Controle de Concorrência

Banco de Dados II
Prof. Guilherme Tavares de Assis

Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM

Controle de Concorrência

- Técnicas de controle de concorrência são utilizadas para garantir a propriedade de isolamento de transações que estão sendo executadas concomitantemente.
- As principais técnicas de controle de concorrência são:
  - bloqueio (*locking*);
  - ordenamento de registro de *timestamp*;
  - multiversão;
  - validação ou certificação (protocolos otimistas).

Técnicas de Bloqueio

- Bloqueios são usados como um meio de sincronizar o acesso aos itens do banco de dados por transações concorrentes.
  - Um bloqueio consiste em uma variável associada a um item de dado, que descreve o *status* do item em relação a possíveis operações que podem ser aplicadas ao mesmo.
  - Em geral, existe um bloqueio para cada item de dado no banco de dados.
- Duas técnicas de bloqueio são:
  - bloqueio binário;
  - bloqueio múltiplo (compartilhado/exclusivo).

Bloqueio Binário

- Um bloqueio binário possui dois estados:
  - bloqueado (*locked*);
  - desbloqueado (*unlocked*).
- As operações necessárias são:
  - **lock_item(X)**: bloqueia o item X;
  - **unlock_item(X)**: desbloqueia o item X.
- O bloqueio binário impõe a exclusão mútua no item de dado.
  - Se uma operação de bloqueio ou desbloqueio de X for iniciada, nenhum entrelaçamento é permitido até que a operação em questão termine ou a transação espere.
  - O comando de espera coloca a transação em uma fila de espera pelo item X até que o mesmo seja desbloqueado.

Bloqueio Binário

- Algoritmos de bloqueio e desbloqueio do item X:

lock_item(X):

```
B: se LOCK(X) = 0 então  (* item desbloqueado *)
        LOCK(X) ← 1       (* bloquear o item *)
   senão início
        esperar até (LOCK(X) = 0 e o gerenciador de bloqueio despertar
                                                    a transação);
        goto B;
   fim;
```

unlock_item(X):

```
LOCK(X) ← 0;     (* desbloquear o item *)
se alguma transação estiver esperando então
      despertar uma das transações em espera;
```


Bloqueio Binário

- Para que a técnica de bloqueio binário possa ser usada, uma transação T deve obedecer às seguintes regras:
  1. T deve emitir um lock_item(X) antes que qualquer read_item(X) ou write_item(X) seja executado;
  2. T deve emitir um unlock_item(X) depois que todos os read_item(X) e write_item(X) tenham sido completados em T;
  3. T não poderá emitir lock_item(X) se X estiver bloqueado por T;
  4. T poderá emitir um unlock_item(X) apenas se tiver bloqueado X.
- O bloqueio binário é o mecanismo mais simples e mais restrito de controle de concorrência.
  - A implementação requer uma tabela de bloqueios (<nome do item de dado, *lock*, transação>) e uma fila de espera.

Bloqueio Múltiplo

- Um esquema de bloqueio múltiplo (*read/write* ou compartilhado/exclusivo) permite que um item de dado seja acessado por mais de uma transação para leitura.
- As operações necessárias são:
  - **read_lock(X)**: bloqueia o item X para leitura, permitindo que outras transações leiam o item X (bloqueio compartilhado);
  - **write_lock(X)**: bloqueia o item X para escrita, mantendo o bloqueio sobre o item X (bloqueio exclusivo);
  - **unlock(X)**: desbloqueia o item X.
- A implementação do bloqueio múltiplo requer uma tabela de bloqueios (<nome do item de dado, *lock*, número de leituras, transações de bloqueio>) e uma fila de espera.

Bloqueio Múltiplo

- Algoritmos de bloqueio e desbloqueio do item X:

read_lock(X):

```
B: se LOCK(X) = "unlocked" então início  (* item desbloqueado *)
        LOCK(X) ← "read-locked";    (* bloquear o item p/ leitura *)
        num_de_leituras(X) ← 1;
   fim
   senão se LOCK(X) = "read-locked" então   (* bloqueado p/ leitura *)
        num_de_leituras(X) ← num_de_leituras(X) + 1;
   senão início
        esperar até (LOCK(X) = "unlocked" e o gerenciador de bloqueio
                                                    despertar a transação);
        goto B;
   fim;
```

Bloqueio Múltiplo

write_lock(X):

```
B: se LOCK(X) = "unlocked" então    (* item desbloqueado *)
       LOCK(X) ← "write-locked"   (* bloquear o item p/ escrita *)
   senão início
       esperar até (LOCK(X) = "unlocked" e o gerenciador de bloqueio
                                                  desperta a transação);
       goto B;
   fim;
```


