FEUP

Licenciatura em Engenharia Informática e Computação
Tecnologia de Sistemas de Gestão de Bases de Dados
2001/2002

Exame de Avaliação

26 de Julho de 2002

Resolução do Exame
(29 de Julho de 2002)

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

   Para guardar de forma persistente os dados de um Sistema de Informação, os SGBD usam ficheiros com registos.

   Descreva os dois formatos possíveis para registos e compare-os por forma a evidenciar as vantagens e desvantagens de cada um deles.

   Resposta:
   1. Formato com registos de tamanho fixo: é mais fácil de implementar já que, como o tamanho do registo é fixo, os registos podem ser guardados consecutivamente; é muito rápido obter os endereços dos registos.

   2. Formato com registos de tamanho variável: é muito mais flexível já que pode guardar e manter registos de tipo diferente no mesmo ficheiro, permite registos com campos de tamanho variável e permite registos com campos repetidos; pode ser implementado usando um símbolo especial para representar o fim do registo; pode também ser implementado usando registos de tamanho fixo e *pointers*.

2. **Indexação e Ordenação [15 pontos]**

   Considere um esquema de relação $R(ABCD)$ e uma dada instância com 1000000 registos e com 10 registos por página. A relação está guardada num ficheiro ordenado no atributo A, sendo A uma chave candidata com valores entre 0 a 999999.

   Considere as seguintes interrogações:

   I1: Mostrar todos os tuplos de R

   I2: Mostrar todos os tuplos de R tais que $A < 100$

   I3: Mostrar todos os tuplos de R tais que $A = 100$

   I4: Mostrar todos os tuplos de R tais que $A > 100$ e $A < 200$

   Considere, por último, 3 estratégias para obter as respostas:

   E1: *scan* de todo o ficheiro

   E2: uso de uma *B+ tree* no atributo $R.A$

   E3: uso de um índice *hash* no atributo $R.A$

   Calcule o custo de obter os registos para cada interrogação I1 a I4 usando cada uma das 3 aproximações E1 a E3 (pode apresentar os resultados, por exemplo, numa tabela com 12 células); com base nesses cálculos escolha a aproximação com menor custo para cada interrogação.

   Resposta:
   Os custos para cada interrogação usando as estratégias indicadas são:

Query	**E1 (*scan*)**	**E2 (*B+ tree*)**	**E3 (*hash*)**
I1	10^5	$h + 10^6/M + 10^6$	$10^6/(cM) + 10^6$
I2	10	$h + 100/M + 10$	$100 + 10$
I3	$log_2(10^5)$	$h + 1$	$1 + 1$
I4	$log_2(10^5) + 10$	$h + 99/M + 10$	$99 + 10$

Considerando $B = 10^5$ (páginas), $h = 3$ (altura da árvore), $M > 10$ (entradas de dados por página) e $c = 0.8$ (factor de enchimento), temos:

para I1: o melhor é fazer um *scan*
para I2: o *scan* (10 I/O) é melhor do que a *B+ tree* (23 I/O)
para I3: *hash* (2 I/O) é melhor do quer a *B+ tree* (4 I/O)
para I4: *B+ tree* (23 I/O) do que *scan* (27 I/O)

3. Optimização de interrogações [15 pontos]

Considere o seguinte esquema de relação de empregados de uma dada empresa:

`Emp(codigo, nome, salario, idade)`

A relação tem registos de 100 bytes guardados em 10000 páginas com 20 registos cada. Foram criados vários índices usando a alternativa (2) com entradas de dados que ocupam 20 bytes:

- índice *hash* em `codigo`
- índice *B+ tree* em `salario`
- índice *B+ tree*, agrupado (*clustered*), em `idade`

a) Apresente o melhor plano e o respectivo custo para responder à seguinte interrogação, considerando uma selectividade de 10% na condição.

```
SELECT *
FROM Emp
WHERE idade > 20;
```

Resposta:
Usando o índice *B+ tree* agrupado em `idade` temos um custo igual ao custo da procura na árvore, mais o custo do *scan* do índice, mais o custo de retirar as páginas ($2 + 10000 * 20/100 * 0.1 + 10000 * 0.1 = 1202$ I/O).

b) Apresente o melhor plano e o respectivo custo para responder à seguinte interrogação, considerando uma selectividade de 10% na condição.

```
SELECT *
FROM Emp
WHERE salario > 1000;
```

Resposta:
Um *scan* tem um custo de 10000 I/O. Usando o índice *B+ tree* em `salario` temos, no pior caso, um custo $2 + 10000 * 20/100 * 0.1 + 10000 * 20 * 0.1 = 20202$ I/O). Por isso o melhor é o *scan*.

c) Apresente o melhor plano e o respectivo custo para responder à seguinte interrogação, considerando uma selectividade de 10% na condição.

```
SELECT AVR(salario)
FROM Emp
WHERE salario > 1000;
```

Resposta:
Pode ser usado um plano só no índice *B+ tree* em `salario`, já que não precisamos dos tuplos, pois basta ter as entradas de dados para calcular a média dos salários. O custo é $2 + 200 = 202$ I/O.

