

Transações Controlo de Concorrência Recuperação de Falhas

Base de Dados - 2018/19

Carlos Costa

1

Introdução

- SGBD é um intermediário entre a aplicação e a base de dados (BD) propriamente dita.
- SGBD tem um sistema de processamento de operações sobre a BD.
- SGBD é multi-utilizador
 - Processamento simultâneo de operações solicitadas por distintos utilizadores.
 - execução intercalada de conjuntos de operações
- Transação é uma unidade lógica de trabalho contendo uma ou mais operações.

2

Transação - Operações de Leitura e Escrita

- De uma forma simples, podemos ver uma **transação** como um **conjunto de operações** de leitura (**read**) e escrita (**write**) sobre a base de dados.
- **read(x)** - transfere o elemento X da base de dados para a área de memória volátil associada à transação que executou a operação de leitura.
- **write(x)** - transfere o elemento X da área de memória afeta à transação para a base de dados.

3

Transação - Exemplo “clássico”

- Supondo que se pretende fazer a transferência (T_i) de 50€ entre 2 contas bancárias, A e B.
- A transação consiste em debitar o valor 50 em A e creditá-lo em B. Pode ser definida como:

T_i :

```

1      Begin Transaction
2          read(A);
3          A:=A-50;
4          write(A);
5          read(B);
6          B:=B+50;
7          write(B);
8      End Transaction

```

Transacção:
unidade lógica contendo
várias operações

4



Transações em SQL Standard

- SQL Padrão (SQL-92)
 - **SET TRANSACTION**
 - **inicia e configura características de uma transação**
 - **COMMIT [WORK]**
 - **encerra a transação (solicita efetivação das suas ações)**
 - **ROLLBACK [WORK]**
 - **solicita que as ações da transação sejam desfeitas**
- **Por defeito, um comando individual é considerado uma transação**
 - exemplo: `DELETE FROM Pacientes WHERE PID=5;`

5



Transação em SQL Server

Iniciada com a instrução:
BEGIN TRANSACTION

Terminada com:

- **Sucesso: COMMIT**
- **Insucesso (Falha): ROLLBACK**

-- Exemplo

```
BEGIN TRANSACTION
UPDATE authors SET au_lname = upper(au_lname)
WHERE au_lname = 'White'
IF @@ROWCOUNT = 2
    COMMIT TRAN
ELSE
BEGIN
    PRINT 'A transaction needs to be rolled
back'
    ROLLBACK TRAN
END
```

ROLLBACK implícito

- **Ocorre se, por alguma razão, a transação não termina de modo esperado (i.e. com COMMIT ou ROLLBACK explícito)**

6

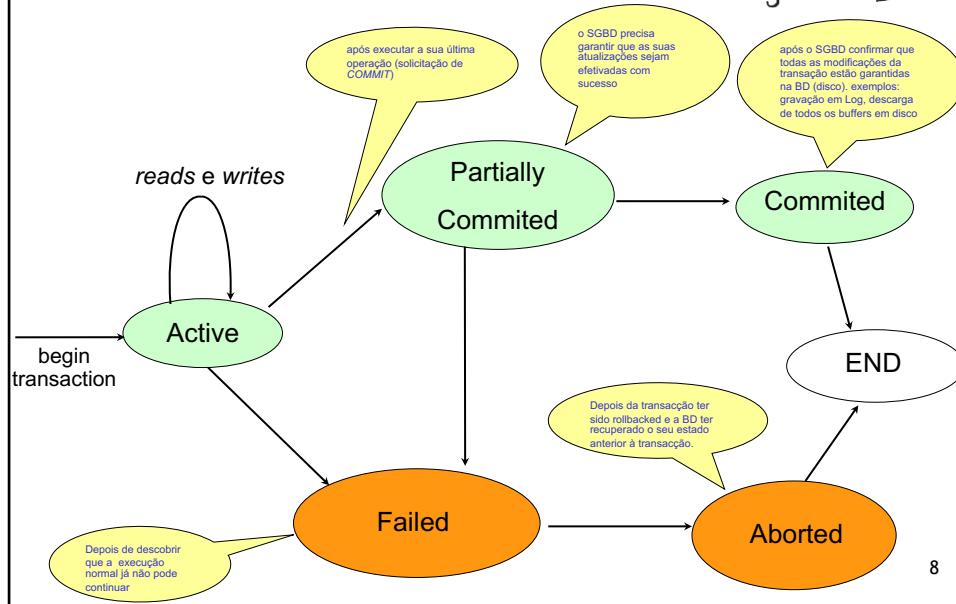
Estado de uma Transação

- Uma transação passa por vários estados que são controlados pelo SGBD
 - que operações já realizou? concluiu as suas operações? deve abortar?
- Estados de uma transação
 - Active; Partially Committed; Committed; ~~Aborted~~; Failed; ~~Concluded~~.
 - Respeita um Grafo de Transição de Estados