Bloqueio Múltiplo

unlock(X):

```
se LOCK(X) = "write_locked" então início  (* bloqueado p/ escrita *)
     LOCK(X) ← "unlocked";    (* desbloquear o item *)
     despertar uma das transações em espera, se houver alguma;
fim
senão se LOCK(X) = "read-locked" então início  (* bloqueado p/ leitura *)
     num_de_leituras(X) ← num_de_leituras - 1;
     se num_de_leituras = 0 então início
          LOCK(X) ← "unlocked";    (* desbloquear o item *)
          despertar uma das transações em espera, se houver alguma;
     fim;
fim;
```


Bloqueio Múltiplo

- Para que a técnica de bloqueio múltiplo possa ser usada, uma transação T deve obedecer às seguintes regras:
  1. T deve emitir um read_lock(X) ou write_lock(X) antes que qualquer read_item(X) seja executado em T;
  2. T deve emitir um write_lock(X) antes que qualquer write_item(X) seja executado em T;
  3. T deve emitir um unlock(X) depois que todos os read_item(X) e write_item(X) tenham sido executados em T;
  4. T não emitirá nenhum read_lock(X) ou write_lock(X) se X já estiver bloqueado por T (de forma compartilhada ou exclusiva);
  5. T poderá emitir um unlock(X) apenas se tiver bloqueado X (de forma compartilhada ou exclusiva).

Bloqueio Múltiplo

- O uso da técnica de bloqueio múltiplo não garante, entretanto, que escalonamentos sejam serializáveis.
- Considere as seguintes transações:

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
X := X + Y;	Y := X + Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Bloqueio Múltiplo

T_1	T_2
read_lock(Y);	
read_item(Y);	
unlock(Y);	
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	Y := X + Y;
	write_item(Y);
	unlock(Y);
write_lock(X);	
read_item(X);	
X := X + Y;	
write_item(X);	
unlock(X);	

Os itens Y em T_1 e X em T_2 foram desbloqueados cedo demais, permitindo um escalonamento não-serializável

Bloqueio em Duas Fases

- Para garantir escalonamentos serializáveis, as operações de bloqueio e desbloqueio nas transações devem seguir protocolos.
- O protocolo mais usado é o protocolo de bloqueio em duas fases (*Two-Phase Locking*).
  - Todas as operações de bloqueio (*read_lock* e *write_lock*) precedem a primeira operação de desbloqueio (*unlock*).
  - As transações são divididas em duas fases:
    - expansão: quando são emitidos todos os bloqueios;
    - contração (encolhimento): quando os desbloqueios são emitidos e nenhum novo bloqueio pode ser emitido.

Bloqueio em Duas Fases

- Considere as seguintes transações:

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
X := X + Y;	Y := X + Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

- As transações seguem o protocolo de bloqueio em duas fases?

Bloqueio em Duas Fases

- Para seguir o protocolo, as transações foram alteradas para:

T_1'	T_2'
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y);	unlock(X);
read_item(X);	read_item(Y);
X := X + Y;	Y := X + Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

- Se todas as transações em um escalonamento seguirem o protocolo de bloqueio em duas fases, o escalonamento é garantidamente serializável.

Bloqueio em Duas Fases

- A técnica apresentada é conhecida como bloqueio em duas fases básico.
- Foram propostas algumas variações, a saber:
  - bloqueio em duas fases conservador;
  - bloqueio em duas fases estrito;
  - bloqueio em duas fases rigoroso.

Bloqueio em Duas Fases

- Bloqueio em duas fases conservador
  - Uma transação bloqueia todos os itens aos quais terá acesso, antes de iniciar o seu processamento.
    - Se algum dos itens não puder ser bloqueado, a transação não bloqueia nenhum item e espera até que todos os itens estejam disponíveis para bloqueio.
  - Evita impasse (*deadlock*), ao contrário das outras variações.
  - É difícil de ser utilizado na prática.

Bloqueio em Duas Fases

- Bloqueio em duas fases estrito
  - Uma transação não libera nenhum de seus bloqueios exclusivos (*write_lock*) até que seja confirmada (*commit*) ou abortada (*rollback*).
  - Garante escalonamentos estritos.
- Bloqueio em duas fases rigoroso
  - Uma transação não libera nenhum de seus bloqueios exclusivos (*write_lock*) ou compartilhados (*read_lock*) até que seja confirmada (*commit*) ou abortada (*rollback*).
  - Garante escalonamento estritos.
  - É mais fácil de ser implementado.

