de serialidade de um plano. A maioria dos métodos de controle de concorrência não testa a serialidade. Foram desenvolvidos para garantir a serialidade.
- O algoritmo de teste usa um grafo de precedência (dirigido) ou serialização. Os nós são as transações e existe uma aresta da T_i para T_k, se alguma das operações de T_j aparece no plano antes de alguma operação conflitante de T_k.

Testando o Conflito Serialidade de um Plano

- Algoritmo: teste de conflito
  1. Para cada transação Ti do plano S, criar um nó rotulado Ti
  2. Para cada caso em S em que Tj executar um ler_item(X) depois que uma Ti executar um escrever_item(X), criar uma seta (Ti → Tj) no grafo
  3. Para cada caso em S em que Tj executar um escrever_item(X) depois que uma Ti executar um ler_item(X), criar uma seta (Ti → Tj) no grafo
  4. Para cada caso em S em que Tj executar um escrever_item(X) depois que uma Ti executar um escrever_item(X), criar uma seta (Ti → Tj) no grafo
  5. O plano S será serializável se, e apenas se, o grafo de precedência não contiver ciclos.

Figura 17.7 Construindo os grafos de precedência para teste de conflito serialidade dos planos *A* a *D* da Figura 17.5. (a) Grafo de precedência para o plano serial *A*. (b) Grafo de precedência para o plano serial *B*. (c) Grafo de precedência para o plano *C* (não serializável). (d) Grafo de precedência para o plano *D* (serializável, equivalente ao plano *A*).

