

Complementos de Bases de Dados – Armazenamento –

Cláudio Miguel Sapateiro
claudio.sapateiro@estsetubal.ips.pt

Sumário

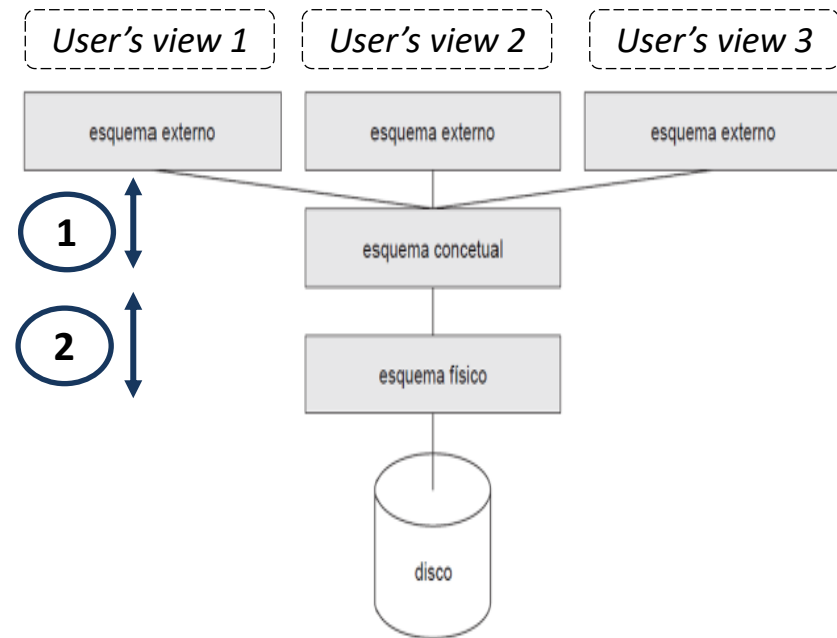
- BD e ficheiros
- Tipos de armazenamento
- RAID
- Estrutura de ficheiros
- *MS SQL files & filegroups*
- Notas Finais

BD

- Porquê?
- Como se fazia antes?
- Exemplos

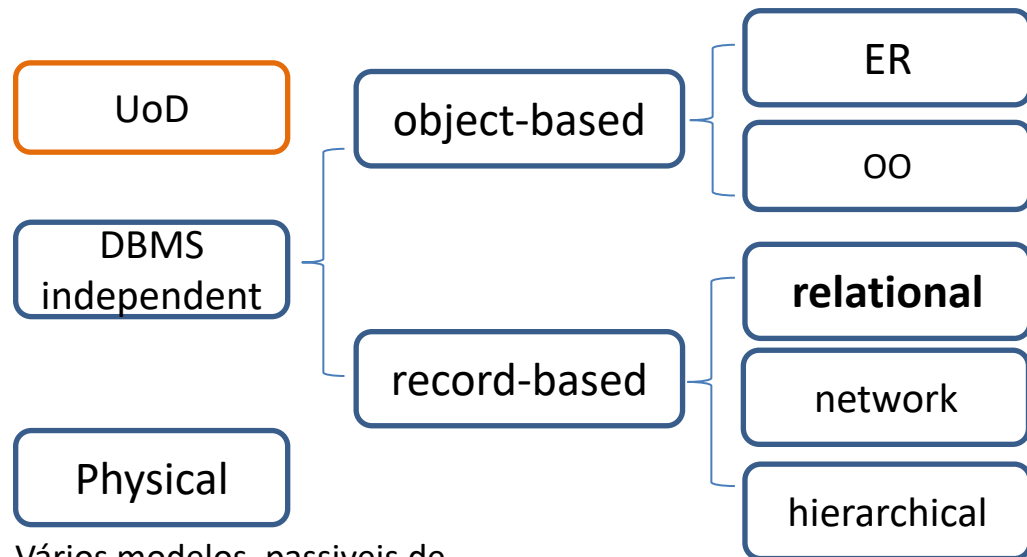
Níveis (de abstração) e Modelos

ANSI-SPARC “Reference” Architecture



1. Logical independence
2. Physical independence

Data Models



Vários modelos, passíveis de diferentes níveis de detalhe.
Exemplo de concretização:

```
struct STAFF {  
    int staffNo;  
    int branchNo;  
    char fName [15];  
    char lName [15];  
    struct date dateOfBirth;  
    float salary;  
    struct STAFF *next;  
};  
index staffNo; index branchNo;
```

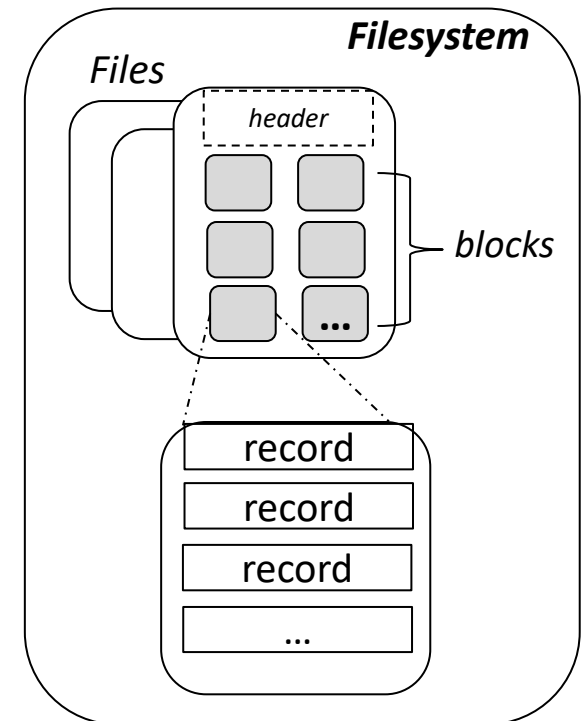
Níveis (de abstração) e Modelos

Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

BD e Ficheiros

Estrutura dos Ficheiros

- Uma BD é mapeada num conjunto de ficheiros persistidos em disco sobre o *filesystem* do SO
- Um ficheiro:
 - Constituído por conjunto de blocos (*blocks/pages*)
 - e.g. 4 a 8 KB (mas pode ser redefinido)
 - Um bloco conterá vários registos (em aplicações específicas um registo poderá estender-se por vários blocos, e.g. imagem)