Bloqueio em Duas Fases

- O protocolo de bloqueio em duas fases garante a serialização mas não admite todos os possíveis escalonamentos serializáveis.
  - Alguns escalonamentos serializáveis não são permitidos pelo protocolo.
- Em muitos casos, as operações de bloqueio (*read_lock* e *write_lock*) são emitidas pelo próprio subsistema de controle de concorrência.
- A utilização de bloqueios pode causar dois problemas:
  - *deadlock* (impasse, bloqueio perpétuo);
  - *starvation* (inanição, espera indefinida).

Deadlock

- *Deadlock* ocorre quando transações de um escalonamento ficam esperando por algum item que esteja bloqueado por outras transações de tal escalonamento.
- Exemplo:

T_1	T_2
read_lock(Y);	
read_item(Y);	
	read_lock(X);
	read_item(X);
write_lock(X);	
	write_lock(Y);

T_1 espera pelo item X bloqueado por T_2 e T_2 espera pelo item Y bloqueado por T_1: as transações T_1 e T_2 não conseguem prosseguir e nenhuma outra transação consegue acessar os itens X e Y.

Deadlock

- Existem protocolos e técnicas para prevenir a ocorrência de *deadlocks* e outros que detectam sua ocorrência e tomam alguma ação para acabar com o impasse.
  - Um exemplo de protocolo de prevenção de *deadlock* é o protocolo de bloqueio em duas fases conservador.
- Algumas técnicas de prevenção utilizam o conceito de registro de *timestamp* da transação, que é um identificador único atribuído a cada transação do escalonamento.
  - Os registros de *timestamp* (TS) são baseados, geralmente, na ordem em que as transações são iniciadas.
    - Se T_1 inicia antes de T_2, então $TS(T_1) < TS(T_2)$.
  - A transação mais antiga tem o menor valor de TS.

Deadlock

- Duas técnicas de prevenção, que utilizam o conceito de registro de *timestamp*, são:
  - *wait-die* (esperar-morrer);
  - *wound-wait* (ferir-esperar).
- Para apresentação de tais técnicas, considere a seguinte situação:
  - Em um determinado escalonamento, a transação T_i tenta bloquear um item X mas não pode fazê-lo porque X está bloqueado pela transação T_j com um bloqueio conflitante.

Deadlock

- *Wait-die* (esperar-morrer)
  - Se $TS(T_i) < TS(T_j)$, ou seja, T_i mas antiga do que T_j, então T_i é autorizada a esperar.
  - Se $TS(T_i) > TS(T_j)$, ou seja, T_i mais nova do que T_j, então T_i é abortada e reiniciada posteriormente com o mesmo valor de registro de *timestamp*.
  - Portanto, em tal esquema, uma transação mais antiga é autorizada a esperar por uma transação mais nova; ademais, uma transação mais nova, que requeira um item bloqueado por uma transação mais antiga, é abortada e reiniciada.

Deadlock

- *Wound-wait* (ferir-esperar)
  - Se $TS(T_i) < TS(T_j)$, ou seja, T_i mas antiga do que T_j, então T_j é abortada (T_i fere T_j) e reiniciada posteriormente com o mesmo valor de registro de *timestamp*.
  - Se $TS(T_i) > TS(T_j)$, ou seja, T_i mais nova do que T_j, então T_i é autorizada a esperar.
  - Portanto, em tal esquema, uma transação mais nova é autorizada a esperar pela mais antiga; ademais, uma transação mais antiga, que requeira um item bloqueado por uma transação mais nova, apropria-se da transação mais nova, abortando a mesma.
- As duas técnicas podem fazer com que transações sejam abortadas e reiniciadas sem necessidade, já que não necessariamente geram um *deadlock*.

Deadlock

- Uma técnica de prevenção, que não utiliza o conceito de registro de *timestamp*, é o algoritmo de espera cautelosa.
  - Considerando a situação anterior, ou seja, " T_i tenta bloquear um item X que se encontra bloqueado pela transação T_j ", se T_j não estiver bloqueada (T_j não estiver esperando por algum outro item bloqueado), então T_i é bloqueada e autorizada a esperar; caso contrário, T_i é abortada.
- Geralmente, as técnicas de prevenção de *deadlock*, usando ou não o conceito de TS, não são utilizadas na prática.
  - Técnicas para detecção de *deadlocks* e *timeouts* são mais práticas.

Deadlock

- Na abordagem de detecção de *deadlock*, é verificado se um estado de *deadlock* existe de fato e, em caso positivo, algumas transações envolvidas são abortadas para que o escalonamento possa prosseguir.
- Na escolha de uma transação a ser abortada (seleção da vítima), deve-se evitar selecionar transações que venham sendo executadas por um longo período e que tenham realizado muitas atualizações.

