Projeto de Bases de Dados

Parte 4

Instituto Superior Técnico

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

	Alunos	**Esforço**
92626	Filipa Costa	14h - 33.(1) %
92648	Simão Leal	15h - 33.(3) %
92649	Sofia Pereira	16h - 33.(5) %

4 de dezembro de 2020

Para além dos ficheiros pedidos, junto com esta entrega submetemos um ficheiro `schema.sql` com pequenas correções e ajustes, um ficheiro `populate.sql` para que as queries desta entrega tivessem resultado não vazio. Em seguida listamos as alterações ao ficheiro `schema.sql`:

- Adição da restrição `UNIQUE(num_venda)` na tabela prescricao_venda necessária à construção do ficheiro `star_schema.sql`.
- Adição de condições `NOT NULL` em quase todos os atributos (exceto nos atributos da consulta na tabela `analise`), uma vez que na entrega 3 não interpretámos como sendo necessário.
- Retificação de algumas imprecisões, nomeadamente foi estabelecido que `quant` (em `analise`, `prescricao` e `venda_farmacia`) tem de ser `INTEGER` e que `preco` em `venda_farmacia` é do tipo `NUMERIC(6,2)`, para dar uma maior amplitude de valores.

Restrições de Integridade

```sql
--RI-100
CREATE OR REPLACE FUNCTION ri_100() RETURNS TRIGGER AS
$$
DECLARE
  total INTEGER;
BEGIN
  SELECT COUNT(*) INTO total
  FROM consulta c
  WHERE c.num_cedula = new.num_cedula AND EXTRACT(WEEK FROM new.data) = EXTRACT(
    WEEK FROM c.data)
  AND EXTRACT(YEAR FROM new.data) = EXTRACT(YEAR FROM c.data) AND new.
    nome_instituicao = c.nome_instituicao;
  IF total > 100 THEN
    RAISE EXCEPTION 'O médico com nº de cédula % já realizou 100 consultas na
    semana % de % na instituição %.',
    new.num_cedula, EXTRACT(WEEK FROM new.data), EXTRACT(YEAR FROM new.data), new.
    nome_instituicao;
  END IF;
  RETURN new;
END;
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS insert_consulta_trigger ON consulta;
CREATE TRIGGER insert_consulta_trigger AFTER INSERT ON consulta
  FOR EACH ROW EXECUTE PROCEDURE ri_100();

DROP TRIGGER IF EXISTS update_consulta_trigger ON consulta;
CREATE TRIGGER update_consulta_trigger AFTER UPDATE ON consulta
  FOR EACH ROW EXECUTE PROCEDURE ri_100();


--RI-analise
CREATE OR REPLACE FUNCTION ri_analise() RETURNS TRIGGER AS
$$
DECLARE
  esp VARCHAR(50);
BEGIN
  IF new.num_cedula IS NOT NULL THEN
    SELECT especialidade INTO esp
    FROM medico m WHERE m.num_cedula = new.num_cedula;
    IF new.especialidade != esp THEN
```

```
39        RAISE EXCEPTION 'A especialidade da análise deve ser igual à do médico da
      consulta.';
40       END IF;
41    END IF;
42    RETURN new;
43  END;
44  $$
45  LANGUAGE plpgsql;
46
47  DROP TRIGGER IF EXISTS insert_analise_trigger ON analise;
48  CREATE TRIGGER insert_analise_trigger AFTER INSERT ON analise
49    FOR EACH ROW EXECUTE PROCEDURE ri_analise();
50
51  DROP TRIGGER IF EXISTS update_analise_trigger ON analise;
52  CREATE TRIGGER update_analise_trigger AFTER UPDATE ON analise
53    FOR EACH ROW EXECUTE PROCEDURE ri_analise();
```

RI.sql

Índices

Exercício 1
Teoricamente o melhor índice a ser implementado perante um caso de seleção por igualdade, seria um índice do tipo HASH sobre o atributo num_doente da tabela `consulta`, cujo código para implementação se encontra abaixo.
No entanto, sabemos que quando a tabela `consulta` é criada, é também criado um índice do tipo BTree da sua chave primária composta `(num_cedula, num_doente, data)`, que potencializa a pesquisa sobre estes três atributos em simultâneo. Mais, como `num_doente` não se encontra no primeiro argumento `(num_cedula, num_doente, data)` não é boa prática usar o índice associado a esta primary key. Ora, neste contexto, o ideal seria alterar a ordem dos atributos da tabela `consulta`, colocando `num_doente` em primeiro lugar, e assim já não havia a necessidade de criar um novo índice. Tendo em conta este facto, os custos associados à criação de um novo índice, poderão não se justificar.

```
DROP INDEX IF EXISTS consulta_index;
CREATE INDEX consulta_index ON consulta USING HASH(num_doente);
```

