Projeto de Base de Dados

Parte 4

Turno BD2251795L10, Qua 09:30-11:00, LAB13

Docente Taras Lykhenko

Grupo 76

Nº	Nome	Carga / horas
79758	Ana Luísa Santo	12
82374	Luana Marques	12
86477	Maria Lopes	12

I. RESTRICOES DE INTEGRIDADE

a) RI-3: "Um Coordenador só pode solicitar vídeos de câmaras colocadas num local cujo accionamento de meios esteja a ser (ou tenha sido) auditado por ele próprio"

Para cumprir esta restrição de integridade optamos por implementar um trigger, acionado antes de ser inserido uma nova solicitação na tabela **solicita**. Caso esta nova solicitação não tenha um **número do processo** socorro que não exista na tabela "**acciona**" e na tabela "**audita**" para o mesmo coordenador, é chamada uma função de erro que interrompe e impede a inserção na tabela.

```
-- R3: Um Coordenador só pode solicitar vídeos de câmaras colocadas num local cujo accionamento de meio
CREATE OR REPLACE FUNCTION coordenador_nao_pode_solicitar() RETURNS TRIGGER AS
$$
    BEGIN
        IF NOT (EXISTS (SELECT numcamara FROM solicita WHERE numcamara IN(
                        SELECT numcamara FROM vigia NATURAL JOIN eventoemergencia NATURAL JOIN acciona
                        WHERE numprocessosocorro IN(
                            SELECT numprocessosocorro
                            FROM solicita s, audita a WHERE s.idcoordenador = a.idcoordenador))))
        THEN RAISE 'O coordenador nao accionou o evento correspondente a camara solicitada'
            USING HINT = 'ver lista de coordenador';
        END IF;
    RETURN NULL;
    END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS update_solicita_coordenador ON solicita;
CREATE TRIGGER update_solicita_coordenador
BEFORE INSERT OR UPDATE ON solicita
FOR EACH ROW
EXECUTE PROCEDURE coordenador_nao_pode_solicitar();
```

a) RI-4: "Um Meio de Apoio só pode ser alocado a Processos de Socorro para os quais tenha sido acionado"

Para cumprir esta restrição de integridade optamos por implementar um trigger, acionado antes de ser inserido uma nova alocação na tabela **alocado**. Caso este novo processo tenha um número do processo socorro correspondente ao mesmo num de processo da tabela "**acciona**" e com os mesmos atributos das restantes tabelas – **meioapoio** e **processo socorro** - é chamada uma função de erro que interrompe e impede a inserção do processo de socorro na tabela.

```
-- R4:Um Meio de Apoio só pode ser alocado a Processos de Socorro para os quais tenha sido accionado
CREATE OR REPLACE FUNCTION apoio_nao_pode_alocar() RETURNS TRIGGER AS
$$
    BEGIN
        IF NOT (EXISTS (
            SELECT * from meioapoio NATURAL JOIN acciona NATURAL JOIN processosocorro
            WHERE numprocessosocorro IN(
                SELECT numprocessosocorro FROM alocado)))
        THEN RAISE 'O meio de apoio nao pode alocar processos de socorro nao selecionados por eles'
            USING HINT = 'ver lista acciona';
        END IF;
    RETURN NULL;
    END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS update_aloca_processo ON alocado;
CREATE TRIGGER update_aloca_processo
BEFORE INSERT OR UPDATE ON alocado
FOR EACH ROW
EXECUTE PROCEDURE apoio_nao_pode_alocar();
```

II. ÍNDICES

Considerações gerais: o PostgreSQL cria por definição índices com as chaves primárias e estrangeiras de cada tabela, e utiliza por definição índices B+Tree.

1. **a) Índice em numCamara:** indice hash-based pois é o mais indicado para uma seleção por igualdade. Este é primário para ambas tabelas *vigia* e *video* pois *numCamara* é a chave primária. É um indice denso pois há pelo menos uma entrada de dados do indice por cada valor da chave de pesquisa. Como não é necessário estar agrupado, pois é indexado por hash, é desagrupado.

