Examples and limits of BYTE and CHAR semantics usage (NLS_LENGTH_SEMANTICS) [ID 144808.1]

Modified: Jan 30, 2012 Type: BULLETIN Status: PUBLISHED Priority: 3

In this Document

Applies to:

Oracle Server - Enterprise Edition - Version: 9.0.1.0 to 11.2.0.3 - Release: 9.0.1 to 11.2
Information in this document applies to any platform.

Purpose

To document limits and best practices when implementing NLS_LENGTH_SEMANTICS.

Scope and Application

Any DBA wanting to use NLS_LENGTH_SEMANTICS.

NLS_LENGTH_SEMANTICS allows you to specify the length of a column datatype in terms of CHARacters rather than in terms of BYTEs. Typically this is when using an AL32UTF8 or other varying width NLS_CHARACTERSET database where one character is not always one byte. While using CHAR semantics has as such no added value in a 7/8 bit characterset it's fully supported so any application code / table setup using CHAR can also be used in a 7/8bit characterset like US7ASCII/WE8MSWIN1252.
This parameter is a 9i (and up) feature and is not available in older releases.

Make sure to check also "Things to know when using CHAR semantics:" as there are certain limitations on when you can use this.

Examples and limits of BYTE and CHAR semantics usage (NLS_LENGTH_SEMANTICS)

A. Setting NLS_LENGTH_SEMANTICS to CHAR

* Use explicit CHAR semantics in SQL and PL/SQL syntax

> For example use CHAR(10 CHAR) or VARCHAR2(10 CHAR) in the columns definition when creating tables or use CHAR(10 CHAR) or VARCHAR2(10 CHAR) in PL/SQL variable definition.
> A VARCHAR2(10 CHAR) will always use CHAR semantics , no matter what setting is used in the session or instance.

* NLS_LENGTH_SEMANTICS can also be set in a current session

```
SQL>ALTER SESSION SET NLS_LENGTH_SEMANTICS=CHAR
```

The NLS_SESSION_PARAMETERS for this session will change and it will be active for that session only.

The session NLS_LENGTH_SEMANTICS parameter setting influence create/alter column statements and pl/sql variables who do not explicitly define the semantic to be used.
This is when using CHAR(10) or VARCHAR2(10) syntax , without specifying explicitly CHAR or BYTE.

CHAR(10 CHAR) or VARCHAR2(10 CHAR) will always use CHAR semantics , no matter what setting is used in the session.

Oracle advices to use explicit CHAR semantics in the SQL or PL/SQL syntax
OR
To make sure your application does an "ALTER SESSION SET NLS_LENGTH_SEMANTICS=CHAR" when connecting if CHAR semantics is required but the semantic is not defined explicit in SQL.

* NLS_LENGTH_SEMANTICS cannot be set as a client side environment/registry parameter in 9i, from 10g onwards NLS_LENGTH_SEMANTICS can be set (please define it in UPPERCASE). If NLS_LENGTH_SEMANTICS is set at client side then any session started from that client will use the value defined in the environment/registry and it can be checked in NLS_SESSION_PARAMETERS.

* If NLS_LENGTH_SEMANTICS is not set at client side or no alter session is done then the session will use the value found in NLS_INSTANCE_PARAMETERS.

* The NLS_LENGTH_SEMANTICS parameter can be set at instance level in the init.ora or Spfile. You will then see the parameter change in NLS_INSTANCE_PARAMETERS.

```
SQL>ALTER SYSTEM SET NLS_LENGTH_SEMANTICS=CHAR scope=spfile;
SQL>-- and restart the instance
```

Note :
* Oracle advices to use explicit CHAR semantics in the SQL or PL/SQL syntax or to make sure your application does an alter session when connecting if CHAR semantics is required and the semantic is not defined explicit in SQL.
* A lot of people think simply setting NLS_LENGTH_SEMANTICS=CHAR in the spfile/init.ora is the only thing to do to change an existing db to NLS_LENGTH_SEMANTICS=CHAR, this is not true.
* Oracle recommends to NOT set the NLS_LENGTH_SEMANTICS parameter to CHAR in the instance or server parameter file when possible.

The session or instance value will only be used when creating NEW tables / columns.
Setting NLS_LENGTH_SEMANTICS to CHAR in a session will NOT adapt / change current existing table column definitions.
In other words, if you have tables with columns who now use BYTE and you change the session or instance parameter to CHAR then those columns will still be BYTE (or inverse).
To change **existing** tables you need to use "alter table". See further down for more information on this.

For more info about the difference between NLS_INSTANCE_PARAMETERS, NLS_DATABASE_PARAMETERS,... see Note 241047.1 The Priority of NLS Parameters Explained.

* The NLS_LENGTH_SEMANTICS value found in NLS_DATABASE_PARAMETERS is the value at database creation time.
The NLS_LENGTH_SEMANTICS setting in NLS_DATABASE_PARAMETERS is not relevant when using NLS_LENGTH_SEMANTICS=CHAR in the NLS_SESSION_PARAMETERS or NLS_INSTANCE_PARAMETERS (init.ora/Spfile).

* It's good practice to avoid using a mixture of BYTE and CHAR semantics columns in the same table.

Things to know when using CHAR semantics:

* CHAR semantics is NOT supported by the Oracle E-Business Suite (yet).

* An ALTER SYSTEM SET NLS_LENGTH_SEMANTICS=CHAR scope=both; needs a restart of the database before it takes effect.

*For Database version lower then 11g do NOT set the NLS_LENGTH_SEMANTICS=CHAR during database creation. Create the database with NLS_LENGTH_SEMANTICS=BYTE (or not set) and set NLS_LENGTH_SEMANTICS=CHAR after the database creation in the init.ora /spfile.
If NLS_LENGTH_SEMANTICS=CHAR was set during database creation then some XDB and SYS objects will then be created using CHAR semantics, which is not supported.
All known issues with this are solved in 11.1.0.6 (and up) (see at the end of this note Bug 5545716 and Bug 4886376).
Note that a database created using BYTE and is using CHAR in the init.ora/spfile is functionally just the same as a database that was created using CHAR and is using CHAR in the init.ora/spfile.
If you run patch scripts or scripts from $ORACLE_HOME/RDBMS/ADMIN like catalog.sql use STARTUP MIGRATE; and run then the scripts. (run them with NLS_LENGTH_SEMANTICS=BYTE)

* Oracle Text does NOT support instance wide NLS_LENGTH_SEMANTICS=CHAR If you are using Oracle Text then the database needs to be created and started with NLS_LENGTH_SEMANTICS=BYTE.
Using CHAR semantics on column definitions itself is supported however.Bug 4465964
This is not documented in the 10.2 docset, it is in the 11g doc: http://download.oracle.com/docs/cd/B28359_01/text.111/b28304/csql.htm#i997737 section "CHARSET COLUMN charset_column_name"

* There is no problem using CHAR semantics in a 8 or 7 bit characterset. You can perfectly define a column as VARCHAR2(20 CHAR) in an US7ASCII or WE8MSWIN1252 database.

