Sistemas de Bases de Dados

1.º teste (com consulta limitada: 1 folha identificada) - Duração: 2 horas

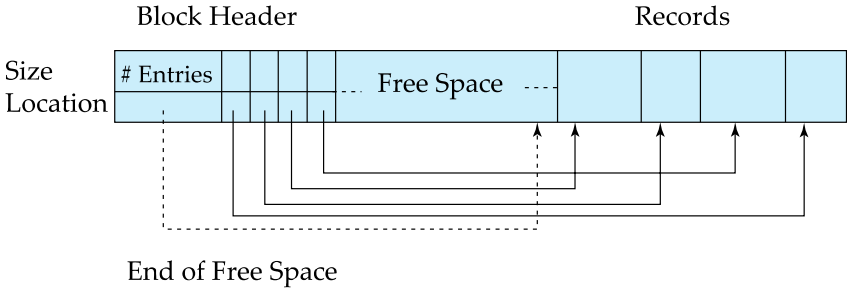
N.º:		**Nome:**	

Grupo 1 (10 valores: 1,4 + 1,4 + 1,8 + 1,8 + 1,6 + 2)

1 a) Indique 2 meios físicos de armazenamento terciário (ou *offline*).

Disco ótico (CD/DVD)	Banda magnética (*tapes*)

1 b) Considere a estrutura de *slotted page* (figura abaixo) para guardar vários registos de tamanho variável. Como sabe, os registos são mantidos contíguos no final do bloco para melhor se aproveitar o espaço livre. Assim, na remoção dum registo, este é marcado como removido na respetiva entrada do cabeçalho, e os outros registos serão encostados à direita, atualizando-se também as suas localizações nas respetivas entradas, tudo dentro do mesmo bloco. Esta deslocação dos registos pode ter implicação na atualização de apontadores para os mesmos (e.g. em índices). O que é que os SGBDs costumam fazer para evitar esse problema?



Não deve haver apontadores diretos para os registos, mas sim para a respetiva entrada no cabeçalho da *slotted page*.

1 c) Quais as vantagens e desvantagens dum índice esparso?

Vantagem: ocupa menos espaço (e a pesquisa no índice é mais rápida).
Desvantagem: obriga a uma pesquisa sequencial no ficheiro dos registos (uma vez que tem de ser um índice primário) no caso em que a chave de pesquisa não se encontra no índice.

1 d) Qual a diferença entre um índice *B+-tree* e uma organização de ficheiro de dados em *B+-tree*?

Numa organização de ficheiro, os nós folha têm os próprios registos, em vez de apontadores.

1 e) Qual a diferença entre *B+-tree* e *B-tree* relativamente a um nó não terminal? Que implicações tal pode ter na eficiência de pesquisa num índice com essa estrutura?

Os nós não terminais da *B-tree* incluem também o apontador para os respetivos registos (ou *buckets*), uma vez que as chaves de pesquisa não se podem repetir. Isso permite encontrar alguns registos mais depressa. No entanto, a altura da árvore tende a aumentar (pois há menos espaço nos nós), podendo levar a mais operações de I/O.
De facto, a percentagem de entradas encontradas ainda em nós não terminais é normalmente baixa e não compensa o aumento da altura (e a implementação para *B-trees* também é mais complexa).

1 f) Mostre o índice fruto da instrução `create bitmap index i on aluno(sexo,nota)` aplicado à seguinte instância da tabela **aluno**:

Id	1	2	3	4	5	6	7	8
sexo	M	M	F	F	M	F	M	F
nota	OK	REP	REP	OK	REP	OK	OK	OK

<M,OK> : 1000 0010
<M,REP> : 0100 1000
<F,OK> : 0001 0101
<F,REP> : 0010 0000

N.º:	

Grupo 2 (10 valores: 2+3+5)

2 a) Considere que queremos ordenar uma relação com os seguintes (33) registos (com apenas uma coluna: um inteiro) em disco: (45,34,4,78,3,14,17,52,65,35,56,9,3,23,18,8,1,33,77,88,55,44,22,11,2, 12,13,76,87,98,32,21,19). Seja o *blocking factor* f=2 e a capacidade da memória M=4 blocos. Aplicando o algoritmo de *sort-merge* (assuma que *buffer blocks* b_b=1), **mostre o conteúdo do primeiro *run*** (onde se percebam os respetivos blocos). **Diga ainda com quantos outros *runs* se juntará este no primeiro *merge pass*** (a sua resposta deve deixar claro quantos *runs* se juntam, i.e. se inclui o próprio ou não).

