

最近测试一个工具 Database Migration for Unicode, 功能是把非 unicode 编码存储数据的数据库转化为 Unicode 的数据库。在 guide 中提到一句, CLOB 类型的列无需转换编码, 这句话引起我的兴趣, 为何别的字符类型 char 或者 varchar2 都需要编码转换, 为什么 CLOB 不需要。在网上搜索, 找到了 DBSNAKE 的这篇文章 (<http://dbsnake.com/2010/07/lob-character-set-endian.html>)。里面提到了一篇 Note [ID 257772.1], 核心意思是, 基本上从 10g 之后的 CLOB 的字符编码无论在何种 CharacterSet 下均为 AL16UTF16。现在特来验证一下。

1. 在 JA16EUC 的数据库中创建测试用户及测试表(LOB 存储为 disable storage in row)

```
select value from nls_database_parameters where PARAMETER='NLS_CHARACTERSET';
```

VALUE

JA16EUC

```
create table tst.tab1(col1 CLOB) tablespace data lob (col1) store as (tablespace DATA disable storage in row nocache);
```

```
insert into tst.tab1 values(' ビジネス運営' );  
commit;
```

找出这一行存储的物理文件和 block

```
select dbms_rowid.rowid_block_number(rowid) bno, col1 from tst.tab1;
```

BNO COL1

135 ビジネス運営

```
select FILE_NAME, FILE_ID from dba_data_files;
```

FILE_NAME	FILE_ID
/u01/app/oracle/oradata/euc2/system01.dbf	1
/u01/app/oracle/oradata/euc2/sysaux01.dbf	2
/u01/app/oracle/oradata/euc2/undotbs01.dbf	3
/u01/app/oracle/oradata/euc2/users01.dbf	4
/u01/app/oracle/oradata/euc2/data.dbf	5

```
alter system dump datafile 5 block 135;
```

在 udump 里找到对应的 trace 文件, 因为使用的是 11gR2, 所以路径略有不同

```
SELECT value FROM v$diag_info WHERE name='Default Trace File';
```

VALUE

/u01/app/oracle/diag/rdbms/euc2/euc2/trace/euc2_ora_6234.trc

在上述 trace 里找到

```

block_row_dump:
tab 0, row 0, @0x1f80
tl: 24 fb: --H-FL-- lb: 0x1  cc: 1
col 0: [20] 00 54 00 01 02 08 80 00 00 02 00 00 00 01 00 00 00 2d c8 b5 -->这里后 10 个字节就是 LobID
LOB
Locator:
  Length:          84(20)
  Version:         1
  Byte Length:    2
  LobID: 00.00.00.01.00.00.00.2d.c8.b5
  Flags[ 0x02 0x08 0x80 0x00 ]:
  Type: CLOB
  Storage: BasicFile
  Disable Storage in Row
  Characterset Format: IMPLICIT -->可以发现这里并没有写明 CLOB 字段所使用的编码
  Partitioned Table: No
  Options: VaringWidthReadWrite

```

这里就需要介绍一下 LOB 在 enable 还是 disable storage in row 这两种模式下的存储格式。
 先看一下我们这个含有 LOB 的 TAB1 表有哪些相关的 segment:
 select SEGMENT_NAME,SEGMENT_TYPE from dba_segments where OWNER='TST' ;

SEGMENT_NAME	SEGMENT_TYPE
TAB1	TABLE
SYS_IL0000017724C00001\$\$	LOBINDEX
SYS_LOB0000017724C00001\$\$	LOBSEGMENT

背景知识, LOB 是 Large Object (大对象) 的简称, 包括 CLOB, NCLOB, BLOB, BFILE。这里我们讨论的只是 CLOB。
 创建每一个 LOB 列除了表中的一个 column 外, 另外同时还创建了两个物理结构: LOB index 和 LOB segment。
 首先, 需要了解的是在默认情况下 LOB segment 是和其所在的 table 存储在一起的, 即 in-line。而另外一种, LOB 的访问是通过 LOB locator 来访问 LOB index, LOB index 再指向真正的物理存储数据的地方即 LOB segment 里的 chunk。LOB locator 就是指向 LOB index 的 Key。

接下来先看一下 Enable storage in row 的方式下 LOB 的存储。这里分为两种情况, 小于 4000 字节和大于 4000 字节。

LOB 字段的 locator 加 Inode 加 LOB 的内容总长度小于 4000 字节时:

Col1	LOB Col
1	Locator1, Inode 1, LOB data 1
2	Locator2, Inode 2, LOB data 2

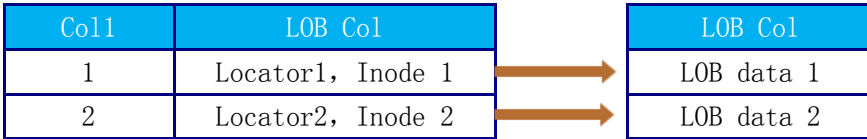
这里 LOB 字段的存储是按如下的格式, 共 36 字节:
 2 字节的 LOB locator 长度 (除这两个长度字节外)
 2 字节的 LOB locator structure 版本
 4 字节的 FLAG
 2 字节的字符集里字符的长度

