Information Systems and Databases

PROJECT ASSIGNMENT

PART II

Group 32

75572 - Orlando Bastos Vaz

75637 - Inês de Miranda de Matos Lourenço

75988 - João Pedro Beirão

SQL instructions to create the database

DROPS:

Esta secção permite eliminar quaisquer tabelas, funções ou *triggers* existentes, com nomes semelhantes aos necessários à base de dados. É útil em sucessivas chamadas da *source projetoGroup32.sql* e a ordem de eliminação das tabelas deve ser primeiro as dependentes (com *foreign keys*) e apenas depois as independentes.

```
drop function if exists regions_overlapping;
drop trigger if exists check_study;

drop table if exists region;
drop table if exists element;
drop table if exists series;
drop table if exists study;
drop table if exists equipment;
drop table if exists request;
drop table if exists appointment;
drop table if exists doctor;
drop table if exists patient;
```

SQL INSTRUCTIONS TO CREATE THE DATABASE:

Nesta secção é criada a base de dados com as tabelas necessárias. Nos comentários "*%*" significa que a coluna é *primary key*.

```
-- int - integer from -2147483648 to 2147483647
-- varchar(n) - variable-size string with maximum n characters
-- numeric(p,d) - p total digits, d fractional digits
-- date - 'YYYY-MM-DD', YYYY - year, MM - month, DD - day
-- datetime - 'YYYY-MM-DD HH:MM:SS', HH - hour, MM - minutes, SS - seconds
```

patient_id definiu-se do tipo 'P-XXX', por isso é uma *string*, nome do paciente é uma *string*, data de nascimento definiu-se do tipo *"date"* (as horas não são relevantes) e a morada do paciente é uma *string*.

```
-- patient(*patient_id*, name, birthday, address)
create table patient
   (patient_id  varchar(255),
    name_pat  varchar(255),
    birthday  date,
    address  varchar(255),
    primary key(patient_id));
```

doctor_id definiu-se do tipo 'D-XXX', por isso é uma *string*, o nome do médico é uma *string* e a especialidade do médico é uma *string*.

```
-- doctor(*doctor_id*, name, specialty)
create table doctor
    (doctor_id  varchar(255),
     name_doc  varchar(255),
     specialty   varchar(255),
     primary key(doctor_id));
```

date_ap corresponde à data e hora da consulta, uma vez que pode haver uma consulta com os mesmos intervenientes, no mesmo dia, no entanto, têm de ser a horas diferentes; e *office* é do tipo 'XY' em que X é uma letra e Y um número inteiro, logo é uma *string*.

```
-- appointment(*patient_id*, *doctor_id*, *date*, office)
--    patient_id: FK(patient)
--    doctor_id: FK(doctor)
create table appointment
    (patient_id  varchar(255),
     doctor_id   varchar(255),
     date_ap  datetime,
     office  varchar(255),
     primary key(patient_id, doctor_id, date_ap),
     foreign key(patient_id) references patient(patient_id),
     foreign key(doctor_id) references doctor(doctor_id));
```

request_number definiu-se do tipo 'R-XXX' e, por isso, é uma *string*.

```
-- request(*request_number*, patient_id, doctor_id, date)
--    patient_id, doctor_id, date: FK(appointment)
create table request
    (request_number  varchar(255),
     patient_id  varchar(255),
     doctor_id  varchar(255),
     date_ap  datetime,
     primary key(request_number),
     foreign key(patient_id, doctor_id, date_ap) references
appointment(patient_id, doctor_id, date_ap));
```

manufacturer é uma marca e, por isso é uma *string* e *serial_number* e *model* são conjuntos de caracteres que podem incluir letras e números, por isso são também *strings*.

```
-- equipment(*manufacturer*, *serial_number*, model)
create table equipment
    (manufacturer  varchar(255),
     serial_number  varchar(255),
     model  varchar(255),
     primary key(manufacturer, serial_number));
```

date_study corresponde à data e hora do estudo.

```
-- study(*request_number*, *description*, date, doctor_id,
manufacturer, serial_number)
--    request_number: FK(request)
--    doctor_id: FK(doctor)
--    manufacturer, serial_number: FK(equipment)
create table study
    (request_number  varchar(255),
     description  varchar(255),
     date_study  datetime,
     doctor_id varchar(255),
     manufacturer  varchar(255),
     serial_number  varchar(255),
     primary key(request_number, description),
     foreign key(request_number) references request(request_number),
     foreign key(doctor_id) references doctor(doctor_id),
     foreign key(manufacturer, serial_number) references
 equipment(manufacturer, serial_number));
```

