FEUP

Licenciatura em Engenharia Informática e Computação
Tecnologia de Sistemas de Gestão de Bases de Dados

2001/2002

Exame de Avaliação

28 de Junho de 2002

NOME: **Resolução do Exame (18 de Julho de 2002)** _____

Observe por favor as seguintes instruções:

- Leia cuidadosamente o exame até ao fim por forma a escolher a sua estratégia.
- O exame tem a duração máxima de duas horas e meia (150 minutos).
- O exame é com consulta de todo o material próprio trazido para o efeito.
- Deve responder nos espaços fornecidos neste exame, podendo usar, em último recurso, o espaço das costas da folha.
- O exame tem 10 perguntas, com as pontuações indicadas, totalizando 100 pontos.

Problema	1	2	3	4	5	6	7	8	9	10	Total	NOTA
Máx. Pontos	5	15	15	5	15	10	10	10	10	5	100	–
Pontos												

João Correia Lopes

1. Armazenamento de Dados: Ficheiros e Índices [5 pontos]

B-trees são estruturas de dados genéricas que podem ser usadas quando se pretende acesso eficiente aos registos correspondentes a uma dada ordenação de uma chave de procura. Estas árvores multi-caminho contêm nós com um conjunto de chaves (por exemplo k) e um conjunto de apontadores (por exemplo k+1) para outros nós da árvore.

a) Em bases de dados usam-se árvores ISAM e B+ trees para conseguir índices que melhoram o desempenho na resposta a interrogações que envolvem procura sequencial em intervalos de uma chave (de procura). Diga quais são as diferenças fundamentais entre ISAM, B-trees e B+ trees.

Resposta:
Ao contrário das B-Trees, nas árvores ISAM e B+ trees os nós intermédios não contêm páginas de dados, contendo apenas páginas de índices; as páginas de dados só se encontram nas folhas. Para além disso, nestas árvores de procura usadas em SGBDs as folhas são encadeadas numa lista duplamente ligada por forma a obter bom desempenho em interrogações que envolvam intervalos ou ordenações da chave de procura.

Árvores ISAM são estruturas estáticas em que as páginas de índice não mudam com as inserções e remoções levando a páginas de overflow nas folhas (páginas de dados) e assim degradando o desempenho. As B+ trees são estruturas dinâmicas crescendo e contraindo (por alteração dos nós intermédios de índices) tal com as B-trees não sendo portanto necessárias páginas de overflow. ISAM é preferível sempre que se esperem poucas alterações; B+ trees são quase preferíveis na prática.

2. Indexação [15 pontos]

Uma organização de ficheiros possível baseia-se na utilização de funções de dispersão (*hash*) para mapear campos de procura em *buckets* por forma a encontrar a página onde esse campo reside.

a) Apresente um exemplo de uma situação que mostre que a utilização de um índice deste tipo leva a melhor desempenho do que uma organização de ficheiro com registos por ordem aleatória (*heap file*).

Resposta:
Se a maioria das interrogações envolver igualdade baseada na chave de procura, compensa construir um índice hash neste campo. Assim, poderá ser encontrado o registo procurado em 2 acessos a disco.

b) Apresente um exemplo de uma situação que mostre que o uso de uma organização de ficheiro com registos por ordem aleatória (*heap file*) leva a melhor desempenho do que a utilização de um índice usando *hashing*.

Resposta:
Quando as perguntas envolverem apenas *scan* do ficheiro é preferível que o ficheiro esteja organizado em *heap* (desordenado) porque um índice *hash*, para além do calculo da função de *hash*, pode levar a mais acessos a disco, nomeadamente quando a ocupação das páginas não é de 100% (tipicamente é 80%).

c) Apresente um exemplo de uma situação que mostre que a utilização de um índice usando *Extensible Hashing* leva a melhor desempenho do que a utilização de um índice usando *Linear Hashing*.

