TÉCNICO LISBOA

Projeto de Bases de Dados - Parte 4

Professor Daniel Faria

Grupo 47 – Turno L07 – 4ª feira às 8h

Alunos	Esforço
Maria Duarte (90415)	20h (33.3%)
Daniela Carvalho (92443)	20h (33.3%)
Laura Baeta (92507)	20h (33.3%)

Restrições de Integridade

```
-- RI-100: um médico não pode dar mais de 100 consultas por semana na mesma
instituição
create or replace function consultas_medico_proc() returns trigger
as $$
begin
	if exists(

		with count_num_cedulas as(
			select m.num_cedula, count(m.num_cedula)
			from ((medico m inner join consulta c on (m.num_cedula =
c.num_cedula)) inner join instituicao i on (i.nome = c.nome_instituicao))
			where c.nome_instituicao = new.nome_instituicao and
c.num_cedula = new.num_cedula and extract(week from c.data_consulta) = extract(week
from new.data_consulta)
			group by m.num_cedula
		)

		select num_cedula
		from count_num_cedulas
		where count>=100
	)
	then
		raise exception 'O medico ja realizou mais de 100 consultas por semana
na mesma instituicao.';
	end if;
	return new;
end;
$$ language plpgsql;

-- o trigger acontece antes de se adicionar a consulta
create trigger max_consultas_medico before insert on consulta
	for each row execute procedure consultas_medico_proc();

-- RI-análise: numa análise, a consulta associada pode estar omissa; não estando, a
especialidade da consulta tem de ser igual à do médico.
create or replace function especialidade_proc() returns trigger
as $$
begin
	if exists(
		select *
		from consulta c inner join medico m on (m.num_cedula = c.num_cedula)
		where new.num_cedula = c.num_cedula and new.num_doente = c.num_doente
		and new.data_consulta = c.data_consulta and new.especialidade !=
m.especialidade
	)
	then
		raise exception 'Especialidade da consulta diferente da especialidade
do medico';
	end if;
	return new;
end;
$$ language plpgsql;

create trigger analise_especialidade before insert on analise
	for each row execute procedure especialidade_proc();
```

Índices

1- Listar as datas de consulta de um doente:

select data **from** consulta **where** num_doente = <um_valor>

Sabendo que os índices hash são ideais para seleção por igualdade, criamos um índice para organizar a coluna "num_doente" da tabela "consulta" através de uma Hash Table, o que facilita e torna mais eficiente a comparação do "num_doente" com o valor dado. Apesar de a chave primária da tabela consulta ser composta pelo "num_cedula", "num_doente" e "data", apenas o atributo "num_doente" é usado na seleção por igualdade após o **WHERE**, pelo que a função de dispersão (hash) recebe apenas o parâmetro "num_doente".

2- Considere que há apenas seis especialidades: "E1" a "E6". Pretende-se saber quantos médicos existem de cada especialidade.

select count(*) **from** medico **where** especialidade = "Ei"

em que Ei é uma das seis especialidades.

Neste caso, apesar de também ser uma comparação com um valor dado, consideramos que deve ser utilizado um índice bitmap sobre o atributo "especialidade", dado ao pequeno número de valores distintos para este atributo (apenas 6) em comparação com o número de rows na tabela "medico" (que ultrapassa em várias ordens de grandeza a memória disponível).

A tabela "medico" tem como chave primária o "num_cedula", porém este atributo não é relevante para a procura que nos é solicitada. Para além disso, como consideramos que o atributo "especialidade" não está sujeito a alterações constantes, a indexação através da criação de bitmaps para cada valor Ei do atributo "especialidade", em princípio, não criará problemas e é a que faz mais sentido.

3- Nomes dos médicos de uma determinada especialidade. Para a resolução desta alínea considere, para além do referido sobre a dimensão das tabelas, os seguintes aspetos:

1. Os blocos do disco são de 2K bytes e cada registo na tabela ocupa 1K bytes.
2. Os médicos estão uniformemente distribuídos pelas 6 especialidades.

select nome **from** medico **where** especialidade = 'Ei'

em que Ei é uma das seis especialidades.

