

Projeto de Bases de Dados

Parte 3

1ºSemestre - 2016-2017

Grupo 5

Turno 4ª Feira às 8h - BD225179L06

Pedro Orvalho nº 81151

Ana Leitão nº 81365

Manuel Galamba nº 81647

Horas de trabalho: 16 horas de trabalho por elemento

SQL

a) Quais os espaços com postos que nunca foram alugados?

Esta query cria a tabela **aceites** que contém o número das reservas com estado = 'Aceite' e junta-a pelo número de reserva à tabela **aluga**, criando uma tabela com o código e morada de todos os alugáveis cujo estado é aceite (portanto os que foram alugados). São depois seleccionados os postos da tabela **posto** que não se encontram na tabela descrita anteriormente, criando uma tabela com a morada e o código dos espaços (os espaços com postos que nunca foram alugados).

```
SELECT DISTINCT morada, codigo_espaco AS codigo
FROM posto
WHERE (codigo, morada) NOT IN (
  SELECT codigo, morada
  FROM aluga NATURAL JOIN (SELECT numero
                           FROM estado
                           WHERE estado = "Aceite") aceites
);
```

b) Quais edifícios com um número de reservas superior à média?

Primeiro contabiliza-se o número de reservas feitas para cada morada (ou seja cada edifício) na tabela **aluga** sendo depois feita a média de todos estes valores. De seguida são seleccionados todos os edifícios cujo número de reservas é maior que a média.

```
SELECT morada
FROM aluga
GROUP BY morada
HAVING COUNT(*) > (SELECT AVG(num_reservas) AS media_moradas
                   FROM (SELECT morada, COUNT(*) AS num_reservas
                         FROM aluga
                         GROUP BY morada) contagem_moradas);
```

c) Quais utilizadores cujos alugáveis foram fiscalizados sempre pelo mesmo fiscal?

É feito um NATURAL JOIN entre as tabelas **arrenda** e **fiscaliza** e são depois seleccionados os nif's dos utilizadores que foram fiscalizados sempre por um fiscal com o mesmo id, isto é, contamos o número de id's distintos para cada nif, e só seleccionamos os nif's que tiverem um contador igual a 1.

```
SELECT nif
FROM arrenda NATURAL JOIN fiscaliza
GROUP BY nif
HAVING COUNT(DISTINCT id) = 1;
```

d) Qual o montante total realizado (pago) por cada espaço durante o ano de 2016?

São criadas duas tabelas, **oferta_montante** e **uniao_postos_espacos**:

- **oferta_montante** - esta tabela contém os atributos morada, código e montante (tarifa diária * número de dias da oferta) das reservas que foram pagas em 2016;
- **uniao_postos_espacos** - contém os atributos morada, código e codigo_espaco de todos os alugáveis. A união da tabela **posto** com a tabela **espaco** é tornada

possível adicionando a coluna `codigo_espaco` (que contém o código do espaço) à tabela **espaco**;

É depois feita uma junção das duas e é somado o montante realizado por cada espaço, pois a soma é feita entre as entradas com morada e `codigo_espaco` igual. A tabela resultante desta junção (denominada `t`) é unida à tabela **espaco** com uma coluna extra de nome `montante-total`, com o valor zero em todas as linhas. O resultado da query é a tabela com a morada e o código de todos os espaços e com a soma do montante total realizado para cada um destes e os seus postos.

```
SELECT morada, codigo, SUM(montante_total) as total
FROM
  (SELECT morada, codigo, 0 as montante_total FROM espaco
  UNION
  (SELECT morada, codigo_espaco as codigo, SUM(montante) AS montante_total
  FROM (SELECT morada, codigo, tarifa*DATEDIFF(data_fim,data_inicio) AS montante
        FROM (SELECT numero
              FROM paga
              WHERE YEAR(data) = "2016") pagas_2016 NATURAL JOIN aluga NATURAL JOIN oferta
        ) oferta_montante NATURAL JOIN (SELECT morada, codigo, codigo_espaco
        FROM posto UNION (SELECT morada, codigo, codigo AS codigo_espaco
                          FROM espaco)
        ) uniao_postos_espacos
  )
  )
GROUP BY morada, uniao_postos_espacos.codigo_espaco)) t
GROUP BY morada, codigo;
```

e) Quais os espaços de trabalho cujos postos nele contidos foram todos alugados?

