TÉCNICO LISBOA

Projeto de Bases de Dados

Parte 4

Nome	***Número de Aluno***	***Percentagem de Contribuição***	***Esforço em horas***
Joana Maria Leal Coutinho	87666	33.3%	10h
João Rafael Pinto Soares	87675	33.3%	10h
Pedro M. S. P. Rodrigues	87696	33.3%	10h

Grupo Nº 3

Turno BD817957L08 (6ª feira, 8:30 – 10:00)

Professor André Vasconcelos

Implementação das Restrições de Integridade

a)

```
create or replace function check_solicita() returns trigger as $$
   begin
      if not exists(
            select *
            from Audita natural join EventoEmergencia natural join Vigia
            where idCoordenador = new.idCoordenador and numCamara = new.numCamara
      ) then
         raise exception 'Um coordenador apenas pode solicitar videos de camaras colocadas num local cujo accionamento de meios esteja a ser (ou tenha side) auditado por ele proprio.';
      end if;
      return new;
   end
$$ language plpgsql;

create trigger check_solicita_constraint before insert or update on Solicita
   for each row execute procedure check_solicita();
```

b)

```
create or replace function check_acciona() returns trigger as $$
   begin
      if not exists(
            select *
            from Acciona
            where numMeio = new.numMeio
               and nomeEntidade = new.nomeEntidade
               and numProcessoSocorro = new.numProcessoSocorro
      ) then
         raise exception 'Um Meio de Apoio só pode ser alocado a Processos de Socorro para os quais tenha sido accionado.';
      end if;
      return new;
   end
$$ language plpgsql;

create trigger check_alocado_acciona_constraint before insert or update on Alocado
   for each row execute procedure check_acciona();
```

Índices

a)

No primeiro query, são utilizadas as tabelas *Video* e *Vigia*. É feito o produto cartesiano entre estas, selecionando entradas cujo *numCamara* para cada uma das tabelas é igual. São escolhidas, em particular, linhas referentes ao *numCamara* 10 e à *moradaLocal* "Loures". Assim, nesta query apenas são usadas **comparações de igualdade**, nomeadamente, entre as colunas *numCamara* e *moradaLocal*. Pelo contexto do problema, é de esperar que uma câmara vigíe um número muito reduzido de locais.

Tendo tudo isto em conta, o melhor tipo de índice para a tabela *Video* é **primário usando Hash** sobre o atributo *numCamara*. Para a tabela *Vigia*, o melhor tipo de índice é também **primário usando Hash** sobre o atributo *numCamara* e secundário, também Hash, sobre o atributo *moradaLocal*. O **índice secundário**, embora se espere, pela razão mencionada acima (número reduzido de locais para cada câmara na tabela Vigia), que não tenha impacto, caso exista, por exemplo, uma câmara num satélite, vigiando assim inúmeras localidades, poderia existir uma melhoria significativa de eficiência.

Este tipo de índices permitiria obter complexidade de $O(1)$ na seleção de *numCamara* para cada tabela, tendo depois complexidade de $O(n)$ (sendo n o número de linhas da tabela *Vigia*) no pior caso. Apesar disto, como foi adicionado um índice secundário sobre *moradaLocal*, acabaria por ser mais eficiente utilizar este índice caso existissem múltiplas localidades para uma só câmara. Portanto, esta **complexidade será** na maioria dos casos $\boldsymbol{O(1)}$**.**

No segundo query, são utilizadas as tabelas *Transporta* e *EventoEmergencia*. É feito o produto cartesiano entre estas, selecionando entradas, agrupadas conjuntamente pelo *numTelefone* e *instanteChamada*, cujo *numProcessoSocorro* para cada uma das tabelas é igual. Assim, tal como na primeira query, apenas são usadas **comparações de igualdade**.

