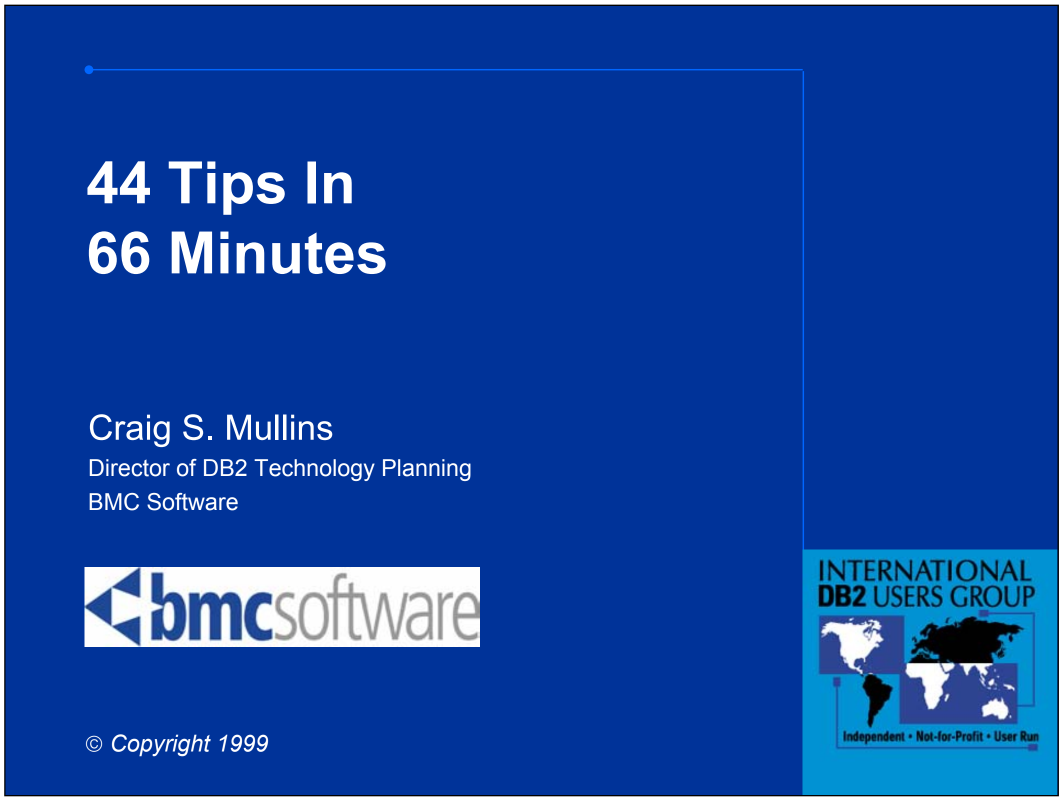
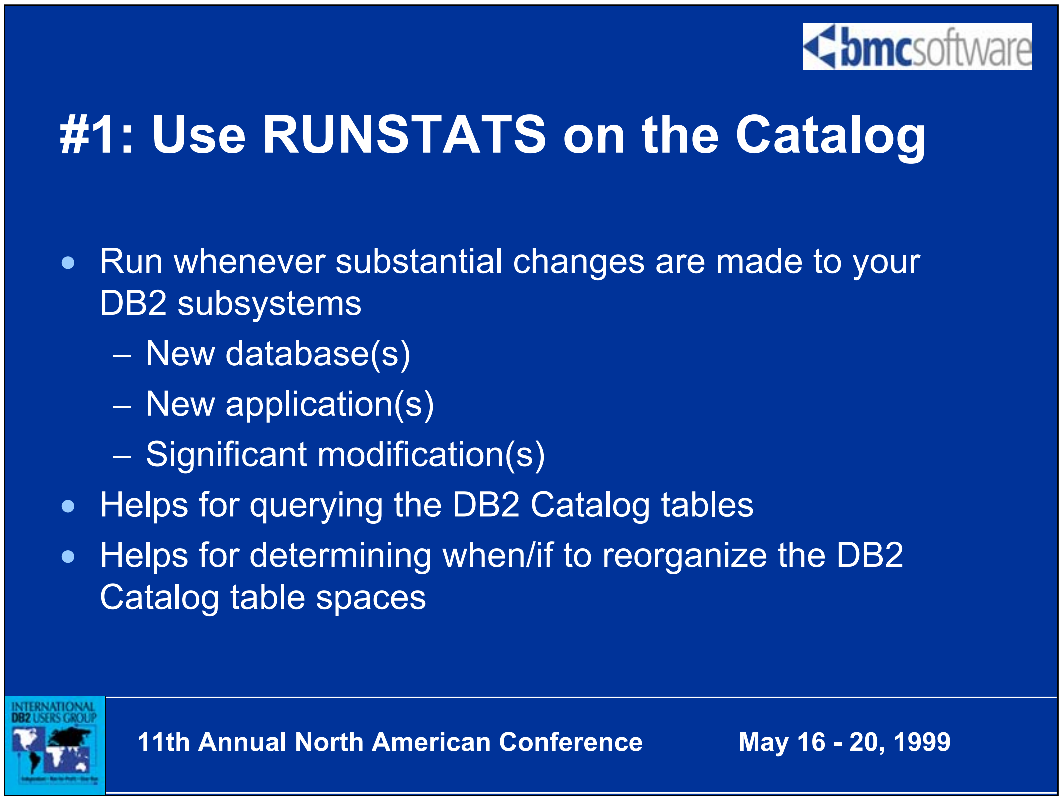


44 Tips In 66 Minutes

Craig S. Mullins
Director of DB2 Technology Planning
BMC Software

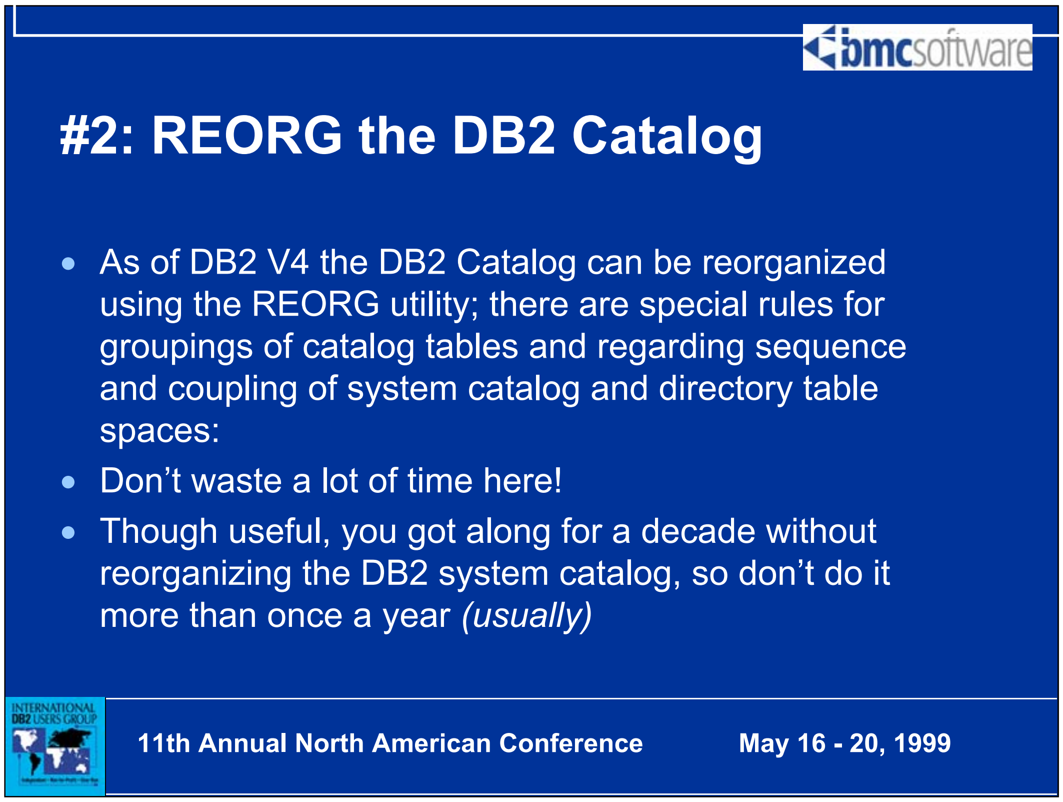


#1: Use RUNSTATS on the Catalog

- Run whenever substantial changes are made to your DB2 subsystems
  - New database(s)
  - New application(s)
  - Significant modification(s)
- Helps for querying the DB2 Catalog tables
- Helps for determining when/if to reorganize the DB2 Catalog table spaces

Although it always has been a wise course of action to execute RUNSTATS on the DB2 Catalog table spaces, it becomes even more important as of DB2 V4 because the DB2 system catalog table spaces can be reorganized.

