Grupo: 1

Projecto de Base de Dados, Parte 2

1.º Semestre 2015/2016

João Catarino	Luís Morais	Ricardo Mota
78877	78416	78131
(20 horas)	(20 horas)	(20 horas)

Turno: BD22517L09

Índice

1 Consultas em SQL

Escreva num ficheiro com um script em SQL as interrogações abaixo indicadas:

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

```
SELECT
    userid AS utilizador
FROM
    login
GROUP BY userid
HAVING COUNT(sucesso) - SUM(sucesso) > SUM(sucesso);
/* numero de tentativas - numero de sucessos
  (numero de falhas) > numero de sucessos */
```

(b) Quais são os registos que aparecem em todas as páginas de um utilizador?

```
SELECT
    R.regcounter as registo
FROM
    registo R
WHERE
    R.ativo = 1
        AND EXISTS( SELECT /* o registo tem um tipo de registo ativo */
            T.typecnt
        FROM
            tipo_registo T
        WHERE
            T.ativo = 1
                AND NOT EXISTS( SELECT /* e nao ha nenhuma pagina ativa */
                    P.pagecounter
                FROM
                    pagina P
                WHERE
                    R.userid = P.userid
                        AND NOT EXISTS( SELECT /* que nao contenha esse mesmo registo */
                            RP.regid
                        FROM
                            reg_pag RP
                        WHERE
                            P.ativa = 1 AND RP.ativa = 1
                                AND RP.typeid = T.typecnt
                                AND RP.pageid = P.pagecounter
                                AND RP.regid = R.regcounter
                                AND R.userid = RP.userid)))
                                /* para um certo utilizador */
```

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

```
SELECT DISTINCT
    U.userid as user
FROM
    utilizador U
WHERE
    (SELECT
            (COUNT(DISTINCT RP.idregpag) / COUNT(DISTINCT P.pagecounter))
            /* utilizador com valor de registos ativos por pagina ativa - media
                        (numero de registos/numero de paginas) */
        FROM
            reg_pag RP,
            pagina P,
            registo R
        WHERE
            RP.ativa AND P.ativa AND R.ativo
                AND RP.userid = P.userid
                AND RP.userid = R.userid
                AND RP.userid = U.userid
                AND RP.pageid = P.pagecounter
                AND RP.regid = R.regcounter) >= ALL (SELECT
            (COUNT(DISTINCT RP2.idregpag) / COUNT(DISTINCT P2.pagecounter))
            /* maior que qualquer outro valor de outro utilizador, de registos ativos
                        por pagina ativa (registos/paginas) */
        FROM
            utilizador U2,
            reg_pag RP2,
            pagina P2,
            registo R2
        WHERE
            RP2.ativa AND P2.ativa AND R2.ativo
                AND RP2.userid = P2.userid
                AND RP2.userid = R2.userid
                AND RP2.userid = U2.userid
                AND RP2.pageid = P2.pagecounter
                AND RP2.regid = R2.regcounter)
GROUP BY user;
```

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

```
SELECT DISTINCT
    U.userid AS utilizador
FROM
    utilizador U
WHERE
        ( SELECT  /* tem de haver menos paginas que tipos de registos */
                COUNT(p.pagecounter)
            FROM
                pagina P
            WHERE
                P.userid = U.userid
                    AND P.ativa)
        <=
        ( SELECT
                COUNT(RP.pageid)
            FROM
                reg_pag RP
            WHERE
                RP.userid = U.userid
                    AND RP.ativa)

        AND EXISTS( SELECT /* esse utilizador tem tipos de registo */
                TR.typecnt
            FROM
                tipo_registo TR
            WHERE
                TR.userid = U.userid
                    AND TR.ativo)

        AND NOT EXISTS( SELECT /* nao existe um tipo de registo de um utilizador */
                TR.typecnt
            FROM
                tipo_registo TR
            WHERE
                TR.userid = U.userid
                    AND TR.ativo
                    AND NOT TR.typecnt IN( SELECT
                    /* que nao esteja entre os tipos de registo de uma pagina desse utilizador */
                            RP.typeid
                        FROM
                            reg_pag RP, registo R, pagina P
                        WHERE
                            U.userid = RP.userid
                                AND R.regcounter = RP.regid
                                AND P.pagecounter = RP.pageid
                                AND P.ativa
                                AND R.ativo
                                AND RP.ativa))
```