3	4	14	17	34	45	52	78

Este *run* juntar-se-á com mais 2. Isto é, serão juntados 3 (M-1) *runs*.

2 b) O algoritmo básico de *nested-loop join* (iteração para todos os pares de tuplos) para 2 relações *r1* e *r2* (em disco) que não cabem em memória, requer no pior caso (M=2: 1 bloco de cada relação) $n_{r1}+b_{r1}$ *seeks* e $n_{r1}*b_{r2} + b_{r1}$ transferências de blocos (onde n representa o número de tuplos, e b o número de blocos da respetiva relação).

I. Pelas expressões, qual conclui ser a relação exterior e qual a interior? Justifique também ambas as expressões (de *seeks* e de transferências).

r1 é a exterior; *r2* é a interior.

seeks: os tuplos de *r1* são carregados bloco a bloco (i.e. b_{r1} vezes), sendo necessário um *seek* de cada vez, pois entretanto a cabeça do disco deslocou-se. Para cada tuplo de *r1* será lido *r2* inteiro, sendo necessário um *seek* de cada vez (i.e. n_{r1} vezes) para o início de *r2* (depois é lido sequencialmente) - daí, $n_{r1} + b_{r1}$.
transferências: é feita uma leitura de *r2* (com b_{r2} blocos) para cada tuplo de *r1* (i.e. $n_{r1} * b_{r2}$). *r1* é transferido uma só vez (bloco a bloco, como descrito acima); ou seja, b_{r1}. Daí o total $n_{r1} * b_{r2} + b_{r1}$.

II. Se a menor relação couber em memória (melhor caso), bastam 2 *seeks* e $b_{r1}+b_{r2}$ transferências de blocos. Justifique.

Seja *r2* a menor. É necessário 1 seek e b_{r2} transferências para a colocar em memória. *r1* é lido sequencialmente (1 *seek*), bloco a bloco (b_{r1} transferências).

2 c) Considere a tabela *temp*(*id*, *city*, *ano*, *mês*, *dia*, *hora*, *val*) de valores de temperatura, criada em SQL da seguinte forma:

```
create table temp(
      id INTEGER NOT NULL primary key,
      City INTEGER NOT NULL references cidades(id),
      Ano INTEGER NOT NULL,
      Mes INTEGER NOT NULL check(mes between 1 and 12),
      Dia INTEGER NOT NULL check(dia between 1 and 31),
      Hora INTEGER NOT NULL check(hora between 0 and 23),
      Val INTEGER NOT NULL,
      unique(cidade,ano,mes,dia,hora));
```

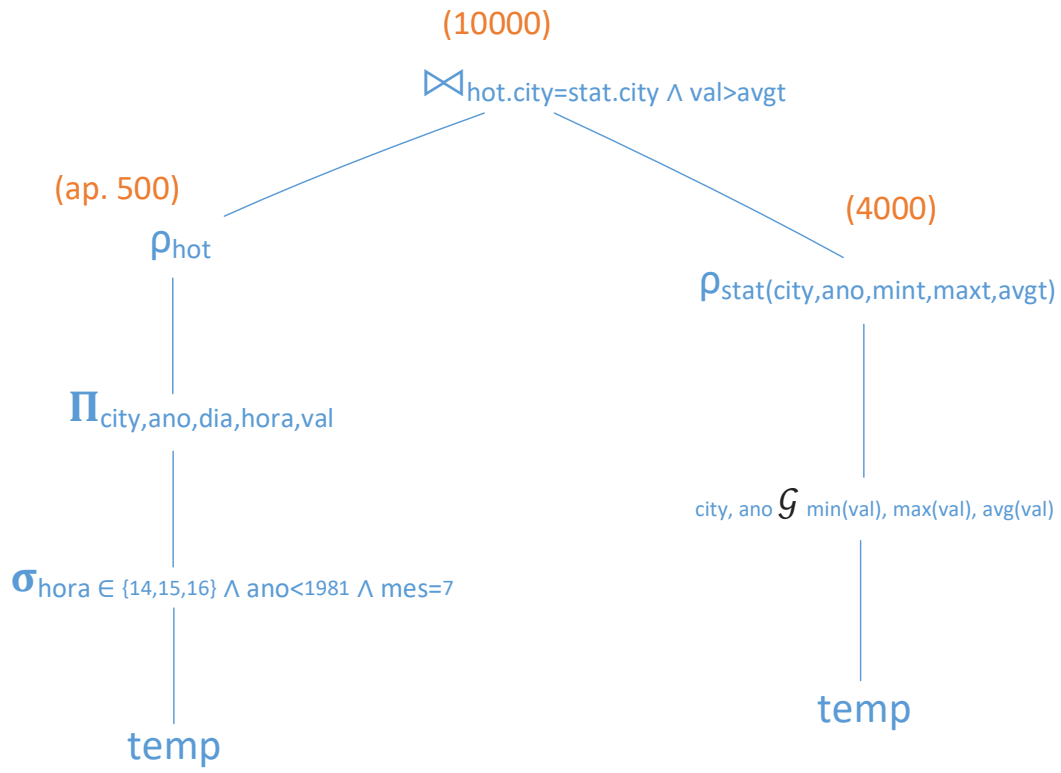
Foram registados um milhão (10^6) de valores de temperatura em 100 cidades europeias entre os anos 1979 e 2018, inclusive (há várias medições em falta, no espaço e no tempo, por diferentes motivos). Considere que neste sistema um inteiro ocupa 10 bytes, um real ocupa 40 bytes, e que são usados blocos de 10000 bytes (i.e. 10^4 bytes).
Imagine a seguinte consulta SQL:

```
with stat as
 (select city, ano, min(val) as mint, max(val) as maxt, avg(val) as avgt
  from temp group by city, ano),
 hot as (select city, ano, dia, hora, val from temp
            where hora in (14,15,16) and ano<1981 and mes=7)
(select * from hot inner join stat
   on (hot.city=stat.city and val > avgt);
```

