CSCI235 Database Systems

PL/pgSQL

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

PL/pgSQL

Outline

PL/pgSQL ? What is it ? Why do we need it ?

PL/pgSQL is a procedural extension of SQL

PL/pgSQL = procedural Programming Language + pgSQL (PostgreSQL)

We need PL/pgSQL to bridge a gap between a high level declarative query language and a procedural programming language

PL/pgSQL =

- Data Manipulation statements of SQL +
- `SELECT` statement +
- variables +
- assignment statement +
- conditional control statements +
- repetition statement +
- exception handling +
- procedure and function statements

PL/pgSQL

Outline

Program structure

PL/pgSQL is a block-structured language

It means that the basic units of PL/pgSQL programs such as functions and procedures have the structures of nested blocks

A block consist of the optional label and declarative componens and a compulsory executable component

Block

```
[ <<label>>]
[ DECLARE
    declarations ]
BEGIN
    statements
END [ label ]
```

Blocks can be nested to any level

A declarative component consists of declarations of constants, variables, types, methods, cursors, etc, and it is optional

An executable component consists of executable code

Program structure

A label is needed to identify the block when `EXIT` statement is used, or to qualify the names of the variables declared in the block

Any statement in an executable component of a block can be a nested block

Nested blocks

```
CREATE FUNCTION nested() RETURNS integer AS $$
DECLARE
    value integer := 5;
BEGIN
    DECLARE
        value integer := 10;
    BEGIN
        RAISE NOTICE 'Inner value= %', value;  -- 10
    END;
    RAISE NOTICE     'Outer value= %', value;  -- 5
    RETURN value;
END;
$$ LANGUAGE plpgsql;
```

PL/pgSQL

Outline

Declarative components

Declarative components contain declarations of variables, constants and cursors

```
DECLARE                                                    Start of declarations
   value INTEGER;                                          Integer variable, 4 bytes
   stock_num DECIMAL(5);                                   Binary coded decimal
   stock_num NUMERIC(5);                                   Equivalent to DECIMAL
   stock_name CHAR(5);                                     fized size string
   stock_name VARCHAR(30);                                 Variable size string, unlimited length
   stock_date DATE;                                        Date type
   stock_required CONSTANT NUMERIC(5) := 30;               Numerical constant
   max_salary CONSTANT NUMERIC(11,2) := 2.45;              Numerical Constant
   course_row COURSE%ROWTYPE;                              Type derived from a row
   course_number COURSE.cnum%TYPE;                         Type derived from a column
   query CURSOR FOR SELECT * FROM COURSE;                  Cursor
```

Executable components

Executable components include assignment statements, conditional control statements, iterative statements, procedure and function calls, SQL statements

Assignment statement

```
department_code := 'SCIT';
```

SELECT statement with assignment

```
SELECT name
INTO STRICT department_name
FROM DEPARTMENT
WHERE code = department_code;
```

IF statement

```
IF (credits < limit) THEN
  credits := credits + 1;
  limit := limit - 1;
ELSIF (credits = limit) THEN
  credits := credits - 2;
ELSE
  limit := limit + 1;
END IF;
```

FOR statement

```
FOR i IN 1..100 LOOP
  total := total + i;
END LOOP;
```

Exception components

Exception component consists of executable statements that service the exceptional situations during execution

No results from SELECT statement

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    INSERT INTO AUDIT_TABLE VALUES( current_timestamp, course_number );
      RAISE NOTICE 'Course % not found', course_number;
```

Two many rows for a single variable

```
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    RAISE EXCEPTION 'Course % not unique', title;
```

Division by zero

```
EXCEPTION
  WHEN DIVISION_BY_ZERO THEN
    RAISE NOTICE 'Caught division_by_zero';
```

Any possible exception

```
EXCEPTION
  WHEN OTHERS
    INSERT INTO AUDIT_TABLE VALUES( current_timestamp, 'Error' );
    RAISE EXCEPTION 'Something went wrong';
END;
```

PL/pgSQL

Outline

Functions

A birds-eye view of a function is the following

```
CREATE OR REPLACE FUNCTION function_name ( parameters )          Statement code
RETURNS type-specification AS                                    Type of value retruned
$$ -- Function body separator                                    Start separator
DECLARE                                                          Keyword
  -- optional declarations                                       Declarations
BEGIN                                                            Keyword
  -- executable statements, at least one statements is required  Statements
RETURN value;                                                    Value returned
EXCEPTION                                                        Keyword
  -- optional exception handlers                                 Exceptions
END;                                                             Keyword
$$ -- Function body separator                                    End separator
LANGUAGE plpgsql;                                                Langage used
```

Functions

A sample function `hello_world`

Fuction hello_world

