Bases de Dados

Universidade do Porto

Faculdade de Engenharia

FEUP

- **INTRODUÇÃO**
- **MODELOS CONCEPTUAIS**
  - Diagrama de Classes UML
  - Modelo Entidade-Associação (E-A)
- **MODELO RELACIONAL**
- **LINGUAGEM DE DEFINIÇÃO DE DADOS**
- **INTERROGAÇÃO DE DADOS**
  - Álgebra relacional
  - Linguagem de Manipulação de Dados (LMD)

Índice

- Transações
- Controlo de concorrência
- Recuperação

Transações

Contexto

- Numa arquitectura cliente/servidor (sobretudo mas não só) a execução de um mesmo programa (ou de programas diferentes) por parte de um ou mais utilizadores pode pôr em risco a consistência da informação contida na BD.
  - Exemplo: *O utilizador U1 lê os dados da conta de um cliente e credita 5€ mas entre o momento de leitura do valor e o momento de escrita do novo valor na BD, o utilizador U2 lê os dados da conta do mesmo cliente e debita 2€. Neste caso, quando U1 escreve o valor que leu adicionado de 5€ na BD, o débito feito por U2 perde-se.*
- Para além disso, interrupções imprevistas na execução de um programa podem também originar inconsistências na BD.
  - Exemplo: *Numa transferência bancária, a quebra de energia eléctrica pode fazer com que o montante a transferir tenha sido debitado na conta de origem mas não tenha sido creditado na conta destino.*

transações

Definição

- ***transação*** = uma unidade lógica de trabalho que envolve diversas operações sobre a base de dados.
- São uma sequência de instruções SQL.
- São definidas de forma:
  - ***implícita***, a partir de uma instrução SQL (por defeito), ou
  - ***explícita***, por parte do programador através de instruções próprias para o efeito (a ver mais à frente), envolvendo várias instruções SQL.

transações

Propriedades

As transações devem ser ***ACID***:

- **Atómicas**: cada uma das instruções da transação só toma efeito se todas tomarem efeito
- **Consistentes**: *a* execução de uma transação isoladamente não coloca a BD inconsistente (esta propriedade é da responsabilidade do programador)
- **Isoláveis**: a execução de uma transação em concorrência produz o mesmo efeito da execução isolada
- **Duráveis**: quando uma transação é dada como bem sucedida, os efeitos devem perdurar na BD mesmo que haja falhas de sistema.

transações

Escalonamento

Estabelece o plano de execução das transações:

- Cada transação é descrita pela sequência de acções
- Cada acção é de um destes quatro tipos:
  - Read: leitura da BD
  - Write: escrita na BD
  - Commit: ação que finaliza uma transação bem sucedida guardando de forma persistente as alterações feitas
  - Abort: ação que finaliza uma transação mal sucedida descartando as alterações feitas pela transação e repondo a BD num estado de consistência
- A lista de ações é obtida pelo SGBD a partir das instruções SQL. Não é da responsabilidade do programador defini-la.

transações

Escalonamento

- Notação:
  - **READ**: Leitura
  - **WRITE**: Escrita
  - **COMMIT**: Cometimento
  - **ABORT**: Cancelamento
  - Entre parêntesis coloca-se o objecto da BD afetado pela operação
- Uma transação é finalizada por um **COMMIT** ou por um **ABORT**:
  - Tanto um como o outro não têm argumentos
  - Estas duas ações são muitas vezes omitidas na representação do escalonamento

Exemplo

T1	T2
READ(A)	
WRITE(A)	
	READ(B)
	WRITE(B)
READ(C)	
WRITE(C)	

transações

Escalonamento

- Os escalonamentos que contêm um *commit* ou um *abort* por cada transação diferente que contenha, designam-se por **escalonamentos completos**
- Os escalonamentos que não têm ações alternadas de diferentes transações, i.e., cada transação só começa quando a transação precedente termina, designam-se por **escalonamentos em série** (ou **seriais**)

transações

Motivação para escalonamentos alternados

- Quando os escalonamentos são em série, algumas transações, especialmente as mais pequenas, podem ter tempos de resposta inesperadamente grandes por serem precedidas por transações grandes
- Permitindo, de forma controlada, a execução alternada de acções de diferentes transações, é possível aumentar o nível de satisfação global dos utilizadores das BDs
- Isso é potenciado pela possibilidade de executar actividades de CPU e de I/O (i.e., de leitura/escrita na BD), em paralelo

transações

Definição informal de escalonamento serializável

- Um escalonamento diz-se **serializável** quando o resultado da sua execução é idêntico ao resultado de uma das possíveis execuções em série das transações que o compõem.
  - É óbvio que a ordem pela qual as transações são executadas (em série) pode afetar o resultado final
  - A execução serializável deve garantir a equivalência para com uma das possíveis execuções em série, não sendo possível determinar a priori qual delas

transações

Possíveis problemas dos escalonamentos alternados (1)