Exercício 2
Sabemos que aquando da criação da tabela `medico` é criado um índice do tipo BTree sobre a primary key `num_cedula`. No entanto, pretende-se procurar pela `especialidade` pelo que faz sentido criar um novo índice. Como a coluna `especialidade` tem baixa cardinalidade (6), ou seja, não existem muitas instâncias de especialidade possíveis, para otimizar esta query pensámos usar um índice do tipo BITMAP sobre o atributo `especialidade` da tabela `medico`. Neste caso, o índice não é suportado pelo SGBD, pelo que sugerimos a criação dum índice do tipo HASH, visto ser um caso de igualdade de valores.

```
DROP INDEX IF EXISTS esp_index;
CREATE INDEX esp_index ON medico USING HASH(especialidade);
```

Exercício 3
Os blocos do disco são de 2K bytes e cada registo na tabela ocupa 1K bytes, pelo que cada bloco tem 2 registos. Uma vez que os médicos estão uniformemente distribuídos pelas 6 especialidades ($p = 1/6$), a probabilidade de um bloco não ter respostas é de $(1-p)^2 \approx 69.(4)\%$. Assim teremos de ler 30,5% dos blocos, o que compensa a utilização de índices.
Estamos perante um caso de seleção por igualdade, pelo que o melhor índice a ser implementado é um índice do tipo HASH sobre o atributo `especialidade` da tabela `medico`, mais eficiente para este tipo de pesquisa.

```
DROP INDEX IF EXISTS especialidade_index;
CREATE INDEX especialidade_index ON medico USING HASH(especialidade);
```

Exercício 4
Quando a tabela `consulta` é criada, é também criado um índice do tipo BTree da sua chave primária composta (`num_cedula, num_doente, data`), que potencializa a pesquisa sobre estes três atributos em simultâneo. Como `num_cedula` encontra-se no primeiro argumento usamos o índice associado a esta primary key. O cenário ideal para um `JOIN` é ter um índice do tipo HASH em ambos os tuplos, no entanto como já existe índice BTree, não compensa criar outro.
De forma idêntica ao exercício 1, o ideal seria alterar a ordem dos atributos da tabela `consulta`, colocando `data` em primeiro lugar, e assim já não havia a necessidade de criar um novo índice. No entanto, caso não seja possível alterar esta ordem, optaríamos por criar um novo índice do tipo BTree, (visto ser um caso de pesquisa de um intervalo de valores), apenas sobre o atributo `data` da tabela `consulta`, mais eficiente para este tipo de pesquisa.

```
DROP INDEX IF EXISTS data_index;
CREATE INDEX data_index ON consulta USING BTREE(data);
```