```
CREATE OR REPLACE FUNCTION hello_world( hello text,
                                        world text )
RETURNS text AS
$$
DECLARE
  hello_message text;
BEGIN
  hello_message := hello || ' ' || world || ' !';
  RETURN hello_message;
END;
$$
LANGUAGE plpgsql;
```

A sample processing of `hello_world` function

Processing of hello_world function

```
SELECT hello_world('Hello', 'world');
  hello_world
---------------
Hello world !
```

Functions

Processing of SQL statements within a function `raise_budget`

```
CREATE OR REPLACE FUNCTION raise_budget( department_name text,                  Function name and paratemeters
                                        budget_ limit NUMERIC)                  Parameters
RETURNS NUMERIC AS $$                                                          Type of value returned and start separator
DECLARE                                                                        Keyword
  current_budget DEPARTMENT.budget%TYPE;                                       Declaration of variable
BEGIN                                                                          Keyword
  SELECT budget INTO current_budget FROM DEPARTMENT WHERE name = department_name;   SELECT statement
  IF current_budget < budget_limit THEN                                        IF statement
    UPDATE DEPARTMENT SET budget = budget_limit WHERE name = department_name;  UPDATE statement
    RETURN budget_limit;                                                       Value returned
  ELSE                                                                         ELSE clause
    INSERT INTO AUDIT VALUES( 'Budget OK', current_budget);                    INSERT statement
      RETURN current_budget;                                                   Value returned
  END IF;                                                                      END IF keyword
END; $$                                                                        END keyword and end separator
LANGUAGE plpgsql;                                                              Language used
```

Procedures

A birds-eye view of a procedure is the following

```
                                                                  Procedure name and parameters
CREATE OR REPLACE PROCEDURE procedure_name ( parameters ) AS
$$ -- Procedure body separator                                    Start separator
DECLARE                                                          Keyword
 -- optional declarations                                        Declarations
BEGIN                                                            Keyword
  -- executable statements, at least one statements is required  Executables
EXCEPTION                                                        Keyword
  -- optional exception handlers                                 Exceptions
END;                                                             Keyword
$$                                                               End separator
LANGUAGE plpgsql;                                                Language used
```

Procedures

A sample `phello_world` procedure

Procedure phello_world

```
CREATE OR REPLACE PROCEDURE phello_world ( hello text,
                                           world text,
                                           OUT message text) AS
$$
BEGIN
  message := hello || ' ' || world || ' !';
END;
$$
LANGUAGE plpgsql;
```

A `CALL` statement can be to process a procedure `phello_world`

Processing of phello_world procedure

```
CALL phello_world('Hello', 'world', NULL);
    message
---------------
 Hello world !
(1 row)
```

Procedures

Processing SQL statements within a procedure `praise_budget`

```
CREATE OR REPLACE PROCEDURE praise_budget(department_name text, budget_limit NUMERIC) AS    Procedure
$$                                                                                          Start separator
DECLARE                                                                                     Keyword
  current_budget DEPARTMENT.budget%TYPE;                                                    Declaration
BEGIN                                                                                       Keyword
SELECT budget INTO current_budget  FROM DEPARTMENT  WHERE name = department_name;           SELECT statement
  IF current_budget < budget_limit THEN                                                     IF statement
    UPDATE DEPARTMENT SET budget = budget_limit WHERE name = department_name;               UPDATE statement
  ELSE                                                                                      Keyword
    INSERT INTO AUDIT VALUES( 'Budget OK', current_budget);                                 INSERT statement
  END IF;                                                                                   Keyword
END;                                                                                        Keyword
$$                                                                                          End separator
LANGUAGE plpgsql;                                                                           Language used
```

PL/pgSQL

Outline

Data types

Some of the most frequently used predefined data types in PL/pgSQL

PL/Predefined data types

```
INTEGER, REAL, DECIMAL, NUMERIC, CHAR, DATE, TIME, VARCHAR, BOOLEAN, TEXT
```

Sample implicit type declarations

Implicit type declarations

