Gerência de Transações

Transações

- Conceito básico para controle de concorrência e recuperação: a transação.
  - Uma seqüência de ações que são consideradas uma unidade atômica (indivisível) de trabalho.
- "Acões elementares" do SGBD:
  - Leituras (*reads*) e escritas (*writes*)
  - Ações especiais: *commit* (compromissamento ou efetivação de transação), *abort* (aborto de transação)
- O SGBD 'vê' cada transação como uma seqüência de leituras e escritas delimitada por comandos *begin* e *commit* (ou *abort*).

As Propriedades *"ACID"*

- **A**tomicidade: Ou todas as ações da transação acontecem, ou nenhuma delas acontece.
- **C**onsistência: Se a transação é consistente e o BD começa consistente, ele termina consistente.
- **I**solação: A execução de uma transação é isolada da execução de outras transações.
- **D**urabilidade: Se uma transação é concluída com sucesso (através de uma operação *commit* bem sucedida), então seus efeitos são persistentes (duráveis).

Satisfazendo as Propriedades *"ACID"*

- Controle de Concorrência
  - Garante a **C**onsistência e a **I**solação, dada a atomicidade das transações.
    - Num SGBD: É tarefa do Módulo Gerente de Bloqueios (*Locks*).
- *Logging* e Recuperação
  - Garantem a **A**tomicidade e a **D**urabilidade.
    - Num SGBD: São tarefas do Módulo Gerente de *Log*.

Concorrência num SGBD

- A execução concorrente de programas dos usuários é essencial para o bom desempenho do SGBD.
  - Como acessos a disco são freqüentes e relativamente lentos, é muito importante manter a CPU ocupada executando vários programas concorrentemente.
- Uma aplicação pode efetuar muitas operações sobre os dados lidos de um BD, mas para o SGBD só importam as leituras e as escritas realizadas.

Concorrência num SGBD (cont.)

- Usuários submetem transações e podem pensar que cada transação roda sozinha, como "dona" da máquina.
  - A concorrência é implementada pelo SGBD, que entrelaça ações (*reads*/*writes* de objetos no BD) das várias transações.
  - Cada transação deve deixar o BD num estado consistente.
    - O SGBD impõe as restrições de integridade especificadas nos comandos CREATE TABLE, mas não 'entende' realmente a semântica dos dados. (Por exemplo: Ele não sabe como computar os juros numa conta de poupança.)
- Questões: Efeitos do entrelaçamento de transações e de quedas do sistema.

Exemplo

- Considere duas transações:

```
T1:   BEGIN   A = A + 100,  B = B - 100    END
T2:   BEGIN   A = 1.06 * A,  B = 1.06 * B   END
```

- Intuitivamente, a primeira transação está transferindo $100 da conta B para a conta A. A segunda está creditando 6% de juros em ambas as contas.
- Não há garantia que T1 vai executar antes de T2 ou vice-versa, se ambas forem submetidas praticamente juntas. Contudo, o efeito visível tem de ser equivalente ao dessas duas transações rodando serialmente (uma depois da outra), numa ordem qualquer.

Exemplo (cont.)

- Considere o seguinte entrelaçamento (escalonamento):

```
T1:   A=A+100,              B=B-100
T2:             A=1.06*A,             B=1.06*B
```

- Tudo bem com o escalonamento acima. Vejamos outro:

```
T1:   A=A+100,                        B=B-100
T2:             A=1.06*A,  B=1.06*B
```

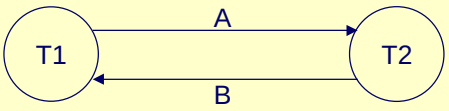
- Como o SGBD vê o segundo escalonamento:

```
T1:   R(A),W(A),                              R(B),W(B)
T2:               R(A),W(A), R(B),W(B)
```

Exemplo (cont.)

- O SGBD não pode permitir escalonamentos como este!

T1:	R(A),W(A),	R(B),W(B)
T2:	R(A),W(A), R(B),W(B)	



Grafo de dependências

- Grafo de dependências: Um nó por transação, flecha de *Ti* para *Tj* se *Tj* ler ou escrever um objeto escrito pela última vez por *Ti*.
- O ciclo no grafo revela o problema. O resultado de T1 depende de T2 e vice-versa.

Escalonando Transações

- Escalonamentos equivalentes: Para qualquer estado do BD, o efeito (no conjunto de objetos no BD) de executar um escalonamento é idêntico ao efeito de executar o outro escalonamento.
- Escalonamento serializável: Um escalonamento que é equivalente a uma execução serial das transações.
  - Se o grafo de dependências de um escalonamento for acíclico (não tiver ciclos), dizemos que o escalonamento é serializável quanto ao conflito *(conflict-serializable*). Tal escalonamento é equivalente a um escalonamento serial.
  - Esta é a condição que é tipicamente assegurada pelos SGBDs. Ela é suficiente (mas não necessária) para que o escalonamento seja serializável.

Assegurando a Seriabilidade

- Protocolo do Travamento Bifásico (*Two-phase Locking*, ou *2PL*):
  - Baseado em bloqueios ou `travas´ (*locks*).
  - Cada transação, antes de ler um objeto, precisa obter um bloqueio S (**s***hared*, ou compartilhado) sobre o objeto. Antes de escrever num objeto, a transação precisa obter um bloqueio X (e**x**clusivo) sobre o objeto.
  - Depois que uma transação retirar algum bloqueio ela não poderá requisitar novos bloqueios.
    - Variação: o protocolo 2PL estrito, no qual cada transação retém seus bloqueios até ser efetivada (commit) ou abortada.
  - Enquanto uma transação detiver algum bloqueio X sobre um objeto, nenhuma outra transação conseguirá um bloqueio (S ou X) sobre o objeto.

