Craig S. Mullins

Return to Home Page

January / February 1993

Technical Support Magazine

A Critical Look at the DB2 Catalog

By Craig S. Mullins

The DB2 Catalog can be considered the brains of DB2. It functions as a knowledge base for all objects defined to DB2. With such an important role to play, it is crucial that the physical design and implementation of the DB2 Catalog is optimized for not only performance, but usability as well. Realistically speaking, how does the DB2 Catalog measure up to the standards of database design? This article will provide some insight, along with several juicy tidbits of DB2 Catalog information that can increase your day-to-day DB2 productivity.

What is The DB2 Catalog ?

The DB2 Catalog is composed of 10 table spaces and 38 tables all within a single database, DSNDB06. Each DB2 Catalog table maintains data about a different aspect of the DB2 environment. In that respect, it functions as a data dictionary for DB2. A data dictionary is a tool that supports and maintains *meta data*, or data about data. The DB2 Catalog supports and maintains data about the DB2 environment. Refer to Figure 1 for a synopsis of the type of information housed in the DB2 Catalog.

Figure 1. ***Type of Information Stored in the DB2 Catalog***

Area	Type of Information
Objects	Stogroups, Databases, Table spaces, Partitions, Tables, Columns, Views, Synonyms, Aliases, Indexes, Index Keys, Foreign Keys, Relationships, Plans, Packages, DBRMs
Security	Database Privileges, Plan Privileges, System Privileges, Table Privileges, View Privileges, Use Privileges
Utility	Image Copy datasets, REORG Executions, LOAD executions, Object Organization Efficiency Information
DB2 Catalog	Links and Relationships between the DB2 Catalog tables

How does it support this data? Well, the tables of the DB2 Catalog, for the most part, can not be modified using standard SQL data manipulation language statements. INSERT statements, DELETE statements, and with a few minor exceptions, UPDATE statements can not be used to modify these tables. Instead, the DB2 Catalog operates as a *semi-active*, *integrated*, and *non-subvertable* data dictionary. Let's define each these three adjectives used to describe the DB2 Catalog in turn.

First, the DB2 Catalog is said to be **semi-active**. Just what does this mean? Well, an active dictionary is built, maintained, and utilized as the result of the creation of the objects which are defined to the dictionary. In other words, as the user is utilizing the intrinsic functions of the DBMS, meta data is being accumulated and populated in the active data dictionary.

The DB2 Catalog, therefore, is active in the sense that whenever standard DB2 SQL is issued, the DB2 Catalog is either updated or accessed. However, the degree to which the information in the DB2 Catalog is maintained is not 100%.

Let's see where the DB2 Catalog operates as an active data dictionary. Remember that there are three types of SQL: DDL, DCL, and DML. When DDL is issued to create DB2 objects such as databases, table spaces, and tables, the pertinent descriptive information is stored in the DB2 Catalog. Whenever a CREATE, DROP, or ALTER statement is issued, information is recorded or updated in the DB2 Catalog. Likewise, with security SQL data control language statements. The GRANT and REVOKE statements cause information to be added or removed from DB2 Catalog tables. Data manipulation language SQL statements will utilize the DB2 Catalog to ensure that the statement accurately references the DB2 objects being manipulated (i.e.. column names, data types, etc).

Why then is the DB2 Catalog classified as only semi-active, and not completely active? There is very important information housed in the DB2 Catalog about the physical organization of DB2 objects. For example, the following type of information is maintained in the DB2 Catalog:

- How many rows are in a given DB2 table? a given DB2 table space?
- How many distinct values are in a given DB2 index?
- What is the physical order of the rows in the table for a certain set of keys?
- Many other types of organizational data are stored in the DB2 Catalog as well.

This information is populated by means of the DB2 RUNSTATS utility. A truly active data dictionary would update this information as data is populated in the application table spaces, tables, and indexes. RUNSTATS would not be needed. This was deemed to be too costly, and for the time-being, rightly so. Therefore, as of now, the DB2 Catalog is only semi-active. When processor speed, memory usage, and external storage devices are optimized to the point where this type of statistical information can be updated on the fly, the DB2 Catalog may then be able to be converted to be completely active.

