Automated SQL Documentation Using APL2

Rexford H. Swain
Independent Consultant
8 South Street
Washington, Connecticut 06793
USA
Telephone: 203 868 0131; 212 242 5816

ABSTRACT

An application programmer working with APL2 and SQL/DS will often want to investigate and/or document the definitions of SQL tables and views, particularly when working with tables created by others. Unlike conventional APL file systems, SQL "knows" quite a bit about the objects it is storing, but this information is scattered throughout several system catalogs. An APL2 tool which combines and neatly formats available information about a table is presented.

Interpretation of this information may point out conditions which are causing SQL to perform less than optimally. Some issues which influence SQL performance are considered, and some general guidelines for improving performance are suggested.

Some familiarity with APL2 and SQL is assumed.

"SQL" should be understood to mean IBM's SQL/DS for VM systems. But other SQL-based database managers, such as DB2, are very similar.

INTRODUCTION

APL2 and SQL

Presumably, I do not need to preach the many virtues of APL to this audience. But the APL workspace has long been recognized as generally inadequate for mass storage of data that must be shared among users, and entirely inappropriate as a repository for data that must be shared by non-APL applications.

Most older APL systems have included an interface to some external file system, but these were generally insufficient in various ways: not very powerful, unaccessible by non-APL programs, and/or totally unstructured.

SQL, arguably *the* database manager for the 1990's, solves these problems. SQL is extremely powerful and versatile, is accessible by any other program, and imposes a structure that is both second nature to APLers and fast becoming an industry standard.

The main criticism leveled at SQL is poor performance, but APLers are generally comfortable with (or at least accustomed to!) trading off some execution speed for vastly more power and productivity.

APL and SQL work incredibly well together. SQL's row and column orientation is a perfect match for APL's arrays. APL2 gives us the ability to have matrices exactly represent arrays of SQL data with varying datatypes and lengths.

Data Documentation

Most APL installations have workspace documentation tools that display function listings, function calling hierarchies, and global variable references. But similarly meaningful automated documentation of external data files has historically been much more difficult. Conventional "flat" or APL "component" file systems impose little or no structure on data -- typically, the file design is embodied in the accompanying *programs*, so little can be learned from an independent analysis of the database.

An application programmer working with SQL will often want to document the definitions of tables and views. Luckily, SQL "knows" quite a bit about the objects it is storing. But this information is scattered throughout several system catalogs.

The tool presented here, DOCTABLE, looks in the SQL system catalogs for information about a table, its columns, its indexes, and the dbspace that it is in. The results are combined into a compact formatted report. The report is useful for ad hoc reminders, and might also be included in a more formal application documentation package.

SQL Performance

Many programmers, concentrating mainly on application functionality, do not have the luxury of time to learn much about SQL performance. And at many sites, there are no formal database administrators or performance analysts who might be able to help.

The DOCTABLE report may shed some light on potential SQL performance problem areas. While reviewing the contents of the report, we will also discuss how each element may reveal conditions which affect SQL performance.

Some familiarity with these performance issues may help to avoid having application performance problems blamed on APL2 when the actual culprit may be SQL!

THE DOCTABLE TOOL

Utility Functions

It is assumed that the user has established the necessary environment by initializing SQL (with SQLINIT EXEC) and invoking APL2 (including auxiliary processor 127).

Four utility functions are needed to communicate with SQL via AP127 and read from the system catalogs.

Communicating With SQL Via AP127

First, a variable must be shared with AP127. The function SQLSHARE (see Figure 1) shares the variable DAT with AP127. Typically, this function would be invoked by ⎕LX in a workspace using AP127.

Communicating with SQL via AP127 is then quite straightforward -- a small utility function will pass requests to AP127 and retrieve its responses. The function AP127 (see Figure 2) passes its argument to AP127. The first item of the result is always a return code (a five element integer vector).

```
    ∇ SQLSHARE;Q;R
[1]  ⍝ Shares the variable DAT with AP127.
[2]  ⍝ Based on OFFER from IBM's SQL workspace.
[3]
[4]   R←127 ⎕SVO 'DAT'                          ⍝ Make offer
[5]   Q←1 0 1 1 ⎕SVC 'DAT'                      ⍝ Set access control
[6]   →(R=2)⍴0                                  ⍝ Exit if offered
[7]   ⎕ES(R=0)/'Variable DAT could not be offered'
[8]   ⎕SVE←5                                    ⍝ Set event time limit
[9]  L1: ⍝ Offer not accepted yet
[10]  →(2=⎕SVO 'DAT')⍴0                         ⍝ Exit if matched
[11]  →(0≠⎕SVE)⍴L1                              ⍝ Try until time elapsed
[12]  ⎕ES 'Variable DAT offered but not accepted'
    ∇
```