Modelo Multidimensional

No modelo multidimensional, para facilitar o cálculo de médias diárias, escolhemos preencher `d_tempo` com todas as datas entre 2019-01-01 e 2021-12-31, tomando partido de sabermos que todas as entradas na nossa base de dados se encontram entre estas datas. Em `f_presc_venda` tomámos como índice o `num_venda` da venda associada. Para isso tivémos de juntar a restrição `UNIQUE` no esquema antigo, como já foi mencionado.

```
DROP TABLE IF EXISTS d_tempo CASCADE;
DROP TABLE IF EXISTS d_instituicao CASCADE;
DROP TABLE IF EXISTS f_analise CASCADE;
DROP TABLE IF EXISTS f_presc_venda CASCADE;

CREATE TABLE d_tempo(
id_tempo SERIAL,
dia INTEGER NOT NULL,
dia_da_semana INTEGER NOT NULL,
semana INTEGER NOT NULL,
mes INTEGER NOT NULL,
trimestre INTEGER NOT NULL,
ano INTEGER NOT NULL,
primary key(id_tempo)
);

CREATE TABLE d_instituicao(
id_inst SERIAL,
nome VARCHAR(50) NOT NULL,
tipo VARCHAR(50) NOT NULL,
num_regiao INTEGER NOT NULL,
num_concelho INTEGER NOT NULL,
primary key(id_inst),
foreign key(nome) references instituicao(nome) on delete cascade on update cascade
    ,
foreign key(num_regiao,num_concelho) references concelho(num_regiao,num_concelho)
    on delete cascade on update cascade
);

CREATE TABLE f_presc_venda(
id_presc_venda INTEGER,
id_medico INTEGER NOT NULL,
num_doente INTEGER NOT NULL,
id_data_registo INTEGER NOT NULL,
id_inst INTEGER NOT NULL,
substancia VARCHAR(50) NOT NULL,
```

```
quant INTEGER NOT NULL,
primary key(id_presc_venda),
foreign key(id_presc_venda) references prescricao_venda(num_venda) on delete cascade on update cascade, --assegurando que num_venda é unique em presc_venda
foreign key(id_medico) references medico(num_cedula) on delete cascade on update cascade,
foreign key(id_data_registo) references d_tempo(id_tempo) on delete cascade on update cascade,
foreign key(id_inst) references d_instituicao(id_inst) on delete cascade on update cascade
);

CREATE TABLE f_analise(
id_analise INTEGER,
id_medico INTEGER,
num_doente INTEGER,
id_data_registo INTEGER NOT NULL,
id_inst INTEGER NOT NULL,
nome VARCHAR(50) NOT NULL,
quant INTEGER NOT NULL,
primary key(id_analise),
foreign key(id_analise) references analise(num_analise) on delete cascade on update cascade,
foreign key(id_medico) references medico(num_cedula) on delete cascade on update cascade,
foreign key(id_data_registo) references d_tempo(id_tempo) on delete cascade on update cascade,
foreign key(id_inst) references d_instituicao(id_inst) on delete cascade on update cascade
);
```

star_schema.sql

```
-- carregar d_tempo
INSERT INTO d_tempo(dia, dia_da_semana, semana, mes, trimestre, ano)
SELECT EXTRACT(DAY FROM s), EXTRACT(DOW FROM s), EXTRACT(WEEK FROM s), EXTRACT(MONTH FROM s), EXTRACT(QUARTER FROM s), EXTRACT(YEAR FROM s) FROM generate_series('2019-01-01', '2021-12-31','1 day'::interval) s;

-- carregar d_instituicao
INSERT INTO d_instituicao(nome, tipo, num_regiao, num_concelho)
SELECT nome, tipo, num_regiao, num_concelho FROM instituicao;

-- carregar f_analise
INSERT INTO f_analise
SELECT a.num_analise, a.num_cedula, a.num_doente, t.id_tempo, i.id_inst, a.nome, a.quant
FROM analise a JOIN d_instituicao i ON a.inst = i.nome
  JOIN d_tempo t ON (EXTRACT(YEAR FROM a.data_registo) = t.ano AND EXTRACT(MONTH FROM a.data_registo) = t.mes AND EXTRACT(DAY FROM a.data_registo) = t.dia);

-- carregar f_presc_venda
INSERT INTO f_presc_venda
SELECT pv.num_venda, pv.num_cedula, pv.num_doente, t.id_tempo, i.id_inst, pv.substancia, vf.quant
FROM prescricao_venda pv JOIN venda_farmacia vf ON pv.num_venda = vf.num_venda
  JOIN d_instituicao i ON vf.inst = i.nome
  JOIN d_tempo t ON (EXTRACT(YEAR FROM vf.data_registo) = t.ano AND EXTRACT(MONTH FROM vf.data_registo) = t.mes AND EXTRACT(DAY FROM vf.data_registo) = t.dia);
```