- Conflito WR[1]: leitura de dados não cometidos
- Conhecido por **leitura suja** (*dirty read*)
- Exemplo:
  - (1) T1 transfere 100€ de A para B começando por debitar o valor de A; (2) T2 incrementa tanto A como B em 2% (juros anuais); e (3) T1 finaliza a transação creditando o valor (entretanto aumentado) em B.

T1	T2
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
	READ(B)
	WRITE(B)
	COMMIT
READ(B)	
WRITE(B)	
COMMIT	

[1] lê-se: T_j faz R após T_i ter feito W sobre o mesmo objecto

transações

Possíveis problemas dos escalonamentos alternados (2)

- Conflito RW: escrita de dados entretanto lidos por outra transação
- Conhecido por **leitura irrepetível**
- Exemplo:
  - (1) T1 lê a quantia disponível em A; (2) T2 lê e credita 20€ em A; e (3) T1 finaliza a transação creditando o valor de 100€ em A.

T1	T2
READ(A)	
	READ(A)
	WRITE(A)
	COMMIT
WRITE(A)	
COMMIT	

Possíveis problemas dos escalonamentos alternados (3)

- Conflito WW: reescrita de dados não cometidos
- Conhecido por **escrita cega** (*blind write*)
- Exemplo:
  - (1) T1 escreve 100 em A; (2) T2 escreve 200 em B; (3) T1 escreve 100 em B; e (4) T2 escreve 200 em A.
  - O resultado deste escalonamento não é idêntico a nenhuma das duas possíveis execuções em série. É só testar ...

T1	T2
WRITE(A)	
	WRITE(B)
WRITE(B)	
	WRITE(A)
COMMIT	
	COMMIT

transações

Exercícios

- Considere as seguintes acções da transação T1 sobre os objectos A e B:
  - READ(A), WRITE(A), READ(B), WRITE(B)

a) Dê um exemplo de uma outra transação T2 que, se executada em concorrência com T1 sem nenhuma forma de controlo de concorrência, possa interferir com T1.

transações

Exercícios

T2: READ(B), WRITE(B)
T1: READ(A); A:=2*A; READ(B); B:=A+B
T2: READ(A); B:=2*A
Exemplo para A=100 e B=10

Escalonamento alternado

T1	T2	A	B
READ(A)		100	
WRITE(A)		2*A=200	
	READ(A)	200	
	WRITE(B)		2*A=400
READ(B)			400
WRITE(B)			A+B=600

Escalonamento em série (T1;T2)

T1	T2	A	B
READ(A)		100	
WRITE(A)		2*A=200	
READ(B)			10
WRITE(B)			A+B=210
	READ(A)	200	
	WRITE(B)		2*A=400

Escalonamento em série (T2;T1)

T1	T2	A	B
	READ(A)	100	
	WRITE(B)		2*A=200
READ(A)		100	
WRITE(A)		2*A=200	
READ(B)			200
WRITE(B)			A+B=400

transações

Definição formal de escalonamento serializável

- Até agora não se considerou a possibilidade de uma transação ser cancelada. Ver exemplo …

- Um **escalonamento serializável** sobre um conjunto de transações *S* é um escalonamento cujo efeito sobre qualquer BD consistente é garantidamente idêntico ao de um escalonamento em série completo sobre o conjunto de transações cometidas de *S*.