Figure 1

```
    ∇ R←AP127 A
[1]  ⍝ Utility to communicate with AP127 via shared variable DAT
[2]  ⍝ Assumes SQLSHARE has done the ⎕SVO
[3]
[4]   DAT←A
[5]   R←DAT
    ∇
```

Figure 2

Trapping Unanticipated Return Codes

Under most circumstances, the return code from AP127 will be five zeros, indicating that all is well. But a second small utility function is in order so that unanticipated return codes can be conveniently trapped. The function SQLTRAP (see Figure 3) traps any non-zero return code and signals it into the calling environment.

Of course, in a production application environment, a more sophisticated method of error handling would be appropriate; this version is presented for illustrative purposes and is quite abrupt.

Selecting From a Table

Selecting data from the system catalogs with the SQL "SELECT" command requires somewhat more work. The function SELECT (Figure 4) preps the command, opens a cursor, and fetches blocks of rows repeatedly until there are no more, and then automatically commits the logical unit of work (LUW).

Note that SELECT treats two return codes as special cases: 0 1 0 2 100, which means no rows were found; and 0 0 1 0 0, which means that more rows may remain to be fetched. All other non-zero return codes are passed to SQLTRAP.

The number of rows fetched at once by AP127 depends on the current SETOPT setting. This defaults to 20 but should generally be set higher to improve efficiency.

SELECT uses the AP127 "MATRIX" option. Its result is a matrix of depth 2 containing the rows and columns selected. For example, Figure 5 shows a 2 by 3 result.

System Catalogs

SQL maintains many internal tables called "system catalogs" which map the contents of the database. Those that contain information of interest to us are:

SYSTEM.SYSCATALOG	Tables and views
SYSTEM.SYSCOLUMNS	Column definitions
SYSTEM.SYSINDEXES	Index definitions
SYSTEM.SYSDBSPACES	Dbspaces

Any SQL user may use ISQL to browse these tables. (Or, of course, DOCTABLE itself may be used to examine them.) Unfortunately, IBM did not use the SQL "COMMENT" command to add remarks describing the columns of the system catalogs; they are documented in the IBM SQL manuals.

Information about a particular table or view can be found in the first three tables by specifying the creator and table name. A column in the SYSCATALOG table (DBSPACENO) points to the pertinent row in the SYSDBSPACES table.

Updating Statistics

Is it important to note that many of the statistics in the system catalogs may not be entirely accurate. To minimize overhead, SQL does not maintain all of the statistics in real time. For instance, to be certain of the number of rows in a table, you must use "SELECT COUNT(*)" rather than rely on the ROWCOUNT column in the SYSCATALOG table.

The "UPDATE STATISTICS" command should be run periodically to keep these statistics up to date. The SQL optimizer uses these statistics in deciding what strategy to use when accessing table data. The optimizer will be able to make more informed decisions if reasonably current statistics are available. It is particularly important to update statistics after a bulk load of data into a table.

The "UPDATE STATISTICS" command only updates statistics about columns that are involved in indexes. The more thorough "UPDATE ALL STATISTICS" command compiles information about *all* table columns. This should also be run periodically, though less frequently.

Unfortunately, these commands are serious resource hogs and will noticeably affect database performance if run concurrently with application programs. One way to handle this is to schedule a job which runs "UPDATE STATISTICS" every Saturday night, substituting "UPDATE ALL STATISTICS" once a month. Of course, if your table data is not updated very often, a less frequent schedule would suffice.

```
    ∇ SQLTRAP A
[1]  ⍝ Signals an error if a non-zero return code from AP127
[2]
[3]   ⍎(1<≡A)/'A←↑A'                          ⍝ If nested, use first
[4]   →(A≡0 0 0 0 0)⍴0                        ⍝ Ok if five zeros
[5]   ⎕ES 'NON-ZERO RC FROM AP127: ',⍕A       ⍝ Else signal error
    ∇
```