Resposta:
Se o conjunto de chaves de procura leva a uma distribuição inviesada de valores de chave de *hash*, *Extensible Hashing* parte *buckets* onde é necessário enquanto que *Linear Hashing* o faz em *round-robin*, o que, neste caso, é inútil. No primeiro caso temos melhor ocupação e cadeias de *overflow* menores do que no segundo caso e, assim, procura por igualdade será mais rápida porque levará a menos acessos a disco.

3. Optimização de interrogações [15 pontos]

Considere o seguinte esquema de relação:

```
Artigo(codigo: integer, nome: string(72),
       qmin: integer, qmax: integer, fornecedor: integer)
```

em que o código de artigo é chave candidata com valores entre 0 e 4 999 999. Considere ainda que os 5 000 000 registos cabem em 500 000 páginas de dados e que a relação está guardada num ficheiro ordenado pelo campo `codigo`, com índices secundários densos.

Considere que, para responder a interrogações SQL, pode usar:

1. acesso directo ao ficheiro ordenado
2. um índice em *Linear Hashing* no atributo `codigo`
3. um índice em *B+ tree* no atributo `codigo`

a) Apresente os cálculos que indiquem claramente qual das três estratégias apresentadas levaria a um custo mínimo para responder à seguinte interrogação em SQL:

```
SELECT * FROM Artigo WHERE codigo=100000;
```

Resposta:

Como codigo é chave candidata só 1 registo será qualificado para a resposta.

1/ envolve uma pesquisa binária com custo log_2 500000 (19 I/O).

2/ calcular a função de *hash* e ler qualificados (1 I/O).

3/ ler páginas da B+ tree e depois os registos qualificados (4 I/O).

A estratégia 2 será a melhor neste caso.

[Nota: o número de registos qualificados, em geral, depende do *clustering*: 1 se agrupado e 1 por registo no caso contrário.]

b) Idem, para responder à seguinte interrogação em SQL:

```
SELECT * FROM Artigo WHERE codigo<100000;
```

Resposta:

1/ Acesso directo ao ficheiro ordenado enquanto codigo<100000 (10 000 I/O).

2/ O índice em *hash* não ajuda e, neste caso, é claramente mau.

3/ No caso da B+ tree é necessário procurar na árvore e por isso é pior do que a estratégia 1.

A estratégia 1 será a melhor neste caso.

c) Idem, para responder à seguinte interrogação em SQL:

```
SELECT * FROM Artigo WHERE codigo>100000 AND codigo<100010;
```

Resposta:

1/ Pesquisa binária para encontrar o primeiro registo e depois acesso a mais uma página de disco (20 I/O) .

2/ fazendo 9 procuras com igualdade (1,25 x 9 + 9 = 20 I/O).

3/ Pesquisa no índice para encontrar o primeiro e depois acesso a mais uma página de disco (4 I/O).

A estratégia 3 será a melhor neste caso.

4. Limitações do Modelo Relacional [5 pontos]

O Modelo Relacional apresenta diversas vantagens em relação a outros modelos de dados mas também tem algumas limitações.

a) Diga em que consiste a limitação do Modelo Relacional conhecida como "desadaptação de impedâncias".

Resposta:

Uma vez que o SQL não é computacionalmente completo (não tem recursão, formatação da saída, suficientes operadores, ...) é necessário usar uma linguagem "convencional" como o C ou Java em implementações de sistemas de informação. Assim o programador tem que manter mapeamento entre dois modelos distintos: conjuntos de registos de cada vez (em SQL), registo

de cada vez (em C ou Java), para além do modelo do mundo real. Chama-se "desadaptação de impedâncias" à necessidade do programador trabalhar simultaneamente com estes dois modelos de dados distintos.

