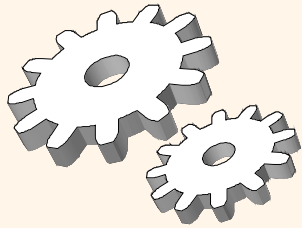
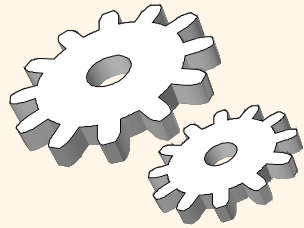


Gerenciamento de Transações

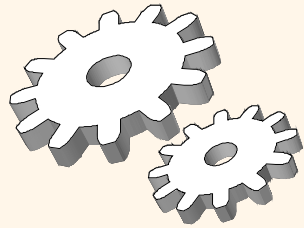
Capítulo 16

Transações



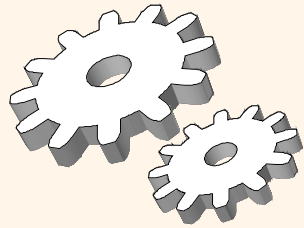
- Uma *transação* é uma visão abstrata que o SGBD tem de um programa de usuário: uma seqüência de leituras (*reads*) e escritas (*writes*).
- A execução concorrente de programas usuário é essencial para o bom desempenho de um SGBD.
  - Como os acessos a disco são freqüentes, e relativamente lentos, é importante manter a CPU trabalhando em vários programas usuários concorrentemente.
- Um programa de usuário pode executar várias operações sobre os dados recuperados da base de dados, mas o SGBD está preocupado apenas sobre quais dados são lidos/escritos da/para a base de dados.

Concorrência em um SGBD



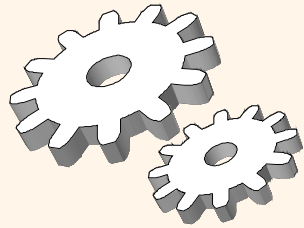
- Usuários submetem transações e podem pensar que cada transação é executada sozinha.
  - A concorrência é aplicada pelo SGBD, que intercala ações (leituras e escritas de objetos do BD) de várias transações.
  - Cada transação deve deixar a base de dados em um estado consistente se o BD está consistente quando a transação é iniciada.
    - O SGBD garantirá algumas restrições de integridade, dependendo das restrições de integridade declaradas nos comandos CREATE TABLE.
    - Além disso, o SGBD não compreende a semântica dos dados (i.e, ele não compreende como o saldo de uma conta bancária é calculado).
- *Problemas:*
  - Efeito de *intercalar* transações;
  - *Quedas no sistema*.

Propriedades ACID de Transações



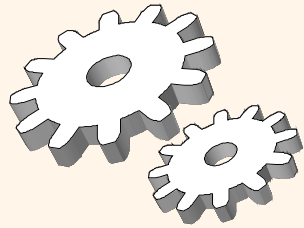
- **Atomicidade:** A execução de uma transação deve ser atômica, ou todas as ações são executadas, ou nenhuma é;
- **Consistência:** Cada transação executada isoladamente deve preservar a consistência do banco de dados;
- **Isolamento:** Cada transação deve ser isolada dos efeitos da execução concorrente de outras transações;
- **Durabilidade:** Toda transação que for finalizada de forma bem-sucedida deve persistir seus resultados em banco mesmo na presença de falhas no sistema.

Consistência e Isolamento



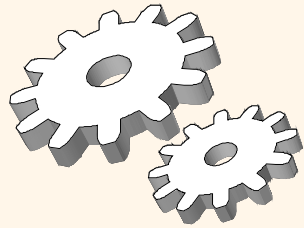
- É o usuário quem deve garantir a consistência da transação.
  - Exemplo: Transferência de fundos entre contas não alteram a quantia total de dinheiro nas contas.
- O isolamento deve garantir que duas transações, executadas concorrentemente, devem ter o mesmo resultado se executadas em ordem serial.
  - Exemplo: T1 concorrente com T2 ⇒T1, T2 ou T2, T1.