* Clients (or servers) older then 9i (8i, 80..) will not see CHAR semantics they will get the length returned in BYTE. A 9i UTF8 database with a VARCHAR2(10 CHAR) column will show up in a 8 client as VARCHAR2(30) - meaning 30 BYTES. This allows the 8i or older client to insert more then 10 characters if those characters are using less then the 30 bytes (the letter 'a' for example). So make sure you are using 9i (or up) clients with CHAR semantics.

For 10.1 you can use the init.ora parameter DB_ALLOWED_LOGON_VERSION=9
For 10.2 and up set at the server side sqlnet.ora: SQLNET.ALLOWED_LOGON_VERSION = 9
(note that DB_ALLOWED_LOGON_VERSION is not to be used in 10.2 and up anymore)
The above parameter setting allows 9i, 10g and 11 versions to access the DB. Version 8i and lower would fail with 'ORA-28040: No matching authentication protocol'.

B. Defaults and limits.

The default setting for NLS_LENGTH_SEMANTICS is BYTE and the default sizing of character data types (CHAR, VARCHAR2) is in BYTES .

For example, if NLS_LENGTH_SEMANTICS is not set or set to BYTE then CHAR(10) in a table definition means 10 bytes not 10 characters.

If NLS_LENGTH_SEMANTICS is set to CHAR then using CHAR(10) will create a column with a 10 CHAR width.

It also possible to explicit define the BYTE or CHAR semantics when creating a column:

CHAR(10 BYTE) - will always be BYTE regardless of the used NLS_LENGTH_SEMANTICS
CHAR(10 CHAR) - will always be CHAR regardless of the used NLS_LENGTH_SEMANTICS

```
SQL> ALTER SESSION SET NLS_LENGTH_SEMANTICS=BYTE;

Session altered.

SQL> Create table scott.test1 (Col1 CHAR(20),Col2 VARCHAR2(100));

Table created.

SQL> Create table scott.test2 (Col1 CHAR(20 CHAR),Col2 VARCHAR2(100 CHAR));

Table created.

SQL> Create table scott.test3 (Col1 CHAR(20 BYTE),Col2 VARCHAR2(100 BYTE));

Table created.

SQL> desc scott.test1
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL1                                               CHAR(20)
 COL2                                               VARCHAR2(100)

SQL> desc scott.test2
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL1                                               CHAR(20 CHAR)
```

```
 COL2                                                VARCHAR2(100 CHAR)

SQL> desc scott.test3
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL1                                                CHAR(20)
 COL2                                                VARCHAR2(100)

SQL> ALTER SESSION SET NLS_LENGTH_SEMANTICS=CHAR;

Session altered.

SQL> desc scott.test1
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL1                                                CHAR(20 BYTE)
 COL2                                                VARCHAR2(100 BYTE)

SQL> desc scott.test2
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL1                                                CHAR(20)
 COL2                                                VARCHAR2(100)

SQL> desc scott.test3
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL1                                                CHAR(20 BYTE)
 COL2                                                VARCHAR2(100 BYTE)

SQL> Create table scott.test4 (Col1 CHAR(20),Col2 VARCHAR2(100));

Table created.

SQL> desc scott.test4
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL1                                                CHAR(20)
 COL2                                                VARCHAR2(100)

SQL> ALTER SESSION SET NLS_LENGTH_SEMANTICS=BYTE;

Session altered.

SQL> desc scott.test4
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL1                                                CHAR(20 CHAR)
 COL2                                                VARCHAR2(100 CHAR)

SQL>
```

Note the dependency of the output of the DESC output on the sessions NLS_LENGTH_SEMANTICS setting.
There is also Bug 9533867 - desribe command does not properly handles nls_length_semantics
When using NLS_LENGTH_SEMANTICS in the spfile/init.ora the DECS command might not give the correct result.
doing an alter session will then give above output.
Alternativly the CHAR_USED column from DBA_TAB_COLUMNS can also be used to to see if a column is created using BYTE or CHAR semantics, it then contains B or C regardless of the used NLS_LENGTH_SEMANTICS session setting.
Note that the problem is pure with the display of DESC , not with the object (creation).

This select will give all columns using CHAR semantics:

```
SELECT C.owner
||'.'
|| C.table_name
||'.'
|| C.column_name
||' ('
|| C.data_type
||' '
|| C.char_length
||' CHAR)'
FROM all_tab_columns C
WHERE C.char_used = 'C'
AND C.table_name NOT IN
(SELECT table_name FROM all_external_tables
)
AND C.data_type IN ('VARCHAR2', 'CHAR')
ORDER BY 1
/
```

When using CHAR semantics Oracle advices to define explicit the CHAR semantics when creating tables.

```
Create table scott.test (Col1 CHAR(20 CHAR),Col2 VARCHAR2(100 CHAR));
```

For a single-byte character set encoding (WE8ISO8859P1, US7ASCII etc..), the character and byte length are the same (one character = one byte).
However, in multi-byte character set encodings like AL32UTF8 the character may use up to 4 bytes to store one character making sizing the column length more difficult.

Hence the reason why CHAR semantics was introduced. However, we still have some physical underlying byte based limits and development has chosen to allow the full usage of the underlying limits. This results in the following table giving the maximum amount of CHARacters occupying the MAX data length that can be stored for a certain datatype in 9i and up.

The table below gives a overview of the limits of CHAR semantics.

All numbers are CHAR definitions

```
              UTF8  |      AL32UTF8 |  AL16UTF16
     (1 to 3 bytes)| (1 to 4 bytes)|  (2 bytes)
          MIN  MAX |      MIN  MAX |  MIN   MAX
-----------------------------------------------
CHAR      2000  666 |     2000  500 |  N/A   N/A
VARCHAR2  4000 1333 |     4000 1000 |  N/A   N/A
-----------------------------------------------
NCHAR     2000  666 |      N/A  N/A | 1000  1000
NVARCHAR2 4000 1333 |      N/A  N/A | 2000  2000
```

(N/A means not possible)

The MIN col is the maximum size that you can *define* and that Oracle can store if all data is the MINIMUM data length (1 byte for UTF8) for that characterset.

