Complements d'SQL per a la codificació de procediments emmagatzemats en PostgreSQL

PID_00253050

Universitat Oberta de Catalunya

Presentació

Aquest document és un complement, en què us presentem una sèrie de conceptes útils sobre la programació de procediments en PostgreSQL.

Propòsit del document

La intenció d'aquest document és:

- Proporcionar un conjunt de conceptes de programació de procediments emmagatzemats en PostgreSQL no capturats en el mòdul didàctic.
- Ajudar al desenvolupament de procediments emmagatzemats en PostgreSQL per a la generació de processos de càrrega.

Conceptes

En les seccions successives, veurem els conceptes següents:

- Blocs anònims
- Transaccions en procediments emmagatzemats
- Blocs BEGIN – END

Tots els conceptes vistos en aquest document estan basats en la versió de PostgreSQL 9.3. És possible que, en versions futures, sigui necessari revisar els manuals tècnics proporcionats per PostgreSQL per a la versió desitjada.

Blocs anònims

Des de la seva versió 9.0, PostgreSQL ha inclòs la possibilitat d'executar blocs de codi de forma anònima, és a dir, sense necessitat de definir aquest bloc de codi com una funció. Aquests blocs anònims es compilen i executen una única vegada, sense arribar a guardar-se en el sistema gestor de bases de dades (SGBD), tal com es realitza amb els procediments emmagatzemats (funcions en PostgreSQL).

La manera de codificar aquests blocs anònims en PostgreSQL és mitjançant la sentencia `DO`, que té les sintaxis següents:

```
DO [ LANGUAGE lang_name ] code

DO code [ LANGUAGE lang_name ]
```

Així, `lang_name` fa referència al nom del llenguatge procedimental utilitzat (per defecte, `plpgsql`) i al codi a executar codificat en el llenguatge procedimental utilitzat. La clàusula `LANGUAGE`, com es veu en el requadre superior, es podria indicar tan abans com després del codi. El bloc de codi `code` és tractat com el cos d'una funció en PostgreSQL, amb les diferències següents: els blocs anònims no permeten paràmetres i aquests sempre retornen un valor `void`.

Per a aclarir aquest concepte, vegem un exemple de bloc anònim: en aquest cas, es realitza una iteració sobre una taula temporal de préstecs de llibres que tenen la paraula 'social' en el nom del llibre. Cadascun dels préstecs que es troben a la taula amb aquesta condició, es mostra per pantalla.

```
DO $$

DECLARE
  -- Variables per al seu ús com a part del LOOP
  v_id_prestec prestec_tmp.id_prestec%type;
  v_nom_llibre prestec_tmp.nom_llibre%type;
  v_autor_llibre prestec.autor_llibre%type;

BEGIN -- Inici del procediment

-- Iterem per cada registre de la taula temporal de prestecs
FOR v_id_prestec, v_nom_llibre, v_autor_llibre IN
  SELECT id_prestec, nom_llibre, autor_llibre
    FROM prestec_tmp
   WHERE nom_llibre LIKE '%social%'
  LOOP -- Inici del LOOP.
       -- Mostrem el registre processat
      RAISE NOTICE 'Prestec: %, Llibre: %, Autor: %',
                   v_id_prestec, v_nom_llibre, v_autor_llibre;
  END LOOP; -- Fi del LOOP

END; -- Fi del procediment
$$language plpgsql;
```

Els blocs anònims ens proporcionen dos beneficis principals:

1. Ens permeten generar i executar el codi `plpgsql` sense haver de compilar-lo i guardar-lo en el servidor, ja que aquest es compila i executa en el mateix moment.

2. Ens permet gestionar errors dins d'un bloc SQL sense que sigui necessari que la gestió de l'error es realitzi en el nivell superior.

Vegem un exemple d'aquest últim punt.

