Projeto de Bases de Dados

Parte 3

Instituto Superior Técnico

Bases de Dados
Turno L07 — 1º semestre 2020/2021
Grupo 55
Docente: Daniel Faria

	Alunos	**Esforço**
92626	Filipa Costa	6h - 20 %
92648	Simão Leal	13h - 42 %
92649	Sofia Pereira	11h30 - 38 %

20 de novembro de 2020

Criação da Base de Dados

```sql
DROP TABLE IF EXISTS prescricao_venda CASCADE;
DROP TABLE IF EXISTS venda_farmacia CASCADE;
DROP TABLE IF EXISTS analise CASCADE;
DROP TABLE IF EXISTS prescricao CASCADE;
DROP TABLE IF EXISTS consulta CASCADE;
DROP TABLE IF EXISTS medico CASCADE;
DROP TABLE IF EXISTS instituicao CASCADE;
DROP TABLE IF EXISTS concelho CASCADE;
DROP TABLE IF EXISTS regiao CASCADE;

CREATE TABLE regiao(
num_regiao INTEGER,
nome VARCHAR(8),
num_habitantes INTEGER NOT NULL,
primary key(num_regiao),
unique(nome),
CHECK(num_regiao>0),
CHECK(nome in ('Norte', 'Centro', 'Lisboa', 'Alentejo', 'Algarve')),
CHECK(num_habitantes>=0)
);

CREATE TABLE concelho(
num_concelho INTEGER,
num_regiao INTEGER,
nome VARCHAR(50),
num_habitantes INTEGER NOT NULL,
primary key(num_concelho,num_regiao),
foreign key(num_regiao) references regiao(num_regiao) on delete cascade on update cascade,
unique(nome),
CHECK(num_concelho>0),
CHECK(num_habitantes>=0)
);

CREATE TABLE instituicao(
nome VARCHAR(50),
tipo VARCHAR(11),
num_regiao INTEGER,
num_concelho INTEGER,
primary key(nome),
foreign key(num_regiao,num_concelho) references concelho(num_regiao, num_concelho) on delete cascade on update cascade,
CHECK(tipo in ('farmacia', 'laboratorio', 'clinica', 'hospital'))
);

CREATE TABLE medico(
num_cedula INTEGER,
nome VARCHAR(50),
especialidade VARCHAR(50),
primary key(num_cedula),
CHECK(num_cedula>0)
);
```

```
51
52 CREATE TABLE consulta(
53 num_cedula INTEGER,
54 num_doente INTEGER,
55 data DATE,
56 nome_instituicao VARCHAR(50),
57 primary key(num_cedula,num_doente,data),
58 foreign key(num_cedula) references medico(num_cedula) on delete cascade on
       update cascade,
59 foreign key(nome_instituicao) references instituicao(nome) on delete
       cascade on update cascade,
60 unique(num_doente,data,nome_instituicao),
61 CHECK(num_doente>0),
62 CHECK(EXTRACT(DOW FROM data) in ('1', '2','3', '4', '5'))
63 );
64
65 CREATE TABLE prescricao(
66 num_cedula INTEGER,
67 num_doente INTEGER,
68 data DATE,
69 substancia VARCHAR(50),
70 quant NUMERIC (10,6),
71 primary key(num_cedula,num_doente,data,substancia),
72 foreign key(num_cedula,num_doente,data) references consulta(num_cedula,
       num_doente,data) on delete cascade on update cascade,
73 CHECK(quant>=0)
74 );
75
76 CREATE TABLE analise(
77 num_analise INTEGER,
78 especialidade VARCHAR(50),
79 num_cedula INTEGER,
80 num_doente INTEGER,
81 data DATE,
82 data_registo DATE,
83 nome VARCHAR(50),
84 quant NUMERIC(10,6),
85 inst VARCHAR(50),
86 primary key(num_analise),
87 foreign key(num_cedula,num_doente,data) references consulta(num_cedula,
       num_doente,data) on delete cascade on update cascade,
88 foreign key(inst) references instituicao(nome),
89 CHECK(quant>=0),
90 CHECK(data <= data_registo)
91 );
92
93 CREATE TABLE venda_farmacia(
94 num_venda INTEGER,
95 data_registo DATE,
96 substancia VARCHAR(50),
97 quant NUMERIC(10,6),
98 preco NUMERIC(4,2),
99 inst VARCHAR(50),
100 primary key(num_venda),
```

```
foreign key(inst) references instituicao(nome) on delete cascade on update
    cascade,
CHECK(num_venda>0),
CHECK(quant>=0),
CHECK(preco>=0)
);

CREATE TABLE prescricao_venda(
num_cedula INTEGER,
num_doente INTEGER,
data DATE,
substancia VARCHAR(50),
num_venda INTEGER,
primary key(num_cedula, num_doente, data, substancia, num_venda),
foreign key(num_venda) references venda_farmacia(num_venda) on delete
    cascade on update cascade,
foreign key(num_cedula, num_doente, data, substancia) references prescricao
    (num_cedula, num_doente, data, substancia) on delete cascade on update
    cascade
);
```

Para garantir a criação correta das tabelas, acrescentámos o comando `DROP TABLE IF EXISTS <nome tabela> CASCADE`, de forma a apagar possíveis tabelas com o mesmo nome anteriormente criadas e introduzir as pretendidas. O comando `CHECK` foi utilizado para garantir que instâncias de atributos estão de acordo com o respetivo domínio. O comando `NOT NULL` foi utilizado para garantir que nenhum dos atributos num_habitantes possa ser nulo.