Assegurando a Seriabilidade (cont.)

- O protocolo 2PL só permite escalonamentos serializáveis quanto ao conflito.
- Problema potencial de *deadlocks*: pode haver um ciclo de transações, *T1*, *T2*, ..., *Tn*, tal que cada *Ti* fica esperando que sua predecessora retire um bloqueio sobre algum objeto.
  - SGBDs lidam com este problema fazendo detecção de *deadlocks*. Uma das transações envolvidas no *deadlock* é abortada pelo SGBD, que libera os bloqueios concedidos a essa transação (para que as demais possam prosseguir).
- Várias granularidades de bloqueio são possíveis.

Atomicidade das Transações

- Uma transação pode dar um *commit* depois de completar todo o seu trabalho, ou pode dar um *abort* (ou ser abortada pelo SGBD) depois de executar algumas ações.
- Uma propriedade muito importante garantida pelo SGBD é que toda as transações são atômicas. O usuário pode pensar que uma transação ou executa todas suas ações "de uma vez só", ou não executa ação nenhuma.
  - O SGBD faz um *log* de todas as ações, para poder desfazer (*undo*) as ações das transações abortadas.
- Isto garante que se cada transação preserva a consistência do BD, então todo escalonamento serializável também preserva a consistência do BD.

Abortando Uma Transação

- Se uma transação *Ti* for abortada, todas as suas ações deverão ser desfeitas. Além disso, se *Tj* tiver lido um objeto escrito por *Ti*, então *Tj* também precisará ser abortada!
- A maioria dos sistemas evita esses "abortos em cascata" retendo os bloqueios de uma transação até que a transação dê *commit* (usando o protocolo 2PL estrito).
  - Se *Ti* escrever num objeto, então *Tj* só vai poder ler o valor escrito depos que *Ti* der *commit*.
- Para desfazer as ações de uma transação abortada, o SGBD mantém um *log* no qual cada escrita é registrada. Este mecanismo é também usado para recuperação de *crashes*: quando o sistema volta, são abortadas todas as transações que estavam ativas no momento da queda.

O "*System Log*"

- As seguintes ações são registradas no *log*:
  - Ti escreve num objeto: são registrados o valor antigo (imagem anterior) e o valor novo (imagem posterior).
    - O registro do log precisa ir para disco antes da modificação no objeto (esta técnica se chama *write-ahead-logging*, ou *WAL*).
  - Ti dá um *commit* ou um *abort*: um registro de log indicando esta ação.
- Registros de *log* são encadeados através de um identificador de transação, de modo que seja fácil desfazer uma transação especificada.
- O log é mantido em disco(s) próprio(s). Para se ter armazenamento estável (*stable storage*) usa-se a técnica de espelhamento de blocos (dois blocos físicos para cada bloco lógico).
- Todas as ações de *log* (e, de fato, todas as ações de controle de concorrência, tais como bloquear/desbloquar dados, lidar com deadlocks, etc.) são efetuadas transparentemente pelo SGBD.

Algoritmo de Recuperação de *Crashes*

- Três fases:
  - Análise: Varre o log para frente (desde o último *checkpoint*) para identificar todas as escritas que pudessem estar pendentes e todas as transações que estavam ativas quando o sistema caiu.
  - *Redo*: Refaz todas as escritas que podem estar pendentes, de modo a assegurar que todos os *writes* registrados no *log* foram de fato efetuados em disco.
    - Usa as imagens posteriores nos registros de *log* dessas escritas.
  - *Undo*: Varre o log para trás, desfazendo as escritas de todas as transações que estavam ativas quando o sistema caiu.
    - Usa as imagens anteriores nos registros de *log* dessas escritas.
- Esta é só uma visão geral do algoritmo de recuperação!
  - Estamos omitindo muitos detalhes. Por exemplo: Como lidar com o caso de um novo *crash* durante a recuperação de um *crash*?

Falhas no Ambiente de BD

- Falha da plataforma cliente, ou da própria transação, ou do meio de comunicação.
  - O SGBD usa o *log* para desfazer a(s) transação(ões).
- Falha da plataforma usada pelo SGBD ou do próprio SGBD.
  - O SGBD usa o *log* para refazer as transações que já deram *commit* mas ainda tem escritas pendentes.
- Falha do meio de armazenamento usado para os dados.

  O DBA precisa tomar as seguintes ações:
  - Refazer o banco de dados a partir da última cópia de segurança.
  - Usando o *log*, refazer todas as transações depois que a cópia de segurança foi gerada. (Executar utilitário específico do SGBD.)

Conclusões

- Dentre os serviços um SGBD, controle de concorrência e recuperação de *crashes* estão entre os mais importantes.
- Usuários (quase) não precisam se preocupar com concorrência.
  - O sistema automaticamente insere comandos *lock*/*unlock* e escalona as ações das várias transações de modo a garantir que o resultado é igual ao de se executar uma transação depois da outra, em alguma ordem.
- A técnica de *write-ahead logging* (WAL) é usada para desfazer as ações das transações abortadas e trazer o sistema para um estado consistente, depois de uma queda.
  - Estado consistente: Só aparecem os efeitos das transações efetivadas (*commited*).