For example a VARCHAR2(4000 CHAR) in UTF8 can store 4000 times an 'a' since 'a' is only one byte in AL32UTF8.

The MAX column is the MAXIMUM amount of CHARACTERS that can be stored occupying the MAXIMUM data length. Seen that UTF8 and AL32UTF8 are VARYING charactersets this means that a string of X chars can be X to X*3 for UTF8 (or X*4 for AL32UTF8) bytes.

For example a VARCHAR2(4000 CHAR) in UTF8 can store only 1333 x the Euro symbol seen the Euro symbol is 3 bytes in UTF8 (and in AL32UTF8). You may have defined the column as 4000 CHAR, but the "limit behind the scene" is 4000 BYTES. 4000 dived by 3 is 1333...

So it does not make sense to declare a column length bigger then VARCHAR2(1333 CHAR) in UTF8.

Same for CHAR: if you try to store more then 666 characters that occupy 3 bytes in UTF8 in a CHAR UTF8 colum you still will get a ORA-01401: inserted value too large for column (or from 10g onwards: ORA-12899: value too large for column) even if you have defined the colum as CHAR (2000 CHAR). Because the MAX *BYTES* is 2000 and that divided by 3 is 666.

N-types (NVARCHAR2, NCHAR) are *always* defined in CHAR semantics, you cannot define them in BYTE. Note 276914.1 The National Character Set in Oracle 9i and 10g

You can easily find columns who define more then the MAX limit with a select like

* for CHAR:

```
SELECT OWNER,
TABLE_NAME,
COLUMN_NAME,
DATA_LENGTH
FROM dba_tab_columns
WHERE DATA_TYPE = 'CHAR'
AND DATA_LENGTH > 666
AND OWNER NOT IN
('SYS','SYSTEM','WKSYS','LBACSYS','XDB','CTXSYS','OUTLN','MDSYS','WMSYS',
'ORDSYS','OLAPSYS','SH','OE')
/
```

It might be a good idea to define the found columns as VARCHAR2 or CLOB's.

* for VARCHAR2:

```
SELECT OWNER,
TABLE_NAME,
COLUMN_NAME,
DATA_LENGTH
FROM dba_tab_columns
WHERE DATA_TYPE = 'VARCHAR2'
AND DATA_LENGTH > 1333
AND OWNER NOT IN
('SYS','SYSTEM','WKSYS','LBACSYS','XDB','CTXSYS','OUTLN','MDSYS','WMSYS',
'ORDSYS','OLAPSYS','SH','OE')
/
```

It might be a good idea to define the found columns as CLOB's.

When using AL32UTF8 you can *also* safely assume that almost all data will use max 3 bytes, 4 bytes are only used for surrogate pairs. The amount of characters that use surrogate pairs is still very small, mostly highly specialized characters or characters used in "dead" languages.

For information about surrogate pairs please see for example http://unicode.org/versions/Unicode4.0.0/ch15.pdf , point 15.5

The only "real world" exception using surrogate pairs are (Windows) clients using HKSCS 2001/2004 charactersets. Note 787371.1 Oracle Database Server support for HKCSC 1999, 2001 and 2004 character sets.

In Oracle, UTF8 does not know the concept of surrogate pairs, but you actually can *store* Surrogate Pairs in UTF8. That character will be stored as 2 * 3 byte character and not like in AL32UTF8 using one 4 byte character. In AL16UTF16 one surrogate pair will use 2 UTF16 codepoints and so occupy 4 bytes.
One surrogate pair will in UTF8 be counted as 2 characters. In AL32UTF8 and AL16UTF16 one surrogate pair will be counted as one character.

So to avoid ORA-01401 / ORA-12899 errors, even with CHAR semantics, using CHAR columns smaller then 666 CHAR and VARCHAR2 columns smaller then 1333 CHAR is the best practice in UTF8/AL32UTF8.

To know for a current data set that will be migrate to a multi byte characterset like UTF8/AL32UTF8 if there is data that will give ORA-01401 / ORA-12899 errors during export/import use Csscan. That data will be logged as "Truncation" in the csscan report. See Note:444701.1 Csscan output explained section B.3.

C. NLS_LENGTH_SEMANTICS has no effect on tables owned by SYS.

This is normal, for sys objects NLS_LENGTH_SEMANTICS is ignored and the these are always treated with byte semantics. Note that it is possible to create a table with explicit CHAR semantics under the sys schema, this is however not supported and should be avoided.

```
SQL> ALTER SESSION SET NLS_LENGTH_SEMANTICS=CHAR;

Session altered.

SQL> Create table sys.test1 (Col1 CHAR(20),Col2 VARCHAR2(100));

Table created.

SQL> Create table sys.test2 (Col1 CHAR(20 CHAR),Col2 VARCHAR2(100 CHAR));

Table created.

SQL> desc test1
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL1                                               CHAR(20 BYTE)
 COL2                                               VARCHAR2(100 BYTE)

SQL> desc test2
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL1                                               CHAR(20)
 COL2                                               VARCHAR2(100)
```

SYSTEM objects are affected by NLS_LENGTH_SEMANTICS, the docset of older Oracle versions is not correct on that.

D. Anonymous blocks will NOT pick up changed NLS_LENGTH_SEMANTICS parameter.

The NLS_LENGTH_SEMANTICS session value active at the time of CREATE PROCEDURE or CREATE PACKAGE statement execution is stored together with the created PL/SQL unit and used for all following implicit recompilations, caused by modification of referenced objects.
An explicit CREATE OR REPLACE or ALTER ... COMPILE statement rereads the session setting (NLS_SESSION_PARAMETERS (!)), uses the value for defining variables, and stores it with the recompiled object.
On the other hand, the ALTER statement may contain the REUSE SETTINGS clause or an explicit specification of NLS_LENGTH_SEMANTICS value, which again override the session parameter.

To be more clear, you can define the same variable in 3 ways:

```
x CHAR(3 BYTE);
x CHAR(3 CHAR);
x CHAR(3);
```

in the first 2 there is no confusion possible as you are explicit defining the used semantics for the variable itself. When declaring/creating the procedure or package however using x CHAR(3) when NLS_LENGTH_SEMANTICS=BYTE then the semantic will stay BYTE based, even after the session is changed to NLS_LENGTH_SEMANTICS = CHAR, until the blocks are recompiled, only then the package or procedure will pick up the changed NLS_LENGTH_SEMANTICS setting.