É utilizada a query a) para seleccionar os espaços com postos que nunca foram alugados e são só devolvidos os espaços que não estão na tabela que resulta dessa query, logo é retornada uma tabela com os espaços em que todos os postos foram alugados.

```
SELECT DISTINCT morada, codigo_espaco AS codigo
FROM posto
WHERE (morada, codigo_espaco) NOT IN (
  SELECT DISTINCT morada, codigo_espaco
  FROM posto
  WHERE (codigo, morada) NOT IN (
    SELECT codigo, morada
    FROM aluga NATURAL JOIN (SELECT numero
                          FROM estado
                          WHERE estado = "Aceite") aceites
  ));
```

Triggers

a) RI-1: "Não podem existir ofertas com datas sobrepostas"

Para cumprir esta restrição de integridade optamos por implementar um trigger, accionado antes de ser inserida uma nova oferta na tabela **oferta**. Caso exista uma oferta para o mesmo alugável que o da nova entrada com datas iniciais e finais que se sobreponham às da oferta que se quer inserir (data final da nova superior à inicial da outra e data inicial da nova inferior à final da outra) é chamada uma função de erro que interrompe e impede a inserção da oferta na tabela.

```

DELIMITER //
DROP TRIGGER IF EXISTS update_oferta;
CREATE TRIGGER update_oferta BEFORE INSERT ON oferta
FOR EACH ROW
BEGIN
    IF (EXISTS (SELECT data_inicio FROM oferta o
                WHERE o.codigo = NEW.codigo AND o.morada = NEW.morada
                AND NEW.data_fim > o.data_inicio AND NEW.data_inicio < o.data_fim))
    THEN CALL interseccao_datas();
END IF;
END //
DELIMITER ;

```

b) RI-2: "A data de pagamento de uma reserva paga tem de ser superior ao timestamp do último estado dessa reserva"

Esta restrição de integridade foi implementada através de um trigger que é acionado antes da inserção de valores na tabela **paga**.

Este trigger consiste na chamada de uma função de erro que interrompe e impede a inserção de uma nova entrada na tabela **paga** quando existe uma entrada na tabela **estado** para a mesma reserva da qual se estão a inserir dados e o timestamp dessa é maior que a data de pagamento da nova entrada.

```

DELIMITER //
DROP TRIGGER IF EXISTS update_paga;
CREATE TRIGGER update_paga BEFORE INSERT ON paga
FOR EACH ROW
BEGIN
    IF (EXISTS (SELECT numero FROM estado e
                WHERE e.numero = NEW.numero AND e.time_stamp > NEW.data) )
    THEN CALL pagamento_data_incorrecta();
END IF;
END //
DELIMITER ;

```

Desenvolvimento da Aplicação

Para explicar a nossa aplicação iremos apresentar um modelo geral de inserção e remoção de entradas nas tabelas que se aplicam a vários dos pontos pedidos

A aplicação é acedida através do endereço <http://web.ist.utl.pt/ist181151/inicio.php> que direcciona à página inicial. A imagem à direita é um *screenshot* desta página que possui as funções da nossa aplicação com correspondência aos pontos pedidos no enunciado. Todo o código relacionado diretamente com a base de dados é executado dentro de um *try* com um *catch* de excepções no final.

Inicio	
Adicionar ou Remover Edificios	a)
Adicionar ou Remover Espacos	a)
Adicionar ou Remover Postos	a)
Adicionar ou Remover Oferta	b)
Criar Reserva	c)
Pagar Reserva	d)
Total realizado pelos Edificios	e)

Todos os ficheiros .php que acedem à base de dados iniciam a secção de php com o código na imagem à direita.

```
$host = "db.ist.utl.pt";
$user = "ist181151";
$password = "bd2016";
$dbname = $user;
$db = new PDO("mysql:host=$host;dbname=$dbname", $user, $password);
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Inserção de dados nas tabelas

```
try
{
    (...) código para aceder à base de dados

    $db->query("start transaction;");
    $sql = "INSERT INTO alugavel (morada, codigo, foto) VALUES (?, ?, ?)";
    $stmt = $db->prepare($sql);
    $stmt->execute(array($morada, $codigo, $foto));
    $db->query("commit;");

    $db->query("start transaction;");
    $sql = "INSERT INTO espaco (morada, codigo) VALUES (?, ?)";
    $stmt = $db->prepare($sql);
    $stmt->execute(array($morada, $codigo));
    $db->query("commit;");