Tipos de Armazenamento

Características

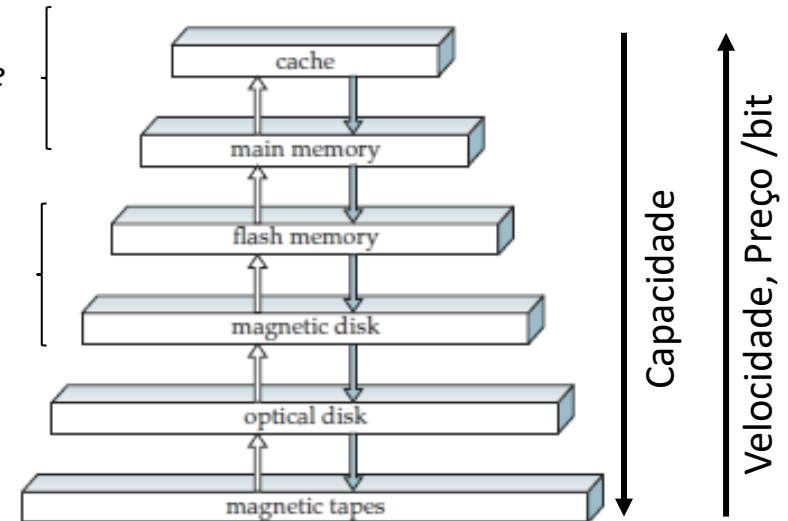
- Capacidade (e disponibilidade)
- Velocidade de acesso
- Preço (por unidade de dados)
- Fiabilidade

➤ Tipicamente temos os níveis:

- **Cache:** rápida, de reduzida capacidade, gerida pelo SO
- **Memoria RAM:** para dados em operação, capacidade média/baixa, muito volátil
- **Flash/SSD:** acesso rápido, capacidade média alta, não voláteis, caros melhora desempenho num nível adicional de cache
- **Discos magnéticos:** Na grande maioria ainda os mais utilizados (devido ao custo) grande capacidade, performance *qb* (dependendo de outros fatores de otimização),
 - Suporta a persistência permanente das alterações à BD;
 - Embora suscetíveis de falha existem precauções possíveis

Primary storage

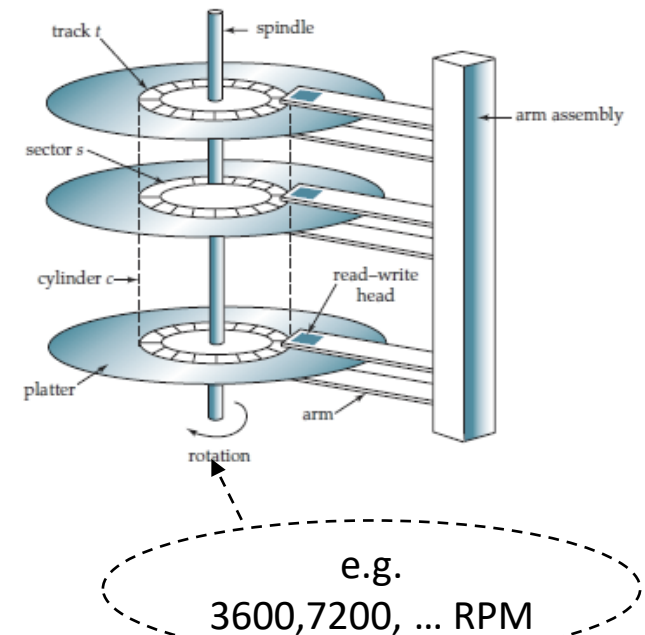
*Secondary /
online storage*



Discos Magnéticos

Características

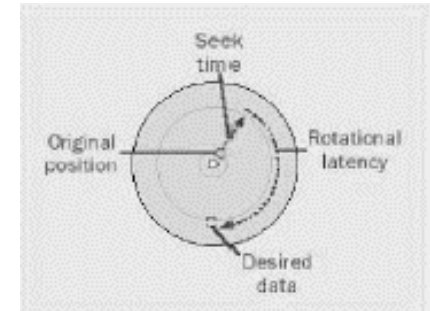
- Ainda o modo mais comum/dissemidado de persistência
- *Prato*: duas faces
- *Sector*: ~ 512 bytes
- *Pistas* (Tracks): ~ 50,000 to 100,000 / prato
 - + Interiores: ~500 a 1000 sectores
 - + Exteriores: ~ 1000 a 2000 sectores
- *Conjunto*: de 1 a 5 pratos
- *Cilindro*: considerando o movimento solidário de todas as cabeças/braços!
- Só uma cabeça está ativa em cada instante
- Controlador/Interfaces (velocidades) : SATA, SATA II, SATA 3; SCSI; ...



Discos Magnéticos

Medidas de Desempenho

- **Velocidade de rotação** (*Rotational Speed*):
número de rotações por minuto
- **Tempo de Busca** (*Seek Time*):
tempo necessário para deslocar a cabeça de leitura para o cilindro pretendido
- **Atraso de Rotação** (*Rotational Latency/ average latency time*):
tempo necessário para o disco na sua rotação, posicionar o sector pretendido, de forma a ser lido pela cabeça de leitura.
- **Track-to-Track Seeks**: tempo necessário para mover a cabeça de uma pista para a adjacente (factor relevante na leitura sequencial) [e.g. 0.6 ms (read); 0.9 ms (write)]
- **Average Seek Time**: tempo que em média as cabeças levam a pesquisar informação em pistas aleatórias (factor relevante na leitura aleatória) [e.g. 5.2 ms (read); 6 ms (write)]
- **Access time**: intervalo de tempo entre pedido de leitura/escrita e início da transferência de dados
- **MTTF**: tempo médio entre falhas, medida de fiabilidade dos disco



Discos Magnéticos

Técnicas de recolha dos dados

- Persistidos em disco segundo uma estrutura de blocos (*blocks/pages*)
- ✓ *Buffering*
- ✓ *Read-ahead*
- ✓ *Scheduling* (e.g. elevator)
- ✓ *File system frag/defrag techniques*
- ✓ *Non-volátil memory (intermediary) buffers*
- ✓ *Log disk w/ sequential read/write operations, pre definitive persistence*

RAID

RAID

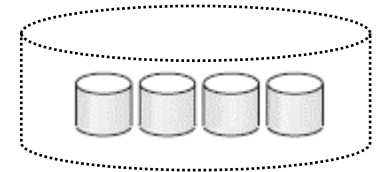
Redundant Array of Independent Disks

- Método de combinar vários discos rígidos de forma a aumentar capacidade e performance, bem como, originar redundância de dados, prevenindo assim falhas do disco rígido.
- > **Redundância** = Fiabilidade (e.g. *mirroring/shadowing*)
 - Não será completamente eficiente
 - Pode também não ser completamente eficaz!
E se falharem os 2 (ou n – e.g. catástrofe natural)
 - Não pode haver a perfeita assunção de que as falhas dos discos são independentes
- > **Paralelismo** = melhoria de Performance (*striping* data, olha para o conjunto como se fosse um disco e blocos são escritos dispersos por vários discos)

RAID

RAID 0

- usa a técnica de *data striping* (segmentação de dados)
 - várias unidades de disco rígido são combinadas para formar um volume grande
 - pode ler e gravar mais rapidamente do que uma configuração não-RAID, visto que segmenta os dados e acede a todos os discos em paralelo
-
- ❖ não permite redundância de dados
 - ❖ necessita de pelo menos 2 unidades de disco rígido.



