What a DBA Should Know or How to Know Your Way Around Database

Vit Spinka
Dbvisit Softare Limited
Auckland, NZ

Keywords
Oracle dictionary, Oracle multitenant, Oracle data structures, object id, data object id

Introduction
The Oracle database is a complex system and each version brings new features and complexities. To properly understand what is happening behind the scenes, one must understand its inner structures. In this session we look at how to identify some of the most common database objects like tables or columns. A table has much more than just one simple id - and one must recognize this in order to understand the inner workings of Oracle Database as well as to write correct scripts querying the Oracle dictionary.

Dictionary views
The usual (and essentially only) way how to get data about the database is through querying the dictionary views. (Note: In this session, we concern ourselves almost exclusively with the database, that means information persistently stored on the disk.)
The basic views—and views that we will concern with are:

- DBA_DATA_FILES
  Describes files in database: file_id, relative_fno
- DBA_OBJECTS
  Tables, indexes, partitions: object_id, data_object_id
- DBA_TABLES
  Tables: cluster_name, iot_type
- DBA_TAB_COLS
  Columns in a table: column_id, segment_column_id, internal_column_id
- DBA_PDBS
  Containers: pdb_id, con_uid, dbid, con_id
- DBA_EDITIONS
  PL/SQL Editions

For all of these views, the usual three flavors exist:

- USER—owner = user
- ALL—user has privileges
- DBA—all in this database

And 12c introduces the fourth flavor:

- CDB—all in all pluggable databases (12c multitenant)

Dictionary tables
The dictionary views on SYS-owned dictionary tables. It's not necessary to go to this level of detail; however, exploring how the views are build on top of them reveal interesting details and tricks. And the DDL of the tables is easily available and well commented, shedding more light of the purpose of the data contained in them. To find the DDL, go to $ORACLE_HOME/rdbms/admin/sql.bsq; older versions had the DDL directly in this file, newer ones use this file as a simple start and include further files with the actual DDL (like dcore.bsq).
So for example, the basic table containing information for all objects in the database is defined as:

```
create table obj$                                       /* object table */
```

```
( obj#          number not null,                            /* object number */
  dataobj#      number,                           /* data layer object number */
  owner#        number not null,                          /* owner user number */
  name          varchar2("M_IDEN") not null,                /* object name */
  namespace     number not null,         /* namespace of object (see KQD.H): */
 /* 1 = TABLE/PROCEDURE/TYPE, 2 = BODY, 3 = TRIGGER, 4 = INDEX, 5 = CLUSTER, */
                                                 /* 8 = LOB, 9 = DIRECTORY, */
  /* 10 = QUEUE, 11 = REPLICATION OBJECT GROUP, 12 = REPLICATION PROPAGATOR, */
                                     /* 13 = JAVA SOURCE, 14 = JAVA RESOURCE */
                                                /* 58 = (Data Mining) MODEL */
  subname       varchar2("M_IDEN"),               /* subordinate to the name */
  type#         number not null,                /* object type (see KQD.H): */
  /* 1 = INDEX, 2 = TABLE, 3 = CLUSTER, 4 = VIEW, 5 = SYNONYM, 6 = SEQUENCE, */
             /* 7 = PROCEDURE, 8 = FUNCTION, 9 = PACKAGE, 10 = NON-EXISTENT, */
              /* 11 = PACKAGE BODY, 12 = TRIGGER, 13 = TYPE, 14 = TYPE BODY, */
      /* 19 = TABLE PARTITION, 20 = INDEX PARTITION, 21 = LOB, 22 = LIBRARY, */
                                           /* 23 = DIRECTORY , 24 = QUEUE, */
    /* 25 = IOT, 26 = REPLICATION OBJECT GROUP, 27 = REPLICATION PROPAGATOR, */
    /* 28 = JAVA SOURCE, 29 = JAVA CLASS, 30 = JAVA RESOURCE, 31 = JAVA JAR, */
                /* 32 = INDEXTYPE, 33 = OPERATOR , 34 = TABLE SUBPARTITION, */
                                            /* 35 = INDEX SUBPARTITION */
                                           /* 82 = (Data Mining) MODEL */
                                 /* 92 = OLAP CUBE DIMENSION,  93 = OLAP CUBE */
                   /* 94 = OLAP MEASURE FOLDER, 95 = OLAP CUBE BUILD PROCESS */
  ctime         date not null,                       /* object creation time */
  mtime         date not null,                      /* DDL modification time */
  stime         date not null,          /* specification timestamp (version) */
  status        number not null,             /* status of object (see KQD.H): */
                                      /* 1 = VALID/AUTHORIZED WITHOUT ERRORS, */
                           /* 2 = VALID/AUTHORIZED WITH AUTHORIZATION ERRORS, */
                             /* 3 = VALID/AUTHORIZED WITH COMPILATION ERRORS, */
                           /* 4 = VALID/UNAUTHORIZED, 5 = INVALID/UNAUTHORIZED */
  remoteowner   varchar2("M_IDEN"),     /* remote owner name (remote object) */
  linkname      varchar2("M_XDBI"),             /* link name (remote object) */
  flags         number,               /* 0x01 = extent map checking required */
                                      /* 0x02 = temporary object             */
                                      /* 0x04 = system generated object      */
                                      /* 0x08 = unbound (invoker's rights)   */
                                      /* 0x10 = secondary object             */
                                      /* 0x20 = in-memory temp table         */
                                      /* 0x80 = dropped table (RecycleBin)   */
                                      /* 0x100 = synonym VPD policies        */
                                      /* 0x200 = synonym VPD groups          */
                                      /* 0x400 = synonym VPD context         */
                                      /* 0x4000 = nested table partition     */
  oid$          raw(16),        /* OID for typed table, typed view, and type */
  spare1        number,                       /* sql version flag: see kpul.h */
  spare2        number,                             /* object version number */
  spare3        number,                                       /* base user# */
  spare4        varchar2(1000),
  spare5        varchar2(1000),
  spare6        date,
  signature     raw(16),                     /* object signature hash value */
  spare7        number,                                       /* future use */
  spare8        number,
  spare9        number
);
```

