Projecto de Bases de Dados, Parte 2

Bruno Cardoso (72619), Lídia Freitas (78559) e Rodrigo Bernardo (78942)

Instituto Superior Técnico

11 de Dezembro de 2015

Grupo 17
Turno: Quinta-Feira, 08h00, LAB 14

Conteúdo

1 Introdução

Este relatório foi elaborado na âmbito da cadeira de Bases de Dados e sumariza o trabalho realizado para a segunda parte do projecto da cadeira.

Nesta disciplina, o projecto envolveu criar um Bloco de Notas, ou seja, uma conjunto de procedimentos e interrogações que melhor descrevem uma forma de armazenamento e consulta de informação. Tal como o bloco de notas tradicional, a implementação deste projecto seguiu as várias categorias para organizar a informações, tal como a implementação de registos, páginas, valores. De forma a limitar o espaço de cada utilizador ao seu bloco de notas, foram criadas diversas verificações para o acesso á informação, garantindo que um utilizador só pode ver informações a que tenha permissão para tal. Este projecto foi implementado utilizando o SGBD Mysql, e as diversas páginas de acesso á informação foram criadas utilizando PHP. Foram tomadas as devidas precauções no que toca ao tratamento de inputs, de forma a manter a segurança da base de dados prevenida, ao fazer a retirada de caracteres indesejados no input. Todos os acessos á base de dados são atómicos, garantido que cada acesso é unico ao mesmo tempo.

Neste pretendemos mostrar a nossa implementação de um bloco de notas assim como os aspectos de interação e melhoria da mesma.

2 Consultas SQL

(a) Quais são os utilizadores que falharam o login mais vezes do que tiveram sucesso?

```
SELECT l.userid
FROM login AS l
WHERE l.sucesso = 0
GROUP BY l.userid
HAVING count(*) > ALL
    (SELECT count(*)
     FROM login AS l1
     WHERE l1.sucesso = 1
         AND l1.userid =l.userid);
```

Esta query é relativamente simples, assim, a nossa implementação baseou-se em contar o número de sucessos de cada utilizador e retorna-lo se este fosse maior que o número de insucessos do mesmo utilizador.

(b) Quais são os registos que aparecem em todas as paginas de um utilizador?
A query utiliza *ID_USER* que deve ser substituido pelo id do utilizador desejado (userid):

```
SELECT r.regcounter
FROM registo AS r
WHERE r.ativo
    AND r.userid = ID_USER
    AND NOT EXISTS
        (SELECT p.pagecounter
         FROM pagina AS p
         WHERE p.userid = ID_USER
             AND r.regcounter NOT IN
                 (SELECT rp.regid
                  FROM reg_pag AS rp
                  WHERE rp.regid = r.regcounter
                      AND rp.pageid = p.pagecounter
                      AND rp.userid = ID_USER
                      AND rp.ativa
                      AND p.ativa
                      AND EXISTS
                          (SELECT tp.typecnt
                           FROM tipo_registo AS tp
                           WHERE tp.typecnt = rp.typeid
                               AND tp.userid = ID_USER
                               AND tp.ativo)))
```

Para esta query seguimos a lógica de que os registos que aparecem em todas as páginas de um utilizador são também os registos para o qual não existe uma página que não tenha esse registo.
Foi também verificado sempre se as entradas nas tabelas se encontravam activas.

(c) Quais os utilizadores que têm o maior número médio de registos por página?

```
SELECT rp.userid
FROM reg_pag as rp
WHERE rp.ativa and
        exists (
                select p.pagecounter
                from pagina p
                where   p.userid = rp.userid and
                        p.pagecounter = rp.pageid and
                        p.ativa) and
        exists (
                select r.regcounter
                from registo r
                where   r.userid = rp.userid and
                        r.regcounter = rp.regid and
                        r.ativo)

GROUP BY rp.userid
HAVING count(*) / count(DISTINCT rp.pageid) >= all
    (SELECT count(*) / count(DISTINCT rp2.pageid)
     FROM reg_pag rp2
     WHERE rp2.ativa and
                exists (
                        select p1.pagecounter
                        from pagina p1
                        where   p1.userid = rp2.userid and
                                p1.pagecounter = rp2.pageid and
                                p1.ativa) and
                exists (
                        select r1.regcounter
                        from registo r1
                        where   r1.userid = rp2.userid and
                                r1.regcounter = rp2.regid and
                                r1.ativo)
             GROUP BY rp2.userid);
```

Para esta query calculamos a média do número de registos por página de um utilizador que tem a média do número de registos por página maior que a de todos os outros utilizadores. Para tal fomos sempre verificando se as entradas estavam activas.