    $db = null;
    echo("<p>Espaco adicionado com sucesso!</p>");
}
```

Este excerto de código corresponde à inserção de um novo espaço na tabela **espaco**.

Começamos por escrever um “sql statement template” para inserir um espaco/posto (para a inserção de edifícios o procedimento é o mesmo, sendo que apenas é feita a inserção na tabela **edificio**), de seguida este é enviado para a base de dados mas os argumentos não são especificados (?). Depois a base de dados compila, faz uma optimização ao “statement” e guarda o resultado sem o executar.

Ao fazermos “\$stmt->execute” a aplicação faz uma conexão entre os valores e os parâmetros não especificados anteriormente.

Remoção de entradas das tabelas

Este excerto de código corresponde à remoção de um edifício da tabela **edificio**.

Aqui também são feitos “prepare statements” como na inserção nas tabelas, a única diferença é que a query em vez de inserir vai remover da tabela em questão.

```
try
{
    (...) código para aceder à base de dados

    $db->query("start transaction;");
    $sql = "DELETE FROM edificio WHERE morada=?";
    $stmt = $db->prepare($sql);
    $stmt->execute(array($morada));
    $db->query("commit;");

    $db = null;
    echo("<p>Edificio removido com sucesso!</p>");
}
```


Pagar Reserva

Escolhendo a opção pagar reserva é apresentada uma tabela com todas as reservas com estado “Aceite” (imagem à direita) (estas são seleccionadas através de uma query sql simples). Depois de escolhida a oferta, a base de dados é alterada através de um insert idêntico ao explicado no ponto “Inserção de dados nas tabelas”.

Pagar Reserva

Morada Codigo

ISEL	DEI	2016-01-01	103246782	2016-12	Pagar
IST	DEI	2016-02-01	120456781	2016-5	Pagar

[Voltar para o inicio](#)

Mostrar o total realizado por cada espaço

Esta funcionalidade é feita recorrendo à query do ponto d) do SQL escolhendo só os espaços de uma determinada morada de um edifício, sendo este inicialmente escolhido pelo utilizador. Os valores que esta devolve são impressos numa tabela juntamente com o código de cada espaço (imagem à direita).

Total por cada espaço do Edifício

Central 510.0000

DG 0.0000

DMKT 180.0000

[Voltar para Totais realizados pelos Edifícios](#) [Voltar para o inicio](#)

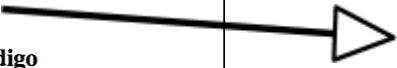
Formato Geral da Aplicação

A imagem em baixo mostra os menus dos postos (igual para espaços/edifícios/ofertas).

Postos
[Adicionar Posto](#)

Morada	Codigo	
Catolica	Sala1	Remover Posto
Catolica	Sala2	Remover Posto
FEUP	Lab1	Remover Posto
FEUP	Lab2	Remover Posto
FEUP	Lab3	Remover Posto
FEUP	Lab4	Remover Posto
IST	Lab1	Remover Posto
IST	Lab2	Remover Posto
IST	Lab3	Remover Posto

[Voltar para o inicio](#)



Escolha o qual o espaço onde quer o seu posto

Morada	Codigo	
Catolica	Central	Escolher Espaco
Catolica	DG	Escolher Espaco
Catolica	DMKT	Escolher Espaco
FEUP	Central	Escolher Espaco
FEUP	DEG	Escolher Espaco
FEUP	DEI	Escolher Espaco
FEUP	DEQ	Escolher Espaco
ISEL	Central	Escolher Espaco
ISEL	DEG	Escolher Espaco
ISEL	DEI	Escolher Espaco
ISEL	DEQ	Escolher Espaco
IST	Central	Escolher Espaco
IST	DEG	Escolher Espaco
IST	DEI	Escolher Espaco
IST	DEQ	Escolher Espaco

[Voltar para o inicio](#)

A opção Adicionar Posto (igual para Espaço/Oferta) direcciona a um menu onde o utilizador escolhe o espaço onde o quer inserir (imagem em cima à direita) (é equivalente para espaços ou ofertas).

Após ser escolhido o espaço, é pedido ao utilizador para preencher os restantes atributos de um posto (imagem à direita) (é equivalente para espaços/ofertas/edifícios).

Escolha o codigo e fotografia do seu posto

Codigo:

Fotografia:

Schema.sql e Populate.sql

Em relação ao ficheiro schema.sql, acrescentamos em todas as tabelas nas foreign keys “ON UPDATE CASCADE”, o que significa que caso estas chaves sejam alteradas a alteração propaga-se para as tabelas que utilizam estas chaves.

Relativamente ao ficheiro populate.sql utilizamos o disponibilizado pelos professores da cadeira.