Exemple de gestió de l'error en blocs anònims

En el codi que presentem a continuació, similar al que hem vist anteriorment, hem canviat la condició de la consulta que obté les dades de préstecs perquè falli (en aquest cas, produirà un error pel tipus de dades utilitzades, ja que s'espera un tipus de dades caràcter). També s'ha afegit un bloc de codi per a capturar l'excepció, com es pot veure en el codi en negreta. En cas d'un error, l'excepció es captura i es mostra el corresponent missatge d'error.

```
DO $$

DECLARE
  -- Variables per al seu us com a part del LOOP
  v_id_prestec prestec_tmp.id_prestec%type;
  v_nom_llibre prestec_tmp.nom_llibre%type;
  v_autor_llibre prestec.autor_llibre%type;

BEGIN  -- Inici del procediment

-- Iterem per cada registre de la taula temporal de prestecs
FOR v_id_prestec, v_nom_llibre, v_autor_llibre IN
  SELECT id_prestec, nom_llibre, autor_llibre
    FROM prestec_tmp
   WHERE nom_llibre LIKE 1
  LOOP -- Inici del LOOP.
       -- Mostrem el registre processat
      RAISE NOTICE 'Prestec: %, Llibre: %, Autor: %',
                   v_id_prestec, v_nom_llibre, v_autor_llibre;
  END LOOP; -- Fi del LOOP

  EXCEPTION WHEN OTHERS THEN
  BEGIN
    RAISE NOTICE 'S'ha produit un error greu!!!';
  END;

END; -- Fi del procediment
$$language plpgsql;
```

Aquest tipus de gestió de l'error no es pot realitzar en blocs SQL de l'estil com el que us mostrem a continuació, ja que PostgreSQL avorta l'execució del bloc en cas d'error (desfent els canvis) i no continua amb l'execució de la resta de sentències, deixant la transacció oberta, transacció que és necessari finalitzar (a través d'un `COMMIT` o un `ROLLBACK`).

```
START TRANSACTION;

-- Elimina les files de la taula temporal de prestecs
DELETE FROM prestec_tmp;

-- Elimina les files de la taula de préstecs desti
DELETE FROM prestec_fact;

-- Insereix dades en la taula temporal de prestecs
SELECT sp_insereix_prestecs_tmp();

-- Insereix les dades de la taula de prestecs desti
SELECT sp_insereix_prestecs();

COMMIT;
```

Per a poder gestionar l'error en SQL o `plpgsql`, s'ha d'implementar sia un bloc anònim sia un procediment emmagatzemat.

Transaccions en procediments emmagatzemats

En PostgreSQL, els procediments emmagatzemats no permeten iniciar o confirmar una transacció (ni tampoc desfer els canvis) de forma explícita. Aquest comportament difereix respecte d'altres SGBD comercials com ara Oracle, on sí que és possible.

PostgreSQL implementa el funcionament següent: assumeix que un procediment emmagatzemat s'executa sempre dins del context d'una transacció iniciada pel nivell superior. Aquest nivell superior podria ser una aplicació (que hagi iniciat la transacció) o bé podria ser un bloc SQL. PostgreSQL té la particularitat que tracta cada sentència SQL com si aquesta s'executés dins d'una transacció. Si volem executar un bloc de sentències SQL dins d'una transacció, hem d'executar aquest bloc dins d'un bloc `START TRANSACTION – COMMIT`.

Nota: en PostgreSQL, l'inici de les transaccions es podria especificar mitjançant les sentències `BEGIN`, `BEGIN TRANSACTION` o `START TRANSACTION`. Per a evitar confusions en el codi, especialment amb els blocs `BEGIN` i `END` en `plpgsql`, utilitzarem en aquest document `START TRANSACTION`.

Vegem una sèrie d'exemples per a explicar el que hem comentat.

Exemple d'error quan iniciem i confirmem/desfem transaccions de forma explícita en procediments emmagatzemats

En aquest primer exemple en codi `plpgsql`, es crea un procediment denominat `sp_insereix_prestecs`, que recorre a la taula de préstecs temporal obtenint aquells préstecs de llibres amb la paraula 'social' en el títol del llibre. Aquestes files s'inseriran en una taula `prestecs_fact`. Vegeu que dins del procediment s'ha iniciat una transacció, confirmant-se aquesta al final del bloc del codi que itera sobre la taula temporal de préstecs:

```
CREATE OR REPLACE FUNCTION sp_insereix_prestecs() RETURNS void AS $$

DECLARE
  -- Variables per al seu us com a part del LOOP
  v_id_prestec prestec_tmp.id_prestec%type;
  v_nom_llibre prestec_tmp.nom_llibre%type;
  v_autor_llibre prestec.autor_llibre%type;

BEGIN  -- Inici del procediment

START TRANSACTION;

-- Iterem per cada registre de la taula temporal de prestecs
FOR v_id_prestec, v_nom_llibre, v_autor_llibre IN
  SELECT id_prestec, nom_llibre, autor_llibre
    FROM prestec_tmp
   WHERE nom_llibre LIKE '%social%'
  LOOP -- Inici del LOOP.
       -- Inserim el registre processat en la taula de fets
      INSERT INTO prestec_fact
        VALUES (v_id_prestec, v_nom_llibre, v_autor_llibre);
  END LOOP; -- Fi del LOOP

COMMIT;

END; -- Fi del procediment
$$language plpgsql;
```

