Projeto Bases De Dados,

Parte 3

Realizado por:

Francisco Duarte, Nº 78682

João Silvestre, Nº80996

Gonçalo Soares, Nº81418

Grupo Nº7, Turno Segunda 8:00h

O esforço, em horas, despendido na realização

deste trabalho por aluno foi: 10 horas

Criação da Base de Dados

Foi criado uma base de dados tendo como base o modelo relacional fornecido, e populada usando o script fornecido pelos docentes.

SQL

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

De todos os espaços são selecionados aqueles cujos postos não têm um sequer um estado "Aceite".

```
select distinct morada, codigo_espaco
from posto
where (morada, codigo, codigo_espaco) not in (
  select morada, codigo, codigo_espaco
  from aluga natural join estado natural join posto
  where estado = "Aceite"
);
```

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

Foi feita uma contagem de qual o número de reservas de cada edificio, e são selecionados os casos em que este número é superior à média de reservas calculada .

```
select morada
from aluga
group by morada
having count(*) > (
  select avg(reserveCount)
  from (
    select count(*) as reserveCount
    from aluga
    group by morada
  ) as avgReserve
);
```

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

Foram selecionados todos os users em que a contagem de id's de fiscais é igual a um.

```
select nif
from fiscaliza natural join arrenda
group by nif
having count(distinct id) = 1;
```

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

Foi calculado o montante para cada posto e para cada espaço, multiplicando a tarifa pela diferença entre a data de fim e inicio de aluguer. De seguida foram agrupados pelo morada e código de espaço fazendo a soma dos respetivos montantes.

```
select morada, codigo, sum(montante)
from(
  select morada, codigo, montante
  from (
    select morada, codigo as codigo_posto, codigo_espaco as codigo, tarifa*datediff(data_fim,data_inicio) as montante
    from aluga natural join estado natural join oferta natural join posto
    where estado = "Paga" and year(time_stamp) = 2016
  ) postos_pagos
  union
  select morada, codigo, tarifa*datediff(data_fim,data_inicio) as montante
  from aluga natural join estado natural join oferta natural join espaco
  where estado = "Paga" and year(time_stamp) = 2016
) tabela_montantes
group by morada, codigo;
```

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

Foi calculado o número de postos com ofertas aceites que cada espaço tem e de seguida é comparado com o número total de postos por espaço. São selecionados todos os espaços em que estes dois valores são iguais.

```
select morada, codigo_espaco
from(
  (select morada, codigo_espaco, count(*) as count_postos
  from posto
  group by morada, codigo_espaco) as countPostos
  natural join
  (select morada, codigo_espaco, count(*) as count_aceites
  from (
    select *
    from aluga natural join estado
    where estado = "Aceite") as aceites natural join posto
  group by morada, codigo_espaco) as countAceites
)
where count_postos = count_aceites;
```

Restrições de Integridade

Para respeitar as restrições de integridade impostas foram usados triggers.

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

Sempre que se tenta inserir uma nova oferta é verificado se a data de inicio e fim da nova oferta não se sobrepõem, caso isto aconteça é chamada uma função que não existe com a finalidade de parar a execução e lançar uma mensagem de erro (nome da função).

```
create trigger dataSobreposta before insert on oferta
for each row
begin
  if exists (select * from oferta
    where (morada = new.morada and codigo = new.codigo and (new.data_inicio < data_fim and new.data_fim > data_inicio)))
  then
    call my_capacity_to_insert_this_Offer;
  end if;
end; //
```

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

Sempre que se tenta efetuar um novo pagamento sobre uma reserva é verificado se a data de pagamento (time_stamp) é superior ao ultimo estado dessa reserva. Caso isto aconteça é chamada uma função que não existe com a finalidade de parar a execução e lançar uma mensagem de erro (nome da função).

```
create trigger dataPagamentoInvalido before insert on paga
for each row
begin
  if new.data <= (select max(time_stamp) from estado where numero=new.numero)
  then
    call my_capacity_to_pay_this;
  end if;
end; //
```

Aplicação

Na resolução desta parte do projeto dividimos todas as ações possíveis em dois ficheiros PHP distintos: um formulário e um script.

O formulário pede os dados necessários e o chama o script. Seguem todos a estrutura seguinte:

```
<html>
    <body>
        <h3>Criar novo Edificio</h3>
        <form action="new_Edificio_script.php" method="post">
            <p><input type="hidden"></p>
            <p>Nova morada: <input type="text" name="morada"/></p>
            <p><input type="submit" value="Submit"/></p>
        </form>
        <form action="../index.html">
          <input type="submit" value="Voltar" />
        </form>
    </body>
</html>
```

Têm um header a explicar qual a ação que está a ser realizada, o script que será executado após o envio do formulário, o(s) input(s) necessário(s) e um botão para retroceder ao menu principal.

Os scripts seguem também uma estrutura similar entre si. Um início onde regista os dados obtidos no formulário em variáveis, uma secção dentro de um try, constituida pela inicialização do handler (igual em todos), a criação da query a ser feita, a realização da query e a escrita no html do resultado da query e por fim uma secção de catch, onde apanhamos todas as excepções que possam ter acontecido na criação do handler ou execução da query e escrevemos uma mensagem de erro para o utilizador. No fim tem ainda um botão para retroceder à página principal.

Eliminar Posto ou Espaço (Completo):

```
<html>
    <body>
<?php

    $morada = $_REQUEST['morada'];
    $codigo = $_REQUEST['codigo'];

    try
    {
        $host = "db.ist.utl.pt";
        $user ="ist180996";
        $password = "banana";
        $dbname = $user;
        $db = new PDO("mysql:host=$host;dbname=$dbname", $user, $password);
        $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $db->query("start transaction;");


        $sql = "DELETE FROM alugavel where morada = '$morada' and codigo = '$codigo';";

        $db->query($sql);

        $db->query("commit;");

        echo("<p>Sucesso</p>");

        $db = null;
    }
    catch (PDOException $e)
    {
        $db->query("rollback;");
        echo("<p>Ocorreu um erro na base de dados com o codigo: {$e->getMessage()}</p>");
    }
?>
      <form action="../index.html">
        <input type="submit" value="Voltar" />
      </form>
    </body>
</html>
```

Nota: Para manter o número de páginas apropriado é-nos impossível apresentar as soluções a todas os itens neste relatório. Optámos portanto por fazer uma descrição das soluções por nós desenvolvidas, sendo o código concreto enviado em anexo.