C

PL/SQL Program Limits

This appendix describes the program limits that are imposed by the PL/SQL language. PL/SQL is based on the programming language Ada. As a result, PL/SQL uses a variant of Descriptive Intermediate Attributed Notation for Ada (DIANA), a tree-structured intermediate language. It is defined using a metanotation called Interface Definition Language (IDL). DIANA is used internally by compilers and other tools.

At compile time, PL/SQL source text is translated into system code. Both the DIANA and system code for a subprogram or package are stored in the database. At run time, they are loaded into the shared memory pool. The DIANA is used to compile dependent subprograms; the system code simply runs.

In the shared memory pool, a package specification, ADT specification, standalone subprogram, or anonymous block is limited to 67108864 (2**26) DIANA nodes which correspond to tokens such as identifiers, keywords, operators, and so on. This allows for ~6,000,000 lines of code unless you exceed limits imposed by the PL/SQL compiler, some of which are given in Table C-1.

Table C-1 PL/SQL Compiler Limits

Item	Limit
bind variables passed to a program unit	32768
exception handlers in a program unit	65536
fields in a record	65536
levels of block nesting	255
levels of record nesting	32
levels of subquery nesting	254
levels of label nesting	98
levels of nested collections	no predefined limit
magnitude of a `PLS_INTEGER` or `BINARY_INTEGER` value	-2147483648..2147483647
number of formal parameters in an explicit cursor, function, or procedure	65536
objects referenced by a program unit	65536
precision of a `FLOAT` value (binary digits)	126
precision of a `NUMBER` value (decimal digits)	38
precision of a `REAL` value (binary digits)	63
size of an identifier (bytes)	128
size of a string literal (bytes)	32767
size of a `CHAR` value (bytes)	32767
size of a `LONG` value (bytes)	32760
size of a `LONG RAW` value (bytes)	32760

Table C-1 (Cont.) PL/SQL Compiler Limits

Item	Limit
size of a RAW value (bytes)	32767
size of a VARCHAR2 value (bytes)	32767
size of an NCHAR value (bytes)	32767
size of an NVARCHAR2 value (bytes)	32767
size of a BFILE value (bytes)	4G * value of DB_BLOCK_SIZE parameter
size of a BLOB value (bytes)	4G * value of DB_BLOCK_SIZE parameter
size of a CLOB value (bytes)	4G * value of DB_BLOCK_SIZE parameter
size of an NCLOB value (bytes)	4G * value of DB_BLOCK_SIZE parameter
size of a trigger	32 K

To estimate how much memory a program unit requires, you can query the static data dictionary view USER_OBJECT_SIZE. The column PARSED_SIZE returns the size (in bytes) of the "flattened" DIANA. For example:

```
CREATE OR REPLACE PACKAGE pkg1 AS
  TYPE numset_t IS TABLE OF NUMBER;
  FUNCTION f1(x NUMBER) RETURN numset_t PIPELINED;
END pkg1;
/

CREATE OR REPLACE PACKAGE BODY pkg1 AS
  -- FUNCTION f1 returns a collection of elements (1,2,3,... x)
  FUNCTION f1(x NUMBER) RETURN numset_t PIPELINED IS
  BEGIN
    FOR i IN 1..x LOOP
      PIPE ROW(i);
    END LOOP;
    RETURN;
  END f1;
END pkg1;
/
```

SQL*Plus commands for formatting results of next query:

```
COLUMN name FORMAT A4
COLUMN type FORMAT A12
COLUMN source_size FORMAT 999
COLUMN parsed_size FORMAT 999
COLUMN code_size FORMAT 999
COLUMN error_size FORMAT 999
```

Query:

```
SELECT * FROM user_object_size WHERE name = 'PKG1' ORDER BY type;
```

Result:

```
NAME TYPE         SOURCE_SIZE PARSED_SIZE CODE_SIZE ERROR_SIZE
---- ------------ ----------- ----------- --------- ----------
PKG1 PACKAGE              112         498       310          79
PKG1 PACKAGE BODY         233         106       334           0
```

Unfortunately, you cannot estimate the number of DIANA nodes from the parsed size. Two program units with the same parsed size might require 1500 and 2000 DIANA nodes, respectively because, for example, the second unit contains more complex SQL statements.

When a PL/SQL block, subprogram, package, or schema-level user-defined type exceeds a size limit, you get an error such as `PLS-00123: program too large`. Typically, this problem occurs with packages or anonymous blocks. With a package, the best solution is to divide it into smaller packages. With an anonymous block, the best solution is to redefine it as a group of subprograms, which can be stored in the database.

For more information about the limits on data types, see PL/SQL Data Types.