Sistemas de Informação e Bases de Dados

2ª Parte do Projeto

Grupo 13

Diogo Proença, 75313 Diogo Martins, 75462
Bernardo Gomes, 75573

25 de Novembro de 2015

Conteúdo

1 Criação das tabelas da base de dados

Para a criação das tabelas na base de dados, as intruções de SQL utilizadas foram as seguintes:

```
create table Patient(
        number varchar(40),
        name varchar(255),
        address varchar(255),
        primary key(number));

create table PAN(
        domain varchar(255),
        phone varchar(20),
        primary key(domain));

create table Device(
        serialnum integer,
        manufacturer varchar(255),
        description varchar(255),
        primary key(serialnum, manufacturer));

create table Sensor(
        snum integer,
        manuf varchar(255),
        units varchar(255),
        primary key(snum, manuf),
        foreign key(snum, manuf)
                        references Device(serialnum, manufacturer));

create table Actuator(
        snum integer,
        manuf varchar(255),
        units varchar(255),
        primary key(snum, manuf),
        foreign key(snum, manuf)
                        references Device(serialnum, manufacturer));

create table Municipality(
        nut4code integer,
        name varchar(255),
        primary key(nut4code));

create table Period(
        start date,
        end date,
        check(start<=end),
        primary key(start, end));
```

```
create table Reading(
        snum integer,
        manuf varchar(255),
        datetime timestamp,
        value numeric(20,2),
        primary key(snum, manuf, datetime),
        foreign key(snum, manuf) references Sensor(snum, manuf));

create table Setting(
        snum integer,
        manuf varchar(255),
        datetime timestamp,
        value numeric(20,2),
        primary key(snum, manuf, datetime),
        foreign key(snum, manuf) references Actuator(snum, manuf));

create table Wears(
        start date,
        end date,
        patient varchar(255),
        pan varchar(255),
        check(start<=end),
        primary key(start, end, patient),
        foreign key(start, end) references Period(start, end),
        foreign key(patient) references Patient(number),
        foreign key(pan) references PAN(domain));

create table Lives(
        start date,
        end date,
        patient varchar(255),
        muni integer,
        check(start<=end),
        primary key(start, end, patient),
        foreign key(start, end) references Period(start, end),
        foreign key(patient) references Patient(number),
        foreign key(muni) references Municipality(nut4code));

create table Connects(
        start date,
        end date,
        snum integer,
        manuf varchar(255),
        pan varchar(255),
        check(start<=end),
        primary key(start, end, snum, manuf),
        foreign key(start, end) references Period(start, end),
        foreign key(snum, manuf) references Device(serialnum, manufacturer),
        foreign key(pan) references PAN(domain));
```

Relativamente às escolhas das variáveis o seguinte conjunto merece especial destaque:

- **variável *number* da tabela *Patient*** → ao termos levado a cabo pesquisa relativa a *Social Security Numbers* (SSN's) válidos, verificámos que este conjunto de números pode conter caracteres não numéricos (-"). Além deste facto, pode também começar pelo número "0", o que invalida o uso de variáveis *integer*, *numeric* e *decimal* caso contrário o número armazenado iria perder dígitos. Utiliza-se assim, a variável *varchar*;

- **variável *phone* da tabela *PAN*** → tendo em conta a existência de números de telefone com prefixo diferente em cada país (*e.g.* Portugal - +351), e não sendo necessário que o paciente tenha um número do país onde o *Medical Center* está localizado, utiliza-se a variável *varchar*;

- **variável *serialnum* da tabela *Device*** → utiliza-se a variável *integer*. No entanto, poder-se-ia considerar também do tipo *varchar* pelo facto de em alguns casos os números de série conterem caracteres. No entanto, não tendo sendo especificado nada no enunciado, e não tendo obtido nenhuma informação sobre os números de série de aparelhos médicos, considera-se o caso em que estes podem ser representados por um número;

- Para as tabelas *Sensor* e *Actuator* é utilizado o mesmo critério que a tabela anterior;

