DI- FCT-NOVA **16 de abril de 2014**

Sistemas de Bases de Dados

2º teste, 2013/14

Duração: 1 hora + 30 minutos (sem consulta)

Grupo 1

Considere parte duma base de dados de uma cadeia de ginásios, onde se regista informação de sócios da cadeia, dos professores das várias modalidades, as aulas em que os sócios estão inscritos, e os movimentos (que podem ser de tipo entrada ou saída) dos sócios e professores nos ginásios. Essa base de dados inclui as seguintes tabelas (onde os atributos que constituem a chave primária estão sublinhados):

pessoas({BI,Nome,Idade,Sx,CatRendimentos,HabLiterarias...}) socios({BI, DataInicio})
aulas({CodAula,Modalidade,DiaSemana,Hora,NumProf,Local}) inscricoes(BI,CodAula)
movimentos(BI,Ano,Dia,Hora,Local,Tipo) professores({BI,Categoria})

Para cada uma destas tabelas existe um índice clustered de árvore B+ sobre o(s) atributo(s) da chave primária (pela ordem de atributos apresentada). Além disso são definidas na base de dados as seguintes *foreign keys* todas elas não permitindo valores nulos: de BI em professores, socios e movimentos para BI em pessoas; de NumProf em aulas, para BI em professores; de BI em inscricoes para BI em socios; de codAula em inscricoes para codAula em aulas.

Sabemos ainda que num dado momento a tabela pessoas tem 10.000 tuplos, a de professores tem 40 tuplos, a de socios 1.000 tuplos, a de aulas 500 tuplos, a de inscriçoes 20.000 tuplos e a de movimentos 2.000.000 tuplos.

Tendo em conta o sistema de gestão de bases de dados usado, tipicamente cabem num bloco 20 tuplos da tabela de pessoas ou da tabela aulas, ou 40 tuplos da tabela movimentos ou ainda 100 tuplos da tabela socios, da tabela professores ou da tabela inscricoes. ***Assuma que o sistema que suporta esta base de dados permite ter em memória apenas 100 blocos.***

Nota: Neste grupo, sempre que se solicitarem exemplos, estes devem ser **exclusivamente** sobre esta base de dados. Além disso, **todas** as respostas deverão conter uma **breve justificação**.

1 a) Apresente dois planos para execução da seguinte pergunta SQL (que retorna, ordenadas de forma crescente, as idades das sócias que entraram como sócias antes de 1 de janeiro), justificando qual deles deverá ter um menor custo na base de dados em causa.

```
select distinct Idade
from pessoas natural inner join socios
where Sx = "F" and DataInicio < 01/01/2014
order by Idade;
```

1 b) Considere que o sistema apenas implementa para junções o algoritmo de *"block nested loop join"*, eventualmente usando índices quando disponíveis.
Para cada uma das junções abaixo indique qual a melhor ordem para fazer as junções e, em cada operação, qual deverá ser a *inner table* e qual deverá ser a *outer table* no *nested loop*:

1. pessoas ⋈ socios
2. pessoas ⋈ inscricoes ⋈ movimentos

1 c) Considere agora que o sistema implementa também o *"hash join"* e *"merge join"*. Para cada uma das junções em **1b)**, qual o melhor algoritmo de junção?

1 d) Apresente um exemplo de pergunta SQL que beneficie da declaração de alguma das *foreign keys* descritas acima. I.e. uma pergunta que caso não existisse uma das chaves estrangeira acima, seria muito menos eficiente.

1 e) Qual o melhor plano para a execução da seguinte pergunta SQL (que retorna o nome dos professores que também são sócios)?

```
select Nome from pessoas natural inner join professores
where exists (select BI from socios where socios.BI = professores.BI)
```

Grupo 2

Nota: Dê respostas **breves**.

2 a) O pseudo-código abaixo (retirado do livro) descreve a 2ª parte do algoritmo *"hash-join"* (após a 1ª parte, em que se particionam as duas tabelas a juntar):

```
/* Perform join on each partition */
for i := 0 to n_h do begin
  read H_si and build an in-memory hash index on it;
  for each tuple t_r in H_ri do begin
    probe the hash index on H_si to locate all tuples t_s
      such that t_s[JoinAttrs] = t_r[JoinAttrs];
    for each matching tuple t_s in H_si do begin
      add t_r ⋈ t_s to the result;
    end
  end
end
```

Para execução do código que aparece logo na 2ª linha, é necessário assumir que é possível construir um *hash-index* para o H_{Si} que caiba em memória. Existe depois uma variante deste algoritmo, que considera *recursive partitioning* para os casos em que tal não é possível, mas na prática é muitíssimo raro ser necessário esse algoritmo mais complexo.

Justifique que de facto, na prática, esses casos são raros, e.g. mostrando, em casos concretos, a partir de que dimensões de relações é necessário recorrer ao *recursive partitioning*.

2 b) Os algoritmos para processamento de "*outer join*" são normalmente obtidos como pequenas variações dos 3 principais algoritmos para processamento de "*inner join*".

Explique brevemente como é que se pode alterar o algoritmo *merge join* por forma a que este passe a calcular *outer joins.*

2 c) Embora em geral haja vantagem em usar um optimizador de perguntas baseado em estimativas de custo de todos (ou quase) os planos de execução possíveis para a pergunta, isto nem sempre é assim. Há situações em que tal processo de optimização é incomportável, e o seu custo excede largamente os possíveis benefícios.

Diga em que condições pode até ser desvantajoso usar uma otimização da pergunta baseada em estimativas de custo (*Cost-based optimization*), dando um exemplo de base de dados e perguntas concretas em que tal se passe.