7

Transição de Estados de uma Transação

(Diagrama)



8

Propriedades de uma Transação

ACID (Atomicity, Consistency, Isolation, Durability):

- Atomicidade: as operações da transação ocorrem de forma indivisível (atómica), i.e.:
 - ou todas (**commit**) - executada com sucesso
 - ou nenhuma (**rollback**) - falha
- Consistência: Após as operações o estado de integridade tem de se manter.
- Isolamento: O sistema deve dar a cada transação a ilusão de ser única. As transações concorrentes não interferem entre si.
- Persistência: os efeitos de uma transação terminada com um **commit** são permanentes e visíveis para outras transações.

9

Atomicidade

- Princípio do “Tudo ou Nada”
 - ou todas as operações da transação são efetivadas com sucesso na BD ou nenhuma delas se efetiva
 - fundamental para preservar a integridade do BD
- É da responsabilidade do SGBD a recuperação de falhas
 - desfazer as operações da transação parcialmente executadas.
- Exemplo “clássico”:
 - E se o sistema falhar a meio da transação?
 - entre o write(A) e o write(B)
 - motivo... falta de energia, falha na máquina ou erros de software
 - Base de dados corrompida -> Estado de Inconsistência
 - desapareceriam 50€ da conta A que nunca chegaram à B
 - Conclusão: Só faz sentido efetuarmos ambas as operações em conjunto.
 - Ação: as operações prévias à falha devem ser desfeitas

TI:	Begin Transaction
1	Begin Transaction
2	read(A);
3	A:=A-50;
4	write(A);
5	read(B);
6	B:=B+50;
7	write(B);
8	End Transaction

10

Consistência

- Uma transação deve **transportar sempre a base de dados de uma estado de integridade para outro estado de integridade.**
- Responsabilidade:
 - do programador da aplicação que codifica a transação
 - do SGBD no caso de falhas (crash) do sistema
- Durante a execução pode ser momentaneamente violada mas no final a integridade tem de ser garantida.
 - Entre a linha 4 e 7 no exemplo anterior ->