I also stated that the DB2 Catalog acted as an **integrated** dictionary. By this I mean that DB2 is inoperable without the DB2 Catalog. The DB2 Catalog and the DB2 DBMS are inherently bound together; neither having purpose or function without the other. The DB2 Catalog without DB2 defines nothing; DB2 without the DB2 Catalog has nothing defined upon which it may operate.

The final adjective used to classify the DB2 Catalog is **non-subvertable**. This simply means that the DB2 Catalog is updated as DB2 is used by standard DB2 features. It is not possible to update the DB2 Catalog behind DB2's back. For example, assume that I create a table with twenty columns. I could not subsequently update the DB2 Catalog to indicate that the table had fifteen columns instead of twenty, without using standard DB2 data definition language SQL statements to drop and re-create the table as desired. The DB2 Catalog and the object that it defines are always in sync.

An Exception to the Rule

As with most things in life, there are exceptions to the basic rule that SQL data manipulation language can not be used to modify DB2 tables. It is possible to modify certain columns used by the DB2 optimizer pertaining to the physical organization of table data. These columns are outlined in Figure 2.

Figure 2. ***Updateable DB2 Catalog Columns***

DB2 Catalog Table	Column	Description	Update?	Used by Optimizer
SYSIBM.SYSCOLUMNS	LOW2KEY	second lowest value for the column	Y	Y
	HIGH2KEY	second highest value for the column	Y	Y
	COLCARD	number of distinct values for the column	Y	Y
	FOREIGNKEY	indicates if column contains BIT DATA	Y	N
SYSIBM.SYSFIELDS	EXITPARM	non-uniform distribution column value	N	Y
	EXITPARML	(percentage * 100) that the value contained in EXITPARM exists within the column	N	Y
SYSIBM.SYSINDEXES	CLUSTERRATIO	percentage of rows in clustered order	Y	Y
	CLUSTERED	whether or not the table space is clustered	N	Y
	FIRSTKEYCARD	number of distinct values for the first column of the index key	Y	Y
	FULLKEYCARD	number of distinct values for full index key	Y	Y
	NLEAF	number of active leaf pages	Y	Y
	NLEVELS	number of index b-tree levels	Y	Y
SYSIBM.SYSTABLES	CARD	number of rows for a table	Y	Y
	NPAGES	number of pages used by the table	Y	Y
	PCTPAGES	percentage of table space pages that actually contain rows for this table	Y	Y
SYSIBM.SYSTABLESPACE	NACTIVE	number of allocated table space pages	N	Y

It is important to note that none of these columns relate to the actual definition of any DB2 objects. They are statistical columns and DB2 permits them to be updated only to enable analysts to influence the DB2 optimizer to enhance SQL performance.

The Benefits of an Active Catalog

The presence of an active catalog is a boon to the DB2 developer. The DB2 Catalog is always synchronized to each actual application database. One can be assured, therefore, that information retrieved from the DB2 Catalog is 100% accurate. Since the DB2 Catalog is composed of DB2 tables (albeit modified somewhat for performance), it is possible to query these tables using standard SQL. The hassle of documenting physical database structures is handled by the active DB2 Catalog and the power of SQL. A whole slew of useful DB2 Catalog queries can be issued by DB2 users to determine such things as table structure, column data types, database organization, and just about anything else you may need to know about DB2 objects.

DB2 Catalog Structure

The DB2 Catalog is structured as DB2 tables. However, they are not standard DB2 tables. In reality, many of the DB2 Catalog tables are tied together hierarchically, not unlike an IMS database. This is accomplished by means of a special type of relationship, called a *link* in DB2 terminology. A link can be thought of as a type of parent-child relationship. These links physically exist between rows within the DB2 Catalog tables. One can determine the nature of these links by querying the DB2 Catalog table, SYSIBM.SYSLINKS. This single DB2 Catalog table stores the pertinent information defining the relationships between other DB2 Catalog tables. To view this information issue the following SQL statement:

```
SELECT      PARENTNAME, TBNAME, LINKNAME,
            CHILDSEQ, COLCOUNT, INSERTRULE
FROM        SYSIBM.SYSLINKS
ORDER BY    PARENTNAME, CHILDSEQ
```

This information can be used to construct the physical composition of the DB2 Catalog links. To accomplish this keep the following rules in mind:

The PARENTNAME is the name of the superior, or parent table in the hierarchy.

TBNAME is the name of the subordinate, or child table in the hierarchy.