2 Restrições de integridade

Defina a seguinte restrição de integridade, recorrendo aos mecanismos mais apropriados para o efeito, e que estejam disponíveis no sistema MySQL:

(a) Todo o valor de *contador_sequencia* existente na relação *sequencia* existe numa e uma vez no universo das relações *tipo_registo*, *pagina*, *campo*, *registo* e *valor*.

De acordo com o que é pedido no enunciado, decidimos implementar um conjunto de 6 triggers, de forma a garantir que o *contador_sequencia* existe uma e uma só vez no conjunto de relações apresentadas. Entre entes, 5 são semelhantes (os referentes às tabelas *tipo_registo*, *pagina*, *campo*, *registo* e *valor*) e 1 um pouco diferente, devido à necessidade de comparar *contador_sequencia* com os *idseq*'s das outras relações.

Em anexo, na pasta "2 – Restricoes" encontram-se os 6 triggers acima referidos, bem como uma breve descrição, para cada um, do seu conteúdo e razão de ser.

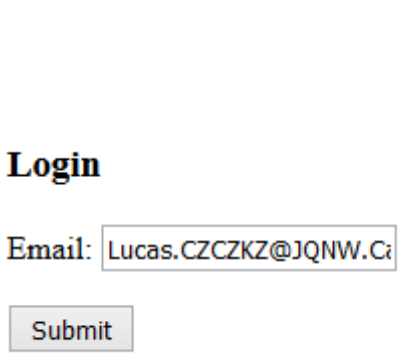
```
/* Instituicao: Instituto Superior Tecnico */
/* Curso: LEIC-A            Data: 9/12/2015*/
/* Autores: João Catarino 78877,
            Ricardo Mota 78131,
            Luis Morais 78416 */

/* Descrição: Para a tabela pagina, caso exista uma ocorrencia de idseq,
nao pode voltar a existir uma com valor equivalente entre as restantes tabelas.
Isto e, verificamos se idseq em qualquer uma das outras quatro tabelas,
para alem da tabela pagina, tem um valor igual ao idseq da tabela pagina que pretendemos acrescentar.
Caso se verifique, levanta um erro. */

Delimiter //
CREATE TRIGGER verifica_pagina BEFORE INSERT ON pagina
FOR EACH ROW
BEGIN
    IF (SELECT COUNT(*) FROM tipo_registo T, registo R, campo C, valor V
        WHERE NEW.idseq = T.idseq
        AND NEW.idseq = R.idseq
        AND NEW.idseq = C.idseq
        AND NEW.idseq = V.idseq) != 0 THEN CALL error; END IF;
end //
```

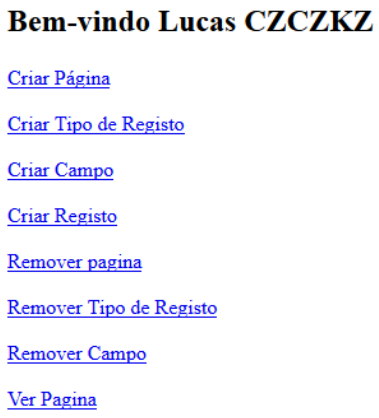
Figura 1 – Um dos triggers, neste caso referente à relação *pagina* (como forma de exemplo).