Deadlock

- Uma forma de se detectar um estado de *deadlock* é construir e manter um grafo de espera.
  - Um nó é criado no grafo de espera para cada transação que esteja sendo executada no momento.
  - Uma aresta direcionada (nó $T_i \rightarrow$ nó T_j) é criada no grafo de espera sempre que uma transação T_i estiver esperando para bloquear um item que esteja bloqueado por uma transação T_j.
    - Quando T_j libera o bloqueio nos itens que T_i está esperando, a aresta direcionada é retirada do grafo de espera.
  - Há um *deadlock* se, e somente se, o grafo de espera tiver um ciclo.

Deadlock

- Outra forma de se detectar um estado de *deadlock* é verificar o *timeout* (intervalo, limite de tempo) definido pelo sistema.
  - Se uma transação espera por um período maior do que o período de *timeout* definido, o sistema supõe que a transação pode estar em *deadlock* e aborta a mesma, independentemente do fato do *deadlock* existir ou não.
  - Consiste em um método prático devido ao seu baixo nível de *overhead* e à sua simplicidade.

Starvation

- *Starvation* ocorre quando uma transação não pode continuar sua execução em um intervalo indefinido de tempo, enquanto outras transações são executadas normalmente.
- Na prática, *starvation* pode ocorrer quando:
  - o esquema de espera para itens bloqueados for injusto, priorizando algumas transações em relação a outras;
    - Solução 1: quanto mais uma transação espera, aumentar a sua prioridade de execução.
    - Solução 2: utilizar um esquema de fila de tal forma que a primeira transação a chegar será a primeira a ser atendida.
  - a mesma transação, no processo de seleção de vítima, for escolhida como vítima repetidamente.
    - Solução: aumentar a prioridade de execução de transações que tenham sido abortadas inúmeras vezes.

Registro de *Timestamp*

- Um registro de *timestamp* (TS) é um identificador único criado pelo SGBD para identificar uma transação.
  - Os registros são gerados por um contador incremental ou utilizam o valor corrente do relógio do sistema (data/hora/minuto/segundo/milissegundo).
- Técnicas de controle de concorrência baseada em registro de *timestamp* utilizam registros de *timestamp* para ordenar a execução de transações de tal forma a produzir um escalonamento serializável.
  - Bloqueios não são utilizados; logo, *deadlocks* não ocorrem.
  - O escalonamento produzido é equivalente à ordem serial específica correspondente à ordem dos registros de *timestamp* das transações.

Registro de *Timestamp*

- A cada item X do banco de dados são associados dois valores de registro de *timestamp*:
  - **read_TS(X)**: é o maior dos registros de *timestamp* relativos às transações que tenham lido com sucesso o item X.
    - read_TS(X) = TS(T), onde T é a transação mais recente que leu X.
  - **write_TS(X)**: é o maior dos registros de *timestamp* relativos às transações que tenham escrito com sucesso o item X.
    - write_TS(X) = TS(T), onde T é a transação mais recente que gravou X.
- Existem duas técnicas para controle de concorrência baseada em registro de *timestamp*, a saber:
  - algoritmo de ordenamento básico de registro de *timestamp*;
  - algoritmo de ordenamento estrito de registro de *timestamp*.

Ordenamento Básico de Registro de *Timestamp*

- Quando a transação T emite uma operação write_item(X):
  - Se read_TS(X) > TS(T) ou write_TS(X) > TS(T), então a operação write_item(X) é rejeitada e T é abortada, sendo submetida novamente ao sistema como uma nova transação com um novo registro de timestamp.
    - Neste caso, alguma transação mais recente que T já leu ou gravou o valor do item X.
  - Caso contrário, a operação write_item(X) de T é executada e write_TS(X) é atualizado para TS(T).
    - Neste caso, T corresponde à transação mais recente a escrever o valor do item X.

Ordenamento Básico de Registro de *Timestamp*

- Quando a transação T emite uma operação read_item(X):
  - Se write_TS(X) > TS(T), então a operação read_item(X) é rejeitada e T é abortada, sendo submetida novamente ao sistema como uma nova transação com um novo registro de timestamp.
    - Neste caso, alguma transação mais recente que T já gravou o valor do item X.
  - Caso contrário, a operação read_item(X) de T é executada e read_TS(X) é atualizado para o maior valor entre TS(T) e o valor corrente de read_TS(X).
    - Neste caso, T corresponde à transação mais recente a ler o valor do item X.