Deste modo, o melhor tipo de índice para a tabela *Transporta* é **primário usando Hash** sobre o atributo *numProcessoSocorro*. Para a tabela *EventoEmergencia*, o melhor tipo de índice é também **primário usando Hash** sobre os atributos *numTelefone* e *instanteChamada* conjuntamente. O **índice secundário** mais apropriado é sobre o atributo *numProcessoSocorro* usando também **Hash**.

b)

create index **video_primary_idx** on **Video** using **hash**(numCamara);

create index vigia_primary_idx on **Vigia** using **hash**(numCamara);
create index vigia_secondary_idx on **Vigia** using **hash**(moradaLocal);

create index transporta_primary_idx on **Transporta** using **hash**(numProcessoSocorro);

create index eventoEmergencia_primary_idx on **EventoEmergencia** using **hash**(numTelefone, instanteChamada);
create index eventoEmergencia_secondary_idx on **EventoEmergencia** using **hash**(numProcessoSocorro);

Modelo Multidimensional

drop table d_evento cascade;
drop table d_meio cascade;
drop table d_tempo cascade;
drop table facts cascade;

create table d_evento
(idEvento **serial not null**,
numTelefone **numeric(9) not null**,
instanteChamada **timestamp not null**,
constraint pk_d_evento primary key(idEvento));

create table d_meio
(idMeio **serial not null**,
numMeio **numeric not null**,
nomeMeio **varchar(80) not null**,
nomeEntidade **varchar(80) not null**,
tipo **varchar(80) not null**,
constraint pk_d_meio primary key(idMeio));

create table d_tempo
(idTempo **serial not null**,
dia **numeric(2) not null**,
mes **numeric(2) not null**,
ano **numeric(4) not null**,
unique(dia, mes, ano),
constraint pk_d_tempo primary key(idTempo));

```
create table facts
        (idEvento serial,
         idMeio serial,
         idTempo serial,
         constraint pk_facts primary key(idEvento, idMeio, idTempo),
         constraint fk_facts_d_evento foreign key(idEvento) references
d_evento(idEvento),
         constraint fk_facts_d_meio foreign key(idMeio) references
d_meio(idMeio),
         constraint fk_facts_d_tempo foreign key(idTempo) references
d_tempo(idTempo));

create or replace function get_dates() returns void as $$
        declare
                min_date date;
                max_date date;
        begin
                Select min(instanteChamada)::timestamp::date into min_date
                From EventoEmergencia;

                Select max(instanteChamada)::timestamp::date into max_date
                From EventoEmergencia;

                while min_date <= max_date LOOP
                insert into d_tempo(dia, mes, ano) values (EXTRACT(day from
min_date), EXTRACT(month from min_date), EXTRACT(year from min_date));
                min_date := min_date + interval '1 day';
                end loop;
        end;
$$ language plpgsql;

insert into d_evento(numTelefone, instanteChamada)
select numTelefone, instanteChamada
from EventoEmergencia;

insert into d_meio(numMeio, nomeEntidade, nomeMeio, tipo)
select *, 'MeioCombate'
from Meio natural join MeioCombate;

insert into d_meio(numMeio, nomeEntidade, nomeMeio, tipo)
select *, 'MeioApoio'
from Meio natural join MeioApoio;
```

```
insert into d_meio(numMeio, nomeEntidade, nomeMeio, tipo)
select *, 'MeioSocorro'
from Meio natural join MeioSocorro;

select get_dates();
insert into facts
select
        idEvento, idMeio, idTempo
from EventoEmergencia natural join Acciona natural join d_evento natural join
d_meio natural join d_tempo
Where EXTRACT(day from instanteChamada) = dia and
        EXTRACT(month from instanteChamada) = mes and
        EXTRACT(year from instanteChamada) = ano;
```

Data Analytics

```
Select ano, mes, tipo, count(tipo)
From facts NATURAL JOIN d_meio NATURAL JOIN d_tempo
Where idEvento = 15
Group by tipo, ano, mes
UNION
Select ano, null, tipo, count(tipo)
From facts NATURAL JOIN d_meio NATURAL JOIN d_tempo
Where idEvento = 15
Group by tipo, ano, mes
UNION
Select null, null, tipo, count(tipo)
From facts NATURAL JOIN d_meio NATURAL JOIN d_tempo
Where idEvento = 15
Group by tipo, ano, mes;
```