UNIVERSIDADE FEDERAL DE SÃO CARLOS
CAMPUS SOROCABA

Bacharelado em Ciência da Computação

Laboratório de Banco de Dados

Professora Sahudy Montenegro González

Projeto integrado LabBD, ES2 e WEB

Países, jogos e times

Andre Luiz Beltrami - 489611

Gabriel Cruz Ferreira - 489700

Jhony Miller Campanha - 489751

Fase Intermediária 2

Sorocaba

26/05/2014

Sumário

Sumário de figuras

1. Consultas SQL

Nosso projeto consta de cinco consultas, sendo duas consultas de propósito geral e três consultas específicas, referente ao tema do projeto: países, jogos e times. Nesse tópico descreveremos os enunciados para cada uma das consultas específicas, bem como definiremos campos de busca, operadores de comparação e campos de visualização dos resultados.

1.1 Consultas específicas

As consultas específicas são aquelas referentes ao tema proposto para o nosso grupo, que envolve países, jogos e times. As três consultas trabalham com duas vertentes, como máximo ou mínimo, e por isso as consultas foram divididas em duas, uma para atender a cada vertente.

1.1.1 Vitórias em casa/fora

Descrição

Quantidade de vitorias em casa ou fora dos times de cada pais, ordenado pela quantidade de vitorias, dos jogos entre as datas <data inicial> e <data final>.

Para realizar essa consulta, foram criadas duas consultas: uma para atender às vitórias em casa, outra para atender às vitórias fora de casa.

Parâmetros

Temos como parâmetros dessa consulta dois campos do tipo *date* (*data inicial* e *data final*), que são opcionais.

Campos de projeção

A consulta irá projetar:

- Número de vitórias de cada time em casa ou fora, dependendo da consulta que for utilizada;
- Nome do time;
- País do time; e
- Esporte do time.

Os resultados serão ordenados decrescentemente a partir da coluna de número de vitórias.

Descrição SQL da consulta

```
-- Vitórias em casa
SELECT b.count as vitorias, nome, pais, esporte
FROM time, (
      SELECT COUNT(1), id_time1
      FROM jogo
      WHERE vencedor=id_time1
      AND data BETWEEN <data inicial> AND <data final>
      GROUP BY id_time1 ) b
WHERE id_time=id_time1
ORDER BY vitorias DESC

-- Vitórias fora de casa
SELECT b.count as vitorias, nome, pais, esporte
FROM time, (
      SELECT COUNT(1), id_time2
      FROM jogo
      WHERE vencedor=id_time2
      AND data BETWEEN <data inicial> AND <data final>
      GROUP BY id_time1 ) b
WHERE id_time=id_time1
ORDER BY vitorias DESC
```

1-1: SQL consulta específica 1

1.1.2 Mais/menos vitoriosos

Para a melhor legibilidade dessa consulta criamos uma *view* chamada *times_vitorias* que será analisada na sessão 3.2.1.

Descrição

Time com o maior ou menor número de vitorias em cada país, do esporte <*esporte*>, ordenado pelo número de vitorias

Duas consultas foram feitas: uma para o maior número de vitórias e outra para o menor número de vitórias.

Parâmetros

Temos como parâmetros dessa consulta um campo do tipo *texto* (*esporte*), que é opcional.

Campos de projeção

A consulta irá projetar:

- Número de vitórias de cada time em casa ou fora, dependendo da consulta que for utilizada;
- Nome do time;

- País do time; e
- Esporte do time.

Os resultados serão ordenados decrescentemente a partir da coluna de número de vitórias, caso sejam os times mais vitoriosos. Caso sejam os times menos vitoriosos, então a ordenação será de forma crescente.

Descrição SQL da consulta

```
-- Maior numero de vitorias
SELECT vitorias, nome, t1.pais, t1.esporte
FROM times_vitorias t1, (
      SELECT MAX(vitorias) as maximo, pais, esporte
      FROM times_vitorias
      WHERE esporte like '%<esporte>%'
      GROUP BY pais, esporte) t2
WHERE t2.maximo = t1.vitorias AND t2.pais=t1.pais
ORDER BY vitorias DESC


-- Menor numero de vitorias
SELECT vitorias, nome, t1.pais, t1.esporte
FROM times_vitorias t1, (
      SELECT MIN(vitorias) as maximo, pais, esporte
      FROM times_vitorias
      WHERE esporte like '%<esporte>%'
      GROUP BY pais, esporte) t2
WHERE t2.maximo = t1.vitorias AND t2.pais=t1.pais
ORDER BY vitorias
```

1-2: SQL consulta específica 2

1.1.3 Quantidade de vitórias por campeonato

Para a melhor legibilidade dessa consulta criamos uma *view* chamada *vitorias_campeonatos* que será analisada na sessão 3.2.2.