T1	T2	Exemplo
READ(A)		
WRITE(A)		A:=A-100
	READ(A)	
	WRITE(A)	A:=1.02*A
	READ(B)	
	WRITE(B)	B:=1.02*B
	COMMIT	
ABORT		

transações

Escalonamento recuperável

- Um escalonamento diz-se **recuperável** se cada transação que o compõe é cometida só depois de (e se) todas as transações cujas alterações foram lidas por essa transação terem sido cometidas
- No último exemplo dado, o escalonamento era irrecuperável.
- Num escalonamento **recuperável** é possível cancelar a transação sem ter de o fazer em cascata.

transações

Possíveis problemas ao cancelar uma transação

- Quando uma transação é cancelada, deve-se colocar a BD num estado consistente, desfazendo as ações até então executadas
- Exemplo de problemas ao cancelar uma transação:
  - Neste exemplo, ao cancelar T1, o valor de A é reposto a 5, perdendo-se as alterações feitas por T2, mesmo que o COMMIT seja feito antes do ABORT
  - Este problema pode acontecer em conflitos WW
  - Para o evitar são necessários mecanismos de recuperação complexos

T1	T2	Exemplo
READ(A)		A:=5
WRITE(A)		A:=6
	WRITE(A)	A:=7
ABORT		
	COMMIT	

Contr. de concorrência c/ bloqueios

Protocolo de bloqueios

- Um **protocolo de bloqueios** é um conjunto de regras que cada transação deve seguir (e que são asseguradas pelo SGBD) de forma a que o escalonamento definido seja serializável

Contr. de concorrência c/ bloqueios

Protocolo em duas fases estrito

- As duas fases são:
  1. Se uma transação T quer ler ou modificar um objecto, tem de previamente requerer respetivamente um bloqueio partilhado ou exclusivo sobre o objecto;
     - Os bloqueios partilhados só permitem ler
     - Os bloqueios exclusivos permitem ler e escrever
     - Uma transação que requer um bloqueio fica em espera até que o SGBD possa executá-lo
  2. Todos os bloqueios requeridos por uma transação só são libertados quando a transação termina. j1
- O cumprimento do protocolo em duas fases estrito garante escalonamentos recuperáveis

Diapositivo 20

j1 Ver se está certo. Isso não é só o estrito?
jmoreira; 04-07-2012

Contr. de concorrência c/ bloqueios

Protocolo em duas fases estrito

T1	T2
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
	READ(B)
	WRITE(B)
	COMMIT
READ(B)	
WRITE(B)	
COMMIT	

- Notação:
  - **RLOCK**: Bloqueio partilhado
  - **WLOCK**: Bloqueio exclusivo
  - **UNLOCK**: Desbloqueio
  - Entre parêntesis coloca-se o objecto da BD afectado pela operação
- O exemplo do lado esquerdo ficaria ...
  - Neste caso resultaria num escalonamento em série, mas nem sempre é assim.

T1	T2
WLOCK(A)	
READ(A)	
WRITE(A)	
WLOCK(B)	
READ(B)	
WRITE(B)	
COMMIT	
UNLOCK(A)	
UNLOCK(B)	
	WLOCK(A)
	READ(A)
	WRITE(A)
	WLOCK(B)
	READ(B)
	WRITE(B)
	COMMIT
	UNLOCK(A)
	UNLOCK(B)

Contr. de concorrência c/ bloqueios

Protocolo em duas fases

- A diferença do protocolo em duas fases para o protocolo em duas fases estrito é:
  - A segunda regra é substituída por: uma transação não pode requerer bloqueios adicionais se já tiver feito algum desbloqueio; j2
    - Ou seja, a diferença para a versão estrita é que a segunda fase, apesar de só conter desbloqueios, é feita de forma progressiva, em vez de desbloquear tudo só no fim
    - O cumprimento do protocolo em duas fases não garante escalonamentos recuperáveis

Diapositivo 22

j2 Rever isto.
jmoreira; 04-07-2012

Contr. de concorrência c/ bloqueios

Equivalência em conflitos

- Diz-se que dois escalonamentos têm **equivalência de conflitos** se compreendem as ações das mesmas transações e ordenam cada par de ações conflituosas de duas transações cometidas da mesma forma
  - Lembrar que há três tipos de acções conflituosas: WR, RW e WW
  - O resultado de um escalonamento só depende da ordem das ações conflituosas
- Um escalonamento diz-se **serializável por conflitos** se tem **equivalência de conflitos** com algum escalonamento em série
- Qualquer escalonamento **serializável por conflitos** é serializável, sob o pressuposto que não se insere nem se apaga registos da BD (mas pode-se alterar valores)
  - O contrário pode não se verificar