Atomicidade de Transações



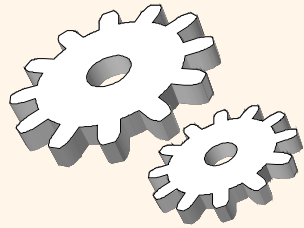
- As falhas em uma transação podem ter como motivo:
  - Interrupção do SGBD;
  - Queda do sistema;
  - Detecção de uma situação inesperada.
- Uma transação interrompida ao meio pode deixar o banco de dados em um estado ***inconsistente***.
- O banco de dados deve prover recursos para remoção dos efeitos de transações incompletas para garantir a **atomicidade**.

Atomicidade de Transações



- O SGBD mantém um registro (***log***) das ações executadas pelo usuário para que estas possam ser desfeitas caso ocorra alguma falha em uma transação.
- O *log* também é utilizado para garantir a ***durabilidade***. Se ocorrer queda do sistema antes que todas as mudanças tenham sido feitas em disco, o *log* é usado para restaurar o estado do banco de dados quando o sistema for reiniciado.

Escalonamento - Exemplo

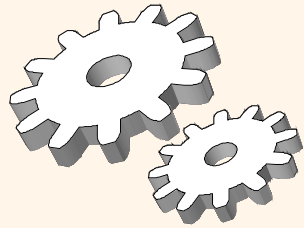


- Considere duas transações:

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

- Intuitivamente, a primeira transação transfere $100 da conta B para a conta A. A segunda acrescenta 6% nas duas contas.
- Se as duas são submetidas juntas, não há garantias de que a transação T1 executará antes da transação T2, ou vice-versa. Entretanto, o efeito final *deve* ser equivalente ao efeito da execução serial, em alguma ordem, das duas transações.
- Um *escalonamento* é uma lista de ações de um conjunto de transações. Representa uma seqüência de execução que deve conservar a mesma ordem de execução das ações das transações presentes nele.

Exemplo (Cont.)



- Considere uma possível intercalação (*escalonamento*):

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

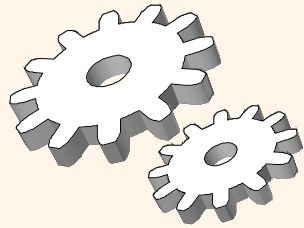
- O resultado, para o exemplo acima, está correto. Considere o próximo caso:

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

- A visão do SGBD para o segundo escalonamento:

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

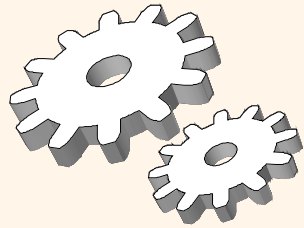
Escalonamento de Transações



- *Escalonamento completo:* O escalonamento insere, para todas as transações, as ações de *abort* ou *commit*.
- *Escalonamento serial:* O escalonamento não intercala as ações de transações diferentes.
- *Escalonamento equivalente*: Para qualquer estado da base de dados, o efeito de executar um primeiro escalonamento é idêntico ao efeito de executar um segundo escalonamento.
- *Escalonamento serializável*: Escalonamento equivalente a alguma execução serial das transações.

(Nota: Se cada transação preserva a consistência, todo escalonamento serializável preserva a consistência)

Defeitos atribuídos a Execuções Intercaladas



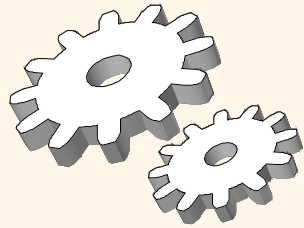
- Duas ações no mesmo objeto conflitam se pelo menos uma delas é uma escrita (W).
- Três defeitos podem ser descritos:
- Leitura de dados que não foram salvos (Conflitos de WR, "leitura suja"):

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

- Leitura não repetível (Conflitos RW):

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

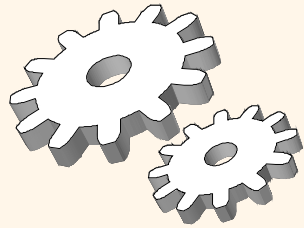
Defeitos (Cont.)