Descrição

Time com o maior ou menor número de vitorias em cada campeonato, do esporte <*esporte*>, ordenado pelo número de vitorias

Duas consultas foram feitas: uma para o maior número de vitórias e outra para o menor número de vitórias.

Parâmetros

Temos como parâmetros dessa consulta um campo do tipo *texto* (*esporte*), que é opcional.

Campos de projeção

A consulta irá projetar:

- Nome do campeonato;
- Nome do time;
- Número de vitórias do time no campeonato; e
- Esporte do time.

Os resultados serão ordenados decrescentemente a partir da coluna de número de vitórias, caso sejam os times mais vitoriosos. Caso sejam os times menos vitoriosos, então a ordenação será de forma crescente.

Descrição SQL da consulta

```
--Maior numero de vitorias
SELECT t1.nome, nomeTeam, vitorias, esporte
FROM vitorias_campeonatos t1, (
      SELECT MAX(vitorias) AS maximo, id_campeonato
      FROM vitorias_campeonatos
      GROUP BY id_campeonato) t2
WHERE t2.maximo = t1.vitorias
AND esporte like '%<esporte>%'
AND t1.id_campeonato = t2.id_campeonato
ORDER BY vitorias DESC

--Menor numero de vitorias
SELECT t1.nome, nomeTeam, vitorias, esporte
FROM vitorias_campeonatos t1, (
      SELECT MIN(vitorias) AS maximo, id_campeonato
      FROM vitorias_campeonatos
      GROUP BY id_campeonato) t2
WHERE t2.maximo = t1.vitorias
AND esporte like '%<esporte>%'
AND t1.id_campeonato = t2.id_campeonato
ORDER BY vitorias
```

1-3: SQL consulta específica 3

1.2 Consultas Gerais

Neste tópico descreveremos as consultas gerais disponíveis em nossa aplicação. Foram requisitadas duas consultas gerais, relacionadas à *Times* e *Jogadores*.

Essas consultas atuam somente sobre uma tabela do banco de dados, trazendo os dados de acordo com parâmetros específicos que podem ser inseridos pelo usuário.

1.2.1 Consulta tabela Time

Parâmetros

Essa consulta aceita os seguintes parâmetros, sendo todos eles opcionais:

- Nome do jogador;
- Sobrenome do jogador;
- Apelido do jogador;
- Data de nascimento do jogador (dia, mês e ano);
- Jogadores aposentados ou ativos;
- Data da aposentadoria;
- Altura do jogador; e
- Peso do jogador.

Campos de projeção

A consulta irá projetar:

- Nome do jogador;
- Sobrenome do jogador;
- Apelido do jogador;
- Data de nascimento do jogador (dia, mês e ano);
- Jogadores aposentados ou ativos;
- Data da aposentadoria;
- Altura do jogador; e
- Peso do jogador.

Os resultados serão ordenados pelo nome do jogador.

Descrição SQL da consulta

```
SELECT nome, esporte, estadio, pais
FROM time
WHERE nome LIKE '%<nome>%'
AND esporte LIKE '%<esporte>%'
AND pais LIKE '%<pais>%'
ORDER BY nome
```

1-4: SQL consulta geral Time

1.2.2 Consulta tabela Jogador

Parâmetros

Essa consulta aceita os seguintes parâmetros, sendo todos eles opcionais:

- Nome do time;
- Esporte do time; e
- País do time.

Campos de projeção

A consulta irá projetar:

- Nome do time;
- Esporte do time;
- Estádio do time; e
- País do time.

Os resultados serão ordenados pelo nome do time.

