“Data Hiding” with IDMS/SQL:

Making an IDD Code Table behave like a “real” SQL Table

Page Contents

Accessing an IDD Table with IDMS/SQL

I expect that most sites (in USA anyway) have an IDD Code Table of American State codes (Encode value in the database) and State Names (Decode value for display).

I have created a shortened version of such a table here. You can display the table in IDD at any time – but unless you specify “FORM FIXED.” Before the display, the output can become very confusing with Encode/Decode values all “smershed” together in a jumble.

```
add TABLE NAME   USACODEC   ver 1
     Description 'USA State Codes'
     TYPE IS CODE SEARCH  LINEAR
     ENCODE DATA  ALPHANUMERIC DECODE DATA ALPHANUMERIC
     TABLE  UNSORTED VALUES ARE (
      SA  'South Aus' MA  'Maine'
      UN  'Unwanted'  ' ' 'Blank'
      NOT FOUND  'Unknown'         )  GENERATE.
*+ I DC601133                       CARD 000022 WORD 03
*+ - GENERATE SUCCESSFUL - 'OBJECT' TABLE SIZE IS 76 BYTES
```

So – let’s just say I want a better display of the Table, and the ability in fact to display any other IDD code table – I could create an SQL View as shown here:

```
CREATE VIEW RSQLSHR.codecTable
    ( CODETABLE, CODETABLENAME, ENCODEVALUE, DECODEVALUE )   AS
      select MOD_NAME_067 as  codeTable     ,
      substr(MOD_NAME_067,1,8) as  codeTableName ,
      cast(rtrim(substr(CMT_INFO_084_1,5,34)) as VARCHAR(34))
           as  encodeValue ,
      concat(substr(CMT_INFO_084_1,39,12) ,
      CMT_INFO_084_2 ) as  decodeValue
FROM  jisdict."MODULE-067" ,
      jisdict."MODCMT-084"
WHERE   "MODULE-MODCMT"
  and   LANG_067 = 'TABLE'
```

```
  and   CMT_ID_084 = -9 and IDD_SEQ_084 = 0
  and   substr(CMT_INFO_084_1,5,1) <> '?'   ;

*+ Status = 0        SQLSTATE = 00000
```

Now I get quite a tidy looking "IDD Code Table Report" – but there are still some somewhat "User Unfriendly" aspects to this display – in particular when we have short Encode values (2 characters as in this case), and short Decode or display values as well.

For example – notice how the output is split because of the default lengths of the table Columns:

```
select codetablename, encodevalue, decodevalue
       from rsqlshr.codectable where codetable = 'USACODEC';

*+ CODETABLENAME  ENCODEVALUE
*+ ------------  -----------
*+ USACODEC       SA
*+ USACODEC       MA
*+ USACODEC       UN
*+ USACODEC
*+
*+ DECODEVALUE
*+ -----------
*+ South Aus
*+ Maine
*+ Unwanted
*+ Blank
*+
*+ 4 rows processed
```

Tailoring an SQL Display of an IDD Table

We can easily fix this with SQL, knowing the specific sizes of the data values in both the Encode and Decode columns, as shown here:

```
select codetablename,
       cast(encodevalue  as  char(2))  as  code,
       cast(decodevalue  as  char(20)) as  value
  from rsqlshr.codectable where codetable = 'USACODEC';

*+ CODETABLENAME  CODE  VALUE
*+ ------------  ----  -----
*+ USACODEC       SA    South Aus
*+ USACODEC       MA    Maine
*+ USACODEC       UN    Unwanted
*+ USACODEC             Blank
*+
*+ 4 rows processed
```

That's nice – for this particular Table. But – you would need to know how and what tailoring to do for other IDD Code Tables so the output could also be tailored to produce such a nice, friendly report.

But **also** - this is just a **static report**. The IDD Table values might be used in Mapping for Encode:Decode purposes – but what if we are developing new applications that won't be using Mapping, and we want to ensure that there is consistency between the names that are displayed in both the old (green screen) and new application?

Here is a "what if" scenario. What if, we defined a View of this specific code Table, the one with the American States, and made it look as though it was a "State table" in an SQL database?

This has some interesting possibilities from an application point of view that are worth exploring. But, first things first.

Data "Hiding" – making an IDD table look like an SQL Table

Let's start by creating a "virtual" State Table – with a View of the underlying IDD Code Table, as follows:

```
CREATE VIEW RSQLSHR.USASTATES
       ( STATE, STATENAME ) AS
select    substr(encodeValue,1,2)     as state  ,
          substr(decodeValue,1,20)    as stateName
  from    rsqlshr.codecTable
 where    codeTable = 'USACODEC'
         ;
*+ Status = 0        SQLSTATE = 00000
```

Notice how we have taken advantage of the existing **codecTable** View – since it already has the knowledge of the "internal" physical characteristics of an IDD Code Table, which we don't want to code again!