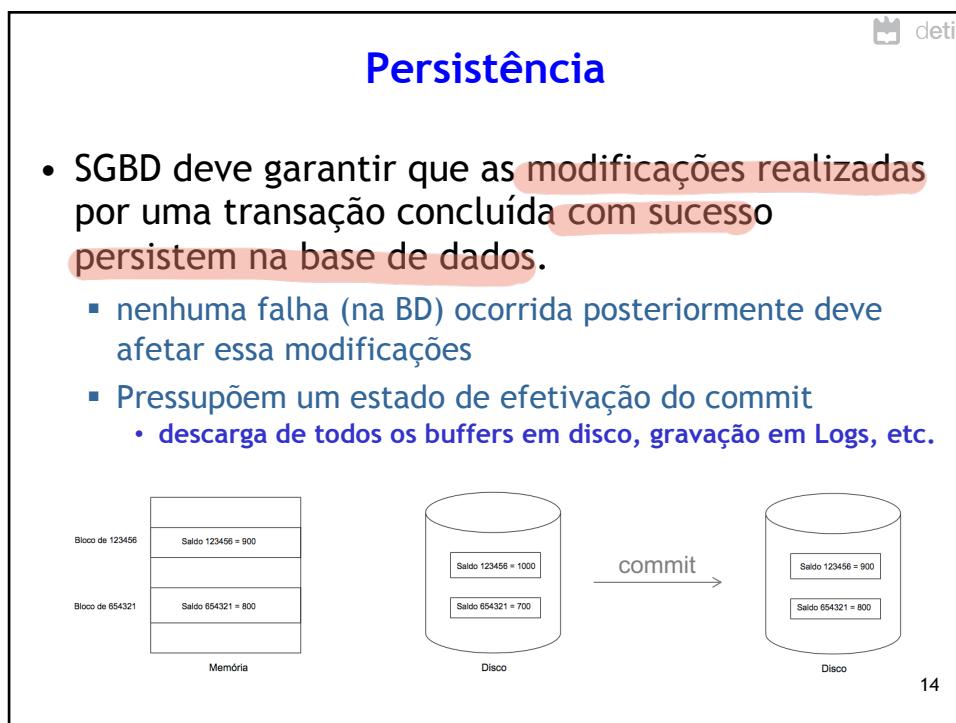
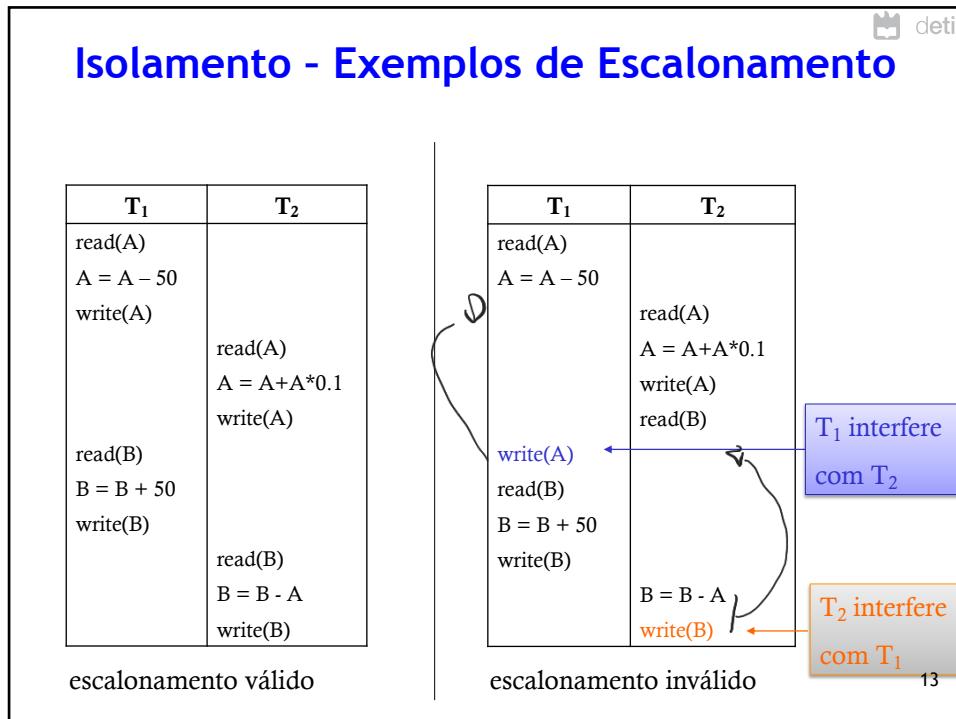
```
Ti:
1   Begin Transaction
2   read(A);
3   A:=A-50;
4   write(A);
5   read(B);
6   B:=B+50;
7   write(B);
8   End Transaction
```

11

Isolamento

- É desejável que as transações possam ser executadas de forma concorrente.
- No entanto, a execução de uma transação Ti deve ser realizada como se ela estivesse a ser executada de forma isolada
 - Ti não deve sofrer interferências de outras transações executadas concorrentemente.
- Garante que a execução simultânea das transações resulta numa estado equivalente ao que seria obtido caso elas tivessem sido executadas em série (uma de cada vez).
- Recurso a técnicas de **escalonamento (schedule)**
 - Define a ordem pela qual são executadas as operações read/write, do conjunto de transações concorrentes.

12



CONTROLO DE CONCORRÊNCIA

15

Controle de Concorrência - Transações

Garantia de isolamento de transações:

- Escalonamento Serializado
 - uma transação executada de cada vez - de forma sequencial
 - solução bastante ineficiente
 - transações podem esperar muito tempo pela execução
 - desperdício de recursos...
- Escalonamento Concorrente Serializado
 - execução concorrente de transações mas de modo a preservar o isolamento
 - obriga a resultados equivalentes ao escalonamento serializado
 - note-se que podem existir sequências distintas com resultados distintos...
 - mais eficiente
 - exemplo: enquanto uma transação faz uma operação de I/O (lenta) outras transações podem ser executadas
- Keyword: Evitar estados de não integridade.