Descrição SQL da consulta

```
SELECT nome, sobrenome, apelido, datanasc_dia, datanasc_mes,
datanasc_ano, aposentado, aposenta_data, altura, peso
FROM jogador
WHERE nome LIKE '%<NOME>%'
AND sobrenome LIKE '%<SOBRENOME>%'
AND apelido LIKE '%<APELIDO%>'
AND datanasc_dia = <DIA>
AND datanasc_mes = <MES>
AND datanasc_ano = <ANO>
AND aposentado= <TRUE|FALSE>
AND aposenta_data= <DATA>
AND altura= <ALTURA>
AND peso=<PESO>
ORDER BY nome
```

1-5: SQL consulta geral Jogador

2. Técnicas de acesso eficiente ao Banco de Dados

Nessa seção descreveremos as mudanças feitas no esquema do banco de dados para melhor o tempo de processamento das consultas. Ainda apresentaremos estruturas criadas e otimizações feitas nas consultas, até chegarmos aos resultados apresentados no capítulo 1.

2.1 Mudanças no esquema

Após analisarmos as tabelas que fazem parte do escopo do projeto, optamos por mudar o esquema de alguns campos, de modo a melhorar a performance das consultas.

2.1.1 Tabela Time

A chave primária da relação *Time* tinha como tipo *varchar(500)*. Desse modo, todas as operações de produto cartesiano envolvendo essa relação acarretava em comparação de *string*. Para contornar esse problema, criamos uma chave artificial auto incremental, denominada *id_time*, e a tornamos chave primária.

2.1.2 Tabela Campeonato

De modo análogo à relação *Time*, também criamos uma chave artificial auto incremental para essa relação, denominada *id_campeonato*.

2.1.3 Tabela Jogo

Com as mudanças nas relações descritas anteriormente, os campos *time_1* e *time_2*, *varchar(500)*, foram alterados para *id_time1* e *id_time2*, respectivamente, sendo *foreign keys* da relação *time*. O campo *campeonato* também foi alterado para *id_campeonato*, *foreign key* da relação *campeonato*.

Também alteramos o campo *data* para o tipo *date* e declaramos como chave primária a composição dos campos *id_time1*, *id_time2*, *data* e *horainicio*.

2.1.4 Inconsistências encontradas

Analisando o conjunto de dados propostos, verificamos algumas inconsistências:

- A relação *Time* não continha chave primária explícita, ocorrendo duplicação de duplas. Corrigido com as alterações do banco;
- Nos dados analisados, haviam jogos em que um time jogava com ele mesmo; e
- Encontramos times de um esporte jogando contra times de outro esporte.

As duas últimas inconsistências foram tratadas a partir da Trigger que será descrita na seção 3.3

2.1.5 População do novo banco

O banco foi populado novamente com dados aleatórios, para que houvessem tuplas suficientes para testar as otimizações propostas.

2.1.6 Tabela comparativa de desempenho

Para testar o desempenho antes e após as mudanças utilizamos uma consulta qualquer nos dois bancos e obtivemos os seguintes resultados:

Banco sem alterações	Novo banco
321 ms	61 ms

2-1 Tabela comparativa banco antigo X banco novo

Conforme observado na tabela 2-1, as mudanças diminuíram drasticamente o tempo de execução da consulta utilizada como parâmetro.

2.1.7 Campo vencedor relação Jogo

A última alteração feita no banco foi quanto ao campo *vencedor* da relação *Jogo*. Primeiramente, esse campo armazenava 1 ou 2, representando se o time vencedor foi *time_1* ou *time_2*, respectivamente. Alteramos o modo como esse campo funciona, sendo que agora ele armazena o código do time que venceu o confronto. Essa mudança otimizou um pouco as consultas específicas. Essa melhora será descrita na seção 2.5

2.2 Índice

Percebemos que a primeira consulta específica (ítem 1.1.1) poderia ser otimizada a partir da criação de um índice pela coluna *data* na relação *jogo*. Chegamos a essa conclusão porque esse é o único campo parametrizado da consulta.

A comparação de desempenho será mostrada no final desse capítulo.

2.3 Otimização consulta específica 2

Primeiramente, a view que essa consulta utilizava continha um operador *OR*. Substituímos esse OR por uma UNION e refizemos a consulta, de modo a melhorar o desempenho.