RAID

RAID 1

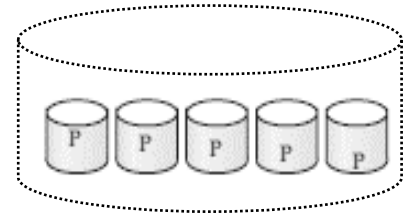
- cria um espelho (*mirror*) do conteúdo de uma unidade, noutra unidade do mesmo tamanho
- a réplica fornece integridade de dados otimizada e acesso imediato aos dados se uma das unidades falhar
- ❖ requer um mínimo de duas unidades de disco rígido e deve consistir em um número par de unidades.



RAID

RAID 5

- proporciona o equilíbrio entre redundância de dados e capacidade de disco
- segmenta todos os discos disponíveis num grande volume
- o espaço equivalente a uma das unidades de disco rígido será utilizado para armazenar bits de paridade
- Se uma unidade de disco rígido falhar, os dados serão recriados através dos bits de paridade.



P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

RAID

RAID 10 (1+0)

- é a combinação de discos espelhados (RAID-1) com a segmentação de dados (*data stripping*) (RAID-0).
- oferece as vantagens da transferência de dados rápida da segmentação de dados, e as características de acessibilidade dos arranjos espelhados
- ❖ A performance do sistema durante a reconstrução de um disco é também melhor que nos arranjos baseados em paridade

RAID

Fatores de Decisão

- Custo de discos extra
- Requisitos de performance
- Requisitos de performance em caso de falha (operacional e de reconstrução)
- Implementação: Software vs Hardware (fiabilidade e desempenho)

Notas

Recomendações RAID e.g.

© SQLServerCurry.com

	Read I/O (small ops)	Read I/O (large ops)	Write I/O (small ops)	Write I/O (large ops)	Fault Tolerance
RAID 0	Good	Very Good	Very Good	Very Good	No
RAID 5	Good	Very Good	Poor	Very Good	Yes
RAID 10	Very Good	Good	Good	Good	Yes

Assumption: These RAIDS have same number of drives

- Raid 1
 - Sistema Operativo
 - *Transaction Log* (devido à boa performance em escrita)
- Raid 5
 - Tabelas de Dados com acesso essencialmente em modo de leitura
 - Pois tem menos *storage overhead*, mas ... maior *overhead* de escrita (paridade)
 - BDs Temporária
 - Backups
- Raid 10
 - Tabelas de dados com acesso frequente em modo de Escrita
 - BDs Temporária (com requisitos de maior performance)

mini Sumário

1. Como são guardados os dados da BD no disco?
2. O que distingue armazenamento primário, secundário e terciário?
3. Que fatores principais devem ser considerados na seleção de uma arquitetura RAID?

10:00

Exercícios



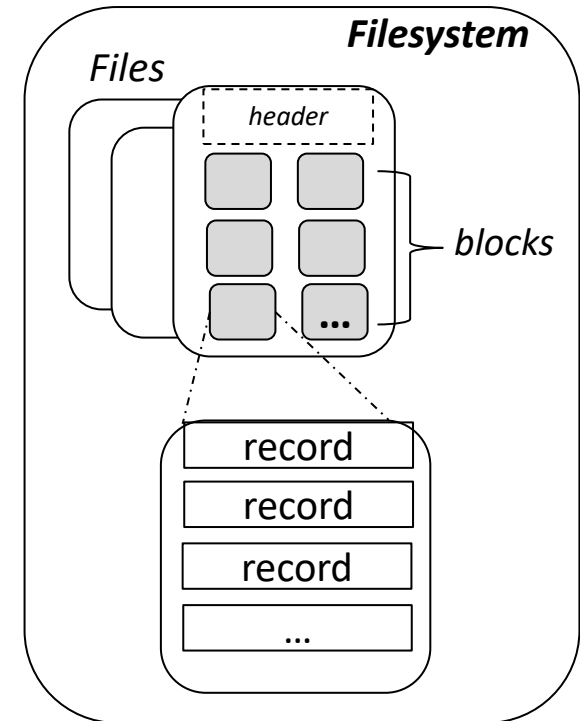
1. Se o objetivo for performance em escrita devo utilizar RAID 0 ou RAID 5?
2. Se o objetivo for maior redundância devo utilizar RAID 0 ou RAID 1?
3. Se o objetivo for maior desempenho na necessidade de recuperação dos dados devo utilizar RAID 1 ou RAID 5?
4. Considere a figura:
 - Estão representados 4 discos
 - B_i representa um bloco de dados
 - P_i representa o bloco de paridade para os blocos B_{4i-3} a B_{4i}

➤ Existe alguma situação em que esta organização se revele problemática?

Disk 1	Disk 2	Disk 3	Disk 4
B_1	B_2	B_3	B_4
P_1	B_5	B_6	B_7
B_8	P_2	B_9	B_{10}
\vdots	\vdots	\vdots	\vdots

Estrutura de Ficheiros

- Uma BD é mapeada num conjunto de ficheiros persistidos em disco sobre o *filesystem* do SO
- Um ficheiro:
 - Constituído por conjunto de blocos (*blocks*)
 - e.g. 4 a 8 KB, mas pode ser redefinido
 - Um bloco conterá vários registos (em aplicações específicas um registo poderá estender-se por vários blocos, e.g. imagem)
- **Registos nos ficheiros**
 - Fixed-length records
 - Variable-length records



Estrutura de Ficheiros

Fixed-Legth Records

Type instructor = record

ID char (5)

name char (20)