- **variável *nut4code* da tabela *Municipality*** → para esta tabela, considera-se apenas os códigos postais semelhantes a Portugal, identificados por quatro números, tal como o nome da variável indica;

- **variáveis *start* e *end* da tabela *Period*** → escolhe-se o tipo *date* pelo facto de as relações com esta entidade não necessitar de precisões ao nível HH:mm:ss;

- para as tabelas *Period*, *Wears*, *Lives* e *Connects* foi colocada uma verificação das datas *check(start<=end)* de forma a garantir que o período está consistente;

- **variável *datetime* das tabelas *Reading* e *Setting*** → escolhe-se o tipo *timestamp* pelo facto de as medições médicas necessitarem de precisões ao nível HH:mm:ss;

- **variável *value* das tabelas *Reading* e *Setting*** → escolhe-se o tipo *numeric* pelo facto de permitir precisão ao nível de casas decimais.

Inicialmente, acrescentam-se as seguintes instruções de forma a apagar eventuais tabelas com o mesmo nome antes da criação das novas:

```
drop table if exists Reading;
drop table if exists Setting;
drop table if exists Sensor;
drop table if exists Actuator;
drop table if exists Connects;
drop table if exists Device;
drop table if exists Lives;
drop table if exists Wears;
drop table if exists Patient;
drop table if exists PAN;
drop table if exists Municipality;
drop table if exists Period;
```

2 *Triggers* para prevenção de *overlapping periods*

A função dos *triggers* é de prevenir a inserção/actualização de associações a PANs por parte de pacientes ou de aparelhos médicos em periodos de tempo sobrepostos.

A *error message* deve ser apresentada de forma a evitar que ocorram casos, para a tabela *Wears*, como os seguintes:

- Paciente 1 está ligado a PAN1 e PAN2 ao mesmo tempo;
- Paciente 1 e Paciente 2 estão ligados à PAN1 ao mesmo tempo.

A *error message* deve ser apresentada de forma a evitar que ocorram casos, para a tabela *Connects*, como o seguinte:

- Device 1 está ligado a PAN1 e PAN2 ao mesmo tempo.

A condição de sobreposição de períodos, independentemente da tabela em questão obtém-se através da análise do problema com a aplicação das leis de De Morgan. Ora, não existe sobreposição de períodos em dois casos:

1. o intervalo inserido/atualizado é completamente anterior aos intervalos já inseridos na tabela;
2. o intervalo inserido/atualizado é completamente posterior aos intervalos já inseridos na tabela.

Seja A um intervalo definido por startA e endA e o intervalo B definido por startB e endB. Os intervalos A e B não se encontram sobrepostos sempre que: (startA$\geq$endB) $\cup$ (endA$\leq$startB).

Negando a condição obtém-se todos os casos para os quais os intervalos A e B se encontram sobrepostos. Assim, através das leis de De Morgan obtem-se a seguinte condição para dois intervalos sobrepostos: (startA$\leq$endB)$\cap$(endA$\geq$startB). Esta condição é aplicada em todos os *triggers* que se apresentam de seguida.

2.1 *Triggers* de *insert*

Para a tabela *Connects*, não se pode inserir o mesmo *device* em duas PANs distintas no mesmo período de tempo. Desta forma, para cada linha dentro da tabela onde se pretende inserir o novo aparelho, será avaliado *a priori* se este já se encontra contido nos registos da tabela. Posteriormente, avalia-se se este *device* possui um PAN diferente, associado em períodos de tempo sobrepostos aos que se encontram nos registos da tabela.