I. Desenhe a árvore da expressão em álgebra relacional correspondente à conversão natural desta consulta, e indique para cada nó o respetivo valor estimado de tuplos.

n(stat) = 100*40 = 4000
n(hot) = 10^6/8/20/12 = 50000/8/12 = 50000/96 > 50000/100 = 500

n(join) = n(hot)*40/2 = 10000



N.º:	

II. Se for aplicado o algoritmo *block nested-loop join* sem recorrer a índices, com M=10, qual deverá ser a relação interior? Porquê?

b_{stat} = 4000*80/10000 = 32
b_{hot} = ⌈500*50/10000⌉ = ⌈2,5⌉ = 3

A relação *hot* cabe em memória; deve ficar como interior (ficando *stat* como exterior). Corresponde ao melhor caso, com custo de 2 *seeks* e $b_{stat}+b_{hot}$ transferências de blocos, pois não é preciso ler *hot* em cada iteração (basta ler uma vez para memória).

Nota: Se não coubesse nenhuma em memória, seria melhor pôr a mais pequena como exterior, pois o número de *seeks* apenas depende da dimensão da exterior e, adicionalmente, esta também aumenta o número de transferências. De facto, no pior caso são necessárias $b_r*b_s+b_r$ transferências de blocos e $2*b_r$ *seeks* (com M=10, o custo é menor, mas o raciocínio mantém-se).

III. Sem fazer contas, discuta sobre se o *Merge-Join* seria uma opção.

O *Merge-Join* geral requer que a junção seja feita por condições de igualdade, o que não é o caso. Assim, à partida, estaria posto de parte, por não ser aplicável.
Poder-se-ia, no entanto, fazer uma variante considerando apenas a igualdade de *city* e, posteriormente (no *merge*), testar a inequação (val>avgt), uma vez que para cada *city* haverá apenas uns 40 tuplos.
(De facto, pode-se considerar que se trata duma junção sobre uma igualdade (city), sujeita posteriormente a outra condição (seleção)).
Ainda assim, ter-se-ia sempre de ordenar as relações por *city* (pelo menos), o que não compensará.

IV. Sem fazer contas, discuta sobre se o *Hash-Join* seria uma opção.

À semelhança do *Merge-Join*, também o *Hash-Join* geral requer que a junção seja feita por condições de igualdade, o que não é o caso. Assim, à partida, estaria posto de parte, por não ser aplicável.
No entanto, também aqui se poderia fazer uma variante considerando apenas a igualdade de *city*, particionando as relações com *hashing* sobre esse campo e, posteriormente (na junção de partições), testar a inequação (val>avgt), uma vez que para cada *city* haverá apenas uns 40 tuplos. Poderá então ser uma hipótese a considerar.