Triggers e Regras

Fernando Lobo

Base de Dados, Universidade do Algarve

Triggers

- Um trigger permite que uma determinada sequência de comandos SQL seja accionada quando um determinado evento ocorre.
- O evento pode ser INSERT, UPDATE, ou DELETE.
- O trigger pode ser accionado imediatamente antes (BEFORE) ou imediatamente depois (AFTER) de cada evento.

Triggers

- A sequência de comandos (procedimento) pode ser programada numa linguagem qualquer desde que devidamente instalada no servidor do SGBD.
- No exemplo que se segue, vamos utilizar PL/pgSQL, uma linguagem procedimental que vem instalada no PostgreSQL e que é muito semelhante à linguagem PL/SQL da Oracle.

Exemplo 1

Ao inserir um filme pretende-se que a duração seja truncada para 300 minutos no caso de se tentar inserir um valor superior.

- Associamos um trigger à tabela de filmes.
- O trigger será accionado imediatamente antes de um INSERT ou UPDATE à tabela de filmes.
- Ao ser disparado, o trigger irá executar um procedimento chamado `td300()`

```
CREATE TRIGGER trunca_duracao
    BEFORE INSERT OR UPDATE
    ON Filmes
    FOR EACH ROW
        EXECUTE PROCEDURE td300();
```

Exemplo 1 (cont.)

- O procedimento verifica se a duração do(s) filme(s) a serem inseridos ou modificados é maior do que 300. Em caso afirmativo força a duração a ser 300.

```
CREATE OR REPLACE FUNCTION td300() RETURNS TRIGGER AS $$
  BEGIN
     IF NEW.duracao > 300 THEN
        NEW.duracao := 300;
     END IF;
     RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```


NEW e OLD

- Quando o evento é um INSERT, UPDATE ou DELETE, podemos fazer referência a duas pseudo-tabelas, NEW e OLD.
- NEW refere-se aos novos tuplos (no caso de INSERTs e UPDATEs).
- OLD refere-se aos velhos tuplos (no caso de DELETEs e UPDATEs).

AFTER e BEFORE

- O trigger pode ser accionado imediatamente após (AFTER)
- ou imediatamente antes (BEFORE) do evento ocorrer.

Exemplo 2

Ao inserir um filme na tabela Filmes, se o estúdio não existir, então automaticamente inserir esse estúdio na tabela Estúdios (com morada a NULL).

```
CREATE TRIGGER insere_estudio
   BEFORE INSERT
   ON Filmes
   FOR EACH ROW
      EXECUTE PROCEDURE insere_estudio_de_filme();
```

Exemplo 2 (cont.)

```
CREATE OR REPLACE FUNCTION insere_estudio_de_filme()
 RETURNS TRIGGER AS $$
   BEGIN
     IF NOT EXISTS (SELECT * FROM Estudios
                    WHERE nome = NEW.nomeEstudio)
     THEN
       INSERT INTO Estudios(nome) VALUES (NEW.nomeEstudio);
     END IF;
     RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```


Exemplo 3

Cada vez que é feito um DELETE ou um UPDATE à tabela de Filmes, guardar um registo numa tabela à parte sobre quem fez a alteração, em que data, e que tipo de operação foi efectuada.

```
CREATE TABLE arquivoFilmes(
    nome           VARCHAR(50),
    ano            INTEGER,
    duracao        INTEGER,
    aCores         BOOLEAN,
    nomeEstudio    VARCHAR(30),
    nomeRealizador VARCHAR(50),
    userchanged    VARCHAR,
    datechanged    DATE,
    operation      VARCHAR
);
```

Exemplo 3 (cont.)

```
CREATE TRIGGER arquiva_filme
  AFTER DELETE OR UPDATE
  ON Filmes
  FOR EACH ROW
    EXECUTE PROCEDURE arquiva_filme();
```


Exemplo 3 (cont.)

```
CREATE OR REPLACE FUNCTION arquiva_filme()
 RETURNS TRIGGER AS $$
  BEGIN
     INSERT INTO arquivoFilmes VALUES
     (OLD.nome, OLD.ano, OLD.duracao, OLD.aCores,
      OLD.nomeEstudio, OLD.nomeRealizador,
       CURRENT_USER, now(),TG_OP);
     RETURN NULL;
  END;
$$ LANGUAGE plpgsql;
```

Regras (Rules)

- As regras não fazem parte do SQL standard, mas são implementadas pelo PostgreSQL e dão muito jeito.
- Permitem especificar uma acção alternativa que é executada em SELECTs, INSERTs, UPDATESs, ou DELETEs.
- Tal como nos triggers, podemos fazer uso das pseudo-tabelas NEW e OLD.