Figure 3

```
    ∇ R←SELECT C;M;N;RC
[1]  ⍝ Simplified utility to perform a SQL SELECT
[2]  ⍝ Usage: R←SELECT 'SELECT ... FROM ... WHERE ... etc...'
[3]  ⍝ Automatically performs COMMIT WORK afterward
[4]  ⍝ Always returns data in AP127 MATRIX format
[5]  ⍝ Accumulates entire SELECT result; beware of WS FULL
[6]  ⍝ Requires functions: AP127 SQLTRAP
[7]
[8]   N←'SQLSEL'                              ⍝ Name of statement
[9]   SQLTRAP AP127 'PREP' N(⍕C)              ⍝ Prep the command
[10]  SQLTRAP AP127 'OPEN' N                  ⍝ Open cursor
[11]  (RC R)←AP127 'FETCH' N 'MATRIX'         ⍝ Fetch, initialize result
[12]  →(RC≡0 1 0 2 100)⍴L2                    ⍝ No rows found?
[13]  →(RC≡0 0 0 0 0)⍴L2                      ⍝ Complete result?
[14]  →(RC≡0 0 1 0 0)⍴L1                      ⍝ More rows coming?
[15]  SQLTRAP RC                              ⍝ Unanticipated return code
[16] L1:                                      ⍝ Try fetching more rows
[17]  (RC M)←AP127 'FETCH' N 'MATRIX'         ⍝ Repeat original fetch
[18]  →(RC≡0 1 0 2 100)⍴L2                    ⍝ There weren't any more?
[19]  R←R,[⎕IO]M                              ⍝ Add to result
[20]  →(RC≡0 0 0 0 0)⍴L2                      ⍝ Was that the end of it?
[21]  →(RC≡0 0 1 0 0)⍴L1                      ⍝ Still more to come?
[22]  SQLTRAP RC                              ⍝ Unanticipated return code
[23] L2:                                      ⍝ Finished fetching
[24]  SQLTRAP AP127 'CLOSE' N                 ⍝ Close cursor
[25]  SQLTRAP AP127 'COMMIT'                  ⍝ Auto commit
    ∇
```

Figure 4

```
      T←'SYSTEM.SYSOPTIONS'
      W←'SQLOPTION > ''R'''
      M←SELECT 'SELECT * FROM' T 'WHERE' W
      DISPLAY M
.→-------------------------------------------------------------.
↓ .→------. .→----. .→------------------------------.          |
| |RELEASE| |2.2.0| |VERSION, RELEASE, MODIFICATION|          |
| '-------' '-----' '------------------------------'          |
| .→---.    .→--.   .→-------------------------------------. |
| |TIME|    |ISO|   |DEFAULT TIME: ISO, JIS, USA, EUR, LOCAL| |
| '----'    '---'   '-------------------------------------' |
'∈-------------------------------------------------------------'
```

Figure 5: A 2 by 3 Result From SELECT

```
      DISPLAY DOCTABLE 'INVENTORY'
.→-------------------------------------------------------------.
↓Table SQLDBA.INVENTORY: Description and quantity for each part|
|  22 rows in 1 page; 0 rows overflowed                         |
|One of 9 tables in PUBLIC dbspace PUBLIC.SAMPLE in pool 1      |
|  5 pages active (1%) of 512 pages total                       |
|  8 header pages, 33% index pages; 5% free; lock mode PAGE     |
|                                                               |
|   Column       Column       Allow  INDEX  Index               |
|#  Name         Type         Nulls  10000   6824  Remarks      |
|                                                               |
|1  PARTNO       SMALLINT       N      U+     o    Part number  |
|2  DESCRIPTION  VARCHAR(24)    Y      o      d+   Description  |
|3  QONHAND      INTEGER        Y      o      2-   Quantity     |
'---------------------------------------------------------------'
```