So without recompile the x CHAR(3) will "act" like x CHAR(3 BYTE) even after the session (!) has been changed to use NLS_LENGTH_SEMANTICS = CHAR. Only after a recompile the x CHAR(3) will "act" like x CHAR(3 CHAR). Double check when recompiling that you see in the NLS_SESSION_PARAMETERS table NLS_LENGTH_SEMANTICS = CHAR.

You can also check the used NLS_LENGTH_SEMANTICS session setting for any PLS/SQL object in the NLS_LENGTH_SEMANTICS column of

DBA_PLSQL_OBJECT_SETTINGS.

```
SELECT C.owner
||'.'
|| C.name
||' - '
|| C.type
FROM DBA_PLSQL_OBJECT_SETTINGS C
WHERE C.NLS_LENGTH_SEMANTICS = 'CHAR'
-- to have a list of all using BYTE semantics use
-- WHERE C.NLS_LENGTH_SEMANTICS = 'BYTE'
-- to check only for a certain users use
-- and C.owner IN ('SCOTT')
ORDER BY C.owner,C.name
/
```

Seen it's rather hard to control the environment settings for 100% we suggest to use explicit CHAR semantics in your code.

```
x CHAR (3 CHAR);
y VARCHAR2 (10 CHAR);
```

This is really something to consider, even if this means updating source code .
Consider using CHAR semantics on tables and you re-create the packages who are using y VARCHAR2 (10); (= no explicit CHAR semantics) using a session using NLS_LENGTH_SEMANTICS = CHAR. Everything is tested and works.
Then a few days/months later for some reason the package is update and re-created using a session using NLS_LENGTH_SEMANTICS = BYTE. Suddenly you will see errors like " ORA-06502 PL/SQL Numeric or value error " on *some* records coming from a VARCHAR2(10 CHAR) column .
What is happening is that as long as the row contains only US7ASCII (= 1 byte characters) the data will "fit" in the package variable (the variable will be acting as VARCHAR2(10 BYTE)), when the row contains non-US7ASCII characters it will not be possible to store the row content in the package variable seen it will be more bytes than the variable is declared to be and lead to erros like ORA-06502 PL/SQL Numeric or value error .

E. How to go to CHAR semantics for existing tables?

E.1) Using exp/imp or expdp/impdp.

The import utility uses the exact semantics defined on the **source** table defintion from the export dump file, not from the source or target database init.ora settings. If the source tables are defined with BYTE semantics then they will be created with BYTE semantics , even when importing into a database that has NLS_LENGTH_SEMANTICS set to CHAR in the init.ora/pfile/spfile.
In other words, setting NLS_LENGTH_SEMANTICS=CHAR in your init.ora will NOT change any table to CHAR semantics during import if those tables (!!!) are using BYTE on the soure side.

It is very likely to see ORA-01401 / ORA-12899 errors during the import from a single byte characterset to an Unicode database.
If you want to migrate tables from "no semantics defined" (=BYTE) to CHAR semantics using export/import to avoid ORA-01401 / ORA-12899 errors during the import you can do 3 things:

* Change the tables explicitly to CHAR before taking the export using alter table (-> point E.2)

* From the export, extract the create table statements with imp show=Y (Note 29765.1) and pre-create the tables on the target side using a session that has NLS_LENGTH_SEMANTICS set to CHAR. This will create any column that has no explicit semantics defined (= CHAR(20)) with CHAR semantics. When using DataPump the impdpd equivalent of show=Y is the SQLFILE option.
Then import the data using the IGNORE=Y parameter for imp or the TABLE_EXISTS_ACTION=TRUNCATE option for Impdp to import the data into the pre-created tables.

* Get the Create table statement in another way and pre-create the tables on the target side with the explicit CHAR semantics in the column definitions (=CHAR (20 CHAR)) or, if the create table has no explicit BYTE or CHAR defintion, use a session that has NLS_LENGTH_SEMANTICS set to CHAR.
To create this file one can for example use SQldveloper, this can be download it from http://www.oracle.com/technology/products/database

/sql_developer/
It has a "database export" in the Tools menu, choose the connection and specify the output file, select "show schema" , click next , in the next screen deselect all except "tables" , click next , select the user(s) and tables, click next and create the file by clicking on "Finish".
After creating the table in the new database import the data using the IGNORE=Y parameter for imp or the TABLE_EXISTS_ACTION=TRUNCATE option for Impdp to import the data into the pre-created tables.

When using "Alter database character set" or Csalter with partial exp/imp approach the columns are typically changed to CHAR semantics using Alter table modify (-> point E.2) after the change to AL32UTF8 but before importing the exported "Convertible" data again.

If there are no ORA-01401 / ORA-12899 during the import but you want to change the tables to CHAR semantics in an other / new database then you can simply change the tables explicitly to CHAR after the import using alter table (-> point E.2).

There is an enhancement request to add to datapump a "feature/parameter" to change the semantics using expdp/impdp Bug 5034478 ENH: ADD PARAMETER TO IMPDP TO OVERWRITE NLS_LENGTH_SEMANTICS

E.2) Use alter table modify.

You can make a script that alters all the column definitions of your table.

The following note has an example of a script that updates all columns from all tables: Note 313175.1 Changing columns to CHAR length semantics

If you have a table containing a CHAR(10) col then you can

```
alter table "<owner>"."<table>" modify "<column>" char (10 char);
```

If there are indexes on CHAR columns and you change them to use CHAR semantics then it might be faster to drop the index, do that alter table and then re-create the index. the reason is simply that all key values of a CHAR/NCHAR index key have to be padded with blanks to the new length and it may be more efficient to drop and recreate the index.

If you have a table containing a VARCHAR2(200) col then you can

```
alter table "<owner>"."<table>" modify "<column>" varchar2 (200 char);
```

Note that there are some restrictions with alter table modify, consult the SQL Reference Guide for more info.

F. Can i use CHAR semantics for TYPES?

Yes, the attributes of a type can also be defined as having either CHAR or BYTE length semantics.
More info can be found in Note 274507.1 Finding out the length semantics of type attributes

If you intend to change types from BYTE to CHAR please be aware of Note 274514.1 ORA-22347 or PLS-719 attempting to change length semantics of type attribute and note that currently an "alter type X add/drop" does not implicitly uses "reuse settings". So your session settings will be used.