```
DECLARE
  course_number COURSE.cnum%TYPE;
  course_title COURSE.title%TYPE;
  course_row COURSE%ROWTYPE;
BEGIN
  course_number := 'CSCI235';
  SELECT title INTO course_title FROM COURSE WHERE cnum = course_number;
  course_row.cnum := 'CSIT115';
  course_row.title := 'Data Management and Security';
  course_row.credits := 6;
  course_row.offered_by := 'Computer Science';
  INSERT INTO COURSE VALUES(course_row.cnum, course_row.title, course_row.credits,
                            course_row.offered_by);
END;
```

PL/pgSQL

Outline

Operators

Arithmetic operators

```
+, -, *, /, ^
```
Arithmetic

Relational operators

```
<, >, >=, <=, =, !=, <>
```
Relational

Comparison operators

```
LIKE, BETWEEN, IN, IS NULL, =, !=, <>
```
Comparison

Boolean operators

```
AND, OR, NOT
```
Boolean

String operator

```
||
```
String

Operator precedence

```
(unary +,-),(^),(*,/),(+,-,||),(comparison),(NOT),(AND),(OR)
```
Precedence

PL/pgSQL

Outline

Conditional control statements

A birds-eye view of conditional control statements is the following

IF statement

```
IF condition THEN
  statement;
  ...
ELSE
  statement;
  ...
END IF;
```

IF statement

```
IF condition THEN
statement;
...
ELSIF condition THEN
  statement;
  ...
ELSIF condition THEN
  statement;
  ...
ELSE
  statement;
...
END IF;
```

Iterative control statements

A birds-eye view of iterative control statements is the following

LOOP statement

```
LOOP
  statement;
  ...
  IF condition THEN EXIT;
    statement;
    ...
  END IF;
  statement;
  ...
END LOOP;
```

FOR statement with LOOP statement

```
FOR variable IN scope
LOOP
  statement;
  ...
END LOOP;
```

FOR statement with LOOP statement

```
FOR variable IN REVERSE scope
LOOP
  statement;
  ...
END LOOP;
```

Iterative control statements

A birds-eye view of iterative control statements is the following

WHILE loop

```
WHILE (condition)
LOOP
  statement;
  ...
END LOOP;
```

EXIT a loop

```
LOOP
  statement;
  ...
   EXIT WHEN condition;
  statement;
  ...
END LOOP;
```

EXIT a block

```
<<label>> BEGIN
  statement;
  ...
   IF condition THEN EXIT label;
  statement;
  ...
END;
```

PL/pgSQL

Outline

Cursors

What happens when `SELECT` statement returns more than one row ?

Stored procedure

```
CREATE OR REPLACE PROCEDURE error() AS $$
DECLARE
  course_title COURSE.title%TYPE;
BEGIN
  SELECT title INTO STRICT course_title FROM COURSE WHERE cnum IN ('CSCI235','CSCI205');
  RAISE NOTICE 'Title: %',course_title;
END;
$$ LANGUAGE plpgsql;
```

Processing error

```
call error();
ERROR:  query returned more than one row
HINT:  Make sure the query returns a single row, or use LIMIT 1.
CONTEXT:  PL/pgSQL function error() line 5 at SQL statement
```

A variable `course_title` with a keyword `STRICT` cannot be used to store several rows retrieved from a relational table

A solution is to process the rows in a row by row mode

A cursor is a construction that allows for processing the rows retrieved from the relational tables in a row by row mode

Cursors

Declaration and processing of a bound cursor

Declaration of a bound cursor variable

```
CREATE OR REPLACE PROCEDURE BCURSOR() AS
$$
DECLARE
  TITLES CURSOR FOR SELECT title FROM COURSE WHERE cnum IN ('CSCI235','CSCI205');
  course_title text;
```

Processing of a bound cursor

```
BEGIN
  OPEN TITLES;
  LOOP
    FETCH TITLES INTO course_title;
    EXIT WHEN NOT FOUND;
    RAISE NOTICE 'Title: %',course_title;
  END LOOP;
CLOSE TITLES;
END;
$$
LANGUAGE plpgsql
```

Processing of a stored procedure with a cursor

```
CALL BCURSOR();
```

Cursors

Declaration and processing of an unbound cursor

Declaration of an unbound cursor variable

```
CREATE OR REPLACE PROCEDURE UCURSOR() AS
$$
DECLARE
  course_title text;
  TITLES REFCURSOR;
```

Processing of an unbound cursor