Figure 6: DOCTABLE Applied to a Table

```
      DISPLAY DOCTABLE 'ORDERS'
.→-------------------------------------------------------------.
↓View SQLDBA.ORDERS: Parts ordered with computed order value  |
|                                                               |
| SELECT NAME,ADDRESS,DESCRIPTION,PRICE*QONHAND                 |
| FROM SUPPLIERS,INVENTORY,QUOTATIONS                           |
| WHERE INVENTORY.PARTNO=QUOTATIONS.PARTNO                      |
| AND SUPPLIERS.SUPPNO=QUOTATIONS.SUPPNO                        |
|                                                               |
|   Column             Column          Allow                    |
|#  Name               Type            Nulls  Remarks           |
|                                                               |
|1  SUPPLIER NAME      CHAR(15)          Y    From SUPPLIERS    |
|2  SUPPLIER ADDRESS   VARCHAR(35)       Y    From SUPPLIERS    |
|3  PART DESCRIPTION   VARCHAR(24)       Y    From INVENTORY    |
|4  PRICE * QONORDER   DECIMAL(15,2)     Y    From QUOTATIONS   |
'---------------------------------------------------------------'
```

Figure 7: DOCTABLE Applied to a View

Displaying a Table or View

The function DOCTABLE (see Appendix) displays the definition of a table or a view. For example, Figure 6 shows the report generated when DOCTABLE is applied to a sample table; Figure 7 shows a view.

ELEMENTS OF THE DOCTABLE REPORT

```
DOCTABLE 'INVENTORY'
```

Invokes the tool. Notice that it is not necessary to specify the creator if there is only one table or view in the database with the specified name. (Had more than one been found, a message would list the possible creators.)

```
Table SQLDBA.INVENTORY:
Description and quantity...
```

The tool has found that INVENTORY is a table and that the creator is SQLDBA. Any comment defined on the table (by the SQL command "COMMENT ON TABLE") is also displayed.

```
22 rows in 1 page
```

This indicates the number of rows in the table and the number of pages occupied by those rows. An SQL page is 4,096 bytes; this gives an idea of the overall size of the table.

```
0 rows overflowed
```

This is a count of the number of rows in the table that have overflowed from their original physical position because VARCHAR or NULL columns grew (after use of the "UPDATE" command) and there was no more room on their original pages.

The data in an overflowed row is divided between two pages, requiring twice the disk I/O to access it. A value of 0 is perfect; higher values indicate more overflow and hence a potentially serious performance problem. A table reorganization (see below) will consolidate overflowed rows.

```
One of 9 tables
in PUBLIC dbspace PUBLIC.SAMPLE
in pool 1
```

The tool has found which dbspace the table is in, how many tables are in it, and which storage pool it is in.

The number of tables in a dbspace can be significant. IBM generally recommends that a large table be created in its own dbspace.

Also, should it ever be necessary to drop the table, the "DROP DBSPACE" command is *much* more efficient than the "DROP TABLE" command.

```
5 pages active (1%)
of 512 pages total
```

This gives an idea of how large the dbspace is, and how close to full it is.

There is an incentive for using large dbspaces, since a dbspace full error will bring an application to a dead halt, and require the acquisition of a new larger dbspace and moving the tables.

However, it is generally not a good idea to have a dbspace that is *much* larger than necessary -- in some circumstances this can mislead the SQL optimizer.

```
8 header pages
```

This indicates the number of pages reserved in the dbspace for header information.

Eight pages is the default when a dbspace is acquired. Once acquired, this parameter cannot be changed. Nor have I ever seen it changed. You could experiment with lower values, but the potential saving in disk space makes it hardly worth the effort.

```
33% index pages
```

This means that a third of the pages in the dbspace are reserved for index data. Most of the remaining two-thirds will be used for row data.

33% is the default when a dbspace is acquired. If you have only one or two indexes on each table in a dbspace, you may be able to get by with a lower index percentage. This would leave more pages for row data and thus require fewer pages overall. The percentage of index pages cannot be changed once a dbspace is acquired, so some advance planning is necessary.

```
5% free
```

This means that 5% of each page in the dbspace will be reserved. 15% is the default when a dbspace is acquired, but this value may be changed at any time with the "ALTER DBSPACE ... (PCTFREE=...)" command.

One of the most common mistakes in SQL database administration is to have a non-zero value here once an application is in production. The idea is to acquire a dbspace with a certain percentage of each page free, load the rows into the table, and then set the percentage to zero. During the load, SQL will reserve the specified percentage of each page, but if the percentage is never reset to zero, SQL will *continue* to maintain that free space, utterly wasting disk space.