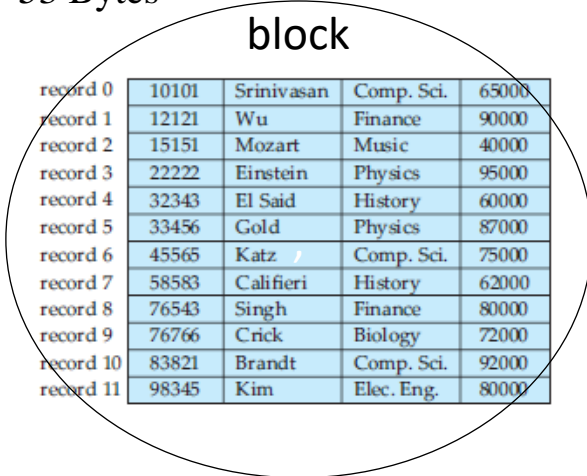
dept_name char (20)

salary numeric (8)

end

Total = 53 Bytes

block



record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

**Quantos registos cabem
num bloco de 8KB?**

$\#records = \text{int} (blockSize / recordSize)$

Estrutura de Ficheiros

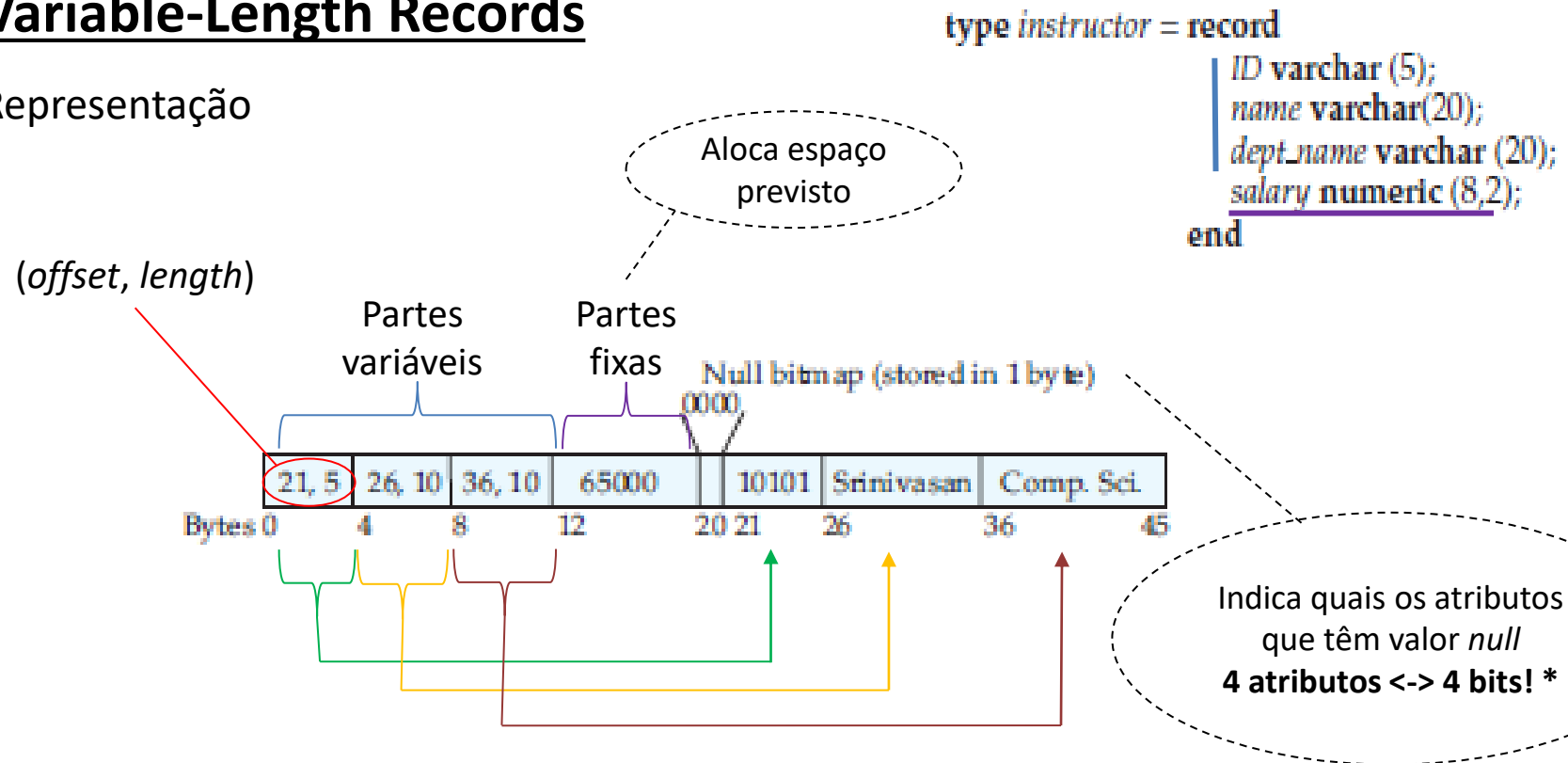
Variable-Length Records

- Armazenamento de múltiplos registos que:
 - contêm campos de tipos com “comprimento variável”
 - ou, oriundos de “entidades tipo” diferentes
- Como representar estes registos? (e armazenar nos blocos?)
 - De forma a que os valores dos seus campos sejam acedidos/manipulados com facilidade

Estrutura de Ficheiros

Variable-Length Records

Representação



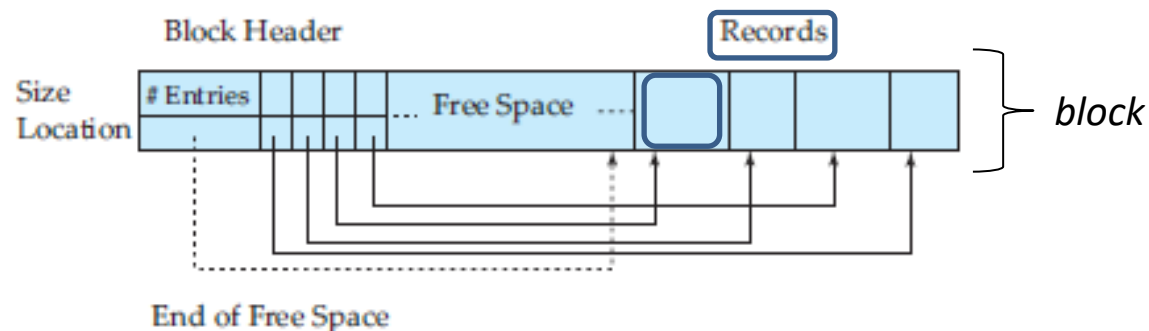
* Existem outras maneiras e.g. colocar *null bitmap* no início e nos campos nulos não indicar par (offset, length), salva espaço à custa de mais complexidade na manipulação/extração dos atributos