Ordenamento Estrito de Registro de *Timestamp*

- O algoritmo básico pode provocar *rollback* em cascata. Com isso, gerou-se o algoritmo de ordenamento estrito:
  - Variação do algoritmo básico com a regra adicional: se uma transação T emitir um read_item(X) ou write_item(X) e TS(T) > write_TS(X), então sua operação de leitura ou escrita deve ser retardada até que a transação T', que gravou o valor de X por último, tenha sido confirmada ou abortada.
  - Para tal retardamento, é necessário simular o bloqueio do item X que foi escrito pela transação T' até que T' tenha sido confirmada ou abortada.
    - Não causa *deadlock*: T espera por T' somente se TS(T)>TS(T').
- Com registros de *timestamp*, pode ocorrer *starvation* se uma transação for continuamente abortada e reiniciada.

Técnicas Multiversão

- As técnicas de controle de concorrência multiversão armazenam os valores antigos de um item de dado quando o mesmo é atualizado.
  - Quando uma transação escreve um item, uma nova versão do item é escrita e a versão antiga do item é retida.
  - Algumas operações de leitura, que seriam rejeitadas em outras técnicas, passam a ser aceitas por meio da leitura de uma versão anterior do item para manter a serialização.
  - Desvantagem: necessidade de um volume maior de armazenamento para manter as versões dos itens do banco.
    - As técnicas multiversão são ideais para banco de dados temporal.

Técnicas Multiversão

- Alguns exemplos de técnicas de controle de concorrência multiversão são:
  - técnica multiversão baseada no ordenamento de registro de *timestamp*;
  - bloqueio em duas fases multiversão utilizando bloqueios de certificação.

Ordenamento de Registro de *Timestamp* Multiversão

- Nesta técnica, diversas versões X_1, X_2, ..., X_k de cada item de dados X são armazenados. Para cada versão, o valor de X_i e os seguintes registros de *timestamp* são mantidos:
  - **read_TS(X_i)**: é o maior dos registros de *timestamp* relativos às transações que tenham lido com sucesso a versão X_i.
    - read_TS(X_i) = TS(T), onde T é a transação mais recente que leu X_i.
  - **write_TS(X_i)**: é o registro de *timestamp* da transação que escreveu com sucesso o valor da versão X_i.
    - write_TS(X_i) = TS(T), onde T é a transação que escreveu o valor da versão X_i.

Ordenamento de Registro de *Timestamp* Multiversão

- Se uma transação T pode emitir uma operação write_item(X), então uma nova versão X_{k+1} do item X será criada e os registros de *timestamp* write_TS(X_{k+1}) e read_TS(X_{k+1}) serão atualizados para TS(T).
- Se uma transação T pode emitir uma operação de leitura do valor da versão X_i, então o registro de *timestamp* read_TS(X_i) será atualizado para o maior valor entre read_TS(X_i) e TS(T) correntes.

Ordenamento de Registro de *Timestamp* Multiversão

- Para garantir a serialização, as seguintes regras são usadas:
  - Se a transação T emite uma operação write_item(X) e X_i possui o mais alto write_TS(X_i) das versões de X (que também é menor ou igual a TS(T)) e read_TS(X_i) > TS(T), então T é abortada e reiniciada; caso contrário, uma nova versão X_j do item X é criada e os registros de *timestamp* write_TS(X_i) e read_TS(X_i) são atualizados para TS(T).
  - Se a transação T emite uma operação read_item(X), deve ser encontrado X_i que possui o mais alto write_TS(X_i) das versões de X (que também é menor ou igual a TS(T)); a partir daí, o valor de X_i é retornado para a transação T e o registro de *timestamp* read_TS(X_i) é atualizado para o maior valor entre read_TS(X_i) e TS(T) correntes.

Bloqueio em Duas Fases Multiversão

- Nesta técnica, há três modos de bloqueio para um item: ler, escrever e certificar, ao invés de apenas ler e escrever.
- Logo, o estado de bloqueio (lock(X)) para X pode ser:
  - *read_locked* (leitura bloqueada);
  - *write_locked* (escrita bloqueada);
  - *certify_locked* (certificação bloqueada: exclusivo);
  - *unlocked* (desbloqueado).
- Tabela de compatibilidade de bloqueio:

Transação T' que solicita o tipo de bloqueio

Transação T que controla o tipo de bloqueio

	Ler	**Escrever**	**Certificar**
Ler	sim	sim	não
Escrever	sim	não	não
Certificar	não	não	não

Bloqueio em Duas Fases Multiversão

- A idéia é permitir que transações T' leiam um item X enquanto uma transação T mantém o bloqueio de escrita em X.
  - Para tanto, deve-se manter duas versões para cada item X: uma versão deve sempre ser escrita por alguma transação confirmada; a outra versão (X') é criada quando uma transação T adquire um bloqueio de escrita no item.
  - Assim, outras transações podem continuar a ler a versão confirmada de X enquanto T mantém o bloqueio de escrita.
  - A transação T pode escrever o valor de X', de acordo com a necessidade, sem afetar o valor da versão confirmada de X.