The CHILDSEQ and COLCOUNT columns refer to the clustering and ordering of the data within the relationship.

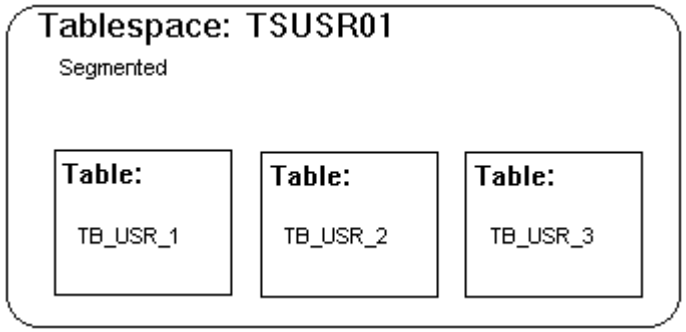
The INSERTRULE determines the order in which data will be inserted into the relationship. This concept is very similar to the insert rule that is defined for IMS databases. Valid insert rules are shown in Figure 3.

Figure 3. ***DB2 Link Insert Rules***

Insert Rule	Meaning	Description
F	FIRST	Insert new data values as the first data value for the relationship.
L	LAST	Insert new data values as the last data value for the relationship.
O	ONE	Permit only one data value for the relationship.
U	UNIQUE	Do not allow duplicate data values for the relationship.

So, to understand how links work, let's examine the DB2 Catalog link between SYSIBM.SYSTABLESPACE and SYSIBM.SYSTABLES. The link named DSNDS#DT defines this relationship. DB2 will store pointers within the rows of these tables to maintain this relationship. Consider the table space and tables as shown in Figure 4.

Figure 4. ***Tables Defined to a Table space***

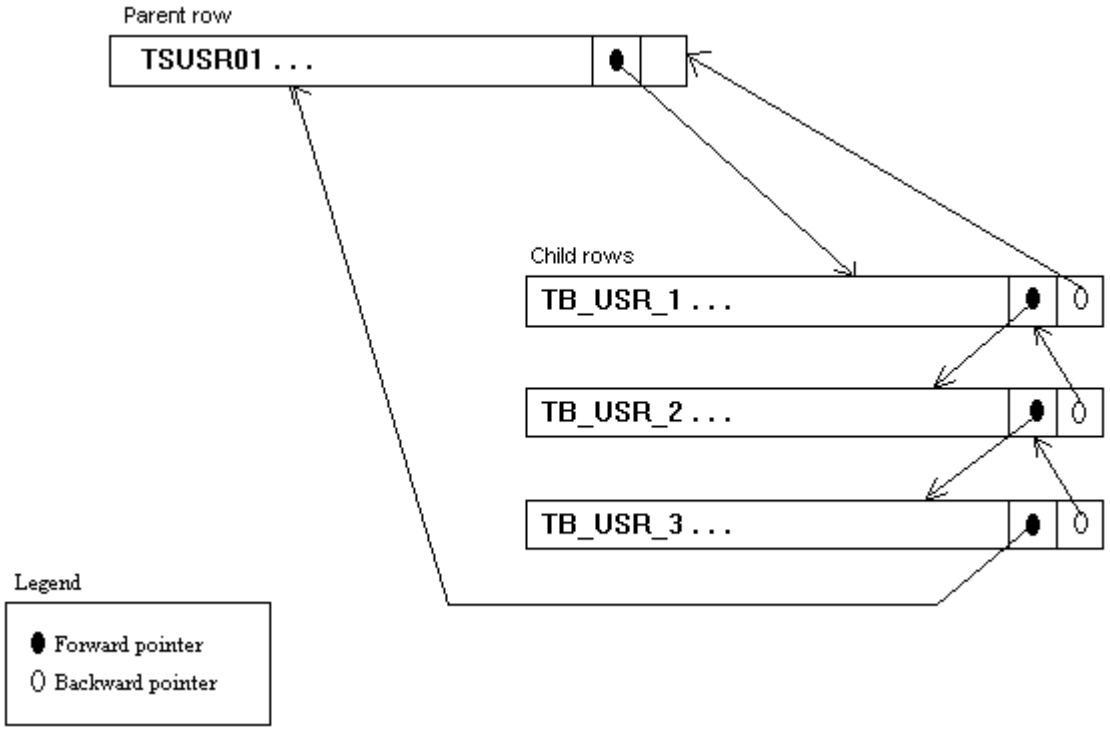


A user defines a new table space with three new tables. This will be recorded in the appropriate DB2 Catalog tables, SYSTABLESPACE and SYSTABLES. In addition, each row inserted into the DB2 Catalog tables will also contain forward and backward pointers defining the link. There are no link pointers stored in the user

table rows. A forward pointer moves forward in the link and a backward pointer moves backward in the link.