Estrutura de Ficheiros

Variable-Length Records

Armazenamento

- Como armazenar então nos blocos registos de comprimento variável?
- Normalmente existe um *header* no início de cada *block*, com:
 1. O numero de registos persistidos,
 2. A indicação do final do espaço livre no bloco;
 3. Um *array* com a indicação da localização e tamanho de cada registo



Estrutura de Ficheiros

Organização dos Registos em Ficheiros


- Vimos como são representados os registos
- Como se podem então organizar os registos pelos ficheiros?
- *Alternativas:*
 - *Heap file organization*
um registo pode ser colocado em qualquer local do ficheiro em que exista o espaço. Não há nenhuma ordenação. Tipicamente um ficheiro por cada “relação” (MR)
 - *Sequential file organization*
os registos são guardados sequencialmente segundo a “chave de pesquisa”
 - *Hashing file organization*
a localização é função do calculo de uma função *hash* que mapeia o registo no seu bloco dentro do ficheiro
 - *Clustering file organization*
registos de diferentes “relações” podem estar colocados num mesmo ficheiro


Estrutura de Ficheiros

Organização dos Registos em Ficheiros

Sequencial

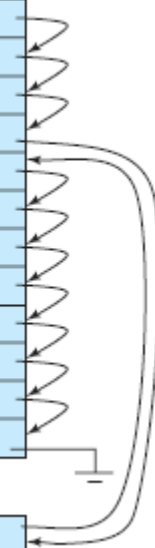
10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	




insert

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	
-------	-------	-------	-------	--



Estrutura de Ficheiros

Organização dos Registos em Ficheiros

Multi-table clustering

- Bases de dados simples e (expectavelmente) “pouco” populadas, têm tipicamente uma estrutura simplificada
 - *Tuplos* de uma relação são representados como *fixed-length records*
 - e as relações podem estar mapeadas numa estrutura de ficheiros simples:
1 relação - 1 ficheiro
- Em BDs mais exigentes (em performance e considerando dimensão e volume de dados)
 - O SGBD fará a gestão do(s) ficheiro(s)
 - dos blocos nos ficheiros
 - dos registos a persistir nos blocos

Estrutura de Ficheiros

Organização dos Registos em Ficheiros

Multi-table clustering

- Nos casos em que múltiplas relações são guardadas no mesmo ficheiro
 - Existe por vezes a preocupação de que os blocos contenham registos de uma só relação
- Contudo, pode justificar o acrescento de complexidade de juntar tuplos de relações diferentes num mesmo bloco

- Consideremos:

select dept name, building, budget, ID, name, salary
from department **join** instructor;

dept_name	building	budget
Comp. Sci.	Taylor	100000
Physics	Watson	70000

- ❖ Estes registos podem estar espalhados por diversos blocos forçando várias leituras de blocos diferentes

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

Estrutura de Ficheiros

Organização dos Registos em Ficheiros

Multi-table clustering

- Uma organização física dos registos das relações *department* e *instructor* que melhoraria a performance da anterior *join query*

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Physics	Watson	70000

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000



Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

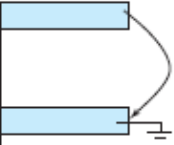
Estrutura de Ficheiros

Organização dos Registos em Ficheiros

Multi-table clustering

- Contudo poderia haver agora *queries* (até mais simples) com pior performance e.g. *SELECT * From department*
- Mesmo mantendo uma “lista” de *departments* através de apontadores,

Comp. Sci.	Taylor	100000	
45564	Katz	75000	
10101	Srinivasan	65000	
83821	Brandt	92000	
Physics	Watson	70000	
33456	Gold	87000	



- Pode requerer acesso a mais blocos pois agora os blocos contendo também *instructors* conterão necessariamente menos *departments*
 - dependerá dos dados
 - das *queries* pretendidas
 - e da performance para aquelas que possam ser mais exigentes

DBMS Buffering

Buffer Manager

- Um dos objetivos de uma implementação de um SGBD é minimizar o numero de transferências de blocos entre o disco e a memória
 - Umas das formas é manter tantos *blocks* quanto possível em memória
- O buffer é o local onde se mantêm cópias em memoria dos dados (parcela) persistidos em disco
- Buffer Management
 - Sempre que há uma chamada query/execução ao SGBD
 1. Se já existir em buffer o(s) bloco(s) respetivos este é (são) disponibilizado(s)
 2. Se o bloco não está no buffer, este terá de alocar espaço para o recolher do disco
 - i. Se necessário “livra-se” de algum(s) existente(s) (e.g. LRU, MRU, ...)
 - a. Copiando-os de volta para o disco se a versão em memória for mais atualizada
 3. Este processo “interno” é transparente para o programa que solicita o SGBD

mini Sumário

1. Que organização poderão ter os ficheiros de dados?
2. O que distingue *fixed-length* e *variable-length records*?
3. O que significa *multi-table clustering*?

10:00

Exercícios



1. O que distingue um *heap file* de um *sequential file*?
2. Num ficheiro de blocos de 8KB que armazenem registos de 100B, quantos registos é possível colocar por bloco?
3. Se tiver necessidade de armazenar 100.000 registos de dimensão unitária de 100B numa tabela persistida num ficheiro com blocos de 8KB, de quantos blocos preciso?
 - a. Qual o tamanho necessário para esse ficheiro? (pode desprezar o header)
 - b. E se este numero de registos tiver uma taxa de crescimento de 10% / ano, qual o espaço necessário a este ficheiro em 3 anos?

