Procedurele SQL met (plpg)sql

Wim.bertels@ucll.be

Objecten op de server

- Stored procedures
- Stored functions
- Triggers

- Def. : hoeveelheid code die opgeslagen is in de catalogus van een DB en die geactiveerd kan worden
- Vergelijkbaar met wat je in programmeren een (statische) methode zou noemen.

Voorbeeld: stored function

```
CREATE OR REPLACE FUNCTION increment(i INT)
        RETURNS INT
        AS
$$
    BEGIN
      RETURN i + 1;
    END;
$$ LANGUAGE 'plpgsql';

-- Een voorbeeld van hoe de functie op te roepen:
SELECT increment(10);
```

Verwerking

Verwerking :

- Vanuit programma wordt procedure/functie opgeroepen
- DBMS ontvangt oproep en zoekt procedure
- Procedure wordt uitgevoerd waarbij de instructies op de database verwerkt worden
- M.b.v. een code wordt aangegeven of procedure correct verwerkt is (sqlcode)

Parameters Algemeen

- Communicatie met buitenwereld
- 3 soorten (signatuur methode) :
  - Invoerparameters
  - Uitvoerparameters(niet bij functions)
  - Invoer/uitvoerparameters
- Parameter best andere naam dan kolom van tabel !
- Return (kan enkel bij functions)

Voorbeeld IN OUT

```
CREATE FUNCTION dup(in int, out f1 int, out f2 text)
    AS
$$
        SELECT $1, CAST($1 AS text) || ' is text'
$$
LANGUAGE SQL;

SELECT * FROM dup(42);
```

Signatuur: Functions vs Procedures

- Algemeen:
  - Het grote verschil is dat functies altijd iets teruggeven
  - Terwijl procedures nooit een return hebben
  - In de praktijk afhankelijk van het concrete DBMS dat je gebruikt

Waarom?

- Historisch verschil tussen berekeningen (functies) en manipulaties (procedures)
- Daarnaast:
  - Transacties kunnen enkel binnen procedures
  - Zo kan de planner hiermee rekening houden

Instructie mogelijkheden (de essentie)

- Declaratie
- Sequentie
- Selectie
  - IF .. THEN .. (ELSE ..) END IF
  - CASE .. WHEN .. THEN .. .. END CASE
- Iteratie
  - FOREACH .. LOOP .. END LOOP
  - FOR .. IN .. LOOP .. END LOOP

Structuur: functies

```
CREATE FUNCTION function_name(arg1,arg2,...)
    RETURNS type
    AS
‘
    BEGIN
        -- logic
    END;
‘
LANGUAGE language_name;
```

Structuur: functies

- Specifieer naam van functie
- Lijst van parameters achter naam van functie
- Definieer return type
- Gevolgd door code binnen begin- en end block
- Procedurele taal meegeven

Verwijderen stored functions

- DROP FUNCTION: verwijdert functie

- Vb.
  - DROP FUNCTION function_name;
  - DROP FUNCTION function_name(signatuur);

SELECT INTO voorbeeld

- Enkel indien een rij als uitvoer! Vb.

```
create function som_boetes_speler(p_spelersnr integer)
        returns decimal(8,2)
        AS
$eenderwat$
        declare     som_boetes decimal(8,2);
        begin
            select  sum(bedrag)
            into    som_boetes
            from    boetes
            where   spelersnr = p_spelersnr;
            return  som_boetes ;
        end;
$eenderwat$
language plpgsql;

select   som_boetes_speler (27) ;
```

$$ delimiter dialect

```
create function som_boetes_speler(p_spelersnr integer)
        returns decimal(8,2)
        AS
$eenderwat$
        declare som_boetes decimal(8,2);
        begin
            select   sum(bedrag)
            into     som_boetes
            from     boetes
            where    spelersnr = p_spelersnr
            return   som_boetes ;
        end;
$eenderwat$
language plpgsql;

select   som_boetes_speler (27) ;
```

PERFORM

- Resultaten van een sql statement moeten opgevangen worden, bv via INTO variabele.
- PERFORM alternatief voor SELECT waarbij het resultaat niet wordt opgevangen
    - Bv SELECT now(); zal een fout geven in een code blok
    - PERFORM now(); niet

- Vergelijkbaar met het void maken van een functie in andere programmeer talen

FOUND globale variabele

- Boolean

- Bijvoorbeeld:
  - PERFORM spelersnr FROM spelers ;
  - IF FOUND THEN .. END IF ;

RETURN(S)

- Signatuur :
  - RETURNS type
  - RETURNS setof type

- Code :
  - RETURN scalair..
  - RETURN QUERY ..
  - RETURN NEXT .. + RETURN
  - RETURN TABLE ..