G. Summary of best practices:

* Check for columns who are > 666 bytes for CHAR datatype and >1333 bytes for VARCHAR2 datatype. Consider to use CLOB for VARCHAR2 or VARCHAR2 for CHAR columns -or- make sure your application layer limits the input string -or- make at least sure your application layer handles any ORA-01401 / ORA-12899 exception gracefully.
* If you transfer data using database links from other databases strongly consider to use CHAR semantics for column definitions on the remote database, even if the remote database is using a 7/8 bit NLS_CHARACTERSET to avoid issues like note 269381.1.
* Use explicit CHAR semantic definitions in PL/Sql procedures, Type definitions etc.
* Use explicit CHAR semantic definitions in create table statements.
* Change all existing table columns to CHAR semantics using one of the 2 options in point E in this note.
* It's a good idea to **not** use a mixture of BYTE and CHAR semantics in the same table(s) or schema to avoid confusion.
* _keep_remote_column_size=true is NOT compatible with using CHAR semantics. Do not use this parameter when using CHAR semantics.

H. More Examples

*** Test with WE8MSWIN1252 character set and AL16UTF16 national character set

```
SQL> select * from v$nls_parameters where parameter like '%CHARACTERSET%';

PARAMETER VALUE
------------------------------ --------------------
NLS_CHARACTERSET WE8MSWIN1252
NLS_NCHAR_CHARACTERSET AL16UTF16

SQL> create table essai
2 ( col_byte CHAR(4),
3 col_char NCHAR(4));

SQL> select COLUMN_NAME, DATA_TYPE,DATA_LENGTH, CHAR_LENGTH,CHAR_USED
2 from dba_tab_columns
3 where table_name='ESSAI';

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH CHAR_USED
------------- ---------- ----------- ----------- ---------
COL_BYTE CHAR 4 4 B
COL_CHAR NCHAR 8 4 C
```

The AL16UTF16 character set is a 2byte character set. Therefore each character is always stored in 2 bytes.

*** NCHAR and NVARCHAR2 are always character semantics:

```
SQL> alter session set nls_length_semantics='BYTE';
Session altered.

SQL> alter table essai add col3 NCHAR(4);
Table altered.

SQL> @sel

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH CHAR_USED
------------- ---------- ----------- ----------- ---------
COL_BYTE CHAR 4 4 B
COL_CHAR NCHAR 8 4 C
COL3 NCHAR 8 4 C
```

*** For single-byte character sets, BYTE or CHAR does not make any difference
*** since 1 character is stored in 1 byte

```
SQL> alter session set nls_length_semantics='CHAR';
Session altered.

SQL> alter table essai add col4 CHAR(4);
Table altered.

SQL> @sel

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH CHAR_USED
------------- ---------- ----------- ----------- ---------
COL_BYTE CHAR 4 4 B
COL_CHAR NCHAR 8 4 C
```

```
COL3 NCHAR 8 4 C
COL4 CHAR 4 4 C
```

*** Test with AL32UTF8 character set and AL16UTF16 national character set

*** For multi-byte character sets like AL32UTF8, BYTE or CHAR does make a difference
*** since 1 character may be stored in several bytes

```
SQL> select * from v$nls_parameters where parameter like '%CHARACTERSET%';

PARAMETER VALUE
------------------------------ --------------------
NLS_CHARACTERSET AL32UTF8
NLS_NCHAR_CHARACTERSET AL16UTF16

SQL> @sel

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH CHAR_USED
------------- ---------- ----------- ----------- ---------
COL_BYTE CHAR 4 4 B
COL_CHAR NCHAR 8 4 C
COL3 NCHAR 8 4 C
COL4 CHAR 16 4 C

SQL> alter session set nls_length_semantics='BYTE';
Session altered.

SQL> alter table essai add col5 CHAR(5 CHAR);
Table altered.

SQL> @sel

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH CHAR_USED
------------- ---------- ----------- ----------- ---------
COL_BYTE CHAR 4 4 B
COL_CHAR NCHAR 8 4 C
COL3 NCHAR 8 4 C
COL4 CHAR 16 4 C
COL5 CHAR 20 5 C
```

*** For CHAR and VARCHAR2, the limits of the bytes stored remain the same

=> 2000 bytes for CHAR

```
SQL> alter table essai add (col499 CHAR(499 CHAR),
2 col500 CHAR(500 CHAR),
3 col501 CHAR(501 CHAR),
4 col2000 CHAR(2000 CHAR));
Table altered.

SQL> alter table essai add c char(2001 byte);
alter table essai add c char(2001 byte)
*
ERROR at line 1:
ORA-00910: specified length too long for its datatype
```

```
SQL> alter table essai add c char(2001 char);
alter table essai add c char(2001 char)
*
ERROR at line 1:
ORA-00910: specified length too long for its datatype

SQL> @sel

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH CHAR_USED
--------------- ---------- ----------- ----------- ---------
COL499 CHAR 1996 499 C
COL500 CHAR 2000 500 C
COL501 CHAR 2000 501 C
COL2000 CHAR 2000 2000 C
```

COL2000 is created with 2000 CHAR : 2000 CHAR are allowed because 2000 CHAR may be stored on 2000 bytes if each character requires 1 byte only. But 2001 CHAR will never be storable on 2000 bytes only.

=> 4000 bytes for VARCHAR2

```
SQL> alter table essai add (col999 VARCHAR2(999 CHAR) ,
2 col1000 VARCHAR2(1000 CHAR) ,
3 col1001 VARCHAR2(1001 CHAR) ,
4 col4000 VARCHAR2(4000 CHAR));
Table altered.

SQL> alter table essai add c varchar2(4001 byte);
alter table essai add c varchar2(4001 byte)
*
ERROR at line 1:
ORA-00910: specified length too long for its datatype


SQL> alter table essai add c varchar2(4001 char);
alter table essai add c varchar2(4001 char)
*
ERROR at line 1:
ORA-00910: specified length too long for its datatype

SQL> @sel

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH CHAR_USED
--------------- ---------- ----------- ----------- ---------
COL999 VARCHAR2 3996 999 C
COL1000 VARCHAR2 4000 1000 C
COL1001 VARCHAR2 4000 1001 C
COL4000 VARCHAR2 4000 4000 C
```

COL4000 is created with 4000 CHAR : 4000 CHAR are allowed because 4000 CHAR may be stored on 4000 bytes if each character requires 1 byte only. But 4000 CHAR will able to store 4000 bytes only.