(d) Quais os utilizadores que, em todas as suas páginas, têm registos de todos os tipos de registos que criaram?

```
SELECT u.userid,
       u.nome
FROM
  ( SELECT t.userid,
           min(t.num_tipos_pagina) AS minimo
   FROM
     ( SELECT rp.userid,
              rp.pageid,
              count(DISTINCT rp.typeid) AS num_tipos_pagina
      FROM registo r,
           tipo_registo tr,
           pagina p,
           reg_pag rp
      WHERE r.ativo
        AND tr.ativo AND p.ativa AND rp.ativa
        AND r.typecounter=tr.typecnt
        AND r.regcounter=rp.regid
        AND p.pagecounter=rp.pageid
        AND r.userid=tr.userid
        AND p.userid=r.userid
        AND rp.userid=p.userid
      GROUP BY rp.pageid) t
   GROUP BY userid) t,
     utilizador u
WHERE t.userid=u.userid
  AND t.minimo =
    (SELECT count(*)
     FROM tipo_registo tr1
     WHERE tr1.ativo
       AND tr1.userid= t.userid);
```

Para esta query seguimos a lógica de que os utilizadores que o utilizador tem em todas as suas páginas registos com todos os tipos de registos que criaram se o mínimo de tipos diferentes que tiver em todas as suas páginas for igual ao número de tipo de registos que criou. Para esta query não conseguimos retirar as sub-queries dos FROMs, mesmo após algum esforço e por isso acabamos por ter de deixar como está.

2.1 Notas sobre as Consultas SQL

2.1.1 Consulta b - Quais são os registos que aparecem em todas as páginas de um utilizador?

Para esta consulta assume-se que se um utilizador não tem páginas então é impossível um registo desse mesmo utilizador aparecer em alguma(s) página(s) (devido à inexistência destas).

Entendeu-se também com esta questão que eram pedidos os registos que aparecem em todas as páginas de um dado utilizador, caso contrário diria "para cada o utilizador". No entanto, como o número de casos em que o facto é verídico é baixo decidimos verificar então quais os registos por todos os utilizadores que verificavam a consulta, e por isso deixamos em baixo a consulta que nos permitiu ver essa informação:

```
SELECT r_0.userid,
       r_0.regcounter
FROM registo AS r_0
WHERE regcounter IN
        (SELECT r.regcounter
         FROM registo AS r
         WHERE r.ativo
             AND r.userid = r_0.userid
             AND NOT EXISTS
                 ( SELECT p.pagecounter
                  FROM pagina AS p
                  WHERE p.userid = r_0.userid
                      AND r.regcounter NOT IN
                          ( SELECT rp.regid
                           FROM reg_pag AS rp
                           WHERE rp.regid = r.regcounter
                               AND rp.pageid = p.pagecounter
                               AND rp.userid = r_0.userid
                               AND rp.ativa
                               AND p.ativa
                               AND EXISTS
                                   ( SELECT tp.typecnt
                                    FROM tipo_registo AS tp
                                    WHERE tp.typecnt = rp.typeid
                                        AND tp.userid = r_0.userid
                                        AND tp.ativo))))
GROUP BY r_0.userid,
         r_0.regcounter
```

3 Restrições de Integridade

As restrições de integridade foram implementadas com triggers.

Como os triggers acabam por ser todos muito semelhantes apenas apresentaremos aqui um trigger para um insert e um para um update. Os restantes estarão no ficheiro zip da entrega.

```
drop trigger if exists contador_sequencia_registo_insert;

delimiter $$

    create trigger contador_sequencia_registo_insert before insert on registo
    for each row
    begin
    if (exists(select * from  tipo_registo tr where tr.idseq = new.idseq) or
        exists(select * from  pagina p where p.idseq = new.idseq) or
        exists(select * from  campo c where c.idseq = new.idseq) or
        exists(select * from  registo r where r.idseq = new.idseq) or
        exists(select * from  valor v where v.idseq = new.idseq))

    then
            call contador_sequencia_registo_insert_trigger();
    end if;
    end$$

delimiter ;

drop trigger if exists contador_sequencia_registo_update;

delimiter $$

    create trigger contador_sequencia_registo_update before update on registo
    for each row
    begin
    if (exists(select * from  tipo_registo tr where tr.idseq = new.idseq) or
        exists(select * from  pagina p where p.idseq = new.idseq) or
        exists(select * from  campo c where c.idseq = new.idseq) or
        exists(select * from  registo r where r.regcounter <> old.regcounter and r.idseq=new.
        exists(select * from  valor v where v.idseq = new.idseq))

    then
            call contador_sequencia_registo_update_trigger();
    end if;
    end$$

delimiter ;
```

