**Projeto de Bases de Dados**, Parte 4



**Grupo nº 38**

**Turno de Segunda feira 12:30 - Lab14**

**Docente: Taras Lykhenko**

|  |  |  |
| --- | --- | --- |
| **Aluno** | **Esforço (em horas)** | **Percentagem relativa de contribuição** |
| Francisco Nicolau - 86419 | 20 | 33.(3)% |
| Manuel Correia     - 86470 | 20 | 33.(3)% |
| Miguel Valério       - 86483 | 20 | 33.(3)% |

IST, LEIC-A, 2017/2018   Base de Dados

**Restrições de Integridade**

As condições foram satisfeitas recorrendo ao uso de *triggers*.

* Um Coordenador só pode solicitar vídeos de câmaras colocadas num local cujo acionamento de meios esteja a ser (ou tenha sido) auditado por ele próprio.

CREATE OR REPLACE FUNCTION **check\_Solicita**() RETURNS TRIGGER AS $body$

BEGIN

IF NOT EXISTS (

SELECT \*

FROM (Vigia NATURAL JOIN EventoEmergencia NATURAL JOIN Audita) T

WHERE T.idCoordenador = new.idCoordenador AND

T.numCamara = new.numCamara )

THEN

RAISE EXCEPTION 'O Coordenador % nao pode solicitar videos da camara %.', new.idCoordenador, new.numCamara

USING HINT = 'Um Coordenador so pode solicitar vídeos de camaras colocadas num local cujo acionamento de meios esteja a ser (ou tenha sido) auditado por ele proprio.';

END IF;

RETURN NEW;

END;

$body$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS **check\_Solicita\_trigger** ON Solicita CASCADE;

CREATE TRIGGER **check\_Solicita\_trigger** BEFORE INSERT OR UPDATE ON Solicita FOR EACH ROW EXECUTE PROCEDURE check\_Solicita();

* Um Meio de Apoio só pode ser alocado a Processos de Socorro para os quais tenha sido accionado.

CREATE OR REPLACE FUNCTION **check\_Alocado**() RETURNS TRIGGER AS $body$

BEGIN

IF NOT EXISTS (

SELECT \*

FROM Acciona A

WHERE A.numMeio = new.numMeio AND

A.nomeEntidade = new.nomeEntidade AND

A.numProcessoSocorro = new.numProcessoSocorro )

THEN

RAISE EXCEPTION 'O Meio de Apoio (%, %) nao pode ser alocado pelo Processo de Socorro %.', new.numMeio, new.nomeEntidade, new.numProcessoSocorro

USING HINT = 'Um Meio de Apoio so pode ser alocado a Processos de Socorro para os quais tenha sido accionado.';

END IF;

RETURN NEW;

END;

$body$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS **check\_Alocado\_trigger** ON Alocado CASCADE;

CREATE TRIGGER **check\_Alocado\_trigger** BEFORE INSERT OR UPDATE ON Alocado FOR EACH ROW EXECUTE PROCEDURE check\_Alocado();

**Índices**

Em relação à primeira consulta deveriam ser criados dois índices para melhorar a performance. Em primeiro lugar, um índice de dispersão dinâmica na tabela Video relativo à coluna numCamara que seja UNIQUE, pois esta coluna contém o número de cada camara e estes são únicos, este é denso e desagrupado. Em segundo lugar, na tabela Vigia, teríamos um índice composto em BTree denso e desagrupado de atributos moradaLocal e numCamara.

CREATE INDEX ncam ON Video USING hash (numCamara);

CREATE INDEX cam\_ind ON Vigia (moradaLocal,numCamara);

O primeiro é útil dado que leva a uma procura facilitada da tabela Video pois pela utilização da *hashing function*, é determinada a linha de interesse ou uma próxima sendo a procura substancialmente reduzida, neste caso para encontrar a(s) de valor 10. O segundo permite uma eficiente seleção de dados na tabela pelo facto de ambos os atributos serem necessários à *query*, quer pela comparação de igualdade com o atributo da outra tabela e se restringir o segundo atributo a um valor especifico.

No que toca à segunda consulta, para ambas as tabelas (Transporta e EventoEmergencia) deveria ser criado um índice BTree relativo à coluna numProcessoSocorro. Em ambos os casos seriam desagrupados e densos. Relativamente aos atributos usados na campo GROUP BY, idealmente (se o *postgres* suporta-se) iriamos criar um índice de dispersão dinâmica de chave composta. No entanto, dado que isso não é possível e que nenhum outro índice é benéfico pelo facto dos atributos usados no campo serem a *Primary Key* da tabela EventoEmergencia, isto devido a já existir um índice interno que é o utilizado.