To more clearly illustrate let's return to our example. The table space row in SYSTABLESPACE for TSUSR01 will contain a forward pointer to the first table row, in this instance, for TB_USR_1. Each row in SYSTABLES will contain a forward pointer to the next table and a backward pointer to the previous table. The backward pointer for the first table row and the forward pointer for the last table row point to the parent row. Refer to Figure 5 for a pictorial representation of this.

Figure 5. ***DB2 Link Pointers***



These pointers define the links that enable DB2 to efficiently process DDL. For example, without explicit pointers, an operation like dropping a table space would be much more inefficient than it is today. Instead of traversing the link list using the pointers to drop the table space and all subordinate objects (such as tables, columns, foreign keys, views, indexes, etc), DB2 would have to scan each table space or, at best, use available indexes.

DB2 Catalog Indexes

Did you know that DB2 indexes are not used by internal DB2 operations? For example, when the BIND command queries the DB2 Catalog for syntax checking and access path selection, only the internal DB2 Catalog links will be utilized. Indexes on the DB2 Catalog can be used when SQL SELECT statements are issued to query indexed tables in the DB2 Catalog. To enhance the performance of your DB2 Catalog queries, be sure to always include predicates containing indexed columns whenever possible.

The Physical Design of the DB2 Catalog

Now that we understand the basic make-up of the DB2 Catalog, let's examine its design. Although a very credible job of database design was performed for a relational system that had its beginnings in the 1970's[1], the DB2 Catalog is not without its faults. The actual physical definition of the DB2 Catalog leaves much to be desired in certain key areas. Design flaws exist in the areas of normalization, data type consistency, redundant data and semantic issues.

Insufficient Normalization

The bare minimum for implementation of a relational table is first normal form (1NF), which is defined in Figure 6. This is important in order for the relational operators to function properly when used against a table. According to the definition in Figure 6, several DB2 Catalog tables are not in 1NF.

Figure 6. ***First Normal Form***

> An entity is said to be in first normal form (1NF) if, and only if, all of its attributes are atomic (single-valued), unique in meaning, and non-repeating.

For example, as of DB2 V2.3, the SYSIBM.SYSFIELDS table is not in first normal form because it contains non-atomic data. Some rows contain information about field procedures, whereas other rows contain non-uniform distribution statistics (NUDS). The contents of the row is determined by the data values contained in specific columns of the table. This is a severe design flaw. What makes it even worse is that this anomaly did not occur until DB2 V2.2, when non-uniform distribution statistics were first stored by DB2. The proper response would have been to create a new table where these values could be stored, leaving SYSIBM.SYSFIELDS to record the field procedure information. With DB2 Version 3, the most recent release of DB2, this situation has been rectified by placing the NUDS into separate tables instead of SYSIBM.SYSFIELDS. These tables are SYSIBM.SYSCOLDIST and SYSIBM.SYSCOLDISTSTATS.

Another violation of first normal form exists within the SYSIBM.SYSCOPY table. Volume serial numbers for image copy datasets are strung together as a repeating group within a single column, DVOLSER. This *is* a design flaw, although a minor one, because users rarely need to query these values.

Finally, although it is probably nitpicking, all of the DB2 Catalog tables which store DB2 authority are not truly in first normal form. Repeating groups of authorization columns exist in each of these tables. Though not quite as bad as storing multiple values in a single column as occurs in SYSIBM.SYSCOPY, technically speaking, this is also a violation of first normal form.

Lack of Data Type Consistency

The relational model provides for the very useful concept of domains. It is generally accepted that physical database design practice follow the domain concept whenever possible (given the technical limitations of the database being used). Minimally, to mimic domain usage with DB2, the database designer should standardize their design, using specific data types and lengths for specific domains of data. The DB2 Catalog does not adhere to this rule of thumb as well as it should.