ALL, DBA and USER views select from these dictionary tables (only limiting what to display based on user and privileges). CDB views query from all pluggable databases using the CONTAINERS (or CDB$VIEW in 12.1.0.1) clause.

The most interesting tables are:

- SYS.OBJ$—objects
- SYS.TAB$, SYS.COL$—tables and columns

- SYS.IND$, SYS.ICOL$—indexes and columns
- SYS.CDEF$, SYS.CCOL$—constraints and columns
- SYS.TS$, SYS.FILE$—tablespaces and files
- SYS.EDITION$—PL/SQL editions
- SYS.CONTAINER$—PDBs

Files
Until Oracle7, a single file number (up to 1023) was enough to identify a datafile since birth to drop. Oracle8 brought higher limit of files in a database and TTS (transportable tablespaces). This called for a new solution: adding "relative file number", guaranteed to be unique in a single tablespace only.
As long as you have less then 1023 datafiles and don't import TTS, absolute and relative file numbers will match. But don't depend on it in your scripts.

Rowids
Rowids were directly affected by the new relative file number and had to be extended to accommodate for this.
Oracle7 has what we now call "restricted rowid" - block, row in block, (absolute) file number. Oracle8 introduced "extended rowid"—data object id, relative file number, block, row in block (and big datafile changes it again a bit).
Relative file number is still limited to 1023 - but now that's the limit for number of datafiles in a single tablespace, not the whole database.
Rowid is not unique: relative file number is not unique, so having many datafiles and/or use of TTS can lead to identical rowids (small chance if random—but high chance if you export–import your own tablespace). A second example is rows in cluster: they share datablocks and rowids.

Object id and data object id
There are two orthogonal concepts - a logical definition of a object and it's storage. So while object_id identifies an object logically (like in DBA_OBJECTS), data_object_id identifies the data segment (like DBA_SEGMENTS). Oracle decided to use the same sequence for both of them - and a newly created (simple heap) table will have indeed both values identical. However, it is necessary to distinguish them.
There are multiple reasons why these ids won't match. For starters, a truncate creates a new segment and assigns a new id. Let's see other examples in following chapters.
Note: data object id is not unique—an imported TTS will keep it's data object id in order to make rowids stable (and thus not breaking existing indexes).

