Stored Procedures
User Defined Functions
and PL / pgSQL

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

Slides by: Pavle Morga & Hui Ma

Outline

- What are stored procedures?
- What is a User Defined Function (UDF)?
- Procedural Language / PostgreSQL (PL/pgSQL)
  - Language Structure
  - Declarations
  - Parameters and Types
  - Special Statements
  - Control Structures
- *Reading:*
  - *PostgreSQL 9.4 Documentation Chapter V.37*

Motives for Using Stored Procedures

- So far we implicitly assumed that we use a database from a client machine and issue SQL statements to a database
- DBMS and the database itself reside on a different machine – database Server machine
- There are a number of situations that make running small database procedures or functions on the server itself preferable
- These small programs are historically known as **database stored procedures**
- SQL:1999 standard calls them **persistent stored modules (PSM)**

Database Stored Procedures

- Persistent procedures/functions (modules) are stored locally and executed by the database server
    - As opposed to execution by clients
- Advantages:
    - If the procedure is needed by many applications, it can be invoked by any of them (thus reduce duplications)
    - Execution by the server reduces communication costs
    - Enhance the modeling power of views
- Disadvantages:
    - Every DBMS has its own syntax and this can make the system less portable

Use Stored Procedures

- When to use:
  - Several applications need the same procedure,
    - Multiple client program can call the procedures, which reside in the database server
  - Data transfer should be reduced,
    - Multiple SQL statement can be invoked with a single procedure, with only the request and the final results needed to be sent across the network
  - Tighter security control is needed,
    - To access specific data items, a UDF can be implemented. Only need to grant access to call the UDF
    - No need any additional authorization to the underlying database objects

Use Stored Procedures

- When to use:
  - More complex types of derived data should be made available to users,
  - Efficient execution of precompiled procedures is needed
- Stored procedures allow for greater flexibility, if they are changed all programs that call them remain intact
  - Code changes are only required in one place
  - Easy maintenance

Declaring a Stored Procedure

- Commercial DBMS allow writing stored procedures combining SQL and general purpose languages
- A stored procedure can be either a procedure or a function
- It is stored as a database object and can be:
  - Used in SELECT statements, or
  - Invoked by a CALL statement
    - CALL <procedure or function name> (<argument list>)
    - Called from within JDBC transaction programs, where a CollableStatement object has to be used

SQL/Persistent Stored Modules (PSMs) (*)

- SQL/PSM was introduced in 1999 as part of SQL standard for writing persistent stored modules
- It includes:
  - Statements to create procedures and functions, and
  - Procedural programming constructs for writing the body of a procedure or a function
  - The body can also be a pure SQL statement
- Procedural constructs include:
  - Conditional IF/THEN/ELSE constructs, and
  - Several looping constructs:
    - WHILE/DO/END WHILE
    - REPEAT/UNTIL/END REPEAT

User Defined Functions (UDFs)

- A DBMS comes with many predefined functions
    - Integer addition,
    - String concatenation
- An ORDBMS allows users to define new functions
- These functions are called server side functions since they reside at the database server
- Unlike client side functions, server side functions may be used within SQL statements
- Each server side function may be called from any client side program

The basic PostgreSQL UDF Syntax

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

`argmode` is one of:

- IN,
- OUT,
- INOUT

A Simple SQL Function

- A user defined function may be written using SQL

```
CREATE FUNCTION poundtokg(float)
RETURNS float
AS ‘SELECT $1*0.545;’
LANGUAGE ‘SQL’;
```

```
SELECT poundtokg(3.0);
```

```
poundtokg
---------------------
1.635
(1 row)
```

- $1 is a positional parameter

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

Advantages of using PL/pg SQL

- Conventional queries:
  - Each SQL statement is submitted from a client to the database server for execution
  - The server returns each result to the client
  - This incurs inter process communication and may incur network overhead
- User defined functions written in PL/pgSQL:
  - Reside on database server
  - Compile once per duration of a database connection
  - Can comprise several SQL queries connected by procedural constructs
  - All SQL queries of a function are executed with no intermediate client/server communication

Supported Data Types

- PL/pgSQL functions accept as arguments and can return as results any of the following data types:
  - Scalar (base and user defined),
  - Array,
  - Composite (row type),
  - Record (a placeholder whose structure is defined by the calling query)
  - Polymorphic types (`anyelement or anyarray`),
  - Set,
  - Void (no useful return value), and
  - Null

Body of a PL/pgSQL Function

```
CREATE OR REPLACE FUNCTION some_func()
RETURNS <data_type> AS
$$--start of the body
-- here comes the function body
[DECLARE
 declarations]
BEGIN - -denotes start of block statements
END;--denotes the end of a block
$$--end of the body
LANGUAGE ‘plpgsql’;
```