For example, tables containing columns defined as date, time, and timestamp data types are defined inconsistently throughout the DB2 Catalog. Consult Figure 7 for a complete listing of all the different date and time formats used in the DB2 Catalog. Notice that the DB2 Catalog never stores dates in DB2's date format and never stores time data in DB2's time format. Even stranger, there are three different timestamp formats used: DB2's TIMESTAMP data type, an eight character internal timestamp format, and a twelve character internal timestamp format. This sure makes it difficult to query, compare, and record DB2 date, time, and timestamp data.

Figure 7. ***Date and Time Data Types in the DB2 Catalog Tables***

DB2 Catalog Table	Column	Data Type
SYSIBM.SYSCOLAUTH	DATEGRANTED	CHAR(6) yymmdd
	TIMEGRANTED	CHAR(8) hhmmssth
SYSIBM.SYSCOPY	TIMESTAMP	TIMESTAMP
SYSIBM.SYSDATABASE	TIMESTAMP	TIMESTAMP
SYSIBM.SYSDBAUTH	DATEGRANTED	CHAR(6) yymmdd
	TIMEGRANTED	CHAR(8) hhmmssth
SYSIBM.SYSDBRM	TIMESTAMP	CHAR(8) FOR BIT DATA (int)
	PRECOMPTIME	CHAR(8) hhmmssth

	PRECOMPDATE	CHAR(6) yymmdd
SYSIBM.SYSPACKAGE	TIMESTAMP (created)	TIMESTAMP
	BINDTIME (last bound)	TIMESTAMP
	PCTIMESTAMP (precompile)	TIMESTAMP
SYSIBM.SYSPACKAUTH	TIMESTAMP	TIMESTAMP
SYSIBM.SYSPACKLIST	TIMESTAMP	TIMESTAMP
SYSIBM.SYSPLAN	BINDDATE	CHAR(6) yymmdd
	BINDTIME	CHAR(8) hhmmssth
SYSIBM.SYSPLANAUTH	BINDDATE	CHAR(6) yymmdd
	BINDTIME	CHAR(8) hhmmssth
	TIMESTAMP	CHAR(12) FOR BIT DATA (int)
SYSIBM.SYSRELS	TIMESTAMP	TIMESTAMP
SYSIBM.SYSRESAUTH	TIMESTAMP	CHAR(12) FOR BIT DATA (int)
	DATEGRANTED	CHAR(6) yymmdd
	TIMEGRANTED	CHAR(8) hhmmssth
SYSIBM.SYSSTOGROUP	SPCDATE	CHAR(5) yyddd
SYSIBM.SYSTABAUTH	DATEGRANTED	CHAR(6) yymmdd
	TIMEGRANTED	CHAR(8) hhmmssth
	TIMESTAMP	CHAR(12) FOR BIT DATA (int)
SYSIBM.SYSUSERAUTH	DATEGRANTED	CHAR(6) yymmdd
	TIMEGRANTED	CHAR(8) hhmmssth
	TIMESTAMP	CHAR(12) FOR BIT DATA (int)

Reliance on Links

Though the concept of links is not overtly negative, the query impact of wholesale reliance upon links can be extremely negative. Simply stated, every relationship that is physically implemented with a link should be logically represented in the DB2 Catalog. Let's take a look at an example of over-reliance upon links.

DB2 records information about storage groups in two DB2 Catalog tables: SYSSTOGROUP and SYSVOLUMES. A link, namely DSNSS#SV, records the relationship between these two tables. There is one row in SYSSTOGROUP for

each DB2 storage group, and one to many rows in SYSVOLUMES for each DASD volume assigned to the storage group.

So far, so good. However, there is an ordering to the volumes assigned to the storage groups that can never be ascertained simply by querying these two tables. When a DB2 storage group is created, the volumes are inserted, in order, into SYSVOLUMES, according to the definition of the DSNSS#SV link. This link is defined as shown below:

CHILD	COL	INSERT
PARENTNAME	TBNAME	LINKNAME
SEQ	COUNT	RULE
SYSSTOGROUP	SYSVOLUMES	DSNSS#SV
1	0	L

So, we can see that with an insert rule of L, volumes will be inserted such that the volumes physically listed first in the DDL will be inserted into the DB2 Catalog first on the linked list. Subsequent volumes will be inserted at the end of the link list. But this list will be traversed using the link only when DB2 is performing its internal operations. So when an object is assigned to a STOGROUP, DB2 will traverse the linked list in the order that it was created to assign the object to a volume. But, when an end user issues a query against the SYSVOLUMES, the volumes will not be returned in any specific order. The order simply can not be determined using SQL. The solution would be to add a SMALLINT column, say VOLNO, storing the order that is physically maintained by the linked list. This will allow end users to "know" what DB2 "knows."