Contr. de concorrência c/ bloqueios

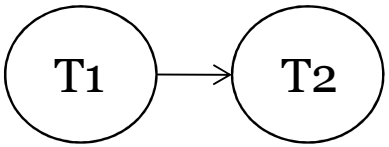
Grafo de precedências

- O **grafo de precedências** para o escalonamento *S* contém
  - Um nó por cada transação cometida em *S*
  - Um arco de T_i para T_j se uma acção de T_i precede e entra em conflito com uma acção de T_j
- Um escalonamento *S* é **serializável por conflitos** se e só se o seu grafo de precedência for acíclico
- Qualquer escalonamento que assegure o bloqueio em duas fases ou em duas fases estrito tem um grafo de precedências acíclico

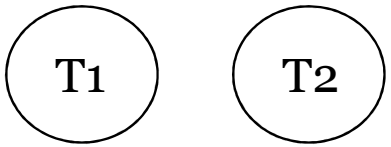
Contr. de concorrência c/ bloqueios

Exemplos: grafo de precedências

T1	T2
WLOCKA)	
READ(A)	
WRITE(A)	
WLOCK(B)	
READ(B)	
WRITE(B)	
COMMIT	
	WLOCK(A)
	READ(A)
	WRITE(A)
	WLOCK(B)
	READ(B)
	WRITE(B)
	COMMIT



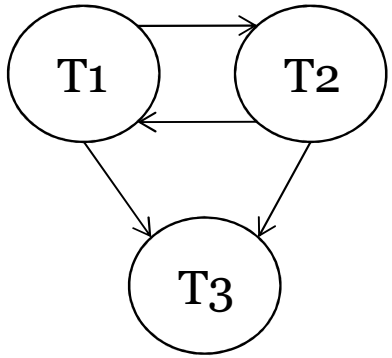
T1	T2
RLOCKA)	
READ(A)	
	RLOCK(A)
	READ(A)
	WLOCK(B)
	READ(B)
	WRITE(B)
	COMMIT
WLOCK(C)	
READ(C)	
WRITE(C)	
COMMIT	



Contr. de concorrência c/ bloqueios

Exemplos: grafo de precedências

T1	T2	T3
READ(A)		
	WRITE(A)	
	COMMIT	
WRITE(A)		
COMMIT		
		WRITE(A)
		COMMIT



Contr. de concorrência c/ bloqueios

Gestor de bloqueios

- A componente do SGBD que guarda informação dos bloqueios existentes designa-se por **gestor de bloqueios**
- O **gestor de bloqueios** mantem uma tabela de bloqueios
  - Cada registo dessa tabela contém o número de transações que têm um bloqueio sobre um dado objecto, a natureza desse bloqueio (partilhado ou exclusivo) e um apontador para a lista de pedidos de bloqueio
  - O número de transações que têm um bloqueio sobre um dado objecto pode ser maior do que um se o bloqueio for partilhado

Contr. de concorrência c/ bloqueios

Pedidos de bloqueio ou de desbloqueio

- Os bloqueios podem ser feitos sobre objetos de diferentes tipos (dependendo do SGBD): páginas, registos, …
  - As páginas são as unidades elementares para acesso a disco. A identificação da página onde se encontra cada registo é feita pelo gestor de ficheiros do SGBD
- Quando uma transação faz um pedido de bloqueio sobre um objeto ao gestor de bloqueios:
  - Se é um bloqueio partilhado, se a fila de pedidos sobre esse objecto está vazia e se o objeto não se encontra bloqueado por um bloqueio exclusivo, o gestor de bloqueios executa o bloqueio e actualiza a tabela de bloqueios
  - Se é um bloqueio exclusivo e não há nenhum bloqueio sobre esse objeto (o que implica não existir nenhum pedido na fila), o bloqueio é executado e a tabela de bloqueios é atualizada
  - Caso contrário, o pedido não é concedido, sendo adicionado à fila de pedidos de bloqueio desse objecto, e a transação fica suspensa
- Quando um bloqueio sobre um objeto é libertado, o próximo pedido na fila de bloqueios desse objeto é avaliado