3 Desenvolvimento da aplicação

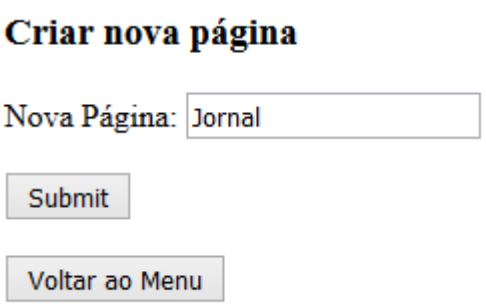
Crie um conjunto de páginas em PHP e HTML simples que permita ao utilizador:

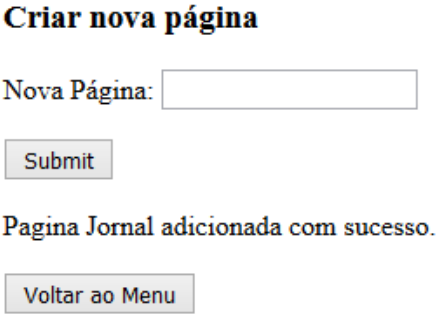
Em anexo, na pasta "3 – PHP e HTML" encontra-se o código da aplicação desenvolvida, bem como comentários ao longo do mesmo a explicar as decisões fulcrais.



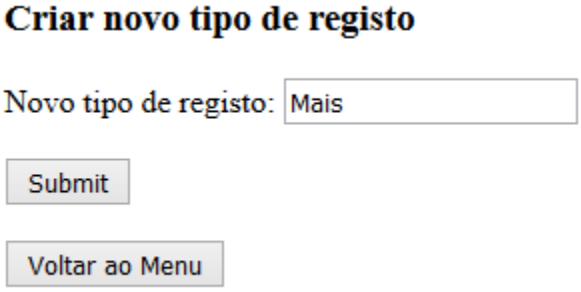


(a) Inserir uma nova página:



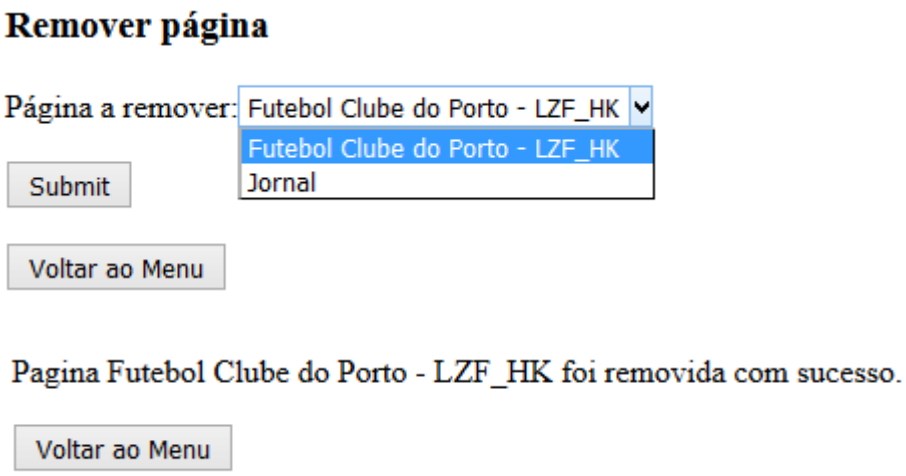


(b) Inserir um novo tipo de registo:

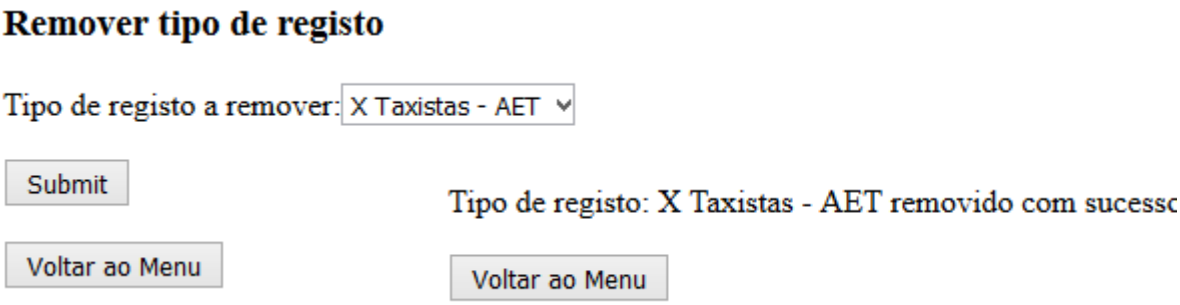


(c) Inserir novos campos para um tipo de registo:

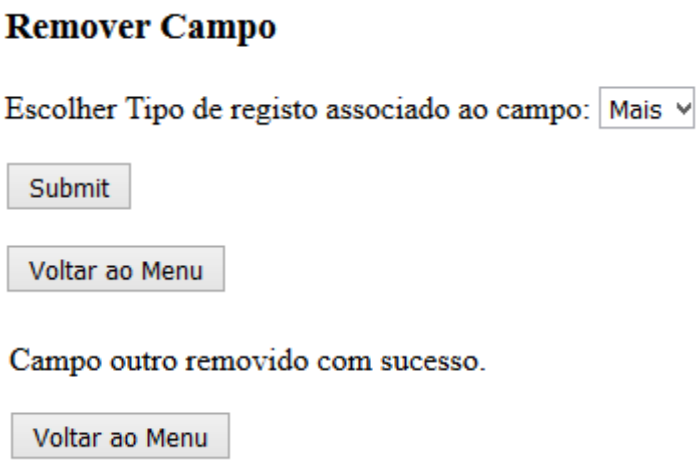
(d) Retirar uma página:

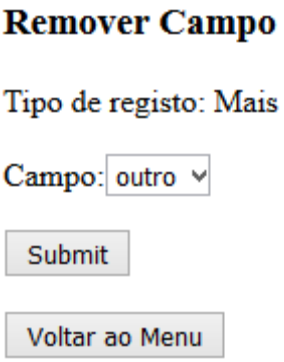


(e) Retirar um tipo de registo:

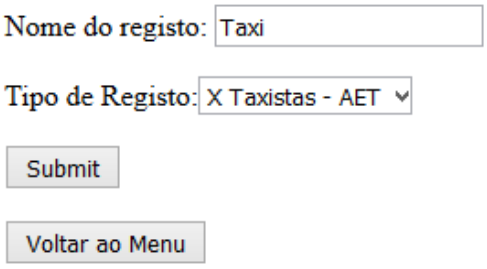


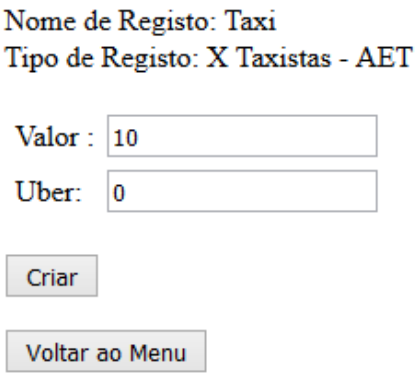
(f) Retirar um campo de um tipo de registo:



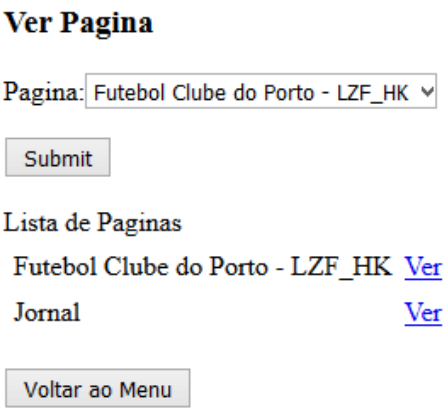


(g) Inserir um registo e os respetivos valores dos campos associados:





(h) Ver uma página com registos nela contidos:



Nome de Registo	Campo	Valor
Lívia		
Isabella		XTUD

Voltar ao Menu

4 Formas Normais

Considere a relação *utilizador*, na qual existem duas chaves candidatas (userid e email).

(a) Em que forma normal se encontra a relação *utilizador*?

Para verificar em que forma normal a relação *utilizador* se encontra, vimos, uma a uma, se as formas normais se verificavam.

Em primeiro lugar, verificou-se que a primeira forma normal (1FN) é satisfeita, pois todos os atributos da tabela são definidos com valores atómicos, isto é, a cada entrada da tabela só corresponde um valor.

Em segundo lugar, verificou-se que a segunda forma normal (2FN) era também satisfeita, visto que todos os atributos não pertencentes à chave dependem da chave através de uma dependência funcional (DF), ou seja, os atributos da relação dependem da chave como um todo.