16

 deti

Escalonamento Concorrente

- Nem todas as execuções concorrentes resultam num estado de integridade.
 - i.e. não produzem resultados iguais aos que obteríamos com um escalonamento serializado
- O resultado final é um **estado inconsistente**
 - Se T1 e T2 fossem executadas em série o resultado final seria:
 - A = 45
 - B = 100
 - Em vez disso temos:
 - A = 50
 - B = 100

Estado inicial: A = 100; B= 50

T1	T2
read(A)	
A = A - 50	
	read(A)
	temp = A*0,1;
	A = A - temp
	write(A)
	read(B)
write(A)	
read(B)	
B = A + 50	
write(B)	

17

 deti

Escalonador - Scheduler

- Entidade responsável pela definição de escalonamentos concorrente de transações.
- Um determinado escalonamento E define uma ordem de execução (intercalada) das operações de várias transações.
 - A ordem das operações dentro de cada transação é preservada.
- Problemas de um escalonamento concorrente**
 - atualização perdida (lost-update)
 - leitura suja (dirty-read)
- Situações de conflito:
 - operações que pertencem a transações diferentes
 - transações acedendo ao mesmo elemento
 - pelo menos uma das operações é write

18



Problema de Atualização Perdida (lost-update)

- Uma transação T1 grava um dado que entretanto já tinha sido lido e utilizado na transação T2...

T1	T2
read(A)	
A = A - 40	
	read(A)
	A = A + 10
write(A)	
read(B)	
	write(A)
B = B + 20	
write(B)	

Diagrama de sequência de operações:

- T1: read(A) → A = A - 40 → write(A)
- T2: read(A) → A = A + 10 → write(A)
- Depois da execução de T1, o valor de A é 40.
- Depois da execução de T2, o valor de A é 50.
- O resultado final é inconsistente (50 ≠ 40 + 10).

Observações:

- msm v. l.
- if writes
- A atualização de A efetuada por T1 foi perdida!

19



Problema de Leitura Suja (dirty-read)

- T1 atualiza um elemento A e, posteriormente, outras transações leem A.
- No entanto T1 falha e as suas operações são desfeitas...

A transação T1 falha e deve repor o valor que A tinha antes de T1 iniciar.

T1	T2
read(A)	
A = A - 10	
write(A)	
	read(A)
	A = A + 20
	write(A)
read(Y)	
abort()	

Diagrama de sequência de operações:

- T1: read(A) → A = A - 10 → write(A)
- T2: read(A) → A = A + 20 → write(A)
- Depois da execução de T1, o valor de A é 10.
- Depois da execução de T2, o valor de A é 30.
- Antes de T1 ser desfeita, T2 lê o valor de A (30).
- Observação: Valo q jec' m vai sozinho

T2 leu um valor de A que mais tarde será rollbacked!

20

Métodos de Controlo de Concorrência

Três tipos principais:

- Mecanismos de locking
- Mecanismos de etiquetagem
- Métodos optimistas
- Os dois primeiros são preventivos pois o objectivo é permitir a execução concorrente de transações até onde for possível e evitar operações que provoquem interferências entre transações.
- O último é optimista porque parte do princípio que as interferências são raras:
 - Se verificar que existiram elementos comuns nas transações concorrentes, estas são rollbacked e reiniciadas.

21

Mecanismos de Etiquetagem

- Quando a transação se inicia é-lhe atribuída uma etiqueta com um número sequencial de chegada ao sistema.
- Sempre que uma transação acede a um elemento (R ou W), marca-o com a sua etiqueta.
- Situação de conflito:
 - Quando uma transação tenta aceder a um elemento cuja valor da etiqueta é superior ao seu...
 - i.e. foi acedido por uma transação que se iniciou mais tarde
 - ... a transação é desfeita e reiniciada com um novo número de etiqueta.