series_id é do tipo 'S-XX' e, por isso é uma *string*, a *name_series* é um nome, logo é uma *string* e a *base_url* é um *url*, logo é também uma *string*.

```
- series(*series_id*, name, base_url, request_number, description)
--    request_number, description: FK(study)
create table series
   (series_id  varchar(255),
    name_series  varchar(255),
    base_url  varchar(255),
    request_number varchar(255),
    description  varchar(255),
    primary key(series_id),
    foreign key(request_number, description) references
study(request_number, description));
```

elem_index é um *index*, ou seja, o número identificativo do elemento, por isso pode ser um inteiro.

```
-- element(*series_id*, *elem_index*)
--    series_id: FK(series)
create table element
 (series_id  varchar (255),
  elem_index  int,
  primary key(series_id, elem_index),
  foreign key(series_id) references series(series_id));
```

x1, *y1*, *x2* e *y2* são coordenadas e, por isso, são decimais.

```
-- region(*series_id*, *elem_index*, *x1*,*y1*,*x2*,*y2*)
--    series_id, index: FK(element)
create table region
 (series_id  varchar (255),
  elem_index  int,
  x1  numeric(20, 2), -- é uma coordenada, por isso é um decimal
  y1  numeric(20, 2), -- é uma coordenada, por isso é um decimal
  x2  numeric(20, 2), -- é uma coordenada, por isso é um decimal
  y2  numeric(20, 2), -- é uma coordenada, por isso é um decimal
  primary key(series_id, elem_index, x1, y1, x2, y2),
  foreign key(series_id, elem_index) references element(series_id,
elem_index));
```

Query

O objetivo da *query* é encontrar o médico que realizou o maior número de estudos com as seguintes características: o estudo realizado é um *X-Ray*, o estudo foi realizado com recurso a um equipamento da marca *Philips* e o estudo foi realizado há menos de 8 dias (há 7 ou menos dias). Desta forma, a *query* é dada por:

```
select doctor.name_doc
from doctor, study
where (doctor.doctor_id = study.doctor_id)
and (study.description like '%X-Ray%')
and (study.manufacturer like 'Philips')
and (datediff(current_date, study.date_study) <= 7)
group by doctor.doctor_id
having count(doctor.doctor_id) >= all (select count(doctor.doctor_id)
                    from doctor, study
                    where (doctor.doctor_id = study.doctor_id)
                    and (study.description like '%X-Ray%')
                    and (study.manufacturer like 'Philips')
                    and (datediff(current_date, study.date_study) <= 7)
                    group by doctor.doctor_id);
```

Dado que o objetivo é retornar o nome do médico que cumpra determinados requisitos, é colocado *doctor.name_doc* no SELECT.

De seguida, é necessário verificar quais os médicos que realizaram um *X-Ray* e que utilizaram equipamentos Philips nos últimos 7 dias. Esta abordagem é feita em 4 passos distintos nos quais começa por se verificar quais os médicos que realizaram estudos e, desses estudos, quais é que eram *X-Rays* e quais é que tinham sido realizados com recurso a um equipamento *Philips*. Por último, verifica-se quais os estudos que foram realizados nos últimos 7 dias.

Depois de toda a seleção procede-se à divisão do resultado em grupos cujo critério é o *doctor_id*. Optou-se por agrupar por *doctor_id* devido ao facto de poderem existir vários médicos com o mesmo nome (mas todos os médicos têm um *doctor_id* diferente).

Depois de se agrupar contabiliza-se o número de estudos associados a cada médico de modo a verificar quem realizou o maior número de estudos. De seguida é necessário fazer-se uma comparação entre todos os grupos criados. Desta forma, cria-se uma "nova tabela" com as mesmas limitações apresentadas anteriormente e agrupada por *doctor_id*. Assim, é possível verificar qual o médico que realizou mais estudos (com as limitações anteriormente descritas) através do seguinte processo:

- Contabilização do número de estudos do médico A da tabela 1;
- Comparação do número de estudos do médico A da tabela 1 com o número de estudos de todos os médicos da tabela 2;
- Caso o número de estudos do médico A da tabela 1 seja maior do que o número de estudos de todos os médicos da tabela 2, retorna o nome desse médico;
- Caso contrário, realiza o mesmo processo para todos os médicos da tabela 1.

