Projeto de Bases de Dados, Parte 4

Grupo 47

Daniel Fernandes 86400 &
Francisco Sousa 86416 & Henrique Ferreira 86432

14 de Dezembro de 2018

Turno BD22517957L09
Sexta 12:30 Lab 8
Prof. Taras Lykhenko

Número de Aluno	**Nome**	**Esforço**
86400	Daniel Fernandes	33% (10h)
86416	Francisco Sousa	33% (10h)
86432	Henrique Ferreira	33% (10h)

1 Restrições de Integridade

Os mecanismos que achamos mais apropriados para a definição destas restrições de integridade são os triggers. Fazem consultas mais complexas do que o `CHECK` e permitem lançar exceções. Ainda, as operações com que trabalham não provocam triggers em cadeia.

Assim, para cada problema, temos:

a)

```
CREATE OR REPLACE FUNCTION chk_coord_local_on_solic_proc()
RETURNS TRIGGER
AS $$
DECLARE morada_camara VARCHAR;
BEGIN
    SELECT morada_local INTO morada_camara FROM vigia WHERE num_camara = NEW.num_camara;
    IF EXISTS
        (SELECT * FROM (audita NATURAL JOIN evento_emergencia) AS r
            WHERE r.morada_local = morada_camara
            AND r.id_coordenador = NEW.id_coordenador)
    THEN
        RETURN NEW;
    ELSE
        RAISE EXCEPTION 'Um coordenador so pode solicitar videos de camaras colocadas num local
        cujo accionamento de meios esteja a ser (ou tenha sido) auditado por ele proprio!';
        RETURN NULL;
    END IF;
END;
$$ LANGUAGE plpgsql;

DROP trigger IF EXISTS chk_coord_local_on_solic ON solicita;
CREATE trigger chk_coord_local_on_solic BEFORE INSERT ON solicita for each ROW EXECUTE procedure chk_coord_lo
```

b)

```
CREATE OR REPLACE FUNCTION chk_acciona_on_alocado_proc()
RETURNS TRIGGER
AS $$
BEGIN
    IF EXISTS
        (SELECT * FROM acciona
            WHERE num_meio = NEW.num_meio
            AND nome_entidade = NEW.nome_entidade
            AND num_processo_socorro = NEw.num_processo_socorro)
    THEN
        RETURN NEW;
    ELSE
        RAISE EXCEPTION 'Um Meio de Apoio so pode ser alocado a Processos de Socorro para
        os quais tenha sido accionado!';
        RETURN NULL;
    END IF;
END;
$$ LANGUAGE plpgsql;

DROP trigger IF EXISTS chk_acciona_on_alocado ON alocado;
CREATE trigger chk_acciona_on_alocado BEFORE INSERT ON alocado for each ROW EXECUTE procedure chk_acciona_on_
```

2 Índices

a)

Primeira Interrogação Os atributos verificados são *num_camara* na tabela **video** e *num_camara* e *morada_local* na tabela **vigia**. Assim, para a tornar mais eficaz, fazemos dois índices:

O primeiro do tipo hash, para *num_camara* na tabela **video**, pois lidamos com comparações entre números, sendo agrupado e esparso.

O segundo do tipo balance tree, para *num_camara* e *morada_local* na tabela **vigia**, sendo desagrupado e denso.

Segunda Interrogação Perante o `GROUP BY`, torna-se adequado usar um índice balanced tree composto para os atributos *num_telefone* e *instante_chamada*. São ainda criados outros dois índices hash para *num_processo_socorro*, tanto na tabela **transporta** como **evento_emergencia**, para tornar mais eficaz a sua comparação, já que são variáveis únicas.

b) Assumindo que não existem quaisquer índices predefinidos sobre as tabelas, foram criados os seguintes:

```
drop index num_camara_idx;
drop index morada_local_idx;
drop index num_processo_socorro_idx;
drop index num_processo_socorro_e_idx;

create index num_camara_idx on video using hash(num_camara);
create index morada_local_idx on vigia using btree(num_camara, morada_local);
create index group_by_idx on evento_emergencia using btree(num_telefone, instante_chamada);
create index num_processo_socorro_e_idx on evento_emergencia using hash(num_processo_socorro);
create index num_processo_socorro_t_idx on transporta using hash(num_processo_socorro);
```

3 Modelo Multidimensional

O esquema em estrela é, representado na figura 1, pode ser definido com as interrogações:

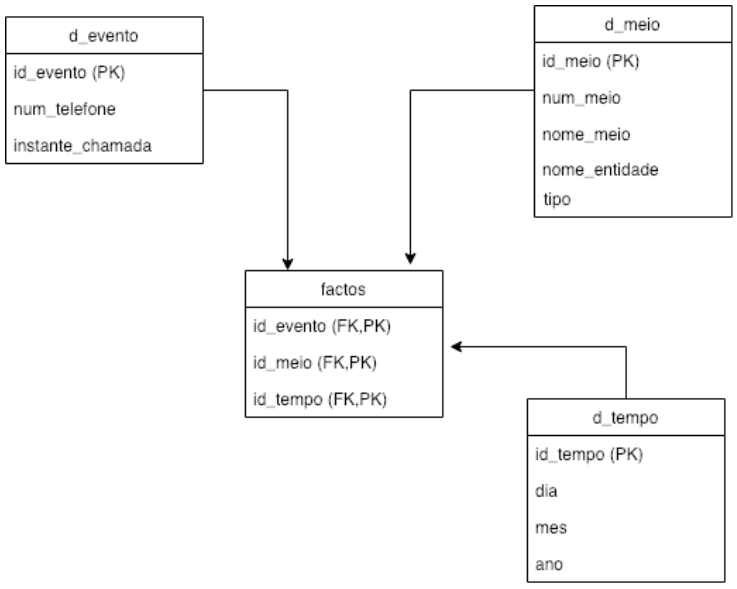

Figura 1: Esquema em estrela

```
drop table factos;
drop table d_evento;
drop table d_meio;
drop table d_tempo;

create table d_evento
    (id_evento serial,
    num_telefone char(9) not null,
    instante_chamada timestamp not null,
    constraint pk_id_evento primary key(id_evento)
);
```

```
create table d_meio
    (id_meio serial,
    num_meio char(5) not null,
    nome_meio varchar(255) not null,
    nome_entidade varchar(20) not null,
    tipo varchar(20) not null,
    constraint pk_id_meio primary key (id_meio)
);

create table d_tempo
    (id_tempo serial,
     dia int not null,
     mes int not null,
     ano int not null,
     constraint pk_id_tempo primary key(id_tempo)
);

create table factos
    (id_tempo int not null,
     id_meio int not null,
     id_evento int not null,
     constraint pk_factos primary key(id_tempo, id_meio, id_evento),
     constraint fk_tempo foreign key (id_tempo) references d_tempo(id_tempo),
     constraint fk_meio foreign key (id_meio) references d_meio(id_meio),
     constraint fk_evento foreign key (id_evento) references d_evento(id_evento)
);
```

Usámos as seguintes queries para popular o esquema em estrela:

```
insert into d_evento (num_telefone, instante_chamada)
    select num_telefone, instante_chamada from evento_emergencia;

insert into d_meio (num_meio, nome_meio, nome_entidade, tipo)
    select num_meio, nome_meio, nome_entidade, 'apoio' from meio natural join meio_apoio;
insert into d_meio (num_meio, nome_meio, nome_entidade, tipo)
    select num_meio, nome_meio, nome_entidade, 'combate' from meio natural join meio_combate;
insert into d_meio (num_meio, nome_meio, nome_entidade, tipo)
    select num_meio, nome_meio, nome_entidade, 'socorro' from meio natural join meio_socorro;

insert into d_tempo (dia, mes, ano) values(22, 4, 2005);
insert into d_tempo (dia, mes, ano) values(15, 2, 2009);
insert into d_tempo (dia, mes, ano) values(29, 2, 2018);
insert into d_tempo (dia, mes, ano) values(27, 8, 1998);
insert into d_tempo (dia, mes, ano) values(28, 8, 1996);
insert into d_tempo (dia, mes, ano) values(1, 10, 1999);
insert into d_tempo (dia, mes, ano) values(7, 8, 1997);
insert into d_tempo (dia, mes, ano) values(27, 1, 1999);
insert into d_tempo (dia, mes, ano) values(3, 7, 2007);
insert into d_tempo (dia, mes, ano) values(3, 9, 2018);
insert into d_tempo (dia, mes, ano) values(9, 8, 1998);
insert into d_tempo (dia, mes, ano) values(4, 7, 2000);
insert into d_tempo (dia, mes, ano) values(11, 9, 2001);
insert into d_tempo (dia, mes, ano) values(24, 7, 2015);

insert into factos
    select id_tempo, id_meio, id_evento from d_evento cross join d_meio cross join d_tempo;
```

4 Data analytics

Considerando o esquema em estrela criado anteriormente, fizemos duas interrogações SQL OLAP para obter o número de meios de cada tipo utilizados no evento número 15, com rollup por ano e mês. Ambas mostram o número de meios ordenados por cada um dos critérios *mes*, *ano*, *tipo*, em ordem ascendente, listando a soma dos tipos, tanto por mês, como por ano, como por tipo.

Para versão de postgres inferior a 9.5, usamos `UNION`:

```
select tipo, ano, mes, count(*)
    from (d_tempo natural join d_meio natural join
        (select * from factos where id_evento = '15') as a)
    group by tipo, ano, mes
union
select tipo, ano, null, count(*)
    from (d_tempo natural join d_meio natural join
        (select * from factos where id_evento = '15') as a)
    group by tipo, ano
union
select tipo, null, null, count(*)
    from (d_tempo natural join d_meio natural join
        (select * from factos where id_evento = '15') as a)
    group by tipo
union
select null, null, null, count(*)
    from (d_tempo natural join d_meio natural join
        (select * from factos where id_evento = '15') as a)
order by tipo, ano, mes;
```

Usando `ROLLUP`, para versão postgres igual ou superior a 9.5:

```
select tipo, ano, mes, count(*)
    from (d_tempo natural join d_meio natural join
    (select * from factos where id_evento = '15') as a)
group by rollup (tipo, ano, mes)
order by tipo, ano, mes;
```