O *trigger* para a tabela *Connects* tem o seguinte código:

```
drop trigger if exists check_overlap_time_period_Device_PAN;


delimiter $$

create trigger check_overlap_time_period_Device_PAN
                before insert on Connects

for each row
begin
if(
        exists(SELECT start, end FROM Connects
                WHERE(
                        (Connects.start <= new.end) and
                         (Connects.end >= new.start)
                        AND ((new.snum = Connects.snum) and
                        (new.manuf = Connects.manuf))
                        AND (new.pan != Connects.pan)
                )
        )
)
        then
        call overlaping_time_periods_for_Device_PAN();

end if;

end$$

delimiter ;
```

Relativamente à tabela *Wears*, o raciocínio aplicado é semelhante. Um paciente não pode ter duas PANs ao mesmo tempo e dois pacientes não podem partilhar a mesma PAN. Este caso, gera uma condição adicional, que não surgia no caso anterior. As condições são assim semelhantes às apresentadas anteriormente, sendo ainda verificado se o paciente se encontra na tabela e se a PAN que se pretende que o mesmo utilize está actualmente em utilização por um paciente distinto. No caso de o paciente se encontrar nos registos, verifica-se se a PAN que está a utilizar é a mesma e única no período de tempo.

O *trigger* para a tabela *Wears* tem o seguinte código:

```
drop trigger if exists check_overlap_time_period_Patient_PAN;

delimiter $$

create trigger check_overlap_time_period_Patient_PAN
                        before insert on Wears

for each row
begin
if(
        exists(SELECT start, end FROM Wears
                WHERE(
                        (Wears.start <= new.end) and
                                        (Wears.end >= new.start)
                        AND (((new.patient != Wears.patient)
                                AND (new.pan = Wears.pan))
                        OR ((new.patient = Wears.patient)
                                AND (new.pan != Wears.pan)))
                )
        )
)
        then
        call overlaping_time_periods_for_Patient_PAN();

end if;

end$$

delimiter ;
```

Por forma a testar o funcionamento dos *triggers* atrás apresentados, utilizámos a base de dados fornecida no ficheiro *database_triggers_insert.sql*. Após correr as instruções, obtêm-se as seguintes tabelas:

```
mysql> select * from Connects;
+------------+------------+-----------+---------+-------------+
| start      | end        | snum      | manuf   | pan         |
+------------+------------+-----------+---------+-------------+
| 2011-10-09 | 2012-12-01 | 123456789 | Philips | www.pan1.pt |
| 2014-12-25 | 2015-01-01 | 123456790 | Philips | www.pan1.pt |
| 2015-04-01 | 2015-10-25 | 123456789 | Philips | www.pan1.pt |
| 2015-10-26 | 2015-11-26 | 123456789 | Philips | www.pan1.pt |
+------------+------------+-----------+---------+-------------+
4 rows in set (0.00 sec)
```

Figura 1: Tabela *Connects*

```
mysql> select * from Wears;
+------------+------------+-------------------+-------------+
| start      | end        | patient           | pan         |
+------------+------------+-------------------+-------------+
| 2011-10-09 | 2012-12-01 | 001-54245-1555555 | www.pan1.pt |
| 2014-12-25 | 2015-01-01 | 001-54245-1555555 | www.pan1.pt |
| 2015-04-01 | 2015-10-25 | 001-54245-1555555 | www.pan1.pt |
| 2015-10-26 | 2015-11-26 | 001-54245-1555555 | www.pan1.pt |
+------------+------------+-------------------+-------------+
4 rows in set (0.00 sec)
```

Figura 2: Tabela *Wears*

Os testes realizados foram sucessivas inserções de um dispositivo em PANs diferentes sobreposto a pelo menos uma entrada da tabela.

Os testes realizados estão contidos nos ficheiros *teste_ insert_ connects.sql* e *teste_ insert_ connects.sql*, apresentados em anexo e que contem uma série de instruções e respectivos comentários.