10 字节的 LOB ID

16 字节的 inode

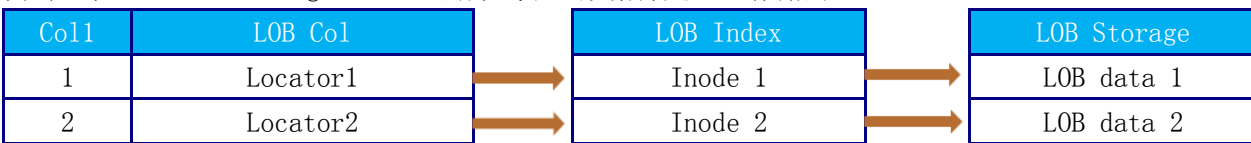
之后就是 LOB data，所以这里 in-line 的情况下理论上最多能存储 $4000-2-2-2-4-10-16=3964$ 字节。

其他情况：



就是说即使 Enable storage in row，LOB 字段在 LOB 的控制信息（即 Locator 加 Inode）长度加 LOB data 长度大于 4000 字节的时候，LOB data 也是不存储在 row 内部的。在表里 LOB 字段的第 37-84 字节存储的是 LOB data 的前 12 个 chunk 的 RDBA。如果 12 个不够的情况下，这里存放的不是 chunk 的 RDBA，而是 LOB index 的 RDBA。

另外，在 Disable storage in row 的方式下，数据都是这么存储的：



LOB index 是一个类似于 unique index 的结构。这里我们把 LOB index 的叶子块的信息 dump 出来。

先找出 TAB1 的 OBJ#：

```
select object_id from dba_objects where object_name='TAB1' ;
```

```
OBJECT_ID
-----
17724
```

再找出 TAB1 里的 LOB index 的 ind#：

```
select IND# from lob$ where OBJ#='17724' ;
```

```
IND#
-----
17726
```

把这个 index 的 tree 给 dump 出来：

```
alter session set events 'immediate trace name treedump level 17726' ;
SELECT value FROM v$diag_info WHERE name='Default Trace File' ;
```

```
VALUE
-----
/u01/app/oracle/diag/rdbms/euc2/euc2/trace/euc2_ora_11704.trc
```

从上面这个 trace 中找到：

```
----- begin tree dump
leaf: 0x1400093 20971667 (0: nrow: 1 rrow: 1)
```

```
----- end tree dump
```

解释：前两列分别是 RDBA 的十六进制和十进制地址，由此可以得出 leaf block 的 number。Nrow 是行数，如果是 branch block 的话就是 child nodes 的个数。

```
select DBMS_UTILITY.data_block_address_block(TO_NUMBER(LTRIM('1400093'),'xxxxxxxx')) BNO from dual;
```

```
      BNO
-----
      147
```

```
alter system dump datafile 5 block 147;
```

得到 50 个字节的 LOB index

```
row#0[7982] flag: -----, lock: 2, len=50, data:(32):
 00 20 03 00 00 00 00 00 00 00 0c 00 00 00 00 00 01 01 40 00 8f 00 00 00 00 00
 00 00 00 00 00 00 00
col 0; len 10; (10): 00 00 00 01 00 00 00 2d c8 b5
col 1; len 4; (4): 00 00 00 00
```

LOB index 的结构是：

长度为 50 个 byte，具体组成如下：

行头 flag (1 个 byte)

lock byte (1 个 byte)

lob index key (32 个 byte, 要么由 16 个 byte 的 lob inode+4 个 RDBA 组成, 要么由 8 个 RDBA 组成。Disable storage in row 的情况下是前者)

lob id 的长度 (1 个 byte) + lob id (10 个 byte)

page number 的长度 (1 个 byte)

page number (4 个 byte)

Inode 的结构是：

2 字节的 in-line 的 data 的长度

1 字节的 inode 标志位：1 代表这是个有效的 lob，2 代表这个 inode 在 LOB index 里，3 代表这个 lob 是个有效的 lob，而且在 LOB index 里

1 字节的预留位

4 字节 full oracle blocks 数量

2 字节最后一个 page 里有多少 byte 的数据

6 字节的 LOB version, 0x0001.00000000 的形式

所以这里 **01 40 00 8f** 就是第一个 chunk 的 RDBA 的地址。

```
select DBMS_UTILITY.data_block_address_block(TO_NUMBER(LTRIM('0140008f'),'xxxxxxxx')) BNO from dual;
```

```
      BNO
-----
      143
```

```
alter system dump datafile 5 block 143;
```

可以从 trace 中得到 CLOB 字段的实际存储的物理格式:

Long field block dump:

Object Id 17725

LobId: 00010002DC8B5 PageNo 0

Version: 0x0000.00000000 pdba: 20971656

30 d3 30 b8 30 cd 30 b9 90 4b 55 b6 00 20 00 20 00 20 00 20 00 20

我们验证一下这里的十六进制码到底跟哪个 CharSet 里的文字的物理格式一样:

```
create table aaa (col1 varchar2(100), col2 nvarchar2(100));
```

```
insert into aaa values ('ビジネス運営', 'ビジネス運営');
```

```
select dump(col1, 1016), dump(col2, 1016) from aaa;
```

DUMP (COL1, 1016)

DUMP (COL2, 1016)

Typ=1 Len=12 CharacterSet=JA16EUC: a5, d3, a5, b8, a5, cd, a5, b9, b1, bf, b1, c4

Typ=1 Len=12 CharacterSet=AL16UTF16: 30, d3, 30, b8, 30, cd, 30, b9, 90, 4b, 55, b6

可见 CLOB 的物理存储格式并不是 JA16EUC, 而是 AL16UTF16。

在研究这个东西的过程中得到了崔华 (<http://dbsnake.com>) 的大力帮助, 在这里表示感谢。

参考:

DSI 402e: P52-P57, P250-P254

<http://dbsnake.com/2010/07/some-lob-internal.html>