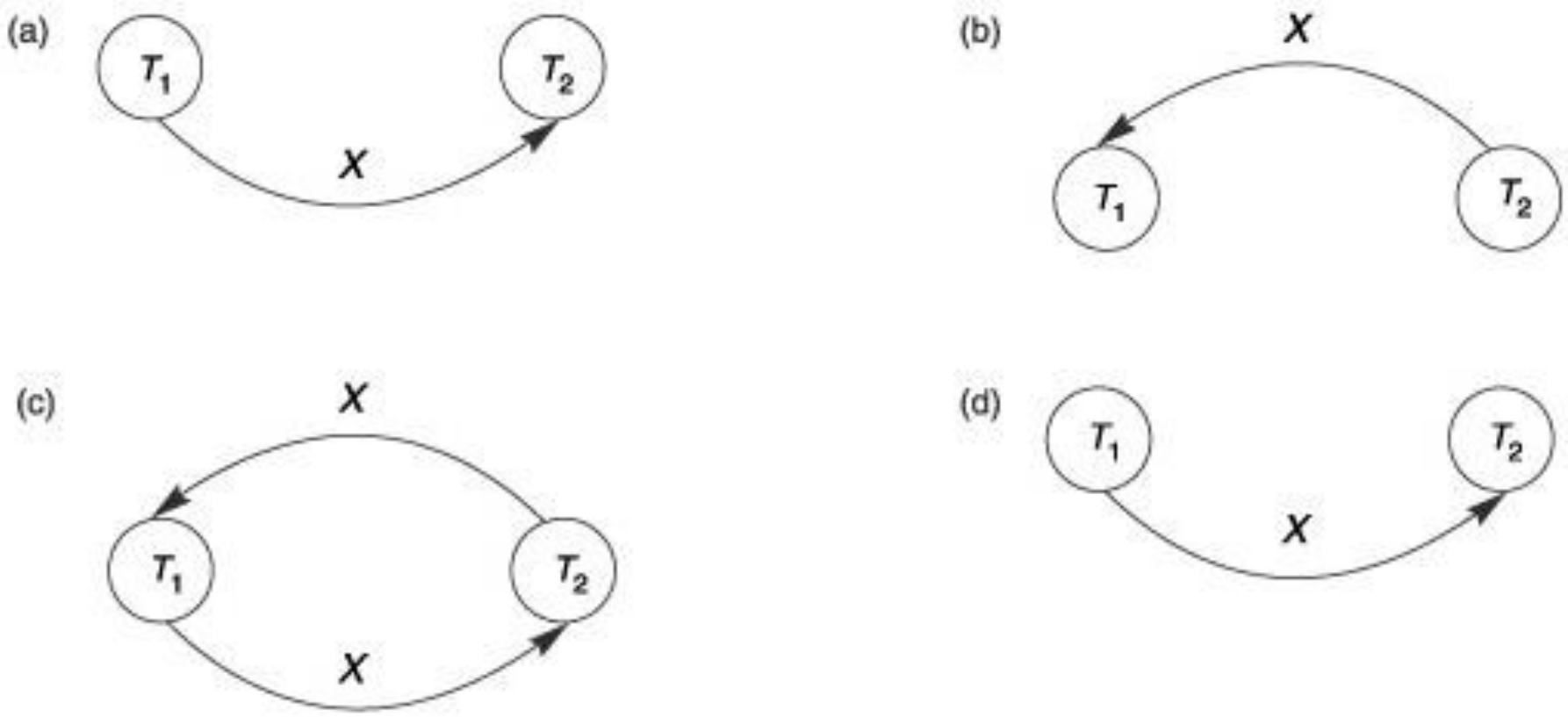
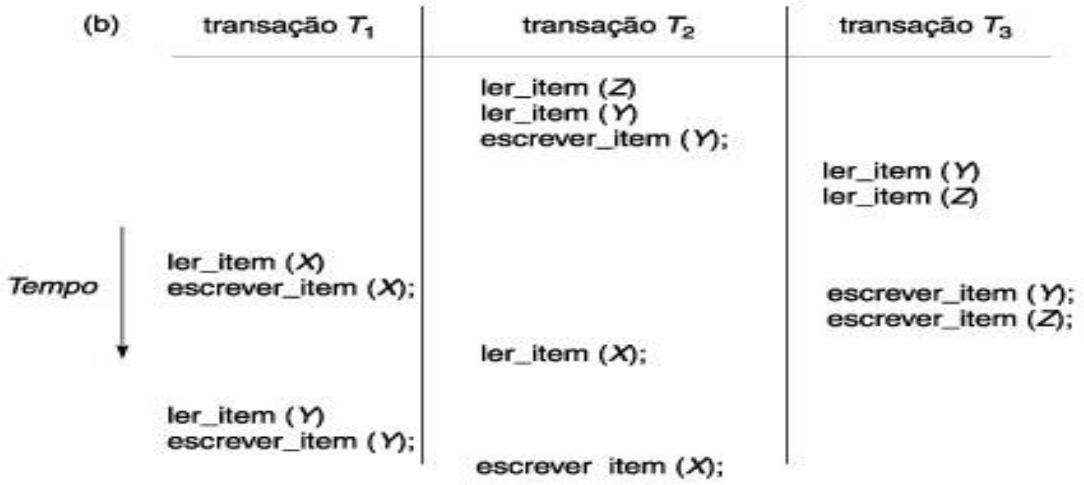
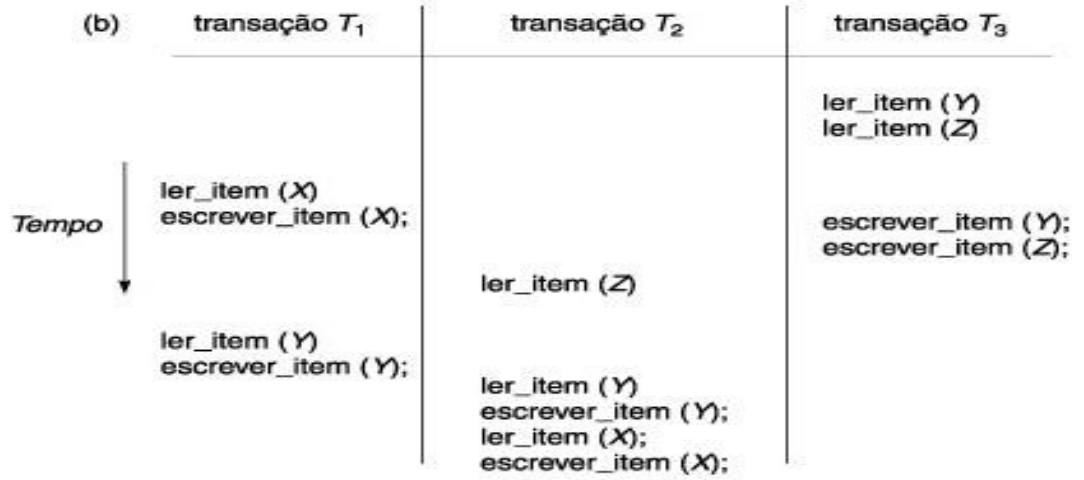