Foutboodschappen

- Foutboodschappen :
- SQL-error-code : beschrijvende tekst
- SQLSTATE: code (getal)

```
BEGIN
	-- code
	RAISE DEBUG 'A debug message % ', variable_that_will_replace_percent;
EXCEPTION
	-- welke fout, bv
	WHEN unique_violation THEN
	-- code
	WHEN division_by_zero THEN
	RAISE .. -- eventueel omzetten naar unique_violation;
	WHEN others THEN
	-- ?
	NULL;
END
```

RAISE

- RAISE;
- RAISE division_by_zero;
- RAISE SQLSTATE '22012';
- RAISE DEBUG/INFO/..
  -- SET client_min_messages TO debug;
- RAISE .. USING
  ERRCODE = 'unique_violation',
  HINT = 'suggestie voor de reden voor de gebruiker',
  DETAIL = 'meer detail fout',
  MESSAGE = 'gedrag van de unique_violation';

ASSERT

- ASSERT condition [, message];

```
do
$$
declare
   film_count integer;
begin
   select   count(*)
   into     film_count
   from     film;
   assert   film_count > 0, 'No films found, check the film table';
end
$$;

-- do is dialect, https://www.postgresql.org/docs/current/sql-do.html
```

SECURITY

- GRANT EXECUTE ON haha() TO jomeke ;

- INVOKER : standaard, veilig
- DEFINER : via de rechten van de eigenaar

```
CREATE OR REPLACE FUNCTION haha()
    RETURNS text
    AS
$code$
    BEGIN
        DROP TABLE ola();
        RETURN ‘pola’;
    END
$code$
LANGUAGE plpgsql
EXTERNAL SECURITY DEFINER;
```

LEVEL of immutability

```
CREATE FUNCTION add(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;
```

- **IMMUTABLE** (alleen afhankelijk van de signatuur)
- STABLE (geen aanpassingen)
- VOLATILE (standaard)

Structuur: stored procedure

CREATE OR REPLACE PROCEDURE <procedure_name>(<arguments>)

AS <block-of-code>

LANGUAGE <implementation-language>;

Transactie voorbeeld: stored procedure

```
CREATE OR REPLACE PROCEDURE voorbeeld(invoer text )
        AS
$code$
        BEGIN
            ...
            IF invoer = ‘niet doen’    THEN RAISE WARNING 'Abort, the ship is sinking';
                ROLLBACK;
            ELSEIF invoer = ‘doen’ THEN RAISE INFO 'Gaon met die banaan';
                COMMIT;
            END IF;
            ...
        END
$code$
LANGUAGE plpgsql;

CALL voorbeeld(‘doen’);

-- https://www.postgresql.org/docs/16/plpgsql-transactions.html
```

Praktisch: script

BEGIN;

code en testen

ROLLBACK;

DROP [procedure|function|trigger] naam

CREATE [procedure|function|trigger] naam

ALTER [procedure|function|trigger] naam

CREATE OR REPLACE [procedure|function|trigger] naam

Triggers

- Def. :

  hoeveelheid code die opgeslagen is in de catalogus en die geactiveerd wordt door het dbms indien een bepaalde operatie wordt uitgevoerd en een conditie waar is.
- Triggers worden door het dbms zelf automatisch opgeroepen (niet door vb. een call)

PostgreSQL

- Triggers roepen

  trigger functies op
- CREATE FUNCTION **trigger_functie**...

  RETURNS TRIGGER

  ..
- CREATE TRIGGER trigger_trg ..

  ..

  EXECUTE PROCEDURE **trigger_functie**..