Contr. de concorrência c/ bloqueios

Possíveis problemas dos protocolos de bloqueios

- Bloqueio ativo (*livelock*):
  - T_1 e T_2 pedem bloqueio de A; T_1 obtém; T_3 pede bloqueio de A; quando T_1 desbloqueia, T_3 é servido e T_2 fica à espera…
  - Hipótese de resolução: servir sempre o pedido mais antigo. Fácil.
- Encravamento (*deadlock*):
  - Seja:
    - T1: WLOCK(A); WLOCK(B); UNLOCK(A); UNLOCK(B);
    - T2: WLOCK(B); WLOCK(A); UNLOCK(B); UNLOCK(A).
  - Caso o escalonamento seja feito, por exemplo, pela ordem:
    - $WLOCK_{T1}(A)$; $WLOCK_{T2}(B)$; $WLOCK_{T1}(B)$; $WLOCK_{T2}(A)$; …
  - Encrava …
  - Hipótese de resolução: complexo … Tema não leccionado.
- O problema fantasma: Tema não leccionado.

Contr. de concorrência c/ bloqueios

transações SQL-92

- As transações são iniciadas de forma automática quando o utilizador executa uma instrução que modifica a base de dados ou o catálogo
- As transações terminam:
  - Explicitamente com uma das instruções:
    - COMMIT: faz o cometimento da transação
    - ROLLBACK: cancela a transação
  - Implicitamente: quando é executada uma instrução LDD-SQL

Contr. de concorrência c/ bloqueios

transações SQL-92

- Características das transações:
  - Dimensão do diagnóstico: especifica o número de erros que podem ser registados na área de diagnóstico
  - Modo de acesso:
    - READ ONLY: só permite leituras
    - READ WRITE: permite leituras e escritas
  - Nível de isolamento: controla o quanto uma transação fica exposta a outras transações concorrentes, ou seja, permite dosear entre escalonamentos mais alternados e o risco de não garantir a seriabilidade desse escalonamento. Existem quatro níveis de isolamento:
    - Serializável (*serializable*)
    - Leitura repetível (*repeatable read*)
    - Leitura cometida (*read commited*)
    - Leitura não cometida (*read uncommitted*)

Contr. de concorrência c/ bloqueios

transações SQL-92

- O nível de isolamento **serializável** assegura que a transação T:
  - Só lê alterações feitas por transações já cometidas;
  - Nenhum valor lido ou escrito por T é alterado por outra transação até T terminar;
  - O conjunto de valores correspondente a um dado critério de pesquisa e que tenha sido lido por T, não pode ser alterado por nenhuma transação até T completar (ou seja, evita o problema fantasma).

Na implementação baseada em bloqueios, uma transação serializável faz bloqueios antes de ler ou escrever nos objetos, incluindo bloqueios em conjuntos de objetos de forma a evitar o problema fantasma

Contr. de concorrência c/ bloqueios

transações SQL-92

- O nível de isolamento **leitura repetível** assegura que a transação T:
  - Só lê alterações feitas por transações já cometidas;
  - Nenhum valor lido ou escrito por T é alterado por outra transação até T terminar.

A única diferença entre os níveis de isolamento serializável e leitura repetível é que este último não faz bloqueio aos índices, ou seja pode bloquear objetos mas não garante o bloqueio a todos os objetos associados (dinamicamente) a um critério de pesquisa

Contr. de concorrência c/ bloqueios

transações SQL-92

- O nível de isolamento **leitura cometida** assegura que a transação T:
  - Só lê alterações feitas por transações já cometidas;
  - Nenhum valor escrito por T é alterado por outra transação até T terminar sendo, no entanto, possível que um valor lido por T seja modificado por outra transação antes de T terminar.

A única diferença entre os níveis de isolamento leitura repetível e leitura cometida é que, neste último, os bloqueios de leitura são libertados logo após a leitura. Os bloqueios de escrita, em ambas as situações, só são libertados no fim.

