

Weird PL/SQL



Steven Feuerstein
PL/SQL Evangelist, Quest Software
www.quest.com steven.feuerstein@quest.com



How to benefit most from this session

- Watch, listen, *ask questions*. Then afterwards....
- Download and use any of my the training materials, available at my "cyber home" on Toad World, a portal for Toad Users and PL/SQL developers:

PL/SQL Obsession

<http://www.ToadWorld.com/SF>

- Download and use any of my scripts (examples, performance scripts, reusable code) from the demo.zip, available from the same place.
`filename_from_demo_zip.sql`
- You have my permission to use *all* these materials to do internal trainings and build your own applications.
 - But they should not considered production ready.
 - You must test them and modify them to fit your needs.



I Love PL/SQL And...

- **We all love PL/SQL.**
- **But we all know there are ways in which it could be improved.**
 - And I will be pointing to lots of such ways in this seminar.
- **I created ILovePLSQLAnd.net to make it easy for PL/SQL users to influence the future direction of the language.**
- **Please use it!**

<http://www.ILovePLSQLAnd.net>



The agenda

- **C'mon, PL/SQL is *not* weird.**
- **SQLERRM and DBMS_OUTPUT.PUT_LINE**
- **Error codes: positive, negative, either, both?**
- **The built-in string parsing program - NOT!**
- **VARCHAR2 memory allocation**
- **VARRAY - what kind of varrying is that?**
- **\$IF I understood \$IF I would be amazing!**
- **The mystery of "reserved words" in PL/SQL**
- **Where's my line number?**
- **Where's my program name?**
- **Zero length strings and NULL**
- **%ROWTYPE and %TYPE**

C'mon, PL/SQL is *not* WEIRD!

- **That's true. It really isn't a weird language.**
 - Anybody ever use LISP?

- **In fact, it is a very enjoyable language.**
 - It's a very powerful and expressive language.
 - It's a very easy to learn language.
 - It's a very fun language.

THANK YOU PL/SQL DEV TEAM!

I do it out of the deepest respect and appreciation - and desire to make PL/SQL even better!

PL/SQL - the *Really* Readable Language

- **Let's compare Java and PL/SQL....**

```
public class Hello {  
    public static void main (String [] args) {  
        System.out.println ("Hello World!");  
    }  
}
```

```
CREATE OR REPLACE PROCEDURE hello (  
IS  
BEGIN  
    DBMS_OUTPUT.PUT_LINE ('Hello world!');  
END hello;
```

- **PL/SQL isn't as powerful as Java, but it's *much* easier to write - and understand!**

SQLERRM and DBMS_OUTPUT.PUT_LINE

- **SQLERRM** returns the error text for the specified error code.
 - If none is provided, then SQLCODE is assumed.
- **But SQLERRM might *truncate your error message!***
 - Currently at 512, earlier at 255.
- **255...255...what does that number remind us of! Ah, yes: DBMS_OUTPUT.PUT_LINE**
- **So use DBMS_UTILITY.FORMAT_ERROR_STACK instead!**

sqlerrm.sql

Error codes: positive, negative, either, both?

- **We all know that Oracle errors are negative.**
- **Well, except for 1 and 100.**
 - Hey, that `NO_DATA_FOUND` is pretty strange all by itself!
- **Oh, and error codes saved by `SAVE EXCEPTIONS`.**
- **And error codes saved by `DBMS_ERRLOG`**
- **At least the `EXCEPTION_INIT` pragma has it straight - more or less.**

STANDARD package in `RDBMS\Admin:`
`stdspec.sql` `stbody.sql`

`bulkexc.sql` `dbms_errlog*.*` `exception_init.sql`



More on errors: the price of laziness - or is it cost-cutting?

- **It's as if someone at Oracle calculated that declaring new exceptions cost \$74 each.**
 - Larry: "Save \$1B in costs this year!"
 - Developers: "OK, we won't declare new exceptions."
- **And so we have "recycled" exceptions:**
 - NO_DATA_FOUND: pure confusion
 - VALUE_ERROR: sub-message text

excquiz6*.sql
value_error_demo.sql

The built-in string parsing program - NOT!