22



Mecanismos de Locking

- Trata-se de um mecanismo muito conhecido/utilizado.
- **Lock** é uma variável associada a determinado elemento da base de dados que, de acordo com o seu valor no momento, permite ou não ser acedido.
 - para obter o acesso (R ou W) a um elemento é necessário obter previamente o lock desse elemento.
 - locks binários: 1 - locked; 0 - unlocked
 - problema: só permitem acessos exclusivos
 - lock leitura/escrita (r/w) (r locked; w locked; unlocked)
 - apenas os acessos para escrita são exclusivos
- Os locks são libertados no fim da transação (COMMIT ou ROLLBACK)
- Obriga a implementação de regras que evitem problemas de **deadlock**
 - As transações bloqueiam-se mutuamente. Cada uma fica eternamente à espera que a outra liberte o recurso pretendido.
- SQL Server suporta vários tipos de locks

23



RECUPERAÇÃO DE FALHAS

24

Introdução

- Como qualquer sistema computacional, os SGBD estão **sujeitos** à ocorrência de **falhas**.
- Falhas **podem comprometer a integridade da BD.**
- Os SGBD devem estar **preparados** para **responder** a falhas.
 - Recuperarem automaticamente ou oferecerem ferramentas para atuar.
- Objectivo: que o estado da **BD recuperada** esteja o **mais próximo** possível do **momento** que **antecedeu** a **falha**.

25

Falhas de um SGBD

Gravidade

- **Menos Graves:** falha numa transação
- **Muito Graves:** perda total ou parcial da base de dados

Mecanismos de Recuperação

- Escalonamentos
- Backups
- Transaction logging

26

Escalonamento vs Recuperação de Falhas

- Temos diferentes categorias de escalonamentos considerando o grau de cooperação num processo de recuperação de falhas de transações:
 - Recuperáveis versus Não-recuperáveis
 - Sem aborts em cascata versus com aborts em cascata
 - Estritos versus Não-estritos

27

Escalonamento Recuperável

- Um escalonamento E diz-se recuperável se nenhuma Ti em E for concluída (committed) até que todas as outras transações que escrevem elementos lidos por Ti tenham sido concluídas.

não-recuperável	
T1	T2
read(A)	
A = A - 15	
write(A)	
	read(A)
	A = A + 35
	write(A)
	commit()
abort()	

recuperável	
T1	T2
read(A)	
A = A - 20	
write(A)	
	read(A)
	A = A + 10
	write(A)
	commit()
	commit()

28

 deti

Escalonamento sem Abort em Cascata

- Um escalonamento recuperável pode gerar aborts de transações em cascata
 - Não desejável: maior complexidade (e tempo) na recuperação da falha
- Um escalonamento E é recuperável e evita aborts em cascata se uma Ti em E só puder ler elementos que tenham sido atualizados por transações que já concluíram.

recuperável com aborts em cascata	
T1	T2
read(X)	
read(A)	
A = A - 15	
write(A)	
	read(A)
	A = A + 35
	write(A)
abort()	...

recuperável sem aborts em cascata	
T1	T2
read(X)	
A = A - 15	
write(A)	
commit()	
	read(A)
	A = A + 35
	write(A)
	...

29

 deti

Escalonamento Estrito

- Um escalonamento E é recuperável, evita aborts em cascata e é estrito se uma Ti em E só puder ler ou atualizar um elemento A depois que todas as transações que atualizaram A tenham sido concluídas.

recuperável sem aborts em cascata e não estrito	
T1	T2
read(A)	
A = A - 15	
write(A)	
	read(B)
	A = B + 35
	write(A)
	commit()
abort()	

recuperável sem aborts em cascata e estrito	
T1	T2
read(A)	
A = A - 15	
write(A)	
commit()	
	read(B)
	A = B + 35
	write(A)
	commit()

30

Backups

- Cópias de segurança efectuadas com regularidade que devem contemplar toda a base de dados.
- Ponto de recuperação caso existam falhas muito graves no sistema.
- Desvantagem: só permite recuperar dados até ao momento em que foi efectuado o backup.
 - Logo, devemos fazer backup com regularidade.
 - No entanto, as operações de backup são processos pesados e onerosas em termos de recursos.

