Projeto de Bases de Dados - Parte 4

Grupo: 90, Turno: L04, Docente: Tiago Oliveira

Nome	Nº Aluno	Esforço
Mafalda Serafim	92512	10h00 (50%)
Rafael Gonçalves	92544	10h00 (50%)

Restrições de Integridade:

1. RI-100:

```
drop trigger if exists ri_100_trigger on consulta;

create or replace function  ri_100_proc() returns trigger as $$
begin
if (select count(*) from consulta where new.num_cedula = num_cedula and new.nome_instituicao = nome_instituicao and extract(isoyear from new.data) = extract(isoyear from data) and extract(week from new.data) = extract(week from data)) >= 100 then
        raise exception 'r-100: % already has 100 consultas on week %, year % in %', new.num_cedula, extract(week from new.data), extract(isoyear from new.data), new.nome_instituicao;
end if;
return new;
end;
$$ language plpgsql;

create trigger ri_100_trigger before insert or update on consulta for each row execute procedure ri_100_proc();
```

2. RI-analise:

```
drop trigger if exists ri_analise_trigger on analise;

create or replace function  ri_analise_proc() returns trigger as $$
begin
if new.num_cedula is not null and new.num_doente is not null and new.data is not null and (select especialidade from medico where new.num_cedula = num_cedula) != new.especialidade then
        raise exception 'r-analise: especialidade % does not match especialidade on medico', new.especialidade;
end if;
return new;
end;
$$ language plpgsql;

create trigger ri_analise_trigger before insert or update on analise
for each row execute procedure ri_analise_proc();
```

Índices:

1. create index doente_index on consulta (num_doente)
   Nesta query seria utilizado o índice implícito “num_doente” (ordenado e esparso), uma vez que este parâmetro pertence à tabela “consulta” e é parte da sua chave primária (pelo que para ordenar por num_doente bastava alterar a lógica da tabela para esta ser a primeira coluna). A utilização deste índice em detrimento da totalidade da chave primária é benéfica porque permite filtrar informação desnecessária para a query, nomeadamente num_cedula e data. Uma vez que estamos perante uma procura por igualdade, seria benéfico utilizar um índice de hash.

2. create index especialidade_index on medico (especialidade)
   Nesta query seria utilizado o índice “especialidade” (não ordenado e denso). A utilização deste índice em detrimento de um índice implícito é benéfica porque seis especialidades ocupam muito pouco espaço em cache e porque nos permite focar apenas na informação necessária, uma vez que apenas precisamos de testar o valor de especialidade e contar o seu número de ocorrências. Uma vez que estamos perante uma procura por igualdade, seria benéfico utilizar um índice de hash.

3. create index especialidade_index on medico (especialidade)
   Nesta query seria idealmente utilizado o índice “especialidade” (ordenado e esparso). Neste cenário, estando as especialidades uniformemente distribuídas, estariam também ordenadas na tabela, permitindo minimizar o número de ponteiros que seria necessário manter para cada especialidade. No entanto, dadas as limitações do SGBD, será imprescindível usar novamente um índice não ordenado e denso, o que, tendo em conta que cada bloco possui apenas 2 registos, é de qualquer maneira benéfico (2 registos para 6 especialidades dá uma probabilidade de um bloco ter resposta de apenas ⅓, ou seja apenas teremos de ler 33% dos blocos!). Uma vez que estamos perante uma procura por igualdade, será, mais uma vez, benéfico utilizar um índice de hash.

4. create index medico_index on consulta (num_cedula, data)
   Neste caso teríamos várias opções de índice (todas implícitas) - num_cedula em hash, data em B+ tree, (num_cedula, data) em B+ tree. Esta última é a mais interessante neste caso. Será um índice ordenado e esparso, uma vez que é parte da primary key - bastaria trocar as colunas num_doente e data para garantir a sua ordenação. Dado o facto de ser um índice ordenado e a query envolver procura de ranges (a operação *between*, no caso) a melhor opção seria adotar um índice de B+ tree.

Modelo Multidimensional:

1. **Criação:**
   (Para este código funcionar é necessário apagar a linha 1086 do populate.sql (insert into prescricao_venda values (11, 15, '2020-12-02', 'Morfina', 57);), uma vez que viola o novo constraint num_venda unique)

```
drop table if exists d_tempo cascade;
drop table if exists d_instituicao cascade;
drop table if exists f_presc_venda cascade;
drop table if exists f_analise cascade;

alter table prescricao_venda drop constraint if exists num_venda_unique;

alter table prescricao_venda add constraint num_venda_unique unique (num_venda);

create table d_tempo (
        id_tempo serial not null,
        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,
        constraint pk_d_tempo primary key(id_tempo)
);

create table d_instituicao (
        id_inst serial not null,
        nome varchar(50) not null,
        tipo varchar(11) not null,
        num_regiao serial not null,
        num_concelho serial not null,
        constraint pk_d_instituicao primary key(id_inst),
        constraint fk_d_instituicao_instituicao foreign key(nome) references instituicao(nome),
        constraint fk_d_instituicao_concelho foreign key(num_regiao, num_concelho) references concelho(num_regiao, num_concelho)
);

create table f_presc_venda (
        id_presc_venda serial not null,
        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,
    constraint pk_f_presc_venda primary key(id_presc_venda),
    constraint fk_f_presc_venda_presc_venda foreign key(id_presc_venda) references prescricao_venda(num_venda),
    constraint fk_f_presc_venda_medico foreign key(id_medico) references medico(num_cedula),
    constraint fk_f_presc_venda_d_tempo foreign key(id_data_registo) references d_tempo(id_tempo),
    constraint fk_f_presc_venda_d_instituicao foreign key(id_inst) references d_instituicao(id_inst)
);

create table f_analise (
    id_analise serial not null,
    id_medico integer,
    num_doente integer,
    id_data_registo integer,
    id_inst integer not null,
    nome varchar(50) not null,
    quant integer not null,
    constraint pk_f_analise primary key(id_analise),
    constraint fk_f_analise_analise foreign key(id_analise) references analise(num_analise),
    constraint fk_f_analise_medico foreign key(id_medico) references medico(num_cedula),
    constraint fk_f_analise_d_tempo foreign key(id_data_registo) references d_tempo(id_tempo),
    constraint fk_f_analise_d_instituicao foreign key(id_inst) references d_instituicao(id_inst)
);
```