*** For NCHAR and NVARCHAR2, the limits of the characters stored remain the same
=> 2000 bytes for NCHAR
=> 4000 bytes for NVARCHAR2

```
SQL> alter table essai add (ncol999 NCHAR(999) ,
2 ncol1000 NCHAR(1000) ,
3 ncol1999 NVARCHAR2(1999) ,
```

```
4 ncol2000 NVARCHAR2(2000));
Table altered.

SQL> alter table system.essai add c NCHAR(1001);
c NCHAR(1001)
*
ERROR at line 2:
ORA-00910: specified length too long for its datatype

SQL> alter table system.essai add c NVARCHAR2(2001);
c NVARCHAR2(2001)
*
ERROR at line 2:
ORA-00910: specified length too long for its datatype

SQL> @sel

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH CHAR_USED
-------------- ---------- ----------- ----------- ---------
COL999 NCHAR 1998 999 C
COL1000 NCHAR 2000 1000 C
COL1999 NVARCHAR2 3998 1999 C
COL2000 NVARCHAR2 4000 2000 C
```

*** Indexes created with a different NLS_LENGTH_SEMANTICS value than the table

1/ The index creation uses the table column definition:
=> the index column COLUMN_LENGTH = table column DATA_LENGTH
the index column CHAR_LENGTH = table column CHAR_LENGTH

```
SQL> alter session set nls_length_semantics='CHAR';
Session altered.

SQL> create table A (c1 CHAR(4));
Table created.

SQL> select COLUMN_NAME, DATA_TYPE,DATA_LENGTH, CHAR_LENGTH,CHAR_USED
2 from dba_tab_columns where table_name='A';

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH CHAR_USED
-------------- ---------- ----------- ----------- ---------
C1 CHAR 16 4 C

SQL> alter session set nls_length_semantics='BYTE';
Session altered.

SQL> create index i_a on a(c1);
Index created.

SQL> select INDEX_NAME,COLUMN_NAME,COLUMN_LENGTH,CHAR_LENGTH
2 from dba_ind_columns where table_name='A';

INDEX_NAME COLUMN_NAME COLUMN_LENGTH CHAR_LENGTH
------------------------------ -------------- ------------- -----------
I_A C1 16 4

SQL> alter table a add c2 CHAR(4);
```

```
Table altered.

SQL> select COLUMN_NAME, DATA_TYPE,DATA_LENGTH, CHAR_LENGTH,CHAR_USED
2 from dba_tab_columns where table_name='A';

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH CHAR_USED
------------- ---------- ----------- ----------- ---------
C1 CHAR 16 4 C
C2 CHAR 4 4 B

SQL> create index i2_a on a (c2);
Index created.

SQL> select INDEX_NAME,COLUMN_NAME,COLUMN_LENGTH,CHAR_LENGTH
2 from dba_ind_columns where table_name='A';

INDEX_NAME COLUMN_NAME COLUMN_LENGTH CHAR_LENGTH
------------------------------ ------------- ------------ -----------
I_A C1 16 4
I2_A C2 4 4
```

2/ If you modify the semantics of a table column, the associated index is automatically modified.

```
SQL> alter table a modify c1 CHAR(4);
Table altered.

SQL> select COLUMN_NAME, DATA_TYPE,DATA_LENGTH, CHAR_LENGTH,CHAR_USED
2 from dba_tab_columns where table_name='A';

COLUMN_NAME DATA_TYPE DATA_LENGTH CHAR_LENGTH C
------------- ---------- ----------- ----------- -
C1 CHAR 4 4 B
C2 CHAR 4 4 B

SQL> select INDEX_NAME,COLUMN_NAME,COLUMN_LENGTH,CHAR_LENGTH
2 from dba_ind_columns where table_name='A';

INDEX_NAME COLUMN_NAME COLUMN_LENGTH CHAR_LENGTH
------------------------------ ------------- ------------ -----------
I_A C1 4 4
I2_A C2 4 4
```

I. Known problems:

* Bug 9533867 - desribe command does not properly handles nls_length_semantics
Fixed Ver(s): not ficed yet
Symptom(s): when NLS_LENGTH_SEMANTICS is not set in the client environment but it is set to CHAR in init.ora/spfile, the session value of the NLS_LENGTH_SEMANTICS is CHAR but SQL*Plus behaves as if it were BYTE.
Only after an explicit ALTER SESSION SET NLS_LENGTH_SEMANTICS=CHAR is issued,
the display works as expected.
Note that this is pure about the DESC *display* ,
Workaround: Do an ALTER SESSION or the CHAR_USED column from DBA_TAB_COLUMNS can also be used to to see if a column is created using BYTE or CHAR semantics, it then contains B or C regardless of the used NLS_LENGTH_SEMANTICS session setting.

* Bug 586825 MALFORMED RCI MARKER REDO DUE TO ORA-1551 IN UPDATE DRIVER.

Fixed-Releases: 10.2.0.5 .11.1.0.6
Details: This is not a bug with NLS_LENGTH_SEMANTICS as such but it may be returning "ORA-12899: value too large for column" errors on DATE or TIMESTAMP columns

rediscovery:
Set on the database SQL> alter system set events '1551 trace name errorstack level 3';
Check the alert.log for tracefiles when the error occures
Check if these tracefiles contain:
ksedmp: internal or fatal error
ORA-01551: extended rollback segment, pinned blocks released
If true then apply patch:5868257

* Not a bug but sometimes an ORA-01450 is seen during index creation on CHAR semantics columns in an UTF8 or AL32UTF8 database and it seams at first sight the (combined) column lenght is not enough to hit the limit.
For information about the ORA-1450 please see Note 136158.1 ORA-01450 and Maximum Key Length - How it is Calculated
It's important to realise the used length here for the index is the BYTE lenght of the column (DATA_LENGTH from dba_tab_columns).
for example the BYTE length of an varchar2(10 CHAR) column is 30 BYTES in UTF8 and 40 BYTES in AL32UTF8 (it does not matter if the column is poplulated or not).

* Note 756180.1 Import a Cluster Table From Single Byte to Multibyte and Changing NLS_LENGTH_SEMANTICS From BYTE to CHAR Fails With ORA-1753

* Bug 8839301 ALTER TABLE COMMAND TO ADD A CHAR(1 CHAR) COLUMN WITH DEFAULT VALUE CAUSES PROB
Fixed Ver(s): fixed in 11.2.0.2 , not applicable to 10.2 and lower.
Symptom(s): when adding a CHAR column using CHAR sementics and a default value to the default value internally a string of ,0,0,0 is added based on the max BYTE length. resulting in that a length on that column gives a larger value then the max defined datatype length.
conditions:
* multi byte db (AL32UTF8 for example)
* if added column is CHAR datatype and using CHAR semantics and has a "Default" value
* when the new CHAR column is populated with the "default" value during the add phase..
* does not happen when using the "default" in subsequent updates
Workaround: After doing an add column do directly an update statement with the default clause value

* bug 6167249 IMP-58 / ORA-1461 can occur while importing a table with a CLOB/BLOB column.
Fixed Ver(s): fixed in 11.2.0.2, 10.2.0.5
Symptom(s): If a table is containing a CLOB is precreated during an import with rows=N and subsequently one performs a rows=Y import of the same table from the same dumpfile, the last import operation will fail with the following errors : .
IMP-58: ORACLE error 1461 encountered
ORA-1461: can bind a LONG value only for insert into a LONG column
Workaround: none. Note: This fix is specific to IMPORT. The fix must be applied to the ORACLE_HOME which contains the "imp" executable being used.