El procediment emmagatzemat es crearia correctament en PostgreSQL. En canvi, a l'hora d'executar-lo, s'obtindria l'error següent:

```
SELECT sp_insereix_prestecs();

ERROR: cannot begin/end transactions in PL/pgSQL
HINT: Use a BEGIN block with an EXCEPTION clause instead.
CONTEXT: PL/pgSQL function sp_insereix_prestecs() line 11 at SQL statement

********** Error **********

ERROR: cannot begin/end transactions in PL/pgSQL
SQL state: 0A000
Hint: Use a BEGIN block with an EXCEPTION clause instead.
Context: PL/pgSQL function sp_insereix_prestecs() line 11 at SQL statement
```

Igual que en els procediments emmagatzemats, en els blocs anònims tampoc no es permet la iniciació i confirmació de transaccions.

Exemple d'execució d'un procediment dins d'una transacció

Suposant el procediment emmagatzemat següent (el mateix codi mostrat en l'exemple anterior sense les sentències d'iniciació i confirmació de transaccions):

```
CREATE OR REPLACE FUNCTION sp_insereix_prestecs() RETURNS void AS $$

DECLARE
  -- Variables per al seu us com a part del LOOP
  v_id_prestec prestec_tmp.id_prestec%type;
  v_nom_llibre prestec_tmp.nom_llibre%type;
  v_autor_llibre prestec.autor_llibre%type;

BEGIN  -- Inici del procediment

-- Iterem per cada registre de la taula temporal de prestecs
FOR v_id_prestec, v_nom_llibre, v_autor_llibre IN
  SELECT id_prestec, nom_llibre, autor_llibre
    FROM prestec_tmp
   WHERE nom_llibre LIKE '%social%'
  LOOP -- Inici del LOOP.
       -- Inserim el registre processat en la taula de fets
      INSERT INTO prestec_fact
        VALUES (v_id_prestec, v_nom_llibre, v_autor_llibre);
  END LOOP; -- Fi del LOOP

END; -- Fi del procediment
$$language plpgsql;
```

L'execució del procediment d'aquesta forma es realitza dins del context d'una transacció:

```
SELECT sp_insereix_prestecs();
```

En el supòsit anterior, la transacció està implícita i seria el mateix que realitzar el següent:

```
START TRANSACTION;

SELECT sp_insereix_prestecs();

COMMIT;
```

El bloc següent també s'executaria com una única transacció.

```
START TRANSACTION;

-- Elimina les files de la taula temporal de prestecs
DELETE FROM prestec_tmp;

-- Elimina les files de la taula de prestecs desti
DELETE FROM prestec_fact;

-- Insereix dades en la taula temporal de prestecs
SELECT sp_insereix_prestecs_tmp();

-- Insereix les dades de la taula de prestecs destí
SELECT sp_insereix_prestecs();

COMMIT;
```

En qualsevol dels exemples proposats, és important recordar, com hem comentat anteriorment, que si es produeix un error dins del bloc `START TRANSACTION – COMMIT`, PostgreSQL desfarà els canvis realitzats fins al moment de l'error, però deixarà la transacció oberta fins que s'executi la sentència de tancament de la transacció (sia `COMMIT` o `ROLLACK`).

L'error que produiria PostgreSQL quan intentem executar sentències SQL després d'un error en un bloc `START TRANSACTION – COMMIT` sense arribar a tancar la transacció (com hem vist anteriorment) seria el següent:

```
ERROR:  current  transaction  is  aborted,  commands  ignored  until  end  of
transaction block

********** Error **********

ERROR:  current  transaction  is  aborted,  commands  ignored  until  end  of
transaction block
SQL state: 25P02
```

Blocs BEGIN – END

El codi `plpgsql` en un procediment emmagatzemat (o bloc anònim) es pot dividir en subblocs mitjançant l'ús de les clàusules `BEGIN` i `END` (no confondre ambdues clàusules `BEGIN` i `END` d'inici i fi de transaccions).