Bloqueio em Duas Fases Multiversão

- Quando T estiver pronta para ser confirmada, ela deve obter um bloqueio de certificação (bloqueio exclusivo) em todos os itens nos quais ela mantenha bloqueios de escrita.
  - Já que o bloqueio de certificação não é compatível com os de leitura, a transação pode ter que retardar sua confirmação até que todos os seus itens bloqueados para escrita sejam liberados por quaisquer transações.
  - Ao adquirir o bloqueio de certificação de X, a versão confirmada de X é ajustada para o valor da versão X', a versão X' é descartada e os bloqueios de certificação são então liberados.

Bloqueio em Duas Fases Multiversão

- Em tal técnica, as leituras podem continuar concorrentemente com uma única operação de escrita.
- Custo: uma transação pode ter que atrasar até obter bloqueios de certificação exclusivos de todos os seus itens atualizados.
- Esse esquema evita *rollback* em cascata, já que as transações somente leêm a versão confirmada de X.
  - Entretanto, podem ocorrer *deadlocks* se a promoção de um bloqueio de leitura para um bloqueio de escrita for permitido (variações da técnica).

Técnicas de Validação (Otimistas)

- Em técnicas de controle de concorrência de validação (ou técnicas otimistas), nenhuma verificação é realizada enquanto a transação está sendo executada, diminuindo um custo adicional durante a execução da transação.
- Geralmente, as técnicas otimistas de controle de concorrência funcionam bem se houver pouca interferência (suposição otimista) entre as transações de um escalonamento.

Técnicas de Validação (Otimistas)

- Uma das técnicas otimistas propõe as seguintes regras:
  - Atualizações na transação não são aplicadas diretamente aos itens do banco de dados até que a transação atinja seu final.
    - Todas as atualizações são aplicadas a cópias locais dos itens de dados.
  - Ao final de uma transação, a fase de validação verifica se qualquer uma das atualizações da transação viola a serialização.
    - Se a serialização não for violada, a transação é confirmada e o banco de dados é atualizado a partir das cópias locais; caso contrário, a transação é abortada e posteriormente reiniciada.

Granularidade de Itens de Dados

- Os mecanismos de controle de concorrência assumem que um banco de dados é formado por uma coleção de itens de dados, sendo que um item de dados pode ser:
  - um registro (tupla) do banco de dados;
  - um valor de campo (atributo) de um registro do banco;
  - um bloco de disco;
  - um arquivo inteiro;
  - um banco de dados inteiro.
- A granularidade de um item de dados, ou seja, o tamanho do item de dados, pode afetar o desempenho do controle de concorrência e da recuperação de falhas.

Nível de Granularidade para o Bloqueio

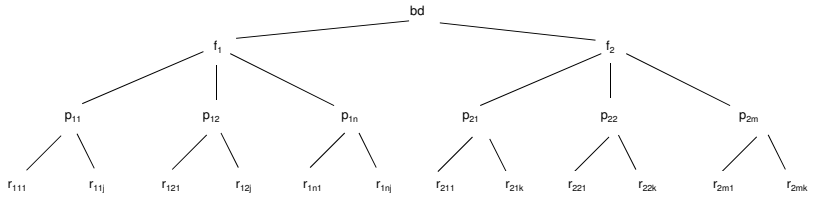
- Quanto maior o tamanho do item de dados, menor é o grau de concorrência permitido.
  - Se o tamanho do item de dados for um bloco de disco, uma transação T, que precisa bloquear um registro A, deve bloquear o bloco de disco X inteiro que contém A. Se outra transação S necessita bloquear o registro B que esteja no mesmo bloco de disco X, ela é forçada a esperar.
  - Se o tamanho do item de dados fosse um único registro, S poderia prosseguir pois estaria bloqueando um registro diferente.
- Quanto menor o tamanho do item de dados, maior é o número de itens no banco de dado a serem gerenciados.
  - Já que cada item está associado a um bloqueio, o sistema terá um grande número de bloqueios ativos a serem tratados pelo gerenciador de bloqueios, gerando um *overhead* maior e necessitando um espaço de armazenamento maior para a tabela de bloqueios.