* bug 7648406 Child cursors not shared for "table_..." cursors when NLS_LENGTH_SEMANTICS = CHAR
Fixed Ver(s): fixed in 11.2.0.2, 10.2.0.5, 11.1.0.7.4 PSU
Symptom(s): High version counts might be observed for pseudo cursors (ie: internal "table_*" cursors) when NLS_LENGTH_SEMANTICS = CHAR
Workaround: Set NLS_LENGTH_SEMANTICS=BYTE

* Bug 6275080 CHAR VALUES SELECTED FROM A SUBQUERY ARE PADDED WITH BLANKS
Fixed Ver(s): fixed in 11g, 10.2.0.4
Symptom(s): CHAR values selected from a subquery are padded with blanks when NLS_LENGTH_SEMANTICS = CHAR and NLS_CHARACTERSET is AL32UTF8.
Workaround: none

* Bug 5838153 CAST operator does NOT consider NLS_LENGTH_SEMANTICS
When using the CAST function to cast to a VARCHAR2, the NLS_LENGTH_SEMANTICS setting is ignored.
The fix for Bug 5838153 is only included in 10.2.0.4, but introduces an other problem hence 5838153 is removed from 10.2.0.5. To solve the issue apply the patch for bug 7154415.
Fixed Ver(s) (for 7154415): 10.2.0.5 and 11.2. Not included in any 11.1 patchset but on off patches on top of 11.1.0.7 exist (or can be asked).
When using CAST (even when not using CHAR SEMANTICS) on 10.2.0.4 please apply Patch 7154415 also.

* Bug 5545716 CATALOG.SQL AND CATPROC.SQL DOES NOT TAKE CARE OF NLS_LENGTH_SEMANTICS
(Internal bug - not visible on metalink sorry)
Fixed Ver(s): fixed in 11.1.0.6 and up

Symptom(s): If the database was created with NLS_LENGTH_SEMANTICS=CHAR then some sys objects will use CHAR semantics, this is not supported.
Select to see if you have objects like this:

```
select C.owner, C.table_name, C.column_name, C.data_type, C.char_length, C.char_used
from all_tab_columns C, all_tables T
where C.owner = T.owner
and T.owner in ('SYS', 'SYSTEM', 'CTXSYS', 'DBSNMP', 'DMSYS',
'EXFSYS', 'MDSYS', 'OLAPSYS', 'ORDSYS', 'OUTLN', 'SYSMAN', 'WKSYS',
'WK_TEST', 'WMSYS', 'XDB')
and C.table_name = T.table_name
and C.char_used = 'C'
and C.table_name not in (select table_name from all_external_tables)
and C.data_type in ('VARCHAR2', 'CHAR');
```

This select should give no rows.
Workaround: do not create database with NLS_LENGTH_SEMANTICS = CHAR but change it to CHAR after the database creation.
Fix: rerun catalog and catproc.sql in a "startup migrate" session.

* Bug 6627733 - ora-00600:[dmlsrvcollenchk_2:dty] consistently .
Fixed Ver(s): fixed in 11.1.0.7, 10.2.0.5 and 11.2.0.1 and up
Symptom(s): ora-00600:[dmlsrvcollenchk_2:dty] error incrrectly reported when inserting NULL values into varchar2 columns using CHAR semantics
Workaround: none, besides not inserting NULL values.

* Bug 3611750 ORA-01450 ONLINE REBUILD OF INDEX FAILS
Fixed Ver(s): fixed in 11.1.0.6, 10.2.0.5 and up
Symptom(s): When doing a online rebuild using CHAR semantics this sometimes fails with ORA-01450: maximum key length (3215) exceeded
Workaround: do " alter session set nls_length_semantics=byte; " before the online rebuild. This will not affect the table/index, the problem is pure with the journaling table.

* Bug 4886376 INVALID SYS.KU$_XMLSCHEMA_VIEW AFTER CREATING DB WITH NLS_CHARSEMANTICS=CHAR
Fixed Ver(s): fixed in 11.1.0.6, 10.2.0.4 and up
Symptom(s): When you have XDB installed export will fail with

```
EXP-00056: ORACLE error 932 encountered
ORA-00932: inconsistent datatypes: expected BLOB, CLOB got CHAR
```

or with XDB installed you get:

```
select * from SYS.KU$_XMLSCHEMA_VIEW
*
ERROR at line 1:
ORA-00932: inconsistent datatypes: expected BLOB, CLOB got CHAR
```

NOTE: If you do NOT have XDB installed then the ORA-932 error when selecting from SYS.KU$_XMLSCHEMA_VIEW can still occur. However, in that case this is not a symptom of this bug, instead this is Bug 5113561. That bug occurs regardless of the length semantics that are set when XDB is not installed. Because if XDB is not installed there is no reason to select from this view, and therefore the error itself can be ignored. If you do not have XDB installed then you will not run into Bug 4886376 and therefore you do not have to apply the workaround below, and you should not have any problems with the exports.
Workaround: do not create database with NLS_LENGTH_SEMANTICS = CHAR but change it to CHAR after the database creation. To correct a existing db, please see the workaround in Bug 4886376

* Bug 4638550 OERI[dmlsrvColLenChk_2:dty] during upgrade from 9.2 to 10.2
Fixed Ver(s): fixed in 11.1.0.6, 10.2.0.3 and up
Symptom(s): ORA-00600 [dmlsrvColLenChk_2:dty], [23]' can occur during upgrade from 9.2 to 10.2 , or on tables after upgrade if NLS_LENGTH_SEMANTICS was CHAR when the table was initially created in 9.2.0.1 or 9.0.1. This can only occur if a table contains a RAW column.
Wrokaround: check for problem COL$ entries, they can be detected before going to 10g using the SQL:
select obj#,name,property,spare3, length
from col$ where bitand(property,8388608)!=0 and type#=23 ;

* Bug 4563316 CPUJUL2005: RUNNING CATCPU.SQL PERMANENTLY CHANGES NLS_LENGTH_SEMANTICS
Fixed Ver(s): in JAN CPU 2006 patch and later
Symptom(s): NLS_LENGTH_SEMANTICS = BYTE after running catcpu.sql when using a spfile.