Em terceiro lugar, verificou-se também a terceira forma normal (3FN), tendo em conta que presumimos que todos os atributos que não pertencem à chave são mutuamente independentes, portanto não existem dependências transitivas, nenhum atributo que não seja chave pode depender de outro atributo que não é chave. É possível assumir que os atributos *resposta1* e *resposta2* são dependentes dos atributos *questao1* e *questao2*. Se tal se verifica-se, a relação utilizador ficaria apenas na segunda forma normal (2FN). No entanto, assumimos que essa dependência não é verificada, tendo em conta a questão que nos é colocada a seguir (alínea b).

Por fim, verificamos a *Boyce Code normal form* (BCNF), visto que entre todas as dependências, os únicos determinantes são chaves candidatas (userid e email).

Podemos então concluir que a relação utilizador se encontra na *Boyce Code normal form* (BCNF).

(b) Considerando que existe um triggers que garante que é sempre verdade que:

nome, password, questao2, resposta2, questao1, resposta1 → email

1- Em que forma normal se encontra agora a relação *utilizador*?

2- Caso esta não se encontre na BCNF, proponha uma decomposição (sem perdas de informação) da mesma de forma a que todas as relações obtidas estejam na BCNF.

A relação *utilizador* não verifica agora a BCNF, visto que existe um determinante que não é chave candidata- conjunto (nome, password, questao2, resposta2, questao1, resposta1).

No entanto, a terceira forma normal ainda se verifica, visto que esta apenas diz que todos os atributos que **não pertencem à chave** têm de ser mutuamente independentes, condição que ainda se verifica, pois o único atributo que agora é determinado por um conjunto não chave, é um atributo que pertence à chave da relação (numa relação R, entre uma DF (X → A), A é uma chave da relação R). Isto significa que os atributos não chave continuam a ser independentes entre si. Para além da terceira forma, a segunda e primeira formas normais ainda se verificam, pelas mesmas razões apresentadas na alínea acima.

Podemos então concluir que a relação *utilizador* se encontra agora na terceira forma normal (3FN).

Como a relação *utilizador* se encontra na terceira forma normal, temos de a decompor em 2 relações (de U para U1 e U1). Teremos então as seguintes relações:

U1 (nome, password, questao2, resposta2, questao1, resposta1, email) onde o conjunto (nome, password, questao2, resposta2, questao1, resposta1) é UNIQUE e é chave candidata de U1, como um tuplo.

nome	password	questao2	resposta2	questao1	resposta1	email

U2 (userid, email, pais, categoria)

userid	email	pais	categoria

5 Índices

Suponha que as seguintes interrogações são muito frequentes no sistema:

(a) Devolver a média do número de registos por página de um utilizador,

(b) Ver o nome dos registos associados à página de um utilizador,

Indique que tipo de índice(s), sobre que atributo(s) e sobre que tabela(s) faria sentido criar de modo a acelerar a execução destas interrogações. Crie o(s) índice(s) em SQL e mostre como a execução de cada interrogação beneficiou a sua existência.

(a) Para esta interrogação foi criado a seguinte consulta em SQL:

```
SELECT DISTINCT
    (COUNT(DISTINCT RP.idregpag) / COUNT(DISTINCT P.pagecounter))
FROM
    reg_pag RP,
    pagina P,
    registo R
WHERE
    RP.ativa AND P.ativa AND R.ativo
        AND RP.userid = P.userid
        AND RP.userid = R.userid
        AND RP.userid = 'userid'
        AND RP.pageid = P.pagecounter
        AND RP.regid = R.regcounter;
```

Para afinar a execução desta interrogação, seria ideal a implementação de um índice que verificasse as seguintes características:

1- Modo de indexação baseado em função de dispersão (hash-based), por se tratar de uma interrogação por igualdades.

2- Denso, pois existe uma entrada de dados no índice para cada valor da chave de pesquisa (trata-se de um índice para *id*'s)

3- Secundário, pois a chave de pesquisa não contém uma *primary key* da tabela referente.