- Sobreposição de dados que não foram salvos (Conflitos WW):

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

Escalonamentos que Envolvem Transações Falhas

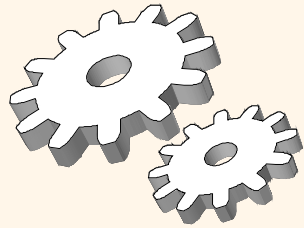


- O escalonamento abaixo é irrecuperável:

T1: R(A), W(A-100),		Abort
T2:	R(A), W(A+6%), R(B), W(B+6%), C	

- Se T2 não tivesse sido completada, para recuperar a consistência, o aborto deveria ser feito em cascata.
- Em um escalonamento recuperável, transações são completadas apenas depois que todas as transações que foram utilizadas por elas, completaram-se.

Escalonamentos que Envolvem Transações Falhas (Cont.)

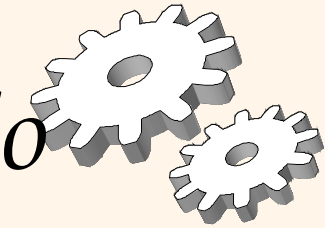


- Outro problema potencial é referente ao tópico sobre desfazer as ações de transações:

T1: R(A), W(A+1),		Abort
T2:	R(A), W(A+1),	C

- Todas as alterações feitas por T1 são desfeitas restaurando os valores dos objetos participantes.
  - as alterações de T2 também são perdidas, mesmo se T2 é completada.
- Problemas desse gênero podem ser resolvidos através de técnicas de controle de concorrência.

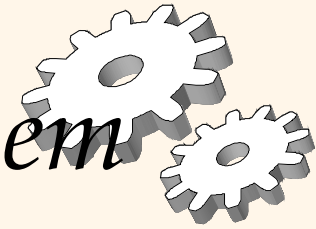
Controle de Concorrência Baseado em Bloqueio



- *Protocolo de bloqueio estrito de duas fases (Strict 2PL)*:
  - Cada transação deve obter:
    - bloqueio *compartilhado (S):* sobre o objeto antes de sua leitura
    - bloqueio *exclusivo (X):* sobre o objeto antes de sua escrita
  - Todos os bloqueios feitos por uma transação são liberados quando a transação é completada.
  - Se uma transação obtém um bloqueio do tipo exclusivo sobre um objeto, nenhuma outra transação pode obter um bloqueio (seja do tipo compartilhado ou exclusivo) sobre aquele objeto.

- Variante ⇒ 2PL não-estrito:
  - Libera os bloqueios a qualquer hora, mas não pode obter mais nenhum bloqueio após a liberação de um bloqueio.

Controle de Concorrência Baseado em Bloqueio



- Escalonamento sem controle de bloqueio:

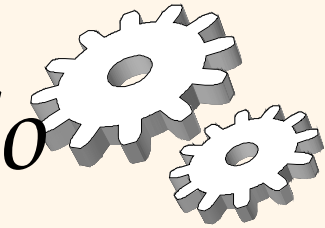
```
T1:  R(A), W(A),                                  R(B), W(B), C
T2:                    R(A), W(A), R(B), W(B), C
```

- Escalonamento com controle de bloqueio:

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

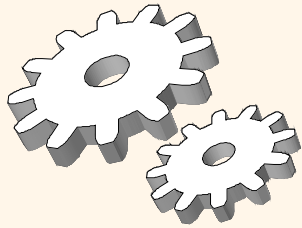
- Se houvesse bloqueio S as transações poderiam ser intercaladas.