Exemplo 1

- Exemplo: a seguinte regra não permite apagar filmes do estúdio Disney.

```
CREATE RULE nao_apaga_filmes_disney AS
     ON DELETE TO Filmes
     WHERE OLD.nomeEstudio = 'Disney'
     DO INSTEAD NOTHING;
```

Regras e Vistas

- Em PostgreSQL, as views são implementadas com regras.
- Em PostgreSQL, as views são “read-only”, mas podemos simular a “escrita” em views utilizando regras.
- Exemplo: criar uma regra de modo a podermos inserir tuplos na view `filmes_disney`.

```
CREATE VIEW filmes_disney (nome,ano,realizador) AS
    SELECT nome, ano, nomeRealizador
      FROM Filmes
     WHERE nomeEstudio = 'Disney';
```


Exemplo (cont.)

```
CREATE RULE insere_filmes_disney AS
   ON INSERT TO filmes_disney
   DO INSTEAD
      INSERT INTO Filmes (nome, ano,
                          nomeRealizador, nomeEstudio)
      VALUES (NEW.nome, NEW.ano, NEW.realizador, 'Disney');
```

Agora já podemos inserir dados na view.

```
INSERT INTO filmes_disney
VALUES ('The Huntchback of Notre Dame',
        1995, 'Steven Spielberg');
```

Outro exemplo

- Inserir dados na view `filmes_actores`

```
CREATE VIEW filmes_actores (filme, ano, estudio,
                            realizador, actor) AS
     SELECT F.nome, F.ano, F.nomeEstudio,
            F.nomeRealizador, P.nomeActor
       FROM Filmes AS F, Participa AS P
      WHERE F.nome = P.nomeFilme
        AND F.ano = P.anoFilme;
```

- A inserção de um tuplo na view, pode fazer com que tenhamos de inserir tuplos na tabela `Filmes`, `Estudios`, `Realizadores`, `Actores` e `Participa`.

Outro exemplo (cont.)

```
CREATE RULE insere_em_filmes_actores AS
   ON INSERT TO filmes_actores
   DO INSTEAD
      (INSERT INTO Filmes(nome, ano,
                          nomeEstudio, nomeRealizador)
            VALUES (NEW.filme, NEW.ano,
                    NEW.estudio, NEW.realizador);

       INSERT INTO Actores(nome) VALUES (NEW.actor);

       INSERT INTO Estudios(nome) VALUES (NEW.estudio);

       INSERT INTO Realizadores(nome) VALUES (NEW.realizador);

       INSERT INTO Participa(nomeFilme,anoFilme,nomeActor)
            VALUES (NEW.filme, NEW.ano, NEW.actor);
      );
```

Outro exemplo (cont.)

Agora podemos inserir dados na view `filmes_actores`.

```
INSERT INTO filmes_actores
     VALUES ('Taxi Driver', 1978, 'Universal',
             'Martin Scorcese', 'Roxana Arquette');
```


- Os INSERTs podem dar erro porque podem violar restrições que hajam nas tabelas. Por exemplo, se o estúdio `Universal` já existir na tabela de estúdios, o INSERT falha porque o nome do estúdio é chave primária.
- A solução para este problema é criar regras adicionais. Por exemplo:

```
CREATE RULE nao_insere_estudios_duplicados AS
  ON INSERT TO Estudios
  WHERE( EXISTS
           (SELECT * FROM Estudios
            WHERE nome=NEW.nome
           )
       )
  DO INSTEAD NOTHING;
```

- A inserção de dados na view “esconde” a inserção efectiva dos dados nas tabela base.
- Ao inserir dados na view ficamos impossibilitados de inserir todos os atributos das tabelas base.
- Esses atributos ficam com o valor NULL (ou com o valor que estiver especificado como DEFAULT).
- Podem consultar mais coisas na documentação online do PostgreSQL (Programmer’s Guide → Server Programming → The Rule System)

  `http://www.postgresql.org/docs/10/static/rules.html`