The RUNSTATS utility collects statistical information that is used by the optimizer to generate access paths. Additionally, these statistics can be analyzed to determine when a REORG should be run. When RUNSTATS is run for a catalog table space, the statistics about that system catalog table space are gathered and then stored the DB2 Catalog tables themselves!



#2: REORG the DB2 Catalog

- As of DB2 V4 the DB2 Catalog can be reorganized using the REORG utility; there are special rules for groupings of catalog tables and regarding sequence and coupling of system catalog and directory table spaces:
- Don’t waste a lot of time here!
- Though useful, you got along for a decade without reorganizing the DB2 system catalog, so don’t do it more than once a year *(usually)*

11th Annual North American Conference **May 16 - 20, 1999**

To determine when to reorganize the system catalog, DBAs can use the same basic indicators used to determine whether application table spaces should be reorganized:

COLUMN	CATALOG TABLE	OBJECT	IMPACT
NEAROFFPOS	SYSIBM.SYSINDEXPART	TABLE SPACE	+
FAROFFPOS	SYSIBM.SYSINDEXPART	TABLE SPACE	++++
CLUSTERRATIO	SYSIBM.SYSINDEXES	INDEX	- - - - -
NEARINDREF	SYSIBM.SYSTABLEPART	INDEX	+
FARINDREF	SYSIBM.SYSTABLEPART	INDEX	++++
LEAFDIST	SYSIBM.SYSINDEXPART	INDEX	+++

The column and table name where the statistic can be found is given in the first two columns of the chart. The third column indicates whether the statistic is applicable for a table space or an index. The fourth column gives an indication of the impact of the statistic. A plus (+) sign indicates that you should REORG more frequently as the value in that column gets larger. A minus (-) sign indicates that you should REORG more frequently as the value gets smaller. As the number of “+” or “-” signs increases, the need to REORG becomes more urgent.

For the SYSDBASE, SYSVIEWS and SYSPLAN catalog table spaces, the value for the FAROFFPOS and NEAROFFPOS columns of SYSINDEXPART can be higher than for other table spaces before they need to be reorganized.

(usually)

#3: More Catalog REORG Scenarios

- To reclaim space and size table spaces appropriately when DB2 catalog and directory data sets are not using a significant portion of their allocated disk space (PRIQTY).
- When it is necessary to move the DB2 catalog and directory to a different device.
- When the DB2 catalog and directory data sets contain a large number of secondary extents.

11th Annual North American Conference May 16 - 20, 1999

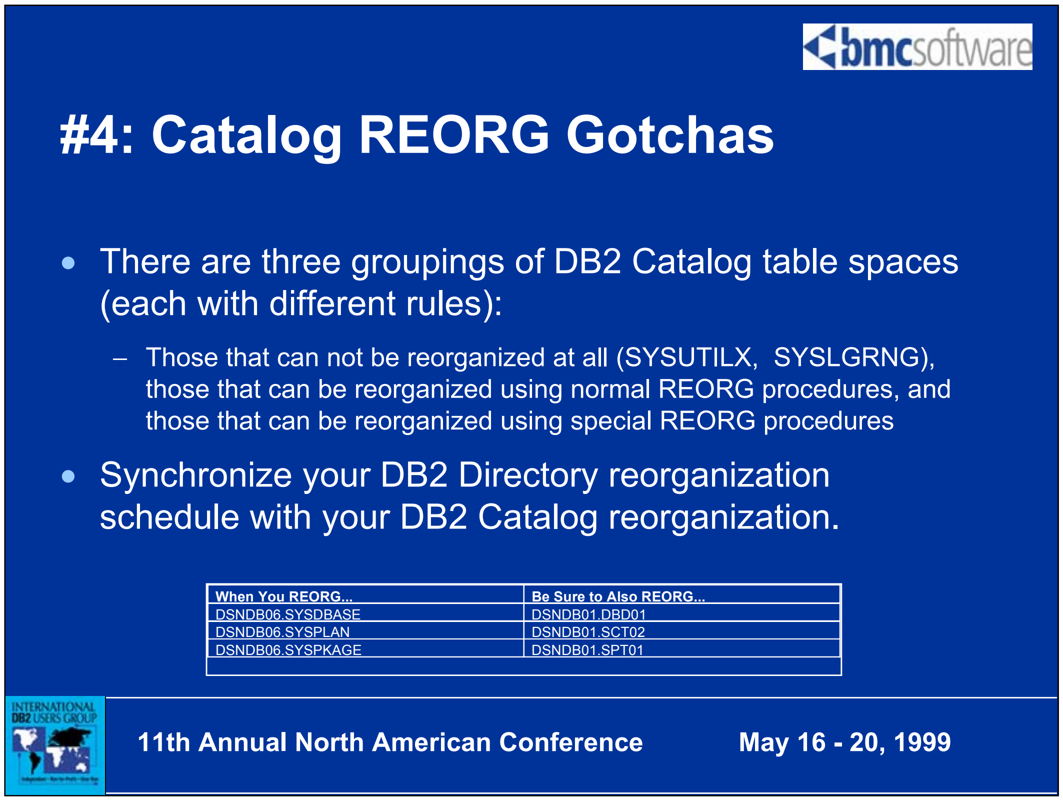
Consider catalog and directory reorganization in the following situations:

- To reclaim space and size table spaces appropriately when DB2 catalog and directory data sets are not using a significant portion of their allocated disk space (PRIQTY).
- When it is necessary to move the DB2 catalog and directory to a different device.
- When the DB2 catalog and directory data sets contain a large number of secondary extents.

When reorganizing the DB2 Catalog (DSNDB06) and DB2 Directory (DSNDB01) table spaces the following options can not be used:

- The UNLOAD ONLY option is not permitted.
- The LOG YES option is not permitted as image copies are explicitly required following a DB2 Catalog and/or DB2 Directory reorganization.

Also, the reorganization of two specific table spaces are treated differently than any other in the manner in which the are tracked by DB2. DB2 records the reorganization of the DSNSB06.SYSCOPY and DSNDB01.DBD01 table spaces in the log instead of SYSCOPY.



#4: Catalog REORG Gotchas

- There are three groupings of DB2 Catalog table spaces (each with different rules):
  - Those that can not be reorganized at all (SYSUTILX, SYSLGRNG), those that can be reorganized using normal REORG procedures, and those that can be reorganized using special REORG procedures
- Synchronize your DB2 Directory reorganization schedule with your DB2 Catalog reorganization.