After your initial table load, set this percentage to zero -- then when new rows are inserted into the table, SQL will try to use the free space to place the new rows in the pages indicated by their clustering index values. (More about clustering below.)

```
lock mode PAGE
```

This indicates that the dbspace will be locked at the page level. The only other possible locking modes are row and dbspace. This parameter is set when a dbspace is acquired, and can later be changed with the "ALTER DBSPACE" command.

SQL defaults to page locking, and this is generally a good choice. IBM recommends using row level locking only when necessary.

A smaller locking unit is more precise, but requires SQL to do more bookkeeping; it can also cause lock escalations, which can in turn cause deadlocks. A larger locking unit is generally more efficient, but also increases the chance of lock contention.

#

The columns of a SQL table are numbered in the order that they were specified in the "CREATE TABLE" command.

The column numbers determine the sequence that the columns will be returned in when a "SELECT * FROM" command is performed.

Column Name, Column Type

These are the column names and types, as specified in the "CREATE TABLE" command.

Allow Nulls

If the "NOT NULL" clause was used when specifying a column in the "CREATE TABLE" command, an "N" appears here. The default is "Y", meaning that nulls are permitted.

The default is to allow nulls, but SQL generally performs more efficiently if nulls are not permitted.

INDEX 10000, Index 6824

The report includes a column for each index on the table. In this sample table, there are two indexes. The position of two-character codes indicate which table columns are involved in the indexes.

A letter marks the primary column of each index. A "U" indicates a unique index; a "D" means that the index allows duplicates. An upper-case letter indicates that the index is clustered; a lower-case letter indicates an unclustered index. (More about clustering below.) Secondary columns involved in an index are numbered. A "+" indicates ascending order (which is the default); a "-" means descending order. A "o" indicates that the column to the left is not involved in the index.

In this sample table, the "U+" indicates that the first index is a unique index on the PARTNO column in ascending order, and that the index is currently clustered. The "d+" indicates that the second index is a non-unique index on the DESCRIPTION column in ascending order, and that the index is currently not clustered; the "2-" means that the secondary column in the same index is QONHAND, in descending order. In other words, the commands used to create the indexes on this table were:

```
CREATE UNIQUE INDEX ... ON INVENTORY(PARTNO)
CREATE INDEX ... ON INVENTORY(DESCRIPTION,QONHAND DESC)
```

The first index created on a table becomes what is known as the "clustering index". DOCTABLE indicates this special index by displaying the word "INDEX" in upper-case letters in the column heading. (It is possible for a table to have indexes but no clustering index! For example, if the two indexes above were created and then the first was dropped, the table would be left with one index but no clustering index.)

When a "SELECT" command without an "ORDER BY" clause is used, SQL uses the clustering index as the "default" order to return rows in. As new rows are inserted into a table, SQL attempts to keep the rows physically ordered by the clustering index, using any space left by PCTFREE. (Ideally, the clustering index should be a unique index so that the preferred location of each row can be precisely determined.) If there is no more free space available (or if PCTFREE was never lowered), SQL places new rows elsewhere, and thus a table gradually becomes unclustered.

When "UPDATE STATISTICS" is run, SQL notes whether each index order matches the physical order. The verdict is recorded in SYSINDEXES.CLUSTER and is reflected in the upper- or lower-case letter in the DOCTABLE report. (If the index is "clustered", an upper-case "A" or "D" appears; otherwise a lower-case letter is used.)

To improve on this simplistic yes-or-no judgement, SQL/DS version 2.2 introduced another statistic in SYSINDEXES.CLUSTERRATIO that indicates the *degree* to which each index is clustered. This value ranges from 0 (worst) to 10000 (perfect), and appears at the top of each index column in the report.

A table with a poor cluster ratio on the clustering index should be reorganized.

Every table, no matter how small, should have at least one index defined on it.

Every column that is used in a SQL join should have an index defined on it.

Ideally, every table should have at least one unique index defined on it, thus ensuring that each row can be distinguished from all other rows. This also helps SQL to exactly pinpoint the location of a row when the index is used in a query.

The clustering index should be on the column(s) that is (are) most frequently used in "WHERE" and "ORDER BY" clauses.

Remarks

Any column remarks defined for the table will be displayed here. These are set by the "COMMENT ON COLUMN" command.