Contr. de concorrência c/ bloqueios

transações SQL-92

- O nível de isolamento **leitura não cometida** assegura que a transação T:
  - Pode ler alterações feitas a um objeto por uma transação em curso

  Na leitura não cometida não são efetuados bloqueios partilhados antes da leitura dos objetos. Devido ao elevado nível de exposição a alterações não cometidas, a SQL só permite associar este nível de isolamento a transações com modo de acesso READ ONLY. Sendo só de leitura e não fazendo bloqueios de leitura, significa que não faz qualquer tipo de bloqueio

Contr. de concorrência c/ bloqueios

transações SQL-92

- Resumo:

Nível	Leitura suja	Leitura irrepetível	Fantasma
Serializável	Não	Não	Não
Leitura repetível	Não	Não	Eventualmente
Leitura cometida	Não	Eventualmente	Eventualmente
Leitura não cometida	Eventualmente	Eventualmente	Eventualmente

Contr. de concorrência c/ bloqueios

transações SQL-92

- O nível de isolamento serializável é o mais seguro, sendo o mais adequado para a maioria das situações. No entanto, se outro nível de isolamento for adequado para uma dada transação, a sua utilização poderá melhorar o desempenho do sistema
  - Exemplo: utilizando o nível de isolamento leitura cometida ou leitura não cometida, é possível calcular a média de idades dos alunos da FEUP de forma mais eficiente e sem distorcer, de forma significativa, o valor verdadeiro, pois a perda ou ganho de um ou dois valores não alterará significativamente a média

Contr. de concorrência c/ bloqueios

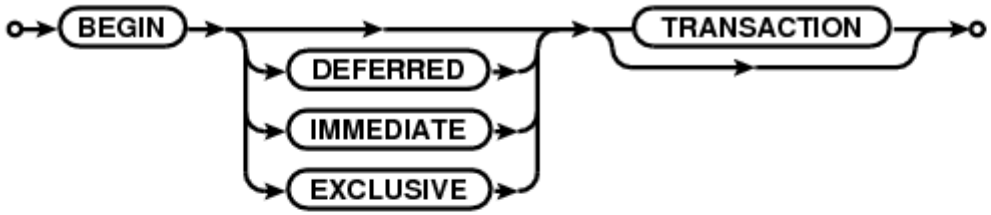
transações SQL-92

- Instrução para definir o modo de acesso e o nível de isolamento de uma transação: SET TRANSACTION
- Usando o exemplo anterior:
  - SET TRANSACTION ISOLATION LEVEL READ UNCOMMITED READ ONLY
- Quando uma transação se inicia, os valores por defeito para o nível de isolamento e o modo de acesso são, respetivamente, SERIALIZABLE e READ WRITE

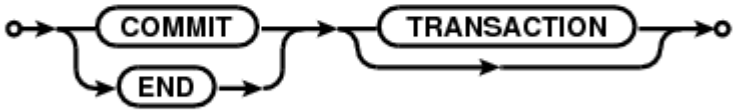
Contr. de concorrência c/ bloqueios

transações em SQLite

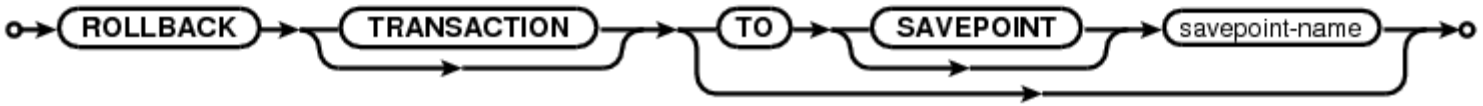
begin-stmt:



commit-stmt:



rollback-stmt:



Contr. de concorrência c/ bloqueios

transações em SQLite

- Modos:
  - ***Deferred***: É o modo usado por defeito. *Deferred* significa que nenhum *LOCK* é feito até à à primeira operação de leitura ou escrita. Ou seja, a declaração *BEGIN* em si não faz nada. A primeira operação de leitura da BD cria um bloqueio partilhado e a primeira operação de gravação cria um bloqueio exclusivo;
  - ***Immediate***: Neste caso, bloqueios exclusivos são feitos logo após o *BEGIN*. Depois de um *BEGIN IMMEDIATE*, nenhum outro processo pode escrever na base de dados nem fazer *BEGIN IMMEDIATE* ou *BEGIN EXCLUSIVE*. Mas podem, no entanto fazer leitura da base de dados;
  - ***Exclusive***: Após um *BEGIN EXCLUSIVE*, nenhuma outra ligação à base de dados, excepto ligações *read_uncommitted* poderão ler a base de dados e nenhum outro processo, sem excepção, poderá escrever na base de dados até a transacção estar completa.