When You REORG...	Be Sure to Also REORG...
DSNDB06.SYSDBASE	DSNDB01.DBD01
DSNDB06.SYSPLAN	DSNDB01.SCT02
DSNDB06.SYSPKAGE	DSNDB01.SPT01

INTERNATIONAL DB2 USERS GROUP

11th Annual North American Conference May 16 - 20, 1999

The following table spaces are those that the REORG utility processes as it would any other table space:

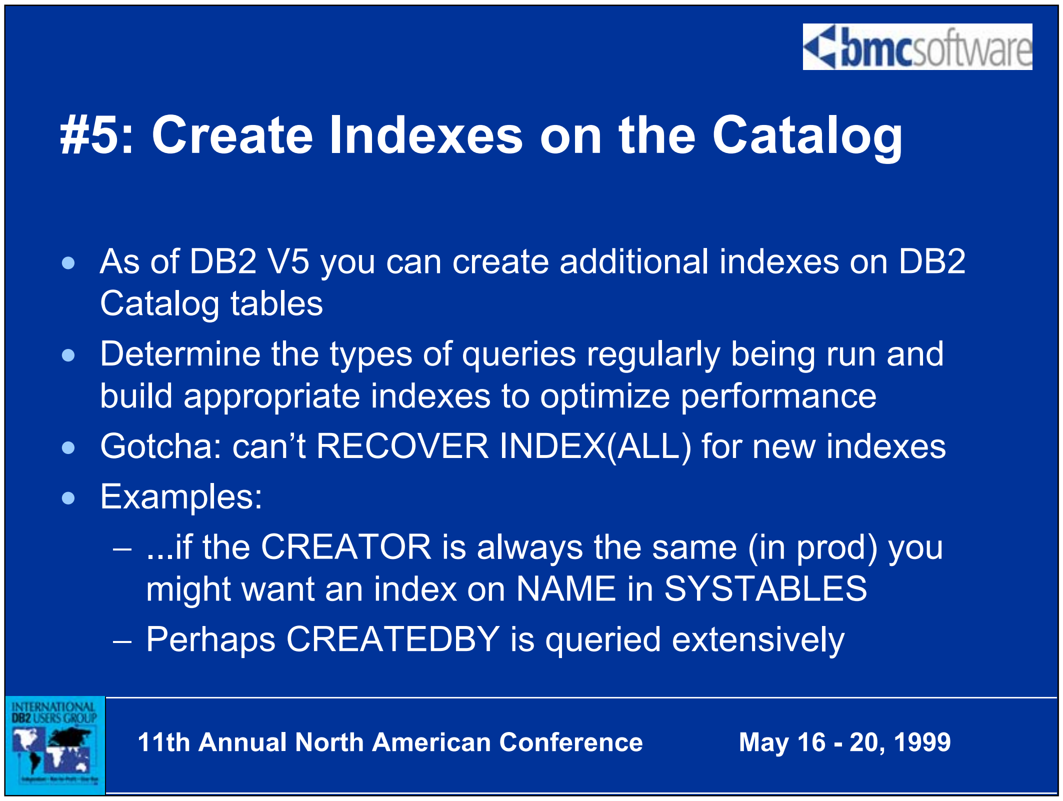
DSNSB06.SYSCOPY, DSNSB06.SYSGPAUT, DSNSB06.SYSPKAGE, DSNSB06.SYSSTATS, DSNSB06.SYSSTR, DSNSB06.SYSUSER, DSNSB01.SCT02, DSNSB01.SPT01

The third, and final grouping of table spaces, must be processed differently than other table spaces. These six table spaces require special “handling and care” because they have a different internal configuration than other table spaces.

DSNDB06.SYSDBASE, DSNDB06.SYSDBAUT, DSNDB06.SYSGROUP, DSNDB06.SYSPLAN, DSNDB06.SYSVIEWS, DSNDB01.DBD01

These table spaces contain internal links. Links are internal pointers that tie the information in their tables together hierarchically. A link can be thought of as a type of parent-child relationship that. Due to these links, the BUILD and SORT phases of the REORG utility are not execute. A different calculation is required for the size of the unload data set used by REORG.

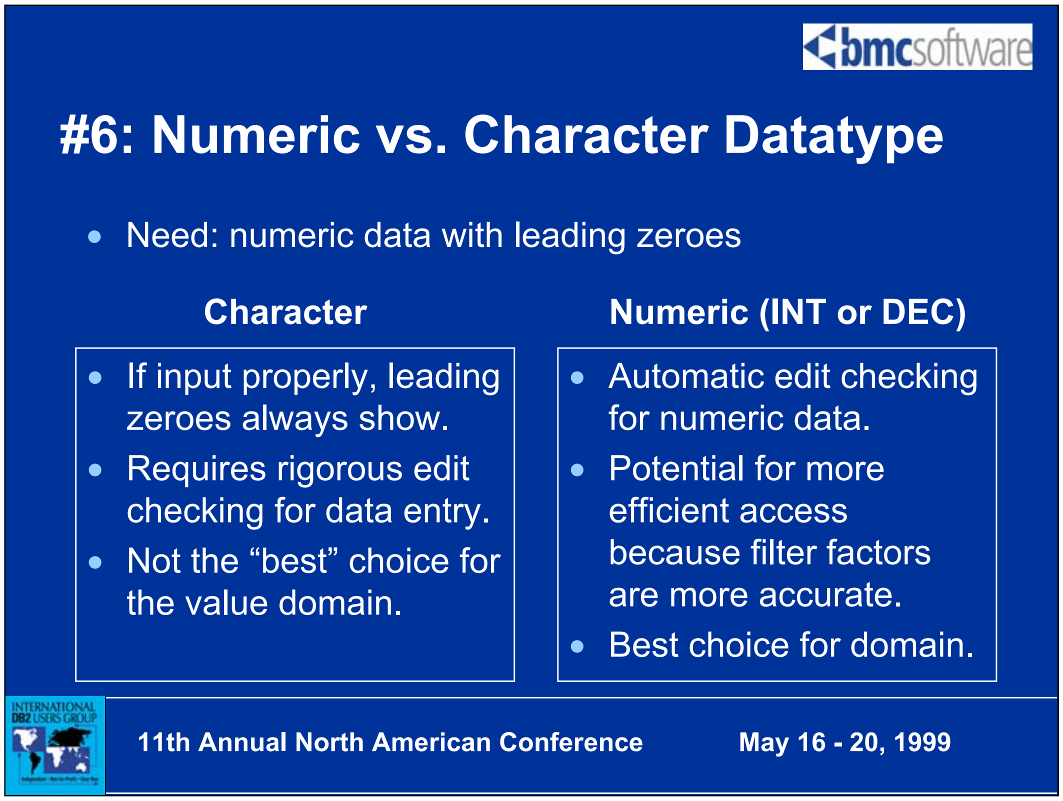
It is a more difficult prospect to determine when the DB2 directory table spaces should be reorganized because RUNSTATS can not be used to gather statistics.



#5: Create Indexes on the Catalog

- As of DB2 V5 you can create additional indexes on DB2 Catalog tables
- Determine the types of queries regularly being run and build appropriate indexes to optimize performance
- Gotcha: can't RECOVER INDEX(ALL) for new indexes
- Examples:
  - ...if the CREATOR is always the same (in prod) you might want an index on NAME in SYSTABLES
  - Perhaps CREATEDBY is queried extensively

Creating additional indexes on tables in the DB2 Catalog can optimize system catalog queries (but not internal DB2 processes like BIND).



#6: Numeric vs. Character Datatype

- Need: numeric data with leading zeroes

Character	Numeric (INT or DEC)
• If input properly, leading zeroes always show. • Requires rigorous edit checking for data entry. • Not the “best” choice for the value domain.	• Automatic edit checking for numeric data. • Potential for more efficient access because filter factors are more accurate. • Best choice for domain.

11th Annual North American Conference May 16 - 20, 1999

A four-byte code is required to identify an entity; all of the codes are numeric and will stay that way. But, for reporting purposes, users wish the codes to print out with leading zeroes. Should the column be defined as CHAR(4) or SMALLINT?