A comparação entre tabelas é feita com recurso ao sinal '>=' dado que ao compararmos o médico com o maior número de estudos da tabela 1 com todos os médicos da tabela 2 vamos também comparar com o mesmo elemento (dado que as tabelas têm a mesma informação). Neste caso, o número de estudos é igual e, por isso, utilizou-se o sinal '>='.

TRIGGER

O objetivo do *trigger* é fazer com que o mesmo *doctor* não possa realizar o *study* do qual ele próprio fez um *request*, bem como que a *date_study* tenha de ser posterior à *date_ap*.

Isto pode ser implementado fazendo com que certas operações sejam proibidas de acontecer. Em ambos os casos, este impedimento será feito com recurso à chamada de uma função que não existe, de modo a ser lançada uma mensagem de erro, que, idealmente, será o mais clara possível no que toca a indicar ao utilizador qual foi o problema que ocorreu na operação. Ao surgir um erro, todo o processo de introdução de um valor na tabela é cancelado.

CHECK_STUDY é o nome do *trigger*, porque serve para testar especificações nos atributos do *Study*. Este *trigger* é chamado *BEFORE INSERT*, porque é antes de introduzir os dados que queremos testar o seu valor. Ou seja, antes de inserir um novo elemento nas tabelas, temos que garantir que ele cumpre certas especificações, e por isso tem que ser usado *FOR EACH ROW*. Por se tratar de um *trigger* no *INSERT*, todos os valores de interesse no *trigger* são *NEW*.

Antes de se inserir um novo *Study* na base de dados, vê-se quem é o *Doctor* que o vai realizar (através do seu id).

Caso este *Doctor* seja o mesmo que fez o requerimento desse mesmo estudo, então lança-se então uma mensagem de erro avisando que não pode ser o mesmo *Doctor* a fazer ambos.

Da mesma maneira, antes de se inserir um novo *Study* na base de dados, testa-se a data deste estudo. Como é natural, um estudo não pode ser realizado antes de ter sido feito o seu requerimento. Compara-se portanto a data do novo estudo com a data em que foi feito o requerimento deste estudo, e se a primeira for anterior à segunda lança-se do mesmo modo uma mensagem de erro.

```
delimiter $$
create trigger check_study before insert on study
for each row
begin
   if new.doctor_id = (select request.doctor_id
                from request
                where request.request_number = new.request_number) then
      call not_allowed_same_doctor_for_request_and_for_study();
   end if;
   if new.date_study < (select request.date_ap
                from request
                where request.request_number = new.request_number) then
      call date_study_must_be_posterior_to_date_app();
   end if;
end $$
delimiter ;
```

FUNCTION

Esta função deteta a sobreposição de regiões (elementos) de duas séries. Para isso recebe dois inputs: *serieA_id* e *serieB_id*, que correspondem aos identificativos das duas séries que queremos avaliar.

A função retorna *TRUE* ou *FALSE*, por isso, é do tipo *boolean*. Dadas duas series A e B retorna *TRUE*, se alguma região de A intersetar alguma região de B e retorna *FALSE*, caso contrário.

A tabela *regions* é duplicada em *regionA* e *regionB* e ambas são aglutinadas, isto é, cada entrada da primeira é associada a todas as entradas da segunda. Posteriormente, são retirados os casos que não interessam para a avaliação:

- entradas que correspondem a *series_id* diferentes das colocadas como inputs;
- entradas de comparação dos mesmos elementos: comparar o elemento X com o elemento X;
- entradas repetidas: comparar o elemento X com o elemento Y e o elemento Y com o elemento X (mantém apenas uma);

De seguida, é avaliada a sobreposição. Para chegar aos intervalos em que existe interseção avaliam-se todos os casos possíveis.

Cada elemento (região) é identificada por duas coordenadas (x1,y1) e (x2,y2). A região de interesse corresponde ao quadrado que tem (x1,y1) como vértice superior esquerdo e (x2,y2) como vértice inferior direito. Para a explicação que se segue, (x1A,y1A) e (x2A,y2A) correspondem a um elemento da região A; e (x1B,y1B) e (x2B,y2B) a um elemento da região B.