etl.sql

Data Analytics

Em relação à query 2, havia alguma ambiguidade sobre como calcular a média diária, na medida em que não era claro se a média devia ser feita sobre os dias todos do período compreendido, se apenas nos dias em que registaram entradas. A nossa interpretação foi a primeira, mas também incluímos a query para a interpretação alternativa.

```
--1
SELECT m.especialidade, t.mes, t.ano, COUNT(*) as numero_de_analises
FROM (f_analise f INNER JOIN medico m ON f.id_medico=m.num_cedula) INNER JOIN
    d_tempo t ON t.id_tempo=f.id_data_registo
WHERE f.nome='glicémia' AND t.ano BETWEEN '2017' AND '2020'
GROUP BY
   CUBE(m.especialidade, t.mes, t.ano)
ORDER BY m.especialidade, t.mes, t.ano;

--2
(SELECT p.substancia, c.nome, t.dia_da_semana, t.mes, SUM(p.quant) AS
    quantidade_total, COUNT(*)::DECIMAL(5,1)/(SELECT COUNT(*) FROM d_tempo WHERE
  trimestre='1' AND ano='2020' AND dia_da_semana=t.dia_da_semana AND mes=t.mes) AS
     nr_medio_presc_diario
FROM d_tempo t JOIN f_presc_venda p ON t.id_tempo=p.id_data_registo JOIN
    d_instituicao i ON p.id_inst = i.id_inst
  JOIN concelho c ON (c.num_regiao=i.num_regiao AND c.num_concelho = i.
    num_concelho) JOIN regiao r ON r.num_regiao = i.num_regiao
WHERE t.trimestre = 1 AND t.ano= 2020 AND r.nome = 'Lisboa'
GROUP BY p.substancia, c.nome, t.dia_da_semana, t.mes)
UNION
(SELECT p.substancia, c.nome, t.dia_da_semana, null, SUM(p.quant) AS
    quantidade_total, COUNT(*)::DECIMAL(5,1)/(SELECT COUNT(*) FROM d_tempo WHERE
     trimestre='1' AND ano='2020' AND dia_da_semana=t.dia_da_semana)  AS
    nr_medio_presc_diario
FROM d_tempo t JOIN f_presc_venda p ON t.id_tempo=p.id_data_registo JOIN
    d_instituicao i ON p.id_inst = i.id_inst
  JOIN concelho c ON (c.num_regiao=i.num_regiao AND c.num_concelho = i.
    num_concelho) JOIN regiao r ON r.num_regiao = i.num_regiao
WHERE t.trimestre = 1 AND t.ano= 2020 AND r.nome = 'Lisboa'
GROUP BY p.substancia, c.nome, t.dia_da_semana)
UNION
(SELECT p.substancia, c.nome,null, null, SUM(p.quant) AS quantidade_total, COUNT
    (*)::DECIMAL(5,1)/(SELECT COUNT(*) FROM d_tempo WHERE
  trimestre='1' AND ano='2020') AS nr_medio_presc_diario
FROM d_tempo t JOIN f_presc_venda p ON t.id_tempo=p.id_data_registo JOIN
    d_instituicao i ON p.id_inst = i.id_inst
  JOIN concelho c ON (c.num_regiao=i.num_regiao AND c.num_concelho = i.
    num_concelho) JOIN regiao r ON r.num_regiao = i.num_regiao
WHERE t.trimestre = 1 AND t.ano= 2020 AND r.nome = 'Lisboa'
GROUP BY p.substancia, c.nome)
UNION
(SELECT p.substancia, null,null, null, SUM(p.quant) AS quantidade_total, COUNT(*)
    ::DECIMAL(5,1)/(SELECT COUNT(*) FROM d_tempo WHERE
  trimestre='1' AND ano='2020') AS nr_medio_presc_diario
FROM d_tempo t JOIN f_presc_venda p ON t.id_tempo=p.id_data_registo JOIN
    d_instituicao i ON p.id_inst = i.id_inst
  JOIN concelho c ON (c.num_regiao=i.num_regiao AND c.num_concelho = i.
    num_concelho) JOIN regiao r ON r.num_regiao = i.num_regiao
WHERE t.trimestre = 1 AND t.ano= 2020 AND r.nome = 'Lisboa'
GROUP BY p.substancia)
ORDER BY substancia, nome, dia_da_semana, mes;

```

```
39
40 --2  alternativa com médias apenas nos dias em que há prescrições
41 SELECT p.substancia, c.nome, t.dia_da_semana, t.mes, SUM(p.quant) AS
       quantidade_total, COUNT(p.id_presc_venda)::DECIMAL(5,1)/COUNT(DISTINCT t.
       id_tempo) AS nr_medio_presc_diario
42 FROM d_tempo t JOIN f_presc_venda p ON t.id_tempo=p.id_data_registo JOIN
       d_instituicao i ON p.id_inst = i.id_inst
43   JOIN concelho c ON (c.num_regiao=i.num_regiao AND c.num_concelho = i.
       num_concelho) JOIN regiao r ON r.num_regiao = i.num_regiao
44 WHERE t.trimestre = 1 AND t.ano= 2020 AND r.nome = 'Lisboa'
45 GROUP BY p.substancia, ROLLUP(c.nome, t.dia_da_semana, t.mes)
46 ORDER BY substancia, nome, dia_da_semana, mes;
```

olap.sql