FEUP Universidade do Porto
Faculdade de Engenharia

Licenciatura em Engenharia Informática e Computação
Tecnologias de Sistemas de Gestão de Bases de Dados
2000/2001

Exame de Avaliação

26 de Julho de 2001

NOME: **Resolução do Exame (31 de Julho de 2001)** _____

1. Armazenamento de Dados: Ficheiros e Índices [10 pontos]

O gestor de buffers de um SGBD pode marcar (*pin*) páginas para impedir que sejam trocadas por outras do disco e mantem uma indicação das páginas de buffer com modificaçoes (*dirty bit*).

a) Descreva o que acontece se há um pedido para uma página que não está em buffer numa altura em que todas as páginas têm modificações.

Resposta:

Se existirem páginas não marcadas (*pinned*) o gestor de buffer escolhe uma página de acordo com uma dada política de troca e passa-a a disco por forma a poder trazer a nova página para uma página de buffer livre.

No caso de não existir nenhuma página nessas condições, o gestor de buffer tem de esperar até que uma página esteja disponível ou então assinala um erro para o nível superior que pediu a nova página.

2. Indexação [15 pontos]

Considere o seguinte esquema de relação:

```
Docentes (bi, nome, telefone, departamento)
```

Considere uma instância com 100000 registos e que o atributo `bi` é chave candidata da relação; este atributo tem valores entre 0 a 99999 e cabem 10 registos por página.

A relação está guardada num ficheiro de registos por ordem leatória (*heap file*) com índices secundários densos.

Considere por último a seguinte interrrogação em SQL:

```
SELECT D.nome
FROM Docentes D
WHERE D.bi > 100 AND D.bi < 200;
```

a) Apresente os cálculos que achar necessários para determinar qual é o plano com custo mínimo para responder à interrogação apresentada considerando que, depois de encontrar a entrada de dados do índice, é necessário mais um acesso a disco para aceder ao registo, para todos os seguintes casos:

1. um `scan` do ficheiro;
2. um índice em B+ tree no campo `bi` com M (por exemplo 10) entradas por página de dados do índice e altura h (por exemplo 2);

3. uma tabela de dispersão (*hash*) no campo `bi` com factor de ocupação c (por exemplo 0,8).

Resposta:
O número de páginas é 100000/10=10000.

1. Um scan do ficheiro tem um custo de 10000 I/O.
2. O uso de uma B+ tree com h=2 e M=10 leva a um custo = (custa de encontrar a primeira página usando o índice) + (custo de percorrer o índice nas entradas de dados da selecção) + (custo de retirar as páginas com os registos seleccionados). Assim, para uma selecção de 99 registos; temos um custo = 2 + 99/M + 99 = 111 I/O.
3. O índice em hash não ajuda nas selecções com intervalos; teremos de fazer 99 acessos usando a tabela de dispersão para depois retirar os 99 registos da selecção; custo = (1/c)*99 + 99 = 223 I/O.

O melhor é o caso 2.

3. Optimização de interrogações [15 pontos]

Considere o seguinte esquema relacional:

```
Emp(eid: integer, salario: integer, idade: real, did: integer)
Proj(pid: integer, codigo: integer, relato:  varchar)
Dept(did: integer, pid: integer, orcamento: real, estado: char(10))
```

e a seguinte interrrogação em SQL:

```
SELECT *
FROM Emp E, Dept D
WHERE E.did=D.did;
```

Considere que existem 20000 tuplos em `Emp` e que cada um ocupa 20 bytes; que existem 5000 tuplos em `Dept` e que cada um ocupa 40 bytes; e que existem 1000 tuplos em `Proj` e que cada um ocupa, em média, 2k bytes. Considere ainda que cada departamento identificado por (*did*) detém, em média, 10 projectos (*pid*).

O sistema de ficheiros suporta página de 4k bytes e existem 12 páginas de buffer.

a) Supondo a existência apenas de um índice aglomerado do tipo tabela de dispersão no campo *did* de `Emp`, enumere os planos que devem ser considerados pelo optimizador para responder à pergunta e calcule uma estimativa de custos para cada um por forma a escolher o melhor.

Resposta:
Em E temos M = 20000*20/4k $\simeq$ 100 páginas; número de tuplos por página P_M = 4K/20 $\simeq$ 200. Em D temos N = 5000*40/4k $\simeq$ 50 páginas; número de tuplos por página P_N = 4K/40 $\simeq$ 100.