Table Reorganization

In order to "reorganize" a table, meaning to physically reorder the rows in a table by the clustering index, you must unload all the rows and then reload them in index order. Typically the SQL DBSU (Database Services Utility) is used for this task.

A reorganization will re-cluster the rows and consolidate overflowed rows. This is a complex area that should be approached with caution. Briefly, you will get even better results if you "DROP DBSPACE" in the middle of the process (shadow pages will be recovered) and re-create indexes after reloading the rows.

CONCLUSION

I hope that DOCTABLE is useful to those pursuing the elusive goal of self-documenting databases, if perhaps only as a prototype.

Note that while SQL gives programmers the *structure* for producing self-documenting tables and views, there is no substitute for descriptive information supplied by the designer -- DOCTABLE will yield its best results if the SQL "COMMENT ON" commands have been used to describe tables and columns.

Other APL2 tools for displaying dbspaces and storage pools are available from the author upon request or via the APL91 software exchange.

I also hope that the brief coverage of SQL performance issues will help alleviate some performance problems, or at least raise awareness of this complex but potentially rewarding area.

ACKNOWLEDGEMENTS

I am indebted primarily to Bob Hendricks; and also to Sherm Fowler, Robert Shaw, Joe Heise, and Mike Teitelbaum; for sharing their SQL experiences with me.

REFERENCES

APL2 Programming: Using Structured Query Language; IBM Order Number SH20-9217.

Nancy Wheeler, *APL2 and SQL: A Tutorial*; APL89 Conference Proceedings.

SQL/DS SQL Reference for IBM VM Systems and VSE; IBM Order Number SH09-8087.

SQL/DS Database Administration for IBM VM Systems; IBM Order Number GH09-8083.

SQL/DS System Administration for IBM VM Systems; IBM Order Number GH09-8084.

SQL/DS Interactive SQL Guide and Reference for IBM VM Systems; IBM Order Number SH09-8085.

SQL/DS Database Services Utility for IBM VM Systems; IBM Order Number SH09-8088.