Contr. de concorrência s/ bloqueios

- A utilização de protocolos de bloqueio para controlo de concorrência é uma abordagem pessimista na medida em que cancelam transações ou bloqueiam objectos de forma preventiva para resolver conflitos. Em sistemas com níveis baixos de concorrência, o custo deste tipo de abordagens pode ser significativo.
- Existem protocolos alternativos que optam por, mais do que prevenir os conflitos, tentar resolvê-los. Parte-se do princípio que os conflitos são eventos raros e, por isso, o custo de resolução, sendo elevado por cada conflito, tem um custo global baixo.

Contr. de concorrência s/ bloqueios

Controlo de concorrência baseada em marcas temporais

- Princípios do controlo de concorrência por marcas temporais:
  1. É atribuída a cada transação Ti uma marca temporal TS(Ti) quando Ti é iniciada
  2. Assegurar durante a execução que, no caso de haver conflito entre a acção Ai de Ti e a acção Aj de Tj, Ai ocorre antes de Aj se TS(Ti) < TS(Tj)
  3. Se uma acção viola esta ordenação, a transação é cancelada e reiniciada
- Para implementar esta forma de controlo de concorrência, define-se, para cada objecto (O), a marca temporal da transação que fez a última leitura e a última escrita, respetivamente TL(O) e TE(O).

Contr. de concorrência s/ bloqueios

Controlo de concorrência baseada em marcas temporais

- Recuperabilidade:
  - Infelizmente, o protocolo de marcas temporais permite escalonamentos que não são recuperáveis (ver exemplo).
  - Este problema pode ser resolvido modificando o protocolo de marcas temporais de forma a não permitir estes escalonamentos, passando todas as ações de escrita para *buffer* até a transação cometer.
  - No exemplo, quando T1 quer escrever A, TE(A) é atualizado, mas a alteração de A só é registada no *buffer*. Quando T2 quer, a seguir, ler A, T2 é bloqueada até T1 completar (depois de TS(T2) ser comparada com TE(A) e a leitura ser vista como permissiva). Se T1 cometer, a alteração que fez a A é copiada do *buffer*; caso contrário, as alterações que constam no *buffer* são descartadas.

T1	T2
WRITE(A)	
	READ(A)
	WRITE(B)
	COMMIT

Recuperação

Gestor de recuperações

- O gestor de recuperações de um SGBD é responsável por assegurar:
  - Atomicidade: desfazendo as ações das transações ainda não cometidas quando uma transação é cancelada
  - Durabilidade: garantindo que todas as ações das transações cometidas perduram a falhas do sistema
- Guarda num jornal (*log*) todas as alterações feitas à BD

Recuperação

Causas para a necessidade de recuperação

- Existem diferentes causas para que seja necessário iniciar um processo de recuperação:
  - Falha lógica de sistema (falha de energia, *core dump*, …)
  - Falha física do sistema de armazenamento (disco, …)
  - Interação de outros algoritmos com o controlo de concorrência

Recuperação

O jornal

- Contém o historial das ações executadas pelo SGBD.
- É importante minimizar a possibilidade de ele perder informação. Exemplo: escrevendo duas cópias em discos diferentes e, porque não, em locais diferentes.
- A componente mais recente do jornal é mantida na memória principal e guardada periodicamente em local seguro.
- A cada entrada do jornal é atribuída um número sequencial crescente (i.e., números maiores correspondem a entradas mais recentes), designado *Last Sequence Number (LSN)*.
- Para facilitar a recuperação, cada página da BD tem o número sequencial da última ação de alteração feita sobre essa página, designado por *pageLSN*.

Recuperação

O jornal

- Acrescenta-se uma entrada ao jornal nas seguintes situações:
  - Página alterada: acrescenta-se um registo do **tipo alteração**. O page*LSN* é atualizado com o valor do *LSN* do novo registo.
  - Transação cometida: acrescenta-se um registo do **tipo cometimento**. De seguida, guarda-se em disco a componente do jornal mantida em memória, incluindo a entrada referente ao cometimento. O cometimento fica, assim, terminado, mas há ainda outros passos a realizar (ex.: remover a transação da tabela de transações).
  - Transação cancelada: acrescenta-se um registo do **tipo cancelamento**. De seguida, a ação desfazer é iniciada.
  - Transação finalizada: acrescenta-se um registo do **tipo fim**. Este registo só é criado após terem sido realizadas todas as ações efectuadas após o cancelamento ou o cometimento de uma transação (já referidas anteriormente).
  - Alteração desfeita: acrescenta-se um registo do **tipo compensação** quando a ação descrita por um registo do tipo alteração é desfeita. Tal pode ocorrer após cancelamento de uma transação ou após recuperação devido a falha.