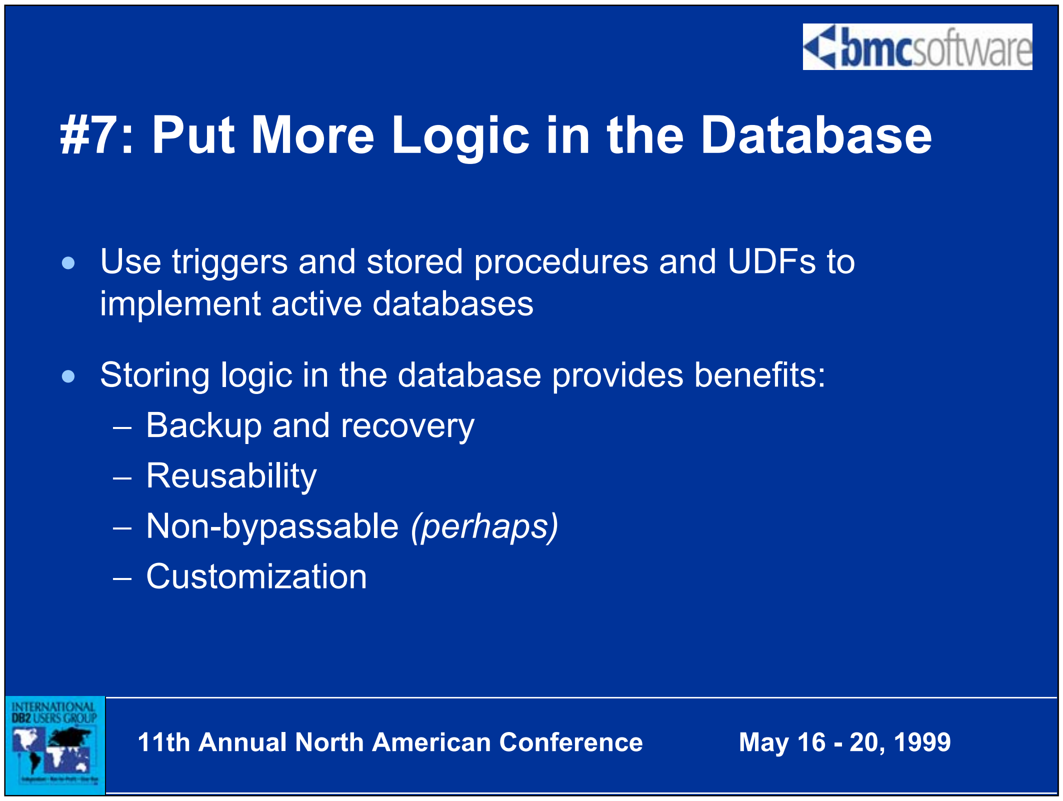
Edit checks: Without proper edit checks, inserts and updates could place invalid alphabetic characters into the product code. This can be a very valid concern if ad hoc data modifications are permitted. This is rare in production databases, but data problems can still occur if the proper edit checks are not coded into every program that can modify the data. If proper edit checks are coded and will never be bypassed, this removes the data integrity question.

Filter factors: Consider the possible number of values that a CHAR(4) column and a SMALLINT column can assume. Even if edit checks are coded for each, DB2 is not aware of these and assumes that all combinations of characters are permitted. DB2 uses base 37 math when it determines access paths for character columns, under the assumption that 26 alphabetic letters, 10 numeric digits, and a space will be used. This adds up to 37 possible characters. For a four-byte character column there are 374 or 1,874,161 possible values.

A SMALLINT column can range from -32,768 to 32,767 producing 65,536 possible small integer values. The drawback here is that negative or 5 digit product codes could be entered. However, if we adhere to our proper edit check assumption, the data integrity problems will be avoided here, as well.

DB2 will use the HIGH2KEY and LOW2KEY values to calculate filter factors. For character columns, the range between HIGH2KEY and LOW2KEY is larger than numeric columns because there are more total values. The filter factor will be larger for the numeric data type than for the character data type which may influence DB2 to choose a different access path. For this reason, favor the SMALLINT over the CHAR(4) definition.

The leading zeroes problem might be able to be solved using other methods. When using QMF, you can ensure that leading zeroes are shown by using the "J" edit code. Report programs can be coded to display leading zeroes easily enough by moving the host variables to appropriate display fields. Ad hoc access through other reporting tools typically provide a parameter that can enable leading zeroes to be displayed.



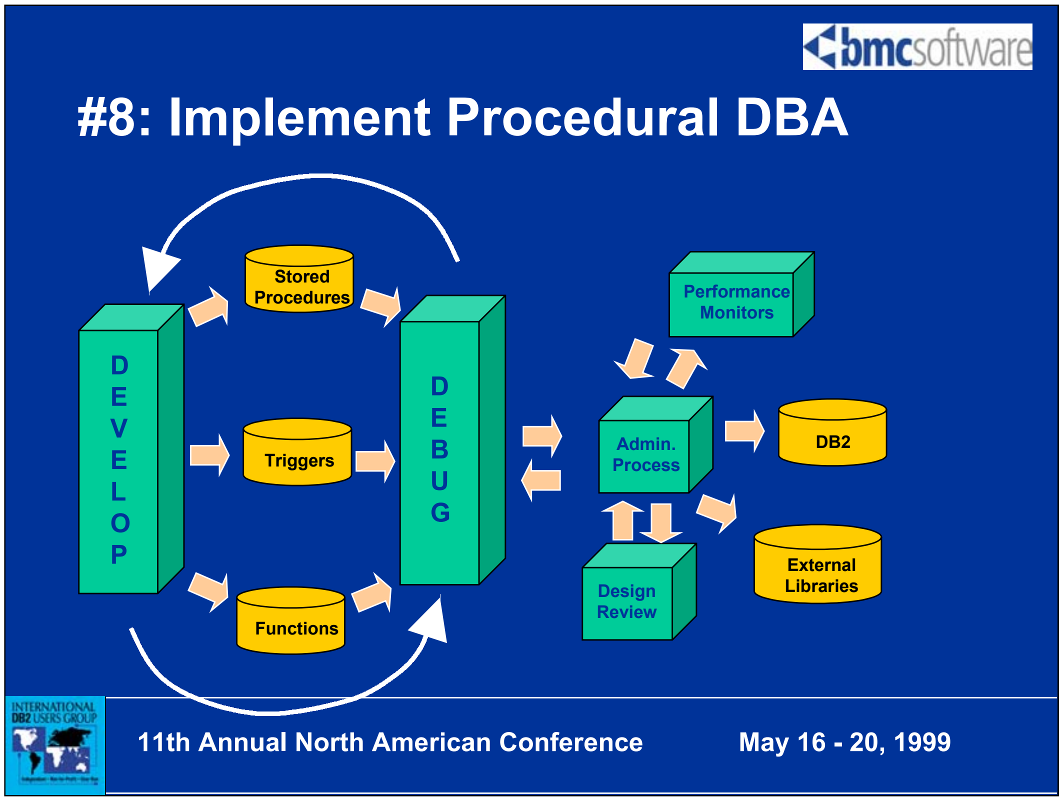
#7: Put More Logic in the Database

- Use triggers and stored procedures and UDFs to implement active databases
- Storing logic in the database provides benefits:
  - Backup and recovery
  - Reusability
  - Non-bypassable *(perhaps)*
  - Customization

11th Annual North American Conference May 16 - 20, 1999

Stored procedures - since DB2 Version 4

Triggers and UDFs - new in DB2 Version 6



#8: Implement Procedural DBA

11th Annual North American Conference May 16 - 20, 1999

Once triggers and functions (UDFs) and stored procedures are coded and made available to DB2, applications and developers will begin to rely upon them. Although the functionality provided by these “code-based” objects is unquestionably useful and desirable, DBAs are presented with a major dilemma. Now that procedural logic is being stored in the DBMS, DBAs must grapple with the issues of quality, maintainability, and availability. How and when will these objects be tested? The impact of a failure is enterprise-wide, not relegated to a single application. This increases the visibility and criticality of these objects. Who is responsible if they fail? The answer must be—a DBA.

With the advent of server code objects, the role of the DBA is expanding to encompass too many responsibilities for a single person to perform the job capably. The solution is to split the DBA’s job into two separate parts based upon the database object to be supported: data objects or “code-based” objects.

