PL/SQL Tutorial

Need for PL/SQL — declarative vs. procedural — anonymous blocks — debugging — a first program — code compilation — code execution — procedures & functions — PL/SQL in SQL — SQL in PL/SQL — cursors & loops — operators & built-in functions reference tables.

Introduction

PL/SQL is a database-oriented programming language that extends Oracle SQL with procedural capabilities. We will review in this lab the fundamental features of the language and learn how to integrate it with SQL to help solve database problems.

Need for PL/SQL

SQL statements are defined in term of constraints we wish to fix on the result of a query. Such a language is commonly referred to as *declarative*. This contrasts with the so called procedural languages where a program specifies a list of operations to be performed sequentially to achieve the desired result. PL/SQL adds *selective* (i.e. if...then...else...) and *iterative* constructs (i.e. loops) to SQL.
PL/SQL is most useful to write *triggers* and *stored procedures*. Stored procedures are units of procedural code stored in a compiled form within the database.

PL/SQL Fundamentals

PL/SQL programs are organised in functions, procedures and packages (somewhat similar to Java packages). There is a limited support for object-oriented programming. PL/SQL is based on the Ada programming language, and as such it shares many elements of its syntax with Pascal.

Your first example in PL/SQL will be an *anonymous block* —that is a short program that is ran once, but that is neither named nor stored persistently in the database.

```
SQL> SET SERVEROUTPUT ON

SQL> BEGIN
  2   dbms_output.put_line('Welcome to PL/SQL');
  3  END;
  4  /
```

- `SET SERVEROUTPUT ON` is the SQL*Plus command[1] to activate the console output. You only need to issue this command once in a SQL*Plus session.
- the keywords `BEGIN...END` define a *scope* and are equivalent to the curly braces in Java `{...}`

- a semi-column character (`;`) marks the end of a statement
- the `put_line` function (in the built-in package `dbms_output`) displays a string in the SQL*Plus console.

You are referred to Table 2 for a list of operators, and to Table 3 for some useful built-in functions.

Compiling your code.

PL/SQL code is compiled by submitting it to SQL*Plus. Remember that it is advisable to type your program in an external editor, as you have done with SQL (see Introduction to Oracle).

Debugging.

Unless your program is an anonymous block, your errors will *not* be reported. Instead, SQL*Plus will display the message ``warning: procedure created with compilation errors''. You will then need to type:

```
SQL> SHOW ERRORS
```

to see your errors listed. If yo do not understand the error message and you are using Oracle on UNIX, you may be able to get a more detailed description using the `oerr` utility, otherwise use Oracle's documentation (see References section). For example, if Oracle reports ``error PLS-00103'', you should type:

```
oerr PLS 00103
```

at the *UNIX command prompt* (i.e. not in SQL*Plus).

Executing PL/SQL

If you have submitted the program above to Oracle, you have probably noticed that it is executed straight away. This is the case for anonymous blocks, but not for procedures and functions. The simplest way to run a function (e.g. `sysdate`) is to call it from within an SQL statement:

```
SQL> SELECT sysdate FROM DUAL
  2  /
```

Next, we will rewrite the anonymous block above as a *procedure*. Note that we now use the `user` function to greet the user.

```
CREATE OR REPLACE PROCEDURE welcome
IS
  user_name VARCHAR2(8) := user;
BEGIN  -- `BEGIN' ex
  dbms_output.put_line('Welcome to PL/SQL, '
    || user_name || '!');
```

```
END;
/
```

Make sure you understand the changes made in the code:

- A variable `user_name` of type `VARCHAR2` is *declared*
- `user_name` is *initialised* using the `user`[2] built-in function
- ``:=" is the *assignment* operator (see. Table 2)

Once you have compiled the procedure, execute it using the `EXEC` command.

```
SQL> EXEC welcome
```

Both procedures and *functions* should remind you of Java methods. The similarities and differences between them are outlined in Table 1.

Table 1: Functions, procedures and Java methods compared.

	Function	**Procedure**	**Java Method**
Parameters	input, output	input, output	input
Returns value	yes	no	optional
Can be called within SQL	yes	no	

PL/SQL Tutorial, Part 2

Need for PL/SQL — declarative vs. procedural — anonymous blocks — debugging — a first program — code compilation — code execution — procedures & functions — PL/SQL in SQL — SQL in PL/SQL — cursors & loops — operators & built-in functions reference tables.