Estrutura de Ficheiros

MS SQL

Estrutura de Ficheiros

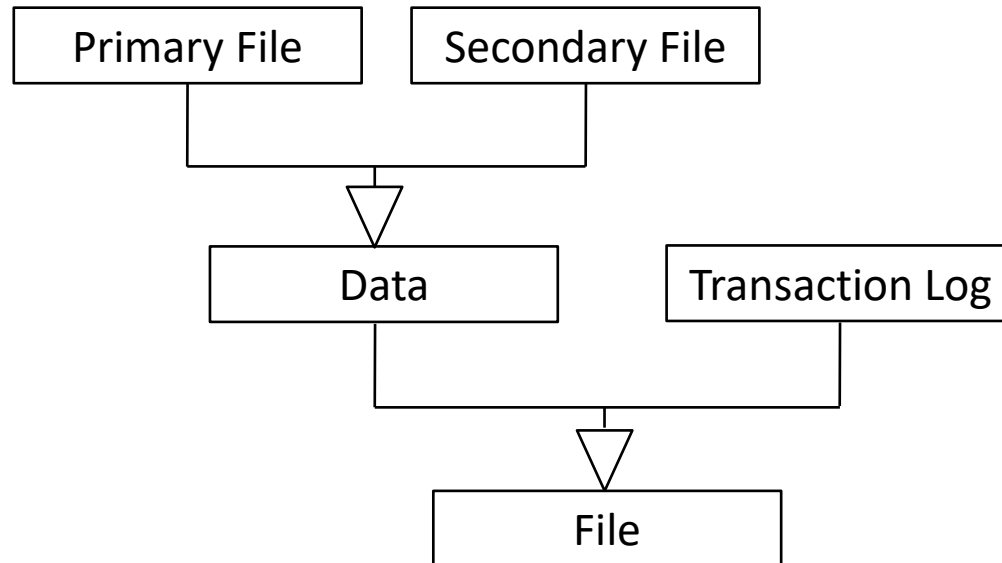
Files & Filegroups

- As bases de dados são criadas tendo como suporte um **conjunto de ficheiros** específicos
- Os ficheiros podem ser agrupados em **filegroups** de forma a facilitar a sua gestão, localização/persistência e performance no seu acesso
- Uma base de dados tem que ser criada com **pelo menos um ficheiro de dados e um ficheiro de log** (exclusivos de cada base de dados)

Estrutura de Ficheiros

Tipos de Ficheiros (da base de dados)

- **Primário** (*Primary data file*)
- **Secundário** (*Secondary data files*)
- **Log de Transacções** (*Transaction log file*)



Estrutura de Ficheiros

Ficheiro Primário

- ✓ Contém informação de inicialização da base de dados
- ✓ Agrupa tabelas (de sistema):
 - geradas/atualizadas automaticamente aquando da criação de uma nova BD
 - contêm os utilizadores, objetos e permissões de acesso (tabelas, índices, views, triggers e stored procedures, ...)
- ✓ Contém referências para os restantes ficheiros da BD
- ✓ Dados do utilizador podem ser persistidos neste ou, num ficheiro secundário
- **Existe apenas e obrigatoriamente um ficheiro primário** por base de dados
- Neste poderão coexistir metadados e dados
- Extensão do ficheiro: .mdf

Estrutura de Ficheiros

Ficheiro(s) Secundário(s)

- ✓ São opcionais
 - a base de dados pode não ter ficheiro secundário, se todos os dados estiverem armazenados num ficheiro primário
- ✓ São definidos pelo utilizador
- ✓ Podem armazenar tabelas/objetos que não tenham sido criados no ficheiro primário
- ✓ Algumas bases de dados beneficiam de múltiplos ficheiros secundários de forma a dispersar os dados por vários discos

- Permitem melhorar desempenho e escalabilidade, se/quando previsível que o *Primary data file* atinja o limite máximo de um ficheiro para o SO

- Extensão dos ficheiros: .ndf

Estrutura de Ficheiros

Log de Transações

Transaction Log File:

- ✓ Utilizado para armazenar o “rasto” de transações
- ✓ Persiste a informação necessária à recuperação da base de dados
- ✓ Todas as bases de dados têm pelo menos um log file

- Extensão dos ficheiros: .ldf

- ❖ Embora em sistemas simples a aproximação *default* é manter os *data e transaction files* na mesma *path*;
 - Em casos mais complexos/exigentes é recomendado que os ficheiros de dados e de *log* de transações sejam colocados em discos separados

Estrutura de Ficheiros

Filegroups

- Permitem aumentar a performance da base de dados ao facilitarem a distribuição dos ficheiros de dados que os constituem, por vários discos ou sistemas RAID.
- **Tipos de *Filegroups***
 - **Primário** (*Primary filegroup*)
 - **Utilizador** (*User-defined filegroup*)
 - **Por Omissão** (*Default filegroup*)
- Cada BD conterà sempre um *Primary filegroup*
 - Este contem *Primary and Secondary data files* (alocados a este grupo!)
- O utilizador pode criar *Filegroups* para agregar *data files* com objetivos p.e.:
 - gestão administrativa, alocação de espaço e localização específica

