Projeto de Base de Dados

Parte 4

Nome	Número	Contribuição	Horas
Maria Inês Morais	83609	33,(3) %	18
João Antunes	87668	33,(3) %	18
Viviana Bernardo	87709	33,(3)%	18

Grupo nº 7

Turno: Terça-feira das 08:30 às 10:00

Docente: Professor Carlos Mendes

TÉCNICO LISBOA

Restrições de Integridade

1. **DROP TRIGGER IF EXISTS** solicita_videos_trigger **ON** solicita;

```
CREATE OR REPLACE FUNCTION solicita_videos() RETURNS TRIGGER AS $$
BEGIN
        IF NOT EXISTS  (SELECT * FROM vigia
                        WHERE numCamara = new.numCamara
                        AND moradaLocal IN
                        (SELECT moradaLocal
                        FROM audita NATURAL JOIN eventoEmergencia
                        WHERE idCoordenador = new.idCoordenador))
        THEN
                RAISE EXCEPTION 'Um Coordenador so pode solicitar videos de camaras colocadas num local cujo accionamento de meios esteja a ser (ou tenha sido) auditado por ele proprio';
        END IF;
        RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER solicita_videos_trigger BEFORE INSERT OR UPDATE ON solicita
        FOR EACH ROW EXECUTE PROCEDURE solicita_videos();
```

2. **DROP TRIGGER IF EXISTS** aloca_meio_trigger **ON** alocado;

```
CREATE OR REPLACE FUNCTION aloca_meio() RETURNS TRIGGER AS $$
BEGIN
        IF (new.numProcessoSocorro NOT IN (SELECT numProcessoSocorro FROM acciona WHERE numMeio = new.numMeio AND nomeEntidade = new.nomeEntidade))
        THEN
                RAISE EXCEPTION 'Um Meio de Apoio so pode ser alocado a Processos de Socorro para os quais tenha sido accionado';
        END IF;
        RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER aloca_meio_trigger BEFORE INSERT OR UPDATE ON alocado
        FOR EACH ROW EXECUTE PROCEDURE aloca_meio();
```

Índices

Tivemos as seguintes considerações iniciais:

- o PostgreSQL cria índices por predefinição em várias situações como, por exemplo, para as chaves primárias de cada tabela ou para valores que tenham a restrição unique. Os índices criados para cada resposta foram escolhidos como se estes não existissem.
- as tabelas da nossa base de dados têm apenas cerca de 100 entradas cada o que não permite ver alterações consideráveis no tempo de execução ao criar índices.

QUERY 1

video_idx - Decidimos criar um índice do tipo hash sobre a tabela video para o atributo numCamera. Escolhemos usar um índice do tipo hash, pois este é o mais eficiente para testes de igualdade. Ao utilizar este tipo de índice para aceder a um dado elemento basta-nos computar a sua função de dispersão para encontrar a sua localização, ao invés de termos de percorrer toda a tabela de dados para encontrar o elemento que satisfaz a condição pretendida. Assim, isto permite-nos acelerar a realização da condição V.numCamara = 10 e V.numCamara = I.numCamara .

vigia_idx – Criámos um índice sobre a tabela vigia para os atributos moradaLocal e numCamara com chave a composta (moradaLocal,numCamara) com o objetivo de tornar as condições V.numCamara = I.numCamara e I.moradaLocal = "Loures" mais eficientes. Dado que na tabela vigia pretendemos analisar tanto o numCamera como a moradaLocal criar um índice composto com estes dois atributos permite-nos apenas ter de aceder ao índice e não à tabela de dados, fazendo Index Only Scan o que torna mais eficientes as comparações no operador WHERE.

```
CREATE INDEX video_idx ON video USING HASH (numCamara);

CREATE INDEX vigia_idx ON vigia(moradaLocal,numCamara);
```

QUERY 2

transporta_idx – Foi criado um índice do tipo hash sobre a tabela transporta para o atributo numProcessoSocorro. Tal como referido anteriormente, um índice com uma a estrutura do tipo hash é o mais eficiente para testes de igualdade, ou seja, isto permite-nos acelerar a comparação T.numProcessoSocorro = E.numProcessoSocorro

eventoEmergencia_idx –Escolhemos criar um índice sobre a tabela eventoEmergencia para os atributos numTelefone, instanteChamada e numProcessoSocorro com chave a composta (numTelefone,instanteChamada,numProcessoSocorro). Dado que estamos a listar o número de vítima transportadas por cada cada evento de emergência ter um índice para o numTelefone e para o instanteChamada torna a operação de group by mais eficiente. Para além disso, adicionamos também o numProcessoSocorro à chave composta com o

numTelefone e instanteChamada, pois assim, temos o atributo numProcessoSocorro logo no índice e não precisamos de aceder à tabela de dados, ou seja, permite-nos fazer Index Only Scan o que poupa os custos de aceder à tabela eventoEmergencia e torna mais eficientes as comparações no operador WHERE.

```
CREATE INDEX transporta_idx ON transporta USING HASH (numProcessoSocorro);

CREATE INDEX eventoEmergencia_idx ON eventoEmergencia
(numTelefone,instanteChamada,numProcessoSocorro);
```