Controle de Concorrência Baseado em Bloqueio

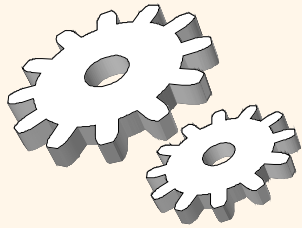


- O 2PL Estrito permite apenas escalonamentos serializáveis.
  - Adicionalmente, simplifica os abortos de transação
  - 2PL não-estrito também permite apenas escalonamentos serializáveis, mas envolve um processamento mais complexo no aborto

Deadlocks



- Imagine o seguinte escalonamento:
  - T1: X(A)
  - T2: X(B)
  - T1: tenta obter acesso exclusivo a B, fica bloqueada
  - T2: tenta obter acesso exclusivo a A, fica bloqueada
- *Deadlocks* devem ser detectados e resolvidos pelo SGBD.
  - Normalmente, usa-se um mecanismo de *timeout*.

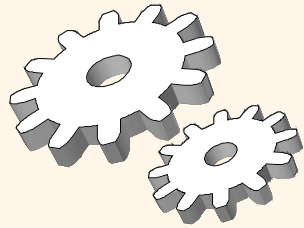


Abortando uma Transação

- Se uma transação *Ti* é abortada, todas as suas ações devem ser desfeitas.
  - se *Tj* lê um objeto que foi escrito pela última vez por *Ti*, *Tj* também deve abortar!
- A maioria dos sistemas evita os *abortos em cascata* liberando os bloqueios da transação apenas no momento do *commit*.
  - Se ***Ti*** altera um objeto, ***Tj*** só pode ler este objeto após o *commit* de *Ti*.

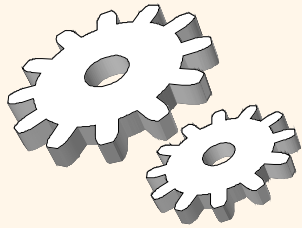
- Para *desfazer* (*undo*) as ações de uma transação abortada, o SGBD mantém um *log* no qual toda escrita é indicada.
  - Este mecanismo também é utilizado para recuperação após queda do sistema:
    - todas as transações ativas no momento da queda são abortadas quando o sistema é reiniciado.

O Log



- As seguintes ações são gravadas no *log*:
  - *Ti escreve um objeto*: valor antigo e valor novo
    - O registro do *log* deve ser salvo no disco *antes* que a página seja modificada!
  - *Ti commit/aborta*: um registro do *log* indicando esta ação
- Registros do *log* são associados a um identificador de transação
  - é fácil desfazer uma transação específica
- O *log* é geralmente *duplicado* e *armazenado* em um meio de armazenamento estável
- Todas as atividades registradas no *log* são tratadas de forma transparente pelo SGBD
  - e de fato, todas as atividades relacionadas ao controle de concorrência como bloqueio/desbloqueio, tratamento de *deadlocks*

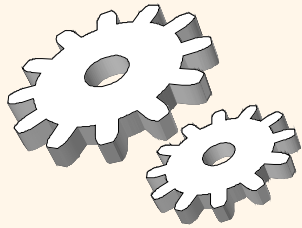
Suporte a Transações em SQL



- Uma transação é iniciada automaticamente, quando é executado um comando de acesso ao BD.
- Uma transação executa um conjunto de comandos, e termina por um dos comandos COMMIT ou ROLLBACK.
- SAVEPOINT (SQL-99) é um ponto da execução que define um estado que pode ser recuperado.
  - Associado ao conceito de transações aninhadas.

    SAVEPOINT <*savepoint name*>

    ROLLBACK TO SAVEPOINT <*savepoint name*>



O que devemos bloquear?

T1:

```
SELECT S.rating,MIN(S.age)
FROM Sailors S
WHERE S.rating=8
```

T2:

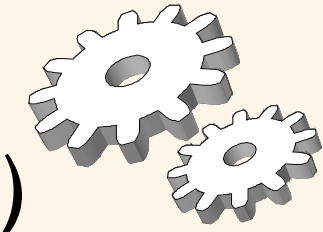
```
UPDATE Sailors S
SET age=20
WHERE S.sid=22
```

- Primeira opção:
  - T1 executa S(Sailors)
  - T2 executa X(Sailors)

  ↳ muito restritivo

  ↳pouca concorrência

O que devemos bloquear? (Cont.)