Neste caso, vai ser necessário a criação de um índice através de uma B+ Tree desagrupado em que a chave de pesquisa é o atributo "nome" da tabela "medico". Isto obriga a uma leitura máxima de $\log_{(n/2)}$ (N). Como cada bloco do disco ocupa 2kB e cada registo ocupa 1kB, podemos então deferir que n≈1024 e, se existir um milhão valores de chave, acede-se apenas a $\log_{512}(1000000)\approx 2$ nós. Ou seja, um índice B+ permite-nos encontrar qualquer chave "nome" de uma forma muito mais eficiente. Resumindo, para blocos muito grandes o B+ Tree mostra-se bastante eficiente.

E um índice Hash denso e desagrupado para a chave "especialidade" da tabela "medico" permite-nos obter mais rapidamente os dados a partir da procura na Hash table, mais eficiente do que a B+ neste caso porque no **WHERE** é feita uma seleção por igualdade.

4- Listar os nomes dos médicos que deram consultas entre duas datas.

select nome **from** medico, consulta

where consulta.num_celula=medico.num_celula AND

consulta.data BETWEEN 'data_1' AND 'data_2'

em que 'data_1' e 'data_2' são duas datas.

Para a seleção de igualdade do atributo "num_cedula" nas tabelas "consulta" e "medico" é usada uma Hash Table na qual passamos como parâmetro para a função de dispersão o atributo "num_cedula" pois estas são ideais para as situações de seleção de igualdade, como é o caso.

Seguidamente, consideramos que faz sentido indexarmos também por B+ Trees em relação ao atributo "data" da tabela "consulta" devido ao facto de estarmos à procura de um intervalo entre duas datas e o B+ Tree permitir percorrer facilmente e com rapidez um intervalo de valores do tipo "date" (que pode ser classificado segundo uma ordem).

Nesta situação, o B+ Tree consegue percorrer o intervalo de valores dado para o atributo "consulta.data" visto que existem ligações por ponteiros ao nível das folhas (o que, por exemplo, não se verifica na B Tree). Isto permite ainda que, partindo da raíz, seja possível chegar a todas as folhas.

Não faz sentido usar a chave primária da tabela consulta pois esta é composta pelo "num_cedula", "num_doente" e "data" e o atributo "num_doente" não é necessário para a indexação, pelo que criamos uma nova chave composta apenas pelos atributos "num_cedula" e "data".

NOTA: No schema.sql da entrega 3, o atributo "data" da tabela consulta foi alterado por nós para "data_consulta", visto que a palavra data estava a gerar problemas quando corríamos no postgresql.

Modelo Multidimensional

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

create table d_tempo(
	id_tempo serial not null,
	dia int not null check (dia < 32 and dia > 0),
	dia_da_semana int not null check (dia_da_semana<=6 and dia_da_semana>=0),
	semana int not null,
	mes int not null check (mes < 13 and mes > 0),
	trimestre int not null check (trimestre < 5 and trimestre > 0),
	ano int not null,
	constraint pk_d_tempo primary key (id_tempo));

create table d_instituicao(
	id_inst serial not null,
	nome varchar(60) not null,
	tipo varchar(11) not null check(tipo in ('Farmacia', 'Laboratorio', 'Clinica',
'Hospital')),
	num_regiao varchar(1) not null,
	num_concelho varchar(3) not null,
	constraint pk_d_instituicao primary key(id_inst));

create table f_presc_venda(
	id_presc_venda varchar(23) not null unique, --id_presc_venda = num_venda
	id_medico varchar(5) not null, --id_medico = num_cedula

	num_doente varchar(9) not null,
```

```
	id_data_registo int not null,
	id_inst int not null,
	substancia varchar(50) not null,
	quant varchar(10) not null,
	constraint pk_f_presc_venda primary key(id_presc_venda),
	constraint fk_presc_venda_tempo foreign key(id_data_registo) references
d_tempo(id_tempo) ON DELETE CASCADE,
	constraint fk_presc_venda_tempo_inst foreign key(id_inst) references
d_instituicao(id_inst) ON DELETE CASCADE);