Figura 17.8 Outro exemplo de teste de serialidade. (a) As operações READ **e** WRITE **das três transações** T_1, T_2, **e** T_3. **(b) Plano** *E*. **(c) Plano** *F*. (***continua***)



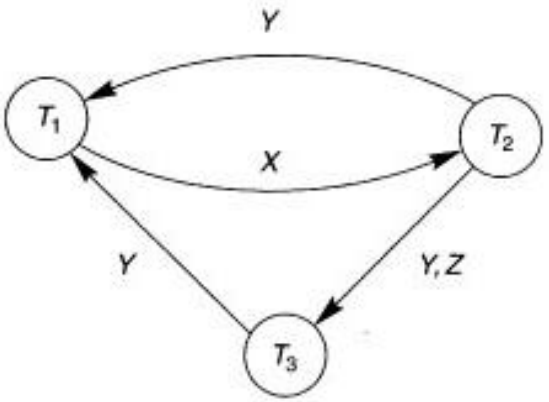
Plano E



Plano F

Figura 17.8 Outro exemplo de teste de serialidade. (d) Grafo de precedência para o plano *E*. (e) Grafo de precedência para o plano *F*. (f) Grafo de precedência com dois planos seriais equivalentes.

(d)



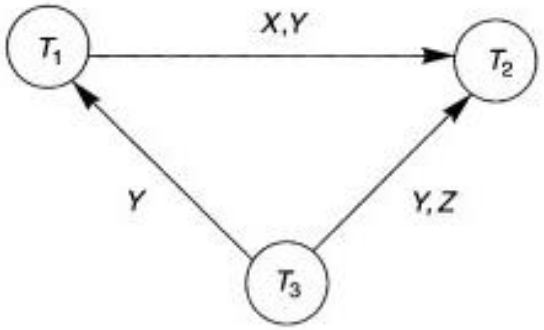
Planos seriais equivalentes

Nenhuma

Razão

ciclo $X(T_1 \rightarrow T_2)$, $Y(T_2 \rightarrow T_1)$
ciclo $X(T_1 \rightarrow T_2)$, $YZ(T_2 \rightarrow T_3)$, $Y(T_3 \rightarrow T_1)$

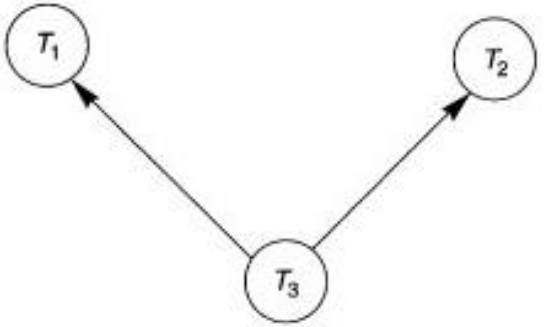
(e)



Planos seriais equivalentes

$T_3 \rightarrow T_1 \rightarrow T_2$

(f)



Planos seriais equivalentes

$T_3 \rightarrow T_1 \rightarrow T_2$

$T_3 \rightarrow T_2 \rightarrow T_1$

Aplicações da Serialidade

- S é serializável, equivalente a um plano serial, equivale a S correto. Um plano serializável fornece os benefícios da execução concorrente, sem deixar de ser correto. Na prática, é muito difícil testar a serialidade de um plano. A concorrência é feita pelo SO. Difícil determinar como as operações de um plano serão intercaladas antecipadamente de modo a garantir a serialidade.
- Testar é impraticável.
- Na prática, determinar métodos que garantam a serialidade.