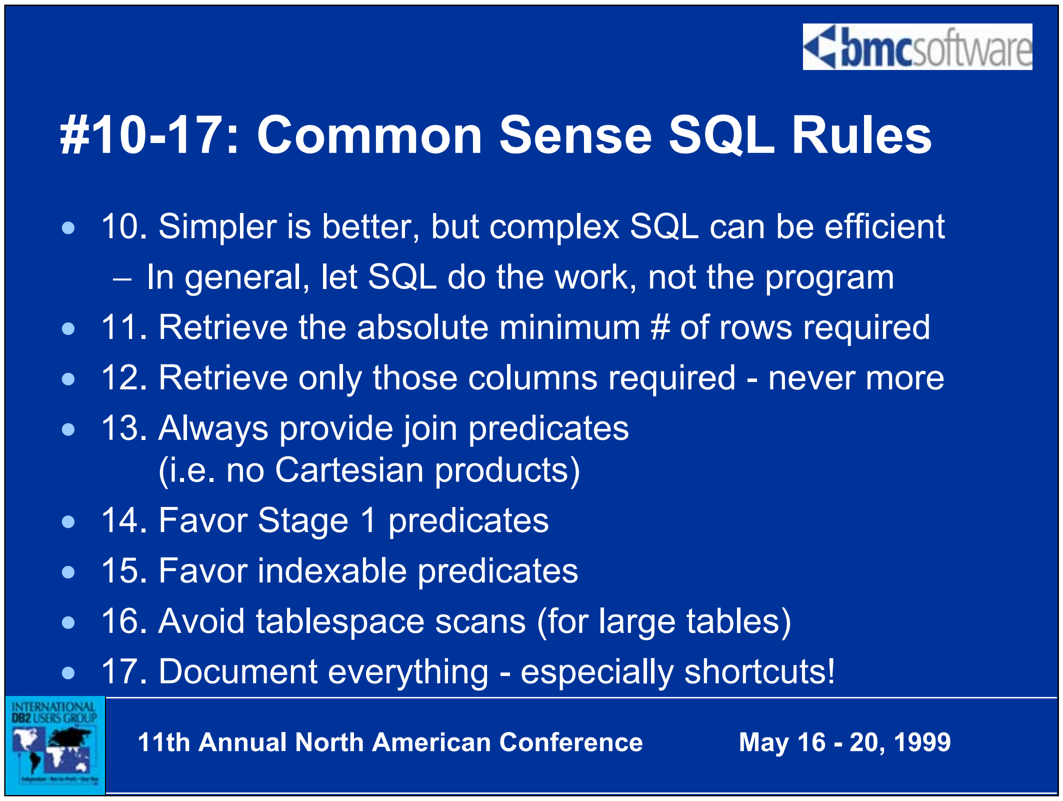
The procedural DBA should be responsible for those database management activities that require procedural logic support and/or coding. Of course, this should include primary responsibility for server code objects. Whether server code objects are actually programmed by the procedural DBA will differ from shop-to-shop. This will depend on the size of the shop, the number of DBAs available, and the scope of server code object implementation. At a minimum, the procedural DBA should participate in and lead the review and administration of “code-based” objects. Additionally, s/he should be on call for trigger/function/proc ABENDs.

Other procedural administrative functions that should be allocated to the procedural DBA include application code reviews, access path review and analysis (from EXPLAIN or show plan), SQL debugging, complex SQL analysis, and re-writing queries for optimal execution. Off-loading these tasks to the procedural DBA will enable the traditional, data-oriented DBAs to concentrate on the actual physical design and implementation of databases.

The procedural DBA should still report through the same management unit as the traditional DBA and not through the application programming staff. This enables better skills sharing between the two distinct DBA types. There is a need for greater synergy between procedural DBAs and application programmers/analysts. The typical job path for procedural DBAs should be from the application programming ranks because this is how development skills are learned.

#9: Stored Procedure Usage

- Keep it simple - make each procedure do one thing only
- Use stored procedures to create reusable "components" of business logic
- Putting multiple types of business logic into a single stored procedure makes it more difficult to tune, modify, and understand
- Document the purpose of each procedure as well as any required input, outputs, and each change made to the stored procedure



#10-17: Common Sense SQL Rules

- 10. Simpler is better, but complex SQL can be efficient
  - In general, let SQL do the work, not the program
- 11. Retrieve the absolute minimum # of rows required
- 12. Retrieve only those columns required - never more
- 13. Always provide join predicates
  (i.e. no Cartesian products)
- 14. Favor Stage 1 predicates
- 15. Favor indexable predicates
- 16. Avoid tablespace scans (for large tables)
- 17. Document everything - especially shortcuts!

11th Annual North American Conference May 16 - 20, 1999

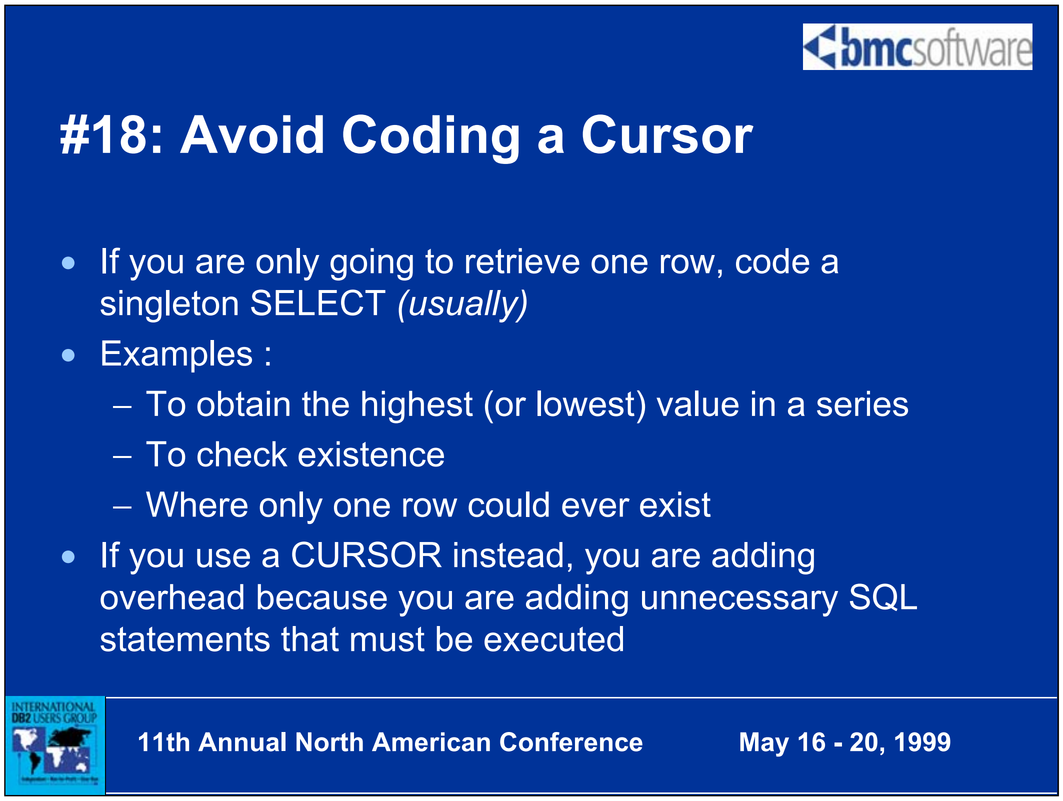
Whenever any of the shortcuts outlined in this article are employed, be sure to document that fact. Use comments in the SQL statement to record the reason behind the formulation of the SQL. Failure to do so can cause programmers to undo the change during future enhancements. At the least, confusion may result as to why the statement was formulated that particular way. So, the SQL from the first example should be coded as follows:

```
--
-- RETRIEVE ALL VPS WITH A GRADE LEVEL OF
-- AT LEAST 10.
--
-- THE SECOND PREDICATE IS COMMENTED
-- OUT BECAUSE THE STARTING GRADE LEVEL
-- OF VICE PRESIDENTS AT OUR COMPANY IS 10.
--
   SELECT FIRST_NAME, LAST_NAME, GRADE_LEVEL

   FROM EMPLOYEE

   WHERE TITLE = 'VP'

-- AND GRADE_LEVEL >= 10;
```



#18: Avoid Coding a Cursor

- If you are only going to retrieve one row, code a singleton SELECT *(usually)*
- Examples :
  - To obtain the highest (or lowest) value in a series
  - To check existence
  - Where only one row could ever exist
- If you use a CURSOR instead, you are adding overhead because you are adding unnecessary SQL statements that must be executed

However, if DB2 chooses a bad access path for the singleton SELECT, it may be more efficient to fetch from a cursor that specifies OPTIMIZE FOR 1 ROW to get an efficient access path. Example of access path problems that a singleton SELECT could incur include sorts, list prefetch, inappropriate join technique, etc. etc.