4- Desagrupado, devido à ordem dos registos de dados não ser equivalente à ordem das entradas de dados do índice.

5- Aplicado sobre os atributos *regid* e *pageid* da tabela *reg_pag*.

Apesar de esta ser a implementação ideal, como foi dito em cima, não é possível concretizá-la na totalidade, visto que a versão do MySQL do sigma não suporta modos de indexação hash-based. Por esta razão, concretizámos o índice mas com um modo de indexação baseado em árvore (tree-based), mais conhecido por B+ tree.

```
CREATE INDEX idx_reg_page
USING BTREE
ON reg_pag (regid, pageid);
```

Através deste índice conseguimos verificar uma diminuição do tempo de execução da interrogação, de um valor aproximado de 0.0001800s para 0.0001300s, o que é significante pois a interrogação já era bastante eficiente por si mesma (fazia apenas as visitas necessárias).

```
| 0.00018000 | select (count(distinct RP.idregpag) /count(distinct P.pagecounter))
```

Figura 2 – Tempo verificado antes de aplicar o índice acima descrito.

```
| 0.00013000 | select (count(distinct RP.idregpag) /count(distinct P.pagecounter))
```

Figura 3 – Tempo verificado depois de aplicar o índice acima descrito.

(b) Para esta interrogação foi criado a seguinte consulta em SQL:

```
SELECT
    R.nome
FROM
    registo R,
    reg_pag RP,
    pagina P
WHERE
    R.regcounter = RP.regid
        AND P.pagecounter = 'pagecounter'
        AND R.userid = 'userid'
        AND P.userid = R.userid
        AND RP.ativa
        AND R.ativo
        AND P.ativa
ORDER BY R.nome;
```

Para esta interrogação, decidimos testar o índice criado para a interrogação anterior, bem com um outro, parecido, mas composto apenas por parte do primeiro.

Para obter uma execução melhorada desta interrogação, percebemos que deveríamos ter um índice que verifica-se:

1- Modo de indexação baseado em função de dispersão (hash-based), por se tratar de uma interrogação por igualdades, tal como na alínea (a)

2- Denso, pois existe uma entrada de dados no índice para cada valor da chave de pesquisa (trata-se de um índice para *id*'s)

3- Secundário, pois a chave de pesquisa, mais uma vez, não contém uma *primary key* da tabela referente.

4- Desagrupado, mais uma vez, devido à ordem dos registos de dados não ser equivalente à ordem das entradas de dados do índice.

5- Aplicado sobre os atributos *regid* e/ou *pageid* da tabela *reg_pag*, ao contrário da alínea (a), onde foi aplicado sobre os 2 atributos ao mesmo tempo.

Mais uma vez, como já exposto em cima, não foi possível usar o modo de indexação hash-based, por não ser suportado. Foi usada então, de novo, uma variação com modo B+ Tree. Os resultados foram de novo calculados tendo em conta esta variante.

Foi testado então o índice da alínea (a) e de seguida, o mesmo índice mas sem o atributo *pageid*:

```
CREATE INDEX idx_reg_page
USING BTREE
ON reg_pag (regid);
```

Foi verificado então, que neste caso, apesar de ambos os índices darem melhorias praticamente equivalentes (como era esperado), o segundo (presente só o atributo regid), mesmo sem o atributo *pageid* melhorava a interrogação na perfeição. Então, tendo em conta que a presença dum segundo atributo como *pageid* não era necessário, decidimos que a melhor solução seria usar o índice criado apenas para este caso (não o da alínea (a)).