   Indice composto em <dataHoraInicio, dataHoraFim>: Outra alternativa, poderia ser usar um índice B+ Tree dinâmica, desagrupado e denso com chave de pesquisa composta <**dataHoraInicio**, **dataHoraFim**> na tabela vídeo. Este tipo de índice permitiria uma pesquisa mais eficiente uma vez que as entradas estariam organizadas pelo valor da chave de pesquisa.

   b) Utilizando o comando "EXPLAIN ANALYZE" obtemos no caso:
   1. Índices default das tabelas vídeo e vigia:

```
                                                                QUERY PLAN

-------------------------------------------------------------------------------------------------------------
-------------------------------------
 Nested Loop  (cost=0.30..16.35 rows=1 width=16) (actual time=0.010..0.010 rows=0 loops=1)
   ->  Index Scan using video_numcamara_key on video v  (cost=0.15..8.17 rows=1 width=28) (actu
al time=0.005..0.005 rows=0 loops=1)
         Index Cond: (numcamara = 10::numeric)
   ->  Index Scan using vigia_numcamara_key on vigia i  (cost=0.14..8.16 rows=1 width=12) (neve
r executed)
         Index Cond: (numcamara = 10::numeric)
         Filter: ((moradalocal)::text = 'address10'::text)
 Planning time: 5.735 ms
 Execution time: 0.096 ms
(8 rows)
```

   Observamos que os índices "vídeo(numCamara)" e "vigia(numCamara)" foram criados por definição. O tempo de execução foi de 0,096 ms.
   2. Índice B+Tree na tabela vídeo:
   CREATE INDEX vídeo_idx ON vídeo (dataHoraInicio, dataHoraFim);

```
                                                          QUERY PLAN

-------------------------------------------------------------------------------------------------------------
-------------------------------------
 Nested Loop  (cost=0.30..16.35 rows=1 width=16) (actual time=0.006..0.006 rows=0 loops=1)
   ->  Index Scan using video_numcamara_key on video v  (cost=0.15..8.17 rows=1 width=28) (actu
al time=0.004..0.004 rows=0 loops=1)
         Index Cond: (numcamara = 10::numeric)
   ->  Index Scan using vigia_numcamara_key on vigia i  (cost=0.14..8.16 rows=1 width=12) (neve
r executed)
         Index Cond: (numcamara = 10::numeric)
         Filter: ((moradalocal)::text = 'address10'::text)
 Planning time: 0.348 ms
 Execution time: 0.045 ms
(8 rows)
```

   Neste caso é utilizada a mesma estratégia antes de criar o índice, e obtemos um tempo inferior, diminuindo para 0,045 ms.
   3. Índice Hash na tabela vídeo:

CREATE INDEX vídeo_hash_idx ON vídeo USING HASH(numCamara);

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------------------
Nested Loop  (cost=0.14..16.19 rows=1 width=16) (actual time=0.006..0.006 rows=0 loops=1)
  ->  Index Scan using video_hash_idx on video v  (cost=0.00..8.02 rows=1 width=28) (actual time=0.005..0.005 rows=0 loops=1)
        Index Cond: (numcamara = 10::numeric)
  ->  Index Scan using vigia_numcamara_key on vigia i  (cost=0.14..8.16 rows=1 width=12) (never executed)
        Index Cond: (numcamara = 10::numeric)
        Filter: ((moradalocal)::text = 'address10'::text)
Planning time: 0.151 ms
Execution time: 0.023 ms
(8 rows)
```

É utilizada a mesma estratégia que antes da criação do índice, mas o índice vídeo_hash_idx é usado para fazer o primeiro index scan, e o tempo de execução diminuiu para 0,023 ms.

2. a) Indice em <numProcessoSocorro>: índice hash-based pois é o mais indicado para uma seleção por igualdade. Este é primário para a tabela transporta pois numProcessoSocorro é a chave primária e é secundária para a tabela EventoEmergencia. É um índice denso pois há pelo menos uma entrada de dados do índice por cada valor da chave de pesquisa. Como não é necessário estar agrupado, pois é indexado por hash, é desagrupado.

Indice em <numTelefone, **instanteChamada>:** Índice B+ Tree com chave de pesquisa composta <numTelefone, instanteChamada> na tabela eventoEmergencia. É um índice denso e desagrupado Este tipo de índice permitiria uma ordenação dos eventos de emergência por número de telefone e respetivos instantes de chamada através da estrutura *btree*, o que tornaria o *group by* da query em análise mais eficiente.

b) Utilizando o comando "EXPLAIN ANALYZE" obtemos para o caso:

1. Índices default:

```
                                                  QUERY PLAN
----------------------------------------------------------------------------------------------------------------
HashAggregate  (cost=25.51..26.76 rows=100 width=72) (actual time=0.005..0.005 rows=0 loops=1)
  Group Key: e.numtelefone, e.instantechamada
  ->  Hash Join  (cost=12.25..24.76 rows=100 width=72) (actual time=0.003..0.003 rows=0 loops=1)
        Hash Cond: (t.numprocessosocorro = e.numprocessosocorro)
        ->  Seq Scan on transporta t  (cost=0.00..11.10 rows=110 width=92) (actual time=0.002..0.002 rows=0 loops=1)
        ->  Hash  (cost=11.00..11.00 rows=100 width=72) (never executed)
              ->  Seq Scan on eventoemergencia e  (cost=0.00..11.00 rows=100 width=72) (never executed)
Planning time: 2.322 ms
Execution time: 0.114 ms
(9 rows)
```

Podemos observar que um índice com a chave de pesquisa <numTelefone, instanteChamada> já foi definido por definição, através da tabela eventoEmergencia. O tempo de execução obtido foi de 0,114 ms.

2. Índice B+Tree na tabela eventoEmergencia:

CREATE INDEX eventoEmergencia_idx ON eventoEmergencia(numTelefone, instanteChamada);