De modo a preservar as relações de foreign key nas atualizações da base de dados, recorremos ao comando `REFERENCES <nome tabela_origem> ON DELETE CASCADE ON UPDATE CASCADE`. Ou seja, quando uma entrada é apagada ou atualizada da tabela origem, todas as entradas da tabela que a referencia também são.

Para além disso, `CHECK` foi usado para garantir a satisfação das restrições de integridade. Note-se que `unique(num_doente,data,nome_instituicao)` certifica a RI-consulta-2. Já `CHECK(EXTRACT(DOW FROM DATE data) in ('1','2','3','4','5'))` verifica a RI-consulta-1.

Note-se que usando apenas `CHECK` não é possível concretizar a RI-analise, já que isso requeria referenciar a tabela médico para poder verificar a respetiva especialidade. No entanto, tivemos isso em consideração quando fizemos a inserção de dados na base de dados.

Não havia nenhuma restrição que fizesse com que o atributo substância em venda_farmacia fosse igual ao de prescricao_venda, mas assumimos isso na inserção de dados na base de dados. Da mesma maneira, como não havia nenhuma restrição que impusesse que as instituições em venda_farmacia fossem do tipo "farmacia" mas tivemos isso em consideração no `populate.sql`.

O ficheiro `populate.sql` foi gerado aleatoriamente, usando um script Python.

SQL

```
-- 1:
SELECT c.nome AS concelho
FROM venda_farmacia v, instituicao i, concelho c
WHERE v.inst=i.nome AND i.num_regiao = c.num_regiao AND i.num_concelho = c.
    num_concelho
AND data_registo = CURRENT_DATE
GROUP BY c.nome
HAVING sum(preco) >= ALL(   SELECT sum(preco)
```

```
                FROM venda_farmacia v, instituicao i, concelho c
               WHERE v.inst=i.nome AND (i.num_regiao = c.num_regiao AND i.
    num_concelho = c.num_concelho)
               AND data_registo = CURRENT_DATE
               GROUP BY c.nome);
-- 2:
SELECT m.nome AS medico, r.nome AS regiao
FROM (prescricao p NATURAL JOIN consulta c NATURAL JOIN medico m) INNER
    JOIN (instituicao i INNER JOIN regiao r ON i.num_regiao = r.num_regiao)
     ON c.nome_instituicao = i.nome
WHERE data BETWEEN '2019-01-01' AND '2019-06-30'
GROUP BY m.nome, r.nome
HAVING COUNT(*) >= ALL(SELECT COUNT(*)
                 FROM (prescricao p NATURAL JOIN consulta c NATURAL JOIN
    medico m) INNER JOIN (instituicao i INNER JOIN regiao r2 ON i.
    num_regiao = r2.num_regiao) ON c.nome_instituicao = i.nome
                 WHERE r2.nome = r.nome AND data BETWEEN '2019-01-01' AND '
    2019-06-30'
                 GROUP BY m.nome);
-- 3:
SELECT m.nome AS medico
FROM ((prescricao_venda p INNER JOIN venda_farmacia v on p.num_venda = v.
    num_venda) NATURAL JOIN medico m) INNER JOIN instituicao i  ON v.inst =
     i.nome
WHERE EXTRACT(YEAR FROM data_registo)= EXTRACT(YEAR FROM CURRENT_DATE) AND
    tipo='farmacia' AND num_concelho='4' AND num_regiao='2' AND p.
    substancia='Aspirina'
GROUP BY m.nome
HAVING COUNT(DISTINCT i.nome) = (SELECT COUNT(DISTINCT i.nome)
                        FROM instituicao i
                        WHERE tipo='farmacia' AND num_concelho='4' AND
    num_regiao='2');
-- 4:
(SELECT num_doente
FROM analise
WHERE EXTRACT(MONTH FROM data) = EXTRACT(MONTH FROM CURRENT_DATE) AND
    EXTRACT(YEAR FROM data) = EXTRACT(YEAR FROM CURRENT_DATE))
EXCEPT
(SELECT num_doente
FROM prescricao_venda
WHERE EXTRACT(MONTH FROM data) = EXTRACT(MONTH FROM CURRENT_DATE) AND
    EXTRACT(YEAR FROM data)=EXTRACT(YEAR FROM CURRENT_DATE));
```

É de notar que na query 2 podem ser retornados vários nomes de médicos para cada região. Tal acontece se houver empates no número de prescrições (é o caso para a nossa base de dados).

Foi usado o comando `NATURAL JOIN` quando os nomes dos atributos eram iguais e `INNER JOIN` quando atributos idênticos tinham nomes diferentes em tabelas diferentes.

Para as queries 1, 2 e 3, considerámos devolver mesmo o nome dos concelhos, regiões e médicos, uma vez que da perspetiva dum sistema de informação, faz mais sentido para um utilizador receber essa informação do que um num_cédula, por exemplo.

NOTA: Escolhemos submeter a parte da aplicação juntamente com a próxima entrega.