Nível de Granularidade para o Bloqueio

- O tamanho do item de dados depende dos tipos de transações envolvidas no sistema.
  - Se uma transação típica acessa um pequeno número de registros, a granularidade de item de dados como um registro é vantajosa.
  - Se uma transação normalmente acessa muitos registros de um mesmo arquivo, a granularidade de bloco de disco ou arquivo, de modo que a transação considere todos esses registros como um ou poucos itens de dados, pode ser melhor.
- Um sistema de banco de dados deve suportar diversos e diferentes níveis de granularidade para atender às várias e possíveis combinações de transações.

Bloqueio com Nível de Granularidade Múltiplo

- A figura abaixo mostra uma hierarquia de granularidade simples em um banco de dados contendo dois arquivos: cada arquivo contém várias páginas de disco e cada página contém vários registros.



- Somente bloqueios compartilhados e exclusivos são ineficientes para vários níveis de granularidade.

Bloqueio com Nível de Granularidade Múltiplo

- Considere bloqueio compartilhado e exclusivo em diferentes níveis de granularidade para atender o banco apresentado.
  - Suponha que T_1 queira atualizar todos os registros do arquivo f_1 recebendo, no caso, um bloqueio exclusivo para f_1. Isso é bom já que um bloqueio único em nível de arquivo é mais eficiente do que vários bloqueios em nível de página ou de registro.
  - Suponha agora que T_2 queira acessar o registro r_{1nj} da página p_{1n} de f_1, solicitando um bloqueio compartilhado em nível de registro em r_{1nj}. Neste caso, o gerenciador de bloqueios deve verificar a compatibilidade desse bloqueio com os mantidos, atravessando a árvore de r_{1nj} para p_{1n} para f_1 para bd. Se houver bloqueio em conflito para algum item, o bloqueio para r_{1nj} é negada e T_2 deve esperar. A travessia feita é muito eficiente.
  - E se a solicitação de T_2 vier primeiro? O bloqueio compartilhado de registro é concedido para r_{1mj} e, quando o bloqueio de arquivo para f_1 for solicitado por T_1, o gerenciador deve verificar todos os nós (páginas e registros) descendentes de f_1 na busca de um bloqueio de conflito. Isso é ineficiente e frustra múltiplos bloqueios em nível de granularidade.

Bloqueio com Nível de Granularidade Múltiplo

- Em um protocolo de bloqueio em duas fases com nível de granularidade múltiplo, um bloqueio pode ser solicitado em qualquer nível.
  - Para tanto, são necessários tipos adicionais de bloqueios, denominados bloqueios de intenção.
  - Em bloqueios de intenção, a transação deve indicar, ao longo do caminho da raiz até o nó desejado, qual o tipo de bloqueio (compartilhado ou exclusivo) que ela requisitará de um dos descendentes do nó.

Bloqueio com Nível de Granularidade Múltiplo

- Além dos bloqueios compartilhado (S) e exclusivo (X), o protocolo utiliza os seguintes bloqueios:
  - compartilhado por intenção (IS): indica que um ou mais bloqueios compartilhados serão solicitados em algum ou alguns nós descendentes;
  - exclusivo por intenção (IX) : indica que um ou mais bloqueios exclusivos serão solicitados em algum ou alguns nós descendentes;
  - compartilhado-exclusivo por intenção (SIX): indica que o nó atual está bloqueado no modo compartilhado mas que um ou mais blocos exclusivos serão solicitados em algum ou alguns nós descendentes.

Bloqueio com Nível de Granularidade Múltiplo

- A matriz de compatibilidade de bloqueios é:

IS	**IX**	**S**	**SIX**	**X**	
IS	sim	sim	sim	sim	não
IX	sim	sim	não	não	não
S	sim	não	sim	não	não
SIX	sim	não	não	não	não
X	não	não	não	não	não

Bloqueio com Nível de Granularidade Múltiplo

- O protocolo consiste nas seguintes regras:
  - a compatibilidade de bloqueios deve ser respeitada;
  - a raiz da árvore deve ser bloqueada primeiro, em qq modo;
  - um nodo N pode ser bloqueado por uma transação T no modo S ou IS somente se o nodo pai de N já estiver bloqueado por T no modo IS ou IX;
  - um nodo N pode ser bloqueado por uma transação T no modo X, IX ou SIX somente se o nodo pai de N já estiver bloqueado por T no modo IX ou SIX;
  - uma transação T pode bloquear um nodo somente se não tiver desbloqueado nenhum nodo (bloqueio em duas fases);
  - uma transação T pode desbloquear um nodo N somente se nenhum dos filhos de N estiver atualmente bloqueado por T.

Bloqueio com Nível de Granularidade Múltiplo

- Para ilustrar o protocolo, considere as seguintes transações:
  - T_1 deseja atualizar os registros r_{111} e r_{211}.
  - T_2 deseja atualizar todos os registros na página p_{12}.
  - T_3 deseja ler o registro r_{11j} e o arquivo f_2 inteiro.