5. SQL3, ADTs e Colecções [15 pontos]

Numa dada escola um aluno inscreve-se num dado departamento e pode frequentar por ano uma cadeira de um curso dado por outro departamento da escola. Considere o esquema SQL3 apresentado de seguida:

```
CREATE ROW TYPE InfoAluno AS (nome STRING, departamento STRING)
CREATE ROW TYPE TipoAluno AS (codigo STRING, info InfoAluno)
CREATE ROW TYPE TipoCadeira AS (id numero, departamento STRING, nome STRING)

CREATE TABLE Aluno OF TYPE TipoAluno PRIMARY KEY (codigo)
CREATE TABLE Cadeira OF TYPE TipoCadeira PRIMARY KEY (id)
CREATE TABLE Frequenta (aluno REF(TipoAluno),
                        cadeira REF(TipoCadeira), nota:  INTEGER)
```

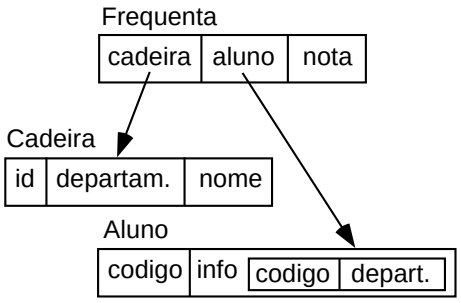

Figura 1: Esquema da BD

a) Escreva uma interrogação SQL3 que encontra os códigos, excluindo os duplicados, de todos os alunos que obtiveram nota superior a 10 numa cadeira de um curso oferecido por um departamento diferente do departamento a que pertencem.

Resposta:

```
SELECT DISTINCT F.aluno->codigo
FROM Frequenta F
WHERE F.nota > 10 AND
        F.cadeira->departamento <> F.aluno->info..departamento;
```

b) Escreva um ou mais triggers que impeçam um aluno de frequentar mais do que uma cadeira de um departamento diferente do departamento onde o aluno está inscrito.

Resposta:

```
// um primeiro trigger para as modificações em Frequenta
CREATE TRIGGER soUmaFora_1
    AFTER
        INSERT ON Frequenta,
        UPDATE OF aluno ON Frequenta,
        UPDATE OF cadeira ON Frequenta
    REFERENCING NEW AS n_freq
    FOR EACH ROW
    WHEN ( 1 <
        ( SELECT count(*)
          FROM Frequenta F
          WHERE F.aluno->codigo = n_freq.aluno->codigo AND
                F.cadeira->departamento <> F.aluno->info..departamento;
        )
    )
```

```
    BEGIN
        ROLLBACK;
    END;

// um segundo trigger para as modificações em Aluno
CREATE TRIGGER soUmaFora_2
    AFTER
        UPDATE OF info ON Aluno
    REFERENCING NEW AS n_al
    FOR EACH ROW
    WHEN ( 1 <
        ( SELECT count(*)
          FROM Frequenta F
          WHERE F.aluno->codigo = n_al.codigo AND
                F.cadeira->departamento <> F.aluno->info..departamento;
        )
    )
    BEGIN
        ROLLBACK;
    END;

// um terceiro trigger para os casos restantes;
// é pouco eficiente pois vamos verificar para todos os alunos;
// este trigger, incluídos os eventos cobertos pelos casos anteriores,
// era suficiente para responder à pergunta do enunciado CREATE TRIGGER soUmaFora_3
    AFTER
        UPDATE OF departamento ON Cadeira
    FOR EACH ROW
    WHEN ( EXISTS
        ( SELECT *
          FROM Aluno A
          WHERE ( 1 <
              ( SELECT count(*)
                FROM Frequenta F
                WHERE F.aluno->codigo = A.codigo AND
                      F.cadeira->departamento <> F.aluno->info..departamento;
              )
        )
    )
    BEGIN
        ROLLBACK;
    END;
```

c) Escreva uma asserção que impeça os alunos do Departamento de "Línguas Modernas" (valor `linguas` no atributo `departamento`) de frequentar a cadeira de "Bases de Dados" (valor `24` no atributo `id`).