create table f_analise(
	id_analise varchar(7) not null unique, --id_analise = num_analise
	id_medico varchar(5) not null,
	num_doente varchar(9) not null,
	id_data_registo int not null,
	id_inst int not null,
	nome varchar(30) not null,
	quant numeric(4,1) not null,
	constraint pk_f_analise primary key(id_analise),
	constraint fk_analise_tempo foreign key(id_data_registo) references
d_tempo(id_tempo) ON DELETE CASCADE,
	constraint fk_analise_inst foreign key(id_inst) references
d_instituicao(id_inst) ON DELETE CASCADE);
```

ETL de carregamento

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

-- id_tempo, dia, dia_da_semana, semana, mes, trimestre, ano
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
where not exists (
	select dia, mes, ano
	from d_tempo
	where dia=extract (day from data_registo) and mes=extract (month from
data_registo) and ano=extract (year from data_registo));

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 ( venda_farmacia vf inner join
prescricao_venda pv on (vf.num_venda = pv.num_venda) )

where not exists (
	select dia, mes, ano
	from d_tempo
	where dia=extract (day from data_registo) and mes=extract (month from
data_registo) and ano=extract (year from data_registo));
```

```
-- id_presc_venda, id_medico, num_doente, id_data_registo, id_inst, substancia, quant
insert into f_presc_venda(id_presc_venda, id_medico, num_doente, id_data_registo, id_inst, substancia, quant)
select pv.num_venda, pv.num_cedula, pv.num_doente, dt.id_tempo, di.id_inst, pv.substancia, vf.quant from (( venda_farmacia vf inner join prescricao_venda pv on (vf.num_venda = pv.num_venda) ) inner join d_tempo dt on (dt.dia = extract(day from vf.data_registo) and dt.mes = extract(month from vf.data_registo) and dt.ano = extract(year from vf.data_registo)) inner join d_instituicao di on (di.nome = vf.inst));

-- id_analise, id_medico, num_doente, id_data_registo, id_inst, nome, quant)
insert into f_analise(id_analise, id_medico, num_doente, id_data_registo, id_inst, nome, quant)

select a.num_analise, a.num_cedula, a.num_doente, dt.id_tempo, di.id_inst, a.nome, a.quant from (analise a inner join d_instituicao di on (a.inst = di.nome) inner join d_tempo dt on (dt.dia = extract(day from a.data_registo) and dt.mes = extract(month from a.data_registo) and dt.ano = extract(year from a.data_registo)));
```

Queries OLAP

1-

```
select a.especialidade, dt.mes, dt.ano, count(a.num_analise) as numero_de_analises
from analise a inner join d_tempo dt on (dt.dia = extract(day from a.data_registo)and dt.mes = extract(month from a.data_registo) and dt.ano = extract(year from a.data_registo))
where a.nome='Glicemia' and dt.ano>=2017 and dt.ano<=2020
group by cube (a.especialidade, dt.mes, dt.ano);
```

2-

```
with tabela_pres as (
	select fpv.substancia, dt.dia_da_semana, dt.mes, c.nome as nome_c, dt.trimestre, r.nome as nome_r, count(fpv.id_presc_venda) as quantidade_total
	from (((f_presc_venda fpv inner join d_instituicao di on (fpv.id_inst = di.id_inst)) inner join d_tempo dt on (fpv.id_data_registo = id_tempo))
	inner join concelho c on (di.num_regiao = c.num_regiao and di.num_concelho = c.num_concelho)) inner join regiao r on (c.num_regiao = r.num_regiao)
	where dt.trimestre=1 and r.nome = 'Lisboa'
	group by rollup (fpv.substancia, dt.dia_da_semana, dt.mes, nome_c, dt.trimestre, nome_r))

select substancia, dia_da_semana, mes, nome_c, trimestre, nome_r, quantidade_total, avg(quantidade_total) as num_medio_diario
from tabela_pres as t
group by (substancia, (dia_da_semana, mes), nome_c, trimestre, nome_r, quantidade_total);
```

Nota: Fizemos pequenas alterações no ficheiro schema.sql, para melhoria do desempenho do código, tais como acrescentar “on delete cascade on update cascade” em relações foreign key entre tabelas, bem como, na tabela “analise”, alterar o tipo do atributo “quant” de varchar(10) para numeric(4,1).