Hipóteses para coordenada x:

- 1º Caso: Se x1B < x1A, para haver interseção, x2B > x1A
- 2º Caso: Se x1B > x1A, para haver interseção, x1B < x2A

Hipóteses para coordenada y:

- 1º Caso: Se y1B < y1A, para haver interseção, y2B > y1A
- 2º Caso: Se y1B > y1A, para haver interseção, y1B < y2A

Assim sendo:

[(x1B < x1A AND x2B > x1A) OR (x1B > x1A AND x1B < x2A)] AND [(y1B < y1A AND y2B > y1A) OR (y1B > y1A AND y1B < y2A)]

Por fim, são contados os elementos existentes na tabela depois da aplicação destas restrições, e o valor da contagem é guardado na variável *answer*.

Caso *answer* seja diferente de zero, existe pelo menos uma sobreposição de regiões de séries diferentes.

```
delimiter $$
create function regions_overlapping(serieA_id varchar(255), serieB_id
varchar(255))
returns boolean
begin
  declare answer int default 0;
  if ((serieA_id in (select region.series_id from region)) and
  (serieB_id in (select region.series_id from region))) then
    select count(regionA.elem_index) into answer from region as
regionA, region as regionB
    where (regionA.series_id = serieA_id) and (regionB.series_id =
    serieB_id)
    and (((regionB.y1 <= regionA.y1) and (regionB.y2 >= regionA.y1)) or
    ((regionB.y1 >= regionA.y1) and (regionB.y1 <= regionA.y2)))
    and (((regionB.x1 <= regionA.x1) and (regionB.x2 >= regionA.x2)) or
    ((regionB.x1 >= regionA.x1) and (regionB.x1 <= regionA.x2)));
  end if;
  if (answer = 0) then
    return FALSE;
  end if;
  if (answer <> 0) then
    return TRUE;
  end if;
end $$
delimiter ;
```

De modo a testar a *query*, o *trigger* e a *function* foram feitos os seguintes *inserts*:

```
insert into patient values ('P-001', 'Cristina', '1990-01-01',
'Lisboa');
insert into patient values ('P-002', 'Joana', '1990-01-02', 'Porto');

insert into doctor values ('D-001', 'Orlanda', 'Cardiologia');
insert into doctor values ('D-002', 'Ricardo', 'Radiologia');
insert into doctor values ('D-003', 'Fernanda', 'Imagiologia');

insert into appointment values ('P-001', 'D-001', '2016-10-04 16:00:01',
'A1');

insert into request values ('R-001', 'P-001', 'D-001', '2016-10-04
16:00:01');
insert into request values ('R-002', 'P-001', 'D-001', '2016-10-04
16:00:01');

insert into equipment values ('Philips', 'S745019V', 'NX370');

insert into study values ('R-001', 'TAC', '2016-11-01 10:25:00', 'D-
002', 'Philips', 'S745019V');
insert into study values ('R-001', 'X-Ray ao braco direito', '2016-11-
15 18:30:00', 'D-002', 'Philips', 'S745019V');
insert into study values ('R-002', 'X-Ray a mao esquerda', '2016-11-15
18:30:00', 'D-002', 'Philips', 'S745019V');
insert into study values ('R-002', 'X-Ray ao pe direito', '2016-11-15
18:30:00', 'D-003', 'Philips', 'S745019V');

-- TESTAR OS TRIGGER
-- insert into study values ('R-001', 'outro X-Ray', '2016-09-10
18:30:00', 'D-003', 'Philips', 'S745019V');
-- insert into study values ('R-002', 'ainda outro X-Ray', '2016-11-17
18:30:00', 'D-001', 'Philips', 'S745019V');

insert into series values ('S-01', 'Radio', 'radio.com', 'R-001', 'X-
Ray ao braco direito');
insert into series values ('S-02', 'Falange', 'falange.com', 'R-002',
'X-Ray a mao esquerda');
insert into series values ('S-03', 'Falangeta', 'falangeta.com', 'R-
002', 'X-Ray a mao esquerda');
insert into series values ('S-04', 'Falanginha', 'falanginha.com', 'R-
002', 'X-Ray a mao esquerda');

insert into element values ('S-01', 1);
insert into element values ('S-01', 2);
insert into element values ('S-02', 1);
insert into element values ('S-03', 1);
insert into element values ('S-04', 1);
```

```
insert into region values ('S-04', 1, 3, 4, 5, 6);
insert into region values ('S-01', 2, 1, 1, 3, 3);
insert into region values ('S-02', 1, 2, 2, 3, 3);
insert into region values ('S-03', 1, 1, 1, 4, 4);
```