Amb l'ús d'aquests blocs, PostgreSQL ens permet generar un context dins d'un procediment, que ens permet gestionar l'error en aquest context com si es tractés d'una subtransacció. D'aquesta forma, en cas d'error, PostgreSQL desfaria els canvis fins a l'inici d'aquest subbloc, sense arribar a desfer els canvis de la transacció principal. Per defecte, PostgreSQL desfà tots els canvis de la transacció iniciada fins a l'inici del `BEGIN` del context on s'ha produït l'error.

Vegem un exemple del que hem explicat.

Exemple d'ús de blocs BEGIN – END

Suposem el bloc anònim següent:

```
DO $$
BEGIN

DROP TABLE IF EXISTS taulaA;
DROP TABLE IF EXISTS taulaB;

CREATE TABLE taulaA (id INTEGER PRIMARY KEY);
CREATE TABLE taulaB (id INTEGER PRIMARY KEY);

INSERT INTO taulaA VALUES (1);
INSERT INTO taulaA VALUES (2);

  -- Subbloc 1
  BEGIN
    DELETE FROM taulaB;
    INSERT INTO taulaB SELECT * FROM taulaA;
    INSERT INTO taulaB VALUES (3);
  END;

  -- Subbloc 2
  BEGIN
    DELETE FROM taulaB;
    INSERT INTO taulaB VALUES (4);
    INSERT INTO taulaB SELECT * FROM taulaA;
    INSERT INTO taulaB SELECT * FROM taulaA;
    EXCEPTION WHEN others THEN
      RAISE NOTICE 'Error, el valor 4 no es posa a la taulaB pero el valor 3';
      RAISE NOTICE 'es queda perque esta fora del context de error!';
  END;

END;
$$
```

Aquest codi s'encarrega d'eliminar (si existeixen) i crear dues taules (`taulaA` i `taulaB`), inserir dos valors a la `taulaA` i executar dos subblocs de codi:

- El primer subbloc esborra les dades de la `taulaB`, insereix a `taulaB` els valors que existeixen a `taulaA`, i després insereix a `taulaB` el valor 3. Al final d'aquest subbloc tindrem a `taulaB` els valors 1, 2 i 3.

- El segon subbloc esborra les dades de la `taulaB`, insereix a la `taulaB` el valor 4 i, a continuació, intenta inserir els valors de la `taulaA` a la `taulaB` dues vegades. Com que tenim una restricció de clau primària, es produeix un error que és capturat per l'excepció definida. Quan es produeix l'error, tots els canvis fins a l'inici del segon

subbloc es desfan, tornant a l'estat inicial quan comença aquest subbloc (`taulaB` amb valors 1, 2 i 3). L'excepció s'encarrega de mostrar el missatge d'error que es pot veure en el codi.

Veiem que el valor 4 no s'ha confirmat, perquè el canvi s'ha desfet fins a l'inici del segon subbloc a causa de l'error produït. El fet d'haver capturat l'excepció (i no propagar-la al nivell superior) ens ha permès acabar la transacció de forma correcta (el bloc executat es llança dins del context d'una transacció), de manera que al final del codi, la `taulaA` comptarà amb 2 valors (1 i 2) i la `taulaB` comptarà amb 3 valors (1, 2 i 3).

Enllaços a la documentació de PostgreSQL

A continuació, us presentem els enllaços a la documentació de PostgreSQL on podem trobar referències sobre el que hem comentat.

Blocs anònims:

http://www.postgresql.org/docs/9.3/static/sql-do.html

Transaccions en procediments emmagatzemats:

http://www.postgresql.org/docs/9.3/static/tutorial-transactions.html

http://www.postgresql.org/docs/9.3/static/plpgsql-structure.html

http://www.postgresql.org/docs/9.3/static/plpgsql-control-structures.html#PLPGSQL-ERROR-TRAPPING

Blocs BEGIN – END:

http://www.postgresql.org/docs/9.3/static/plpgsql-structure.html

http://www.postgresql.org/docs/9.3/static/plpgsql-control-structures.html#PLPGSQL-ERROR-TRAPPING

http://www.postgresql.org/docs/9.3/static/sql-begin.html

http://www.postgresql.org/docs/9.3/static/sql-end.html