APPENDIX

```
    ∇ R←DOCTABLE TN;C;CH;CR;D;DQ;FMT;H;I;IN;L;PL;PM;Q;S;T;UD;V;W;Y;⎕IO
[1]  ⍝ Show definition of an SQL table or view.
[2]  ⍝ Argument should be a table or view name with optional creator.
[3]  ⍝ Report width is limited to ⎕PW.
[4]  ⍝ Requires function: SELECT
[5]
[6]   ⎕IO←1
[7]
[8]  ⍝ Parse argument
[9]
[10]  TN←⍕∈TN                              ⍝ Force to character vector
[11]  I←(1+⍴TN)|TN⍳'.'                     ⍝ Find creator.tname period
[12]  CR←I⍴TN                              ⍝ Creator (if specified)
[13]  TN←I↓TN                              ⍝ Table name
[14]
[15] ⍝ Get information about table or view
[16]
[17]  W←'TNAME=''',TN,'''',(0≠⍴CR)/' AND CREATOR=''',(¯1↓CR),''''
[18]  T←'SYSTEM.SYSCATALOG'
[19]  S←SELECT 'SELECT * FROM' T 'WHERE' W
[20]
[21] ⍝ Verify that there is exactly one table with name as specified
[22]
[23]  ⎕ES(0∈⍴S)/'Table/view not found: ',CR,TN
[24]  Q←S[;2]                              ⍝ CREATORs
[25]  ⎕ES(1≠⍴Q)/'Specify creator:',3↓∈(⊂' or '),¨(-¯1+(Q=' ')⊥¨1)↓¨Q
[26]
[27] ⍝ Define some formatting utilities
[28]
[29] ⍝ Delete trailing blanks and use double quotes if embedded blanks
[30]  Q←⎕FX 'R←DQ A' 'R←(⌽∨\⌽A≠'' '')/A' '→('' ''∈R)↓0' 'R←1⌽''""'',R'
[31] ⍝ Format integer with commas; use ? if no UPDATE STATISTICS yet
[32]  Q←⎕FX 'R←FMT A' 'R←(1+A≥0)⊃''?''(''555,555,510''⍕A)'
      'R←(∨\R≠'' '')/R'
[33] ⍝ Format number and pluralize word
[34]  Q←⎕FX 'R←N PL A' 'R←(-N=1)↓(FMT N),'' '',A'
[35]
[36] ⍝ Initialize header section of report
[37]
[38]  S←,S                                 ⍝ Just one found
[39]  (CR TN)←S[2 1]                       ⍝ CREATOR, TNAME
[40]  Y←3⊃S                                ⍝ TABLETYPE
[41]  T←('RV'⍳Y)⊃'Table ' 'View '          ⍝ R=table, V=view
[42]  T←T,(DQ CR),'.',DQ TN                ⍝ Creator and table/view name
[43]  T←T,': ',5⊃S                         ⍝ REMARKS
[44]  H←⊂T                                 ⍝ Initialize header
[45]
[46]  →(Y='V')⍴L1                          ⍝ Branch if view
[47]
[48]  T←'  ',S[12]PL 'rows'                          ⍝ ROWCOUNT
[49]  T←T,' in ',(S[13]PL 'pages'),'; '              ⍝ NPAGES
[50]  T←T,(S[15]PL 'rows'),' overflowed'             ⍝ NOVERFLOW
[51]  H←H,⊂T
```

```
[52]
[53] ⍝ Add information about dbspace to header
[54]
[55]  T←'SYSTEM.SYSDBSPACES'
[56]  W←'DBSPACENO=',⍕S[6]                          ⍝ DBSPACENO
[57]  D←,SELECT 'SELECT * FROM' T 'WHERE' W
[58]
[59]  Q←D[5]                                        ⍝ NTABS
[60]  T←(2≤Q)⊃'Only table'('One of ',(⍕Q),' tables')
[61]  T←T,' in ',D[4]⊃'PUBLIC' 'PRIVATE'            ⍝ DBSPACETYPE
[62]  T←T,' dbspace ',(DQ 3⊃D),'.',DQ 1⊃D           ⍝ OWNER, DBSPACENAME
[63]  T←T,' in pool ',⍕D[12]                        ⍝ POOL
[64]  H←H,⊂T
[65]
[66]  T←'  ',(D[11]PL 'pages'),' active '           ⍝ NACTIVE
[67]  T←T,'(',(FMT⌊.5+100×÷/D[11 6]),'%)'           ⍝ NACTIVE÷NPAGES
[68]  T←T,' of ',(D[6]PL 'pages'),' total'          ⍝ NPAGES
[69]  H←H,⊂T
[70]
[71]  T←'  ',(D[7]PL 'header pages'),', '                    ⍝ NRHEADER
[72]  T←T,(⍕D[8]),'% index pages; '                          ⍝ PCTINDX
[73]  T←T,(⍕D[9]),'% free; '                                 ⍝ FREEPCT
[74]  T←T,'lock mode ',('SPT'⍳10⊃D)⊃'DBSPACE' 'PAGE' 'ROW'   ⍝ LOCKMODE
[75]  H←H,⊂T
[76]  →L2                                  ⍝ Skip view definition
[77]
[78] L1: ⍝ View: show its definition
[79]
[80]  T←'SYSTEM.SYSVIEWS'
[81]  W←'VIEWNAME=''',TN,''' AND VCREATOR=''',CR,''''
[82]  V←SELECT 'SELECT * FROM' T 'WHERE' W 'ORDER BY SEQNO'
[83]
[84]  I←'SELECT' 'FROM' 'WHERE' 'AND'   ⍝ Key words for line breaks
[85]  I←I,'OR' 'GROUP BY' 'HAVING' 'ORDER BY' '(SELECT'
[86]  I←' ',¨I,¨' '                     ⍝ Add leading/trailing blanks
[87]
[88]  V←1↓∊' ',¨V[;4]                   ⍝ Combine VIEWTEXT rows