```
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------
----------------------
 HashAggregate  (cost=25.51..26.76 rows=100 width=72) (actual time=0.004..0.004 rows=0 loops=1)
   Group Key: e.numtelefone, e.instantechamada
   ->  Hash Join  (cost=12.25..24.76 rows=100 width=72) (actual time=0.002..0.002 rows=0 loops=
1)
         Hash Cond: (t.numprocessosocorro = e.numprocessosocorro)
         ->  Seq Scan on transporta t  (cost=0.00..11.10 rows=110 width=92) (actual time=0.002.
.0.002 rows=0 loops=1)
         ->  Hash  (cost=11.00..11.00 rows=100 width=72) (never executed)
               ->  Seq Scan on eventoemergencia e  (cost=0.00..11.00 rows=100 width=72) (never
executed)
 Planning time: 0.320 ms
 Execution time: 0.061 ms
(9 rows)
```

observámos que o tempo de execução diminui, passando a ser 0,061 ms, mas não é utilizado o índice criado, eventoEmergencia_idx.

I. Data Warehouse

Esquema em estrela

A tabela **fact** contém informação de cada evento, data do evento e meio utilizado. A tabela **d_tempo** caracteriza cada dia, mês e ano. A tabela **d_evento** contém detalhes de um evento, tal como o número de telefone e instante de chamada e a tabela **d_meio** contém informação dos meios, tais como o nome da entidade, nome e tipo de meio.

Tabela d_tempo:

Esta tabela é populada com o dia, mês e ano do **instanteChamada** da tabela ***eventoEmergencia***. Para darmos split do timestamp em mês, dia e ano, usamos uma função predefinida *date_part* do postgre.

```
INSERT INTO d_tempo(dia,mes, ano)
  SELECT
    date_part('day', TIMESTAMP instanteChamada) AS dia,
    date_part('month', TIMESTAMP instanteChamada) AS mes,
    date_part('year', TIMESTAMP instanteChamada) AS ano
  FROM eventoEmergencia;
```

Tabela d_evento:

Esta tabela é populada com a **numTelefone** e **instanteChamada** da tabela ***eventoEmergencia*** atraves do seguinte código postgre:

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

Tabela d_meio:

Esta tabela é populada com com a informação proveniente da tabela ***meio*** em conjugação com as seguintes 3 tabelas: ***meioCombate***, ***meioApoio*** *e* ***meioSocorro.*** Por cada uma das tabelas de tipos de meios, vamos buscar informação relativa ao numMeio, nomeMeio e nomEntidade.

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

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

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

Tabela fact:

Esta tabela é populada com os identificadores de cada tabela que a mesma relaciona, de modo a representar um *summary navegation* das tabelas.

```
INSERT INTO fact (idTempo, idEvento, idMeio)
  SELECT DISTINCT idTempo, idEvento, idMeio
  FROM d_tempo NATURAL JOIN d_evento NATURAL JOIN d_meio;
```

II. Consulta OLAP

Para o evento cujo idEvento é 15, contam-se as ocorrencias de todos os tipos de meios, agrupados por **mes** e **ano**, obtendo-se os subtotais por mes e por ano com o auxilio da instrução de ROLLUP;

Faz-se UNION com a tabela que se obtem a partir da projeção do evento com idEvento 15, em que as ocorrencias de cada tipos de meios são agrupados por **ano**;

Por fim, projeta-se **todos os subtotais** por cada tipo de meio, que representam o número de meios de cada tipo utilizados no evento número 15, por **ano** e **mês**.

```
  SELECT ano, mes, tipo, count(*)
  FROM fact NATURAL JOIN d_meio NATURAL JOIN d_tempo
  WHERE idEvento = 15
  GROUP BY ano, mes, tipo WITH ROLLUP
UNION
  SELECT ano, NULL, tipo, count(*)
  FROM fact NATURAL JOIN d_meio NATURAL JOIN d_tempo
  WHERE idEvento = 15
  GROUP BY ano, tipo
UNION
  SELECT NULL, NULL, tipo, count(*)
  FROM fact NATURAL JOIN d_meio NATURAL JOIN d_tempo
  WHERE idEvento = 15
  GROUP BY tipo
  ORDER BY ano, mes ASC;
```