Recuperação

O jornal

- Atributos comuns a todos os registos (seja qual for o tipo):
  - *prevLSN*: *LSN* do registo anterior referente à mesma transação;
  - *transID*: *ID* da transação que gerou o registo;
  - tipo: tipo do registo (alteração, cometimento, cancelamento, fim ou compensação).
- Alguns dos atributos adicionais dos registos do tipo 'alteração':
  - *pageID*: *ID* da página alterada.
  - *before_image*: valor antes da alteração. Permite desfazer uma alteração.
  - *after_image*: valor após a alteração. Permite refazer uma alteração.
- Atributo adicional dos registos de 'compensação':
  - *before_image*:
  - *undoNextLSN*: contém o *LSN* do próximo registo do jornal a desfazer. Este atributo terá o valor de *prevLSN* do registo da alteração a ser desfeita.

Recuperação

Ponto de verificação

- Até onde percorrer o jornal numa recuperação?
- Operação de verificação (*checkpoint*)
  - proibir o início de transações e esperar que as activas estejam cometidas ou abortem
  - copiar os blocos alterados na memória central para memória estável
  - registar no jornal o ponto de verificação e escrever o jornal
- Só é necessário analisar o jornal até ao último ponto de verificação, para recuperar de uma falha
  - a parte anterior do jornal só interessa para arquivo de segurança

Recuperação

Algoritmo de recuperação ARIES

- **ARIES** (**A**ggregate **R**ecoverable **I**tem **E**valuation **S**ystem)
- Quando o gestor de recuperações é invocado após falha do sistema, a reinicialização é feita em três fases:
  - **Analisar**: identifica páginas sujas do *buffer* (i.e., que não tenham sido escritas em disco) e transações ativas no momento da falha do sistema
  - **Refazer**: repete todas as ações desde um determinado ponto do jornal, designado por ponto de verificação (*checkpoint*), repondo a base de dados no estado em que se encontrava antes da falha do sistema.
    - **ATENÇÃO**: As transações não são re-executadas! O que acontece é que são atrbuídos aos objetos os valores consequentes das ações. Exemplo: Se a ação foi a de atribuir ao objeto A o valor de A + 5, e se o resultado dessa operação foi 20, ao refazer, atrbui-se a A o valor de 20, de acordo com a informação do jornal, mas a operação não é feita de novo, só o valor é que é reposto.
  - **Desfazer**: desfaz as ações das transações que não foram cometidas

Recuperação

Exemplo: algoritmo de recuperação ARIES

LSN	Jornal
1	Update: T1 writes P5
2	Update: T2 writes P3
3	T2 commit
4	T2 end
5	Update: T3 writes P1
6	Update: T3 writes P3
x	CRASH, RESTART

LSN: Log Sequence Number

- **Analisar**: identifica as transações que estavam activas quando o sistema falhou, T1 e T3; identifica as transações cometidas, T2; e identifica as páginas que possam estar sujas, P1, P3 e P5.
- **Refazer**: repõe os valores que estão guardados no jornal, pela ordem registada no jornal, para as transações T1, T2 e T3.
- **Desfazer**: finalmente, as acções T1 e T3 são desfeitas, nomeadamente, as escritas de P3 por T3, de P1 por T3 e de P5 por T1.

Recuperação

Algoritmo de recuperação ARIES

- Princípios básicos:
  - Escrita prévia no jornal: uma alteração, antes de ser realizada, é registada no jornal; o registo feito no jornal deve ser guardado em local seguro (disco) antes de a alteração à base de dados ser escrita em disco.
  - Repetir todas as ações durante o refazer: depois de uma falha do sistema, todas as ações são feitas de novo mesmo as ações daquelas transações que, não tendo sido cometidas no momento da falha, tenham de ser canceladas.
  - Registar no jornal as ações da fase 'desfazer': as alterações feitas à base de dados quando se desfazem as ações das transações que não foram cometidas, devem ser registadas no jornal para garantir a recuperabilidade no caso de haver nova falha de sistema. O seguro morreu de velho ☺