[89]  V←¯1↓V                            ⍝ Drop trailing semicolon
[90]  V←(1+('AS SELECT'≡V)⍳1)↓V         ⍝ Drop CREATE VIEW ... AS
[91]  I←↑∨/I≡¨⊂V                        ⍝ Find any key word
[92]  W←(1++\I)⊂V                       ⍝ Partition by key words
[93]  I←I∨0=⎕PW|¯1+∊⍳¨⍴¨W               ⍝ Fold each partition by ⎕PW
[94]  V←(1++\I)⊂V                       ⍝ Re-partition by keys and ⎕PW
[95]  V←(⊂''),V                         ⍝ Add blank line
[96]  H←H,V                             ⍝ Add to header
[97]
[98] L2: ⍝ Get column information for table or view
[99]
[100] T←'SYSTEM.SYSCOLUMNS'
[101] W←'TNAME=''',TN,''' AND CREATOR=''',CR,''''
[102] S←SELECT 'SELECT * FROM' T 'WHERE' W 'ORDER BY COLNO'
[103]
[104]⍝ Initialize column section of report
[105]
[106] CH←2 1⍴'' '#'                     ⍝ Initialize column headings
```

```
[107] R←S[;,4]                                  ⍝ Init column report: COLNO
[108] CH←CH,' ',¨'Column' 'Name'                ⍝ Add to column headings
[109] R←R,' ',¨S[;1]                            ⍝ Add CNAME to column report
[110] T←S[;5]~¨' '                              ⍝ COLTYPE without blanks
[111] L←S[;6]~¨⊂' ',↑⎕AV                        ⍝ LENGTH "( 6, 2)" w/out blnks
[112] Q←'('≠↑¨L,¨'('                            ⍝ Decimals already have parens
[113] L←(Q⍴¨'('),¨L,¨Q⍴¨')'                     ⍝ Wrap parens around others
[114] CH←CH,' ',¨'Column' 'Type'                ⍝ Add to column headings
[115] R←R,' ',¨T,¨L                             ⍝ Combine, add to report
[116] CH←CH,' ',¨'Allow' 'Nulls'                ⍝ Add to column headings
[117] R←R,(⊂'    '),¨S[;8]                      ⍝ Add NULLS to report
[118]
[119]⍝ Get index information
[120]
[121] →(Y='V')⍴L3                                  ⍝ Skip if view
[122] T←'SYSTEM.SYSINDEXES'
[123] I←SELECT 'SELECT * FROM' T 'WHERE' W      ⍝ Re-use WHERE clause
[124] →(0∈⍴I)⍴L3                                   ⍝ Skip if no indexes
[125]
[126]⍝ Format index information to align with column list
[127]
[128] C←∈I[;7]                                  ⍝ CLUSTERing ('CNFW')
[129] Q←('Index' 'INDEX')[1+C∈'FW']             ⍝ Capitalize clustering index
[130] Q←Q,[0.5]⍕¯5↑¨⍕¨I[;17]                    ⍝ Add CLUSTERRATIOs
[131] CH←CH,' ',¨Q                              ⍝ Add to column headings
[132] UD←∈I[;6]                                 ⍝ INDEXTYPE ('UD') unique/dups
[133] UD←'UDud'[('UD'⍳UD)+2×C∈'WN']             ⍝ Lowercase if not clustered
[134] IN←I[;5]                                  ⍝ COLNAMES ('+A, -B,')
[135] IN←(IN≠' ')⊂¨IN                           ⍝ Nest by column
[136] PM←↑¨¨IN                                  ⍝ Extract +/- sort direction
[137] IN←IN~¨¨⊂⊂'+-,'                           ⍝ Remove commas and +/-
[138] I←IN⍳¨⊂S[;1]                              ⍝ Find column names in indexes
[139] I←⊂¨I                                     ⍝ Need this twice
[140] UD←I⌷¨UD,¨('5'⍕¨1↓¨⍳¨⍴¨IN),¨'o'           ⍝ Extend Unique/Duplicates
[141] PM←I⌷¨PM,¨' '                             ⍝ Extend +/-
[142] I←⍉⊃UD,¨¨PM                               ⍝ Glue together
[143] R←R,(⊂'    '),¨I                          ⍝ Add to report
[144]
[145]L3: ⍝ Combine header with columns table, keep within ⎕PW
[146]
[147] CH←CH,' ',¨'' 'Remarks'                   ⍝ Col heading for remarks
[148] R←R,' ',¨S[;9]                            ⍝ Add column REMARKS
[149] R←CH,[1](⊂''),[1]R                        ⍝ Add column headings
[150] R←1↓[2]⍕R                                 ⍝ Column section of report
[151] H←⊃H,⊂''                                  ⍝ Header section of report
[152] W←⎕PW⌊¯1↑(⍴H)⌈⍴R                          ⍝ Max width, but not > ⎕PW
[153] R←(W↑[2]H),[1]W↑[2]R                      ⍝ Combine sections
    ∇
```