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	P	NULL	const	PRIMARY,userid	PRIMARY	8	const,const	1	100.00	Using temporary; Using filesort
	1	SIMPLE	R	NULL	ref	PRIMARY,userid,userid_2	PRIMARY	4	const	3	90.00	Using where
	1	SIMPLE	RP	NULL	ALL	NULL	NULL	NULL	NULL	200	9.00	Using where; Using join buffer (Block Nested Lo…

Figura 4 – "Rows" visitadas antes de aplicar o índice acima descrito (tabela gerada pelo comando *explain*)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	P	NULL	const	PRIMARY,userid	PRIMARY	8	const,const	1	100.00	Using filesort
1	SIMPLE	R	NULL	ref	PRIMARY,userid,userid_2	PRIMARY	4	const	3	90.00	Using where
1	SIMPLE	RP	NULL	ref	idx_reg_pag	idx_reg_pag	4	sys.R.regcounter	1	90.00	Using where

Figura 5 – "Rows" visitadas após aplicar o índice acima descrito (tabela gerada pelo comando *explain*)

Desta vez, não só houve um melhoramento na velocidade, como foi possível notar uma grande diminuição de "rows" visitadas.

Em relação aos tempos obtidos, através deste índice conseguimos verificar uma diminuição do tempo de execução da interrogação, de um valor aproximado de 0.000941s para 0.000155s, um salto maior do que o verificado na alínea anterior.

```
| 0.00094100 | select R.nome from registo R, reg_pag RP, pagina P
```

Figura 6 – Tempo verificado antes de aplicar o índice acima descrito.

```
| 0.00015500 | select R.nome from registo R, reg_pag RP, pagina P
```

Figura 7 – Tempo verificado após aplicar o índice acima descrito.

6 Transações

Considere que há vários programas a aceder simultaneamente à base de dados. Considere ainda que cada registo e os valores dos seus campos têm que ser inseridos de forma atómica. Reescreva o código PHP de modo a garantir esta nova funcionalidade.

As transações permitem executar um conjunto de interações de forma atómica. Esta ferramenta é particularmente útil quando são efectuados vários acessos simultâneos à mesma base de dados e queremos manter a consistência dos dados, o que é o caso neste projecto.

Deste modo, sempre que interagimos com tabelas do nosso modelo (INSERT, UPDATE, SELECT) queremos usufruir deste mecanismo.

A transformação do código PHP é feita de forma trivial, pelo que a estrutura não é modificada, apenas se adicionam algumas linhas.

Usamos "exceptions" para garantir que caso seja detetado um erro seja possível realizar a operação de "rollback" (nenhuma alteração da base de dados é realizada). No início do corpo do script de PHP damos como iniciada a transação com o comando *beginTransaction()*. Após as operações normais a realizar, efetuam-se as alterações, isto é, executa-se o comando *commit()*. Caso haja um erro e não seja executado o "commit", tratamos das excepções com o "catch", onde, como previamente referimos, chamamos o *rollBack()*.

Em anexo, na mesma pasta que para o ponto 3, (pasta "3 – PHP e HTML") encontra-se o código da aplicação desenvolvida incluindo as transações feitas ao longo do mesmo, e devidamente comentado (onde se encontram as transações).

```
try
{
        // inicia a transacao, esta linha nao é necessária para a versão sem transações
        $db->beginTransaction();

        ... código ...

        // termina a transação efectuando as mudanças desde o inicio da transação
        // esta linha nao é necessária para a versão sem transações
        $db->commit();
}
catch (PDOException $e)
{
        // feito o rollback se for apanhada alguma exceção durante a transação
        // esta linha não é necessária para a versão sem transações
        $db->rollBack();
        echo("<p>ERROR: {$e->getMessage()}</p>");
}
```

Figura 8 – Exemplo de uma transação.

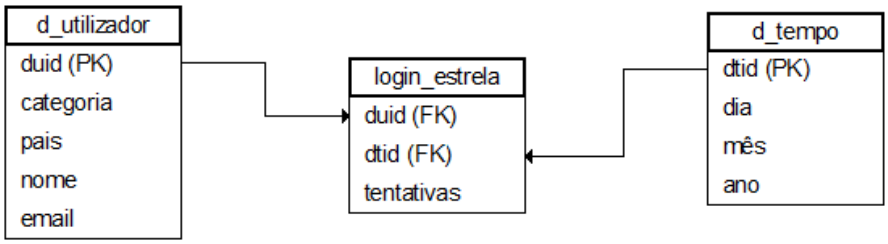
7 Data Warehouse

(a) Crie na base de dados o esquema de uma estrela com informação de número de tentativas de login tendo como dimensões: *d_utilizador(email, nome, pais, categoria)* e *d_tempo(dia, mes, ano)*. Escreva as instruções SQL necessárias para carregar o esquema em estrela a partir das tabelas existentes.