Resposta:

```
CREATE ASSERTION linguasFora
    AFTER
        INSERT ON Frequenta,
        UPDATE OF aluno ON Frequenta,
        UPDATE OF cadeira ON Frequenta,
        UPDATE OF departamento ON Cadeira,
        UPDATE OF info ON Aluno
    CHECK ( NOT EXISTS
```

```
        ( SELECT *
          FROM Frequenta F
          WHERE F.cadeira->id = 24 AND
                F.aluno->info..departamento = 'linguas'
        )
);
```

6. Persistência, Colecções e Módulos [10 pontos]

Considere novamente a base de dados do problema 5.

a) Apresente um módulo persistente de servidor com a função e o procedimento seguintes:

```
positivas_de(cadeira: STRING) : integer; // numero de notas positivas
altera_nota(aluno: STRING, cadeira: STRING, nota: INTEGER); // altera nota
```

Resposta:

```
CREATE MODULE Escola
        LANGUAGE SQL;
    CREATE FUNCTION positivasDe(idCadeira: STRING): INTEGER
    BEGIN
        DECLARE cnt INTEGER;
        SELECT COUNT(*) INTO cnt
            FROM Frequenta
            WHERE nota > 9 AND cadeira->id = idCadeira;
        RETURN cnt;
    END;
    CREATE PROCEDURE alteraNota(codAluno: STRING,
                    idCadeira: STRING, novaNota: INTEGER)
    BEGIN
        UPDATE Frequenta
            SET nota = novaNota
            WHERE aluno->codigo = codAluno AND cadeira->id = idCadeira;
    END;
END MODULE;
```

7. Estrutura Lógica de Documentos XML [10 pontos]

Considere o seguinte DTD (`Futebol.dtd`) para documentos XML:

```
<!DOCTYPE Futebol [
     <!ELEMENT DOC-FUTEBOL (ESTADIO+, EQUIPA+, JOGO*)>
     <!ELEMENT ESTADIO EMPTY>
     <!ELEMENT EQUIPA (NOME, CIDADE)>
     <!ELEMENT JOGO (VISITADO, VISITANTE?)>
     <!ELEMENT NOME (#PCDATA)>
     <!ELEMENT CIDADE (#PCDATA)>
     <!ELEMENT VISITADO (GOLOS | PONTOS)>
     <!ELEMENT VISITANTE (GOLOS | PONTOS)>
     <!ELEMENT GOLOS (#PCDATA)>
     <!ELEMENT PONTOS (#PCDATA)>
     <!ATTLIST ESTADIO Cod ID #REQUIRED Nome CDATA>
     <!ATTLIST EQUIPA Cod ID #REQUIRED>
     <!ATTLIST JOGO Estadio IDREF #REQUIRED>
     <!ATTLIST VISITADO Equipa IDREF>
     <!ATTLIST VISITANTE Equipa IDREF>
  ]>
```

a) Verifique se o documento XML seguinte é bem formado e se está conforme com o DTD apresentado (isto é, se é válido); no caso de não estar, assinale os pontos onde isso se verifica.

```
<?XML VERSION="1.0" STANDALONE="no"?>
<!DOCTYPE Exames SYSTEM "../DTDs/Futebol.dtd"> // A
  <DOC-FUTEBOL>
    <ESTADIO Cod="ES1" Nome="Antas"/>
    <EQUIPA Cod="EQ1"> // 1
      <NOME>F.C.Porto</NOME>
      <CIDADE>Porto</CIDADE>
    </EQUIPA>
    <ESTADIO Cod="ES2"></ESTADIO>
    <EQUIPA Cod="EQ2">
      <NOME>F.C.Porto B</NOME>
    </EQUIPA> // 2
    <EQUIPA Cod="ES3">
      <NOME>F.C.Porto C</NOME>
    </EQUIPA> // 2
    <JOGO Estadio="Antas"> // 3
      <VISITADO equipa="EQ1">
        <GOLOS>5</GOLOS>
        <PONTOS>3</GOLOS> // 4 B
      </VISITADO>
      <VISITANTE equipa="ES3">
        <GOLOS>0</GOLOS>
        <PONTOS>0</GOLOS> // 4 B
      </VISITANTE>
    </JOGO>
    <JOGO Estadio="ES1">
      <VISITADO equipa="EQ1">
        <GOLOS>5</GOLOS></VISITADO>
      <VISITANTE equipa="EQ2">
        <GOLOS>0</GOLOS></VISITANTE>
    </JOGO>
    <JOGO> // 5
      <VISITADO equipa="EQ1">
        <PONTOS>0</PONTOS></VISITADO>
      <VISITANTE equipa="EQ1">
        <PONTOS>0</PONTOS></VISITANTE>
    </JOGO>
  </DOC-FUTEBOL>
```