CREATE INDEX nProc ON Transporta (numProcessoSocorro);

CREATE INDEX nProc ON EventoEmergencia (numProcessoSocorro);

Ainda, após termos realizado testes na base de dados constatamos que apesar de em teoria estes índices serem benéficos não são utilizados pelo SGBD.

**Modelo Multidimensional**

***Drop* de possíveis tabelas anteriores** com o mesmo nome das que serão criadas:

drop table if exists **d\_evento** cascade;

drop table if exists **d\_meio** cascade;

drop table if exists **d\_tempo** cascade;

drop table if exists **facts** cascade;

**Criação das novas tabelas**:

create table **d\_evento** (

idEvento serial,

numTelefone varchar(9) not null,

instanteChamada timestamp not null,

primary key (idEvento) );

create table **d\_meio** (

idMeio serial,

numMeio integer not null,

nomeMeio varchar(30) not null,

nomeEntidade varchar(200) not null,

tipo varchar(7) not null,

primary key (idMeio) );

create table **d\_tempo** (

idData serial,

dia integer not null,

mes integer not null,

ano integer not null,

primary key (idData) );

create table **facts** (

idFact serial,

idEvento integer not null,

idMeio integer not null,

idData integer not null,

primary key(idFact),

foreign key (idEvento)

references d\_evento(idEvento) on delete cascade on update cascade,

foreign key (idMeio)

references d\_meio(idMeio) on delete cascade on update cascade,

foreign key (idData)

references d\_tempo(idData) on delete cascade on update cascade );

**Inserção de valores** nas tabelas:

INSERT INTO **d\_evento** (numTelefone, instanteChamada)

SELECT numTelefone, instanteChamada FROM EventoEmergencia

ORDER BY instanteChamada, numtelefone;

INSERT INTO **d\_tempo** (dia, mes, ano)

SELECT extract(day from instanteChamada) as dia,

extract(months from instanteChamada) as mes,

extract(year from instanteChamada) as ano

FROM EventoEmergencia

ORDER BY ano, mes, dia;

INSERT INTO **d\_meio** (numMeio, nomeMeio, nomeEntidade, tipo)

SELECT \*

FROM (

SELECT numMeio, nomeMeio, nomeEntidade, 'Apoio'

FROM MeioApoio NATURAL JOIN Meio

UNION

SELECT numMeio, nomeMeio, nomeEntidade, 'Socorro'

FROM MeioSocorro NATURAL JOIN Meio

UNION

SELECT numMeio, nomeMeio, nomeEntidade, 'Combate'

FROM MeioCombate NATURAL JOIN Meio

UNION

SELECT numMeio, nomeMeio, nomeEntidade, 'Nenhum'

FROM Meio NATURAL JOIN (

SELECT numMeio, nomeEntidade

FROM Meio

EXCEPT

( SELECT \*

FROM MeioApoio

UNION

SELECT \*

FROM MeioSocorro

UNION

SELECT \*

FROM MeioCombate )

) Meios\_Sem\_Tipo

) T\_Meio

ORDER BY numMeio, nomeEntidade, nomeMeio;

INSERT INTO **facts** (idEvento, idMeio, idData)

SELECT idEvento, idMeio, idData

FROM d\_evento

NATURAL JOIN EventoEmergencia E

NATURAL JOIN Acciona A

NATURAL JOIN d\_meio M

INNER JOIN d\_tempo T ON (

extract(DAY FROM E.instantechamada) = T.dia AND

extract(MONTH FROM E.instantechamada) = T.mes AND

extract(YEAR FROM E.instantechamada) = T.ano)

ORDER BY idEvento, idMeio, idData;

**Data Analytics**

Para se obter o número de meios de cada tipo distinto utilizados no evento número 15, agrupados por mês e ano, pensamos em usar uma QUERY OLAP com GROUP BY ROLLUP (ano, mês). No entanto, devido à versão do *postgres* usada, esta não é suportada e como tal recorremos à utilização de GROUP BY e UNION para atingir o mesmo resultado.

WITH Tmeio15 AS (

SELECT \*

FROM d\_meio NATURAL JOIN facts NATURAL JOIN d\_tempo

WHERE idEvento = 15

)

SELECT \*

FROM (

SELECT tipo, ano, mes, count(\*) AS nMeios

FROM Tmeio15

GROUP BY tipo, ano, mês

UNION

SELECT tipo, ano, null, count(\*) AS nMeios

FROM Tmeio15

GROUP BY tipo, ano

UNION

SELECT tipo, null, null, count(\*) AS nMeios

FROM Tmeio15

GROUP BY tipo

) T

ORDER BY tipo, ano, mes, nMeios;