2. **Script de carregamento:**

```
insert into d_tempo (dia, dia_da_semana, semana, mes, trimestre, ano)
    select distinct extract(day from data) as dia,
                    extract(dow from data) as dia_da_semana,
                    extract(week from data) as semana,
                    extract(month from data) as mes,
                    extract(quarter from data) as trimestre,
                    extract(year from data) as ano
    from prescricao_venda;

insert into d_tempo(dia, dia_da_semana, semana, mes, trimestre, ano)
```

```
        select distinct extract(day from data_registo) as dia,
                        extract(dow from data_registo) as dia_da_semana,
                        extract(week from data_registo) as semana,
                        extract(month from data_registo) as mes,
                        extract(quarter from data_registo) as trimestre,
                        extract(year from data_registo) as ano
        from analise;

delete from d_tempo t_presc_venda using d_tempo t_analise
where t_presc_venda.id_tempo > t_analise.id_tempo and
        t_presc_venda.dia = t_analise.dia and
        t_presc_venda.mes = t_analise.mes and
        t_presc_venda.ano = t_analise.ano;

insert into d_instituicao(nome, tipo, num_regiao, num_concelho)
        select distinct nome, tipo, num_regiao, num_concelho
        from instituicao;

insert into f_presc_venda(id_presc_venda, id_medico, num_doente,
id_data_registo, id_inst, substancia, quant)
        select distinct prescricao_venda.num_venda as id_presc_venda,
num_cedula as id_medico, num_doente, id_tempo as id_data_registo, id_inst,
prescricao_venda.substancia, quant
        from ((d_tempo
        natural join d_instituicao)
        inner join prescricao_venda on (
                extract(day from prescricao_venda.data) = d_tempo.dia and
                extract(month from prescricao_venda.data)= d_tempo.mes and
                extract(year from prescricao_venda.data) = d_tempo.ano))
        inner join venda_farmacia on
                (venda_farmacia.num_venda = prescricao_venda.num_venda and
                 venda_farmacia.inst = d_instituicao.nome);

insert into f_analise(id_analise, id_medico, num_doente, id_data_registo, id_inst,
nome, quant)
        select distinct num_analise as id_analise, num_cedula as id_medico,
num_doente, id_tempo as id_data_registo, id_inst, analise.nome, quant
        from (d_tempo
        natural join d_instituicao)
        inner join analise on (
                extract(day from analise.data_registo) = d_tempo.dia and
                extract(month from analise.data_registo)= d_tempo.mes and
                extract(year from analise.data_registo) = d_tempo.ano and
                analise.inst = d_instituicao.nome);
```

Data Analytics:

1. **Número de análises de glicémia:**

```
SELECT d_tempo.mes, d_tempo.ano, analise.especialidade, COUNT(*)
FROM f_analise
INNER JOIN d_tempo ON (f_analise.id_data_registo = d_tempo.id_tempo)
INNER JOIN analise ON (f_analise.id_analise = analise.num_analise)
WHERE analise.nome = 'Glicémia (mg/dL)' AND d_tempo.ano >= 2017 AND d_tempo.ano <= 2020
GROUP BY
    GROUPING SETS ((d_tempo.mes, analise.especialidade ), (d_tempo.ano, analise.especialidade))
ORDER BY analise.especialidade, d_tempo.mes, d_tempo.ano;
```

2. **Quantidade total e nº médio de prescrições diário:**
   (Com o populate submetido no projeto anterior, esta query não apresentará resultados, uma vez que não existem prescrições na região de Lisboa no 1º semestre de 2020)
   (Os averages apenas têm em conta os dias em que existem prescrições-venda por uma questão de utilidade, uma vez que caso contrário todas as entradas seriam 0)

```
SELECT DISTINCT d_instituicao.num_regiao, d_instituicao.num_concelho, d_tempo.dia_da_semana, d_tempo.mes, COUNT(*) as total, COUNT(*) / COUNT(DISTINCT (d_tempo.semana, d_tempo.dia_da_semana)) AS average_by_day, COUNT(*) / COUNT(DISTINCT (d_tempo.dia, d_tempo.mes)) AS average_by_month, COUNT(*) / COUNT(DISTINCT (d_instituicao.num_concelho, d_tempo.dia, d_tempo.mes)) AS average_by_concelho
FROM f_presc_venda
INNER JOIN d_tempo ON f_presc_venda.id_data_registo = d_tempo.id_tempo
INNER JOIN d_instituicao ON f_presc_venda.id_inst = d_instituicao.id_inst
WHERE d_tempo.trimestre = 1 AND d_tempo.ano = '2020' AND d_instituicao.num_regiao = 3
GROUP BY
    ROLLUP ((d_instituicao.num_regiao, d_instituicao.num_concelho), (d_tempo.dia_da_semana), (d_tempo.mes))
HAVING COUNT(*) > 0
ORDER BY d_instituicao.num_regiao, d_instituicao.num_concelho, d_tempo.dia_da_semana, d_tempo.mes;
```