Equivalência de Visão e Visão Serialidade

- Existe outra equivalência de planos menos restritiva → equivalência de visão
- Dois planos S e S´ são visão equivalentes se:
  - O conjunto de transações participantes em S e S´ seja o mesmo e que S e S´ contenham as mesmas operações dessas transações.
  - Para cada operação ri(X) de Ti em S, se o valor X tiver sido alterado por wj(X) de Tj (ou se ele for o valor original de X antes do plano iniciar), a mesma condição deverá ser garantida para o valor X lido por ri(X) de Ti em S´.
  - Se a operação wk(Y) de Tk
    for a última operação a gravar o item Y em S, então wk(Y) de Tk também deverá ser a última operação a gravar Y em S´.

Outros Tipos de Equivalência de Planos

- A serialização de planos é considerada muito restritiva como condição para garantir a corretude de execuções concorrentes. Algumas aplicações podem produzir planos corretos satisfazendo condições menos rigorosas. Conhecimento adicional do domínio é necessário. Ex: transação débito-crédito.

Suporte de Transações em SQL

- Transação em SQL similar aos conceitos de transação já definidos.
- Em SQL não há declarações Begin-Transaction explícita. Ocorre quando declarações SQL são encontradas.
- Toda transação precisa ter uma declaração explicita de término, um COMMIT ou ROLLBACK; e tem características que são especificadas pela declaração SET TRANSACTION.
- As características: modo de acesso, tamanho da área de diagnóstico e nível de isolamento.

Suporte de Transações em SQL

- O modo de acesso: pode ser READ ONLY ou READ WRITE.O padrão é RW, a não ser que seja especificado READ UNCOMMITED como nível de isolamento → RO.

- O nível de isolamento: pode ser READ UNCOMMITED, READ COMMITED, REPEATABLE READ, SERIALIZABLE. Este último é o padrão que permite que não existam violações.

Suporte de Transações em SQL

- Violações:
  - Leitura de sujeira: Uma transação T1 pode ler uma atualização ainda não efetivada de uma transação T2.
  - Leitura não-repetível: Uma transação T1 pode ler um dado valor em uma tabela. Se depois, uma transação T2 atualizar esse valor e T1 lê-lo novamente, T1 enxergará outro valor.
  - Fantasmas: Uma transação T1 pode ler um conjunto de linhas de uma tabela, baseada na cláusula WHERE. Suponha que uma T2 insira uma nova linha que também satisfaça a condição da cláusula WHERE. Se T1 for repetida então vai ler um fantasma, uma linha inexistente antes.

Suporte de Transações em SQL

	Tipo de violação		
Nivel de isolamento	Leitura suja	Não-repetível	Fantasmas
Leitura não efetivada	Sim	Sim	Sim
Leitura efetivada	Não	Sim	Sim
Leitura repetitiva	Não	Não	Sim
Serializável	Não	Não	Não

Suporte de Transações em SQL

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
        READ WRITE
        ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO EMPREGADO (PNOME, UNOME, SSN, DNO, SALARIO)
        VALUES (`Robert´,`Smith´, `991004321´, 2, 35000);
EXEC SQL UPDATE EMPREGADO
            SET SALARIO = SALARIO * 1.1 WHERE DNO = 2;
EXEC SQL     COMMIT;
GOTO THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END: ...;
```

O DBA ou os programadores podem tirar vantagem destas opções