- -

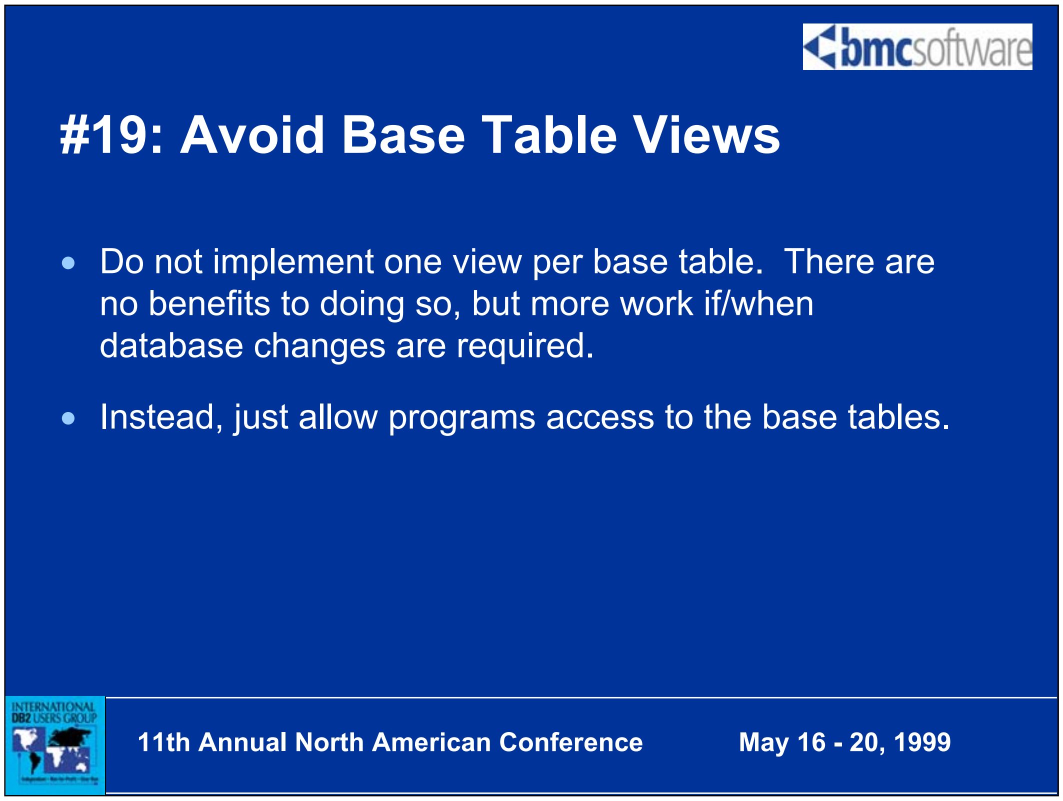
Check the SQLCODE to make sure more than one row was not encountered:

SQLCODE	SQLSTATE
+000 : one row returned	‘00000’
+100 : no rows found	‘02000’
-811 : more than one row found	‘21000’

- -

Existence check example:

```
SELECT 1
INTO :HV-CONSTANT
FROM TABLE_NAME
WHERE COL_NAME = :HV-VALUE
```



#19: Avoid Base Table Views

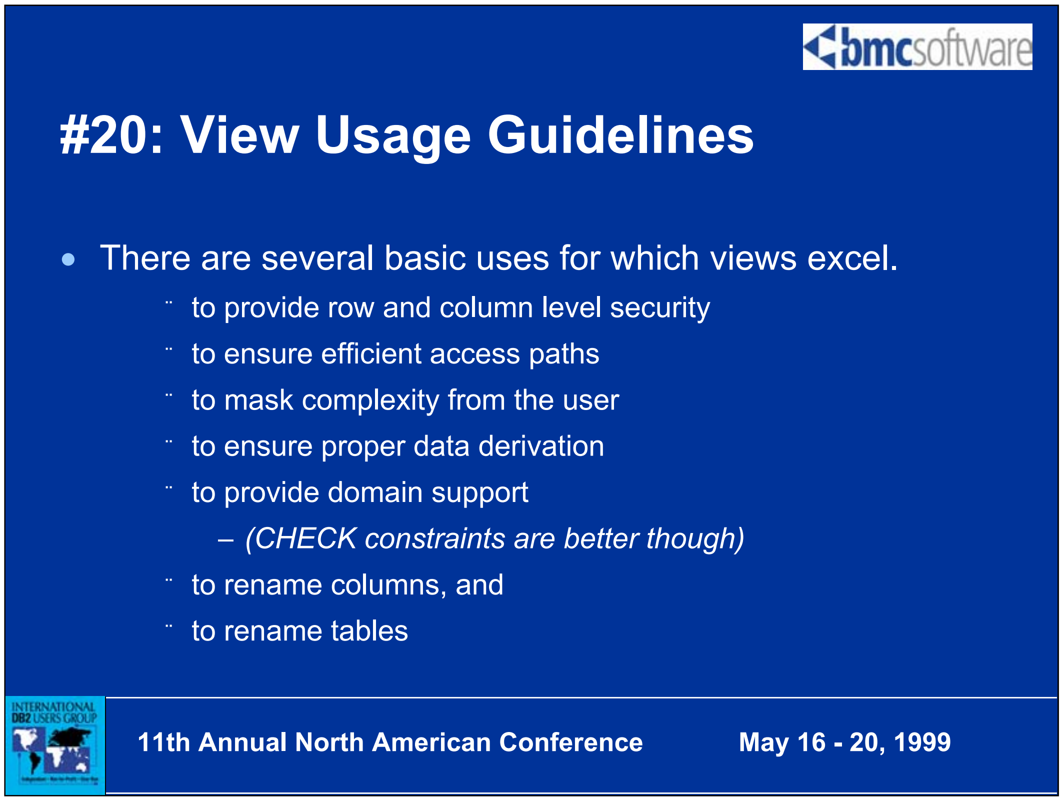
- Do not implement one view per base table. There are no benefits to doing so, but more work if/when database changes are required.
- Instead, just allow programs access to the base tables.

11th Annual North American Conference May 16 - 20, 1999

Often times the dubious recommendation is made to create one view for each base table in a DB2 application system. This is what I call The Big View Myth. The reasoning behind The Big View Myth revolves around the desire to insulate application programs from database changes. This insulation is purported to be achieved because all programs are written to access views instead of base tables. When a change is made to the base table, the programs do not need to be modified because they access a view - not the base table.

Although this sounds like a good idea in principle, indiscriminate view creation should be avoided. The implementation of database changes requires scrupulous analysis regardless of whether views or base tables are used by your applications. Consider the simplest type of database change--adding a column to a table. If you do not add the column to the view, no programs can access that column unless another view is created that contains that column. But if you create a new view every time you add a new column it will not take long for your environment to be swamped with views. Even more troublesome is which view should be used by which program? Similar arguments can be made for removing columns, renaming tables and columns, combining tables, and splitting tables.

