Instituto Superior Técnico

Projeto de Bases de Dados

Parte 3

Eduardo Janicas: 78974 (21H)

Diana Antunes: 82448 (30H)

Nuno Fernande: 80774 (21H)

Número de grupo: 44

Turno: BD225179L06

Criação da Base de Dados

Para a criação da Base de Dados usámos dois ficheiros fornecidos pelos professores, o schema.sql e o populate.sql que foi utilizado para popular a base de dados.

SQL

a)

SELECT DISTINCT morada, codigo\_espaco

FROM posto p

WHERE NOT EXISTS (

SELECT morada

FROM aluga a

WHERE p.codigo = a.codigo

);

Selecionámos a morada e o código de espaço de todos os postos exceto dos que já foram ou estão alugados, ou seja, na tabela aluga. Para tal usámos uma sub-query que seleciona a morada dos postos alugados. Ficando apenas com os espaços que têm postos que nunca foram alugados.

b)

SELECT morada

FROM aluga NATURAL JOIN reserva

GROUP BY morada

HAVING COUNT(morada) >= (

SELECT AVG(temp.count)

FROM (

SELECT COUNT(morada) as count

FROM aluga NATURAL JOIN reserva

GROUP BY morada) as temp

);

Selecionámos a morada dos edifícios onde, por morada, verificámos que o número de reservas é superior à média de número de reservas de todos os edifícios. Para tal usámos uma sub-query que seleciona o número de vezes que a morada se encontra reservada e comparamos esse número com a média de modo a ficar com os edifícios com maior número de reservas que a média.

c)

SELECT nif, nome

FROM user

NATURAL JOIN(

SELECT nif

FROM aluga NATURAL JOIN fiscaliza

GROUP BY nif

HAVING COUNT(DISTINCT id) = 1

) AS temp;

Selecionámos o nif e o nome do utilizador cujos alugáveis foram fiscalizados sempre pelo mesmo fiscal. Para tal usámos uma sub-query que para cada nif de utilizador seleciona-o se tiver apenas um id de fiscal associado.

d)

SELECT morada, codigo\_espaco, pago

FROM

(SELECT morada, codigo\_espaco, sum((datediff(data\_fim, data\_inicio))\*tarifa)

as pago

FROM oferta NATURAL JOIN aluga NATURAL JOIN paga NATURAL JOIN posto

WHERE (data between '2016-01-01 00:00:01' and '2016-12-31 23:59:59')

GROUP BY codigo\_espaco, morada) as temp

UNION

(SELECT morada, codigo as codigo\_espaco, sum((datediff(data\_fim, data\_inicio))\*tarifa) as pago

FROM oferta NATURAL JOIN aluga NATURAL JOIN paga NATURAL JOIN espaco

WHERE (data between '2016-01-01 00:00:01' and '2016-12-31 23:59:59')

GROUP BY codigo\_espaco, morada);

Selecionámos em ambas as sub-queries a morada, código de espaço e soma da tarifa total realizada por cada um dos espaços em 2016. A primeira sub-query funciona para espaços e a segunda para postos. Fizemos um UNION entre ambas, o que permitiu juntar todos os valores numa só tabela o que permite com que possamos selecionar a morada, código de espaço e montante total realizado durante o ano de 2016 pelo espaço.

e)

SELECT morada, codigo\_espaco

FROM (SELECT morada, codigo\_espaco, COUNT(codigo) as count

FROM posto

GROUP BY morada, codigo\_espaco) as temp

NATURAL JOIN

(SELECT aceite.morada as morada, aceite.codigo\_espaco as codigo\_espaco, count(aceite.codigo) as count

FROM (SELECT DISTINCT morada, codigo\_espaco, codigo

FROM posto NATURAL JOIN aluga NATURAL JOIN estado

WHERE estado = 'aceite') AS aceite

GROUP BY morada, codigo\_espaco) as total\_aceite;

Selecionámos a morada e código\_espaco do NATURAL JOIN de duas sub-queries de forma a obter os espaços de trabalho cujos postos nele contidos foram todos alugados. Começámos por criar uma sub-query que resulta numa tabela com as colunas morada, código de espaço e número de postos que o espaço contém. De seguida criámos outra tabela usando uma sub-query com as colunas morada, código de espaço e número de postos que o espaço contém que já tiveram um estado aceite (já foram alugados). Como ambas as tabelas têm os mesmos nomes das colunas fazemos NATURAL JOIN entre as duas de forma a obter as linhas em que o número de postos é igual ao número de postos que já tiveram postos aceites.