31

Transactions Logs - 1/2

- Um sistema de log que regista todas as operações realizadas nas transações da base de dados, incluído o commit
- O registo de log também se reparte entre a memória e o disco
- Os logs também guardam uma imagem dos dados alterados:
 - Antes da transação: before-image
 - Depois da transação: after-image

32

Transactions Logs - 2/2

Data Flow

1. O log regista de forma sequencial todas as operações da transação, incluindo o commit
2. Só no final do registo do commit no log, os dados podem ser guardados em disco
 - gestão de I/O - dados são alterados em memória volátil (buffers) e só mais tarde efectivados em disco.
3. Antes dos dados da BD serem escritos em disco, os respetivos dados do log têm de ser escritos

33

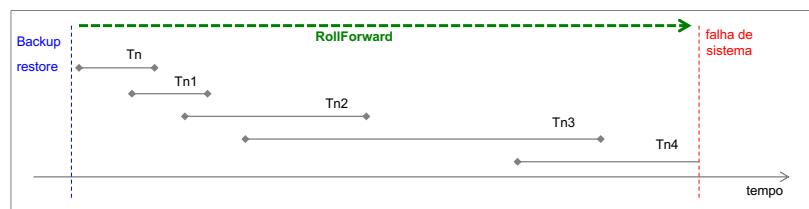
Transaction Logs - Recuperação de Falhas

- Como referido, podemos ter várias operações sobre dados efectuadas em memória volátil (buffers) que podem não ser guardadas em disco caso ocorra uma falha no sistema.
- No entanto, o registo de logs já está em disco, pelo que pode ser utilizado para recuperação da falha.
- Os backups + transaction logs podem ser utilizados para recuperação de diferentes tipos de falhas:
 - Disco
 - Transação
 - Sistema

34

Recuperação de Falha de Disco

- Existe uma falha nos discos em que está a base de dados
- Caso mais grave de falha pois obriga à reconstrução de toda a base de dados
- Processo de recuperação:
 1. Fazer o restore do último backup
 2. Fazer o rollforward
 - Utilizar as after images do transaction log para atualizar a base de dados até ao momento da falha



Apenas Tn4 não é recuperada.

35

Recuperação de Falha de Transação

- Menos Grave
- Basta utilizar a before-image do transaction-logging capturada antes da transação para fazer rollback

36

Recuperação de Falha de Sistema

- Erros no sistema operativo ou no SGBD
- Nestas condições considera-se que a base de dados está corrompida e é necessário regressar a um estado anterior válido (integro) utilizando:
 1. Rollback com as before-images do transaction-logging
 2. Rollforward com as after-images do transaction-logging
- Dificuldade
 - Detectar o ponto de integridade até ao qual devemos desfazer as transações.

37

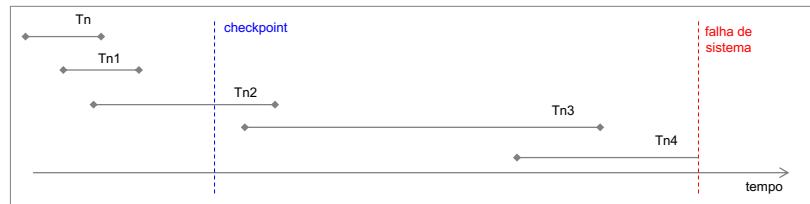
Rollback - até que ponto?

- Quando necessitamos de fazer rollback a questão que se coloca é:
 - Até que ponto do transaction log devemos recuar?
 - Deverá ser o momento em que o transaction log e a base de dados estão sincronizados.
 - Só a partir desse ponto é que nos interessa refazer as transações.
- Solução segura: último backup!
 - Operação lenta ...
 - ... pois pode ser um momento muito recuado o que obrigará a um grande esforço pois temos de refazer todos as transação até ao momento da falha!!!
- Solução baseada em Chekpoint
 - Marca no transaction log que identifica o momento em que os buffers são escritos para disco.
 - Ponto de sincronismo (em disco) entre o transaction log e a BD

38

Checkpoint

- São fundamentais para limitar a amplitude dos processos de rollback e rollforward.