Em primeiro lugar, começámos por criar as tabelas para a data warehouse, como é possível verificar no código SQL apresentado abaixo.

```
CREATE TABLE IF NOT EXISTS d_utilizador (
    duid INT NOT NULL AUTO_INCREMENT,
    email VARCHAR(255) NOT NULL,
    nome VARCHAR(255) NOT NULL,
    pais VARCHAR(45) NOT NULL,
    categoria VARCHAR(45) NOT NULL,
PRIMARY KEY (duid)
);

CREATE TABLE IF NOT EXISTS d_tempo (
    dtid INT NOT NULL AUTO_INCREMENT,
    dia TINYINT(2) NOT NULL,
    mes TINYINT(2) NOT NULL,
    ano SMALLINT NOT NULL,
PRIMARY KEY (dtid)
);

CREATE TABLE IF NOT EXISTS login_estrela (
    duid INT NOT NULL,
    dtid INT NOT NULL,
    tentativas INT NOT NULL,
PRIMARY KEY (duid, dtid),
FOREIGN KEY (duid) REFERENCES d_utilizador (duid),
FOREIGN KEY (dtid) REFERENCES d_tempo (dtid)
);
```

Este trecho de código segue um modelo em estrela, como pedido no enunciado. Tivemos em consideração o seguinte esquema:



Foram adicionados índices, como *primary key*, em cada dimensão (*duid* na tabela *d_utilizadores* e *dtid* na tabela *d_tempo*) de modo a identificar cada entrada. A tabela factual *login_estrela* usa a medida de *tentativas* que representa o número de tentativas que determinado utilizador fez (FK *duid*) num determinado dia (FK *dtid*).

De forma a popular as tabelas do esquema estrela criado acima, foi desenvolvido um script que apresentamos abaixo:

```
DELIMITER ;
-- desactiva a verificacao das chaves estrangeiras
SET foreign_key_checks = 0 ;

TRUNCATE TABLE login_estrela;
TRUNCATE TABLE d_utilizador;
TRUNCATE TABLE d_tempo;

SET foreign_key_checks = 1 ;

START TRANSACTION;

INSERT INTO d_utilizador (email, nome, pais, categoria)
SELECT email, nome, pais, categoria FROM utilizador;

INSERT INTO d_tempo (dia, mes, ano)
SELECT DISTINCT DAY(moment), MONTH(moment), YEAR(moment)
FROM login;

INSERT INTO login_estrela (duid, dtid, tentativas)
SELECT DU.duid, DT.dtid, COUNT(sucesso)
FROM login L, utilizador U, d_utilizador DU, d_tempo DT
WHERE
    L.userid = U.userid
        AND U.email = DU.email
        AND DAY(L.moment) = DT.dia
        AND MONTH(L.moment) = DT.mes
        AND YEAR(L.moment) = DT.ano
GROUP BY L.moment , L.userid;


COMMIT;
```

Como notamos, todos os utilizadores da base de dados são carregados na tabela *d_utilizador* (e consequentemente na tabela *login_estrela* visto que esta é carregada com o auxílio da anterior), e não só os que realizaram login.

(b) Considerando o esquema da estrela criado em (a), escreva a interrogação em MySQL para obter a média de tentativas de login para todos os utilizadores de Portugal, em cada categoria, com rollup por ano e mês.

```
SELECT
    DU.categoria AS Categoria,
    DT.ANO AS Ano,
    DT.MES AS Mes,
    SUM(LE.tentativas) / COUNT(LE.tentativas) AS Média
FROM
    login_estrela LE,
    d_utilizador DU,
    d_tempo DT
WHERE
    LE.duid = DU.duid AND LE.dtid = DT.dtid
        AND DU.pais = 'Portugal'
GROUP BY DU.categoria , DT.ano , DT.mes WITH ROLLUP;
```