Apesar de obtermos sucesso, com a mudança do campo *vencedor* da relação *jogo* (descrito no ítem 2.1.7), a consulta foi refeita novamente e tivemos mais um ganho de desempenho.

```
SELECT *
    FROM time t2 INNER JOIN (
                  select COUNT(*) as vitorias, id_time
                  from jogo, time
                  WHERE (
                        (vencedor=1 AND id_time=id_time1)
                                    OR
                        (vencedor=2 AND id_time=id_time2)
                  )
                  AND data BETWEEN '1800-06-11' AND '2120-06-20'
                  group by id_time) b
      using (id_time)
```

2-2: SQL times_vitorias - OR

```
SELECT *
    FROM time t2 INNER JOIN (
                  select COUNT(*) as vitorias, id_time
                  from jogo, time
                  WHERE vencedor=1 AND id_time=id_time1
                  AND data BETWEEN '1800-06-11' AND '2120-06-20'
                  group by id_time
            UNION
                  select COUNT(*) as vitorias, id_time
                  from jogo, time
                  WHERE vencedor=2 AND id_time=id_time2
                  AND data BETWEEN '1800-06-11' AND '2120-06-20'
                  group by id_time) b
      using (id_time)
```

2-3: SQL times_vitorias - UNION

```
SELECT *
    FROM time t2 INNER JOIN (
                  select COUNT(*) as vitorias, id_time
                  from jogo, time
                  WHERE id_time=vencedor
                  group by id_time) b
      using (id_time)
```

2-4: SQL times_vitorias final

2.4 Otimização consulta específica 3

A consulta 3 funciona de forma análoga à consulta 2, e do mesmo modo utilizava uma view que continha um OR. O procedimento de otimização foi feito do mesmo modo que a consulta anterior, e também obtivemos uma melhora de desempenho.

```
select *
      from (
            select COUNT(1) as vitorias, id_time, time.nome as time,
time.esporte as esporte, id_campeonato
            from jogo, time
            WHERE id_campeonato IS NOT NULL
            and vencedor=1 and id_time=id_time1
                        or
```

```
            vencedor=2 and id_time=id_time2
            group by id_time, id_campeonato) b
      INNER JOIN campeonato using (id_campeonato)
```

2-5: SQL vitorias_campeonatos - OR

```
select *
      from (
            select COUNT(1) as vitorias, id_time, time.nome as time,
time.esporte as esporte, id_campeonato
            from jogo, time
            WHERE (vencedor=1 and id_time=id_time1)
            group by id_time, id_campeonato
      UNION
            select COUNT(1) as vitorias, id_time, time.nome as time,
time.esporte as esporte, id_campeonato
            from jogo, time
            WHERE (vencedor=2 and id_time=id_time2)
            group by id_time, id_campeonato) b
      INNER JOIN campeonato using (id_campeonato)
```

2-6: SQL vitorias_campeonatos - UNION

```
select *
      from (
            select COUNT(*) as vitorias, time.nome as nomeTeam,
id_campeonato, esporte
            from jogo, time
            WHERE vencedor=id_time
            group by id_campeonato,id_time,time.nome) b
      INNER JOIN campeonato using (id_campeonato)
```

2-7: SQL vitorias_campeonatos final

2.5 Tabela comparativa de desempenho

Nessa seção mostraremos tabelas comparativas do desempenho das consultas descritas anteriormente.

A tabela a seguir mostra o desempenho da consulta 1, antes e depois da criação do índice *data* na relação *jogo*:

Sem índice	Com índice
1061 ms	749 ms

2-8: Tabela comparativa índice X sem índice

A tabela a seguir mostra o desempenho das consultas 2 e 3, após cada alteração feita na consulta:

	Consultas primitivas	Substituição: OR por UNION	Mudança no campo Vencedor
Consulta 2	4150 ms	562 ms	150 ms
Consulta 3	4353 ms	1529 ms	750 ms

2-9: Tabela comparativa consultas 2 e 3

Segue anexo a este documento imagens com o plano de execução das consultas, mostrando a melhoria de desempenho.

3. Programação com banco de dados

Nessa seção será apresentado os itens de programação no banco de dados: *Stored Procedures*, *Trigger* e *Views*.

3.1 Stored Procedures

Criamos duas *Stored Procedures*, uma que calcula o aproveitamento de um time em casa, fora e total, e outra que retorna os dados de um time, juntamente com a quantidade total de jogos dele.