Embedding SQL in PL/SQL

PL/SQL alone does not allow us to query a database, and use the resulting data in our program. However, any SQL (i.e. DML) may be embedded in PL/SQL code. In particular, there exists a form of the ``SELECT" statement for assigning the result of a query to a variable. Note the following code requires the `books` and `book_reviews` tables that you should have created during the first Oracle tutorial.

```
1  CREATE OR REPLACE PROCEDURE count_reviews
2    (author_param VARCHAR2)
3  IS
4    review_count NUMBER;
5  BEGIN
6    SELECT COUNT(*) INTO review_count
7    FROM book_reviews r, books b
8    WHERE b.isbn = r.isbn  AND author = author_param;
9
10   IF review_count > 1 THEN
11    dbms_output.put_line('There are '
12      || review_count || ' reviews.');
12   ELSIF review_count = 1 THEN
14     dbms_output.put_line('There is 1 review.');
15   ELSE
16     dbms_output.put_line('There is no review.');
17   END IF;
18 END;
19 /
```

Note in the code above how:

- the procedure takes one parameter `author_param` of type `VARCHAR2`
- a value from an SQL query is assigned to a PL/SQL variable (i.e. `review_count`) using `SELECT...INTO...` (line 6)
- a value from a PL/SQL variable is used in an SQL statement (line 8)

Try the programs with different authors:

```
EXEC count_reviews('Oscar Wilde')
EXEC count_reviews('Charles Dickens')
```

Working with Cursors

The last program we are going to write will display the number of reviews relevant to each author. Notice that the query may now return multiple rows. However, a `SELECT...INTO...` statement can only retrieve data from (at most) *one* tuple into individual variables.
Cursors[3] provide a means to retrieve multiple rows into a buffer (when you `OPEN` the cursor) that can then be traversed sequentially (`FETCH`) to retrieve individual rows—until there is no more data (`cur_revs%NOTFOUND` becomes `true`).

```
CREATE OR REPLACE PROCEDURE count_by_author
IS
  auth VARCHAR2(30);
  cnt NUMBER;
  CURSOR cur_revs IS
    SELECT author, COUNT(author) AS revs_cnt
    FROM books b, book_reviews r
    WHERE b.isbn = r.isbn GROUP BY author;
BEGIN
  OPEN cur_revs;
```

```
  LOOP
    FETCH cur_revs INTO auth, cnt;
  EXIT WHEN cur_revs%NOTFOUND;

  IF cnt = 1 THEN dbms_output.put_line('1 review for '
    || auth);
  ELSE
    dbms_output.put_line(cnt || ' reviews for ' || auth);
  END IF;

  END LOOP;
CLOSE CUR_REVS;
END;
/
```

Execute `count_by_author`, adding more data to the tables if necessary.

Table 2: PL/SQL operators.

Operator	Description
`+ - / *`	arithmetic
`=`	equality
`!=` or `<>`	inequality
`||`	string concatenation
`:=`	assignment

Table 3: Some Oracle built-in functions. You are referred to Oracles's documentation (see References section) for specific usage examples.

Function	Description
String Functions	
`upper(s), lower(s)`	convert string `s` to upper/lower-case
`initcap(s)`	capitalise first letter of each word
`ltrim(s), rtrim(s)`	remove blank char. from left/right
`substr(s,start,len)`	sub-string of length `len` from position `start`
`length(s)`	length of `s`
Date Functions	

`to_date(date, format)`	date formatting
Number Functions	
`round(x)`	round real number `x` to integer
`mod(n,p)`	`n` modulus `p`
`abs(x)`	absolute value of `x`
`dbms_random.random()`	generate a random integer
Type Conversion Functions	
`to_char()`	convert to string
`to_date()`	convert to date
`to_number()`	convert to number
Miscellaneous Functions	

References

You can copy & paste the following URI (note that you will need a username/password to access Oracle's web site. You can use `database@example.com/database`):

PL/SQL User's Guide and Reference:

http://download-west.oracle.com/docs/cd/A91202_01/901_doc/appdev.901/a89856/toc.htm