O trigger de insert funciona da seguinte maneira, quando um utilizador tenta inserir uma nova entrada na tabela de registos, se já existir alguma sequencia com o mesmo valor noutra tabela chama uma função inexistente que fará com que o mysql dê erro e aborte a inserção da entrada. Impedindo assim resultados incorrectos

No trigger de update é necessário verificar também se já existe uma sequência nas outras tabelas com a mesma sequência e neste caso deve também verificar se já existe um registo diferente do que se tenta inserir que tenha o mesmo descritivo de sequência do que o que estamos a tentar inserir. Caso aconteça é chamada uma função inexistente para proibir a modificação da tabela.

4 Formas Normais

(a) A relação utilizador, tem apenas as dependências funcionais (DFs) da forma $X \rightarrow A$, com A pertencente aos atributos desta relação, $X \subseteq$ *{userid, email}* e $X \neq \emptyset$ (para além das DFs triviais). Como em todas estas DFs se tem que o determinante é chave, a relação utilizador encontra-se na *Boyce-Codd Normal Form* (BCNF).

(b) Para além das DFs anteriores, a relação tem agora uma nova DF na qual o determinante não é chave, mas o dependente é. Assim, a relação utilizador encontra-se na terceira forma normal.

Propomos a seguinte decomposição:

$$R_1(\underline{userid}, email, pais, categoria)$$
$$R_2(\underline{userid}, nome, questao1, resposta1, questao2, resposta2).$$

Ambas as relações estão agora na BCNF, pois todas as suas DFs têm chaves como determinantes. Para além disso, pelo Teorema de Heath, esta é uma decomposição sem perdas de informação, pois *userid* é chave primária nas relações R, R_1 e R_2.

5 Índices

O código para esta secção encontra-se no ficheiro "indices.sql".

Para a alínea a) queremos devolver a média do número de registos por página dado um certo utilizador. Como para o cálculo da média apenas é necessário ter conhecimento das quantidades de registos e de páginas desse utilizador e não das suas informações específicas, queremos obter essa informação utilizando apenas o índice. Para aproveitarmos isso, criámos um índice para a tabela reg_pag com uma chave composta que inclui os atributos regid e pageid. Para além disso, o índice deve ser denso.

(Na realidade, para este projecto, o argumento acima mencionado cai por terra, pois para obtermos dados com significado é necessário que a nossa query faça as verificações da cláusula where. Essas verificações implicam que a informação não está toda no índice.)

O índice seria, idealmente, implementado através de uma hash table, visto ser mais rápida que uma árvore binária, quando aplicável. Neste caso seria, pois as condições da cláusula where são todas igualdades. No entanto, o MySQL apenas implementa índices recorrendo a BTREEs.

Na alínea b) queremos ver o nome dos registos associados à página de um dado utilizador. Neste caso não podemos aplicar o truque da alínea anterior, em que o índice fornecia a informação sobre aquilo que queríamos saber. Assim, criámos três indíces densos (que seriam, idealmente, implementados por hash tables) com os índices de forma a acelerar o acesso aos campos verificados na cláusula where e aos campos dos nomes dos registos.

Qualquer benchmark que utilizássemos para tentar provar que os nossos índices beneficiavam o desempenho da das queries seria desprovido de significado devido ao facto de o MySQL não implementar índices como hash tables.

6 Transacções

Para as transações, de forma a não existirem incoerências devido a acessos simultâneos às mesmas variáveis foi necessário criar transações. As transações fazem o acesso às variáveis dentro de uma transação seja efectuado em regime de exclusividade.