Resposta:
O documento XML não é bem formado:

A Deveria ser <!DOCTYPE <Futebol> ... (CUT+PASTE error ;-)

B A tag de fecho de <PONTOS> é </PONTOS>

O documento XML viola o DTD nos seguintes pontos:

1 <EQUIPA> depois de todos os <ESTADIO>

2 <EQUIPA> contém <CIDADE> depois de <NOME>

3 ID inexistente

4 <VISITADO> e <VISITANTE> não podem conter <PONTOS> e <GOLOS>

5 O atributo "Estadio" (IDREF) é obrigatório

8. Transformação e apresentação de XML [10 pontos]

Considere novamente o DTD apresentado no problema 7.

a) Apresente um conjunto de regras de transformação XSLT que permitam passar para HTML, para ser mostrado num navegador Web, o nome da equipa visitada em cada jogo e o número de pontos que lhe estão associados em instâncias de documentos XML de acordo com este DTD.

Resposta:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="htpp://www.w3.org/XSL/TRANSFORM/1.0">
<xsl:template match="/">
    <HTML>
    <BODY>
    <xsl:for-each select="//JOGO">
        <xsl:apply-templates select="VISITADO" />
            <P> Visitado:
            <xsl:value-of select="id({@Equipa})/NOME" />
            <xsl:apply-templates/>
    </xsl:for-each>
    </BODY>
    </HTML>
</xsl:template>
<xsl:template match="PONTOS">
    <BR> Pontos:
    <xsl:value-of select="." />
</xsl:template>
</xsl:stylesheet>
```

9. XML Schemas [10 pontos]

Considere novamente o DTD apresentado no problema 7.

a) Considerando apenas os elementos ESTADIO e EQUIPA, apresente a parte de um XML Schema equivalente ao DTD (isto é, que permita as mesmas instâncias de documentos XML)

Resposta:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="htpp://www.w3.org/2001/XMLSchema"
            targetNamespace="htpp://www.Futebol.org"
            xmlns="htpp://www.Futebol.org" />
...
    <xsd:element name="ESTADIO" maxOccurs="Unbound">
        <xsd:complexType>
            <xsd:attribute name="Cod" use="required" type="xsd:ID" />
            <xsd:attribute name="Nome" type="xsd:string" />
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="EQUIPA" maxOccurs="Unbound">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="NOME" type="xsd:string" />
                <xsd:element name="CIDADE" type="xsd:string" />
            </xsd:sequence>
            <xsd:attribute name="Cod" use="required" type="xsd:ID" />
        </xsd:complexType>
    </xsd:element>
```

...
</xsd:schema>

10. Gestão de Transacções [5 pontos]

Considere o seguinte escalonamento de transacções (ainda incompleto), que contém, por ordem temporal, as seguintes operações de leitura e escrita:

`T1:R(A), T1:W(B), T3:W(B), T2:R(B), T1:W(A), T2:R(B)`

T1	T2	T3
R(A)		
W(B)		
		W(B)
	R(B)	
W(A)		
	R(B)	

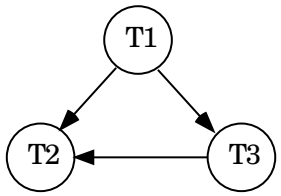

Figura 2: Grafo de Serialização

em que, por exemplo, `T1:R(A)` é operação de leitura no item `A` efectuada pela transacção `T1` e `T3:W(B)` é operação de escrita no item `B` efectuada pela transacção `T3`.

a) Averigue se o escalonamento apresentado é possível e seriável; no caso afirmativo apresente o escalonamento série equivalente e no caso contrário apresente acções que o tornam seriável.

Resposta:
O escalonamento apresentado é possível. Como o grafo de serialização não contém ciclos (ver figura 2), o escalonamento apresentado é serializável. O escalonamento série equivalente será: T1, T3, T2.

FIM.