Podem considerar-se os seguintes planos:

1. *Nested Loops Join* orientado ao tuplo com E como externa: M + M * P_M * N = 100 + 20000 * 50 = 1000100 I/O.
2. *Nested Loops Join* orientado à página com E como externa: M + M * N = 100 + 100 * 50 = 5100 I/O.
3. *Blocked Nested Loops Join* com E como externa: M + (blocos da externa 100/10 = 10) * N = 100 + 10 * 50 = 600 I/O.
4. *Blocked Nested Loops Join* com D como externa: N + (blocos da externa 50/10) = 5) * M = 50 + 5 * 100 = 550 I/O.
5. *Sort Merge Join*: M log N + N log M + (M+N) = 100 log 50 + 50 log 100 + 150 = 420 I/O.
6. *Hash Join* (com memória suficiente): 3 * (M+N) = 450 I/O.

7. *Index Nested Loops* com D como externa e usando o índice em E: N + ((N * P_N) * custo da selecção em E) = 50 + (50 * 100 * (1,2+1) = 11050 I/O.

[E eu a pensar que INL deveria ser o melhor...]

4. Limitações do Modelo Relacional [10 pontos]

Um SGBD Orientado por Objectos tem de fornecer suporte para guardar objectos persistentes, isto é, que sobrevivem ao programa que os criou.

A norma ODMG2.0 tem estabelecidos *bindings* para diversas linguagens de progamação, nomeadamente para Java, caso em que os objectos não podem ser explicitamente apagados quando deixam de ser úteis.

a) Identifique e descreva brevemente o método especificado na norma para implementar o suporte aos objectos Java capazes de persistir.

Resposta:
Para Java é especificada persistência por atingibilidade: todos os objectos atingíveis a partir de uma qualquer raiz de persistência são persistentes (um objecto torna-se persistente se for referido por um objecto persistente); nas implementações basta calcular o fecho transitivo a partir das raízes, na altura de um comprometimento (*commit*).

5. SQL3, ADTs e Colecções [15 pontos]

Suponha que pretende guardar alguma informação de uma companhia de aviação.

Os tipos de avião em uso são identificados por um código, têm uma designação (Airbus 430, Boeing 343, Concorde), uma autonomia (distância máxima em km que podem percorrer sem abastecer) e é ainda guardada uma foto.

Os empregados da companhia podem ser identificados por um código, é ainda guardado o nome e o seu salário e pode saber-se, em qualquer altura, a sua idade.

Para os empregados que são pilotos é possível identificar os tipos de avião que podem pilotar.

Considere ainda válidas as seguintes restrições de integridade:

R1: Nenhum piloto ganha menos do que 3000.

R2: Qualquer um dos pilotos ganha mais do que qualquer um dos empregados que não é piloto.

R3: Um piloto perde a capacidade para pilotar aviões com autonomia superior a 10000 km quando o seu salário subir acima de 10000.

a) Considere que tem ao seu dispor um SGBD relacional-objecto, SQL3, com a possibilidade de definir tipos abstractos, tipos colecção, tipos referência e tabelas encaixadas. Apresente um esquema relacional-objecto, por exemplo usando a notação apresentada nas aulas, para os requisitos enumerados para a aplicação referida, sem esquecer a primeira restrição de integridade (R1).

Resposta:

```
// usando a notação usada nas Teóricas
CREATE TYPE T_Aviao (
    codigo      INTEGER UNIQUE NOT NULL,
    designa     CHAR(16),
    autonomia   INTEGER,
    foto        BLOB(1M)
);
CREATE TABLE Avioes OF T_Aviao (
    codigo PRIMARY KEY
);
```

```
CREATE TYPE T_Empregado (
    codigo      INTEGER UNIQUE NOT NULL,
    nome        CHAR(50),
    salario     INTEGER,
    nascimento  DATE,
    pilota      LIST(REF(T_AVIAO)),
    public FUNCTION idade(): returns INTEGER
);
CREATE TABLE Empregados OF T_Empregado (
    codigo PRIMARY KEY,
    CHECK ((0 = (SELECT COUNT(*) FROM T_EMPREGADOS E, TABLE(E.pilota)))
            OR salario > 3000) );
```

6. Módulos Persistentes em SQL3 [15 pontos]

Considere novamente a base de dados do problema 5.

a) Apresente um módulo persistente de servidor com as seguintes funções e procedimentos:

```
// devolve o total de pilotos com capacidade para pilotar o tipo
// de avião tpa e que têm salario inferior a sal.
quantosPilotam(tpa, sal): integer;
// escreve no stdout o tipo dos aviões em que todos os empregados
// que os pilotam ganham mais do que o salario sal;
todosPilotam(sal);
```

supondo que dispõe da função `printout(arg)` que escreve no *standard output* o valor de `arg`.