Now look at the "report" we can get from SQL showing our American State Table content – with no user tailoring required – and look – it looks just like an SQL State Table! You would never know that it is actually an IDD Code Table.

```
select * from rsqlshr.USAstates;

*+ STATE  STATENAME
*+ -----  ---------
*+ SA     South Aus
*+ MA     Maine
*+ UN     Unwanted
*+        Blank
*+
*+ 4 rows processed
```

Well, because it **looks** like an SQL Table, and **smells** like an SQL Table, **does it also behave like an SQL Table?** Can we do an SQL JOIN with it?

The End Game – Using our SQL State Table

Let's extend our scenario to now display the Offices from the EMPSCHM demo database that comes with IDMS when it is installed. Here is an example of an SQL Select against the underlying network EMPSCHM Office table:

```
select OFFICE_CODE_0450 as Offc,
       OFFICE_STREET_0450 as Street,
       OFFICE_CITY_0450   as City,
       OFFICE_STATE_0450  as St
  from empschm.office
 order by office_city_0450;

*+ OFFC  STREET                CITY            ST
*+ ----  ------                ----            --
*+ 002   567 BOYLSTON ST       BOSTON          MA
*+ 012   734 MASS. AVE         CAMBRIDGE
*+ 005   7690 NEAR SIGHT AVE   GLASSTER        SA
*+ 001   20 W BLOOMFIELD ST    SPRINGFIELD     MA
*+ 008   910 E NORTHSOUTH AVE  WESTON          MA
*+
*+ 5 rows processed
```

Well that's fine as far as it goes – but our users are used to seeing the State Names, not just the State Codes – so how can we easily see the State names, without redundantly creating, populating and maintaining an SQL State Code Table? Well – why don't we take advantage of our existing IDD Code Table and the View we have of it that "looks and smells" like an SQL Table?

Here's what the SQL would look like – and the resultant output:

```
select OFFICE_CODE_0450 as Offc,
       OFFICE_STREET_0450 as Street,
       OFFICE_CITY_0450   as City,
       OFFICE_STATE_0450  as St ,
       StateName
  from empschm.office     ,
       rsqlshr.USAstates
 where office_state_0450 = State
 order by office_city_0450;

*+ OFFC  STREET                CITY            ST  STATENAME
*+ ----  ------                ----            --  ---------
*+ 002   567 BOYLSTON ST       BOSTON          MA  Maine
*+ 012   734 MASS. AVE         CAMBRIDGE           Blank
*+ 005   7690 NEAR SIGHT AVE   GLASSTER        SA  South Aus
*+ 001   20 W BLOOMFIELD ST    SPRINGFIELD     MA  Maine
```

```
*+ 008   910 E NORTHSOUTH AVE  WESTON          MA  Maine
*+
*+ 5 rows processed
```

So there we have it – an IDD Code Table that behaves just like an SQL Table – not too shabby – and all done with IDMS/SQL!

An IDD Code Table that behaves like an SQL Table – but don't tell anybody

I hope that this gives you some new ideas of how you can "package" SQL into Views, and then re-package those Views into higher levels of abstraction for specific purposes – so that you can then create some opportunities for effective, simplified use of SQL to access a variety of IDMS data, which your developers may currently think of as being "too hard" to use in their new applications.

Before You Start – Some Things That Some will Have and Others Will not

You will need, at least, two SQL Schemas in order to carry out the work that is described in this document. I'll describe 3 of them for completeness:

- The **JISDICT** Schema referenced above
  - An SQL Schema that allows you to access the IDD Database
  - Access a Network Schema with SQL
  - Establish this Schema with the following syntax (using the Schema Name referenced in this document):

```
CREATE SCHEMA JISDICT
    FOR NONSQL SCHEMA JISDICT.IDMSNTWK VERSION 1
        DBNAME JISDICT  ;
```

Note: you can **not** CREATE additional components within a "Schema for nonSQL" schema – hence the need for an SQL schema to create Views (Procedures, Functions) which operate against this schema!

- The **RSQLSHR** Schema referenced above
  - An SQL Schema which we use for creating VIEWs in the examples in this document -optionally used for creating "native" IDMS/SQL Tables
  - The syntax is:

```
create Schema RSQLSHR     ;
```

- The **EMPSCHM** Schema
  - An SQL Schema distributed with CA-IDMS and which may optionally be installed during initial installation of IDMS, or at any time afterwards
  - We do not provide the "create" syntax for this schema – check it out in the IDMS Installation Guide