3.1.1 Aproveitamento

Essa *Stored Procedure* recebe como parâmetro de entrada o código do time e como parâmetros de saída três *reais*, representando o aproveitamento total, jogando em casa e jogando fora de casa.

```
CREATE or REPLACE FUNCTION aproveitamento (in integer, total out real,
casa out real, fora out real) as $aproveitamento$
DECLARE
      totalCasa integer;
      totalFora integer;
      vitoriasCasa integer;
      derrotasCasa integer;
      vitoriasFora integer;
      derrotasFora integer;
BEGIN
      -- Quantidade de jogos em casa
      SELECT count (id_time1) INTO totalCasa FROM jogo
            WHERE id_time1 = $1;

      -- Quantidade de jogos fora
      SELECT count (id_time2) INTO totalFora FROM jogo
            WHERE id_time2 = $1;

      IF (totalCasa + totalFora) = 0 THEN
            RAISE EXCEPTION 'Time não cadastrado em nenhuma partida';
      END IF;
      -- Vitorias em casa
      SELECT count(id_time1) INTO vitoriasCasa FROM jogo
            WHERE vencedor = $1
            AND id_time1 = $1;

      -- Derrotas em casa
      SELECT count(id_time1) INTO derrotasCasa FROM jogo
            WHERE vencedor <> $1
            AND id_time1 = $1;

      -- Vitorias fora
      SELECT count(id_time2) INTO vitoriasFora FROM jogo
```

```
            WHERE vencedor = $1
            AND id_time2 = $1;

      -- Derrotas fora
      SELECT count(id_time2) INTO derrotasFora FROM jogo
            WHERE vencedor <> $1
            AND id_time2 = $1;

      total := ((vitoriasCasa + vitoriasFora) * 100) / (totalCasa +
totalFora);
      casa := (vitoriasFora * 100) / totalCasa;
      fora := (derrotasCasa *100) / totalFora;
END;
$aproveitamento$ LANGUAGE plpgsql;
```

3-1: Stored Procedure aproveitamento

Primeiramente recuperamos os dados das vitórias e derrotas totais, em casa e fora. Caso o time não esteja cadastrado em nenhum jogo, a *procedure* lança uma exceção, avisando que o time não está cadastrado em nenhuma partida.

Após isso o cálculo do aproveitamento é feito e armazenado nos parâmetros de saída da *procedure*.

Exemplos de chamadas da função:

```
select * from aproveitamento(20)
-- Saída: 49 | 46 | 56
select * from aproveitamento(40)
-- Saída: 47 | 40 | 55
select * from aproveitamento(4000)
-- Saída: ERRO:  Time não cadastrado em nenhuma partida
```

3-2: Chamadas SP aproveitamento

3.1.2 Detalhe time

Essa *stored procedure* tem um único parâmetro de entrada, um inteiro que representa o código do time, e retorna uma string, com os dados do time.

```
CREATE or REPLACE FUNCTION dadosTime (integer) RETURNS text as
$dadosTime$
DECLARE
      id ALIAS FOR $1;
      jogos integer;
      texto text;
      timerow time%ROWTYPE;
      nome text;
      esporte text;
      pais text;
      estadio text;
BEGIN
      -- Quantidade de jogos
      SELECT sum(count) INTO jogos FROM (
            select count(1) from jogo
            where id_time1 = id
            UNION
            select count(1) from jogo
            where id_time2 = id) as foo;
```

```
      -- Informações do time
      SELECT * INTO timerow FROM time where id_time = id;
      IF timerow IS NULL THEN
            RETURN 'Time não cadastrado';
      END IF;

      -- Atribuição às variáveis
      nome := timerow.nome;
      IF nome IS NULL THEN
            nome:= '';
      END IF;

      esporte := timerow.esporte;
      IF esporte IS NULL THEN
            esporte := '';
      END IF;

      pais := timerow.pais;
      IF pais IS NULL THEN
            pais := '';
      END IF;

      IF timerow.estadio IS NOT NULL THEN
            estadio := ', estádio ' || timerow.estadio;
      ELSE
            estadio := '';
      END IF;


      texto := nome || ', ' || esporte || ', ' || pais || estadio ||
'. ' || jogos || ' partidas jogadas.';
      RETURN texto;

END;
$dadosTime$ LANGUAGE plpgsql;
```

3-3: Stored Procedure dadosTime

Primeiramente recuperamos a quantidade total de jogos do time e as informações do time presentes na relação *time*.