Resposta:

```
CREATE MODULE Pilotos
        LANGUAGE SQL;
    CREATE FUNCTION quantosPilotam (tpa: INTEGER, sal: INTEGER): INTEGER
    BEGIN
        DECLARE c INTEGER;
        SELECT COUNT(*) INTO c
        FROM Empregados E, TABLE(E.pilota) P
        WHERE P..codigo= tpa AND E.salario < sal;
        RETURN c;
    END;
    CREATE PROCEDURE todosPilotam (sal: INTEGER)
    BEGIN
        DECLARE tp INTEGER;
        DECLARE Cursor c (
            ( SELECT DISTINCT P.. codigo
              FROM Empregados E, TABLE(E.pilota) P )
            EXCEPT
            ( SELECT DISTINCT P.. codigo
              FROM Empregados E, TABLE(E.pilota) P
              WHERE E.salario <= sal )
        );
        OPEN c;
        LOOP
            FETCH c INTO tp;
            IF c%NOTFOUND THEN
                    LEAVE;
            printout(tp);
```

```
        ENDLOOP;
    END;
END MODULE;
```

7. Restrições de Integridade e Gatilhos [10 pontos]

Considere novamente a base de dados do problema 5.

a) Escreva uma asserção para impor a restrição R3.

Resposta:

```
CREATE ASSERTION pilotos_R3
    AFTER
        INSERT ON Empregados
        UPDATE OF salario ON Empregados
        UPDATE OF pilota ON Empregados
    CHECK ( NOT EXISTS
        ( SELECT *
          FROM Empregados E, TABLE(E.pilota) P
          WHERE E.salario > 10000 AND P..autonomia > 10000
        )
);
```

b) Escreva um ou mais gatilhos em SQL para impor a restrição R2 de forma incremental ao ser actualizada a informação sobre empregados.

Resposta:

```
CREATE TRIGGER pilotos_R2
    AFTER
        INSERT ON Empregados,
        UPDATE OF salario ON Empregados,
        UPDATE OF pilota ON Empregados
    FOR EACH ROW
    WHEN ( EXISTS
        ( SELECT *
          FROM Empregados E1
          WHERE 0 <
              ( SELECT COUNT(*)
                FROM TABLE(E1.pilota) P1 ) // Piloto
              AND E1.salario < ANY
                  ( SELECT E2.salario
                    FROM Empregados E2
                    WHERE 0 =
                        ( SELECT COUNT(*)
                          FROM TABLE(E2.pilota) P // N/ Piloto
                        )
                  )
        )
    )
    BEGIN
        ROLLBACK;
    END;
```

8. Estrutura Lógica de Documentos XML [10 pontos]

Considere o seguinte DTD para documentos XML:

```
<!DOCTYPE Exames [
  <!ELEMENT TRAP-AirP (TIPOAVIAO | VOO)*>
```

```
  <!ELEMENT TIPOAVIAO (AVIAO)*>
  <!ATTLIST TIPOAVIAO Cod ID #REQUIRED Nome CDATA>
  <!ELEMENT AVIAO EMPTY>
  <!ATTLIST AVIAO Cod ID #REQUIRED Horas CDATA>
  <!ELEMENT VOO (DATA)*>
  <!ATTLIST VOO Cod ID #REQUIRED De CDATA #REQUIRED Para CDATA #REQUIRED
            TipoAviao IDREFS #REQUIRED>
  <!ELEMENT DATA (AVIAO, (PILOTO)+)>
  <!ATTLIST DATA Dia CDATA #REQUIRED Mes CDATA #REQUIRED Ano CDATA #REQUIRED>
  <!ELEMENT PILOTO EMPTY>
  <!ATTLIST PILOTO Cod CDATA #REQUIRED Horas CDATA>
]>
```