Redundant Data

It is also a good design practice to avoid redundant data. When redundant data is stored, two possible negative ramifications may occur:

1. Users of the data may become confused as to the difference between the redundant data items, possibly ignoring or by-passing the useful data

2. The redundant data items may not be kept synchronized, further complicating not only end user access, but also the integrity of the database

Several tables contain redundant data that is probably not required even for performance reasons. For example:

- The SYSIBM.SYSSTMT and SYSIBM.SYSDBRM tables carry both the PLNAME and PLCREATOR column when PLNAME is sufficient to identify a plan (as plan names are unique within a DB2 subsystem). There is really no logical reason to carry the PLCREATOR column in the SYSIBM.SYSSMT table (however, there may be a technical implementation or performance reason that I am not aware of).

- The SYSIBM.SYSCOPY table and all of the authority tables (i.e.. SYSPLANAUTH, SYSDBAUTH, etc.) carry both a TIMESTAMP columns as well as a DATE and TIME column. This is unnecessary as a timestamp will suffice to uniquely identify a point in time. To be fair though, the documentation states that the DATE and TIME columns should not be used as they will be eliminated in a future release of DB2.

- Data redundancy is also created by the manner in which views are stored in the DB2 Catalog. Two tables (SYSIBM.SYSVTREE and SYSIBM.SYSVLTREE) are used to hold the view parse tree, when one properly designed table would have sufficed.

Semantics

There is a semantic problem with the DB2 Catalog in that the tables and columns are not functionally named. The name of a table should convey the contents of its data. Several good examples of the non-intuitive manner in which DB2 Catalog tables are named follow:

- SYSIBM.SYSVIEWS does not contain one row per DB2 view, but instead may contain multiple rows per view. SYSIBM.SYSVTREE contains the parse tree for the view but will only contain one row per each view. Long parse trees are handled by adding rows to an overflow table named SYSIBM.SYSVLTREE. This is anything but obvious given the table names.

- SYSIBM.SYSKEYS contains one row for every column of every DB2 index. It does not contain primary or foreign key information as could easily be misconstrued given the table's name.

- Several columns are also inappropriately named. For example who would guess that STNAME actually refers to a view name or that DNAME refers to a plan name.

I also contend that the column names throughout the DB2 Catalog should be named much more uniformly. Why, for example, is the column for the table name attribute called NAME in SYSIBM.SYSTABLES, TBNAME in SYSIBM.SYSCOLUMNS, and TNAME in SYSIBM.SYSCOLAUTH? Surely a more standardized and uniform column naming convention could have been used.

You might want to consider creating views of all DB2 Catalog tables for ad hoc access. These views could contain standardized, more user-friendly names for all DB2 Catalog tables and columns.

DB2 Catalog Performance Hints

Design considerations aside, we must learn to make the best of the DB2 Catalog as it is currently implemented. The final section of this article will present a series of performance tuning tips and techniques for the DB2 Catalog.

Plans and Packages

Use packages instead of multiple DBRMs bound into one plan, especially when you have a DBRM for a common routine that is referenced by many programs; this will reduce the size of SYSIBM.SYSSTMT.

Be sure to free old versions of packages that will never be used; a general rule of thumb is to free the version when the load module no longer exists (or can no longer be restored). Every version of every package is stored in the DB2 Catalog until it is either freed or dropped. Failure to clean up old package versions will result in an ever-expanding DSNDB06.SYSPKAGE table space.

FREE all unused plans and packages; when a program will no longer be executed, free its plan or package. This will reduce the size not only of the DB2 Catalog but also of the DB2 Directory SCT01 and SPT02 "tables".