4. Limitações do Modelo Relacional [5 pontos]

Para suprir as limitações do Modelo Relacional foram propostos novos modelos de dados, nomeadamente o Modelo Relacional-Objecto e o Modelo Orientado aos Objectos. ODMG é uma norma estabelecida para SGBDs O-O e inclui um modelo de dados e uma linguagem de interrogação.

Faça uma breve comparação do standard proposto em ODMG2.0 para linguagem de interrogação (OQL) com o standard existente para o modelo relacional (SQL92).

Resposta:
OQL possui objectos e identidade, expressões de caminho, invocação de métodos, polimorfismo e *late binding* mas, ao contrário de SQL92, não possui DML (operações de INSERT, UPDATE e DELETE).

5. SQL3, ADTs e Colecções [10 pontos]

Considere o esquema SQL3 apresentado de seguida:

```
CREATE ROW TYPE Address AS (
    street STRING, city STRING, country STRING
);

CREATE ROW TYPE Supplier AS (
    id INTEGER, name STRING, address LIST(Address)
);

CREATE ROW TYPE Part AS (
    id INTEGER, name STRING, colour STRING
);

CREATE TABLE Suppliers OF TYPE Supplier
    PRIMARY KEY (id);

CREATE TABLE Parts OF TYPE Part
    PRIMARY KEY (id);

CREATE TABLE Catalog (
    supplier   REF(Supplier),
    part       REF(Part),
    Cost       REAL
    PRIMARY KEY (supplier, part)
);
```

Escreva uma interrogação SQL3 que mostre o nome dos fornecedores (`Suppliers`) que fornecem todas as peças de cor vermelha (`red`)

Resposta:

```
SELECT S.name
FROM Suppliers S
WHERE ( NOT EXISTS
    ( SELECT P.id
      FROM Parts P
      WHERE colour='red' )
    EXCEPT
    ( SELECT C.part->id
      FROM Catalog C
      WHERE C.supplier->id = S.id AND C.part->id = P.id AND C.part->colour='red' )
    );
```

6. Módulos Persistentes em SQL3 [10 pontos]

Considere novamente a base de dados do problema 5.

Apresente o código SQL3 de um módulo persistente de servidor com o procedimento `increaseCost`, que soma `amount` ao custo da peça `partid`, e a função `numberOf`, que devolve o número de fornecedores que cobram por uma peça mais do que a média dos preços dessa peça.

```
increaseCost(partid: INTEGER, amount: REAL);
numberOf(): INTEGER;
```

Resposta:

```
CREATE MODULE Sales
        LANGUAGE SQL;
    CREATE PROCEDURE increaseCost(partid: INTEGER, amount: REAL)
    BEGIN
        UPDATE Catalog
            SET cost = cost + amount
            WHERE part->id = partid;
    END;
    CREATE FUNCTION numberOf(): INTEGER
    BEGIN
        DECLARE cnt INTEGER;
        SELECT COUNT(DISTINCT C1.supplier->id) INTO cnt
            FROM Catalog C1
```

```
            WHERE C1.cost >
            ( SELECT AVR(C2.cost)
              FROM Catalog C2
              WHERE C2. part->id = C1.part->id )
        RETURN cnt;
    END;
END MODULE;
```

7. **Restrições de Integridade e Gatilhos [10 pontos]**

Considere novamente a base de dados do problema 5.

a) Escreva um ou mais gatilhos em SQL3 para impor a seguinte regra de negócio: "não existe nenhuma peça fornecida por 'LOPESES, LDa' (id=32) e por mais nenhum fornecedor".

Resposta:

```
CREATE TRIGGER sales_T1
    AFTER
        INSERT ON Catalog,
        UPDATE OF supplier ON Catalog,
        UPDATE OF part ON Catalog
    FOR EACH STATEMENT
    WHEN ( EXISTS (
        SELECT C1.supplier->id
        FROM Catalog C1
        WHERE C1.supplier->id = 32 AND NOT EXISTS
            ( SELECT C2.supplier->id
              FROM Catalog C2
              WHERE C2.supplier->id <> 32 AND C1.part->id = C2.part->id ))
    )
    BEGIN
        ROLLBACK;
    END;
```

b) Escreva uma asserção em SQL3 para impor a seguinte regra de negócio: "um fornecedor não pode vender peças verdes e também peças vermelhas".

