User Defined Functions and Triggers Tutorial

SWEN 304

Trimester 2, 2017

Lecturer: Dr Hui Ma

Engineering and Computer Science

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Outline

- Syntax of a UDF
- SQL UDFs and tables
- Procedural Language / PosgreSQL (PL/pgSQL)
  - Special Features of PL/pgSQL
    - SELECT ... INTO ...
    - FOUND
    - RAISE
    - The Special TRIGGER Syntax

Reading:

- *PostgreSQL 9.4 Documentation Chapter V.37*

The Basic PostgreSQL UDF Syntax

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ]
  argtype [, ...] ] )
    [ RETURNS rettype ]
  AS 'definition'
    LANGUAGE langname
```

Functions in SQL Queries With Tables (1)

- A powerful use of functions is in queries that retrieve data from tables

```
CREATE TABLE Rectangle
(RectId int, a real, b real);

INSERT INTO Rectangle
VALUES(1, 5.5, 6.6);

INSERT INTO Rectangle
VALUES(2, 3.3, 4.4);
```

RectId	a	b
1	5.5	6.6
2	3.3	4.4

Functions in SQL Queries With Tables (2)

```
CREATE OR REPLACE FUNCTION
area(real, real) RETURNS real
AS 'SELECT $1*$2;' LANGUAGE 'SQL';
```

```
SELECT RectId, area(a, b) as Rec_Area
FROM Rectangle;
```

RectId	Rec_Area
1	36.3
2	14.52

What is PL/pgSQL

- PL/pgSQL is a language that combines:
  - The expressive power of SQL with
  - The more typical features of a procedural programming language:
    - Control structures
    - Special SQL statements (SELECT . . . INTO. . .)

- It is aimed for:
  - Creating user defined functions
  - Creating trigger procedures
  - Efficient execution (vaguely speaking it is precompiled)
  - Easy of use

SELECT . . . INTO

- The result of a SELECT command yielding multiple columns (but only one row) can be assigned to a `record` variable

```
SELECT <attrbute_list> INTO <target>
FROM …
```

- If the type of `target` is `record`, `target` automatically configures to the query result
- If the result is empty, `target` will be

$$(\omega, \ldots, \omega)$$

a structure with null components

- If a `SELECT` command returns multiple rows only the first one retrieved is assigned to the `target`

A SELECT. . . INTO . . . Example

```
DECLARE
   s record;
BEGIN
   SELECT * INTO s FROM student
    where   studentid = 4;
   IF s.StudentId IS NULL THEN
        RAISE NOTICE ‘There is no
             student with id = 4';
        RETURN NULL;
   ELSE
        RETURN s;
   END IF;
END;
```

Obtaining the Result Status (FOUND)

- To determine the result of a command, you can check a special variable named `FOUND` of the `boolean` type
- The following commands set `FOUND`:
  - `SELECT…INTO…` sets it true if it returns any row, false otherwise
  - `PERFORM` sets it true if it returns (and discards) any row, false otherwise
  - `UPDATE`, `INSERT`, and `DELETE` set it true if at least one row is affected, false otherwise
- `FOUND` is a local variable within each PL/pgSQL function

An Example for the FOUND Variable

```
DECLARE
  t record;
BEGIN
   SELECT * INTO t FROM Grades
   WHERE StudentId = 7007
    AND CourseId = 'COMP302';
   IF NOT FOUND THEN
        SELECT * INTO t FROM Student
        WHERE StudentId = 7007;
   END IF;
   RETURN t;
END;
```

RAISE Statement

- Even though PL/pgSQL doesn't offer a way to intercept errors, the `RAISE` statement is provided to raise an error
- Syntax:

```
RAISE severity 'message' [, variable []];
```

  where `severity` can be:
  - `DEBUG`
  - `NOTICE` – just to send a message to the client application
  - `EXCEPTION` – to raise an exception, which is handled as described in the manual

A RAISE NOTICE Examples

```
DECLARE
    s record;
BEGIN
    SELECT * INTO s FROM Student;
    IF NOT FOUND THEN
         RAISE NOTICE 'Table is empty';
         RETURN null;
    ELSE
         RETURN s;
    END IF;
END;
```

- Another example

```
  SELECT * INTO s FROM Student
  WHERE NEW.StudId = StudId;
  IF NOT FOUND THEN
     RAISE NOTICE 'There is no student %',
      NEW.StudId;
  ...
```

The Trigger Syntax