T1:

```
SELECT S.rating,MIN(S.age)
FROM Sailors S
WHERE S.rating=8
```

T2:

```
UPDATE Sailors S
SET age=20
WHERE S.sid=22
```

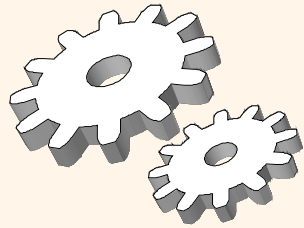
- Segunda opção:
  - T1 bloqueia somente linhas com rating=8
  - T2 bloqueia somente a linha sid=22

  ↳ maior granularidade

  ↳mais complicado

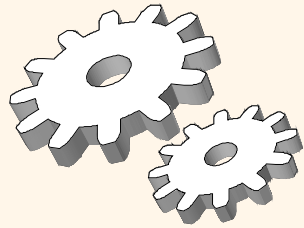
  ↳não impede o fenômeno de "fantasma" (*phantom*)

Características das transações em SQL



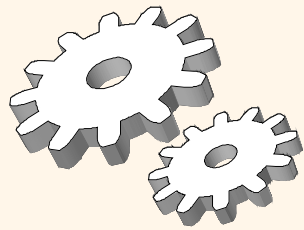
- Modo de acesso:
  - READ ONLY
  - READ WRITE
- Nível de Isolamento:
  - SERIALIZABLE ⇒ mais isolado
  - REPEATABLE READ ⇒ fantasma
  - READ COMMITTED ⇒ fantasma, leitura não repetível
  - READ UNCOMMITTED ⇒ fantasma, leitura não repetível, leitura de dados não salvos
- O sistema faz os bloqueios automaticamente, conforme o nível de isolamento.

Recuperação de Falhas



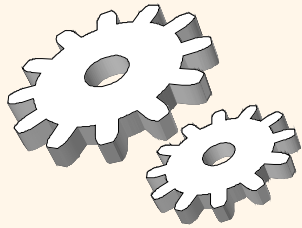
- O Gerenciador de Recuperação de Falhas garante a *atomicidade* e a *durabilidade* das transações.
  - Atomicidade: Desfazendo as ações das transações que não realizaram o *commit*.
  - Durabilidade: Todas as ações das transações que fizeram *commit* serão persistentes.

- O Gerenciador é responsável por recuperar a consistência, após uma queda do sistema.



Recuperação de Falhas

- Há 3 fases no algoritmo de recuperação de *ARIES*:
  - *Analisar*: Percorre o *log* para frente (a partir do ponto de verificação mais recente) para identificar todas as transações que estavam ativas, e todas as "páginas sujas" no momento da falha.
  - *Refazer*: Refaz todas as atualizações das "páginas sujas", quando necessário, para garantir que todas as atualizações registradas no *log* foram realizadas e escritas no disco.
  - *Desfazer*: As escritas de todas as transações que estavam ativas no momento da falha são desfeitas (recuperando o *valor anterior* à atualização registrado no *log* da atualização), varrendo o log de trás para frente.
    - cuidado ao tratar do caso de uma falha que ocorra durante o processo de recuperação!



Resumo

- Controle de concorrência e recuperação estão entre as funções mais importantes oferecidas por um SGBD.
- Os usuários não precisam se preocupar com a concorrência.
  - O sistema automaticamente insere pedidos de bloqueio/desbloqueio e escalona as ações de diferentes transações de forma a garantir que a execução resultante seja equivalente à execução serial das transações.
- *Write-ahead logging* (WAL) é usado para desfazer ações de transações abortadas e para restaurar o sistema a um estado consistente após uma falha.
  - *Estado consistente*: Apenas os resultados das transações consolidadas (*committed*) podem ser vistos.