a) Verifique se o seguinte documentos XML está conforme com o DTD apresentado (é válido) e, no caso de não estar, assinale os pontos onde isso se verifica e diga qual é o erro.

```
<?XML VERSION="1.0" STANDALONE="no"?>
<!DOCTYPE Exames SYSTEM "../DTDs/Exame.dtd">
  <TRAP-AirP>
    <TIPOAVIAO Cod="T10" Nome="Concorde">
      <AVIAO Cod="A100" Horas="44232"/>
      <AVIAO Cod="A101" Horas="34552"/>
      <AVIAO Cod="A102" Horas="88597"/>
    </TIPOAVIAO>
    <TIPOAVIAO Cod="T11" Nome="A300">
      <AVIAO Cod="A200" Horas="123545"/>
      <AVIAO Cod="A201" Horas="433545"/>
      <AVIAO Cod="A202" Horas="685897"/>
    </TIPOAVIAO>
    <VOO Cod="TP100" De="Porto" Para="New York" TipoAviao="T20"> // 3.
      <DATA Dia="26" Mes="Julho" Ano="2001">
        <AVIAO Cod="A100"/> // 5.
        <PILOTO Cod="P10" Horas="10234"/>
        <PILOTO Cod="P24" Horas="15734"/>
      </DATA>
      <DATA Dia="29" Mes="Julho" Ano="2001">
        <AVIAO Cod="A200"/> // 5.
        <PILOTO Cod="P34" Horas="23534"/>
        <AVIAO Cod="A201"/> // 2. 5.
        <PILOTO Cod="P24" Horas="34734"/>
      </DATA>
    </VOO>
    <VOO Cod="TP200" De="Porto" Para="Heathrow" TipoAviao="T11">
      <DATA Dia="26" Mes="Julho" Ano="2001">
        <AVIAO Cod="A201"/> // 5.
        <PILOTO Cod="P34"/>
        <PILOTO Cod="P24"/>
      </DATA> // 1.
    <VOO Cod="TP300" De="Porto" Para="Los Angeles"> // 4.
    </VOO>
  </TRAP-AirP>
```

Resposta:

1/ Falta <VOO>

2/ Só um elemento <AVIAO> em <DATA>
3/ ID inexistente
4/ Atributo <Tipoaviao> é obrigatório
5/ Código de <AVIAO> já existe e é ID [Erro semântico: o melhor seria <DATA> ter atributo <Aviao> IDREF ou o atributo Cod ser CDATA em vez de ID. BTW, <PILOTO> também poderia ter atributo Cod ID e <DATA> Empty com atributo Pilotos IDREFS.]

9. Transformação e apresentação de XML [10 pontos]

Considere novamente o DTD apresentado no problema 8.

a) Apresente um conjunto de regras de transformação XSLT que permitam passar para HTML para ser mostrado num navegador Web, as datas e os pilotos do voo TP100 existentes em documentos XML de acordo com o DTD apresentado.

Resposta:

```
<?XML VERSION="1.0"?>
<xsl:stylesheet xmlns:xsl="htpp://www.w3.org/XSL/TRANSFORM/1.0">
<xsl:template match="/">
    <HTML>
    <BODY>
    <xsl:aply-template select="VOO[@Cod='TP100']"/>
    <xsl:for-each select="DATA">
        <xsl:value-of select="@Dia">
        <xsl:value-of select="@Mes">
        <xsl:value-of select="@Ano">
        <BR><P>
        <xsl:for-each select="PILOTO">
          <xsl:value-of select="@Cod">
          <BR>
        </xsl:for-each>
    </xsl:for-each>
    </BODY>
    </HTML>
</xsl:template>
</xsl:stylesheet>
```

10. Gestão de Transacções [10 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:R(B), T1:W(A), T2:R(B), T3:W(B), T1:W(A), T2:R(B)`

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 serializável; no caso afirmativo apresente o escalonamento série equivalente e no caso contrário apresente acções que o tornam serializável.

Resposta:

Como o grafo contém um ciclo (ver figura 1), o escalonamento apresentado não é serializável.

Basta abortar uma das transacções envolvidas no ciclo (T2 ou T3) para que o escalonamento seja serializável.

FIM.

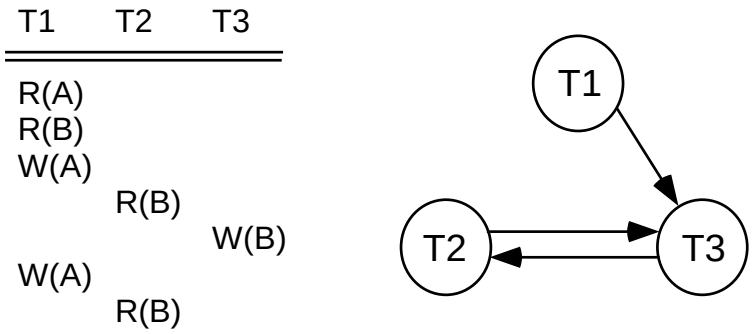

Figura 1: Grafo de Serialização