- Surely any robust programming language provides a program to parse a delimited string.
 - And PL/SQL is no exception....
- Well, sort of. Oracle gave us:

DBMS_UTILITY.COMMA_TO_TABLE

- String cannot be NULL - raises error!
- Only parses comma-delimited strings.
- Every element in the list must be a *valid PL/SQL identifier*.
 - You can't make this stuff up!

commatortable.tst
sqlerrm.sql

Parsing programs to the rescue

- **Well, the main thing is that *you* should not have to write these algorithms over and over yourself.**
- **So here are some options:**
 - parse.pkg - parse any delimited string, returns a collection of type defined in package.
 - str2list.pkg - parse any delimited string, returns a collection based *on your own type*.
 - Regular expressions

parse.pkg
str2list.pkg
str2list2.pkg

VARCHAR2 memory allocation

- Memory for VARCHAR2 variables is only allocated as needed, right?
 - *Variable-length strings!*
- OK, then which of the following variables consume the most memory?

```
DECLARE
  l_var1  VARCHAR2 (32767);
  l_var2  VARCHAR2 (4050);
  l_var3  VARCHAR2 (500);
BEGIN
  l_var1 := 'a';
  l_var2 := 'a';
  l_var3 := 'a';
END;
/
```

That's right!

l_var3

VARRAY - what kind of varrying is that?

- **Oracle offers three kinds of collections**
 - Associative Array
 - Nested table
 - Varray, a.k.a, "Varrying array"
- **That's weird enough, all by itself, really.**
- **But what I find truly odd is the use of the name "varrying array".**
- **It is the *least varrying* of all the different types of collections!**

associative_array_example.sql
nested_table_example.sql
varray_example.sql
collection_of_records.sql

\$IF I understood \$IF I would be amazing!

- Have you ever used the \$ syntax added to PL/SQL in Oracle10g Release 2?
- It looks *very weird*.
- But it is *very useful*.
- **With conditional compilation, you can....**
 - Write code that will compile and run under different versions of Oracle (relevant for future releases).
 - Run different code for test, debug and production phases. That is, compile debug statements in and out of your code.
 - Expose private modules for unit testing.

cc_version_check.sql
cc_debug_trace.sql
11g_emplu.pkg

The mystery of "reserved words" in PL/SQL

- **What will happen when I run this code?**

```
DECLARE
  PLS_INTEGER    VARCHAR2 (1);
  NO_DATA_FOUND  EXCEPTION;
BEGIN
  SELECT dummy
  INTO PLS_INTEGER
  FROM DUAL
  WHERE 1 = 2;

  IF PLS_INTEGER IS NULL
  THEN
    RAISE NO_DATA_FOUND;
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND
  THEN
    DBMS_OUTPUT.put_line (' No dummy! ');
END;
```

excquiz5.sql



Exploring the STANDARD package

- **Much of what we consider to be the base PL/SQL language is defined in the STANDARD package.**
 - One of two "default" packages; the other is DBMS_STANDARD.
- **You can view the contents of the STANDARD package (and other "supplied" packages by visiting the appropriate variation of:**

`$ORACLE_HOME/Rdbms/Admin`

Where's my line number?

- An exception is raised.
- I trap it and log the error.
- I want to see the line number on which the error is raised.
- If I am not yet on Oracle 10g Release 2, I am *totally out of luck*.
- Keep up with Oracle, though, and then you can take advantage of **DBMS_UTILITY.FORMAT_ERROR_BACKTRACE**
 - Call it *every time* you log an error!

Backtrace.sql
Quest Error Manager -
www.ToadWorld.com - Downloads



Where's my program name?

- **DBMS_UTILITY.FORMAT_CALL_STACK**
answers the question: "How did I get here?"
- **DBMS_UTILITY.FORMAT_ERROR_BACKTRACE**
answers the question: "Where did my error come from?"
- **But in both cases....if the program in question is located inside a package, I don't see the name of that subprogram.**
- **Please, Oracle, give me my program name!**
 - Heck, and my overloading, while you are at it.

Zero length strings and NULL

- A NULL string and a zero length string (") are supposed to be treated the same way inside Oracle.
 - And for the most part it is true.
- But when it comes to

ORA-06502: PL/SQL: numeric or value error: NULL
index table key value

- You can get some rather *weird* behavior!
 - Especially on Oracle11g....

null_index_value.sql
zero_length_string.sql



%TYPE *and* %ROWTYPE?

- **I have enormous respect for the PL/SQL team.**
 - Not only are they all smarter than me, they are *real* computer scientists. I am an amateur by comparison.
- **And their design of the PL/SQL syntax is for the most part very elegant and minimalist.**
- **But why do we have to have both %TYPE *and* %ROWTYPE?**



So I hope you now agree with me...

- PL/SQL can be a little bit weird sometimes.
- But it is still a truly **elegant and straightforward** language.
- Those features make it, in turn, **accessible and productive**.
- And that's why so many of us have achieved ***personal success***, and have implemented so many ***successful applications*** with the PL/SQL language.