Restrições de Integridade

a)

DELIMITER //

DROP TRIGGER IF exists overlapping\_offers;

CREATE TRIGGER overlapping\_offers

BEFORE INSERT ON oferta

FOR EACH ROW

BEGIN

IF exists(SELECT \* FROM oferta WHERE codigo = NEW.codigo AND morada = NEW.morada

AND ((new.data\_inicio >= data\_inicio AND new.data\_inicio <= data\_fim)

OR (new.data\_fim >= data\_inicio AND new.data\_fim <= data\_fim)

OR (new.data\_inicio <= data\_inicio AND new.data\_fim >= data\_fim)))

THEN call erro\_overlapping\_offers();

END IF;

END //

DELIMITER ;

Com esta restrição fazemos com que não seja possível adicionar ofertas para o mesmo alugável com datas sobrepostas. Para definir esta restrição utilizámos um TRIGGER.

b)

DELIMITER //

DROP TRIGGER IF exists pay\_date\_check;

CREATE TRIGGER pay\_date\_check

BEFORE INSERT ON paga

FOR EACH ROW

BEGIN

IF exists(SELECT \* FROM estado WHERE numero = NEW.numero AND NEW.data <= time\_stamp)

THEN call erro\_pay\_date\_check();

END IF;

END //

DELIMITER ;

Com esta restrição fazemos com que a data de pagamento de uma reserva paga não seja superior ao timestamp do último estado dessa reserva. Para defenir esta restrição utilizámos um TRIGGER.

Aplicação

Separámos a aplicação em duas partes:

- Edifícios, onde se pode inserir e remover Edifícios, Espaços e Postos de trabalho; e também se pode listar o total realizado por Espaço de trabalho.

- Ofertas, onde se pode criar e remover Ofertas, criar Reservas sobre Ofertas e pagar Reservas.

Utilizamos prepared statements na maior parte do código php, com bindParam para evitar SQL Injection e tornar a aplicação mais eficiente.

Utilizamos os métodos Post, Get e a superglobal \_SESSION para armazenar e passar variáveis entre ficheiros.

Sempre que actualizamos a base de dados, tanto no ficheiro insert.php como no remove.php, utilizamos transacções, que executam rollback caso a actualização não seja bem sucedida.

Edifícios

Criar Edifício:

INSERT INTO edificio (morada) VALUES(:morada)

Remover Edifício:

-Remover Alugáveis contidos no Edifício (codigo NULL):

-Remover Postos contidos no Edifício (codigo e codigo\_espaco NULL):

DELETE FROM posto WHERE morada = :morada AND (:codigo IS NULL OR codigo = :codigo)

AND (:codigo\_espaco IS NULL OR codigo\_espaco = :codigo\_espaco);

-Remover Espaços de Trabalho contidos no Edifício (codigo NULL):

DELETE FROM espaco WHERE morada = :morada AND (:codigo IS NULL OR codigo = :codigo);

-Remover Arrenda relacionados com o Edifício (codigo NULL):

-Remover Fiscaliza relacionados com o Edifício (codigo NULL):

DELETE FROM fiscaliza WHERE morada = :morada AND (:codigo IS NULL OR codigo = :codigo);

DELETE FROM arrenda WHERE morada = :morada AND (:codigo IS NULL OR codigo = :codigo);

-Remover Ofertas relacionados com o Edifício (codigo NULL e data\_inicio NULL):

-Remover Aluga relacionados com o Edifício (codigo e data\_inicio NULL):

DELETE FROM aluga WHERE (morada = :morada

AND (:codigo IS NULL OR codigo = :codigo)

AND (:data\_inicio IS NULL OR data\_inicio = :data\_inicio));

DELETE FROM oferta WHERE (morada = :morada

AND (:codigo IS NULL OR codigo = :codigo)

AND (:data\_inicio IS NULL OR data\_inicio = :data\_inicio));