Modelo Multidimensional

```
DROP TABLE IF EXISTS eventos_meios_facts;
DROP TABLE IF EXISTS d_evento;
DROP TABLE IF EXISTS d_meio;
DROP TABLE IF EXISTS d_tempo;

CREATE TABLE d_tempo(idTempo SERIAL, dia NUMERIC(2) NOT NULL, mes NUMERIC(2) NOT
NULL, ano NUMERIC(4) NOT NULL, CONSTRAINT pk_d_tempo PRIMARY KEY(idTempo));

CREATE TABLE d_meio(idMeio SERIAL, numMeio NUMERIC(4) NOT NULL, nomeMeio
VARCHAR(50) NOT NULL, nomeEntidade VARCHAR(50)  NOT NULL, tipo VARCHAR(7),
CONSTRAINT pk_d_meio PRIMARY KEY(idMeio));

CREATE TABLE d_evento(idEvento SERIAL, numTelefone NUMERIC(9) NOT NULL,
instanteChamada TIMESTAMP NOT NULL,CONSTRAINT pk_d_evento PRIMARY KEY(idEvento));

CREATE TABLE eventos_meios_facts(idFact SERIAL, idTempo INTEGER NOT NULL, idMeio
INTEGER NOT NULL, idEvento INTEGER NOT NULL, CONSTRAINT pk_eventos_meios_facts
PRIMARY KEY(idFacts), CONSTRAINT fk_facts_tempo FOREIGN KEY (idTempo) REFERENCES
d_tempo(idTempo) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT fk_facts_meio
FOREIGN KEY (idMeio) REFERENCES d_meio(idMeio) ON DELETE CASCADE ON UPDATE
CASCADE, CONSTRAINT fk_facts_evento FOREIGN KEY (idEvento) REFERENCES
d_evento(idEvento) ON DELETE CASCADE ON UPDATE CASCADE);

DROP FUNCTION IF EXISTS insert_d_tempo();

CREATE OR REPLACE FUNCTION insert_d_tempo() RETURNS VOID AS $$
        DECLARE tempo DATE;
        BEGIN
                tempo = date '2018-01-01';
                WHILE  (tempo < '2019-01-01') LOOP
                        INSERT INTO d_tempo (dia, mes, ano) VALUES (
                                date_part('day',tempo),
                                date_part('month',tempo),
                                date_part('year',tempo));
```

```
                tempo = tempo + INTERVAL '1 DAY';
            END LOOP;
        END
        $$ LANGUAGE plpgsql;

SELECT insert_d_tempo();

INSERT INTO d_evento(numTelefone,instanteChamada)
SELECT numTelefone, instanteChamada FROM eventoEmergencia;

INSERT INTO d_meio(numMeio, nomeMeio, nomeEntidade, tipo)
SELECT numMeio, nomeMeio, nomeEntidade, 'Apoio' FROM meioApoio NATURAL JOIN meio;

INSERT INTO d_meio(numMeio, nomeMeio, nomeEntidade, tipo)
SELECT numMeio, nomeMeio, nomeEntidade, 'Socorro'
FROM meioSocorro NATURAL JOIN meio;

INSERT INTO d_meio(numMeio, nomeMeio, nomeEntidade, tipo)
SELECT numMeio, nomeMeio, nomeEntidade, 'Combate'
FROM meioCombate NATURAL JOIN meio;

INSERT INTO d_meio(numMeio, nomeMeio, nomeEntidade, tipo)
SELECT numMeio, nomeMeio, nomeEntidade, NULL
FROM meio WHERE (numMeio, nomeEntidade) NOT IN
(SELECT * FROM meioApoio UNION SELECT * FROM meioCombate
UNION SELECT * FROM meioSocorro);

INSERT INTO eventos_meios_facts(idTempo,idMeio,idEvento)
SELECT idTempo, idMeio, idEvento
FROM (SELECT idMeio,idEvento,
            date_part('day', instanteChamada) AS dia,
            date_part('month', instanteChamada) AS mes,
            date_part('year', instanteChamada) AS ano
        FROM acciona NATURAL JOIN processoSocorro NATURAL JOIN eventoEmergencia
        NATURAL JOIN d_meio NATURAL JOIN d_evento) as foo NATURAL JOIN d_tempo;
```

Data Analytics

```
SELECT tipo,ano,mes, COUNT(idMeio)
FROM eventos_meios_facts NATURAL JOIN d_meio NATURAL JOIN d_tempo
WHERE idEvento = 15
GROUP BY ROLLUP(tipo, ano, mes);
```