Resposta:

```
CREATE ASSERTION sales_A1
    AFTER
        INSERT ON Catalog,
        UPDATE OF part ON Catalog,
        UPDATE OF supplier ON Catalog,
        UPDATE OF colour ON Catalog
    CHECK ( NOT EXISTS
        ( SELECT DISTINCT C1.supplier->id
          FROM Catalog C1
          WHERE C1.part->colour = 'green' )
        INTERSECT
        ( SELECT DISTINCT C2.supplier->id
          FROM Catalog C2
          WHERE C2.part->colour = 'red' )
        )
);
```

8. **Estrutura Lógica de Documentos XML [10 pontos]**

Considere o seguinte DTD para documentos XML:

```
<!DOCTYPE Sales [
   <!ELEMENT SALES (CATALOG*)>
   <!ELEMENT CATALOG (PART+, SUPPLIER+, SALE*)>
   <!ELEMENT PART EMPTY>
   <!ATTLIST PART Cod ID #REQUIRED Name CDATA Colour CDATA>
   <!ELEMENT SUPPLIER (ADDRESS* | PHONE*)>
   <!ATTLIST SUPPLIER Cod ID #REQUIRED Name CDATA>
   <!ELEMENT ADDRESS EMPTY>
```

```
   <!ATTLIST ADDRESS Street CDATA City CDATA Country CDATA>
   <!ELEMENT SALE EMPTY>
   <!ATTLIST SALE Part IDREF #REQUIRED Supplier IDREF #REQUIRED Cost CDATA>
   <!ELEMENT PHONE (#PCDATA)>
]>
```

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 Sales SYSTEM "../DTDs/sales.dtd">
<SALES>
   <CATALOG>
      <PART Cod="P1" Name="Part1" Colour="blue"> // A
      <PART Cod="P2" Name="Part2" Colour="blue"></PART>
      <SUPPLIER Cod="S1" Name="LOPESES LDa"/>
      <SUPPLIER Cod="S2" Name="LOPESES LDa"/> // 1
         <ADDRESS Street="Robert Colds" City="Porto" Country="PT"/>
         <PHONE>22.222.2345</PHONE> // 2
         <ADDRESS Street="Braggs" City="Porto" Country="PT"/>
      </SUPPLIER>
      <SALE Part="P1" Supplier="S2" Cost="100.50"/>
      <SALE Part="P1" Supplier="S1" Cost="105.50"/>
      <SALE Part="P1" Supplier="P2" Cost="110.50"/>
   </CATALOG>
   <CATALOG>
      <PART Cod="P1" Name="Part1" Colour="blue"/> // 3
      <PART Cod="P2" Name="Part2" Colour="blue"/> // 3
      <SALE Part="P1" Supplier="S1" Cost="115.50"/> // 4
   </CATALOG>
</SALES>
```

Resposta:
O documento XML não é bem formado:
A. Não existe *tag* de fecho em <PART>
O documento XML viola o DTD nos seguintes pontos:
1. <SUPPLIER> não é EMPTY
2. O elemento <PHONE> vem depois do elemento <ADDRESS> e não podem coexistir
3. ID repetido
4. Elemento <SALE> deve ter pelo menos um elemento <SUPPLIER>

9. Transformação e apresentação de XML [10 pontos]

Considere novamente o DTD apresentado no problema 8 e instâncias de documentos XML de acordo com esse DTD.

Apresente um conjunto de regras de transformação XSLT que permitam passar para HTML, para ser mostrado num navegador Web, o nome do fornecedor 'S1' e o código e o preço de todas as suas vendas em todos os catálogos.

Resposta:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="htpp://www.w3.org/XSL/TRANSFORM/1.0">
<xsl:template match="/">
    <HTML>
    <BODY>
    <H1> <xsl:value-of select="id('S1')/@Name"/> </H1>
    <xsl:for-each select="//Sales/@[Supplier='S1']">
        <P> Part: <xsl:value-of select="@Part"/>
        <BR> Price: <xsl:value-of select="@Cost"/>
    </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

10. XML Schemas [10 pontos]

Para o DTD apresentado no problema 8 e considerando apenas o elemento `SUPPLIER`, apresente a parte correspondente de um XML Schema equivalente ao DTD (isto é, que permita as mesmas instâncias de documentos XML)

Resposta:

```
<xsd:element name="SUPPLIER" minOccurs="1" maxOccurs="Unbound">
    <xsd:complexType>
        <xsd:choice>
            <xsd:element name="ADDRESS" minOccurs="0" maxOccurs="Unbound">
                <xsd:complexType>
                    <xsd:attribute name="Street" type="xsd:string"/>
                    <xsd:attribute name="City" type="xsd:string"/>
                    <xsd:attribute name="Country" type="xsd:string"/>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="PHONE" minOccurs="0" maxOccurs="Unbound"/ >
        </xsd:choice>
        <xsd:attribute name="Cod" use="required" type="xsd:ID"/>
        <xsd:attribute name="Nome" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>
```

FIM.