Views

Do not use one view per base table. This provides no program insulation as commonly believed, but does increase the size of four DB2 Catalog tables. These are SYSIBM.SYSVIEWS, SYSIBM.SYSVIEWDEP, SYSIBM.SYSVTREE, and SYSIBM.SYSVLTREE.

Drop any unused views. This can be determined by querying the DB2 Catalog SYSIBM.SYSPLANDEP and SYSIBM.SYSPACKDEP tables. If a given view does not appear in either of these tables, then no static SQL accesses the view. Be careful though not to drop views being queried by ad hoc users or programs containing dynamic SQL. The following query can be used to determine if any QMF queries access a given table:

```
SELECT     DISTINCT OWNER, NAME, TYPE
FROM       Q.OBJECT_DATA
WHERE      APPLDATA LIKE '%viewname%';
```

Image Copies

Run the MODIFY RECOVERY utility to remove old image copies from the SYSIBM.SYSCOPY table. I have seen shops that never executed this utility, thereby maintaining a list of every image copy ever taken since the DB2 subsystem was established. How would you like to be the DBA performing a RECOVER using an image copy from four years ago? If you do nothing else suggested in this article develop a procedure for keeping up-to-date image copies and purging image copies that can never reasonably be expected to be used.

Authority

- DB2 allows duplicate entries in all of the authority tables. For example, issuing the following two SQL statements will cause two rows recording that USER1 has execute authority on PLAN1 to be inserted into SYSIBM.SYSPLANAUTH.

  ```
  GRANT EXECUTE ON PLAN1 TO USER1;

  GRANT EXECUTE ON PLAN1 TO USER1;
  ```

- Although DB2 should probably monitor for this condition and prohibit duplicate rows, it is a fact of life that it currently does not. For this reason use due diligence to avoid duplicate authorization granting. For example, avoid automatically granting plan and/or package execution authority immediately after binding.

- Reduce the size of the authority tables by granting multiple authorizations using a single SQL GRANT statement. Every GRANT execution causes at least one new row to be inserted into an authority table. By judiciously coding your GRANT statements, you can avoid undue authority table growth. For example, the two scenarios in Figure 8 are functionally equivalent, but scenario 1 will cause two rows to be inserted into SYSIBM.SYSPLANAUTH, one for each GRANT. Scenario 2 will cause only one row to be inserted, recording both authorities in a single row.

Figure 8. ***DB2 Authority Scenarios***

Scenario 1	**Scenario2**
GRANT EXECUTE ON PLAN1 TO USER1;	GRANT BIND, EXECUTE ON PLAN1 TO USER1;
GRANT BIND ON PLAN1 TO USER1;	

General Ideas

Drop all unused objects. This includes all unused STOGROUPs, databases, table spaces, tables, views, aliases, synonyms, and indexes.

Free all unused plans and free or drop all unused packages.

Consider removing the distribution statistics from SYSFIELDS for evenly distributed tables. This can be accomplished with the MODIFY STATISTICS utility.

Do not forget to execute RUNSTATS on your DB2 Catalog table spaces. These table space are listed below:

DSNDB06.SYSCOPY	DSNDB06.SYSDBASE
DSNDB06.SYSDBAUT	DSNDB06.SYSGPAUT
DSNDB06.SYSGROUP	DSNDB06.SYSPKAGE
DSNDB06.SYSPLAN	DSNDB06.SYSSTR

DSNDB06.SYSUSER DSNDB06.SYSVIEWS

For DB2 Version 3, the SYSIBM.SYSSTATS table space has been added. It contains the statistically-oriented DB2 Catalog tables.

> Although the DB2 Catalog table spaces can not be reorganized without reallocating them, the indexes on the DB2 Catalog tables can be reorganized using the RECOVER utility. Be sure to monitor all DB2 catalog indexes and reorganize them using RECOVER whenever necessary.

Periodically produce DB2 Catalog reports to assist your clean up efforts.

Synopsis

Hopefully this article has presented some food for thought in terms of what the DB2 Catalog should be and how to best optimize your usage of the DB2 Catalog as it currently exists. The DB2 Catalog is a very useful tool for all users of DB2; use the information and techniques contained in this article to make your usage of the DB2 Catalog more efficient and to make your day more productive!

---

[1] DB2 is based on the relational prototype, System R, which IBM initiated in the early 1970's.

From Technical Support, Jan and Feb 1993.

Home.