- Tn4 não é recuperável
- Tn2 e Tn3 são refeitas: primeiro são desfeitas (rollbacked) e depois executadas novamente (rollforward)
- Tn e Tn1 não necessitam de intervenção

39

Savepoint

- Alguns sistemas suportam Savepoint numa transação.
 - Permite reconstruir a transação até esses pontos
- Savepoint versus Commit
 - Savepoint é interno à transação
 - Commit efetiva (BD) as operações e torna-as visíveis para outras

```
BEGIN TRANSACTION
...
Save Point X
...
Save Point Y
...
END TRANSACTION
```

40

Recuperação de Falhas - Custos

- Os mecanismos de recuperação de falhas têm custos:
 - Maior número de acessos ao disco
 - Ficheiros de recuperação constantemente atualizados - transactions logs
 - Maior volume de dados armazenados
 - Redundância de dados (backup e transaction logs)
 - Sobrecarga de processamento
 - Utilização de CPU (menos significativo)

41

SQL SERVER

42

 deti

Resumo da Sintaxe

Iniciar transação

```
BEGIN TRAN[SACTION] [<trans_name> | <@trans_name_variable>]
```

Commit da transação

```
COMMIT TRAN[SACTION] [<trans_name> | <@trans_name_variable>]
```

Rollback da transação

```
ROLLBACK TRAN[SACTION]
[<trans_name> | <@trans_name_variable> | <save point
name> | <@savepoint variable>]
```

Save Point

```
SAVE TRAN[SACTION] [<savepoint name> | <@savepoint variable>]
```

43

 deti

Transações em SQL Server - Isolamento

Instrução:

[SET TRANSACTION](#)

Nível de isolamento

- **ISOLATION LEVEL** *nível*
- *nível* que uma transação T_i pode assumir:
 - **SERIALIZABLE** (T_i executa com completo isolamento)
 - **REPEATABLE READ** (T_i só lê dados efetivados (committed) e outras transações não podem modificar dados lidos por T_i)
 - **READ COMMITTED** (T_i só lê dados efetivados, mas outras transações podem modificar dados lidos por T_i)*
 - **READ UNCOMMITTED** (T_i pode ler dados que ainda não sofreram efetivação)
 - **SNAPSHOT** (T_i vê uma imagem dos dados que existiam antes de se iniciar a transação - alterações committed entretanto não são visíveis)

```
-- Exemplo em SQL Server
SET TRANSACTION ISOLATION LEVEL REPEATABLE
READ;
GO
BEGIN TRANSACTION;
.....
.....
COMMIT TRANSACTION;
```

44

* defeito em SQL Server

Exemplos

```
-- Transação cujo nome é uma variável
DECLARE @TranName VARCHAR(20)
SELECT @TranName = 'MyTransaction'

BEGIN TRANSACTION @TranName
UPDATE roysched
SET royalty = royalty * 1.10
WHERE title_id LIKE 'Pc%'

....
.

COMMIT TRANSACTION MyTransaction
GO
```

```
-- Transação com Save Point and Rollback
BEGIN TRAN
PRINT 'First Transaction: ' + CONVERT(VARCHAR,@@TRANCOUNT)

INSERT INTO People VALUES ('Tom')

SAVE TRAN Savepoint1
PRINT 'Second Transaction: ' + CONVERT(VARCHAR,@@TRANCOUNT)

INSERT INTO People VALUES ('Dick')

ROLLBACK TRAN Savepoint1
PRINT 'Rollback: ' + CONVERT(VARCHAR,@@TRANCOUNT)

COMMIT TRAN
PRINT 'Complete: ' + CONVERT(VARCHAR,@@TRANCOUNT)
```

```
-- Transação com Rollback
GO
BEGIN TRAN
PRINT 'Transaction: ' + CONVERT(VARCHAR,@@TRANCOUNT)

INSERT INTO People VALUES ('Tom')
.

ROLLBACK TRAN
PRINT 'Rollback: ' + CONVERT(VARCHAR,@@TRANCOUNT)
GO
```

45

Transações Encadeadas

- Podemos ter transações dentro de transações
- **@@TRANCOUNT** - conta o número de transações ativas
- Os **Rollbacks** em transações internas revertem toda a transação até ao primeiro BEGIN TRAN
 - Devemos utilizar save points para evitar reversão total

```
-- Transação Encadeada sem Save Points
GO
BEGIN TRAN
PRINT 'First Tran: ' + CONVERT(VARCHAR,@@TRANCOUNT)

insert into dependent values('183623612', 'Luis Pinto', 'M', null, null);

BEGIN TRAN
PRINT 'Second Tran: ' + CONVERT(VARCHAR,@@TRANCOUNT)

insert into dependent values('183623612', 'Maria Pinto', 'F', null, null);

ROLLBACK TRAN
PRINT 'Rollback: ' + CONVERT(VARCHAR,@@TRANCOUNT)
GO
```