Correndo os ficheiros de teste verifica-se um de quatro dos seguintes resultados, consoante as tabelas a alterar e cosoante seja uma um *update*/inserção válido ou inválido:

```
ERROR 1305 (42000): PROCEDURE ist175573.overlaping_time_periods_for_Device_PAN d
oes not exist
```

Figura 3: Mensagem de erro de inserção para a tabela *Connects*

```
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint f
ails (`ist175573`.`Connects`, CONSTRAINT `Connects_ibfk_1` FOREIGN KEY (`start`,
 `end`) REFERENCES `Period` (`start`, `end`))
```

Figura 4: Mensagem sucesso de inserção para a tabela *Connects*

```
ERROR 1305 (42000): PROCEDURE ist175462.overlaping_time_periods_for_Patient_PAN
does not exist
```

Figura 5: Resultados dos testes do *trigger* de inserção para a tabela *Connects*

```
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint f
ails (`ist175462`.`Wears`, CONSTRAINT `Wears_ibfk_1` FOREIGN KEY (`start`, `end`
) REFERENCES `Period` (`start`, `end`))
```

Figura 6: Resultados dos testes do *trigger* de inserção para a tabela *Connects*

Em caso de sucesso, ou seja, sem sobreposição, surge uma mensagem de erro devido ao facto de o período inserido não existir na tabela *Period*. Caso estivesse, não ocorreria o erro e a tabela seria atualizada.

2.2 *Triggers* de *update*

À semelhança dos testes que realizámos para os *triggers* de *insert* para as duas tabelas, os ficheiros de teste para os *triggers* de *update* encontram-se no ficheiro .zip, com os nomes *teste_update_connects.sql* e *teste_update_wears.sql*.

Os *triggers* relativos a esta secção só diferem dos anteriores na medida em que são efectuados antes de um *update* em vez de antes de um *insert*. Assim, nos troços de código anteriores, onde se lê

create trigger check_overlap_time_period_Patient_PAN before insert on Wears deve-se ler

create trigger check_overlap_time_period_Patient_PAN before update on Wears e onde se lê

create trigger check_overlap_time_period_Device_PAN before insert on Connects deve-se ler

create trigger check_overlap_time_period_Device_PAN before insert on Connects.

No entanto, para esta secção foi utilizado um maior número de testes pois existe um maior número de casos de atualizações possíveis do que inserções.

Assim, a base de dados utilizada para os testes foi a presente no ficheiro *database_triggers_update.sql*, sendo obtidas as seguintes tabelas:

```
mysql> select * from Connects;
+------------+------------+-----------+---------+-------------+
| start      | end        | snum      | manuf   | pan         |
+------------+------------+-----------+---------+-------------+
| 2011-10-09 | 2012-12-01 | 123456789 | Philips | www.pan1.pt |
| 2014-12-25 | 2015-01-01 | 123456790 | Philips | www.pan1.pt |
| 2015-04-01 | 2015-10-25 | 123456789 | Philips | www.pan1.pt |
| 2015-10-26 | 2015-11-26 | 123456789 | Philips | www.pan1.pt |
| 2013-11-25 | 2013-12-01 | 123456789 | Philips | www.pan3.pt |
+------------+------------+-----------+---------+-------------+
5 rows in set (0.00 sec)
```

Figura 7: Tabela *Connects*

```
mysql> select * from Wears;
+------------+------------+-------------------+-------------+
| start      | end        | patient           | pan         |
+------------+------------+-------------------+-------------+
| 2011-10-09 | 2012-12-01 | 001-54245-1555555 | www.pan1.pt |
| 2013-11-25 | 2013-12-01 | 001-54245-1555575 | www.pan1.pt |
| 2015-04-01 | 2015-10-25 | 001-54245-1555555 | www.pan1.pt |
| 2015-10-26 | 2015-11-26 | 001-54245-1555555 | www.pan1.pt |
| 2014-12-25 | 2015-01-01 | 001-54245-1555555 | www.pan3.pt |
+------------+------------+-------------------+-------------+
5 rows in set (0.00 sec)
```

Figura 8: Tabela *Wears*

Na tabela da figura 7, a linha que é atualizada é sempre a última, correspondente a um *device* igual a pelo menos um que se encontra na tabela. Porém, possui um PAN diferente destes.

Na tabela da figura 8, as linhas que são atualizadas são a segunda e última linhas que testam, respectivamente, dois pacientes possuirem a mesma PAN simultaneamente e o mesmo paciente ter duas PANs simultaneamente.

3 *SQL queries*