Obs: **(***)** corresponde às operações "unlock(x)" restantes.

T_1	T_2	T_3
IX (db)		
IX (f_1)		
	IX (db)	
		IS (db)
		IS (f_1)
		IS (p_{11})
IX (p_{11})		
X (r_{111})		
	IX (f_1)	
	X (p_{12})	
		S (r_{11j})
IX (f_2)		
IX (p_{21})		
X (r_{211})		
unlock (r_{211})		
unlock (p_{21})		
unlock (f_2)		
		S (f_2)
	(***)	
(***)		
		(***)

Bloqueios para Controle de Concorrência em Índices

- O bloqueio em duas fases pode também ser aplicado para controle de concorrência em índices.
  - A abordagem tradicional degrada a eficiência, pois a pesquisa de um índice sempre começa pela raiz da árvore de índice.
    - Por exemplo, se uma transação T deseja inserir um registro (operação de escrita), a raiz será bloqueada em modo exclusivo e, assim, outros bloqueios conflitantes solicitados para o índice devem esperar até que T entre em sua fase de contração.
    - Logo, ocorre a interrupção de todas as outras transações que desejam acessar o índice.

Bloqueios para Controle de Concorrência em Índices

- Aproveitando a estrutura de árvore do índice, pode-se desenvolver outro esquema de controle de concorrência:
  - Quando uma pesquisa no índice (leitura) estiver sendo executada, um caminho na árvore é percorrido da raiz à folha; assim, uma vez que um nó já tenha sido acessado, os nós de nível acima não seriam acessados novamente. Logo, para bloqueios de leitura, uma vez que o bloqueio no nó filho seja obtido, o bloqueio no pai pode ser liberado.
  - Quando uma inserção (gravação) estiver sendo aplicada a um nó folha, este deve ser bloqueado em modo exclusivo; entretanto, se esse nó não estiver cheio, a inserção não causará alterações em nós de índice de nível acima na árvore, implicando que eles não necessitam estar bloqueados exclusivamente.

Bloqueios para Controle de Concorrência em Índices

- Uma abordagem conservadora para inserções:
  - Bloquear o nó raiz em modo exclusivo e, então, acessar o nó filho apropriado da raiz.
  - Se o nó filho não estiver cheio, o bloqueio no nó raiz pode ser liberado.
  - Essa abordagem pode ser aplicada em todo o caminho da árvore até a folha.
- Uma abordagem alternativa mais otimista para inserções:
  - Requisitar e manter bloqueios compartilhados nos nós que levam ao nó folha, com um bloqueio exclusivo na folha.
  - Se a inserção causar divisão na folha, a mesma se propagaria para nós de nível acima; nesse caso, os bloqueios em tais nós podem ser alterados para o modo exclusivo.

Registros Fantasmas

- O problema de registro fantasma (*phanton*) ocorre quando um novo registro que esteja sendo inserido pela transação T satisfaz a mesma condição que um conjunto de registros acessados por outra transação T'.
  - Por exemplo, T está inserindo um registro de EMPREGADO cujo NumDepto = 5, enquanto T' está acessando todos os registros de EMPREGADO cujo NumDepto = 5.
    - Se a ordem serial equivalente for T seguida de T', então T' lerá o novo registro de EMPREGADO. Para a ordem serial T' seguida de T, o novo registro não será considerado.
  - As transações conflitam logicamente e um registro fantasma pode aparecer repentinamente no banco de dados.
    - Se outras operações nas duas transações conflitarem, o conflito causado pelo registro fantasma pode não ser reconhecido pelo protocolo de controle de concorrência.

Registros Fantasmas

- Uma solução para detectar um registro fantasma é usar o bloqueio de índice.
  - Se a entrada de índice for bloqueada antes que o registro possa ser acessado, então o conflito do registro fantasma pode ser detectado.
    - A transação T' solicitaria um bloqueio de leitura na entrada de índice para NumDepto = 5, e T solicitaria um bloqueio de escrita na mesma entrada antes que se coloque o bloqueio propriamente nos registros.
    - Assim, uma vez que o bloqueio de índice gera conflito, o conflito do fantasma seria detectado.

Travas

- Travas (*latches*) são bloqueios de curta duração.
- As travas não seguem os protocolos usuais de controle de concorrência como o bloqueio em duas fases.
- Por exemplo, uma trava pode ser utilizada para garantir a integridade física de uma página quando a mesma estiver sendo gravada do *buffer* para o disco.
  - Uma trava seria fornecida para a página, a página seria gravada para o disco e, em seguida, a trava seria liberada.