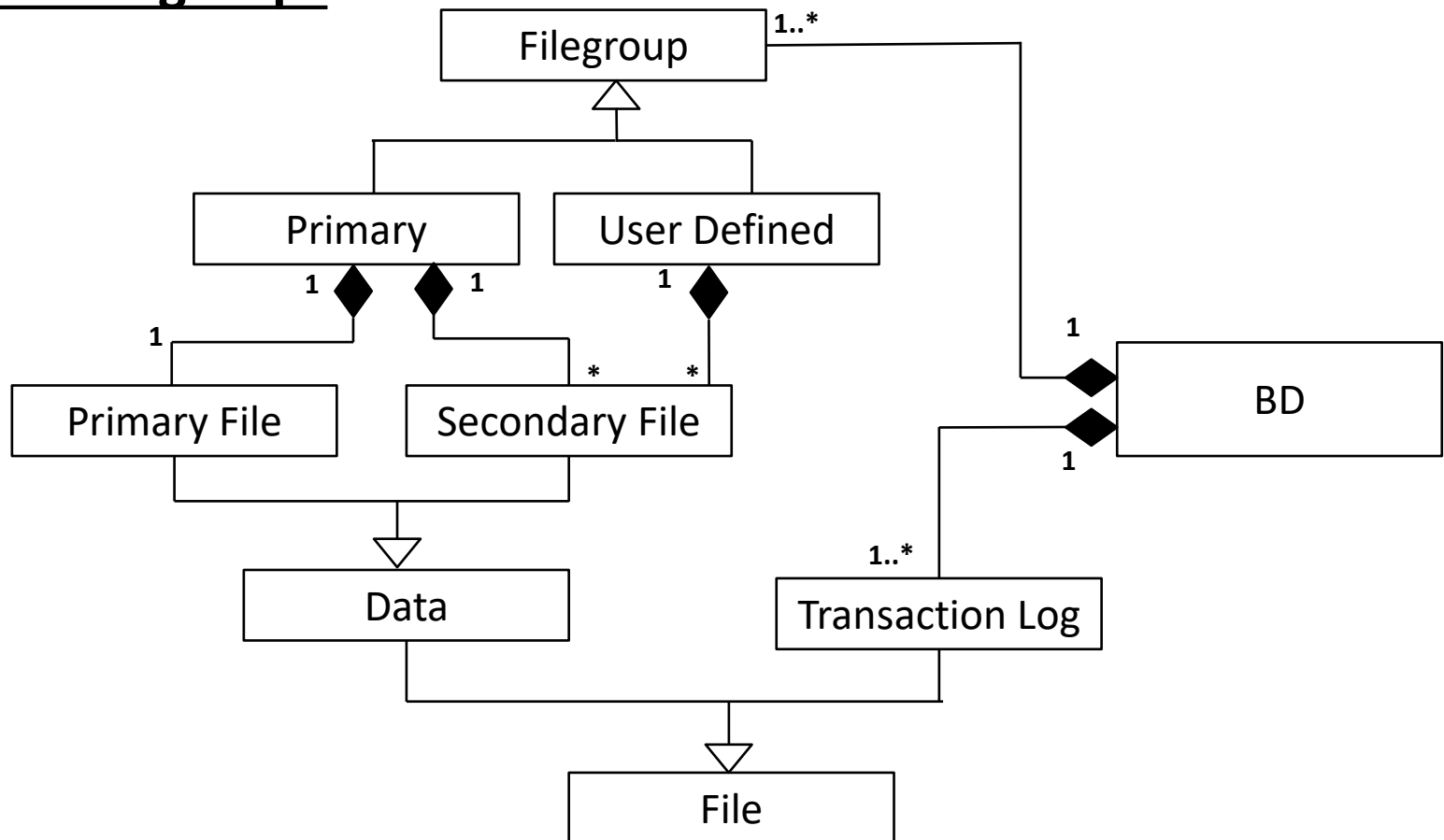
Estrutura de Ficheiros

Filegroups

- **Primário**
 - Contém o ficheiro primário e todos os ficheiros que não tenham sido explicitamente colocados noutra *filegroup*
 - Todas as tabelas de sistema estão no *Primary filegroup*
- **Utilizador**
 - Inclui qualquer *filegroup* definido pelo utilizador durante o processo de criação (ou alteração) da base de dados
 - As tabelas e índices podem ser criadas e localizadas num *filegroup* específico, definido pelo utilizador
- **Por Omissão (default)**
 - Armazena as páginas das tabelas e índices para os quais não tenha sido atribuído um *filegroup* específico
 - Apenas um *filegroup* pode ser, num dado momento, *filegroup* por omissão
 - Se não tiver sido especificado *filegroup* por omissão, é considerado como tal o *filegroup* primário
 - Utilizadores com role *db_owner* podem alterar o *filegroup* por omissão

Estrutura de Ficheiros

Files & Filegroups



Estrutura de Ficheiros

Exemplo

- Os *data files* **Data1.ndf**, **Data2.ndf** e **Data3.ndf** podem ser criados respetivamente em 3 discos diferentes e atribuídos a 1 *filegroup* e.g. **fgroup1**.
- *Queries* aos dados das tabelas persistidas nestes ficheiros terão eventualmente um melhor desempenho dado que solicitarão informação distribuídas pelos 3 discos
- ❖ Um esquema semelhante pode ser obtido com um único ficheiro mas criado sobre RAID (**R**edundant **A**rray of **I**ndependet **D**isks)

Estrutura de Ficheiros

Exemplo: Criação de uma BD com a especificação dos *data e log files*

```
USE master;
GO
CREATE DATABASE Sales
ON
( NAME = Sales_dat,
  FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\saledat.mdf',
  SIZE = 10,
  MAXSIZE = 50,
  FILEGROWTH = 5 )
LOG ON
( NAME = Sales_log,
  FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\salelog.ldf',
  SIZE = 5MB,
  MAXSIZE = 25MB,
  FILEGROWTH = 5MB ) ;
GO
```

Primary Data File
(implicit)

Como não especificado: MB

Log File

Estrutura de Ficheiros

Exemplo: Criação da BD com 3 *data files* e *log file* especificados

USE master;

GO

CREATE DATABASE Archive

ON

PRIMARY



Primary
explicito

```
(NAME = Arch1,  
FILENAME = 'D:\SalesData\archdat1.mdf',  
SIZE = 100MB,  
MAXSIZE = 200,  
FILEGROWTH = 20),  
(NAME = Arch2,  
FILENAME = 'D:\SalesData\archdat2.ndf',  
SIZE = 100MB,  
MAXSIZE = 200,  
FILEGROWTH = 20),
```

```
(NAME = Arch3,  
FILENAME = 'D:\SalesData\archdat3.ndf',  
SIZE = 100MB,  
MAXSIZE = 200,  
FILEGROWTH = 20)
```

LOG ON

```
(NAME = Archlog1,  
FILENAME = 'D:\SalesData\archlog1.ldf',  
SIZE = 100MB,  
MAXSIZE = 200,  
FILEGROWTH = 20);
```

GO

Estrutura de Ficheiros

Exemplo: Adicionar um *filegroup* e adicionar-lhe 2 ficheiros

USE master

GO

ALTER DATABASE AdventureWorks2012

ADD FILEGROUP Test1FG1;

GO

ALTER DATABASE AdventureWorks2012

ADD FILE

(NAME = **test1dat3**,

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\t1dat3.ndf',

SIZE = 5MB,

MAXSIZE = 100MB,

FILEGROWTH = 5MB

),

(NAME = **test1dat4**,

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\t1dat4.ndf',

SIZE = 5MB,

MAXSIZE = 100MB,

FILEGROWTH = 5MB

)

TO FILEGROUP Test1FG1;

GO

Estrutura de Ficheiros

- Calcular dimensões dos ficheiros
 - Parametros na criação
 - SIZE = #,
 - MAXSIZE = #,
 - FILEGROWTH = #

* # por defeito em MB

Estrutura de Ficheiros

- Obter o espaço ocupado por uma BD
 - Utilizar metadados

exec sp_spaceused 'SalesLT.Customer'

name	rows	reserved	data	index_size	unused
Customer	848	568 KB	272 KB	144 KB	152 KB

* Dimensão do bloco 8KB

Estrutura de Ficheiros

- Obter o espaço ocupado por uma BD
 - Somar espaço máximo que um registo pode ocupar

```
select name, max_length  
from sys.columns  
where object_NAME(object_id) = 'Customer'
```

	name	max_length
1	CustomerID	4
2	NameStyle	1
3	Title	16
4	FirstName	100
5	MiddleName	100
6	LastName	100
7	Suffix	20
8	CompanyName	256
9	SalesPerson	512
10	EmailAddress	100
11	Phone	50
12	PasswordHash	128
13	PasswordSalt	10
14	rowguid	16
15	ModifiedDate	8