```
BEGIN
  OPEN TITLES FOR SELECT title FROM COURSE WHERE cnum IN ('CSCI235','CSCI205');
  LOOP
    FETCH TITLES INTO course_title;
    EXIT WHEN NOT FOUND;
    RAISE NOTICE 'Title: %',course_title;
  END LOOP;
CLOSE TITLES;
END;
$$
LANGUAGE plpgsql;
```

Processing of a stored procedure with a cursor

```
CALL UCURSOR();
```

Cursors

Declaration and implicit processing of a bound cursor

Declaration of a bound cursor variable

```
CREATE OR REPLACE PROCEDURE IBCURSOR() AS
$$
DECLARE
  TITLES CURSOR FOR SELECT title FROM COURSE WHERE cnum IN ('CSCI235','CSCI205');;
```

Implicit processing of a bound cursor

```
BEGIN
  FOR trow IN TITLES
  LOOP
    RAISE NOTICE 'Title: %',trow.title;
  END LOOP;
END;
$$
LANGUAGE plpgsql;
```

Processing of a stored procedure with a cursor

```
CALL IBCURSOR();
```

PL/pgSQL

Outline

Exceptions

An exception is an internally defined or user defined error condition, e.g. divide by zero, no rows selected by `SELECT` statement with `INTO` clause, failure of `FETCH` statement, use of a cursor which has not been opened yet, etc.

Handling of no rows selected exception

No rows selected

```
CREATE OR REPLACE PROCEDURE exception1() AS
$$
DECLARE
  course_title COURSE.title%TYPE;
BEGIN
  SELECT title INTO STRICT course_title FROM COURSE WHERE cnum = 'ABCD123';
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE NOTICE 'Course ABCD123 not found';
END;
$$
LANGUAGE plpgsql;
```

No rows selected

```
call exception1();
NOTICE:  Course ABCD123 not found
```

Exceptions

Handling too many rows selected exception

Too many rows selected

```
CREATE OR REPLACE PROCEDURE exception2() AS
$$
DECLARE
  course_title COURSE.title%TYPE;
BEGIN
  SELECT title INTO STRICT course_title FROM COURSE;
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    RAISE NOTICE 'Too many rows found';
END;
$$
LANGUAGE plpgsql;
```

Too many rows selected

```
call exception2();
NOTICE:  Too many rows found
```

Exceptions

Handling of other exception together with a diagnostics report

Other

```
CREATE OR REPLACE PROCEDURE exception3() AS $$
DECLARE
  message text;
  stack text;
  i integer := 5;
BEGIN
  i := i/0.0;
EXCEPTION
  WHEN OTHERS THEN
    GET STACKED DIAGNOSTICS message := MESSAGE_TEXT,
                            stack := PG_CONTEXT;
    RAISE NOTICE 'Message text: %', message;
    RAISE NOTICE E'--- Call Stack ---\n%', stack;
END; $$
LANGUAGE plpgsql;
```

Other

```
call exception3();
NOTICE:  Message text: division by zero
NOTICE:  --- Call Stack ---
PL/pgSQL function exception3() line 10 at GET STACKED DIAGNOSTICS
```

PL/pgSQL

Outline

Anonymous code blocks

Anonymous code block is a transient anonymous function in a procedural language

Syntactically anonymous code block is a block of code without `CREATE FUNCTION` or `CREATE PROCEDURE` header

A statement `DO` can be used to process anonymous code block

A sample anonymous code block

Anonymous code block

```
DO
$$
BEGIN
  IF (SELECT count(*) FROM course) > 1 THEN
    RAISE NOTICE 'COURSE table is not empty';
  END IF;
END;
$$
LANGUAGE plpgsql;
```

Anonymous code blocks

Yet another sample anonymous code block

Anonymous code block

```
DO
$$
DECLARE
 dept_name DEPARTMENT.name%TYPE := 'Computer Science';
 dept_code DEPARTMENT.code%TYPE;
BEGIN
  SELECT name, code
  INTO dept_name, dept_code
  FROM DEPARTMENT
  WHERE name = dept_name;

  RAISE NOTICE 'A code of department % is %', dept_name, dept_code;
END;
$$
LANGUAGE plpgsql;
```

References

PostgreSQL 16.1 Documentation, Chapter 43. PL/pgSQL — SQL Procedural Language

PostgreSQL Tutorial, PostgreSQL PL/pgSQL

T. Connoly, C. Begg, Database Systems, A Practical Approach to Design, Implementation, and Management, Chapter 8 Advanced SQL, Pearson Education Ltd, 2015