De forma a não proibir o acesso durante muito tempo ao programa, pretendemos ser breves nas transações e apenas realizá-las quando

Este facto pode ser verificado no código onde uma transação é inicializada com $connection->beginTransaction() e terminam com $connection->commit().

```
<?php

require "connect.php";

if (($_SERVER["REQUEST_METHOD"] == "POST")  && ($_POST["nome"] != "") &&
 ($_POST["email"] != "") && ($_POST["questao1"] != "") &&
 ($_POST["questao2"] != "") && ($_POST["resposta1"] != "")
 && ($_POST["resposta2"] != "")){

    session_start();

    $nome = $_POST["nome"];
    $email = $_POST["email"];
    $password = $_POST["password"];
    $questao1 = $_POST["questao1"];
    $resposta1 = $_POST["resposta1"];
    $questao2 = $_POST["questao2"];
    $resposta2 = $_POST["resposta2"];
    $pais = $_POST["pais"];
    $categoria = $_POST["categoria"];

    (...)
    $connection->beginTransaction();

    $query_cria  = (...)

    $utilizador_obj = $connection->prepare($query_cria);
    $utilizador_obj->bindParam(":userid", $userid_aux);
    $utilizador_obj->bindParam(":email", $email_aux);
    $utilizador_obj->bindParam(":nome", $nome_aux);
    $utilizador_obj->bindParam(":password", $password_aux);
    $utilizador_obj->bindParam(":questao1", $questao1_aux);
    $utilizador_obj->bindParam(":resposta1", $resposta1_aux);
    $utilizador_obj->bindParam(":questao2", $questao2_aux);
    $utilizador_obj->bindParam(":resposta2", $resposta2_aux);
    $utilizador_obj->bindParam(":pais", $pais_aux);
    $utilizador_obj->bindParam(":categoria", $categoria_aux);


            (...)

    $connection->commit();
    header("Location:␣principal.php");
    }

$connection = null;
?>
```

7 Data Warehouse

A tabela d_user corresponde à dimensão d_utilizador do enunciado. Foram incluídos os atributos uid e userid. Assim, torna-se possível alterar o email de um utilizador sem se perder informação. Se for alterado o email de um utilizador com userid = UID, é criado um novo registo nesta página com os mesmos valores para os atributos, excepto para o email (obviamente) e para o uid. Se fosse colocado o atributo email como chave não seria possível mudar o email. Se fosse colocado o atributo userid como chave não seria possível mudar o email sem se perder o email anterior.

A tabela d_time corresponde à dimensão d_tempo do enunciado. O atributo chave, tid, é redundante no nosso modelo, mas permite que se criem queries de forma mais sucinta do que fazendo o triplo (ano,mes,dia) chave primária da relação. A coerência desta tabela (e das outras duas do diagrama em estrela), é mantida através de um trigger, que será descrito mais adiante.

A tabela f_login corresponde à tabela de factos do diagrama em estrela. Contém como atributos da primary key os atributos que são foreign key para as tabelas d_user e d_time. Como medida é incluído uma coluna attempts que indica o número de tentativas de login associado ao par (uid,tid).

O trigger permite manter a coerência das tabelas do diagrama em estrela. Sempre que são inseridos novos registos na tabela login este trigger é activado e insere as informações desse registo nas tabelas apropriadas. Se um novo utilizador fizer uma tentativa de login, a informação desse utilizador é inserida na tabela d_user. Se um determinado utilizador já tiver feito uma tentativa no mesmo dia, incrementa-se o número de tentativas do par (uid,tid), que identifica o utilizador e o dia, respectivamente. Se não, registamos esse novo par e inicializamos a sua entrada attempts igual a 1 (primeira tentativa do dia para esse utilizador).

A interrogação torna-se simples devido á forma como estruturámos o diagrama em estrela, sendo feito através de um simple AVG à coluna attempts da tabela de factos.

O código encontra-se no ficheiro "dw.sql".

8 Conclusão

Concluindo este relatório, achamos que aprendemos imenso com este projecto. Infelizmente tivemos imensos problemas com debugg do PHP e algumas dificuldades com os indices. Pensamos ter ultrapassado a maior parte das nossas dificuldades mas infelizmente não conseguimos resolver todos os problemas do código do projecto.

Iremos tentar termina-lo na mesma, mesmo fora do âmbito da cadeira pois pensamos que nos é de bastante interesse.