Estrutura de Ficheiros

- Associar “objectos” a um *filegroup*

CREATE TABLE dbo.*Table1*

(TableId int NULL, TableDesc varchar(50) NULL)

ON [*UserData_FG*]

CREATE UNIQUE INDEX i1 ON dbo.*Table1*(TableDesc DESC

ON [*UserIndex_FG*]

mini Sumário

Em MS SQL:

1. Tipos de ficheiro em MS SQL
2. Tipos de *Filegroups*?
3. Quais os ficheiros e *filegroup default*?

10:00

Exercícios



Em MS SQL:

1. Quais os ficheiros obrigatórios?
2. Quais os *filegroups* obrigatórios?
3. É possível acumular dados e metadados num ficheiro?
4. Considere o cenário do slide seguinte:

5:00 Estrutura de Ficheiros

Considere o cenário e *discuta a sua utilização*:

- Criar um *filegroup*: ***UserData_FG***, consistindo em 3 ficheiros que estarão dispersos por 3 discos. Criar uma tabela na BD sobre o ***UserData_FG*** *filegroup*.

```
CREATE DATABASE [mydb]
```

```
ON PRIMARY
```

```
( NAME = mydb,
```

```
FILENAME = 'C:\mssql2005\data\mydb.mdf',
```

```
SIZE = 2048KB , FILEGROWTH = 1024KB ),
```

```
FILEGROUP [UserData_FG]
```

```
( NAME = mydb_userdata1,
```

```
FILENAME = 'D:\mssql2005\data\mydb_userdata1.ndf',
```

```
SIZE = 2048KB , FILEGROWTH = 1024KB ),
```

```
( NAME = mydb_userdata2,
```

```
FILENAME = 'E:\mssql2005\data\mydb_userdata2.ndf',
```

```
SIZE = 2048KB , FILEGROWTH = 1024KB ),
```

```
( NAME = mydb_userdata3,
```

```
FILENAME = 'F:\mssql2005\data\mydb_userdata3.ndf',
```

```
SIZE = 2048KB , FILEGROWTH = 1024KB )
```

```
LOG ON
```

```
( NAME = mydb_log,
```

```
FILENAME = 'L:\mssql2005\log\mydb_log.ldf' ,
```

```
SIZE = 1024KB , FILEGROWTH = 10%)
```

```
CREATE TABLE dbo.Table1
```

```
(TableId int NULL, TableDesc varchar(50) NULL)
```

```
ON [UserData_FG]
```

Notas Finais

Fatores a considerar no projeto/dimensionamento da BD

- A função do Sistema - *tipo de Aplicações*
 - OLTP – muitos utilizadores simultâneos; tempo de resposta limitado
 - “OLAP”/DSS – consultas complexas com tempos de resposta alargado
 - Batch – processamento intenso sem interface c/ utilizador
- Requisitos não Funcionais:
 - Performance mínima para as transações do sistema
 - Capacidade do sistema (espaço em disco, utilizadores, comunicações, ...)
 - Período(s) de Disponibilidade (*Uptime*)

Notas Finais

Fatores a considerar no projeto/dimensionamento da BD

- Uma das fases mais importantes do desenho do sistema é a determinação da localização dos ficheiros (de dados e de transações) da base de dados
- Considerar:
 - Uma tabela frequentemente acedida deve ser colocada num *filegroup* localizado num array com grande número de discos
 - O conteúdo da tabela será assim disseminado por um grande número de discos, permitindo um mais elevado número de operações paralelas de I/O
 - Se não for utilizado RAID mas houver múltiplas drives de disco, pode-se ainda utilizar filegroups
 - Pode ser colocado um ficheiro por drive de disco e por *filegroup* definido pelo utilizador.
 - Isso permite alojar cada tabela ou índice em ficheiro (ou disco) específicos, designando o filegroup no momento da criação do objecto

Notas Finais

Referências

- **Files & Filegroups + T-SQL**

[https://msdn.microsoft.com/en-us/library/ms189563\(v=sql.110\).aspx](https://msdn.microsoft.com/en-us/library/ms189563(v=sql.110).aspx)

Exercício

Suponha que é DBA numa empresa de distribuição.

- Prevê-se que a base de dados a criar terá uma taxa de crescimento de registo de 10% ao ano.
- As tabelas que irão possuir maior número de acessos em *escrita* serão as de
 - **compras** (250 bytes/registo) e
 - **vendas** (150 bytes/registo).

Cada uma dessas tabelas irá possuir respetivamente 1 e 2 milhões de registos.

- As tabelas de **clientes** (100 bytes/registo), **fornecedores** (150 bytes/registo) e **artigos** (50 bytes/registo), serão essencialmente acedidas em leitura.
 - Cada uma dessas tabelas irá possuir respetivamente 1, 0.5 e 2 milhões de registos.
- Existem muitas consultas com *join* entre a tabela de artigos e as tabelas de fornecedores e clientes.
- Suponha que tem disponível sistemas de RAID (1, 5 e 10). Pretende-se como DBA, que:
 1. Desenhe o *layout* do sistema de ficheiros da base de dados;
 2. Preveja as necessidades de armazenamento ao fim de 3 anos
 3. Implemente o referido *layout* no SQL Server.

Exercício

➤ Files/Filegroups (sugestão)

RAID

- | | |
|---|---------|
| 1. Primary (<i>primary file on primary filegroup</i>)
(mdf file) | RAID 1 |
| 2. Log (não pertence a nenhum <i>filegroup</i>)
(ldf file) | RAID 1 |
| 3. UFG1: Compras + Vendas
(ndf file) | RAID 10 |
| 4. UFG2: Clientes + Fornecedores + Artigos
(ndf file) | RAID 5 |

Exercício

- Previsão das necessidades de armazenamento
 - Blocos = 8KB (cabeçalho ocupa 100B)

Tabela	Nº Registos	Dimensão/ Registo	Registo/ Pagina	Nº de páginas	MB atuais	Taxa Cresc.	MB em 3 anos
compras	10^6	250	31	32258	258	133%	344
vendas	$2 \cdot 10^6$	150					

Complementos de Bases de Dados – Armazenamento –

Engenharia Informática
2º Ano / 1º Semestre

Cláudio Miguel Sapateiro
claudio.sapateiro@estsetubal.ips.pt