Voorbeeld: tabel

```
create table mutaties (
    gebruiker              varchar(30)   not null,
    mut_tijdstip           timestamp     not null,
    mut_spelersnr          smallint      not null,
    mut_type               char(1)       not null,
    mut_spelersnr_new      smallint                ,
    primary key
        (gebruiker, mut_tijdstip, mut_spelersnr, mut_type)) ;

-- tabel om wijzigingen bij te houden
```

Voorbeeld: trigger functie

```
create or replace function insert_speler() returns trigger as
$body$
begin
    insert into mutaties values
        (user, current_date(), new.spelersnr, 'I', null) ;
end;
$body$
language sql;
```

Voorbeeld: trigger

```
create or replace trigger insert_speler.trg
    after insert
    on spelers
    -- when new.spelersnr < 10
    for each row
    execute procedure insert_speler() ;


-- OLD en NEW verwijzen naar de huidige toestand en de nieuwe toestand
-- welke verschillende onderdelen zie je hier die typisch voor een
-- trigger zijn ?
```

Onderdelen Trigger

- Trigger-moment + Trigger-gebeurtenis:
  - Wanneer activeren?
    - AFTER : nadat triggering instructie is verwerkt
    - BEFORE : eerst de trigger-actie
    - INSTEAD OF : alleen de trigger-actie
  - Voor welke rij activeren ?
    - FOR EACH ROW : voor elke rij
    - FOR EACH STATEMENT : voor een statement
  - Voor welke gebeurtenis?
    - INSERT, UPDATE, DELETE, (TRUNCATE)
- Trigger-Conditie: WHEN
- Trigger-actie: wat doet de trigger?

Syntax

```
CREATE [ OR REPLACE ] [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
    [ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_relation_name } [ ... ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE { FUNCTION | PROCEDURE } function_name ( arguments )
```

where event can be one of:

```
    INSERT
    UPDATE [ OF column_name [, ... ] ]
    DELETE
    TRUNCATE
```

URL: https://www.postgresql.org/docs/current/sql-createtrigger.html

Syntax

CREATE/ALTER/DROP TRIGGER

ALTER TABLE .. ENABLE/DISABLE TRIGGER ..

GRANT/REVOKE TRIGGER ON TABLE .. TO ..

(CREATE EVENT TRIGGER – DDL)

Standaard

- De standaard laat meer acties dan een trigger functie toe (udf : user defined function)
- Bv
  - create trigger delete_spelers
    after delete on spelers for each row
    begin
      delete from spelers_wed
      where spelersnr = old.spelersnr;
    end ;

    -- geen verwijzing naar udf, maar rechtstreeks de sql code

Notas

- Er zitten verschillen tussen producten :
  - Meerdere triggers op 1 tabel ? En wat met de volgorde ?
  - Kan een trigger een andere trigger activeren ? (waterval/domino !)
  - Wat mag een trigger allemaal doen ?
  - Hoe worden (complexere) triggers juist verwerkt ?
  - ..

Gebruik?

- Denk gebeurtenis gestuurd
- Voorbeelden :
  - (Integriteits)regels
  - Audit
  - Consistentie
  - Beveiliging
  - Afgeleide waarden berekenen
  - (Data)validatie
  - ..

Voordelen

- Onderhoudbaarheid:
  Vb. Snellere uitvoering door meer instructies in macro
- Verwerkingssnelheid
  Vb. minimaliseert netwerkverkeer
- « Precompilatie » bij stored procedures/functions/triggers
  - Planning en caching
- Werkt in verschillende host-languages

Nadelen

- Nadenken over architectuur en code organisatie
- Algemeen zoals bij andere talen : logische fouten vs syntaxfouten
  - Bv Onverwachte neveneffecten bij gebruik van (veel) (overlappende) triggers

Referenties

Slides: Stored Procedures Functions Triggers, P. Demazière, 2018
Slides: Procedurel SQL, H.Martens, W.Bertels,2014
Postgresql 11 Server Side Programming Quick Start Guide, L. Ferrari, 2018
https://www.postgresqltutorial.com/postgresql-plpgsql
https://www.postgresql.org/docs/current/plpgsql-trigger.html
https://www.postgresql.org/docs/current/sql-grant.html
https://www.postgresql.org/docs/current/triggers.html
https://www.postgresql.org/docs/current/sql-createtrigger.html
https://www.postgresql.org/docs/current/plpgsql.html
https://www.postgresql.org/docs/current/xfunc-sql.html