Nenhum dos inserts é efectivado

First Tran: 1	Second Tran: 2
Second Tran: 2	Rollback: 2
Rollback: 0	Complete: 0

```
-- Transação Encadeada com Save Point
GO
BEGIN TRAN
PRINT 'First Tran: ' + CONVERT(VARCHAR,@@TRANCOUNT)

insert into dependent values('183623612', 'Luis Pinto', 'M', null, null);

SAVE TRAN savepoint1
BEGIN TRAN
PRINT 'Second Tran: ' + CONVERT(VARCHAR,@@TRANCOUNT)

insert into dependent values('183623612', 'Maria Pinto', 'F', null, null);

ROLLBACK TRAN savepoint1
PRINT 'Rollback: ' + CONVERT(VARCHAR,@@TRANCOUNT)

COMMIT TRAN
COMMIT TRAN
PRINT 'Complete: ' + CONVERT(VARCHAR,@@TRANCOUNT)
GO
```

Só o primeiro insert é efectivado

Stored Procedures e Rollbacks

- À semelhança do que acontece com as transações encadeadas, um rollback num stored procedure (SP) reverte as suas operações mas também as exteriores ao SP!
- Devemos utilizar save points

```
-- Stored Procedure com Save Point
CREATE PROC MyProc
AS
BEGIN
    BEGIN TRAN
    SAVE TRAN Savepoint1
    ...
    ROLLBACK TRAN Savepoint1
    COMMIT TRAN
END
```

```
-- Invocação do Stored Procedure
GO
BEGIN TRAN
EXEC MyProc
COMMIT TRAN
```

47

Transações - SET XACT_ABORT ON|OFF

- Por defeito, quando ocorre um erro a transação é toda desfeita e as instruções seguintes não são executadas
 - tudo ou nada...
- No entanto temos possibilidade de permitir que a transação continue mesmo com erro!

```
-- Transacção com SET XACT_ABORT OFF
CREATE TRIGGER deleteCustomer ON customers
instead of delete
AS
BEGIN
    BEGIN TRAN
    DECLARE @id as int;
    SELECT @id=customerID from deleted;

    IF (NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES
        WHERE TABLE_SCHEMA = 'dbo' AND TABLE_NAME = 'customers_deleted'))
        CREATE TABLE dbo.customers_deleted (...);

    -- SET XACT_ABORT OFF
    INSERT into dbo.customers_deleted select * from deleted; -- *
    DELETE from customers where customerID =@id; -- **

    IF (@@error > 0)
        raiserror ('Delete Error', 16, 1); -- **

    COMMIT TRAN
END
```

A ocorrência de um erro no delete * leva a um rollback de toda transacção, incluindo a possível criação da tabela customers_deleted

SET XACT_ABORT OFF
A ocorrência de um erro no delete * não desfaz operações anteriores e faz display da msg de erro **

48

Transações - Try ... Catch

- Quando há um erro dentro do bloco Try, as operações anteriores são desfeitas e “salta” para o bloco Catch. Depois do End Catch são executadas as restantes instruções.

```
-- Transação com Try ... Catch
CREATE TRIGGER deleteCustomer ON customers
instead of delete
AS
BEGIN
    BEGIN TRAN
    DECLARE @id as int;
    SELECT @id=custID from deleted;

    IF (NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES
                     WHERE TABLE_SCHEMA = 'dbo' AND TABLE_NAME = 'customers_deleted'))
        CREATE TABLE dbo.customers_deleted (...);

    BEGIN TRY
        INSERT into dbo.customers_deleted select * from deleted; -- *
        DELETE from customers where custID =@id;
    END TRY
    BEGIN CATCH
        raiserror ('Delete Error', 16, 1); -- **
        Print 'Cheguei aqui...'; -- ***
        COMMIT TRAN
    END CATCH
END
```

A ocorrência de um erro no delete * leva a um rollback de todas as operações anteriores da transação.
No entanto, as instruções ** e *** são executadas

49

Resumo

- Transações
- Controlo de Concorrência
- Recuperação de Falhas
- Ambiente SQL Server

50