Clustered tables
Often neglected data structure is a cluster. That is—neglected by user applications. Internally Oracle uses them heavily in the data dictionary, though.
In a cluster, one or more tables share the same data segment. This implies they will have the same data object id—keeping object id unique.

```
SQL> select obj#, dataobj#, tab# from sys.tab$ where …
      OBJ#   DATAOBJ#       TAB#
---------- ---------- ----------
    482583     482581          1
    482584     482581          2
    482585     482581          3
```

Partitions
A table has one object id; each of its partitions and subpartitions is also an object with its object id and data object id. So indirectly the table has many data object ids.
The (sub)partitions are linked to the "parent" table by tabpart$.bo# (base object id) or obj$.subname (name of the parent table).

Index-Organized Tables
An IOT is composed of multiple objects and each has an object id:
- Object id of index
- Object id of table
- Object id of overflow segment (columns not included in the index)
- Object id of mapping table (auxiliary table for bitmap indexes on IOT)

The index, overflow segment and mapping table each have a data object id—and can be partitioned, multiplying the number of the ids.
The link between these objects is through bobj# (base object id) in tab$. (Generally, base object id is the object id of the "parent" or "base" object—e.g. base table of an index.)

```
SQL> select obj#, tab$.bobj#, name from tab$, obj$ …
      OBJ#         BOBJ# NAME
---------- ---------- ------------------------------
    476774     476773 SYS_IOT_OVER_476773
    476773     476774 IOTTABLE
```

Columns
There are three different ids for a column.
- col#: column id—for internal columns, this is 0. The value usually presented to the user.
- intcol#: internal column id. Unique and thus set for each column.
- segcol#: position in segment. 0 for columns not stored (e.g. virtual or SYS_NC_ROWINFO$)).

col#
This is the id given to the user. It's 0 for internal columns, however. Note that Oracle implies 1000 column limit in a single table - and that includes all of them, including those internal ones. So don't depend just on visible columns or on highest column id.
It's also not guaranteed to be unique, e.g. when a user facing ADT is stored decomposed:

```
COL#  SEGCOL#    INTCOL# NAME
----- ------- ---------- ----------------------
    1       1          1 CUSTID
    2       0          2 ADDRESS
    2       2          3 SYS_NC0000200003$
```

intcol#
This is the internal column id and it's guaranteed to be unique. Having such id even for internal columns gives Oracle the ability to create indexes or constrains on these columns.
Note that Oracle often uses both col# and intcol# in other tables (icol$, ccol$).

segcol#
Segment column id is the position of the column in the actual data blocks. There are some reasons why this is not always same as intcol#:
- Some columns are present in the dictionary, but not stored - nested table column, virtual specified by user, virtual created by Oracle (e.g. extended stats), pseudo nc_rowinfo$ used to access XMLType value
- A LONG value must be always the last column in a table - so it will have the highest segcol#
- Cluster keys are always first columns in the table (as they are actually stored in a different table)

Editions
Edition based redefinition introduces new naming resolution and new name scope. Internally, each version of each object has a new object id.
It also introduces one level of indirection in dictionary views—instead of obj$, they are built on top of _current_edition_obj which select objects for the currently enabled edition only. (And Oracle add %_AE views that show all objects.)

RAC
RAC is really an instance-only concept—all data is shared across nodes and thus all said before is still valid.

Multitenant
Pluggable databases bring another level of indirection—all of the things mentioned so far are valid at a single PDB. Even the dictionary tables are stored in each PDB (with some magic linking happening for global objects).
Thus for most of queries it is easiest just to connect to a single PDB and think at the PDB level only. If a query needs to be done at the global level, all CDB views have a con_id, which is current container id (and which can change on unplug/plug). If you need a permanent id, use con_uid, which is stable like dbid.
You can also query all of the PDBs at the same time - using the CONTAINERS clause. (Beware that this clause excludes PDB$SEED, though.)
Still, multitenant keep some data global: absolute file numbers are still unique and OMF uses unique per-PDB directories to keep files separated.

Contact Address:
Vit Spinka
Dbvisit Software Limited
Level 1, 506 Point Chevalier Road
Point Chevalier
Auckland 1022
New Zealand

Telefon:	+64 9 950 3301
Fax:	+64 9 950 3302
E-Mail	vit.spinka@dbvisit.com
Internet:	www.dbvisit.de