In general, if you follow good DB2/SQL programming practices, you will usually not encounter situations where the usage of views initially would have helped program/data isolation anyway. By dispelling The Big View Myth you will decrease the administrative burden of creating and maintaining an avalanche of base table views.



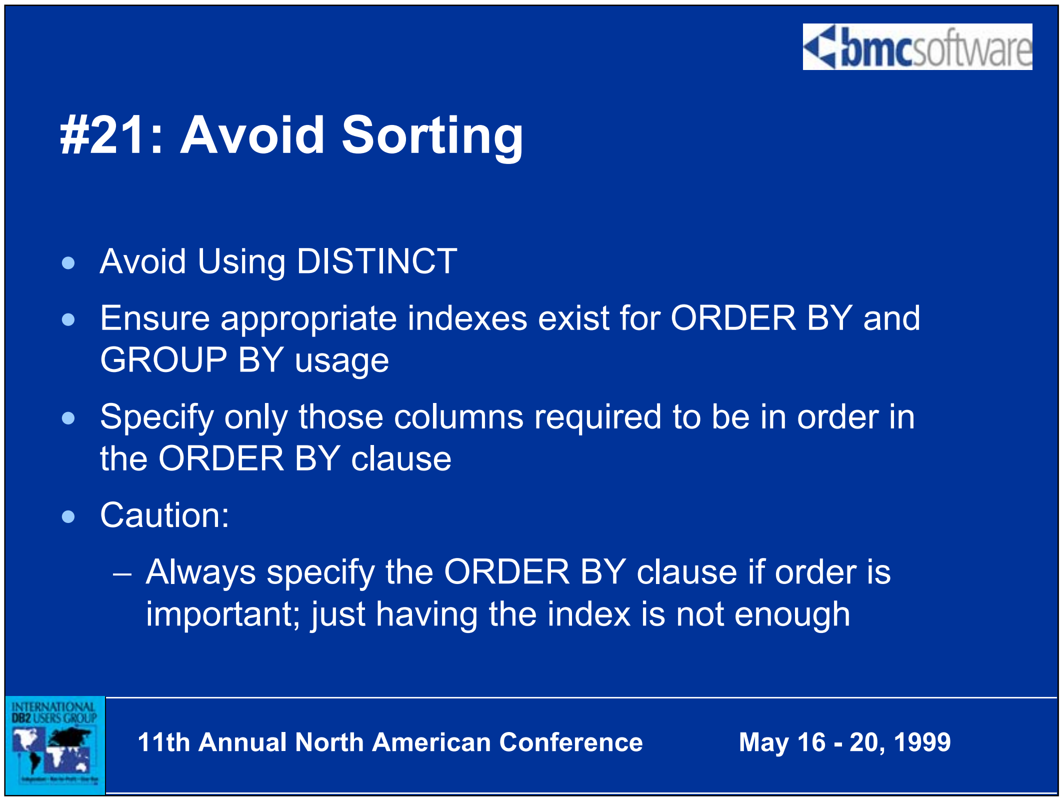
#20: View Usage Guidelines

- There are several basic uses for which views excel.
  - to provide row and column level security
  - to ensure efficient access paths
  - to mask complexity from the user
  - to ensure proper data derivation
  - to provide domain support
    - *(CHECK constraints are better though)*
  - to rename columns, and
  - to rename tables

11th Annual North American Conference **May 16 - 20, 1999**

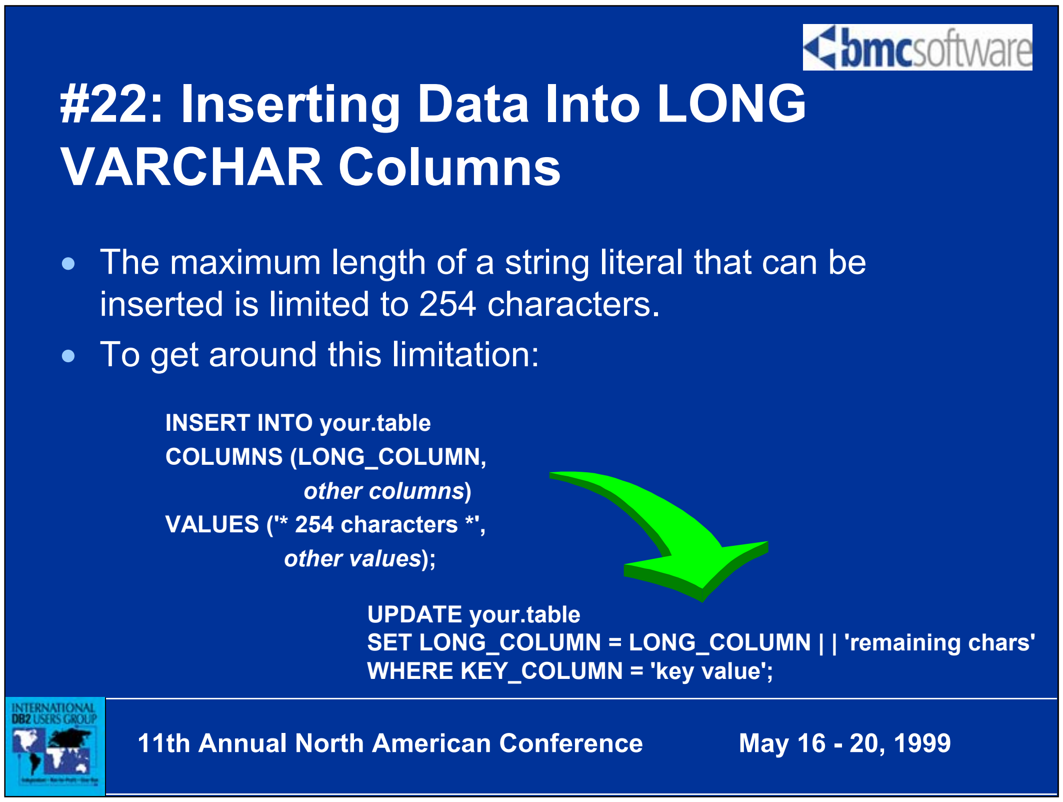
For more details on how best to use views in DB2 check out “Views on Views” as published in ***DB2 Update***. The article can be accessed via the following URL:

http://www.craigsmullins.com/viewnw.htm



#21: Avoid Sorting

- Avoid Using DISTINCT
- Ensure appropriate indexes exist for ORDER BY and GROUP BY usage
- Specify only those columns required to be in order in the ORDER BY clause
- Caution:
  - Always specify the ORDER BY clause if order is important; just having the index is not enough



#22: Inserting Data Into LONG VARCHAR Columns

- The maximum length of a string literal that can be inserted is limited to 254 characters.
- To get around this limitation:

```
INSERT INTO your.table
COLUMNS (LONG_COLUMN,
          other columns)
VALUES ('* 254 characters *',
          other values);

UPDATE your.table
SET LONG_COLUMN = LONG_COLUMN | | 'remaining chars'
WHERE KEY_COLUMN = 'key value';
```

11th Annual North American Conference May 16 - 20, 1999

Problem: one of your databases uses LONG VARCHAR columns. Initially, data was only inserted into these columns using application programs. Using host variables, we could insert long strings into these columns. However, usage of both DB2 and the data in the database has matured, and we would like to allow our more sophisticated users to insert data directly into these columns using QMF. However, the maximum length of a string literal that can be inserted into DB2 is limited to 254 characters. Is there any way around this limitation?