```
<trigger>::= CREATE TRIGER
<trigger_name>
  {AFTER | BEFORE}
<triggering_event> [OR...]
 ON <table_name>
 [FOR [EACH] { ROW | STATEMENT }]
 EXECUTE PROCEDURE
<function_name>(arguments);


<triggering_event>::=   {INSERT |
DELETE | UPDATE}
```

Trigger Procedures in PL/pg SQL

- A trigger procedure is created with command

  `CREATE FUNCTION`

  - Does not have any parameters, and
  - Has a `trigger` return type
- When a PL/pgSQL function is called as a trigger, several special variables are created automatically in the top level block
- The most important automatically created variables:
  - `NEW` of data type `RECORD` holding the new database row for `INSERT/UPDATE` operations in row-level triggers
  - `OLD` of data type `RECORD` holding the old database row for `UPDATE/DELETE` operations in row-level triggers

RETURN Type

- A trigger has to return:
  - Either `NULL`, or
  - A record/row value having exactly the structure of the table the trigger was fired for
- The return value of a:
  - `BEFORE` or `AFTER` per-statement trigger, or an `AFTER` row-level trigger is always ignored
    - Both should be `NULL`
  - But these triggers can still abort an entire operation by raising an error

Return Value of a BEFORE ROW Trigger

- A row level trigger fired `BEFORE` may return `NULL` to signal the trigger manager to skip the rest of operations for this row (the `INSERT/DELETE/UPDATE` will not be executed for this row)
- If a `BEFORE` row level trigger returns a not null value, the operation proceeds with that row value:
  - Returning a row value (different from the original value) of `NEW` alters the row that will be inserted or updated (but has no influence on delete operation)
  - Altering a row to be stored is accomplished either by replacing single values directly in `NEW`, or building a completely new record/row

Triggers - Examples

- Consider the following part of a relational database schema:

```
Student(StudId, Name, NoOfPts, Degree)
Exam(StudId, CourseId, Term, Grade)
```

- Suppose DBMS does not support referential integrity constraints

Referential Integrity Trigger - INSERT

```
CREATE OR REPLACE FUNCTION ins_ref_int()
RETURNS trigger AS $$
DECLARE s RECORD;
BEGIN
  SELECT * INTO s FROM Student WHERE NEW.StudId =
    StudId;
  IF NOT FOUND THEN
    RAISE NOTICE 'There is no student %',NEW.StudId;
    RETURN NULL;
  ELSE
    RETURN NEW;
  END IF;
END;
$$ LANGUAGE 'PLpgSQL';
```

```
CREATE TRIGGER ins_ref_int
BEFORE INSERT ON Exam FOR EACH ROW
EXECUTE PROCEDURE ins_ref_int();
```

Another Example: Database and Triggers (1)

```
DROP TRIGGER phonebook on addressbook;
DROP FUNCTION add_to_phonebook();
DROP TABLE addressbook;
DROP TABLE phonebook;
```

```
CREATE TABLE addressbook (
    id integer,
    name text,
    address1 text,
    address2 text,
    address3 text,
    phonenum text);

CREATE TABLE phonebook (
    id integer,
    name text,
    phonenum text);
```

Another Example: Database and Triggers (2)

```
CREATE OR REPLACE FUNCTION add_to_phonebook()
RETURNS TRIGGER AS $phonebook$
DECLARE
new_name varchar;
new_phonenum varchar;
BEGIN
  IF(TG_OP='INSERT') THEN
  INSERT INTO phonebook(name, phonenum)
    VALUES(NEW.name, NEW.phonenum);
  END IF;
  RETURN NEW;
END;
$phonebook$ LANGUAGE plpgsql;
```

Another Example (3)

```
CREATE TRIGGER phonebook
 AFTER INSERT
ON addressbook FOR EACH ROW
EXECUTE PROCEDURE add_to_phonebook();
```

```
INSERT INTO addressbook (id, name, address1,
   address2,  address3, phonenum) values (4055,
   'Peter', 'Kelburn', 'Wellington', 'NZ', 4567890 );
```

```
SELECT * FROM phonebook;
```

```
id  |  name   |  phonenum
------+---------+-----------
   | Peter     | 4567890
```

Can we improve the trigger?

Summary

- PL/pgSQL is a simple block structured language that combines procedural constructs with SQL statements
- It is designed to provide for:
  - Creating user defined functions
  - Creating trigger procedures

- Triggers are active rules that are automatically fired when an event occurs
  - In the PostgreSQL environment triggers use PL/pgSQL procedures
  - Triggers are extensively used to implement constraints