Language Structure

- PL/pgSQL is a block structured language
  - Each block starts with either a `DECLARE`, or `BEGIN` statement
  - Ends with an `END;` statement
  - Each block can contain another block (also called subblock)

```
BEGIN
    statements
    BEGIN
          statements
    END;
END;
```

- `BEGIN/END` statements do not denote the start and the end of a transaction, but only the boundaries of a block

Declarations

- All variables in a block must be declared in the declaration section of the block (the only exception is the variable of the `for` loop, which is an `integer` by default)

Name [CONSTANT] type [NOT NULL] [{DEFAULT | := } expression]

- Examples:
  - `StudenName varchar;--initialized to null`
  - `NumOfMarks real := 100.00; --assignment`
  - `Grade char(2) DEFAULT ‘A+’;`

Function Parameters (1)

- There are two ways to define function parameters
- One is to define their types only

```
CREATE FUNCTION area(int, int) RETURNS
int
AS $$
   BEGIN
        RETURN $1*$2;
   END;
$$ LANGUAGE ‘plpgsql’;
```

- Here, `$1` contains the value of the first parameter, and `$2` contains the value of the second parameter

Function Parameters (2)

- The other way to define function parameters is

```
CREATE FUNCTION area(a int, b int)
RETURNS int
AS $$
   BEGIN
        RETURN a*b;
   END;
$$ LANGUAGE ‘plpgsql’;
```

- Here, a contains the value of the first parameter, and b contains the value of the second parameter

Record Type

- Variables of the `record type` have no predefined structure
- They take on the actual row structure of the row they are assigned to during a `SELECT` command
- The structure of a `record` variable can change each time it is assigned to
- The `record` type is not a true type, only a record placeholder

An Example for the Record Type

```
DECLARE
  t record;
BEGIN
  IF p < 0 THEN  --p is a variable
   SELECT * INTO t FROM Student
   WHERE StudentId = 7007;
  ELSE
   SELECT * INTO t FROM Grades
   WHERE StudentId = 7007 AND CourseId
   = ‘COMP302’;
  END IF;
  return t;
END;
```

Obtaining the Result Status (`FOUND`)

- To determine the result of a SQL command, you can check a special variable named `FOUND` of the type `boolean`
- The following commands set `FOUND`:
  - `SELECT. . .INTO. . .` sets it true if it returns 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

Obtaining the Result Status (FOUND): Example

```
SELECT * INTO myrec FROM emp
WHERE exmpname = myname;
IF NOT FOUND THEN
   RAISE EXCEPTION ‘exployee % not found’,
                    myname;
END IF;
```

RETURN Statement

- To return single valued data from a function we use:

  `RETURN expression`

  statement
- The RETURN statement with expression terminates the function and returns the value of the expression to the caller
- The return value of a function must be defined:
  - If control reaches the end of the top-level block without hitting a RETURN statement, a run-time error will occur
  - Even if the function's return value is declared to be void, a RETURN statement must still be provided (but will return nothing)

Control Statements

- PL/pgSQL supports different forms of `IF` conditionals:
  - `IF. . .THEN. . .END IF;`
  - `IF. . .THEN. . .ELSEIF. . .THEN. . .ELSE. . . END IF;`
- An Example

```
If number = 0 THEN
   Result := ‘zero’;
ELSIF number > 0 THEN
   result := ‘positive’
ELSIF number < 0 THEN
   result := ‘negative’
ELSE
   result := ‘NULL’;
END IF;
```

Control Statements

- PL/pgSQL supports the following looping statements:
  - `LOOP,`
  - `EXIT,`
  - `WHILE`, and
  - `FOR`
  - Have a look in the manual for particulars
- An example

```
LOOP
  --some computations
  IF count > 0 THEN
    EXIT; -- exit loop
  END IF;
END LOOP;
```

Summary (1)

- Stored Procedures have a widespread use in industry since they allow for:
  - code sharing,
  - tighter security control,
  - efficient execution
- User Defined Functions (UDFs):
  - can be defined using SQL only, or a combination of SQL and a procedural language
  - can be used as stored procedures,
  - can be used as trigger procedures

Summary (2)

- PL/pgSQL is a simple language that combines procedural constructs with SQL statements
- It is designed to provide for:
  - Creating user defined functions
  - Creating trigger procedures
  - Efficient execution (vaguely speaking it is precompiled)
  - Ease of use
- It is block structured
- Supports
  - A big range of data types
  - Conditionals
  - Looping
  - Special statements (`SELECT` **`INTO`**)

Next topic

- Triggers
- Readings:
  - *Textbook, Section 24.1*
  - *PostgreSQL 9.4 Documentation , CREATE TRIGGER statement, Chapter V.34 and Chapter V.37*