Após isso, como alguns campos da relação podem ser nulos, é feito um tratamento para a exibição correta da *string* no retorno.

Depois de todos os tratamentos, todas as informações são concatenadas e o resultante é retornado.

Exemplos de chamadas da função:

```
select dadosTime(1)
-- Saída: "Pittsburg Alleghenys, Beisebol, US, estádio Exposition Park
I. 348 partidas jogadas."
select dadosTime(400)
-- Saída: "Veranopolis ECRC, Futebol, BR. 113 partidas jogadas."
select dadosTime(4000)
-- Saída: "Time não cadastrado"
```

3-4: Chamadas SP dadosTime

3.2 Views

Duas views foram criadas para auxiliar nas consultas específicas 2 e 3

3.2.1 Times_vitorias

Essa view relaciona cada time com a quantidade de vitórias dele. Na consulta específica 2 fazemos um produto cartesiano dessa view com ela mesma. Utilizando a view essa consulta fica muito mais legível.

```
create or replace view times_vitorias as
SELECT *
    FROM time t2 INNER JOIN (
                  select COUNT(*) as vitorias, id_time
                  from jogo, time
                  WHERE id_time=vencedor
                  group by id_time) b
      using (id_time)
```

3-5: SQL view times_vitorias

3.2.2 Vitorias_campeonatos

View com função análoga à view anterior. Utilizada na consulta específica 3.

```
create or replace view vitorias_campeonatos as
select *
      from (
            select COUNT(*) as vitorias, time.nome as nomeTeam,
id_campeonato, esporte
            from jogo, time
            WHERE vencedor=id_time
            group by id_campeonato,id_time,time.nome) b
      INNER JOIN campeonato using (id_campeonato)
```

3-6: SQL view vitorias_campeonatos

3.3 Trigger

Foi criada uma trigger vinculada à inserção na relação *jogo*, para evitar inserção de dados redundantes.

```
CREATE OR REPLACE FUNCTION jogo_check() RETURNS TRIGGER AS $checajogo$
      DECLARE
            esporte1 time.esporte%TYPE;
            esporte2 time.esporte%TYPE;
      BEGIN
            -- Verifica se os dois times são o mesmo
            IF NEW.id_time1 = NEW.id_time2 THEN
                  RAISE EXCEPTION 'os times devem ter nomes diferentes';
            END IF;
            -- Verifica se o vencedor participou do jogo
```

```
            IF (NEW.vencedor <> NEW.id_time1) AND (NEW.vencedor <>
NEW.id_time2) THEN
                  RAISE EXCEPTION 'o vencedor deve participar do jogo';
            END IF;

            -- Verifica se ambos os times são do mesmo esporte
            SELECT esporte INTO esporte1 FROM time where id_time =
NEW.id_time1;
            SELECT esporte INTO esporte2 FROM time where id_time =
NEW.id_time2;
            IF esporte1 <> esporte2 THEN
                  RAISE EXCEPTION 'os times devem ser do mesmo esporte';
            END IF;

            RETURN NEW;
      END;
$checajogo$ LANGUAGE plpgsql;

CREATE TRIGGER jogo_check BEFORE INSERT OR UPDATE ON jogo
      FOR EACH ROW EXECUTE PROCEDURE jogo_check();
```

3-7: SQL trigger jogo_check

Primeiramente a trigger verifica se ambos os times são do mesmo esporte. Depois se o campo vencedor contém o código de um dos dois times envolvidos na partida. Por fim, se ambos são do mesmo esporte.

Caso algumas dessas condições não seja atendida, a trigger lança uma exceção e o novo dado não é inserido na relação.

Exemplos de tentativas de inserção:

```
insert into jogo values (1,1, '2014-05-25', 20, 'estadio1', 1, 21);
-- Saída: ERRO:  os times devem ter nomes diferentes
insert into jogo values (1,2, '2014-05-25', 20, 'estadio1', 3, 21);
-- Saída: ERRO:  o vencedor deve participar do jogo
insert into jogo values (1,462, '2014-05-25', 20, 'estadio1', 1, 21);
-- Saída: ERRO:  os times devem ser do mesmo esporte
```

3-8: Inserções tabela Jogo