Workaround: IF you use NLS_LENGTH_SEMANTICS = CHAR then after running catcpu ensure you run an

```
ALTER SYSTEM SET NLS_LENGTH_SEMANTICS=CHAR scope=spfile;
```

and restart the instance to ensure all sessions will use the correct setting. Note that ALTER SYSTEM *without* a restart is NOT enough (see next bug).

* Bug 1488174 UNICODE: ALTER SYSTEM SET NLS_LENGTH_SEMANTICS DOESN'T TAKE EFFECT
(this is an internal bug, so not visible on metalink, sorry)
Fixed Ver(s): Not yet fixed - only the fact scope=both; does not give an error is the real bug
Testcase:

```
alter system set nls_length_semantics='CHAR' scope=both;
create table test(v1 varchar2(10));
select CHAR_USED from dba_tab_columns where table_name='TEST';
```

you would expect 10 character (C) , but actually, it is 10 BYTE (B);
Problem: ALTER SYSTEM does not change the setting of NLS_LENGTH_SEMANTICS for the current and new (!) sessions.
Workaround: Don't use ALTER SYSTEM SET NLS_LENGTH_SEMANTICS (scope=both); but set NLS_LENGTH_SEMANTICS as a init.ora parameter or issue ALTER SYSTEM SET NLS_LENGTH_SEMANTICS=CHAR scope=spfile; and bounce the database.
or
use alter session to change the current session without bouncing the database (but new sessions will still use the old value...).

* Bug 4462785 Abstract: VARCHAR2(X CHAR) COLUMN ALLOW A STRING > X CHARS
IN UTF8 USING APPEND INSERT
Fixed Ver(s): 10gR1 and up , NOT fixed in 9i
Testcase:

```
create table source ( a varchar2(2 char) );
create table dest ( a varchar2(1 char) );
insert into source a values ('12');
commit;
insert /*+ APPEND */ into dest select * from source;
1 row created.
```

When leaving out the APPEND hint, you'll get the proper database response:

```
insert into dest select * from source;
ORA-1401: inserted value too large for column
```

Problem: using APPEND does not take CHAR semantics into account in certain cases.
Workaround: use blank_trimming=true or a check constraint on the column (alter table <table> add check constraint check_length<col> (length(<col>) <= <max length>))
In 10 g you get the correct error:

```
SQL>insert /*+ APPEND */ into dest select * from source;
insert /*+ APPEND */ into dest select * from source
*
ERROR at line 1:
ORA-12899: value too large for column "SCOTT"."DEST"."A" (actual: 2, maximum: 1)
```

* Bug 4122805 NLS_COMP=ANSI DISABLES NLS_LENGTH_SEMANTICS=CHAR
Fixed Ver(s): 9.2.0.7 , 10.1.0.5, 10gR2 and up
Testcase:
alter session set nls_length_semantics = CHAR;
alter session set nls_comp = ansi;
select * from nls_session_parameters where parameter=NLS_LENGTH_SEMANTICS
PARAMETER VALUE
------------------------------ --
NLS_LENGTH_SEMANTICS BYTE <=== VALUE HAS CHANGED!
Problem:Changing the value of NLS_COMP also resets NLS_LENGTH_SEMANTICS
(this happens also in recursive sql)
Workaround: none, besides using NLS_COMP=BINARY or define the CHAR explicit

* Bug 3040360 NLS_LENGTH_SEMANTICS PARAMETER NOT WORKING PROPERLY ORA-1401
Fixed Ver(s): 9.2.0.5 , 10gR1 and up

Symptom(s): - With a database created with a multibyte character set such as AL32UTF8 and with nls_length_semantics=char and BLANK_TRIMMING=FALSE, a before insert trigger will blank-pad a column to its full data_length instead of only to its char_length.
Workaround: set blank_trimming=true in the init.ora or Use BYTE length semantics for the database.

* Note 274514.1 ORA-22347 or PLS-719 attempting to change length semantics of type attribute
Fixed Ver(s): 9.2.0.6, 10.1.0.3, 10gR2 and up
Symptom(s): ORA-22347 or PLS-719
For more info about this bug please see the mentioned note.

* Bug 2346368 ORA-06502 in multibyte using NLS_LENGTH_SEMANTICS=CHAR
Fixed Ver(s): 9.2.0.2 , 10gR1 and up
Symptom(s): - With NLS_LENGTH_SEMANTICS=CHAR and a database with a multibyte character set ORA-06502 may occur when using a string literal as an actual parameter to a function.
Workaround: Use BYTE length semantics
References: Note 227623.1 Executing PLSQL Using String Literals With Semantics Fails With ORA-06502

* Bug 2743295 ORA-22973 CREATE VIEW SYS.KU$_XMLSCHEMA_VIEW WITH NLS_LENGTH_SEMANTICS=CHAR
Fixed Ver(s): 9.2.0.4 , 10gR1 and up
Symptom(s): Creating XDB in a unicode database with NLS_LENGTH_SEMANTICS=CHAR fails.
Workaround: Use BYTE length semantics for the database.

* Note 250802.1 Changing character set takes a very long time and uses lots of rollback space
Fixed Ver(s): 9.2.0.4 , 10gR1 and up
Symptom(s): ALTER DATABASE CHARACTER SET may exhaust shared pool and/or run out of rollback segment space on a character semantics database.
Workaround: none

* Note 235327.1 Import into a Multi-byte Database with NLS_LENGTH_SEMANTICS='CHAR' Fails with IMP-19, IMP-3, ORA-1404 (see also point E. in this note)

--

References

NOTE:241047.1 - The Priority of NLS Parameters Explained (Where To Define NLS Parameters)
NOTE:269381.1 - ORA-01406 or ORA-06502 in PLSQL when querying data in (AL32)UTF8 db from remote non-UTF8 db using cursor
NOTE:276914.1 - The National Character Set (NLS_NCHAR_CHARACTERSET) in Oracle 9i, 10g and 11g
NOTE:313175.1 - SCRIPT: Changing columns to CHAR length semantics (NLS_LENGTH_SEMANTICS)
NOTE:756180.1 - Import a Cluster Table From Single Byte to Multibyte and Changing NLS_LENGTH_SEMANTICS From BYTE to CHAR Fails With ORA-1753
NOTE:788156.1 - AL32UTF8 / UTF8 (Unicode) Database Character Set Implications