Solution:

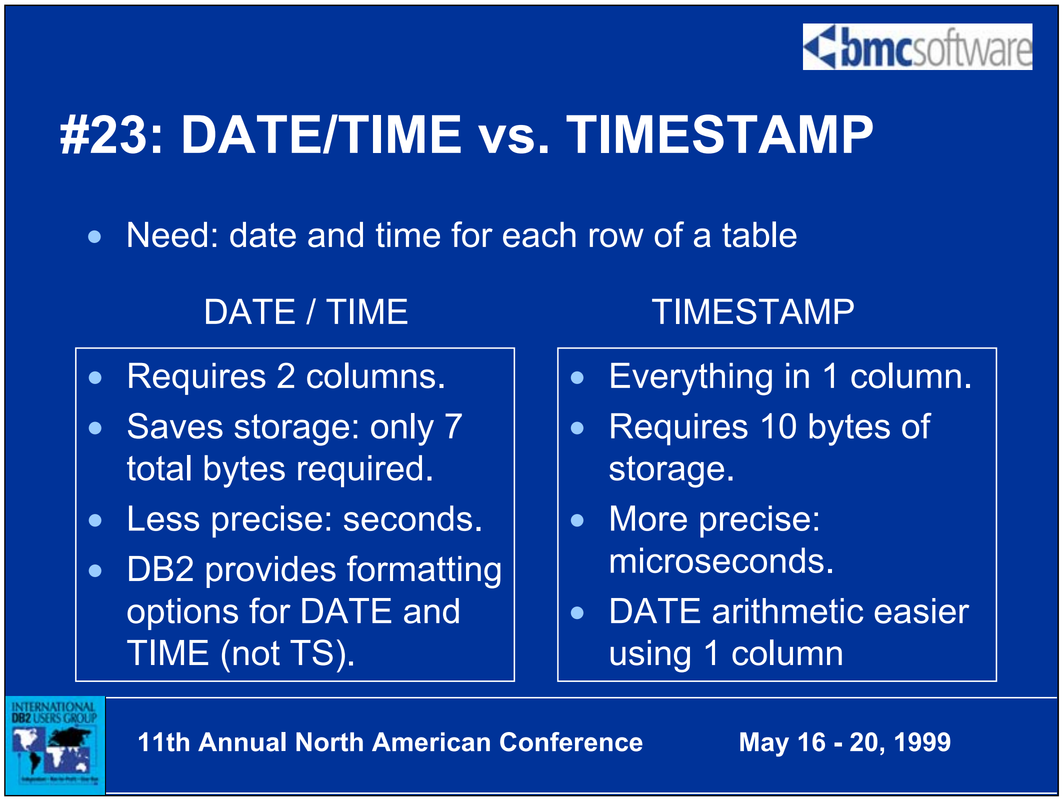
Instead of issuing a single INSERT, try issuing an INSERT followed by an UPDATE. For example, if you need to insert 300 bytes of data into the LONG VARCHAR column, use the following technique:

Begin by inserting the first 254 bytes into the appropriate column as shown below:

```
INSERT INTO your.table
COLUMNS (LONG_COLUMN,
other columns)
VALUES ('* 254 characters *',
other values);
```

Next, issue an UPDATE statement to add the rest of the data to the column. This can be accomplished using the concatenation operator as shown below:

```
UPDATE your.table
SET LONG_COLUMN = LONG_COLUMN | | '* remaining characters *'
WHERE KEY_COLUMN = 'key value';
```



#23: DATE/TIME vs. TIMESTAMP

- Need: date and time for each row of a table

DATE / TIME	TIMESTAMP
• Requires 2 columns. • Saves storage: only 7 total bytes required. • Less precise: seconds. • DB2 provides formatting options for DATE and TIME (not TS).	• Everything in 1 column. • Requires 10 bytes of storage. • More precise: microseconds. • DATE arithmetic easier using 1 column

11th Annual North American Conference May 16 - 20, 1999

Problem: You have the need to store both date and time information on a single row in DB2. Is it better to use a single TIMESTAMP column or two columns, one DATE and the other TIME?
Solution: The answer to this question depends on several factors specific to your situation. Consider the following points before making your decision:

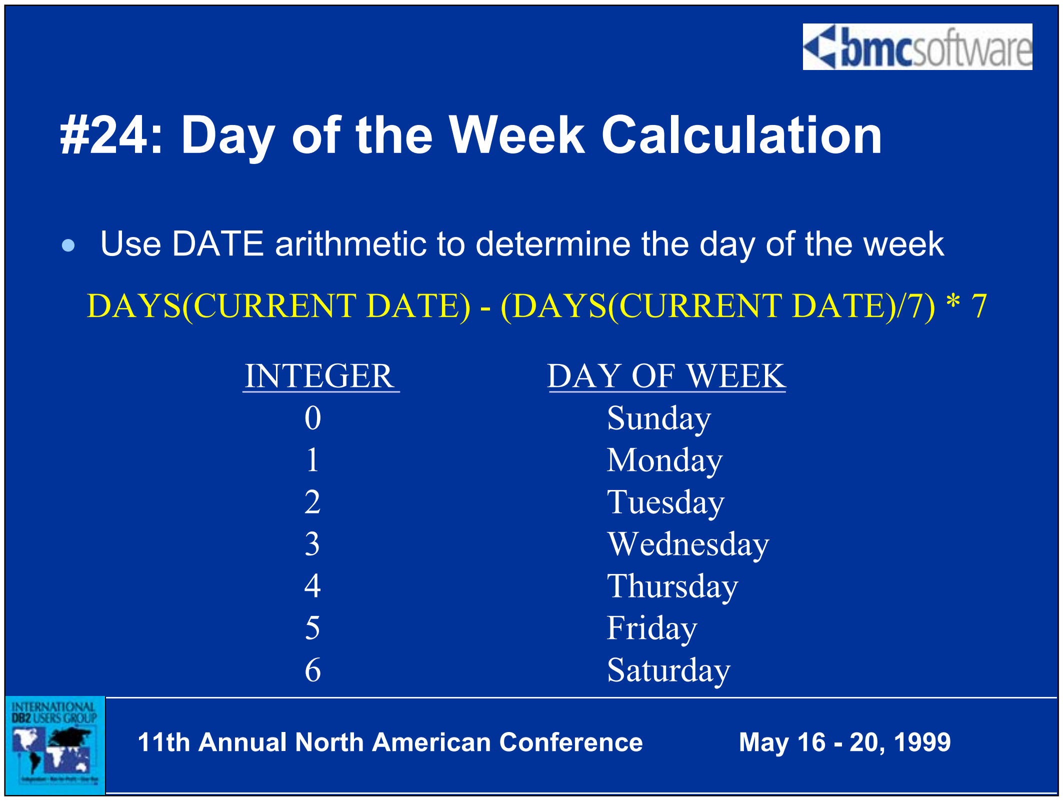
•With DATE and TIME you must use two columns. TIMESTAMP uses one column, thereby simplifying data access and modification

•The combination of DATE and TIME columns requires 7 bytes of storage, while a TIMESTAMP column always requires 10 bytes of storage. Using the combination of DATE and TIME columns will save space.

•TIMESTAMP provides greater time accuracy, down to the microsecond level. TIME provides accuracy only to the second level. If precision is important, use TIMESTAMP. Use TIME if you want to ensure that the actual time is NOT stored down to the microsecond level.

•Date and time arithmetic is easier to implement using TIMESTAMP data instead of a combination of DATE and TIME. Subtracting one TIMESTAMP from another results in a TIMESTAMP duration. To calculate a duration using DATE and TIME columns, two subtraction operations must occur: one for the DATE column and one for the TIME column.

•DB2 provides for the formatting of DATE and TIME columns via local DATE and TIME exits, the CHAR function, and the DATE and TIME precompiler options. These facilities are not available for TIMESTAMP columns. If the date and time information is to be extracted and displayed on a report or by an online application, the availability of these DB2-provided facilities for



#24: Day of the Week Calculation

- Use DATE arithmetic to determine the day of the week

```
DAYS(CURRENT DATE) - (DAYS(CURRENT DATE)/7) * 7
```

INTEGER	DAY OF WEEK
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

#25: Avoid Arithmetic Expressions

- Avoid arithmetic expressions in SQL predicates by moving the arithmetic to the host language:

```
SELECT EMPNO, LASTNAME
FROM DAN8510.EMP
WHERE SALARY > :HV-SALARY * 1.1
```

COBOL

```
COMPUTE HV-SALARY = HV-SALARY * 1.1
```

SQL

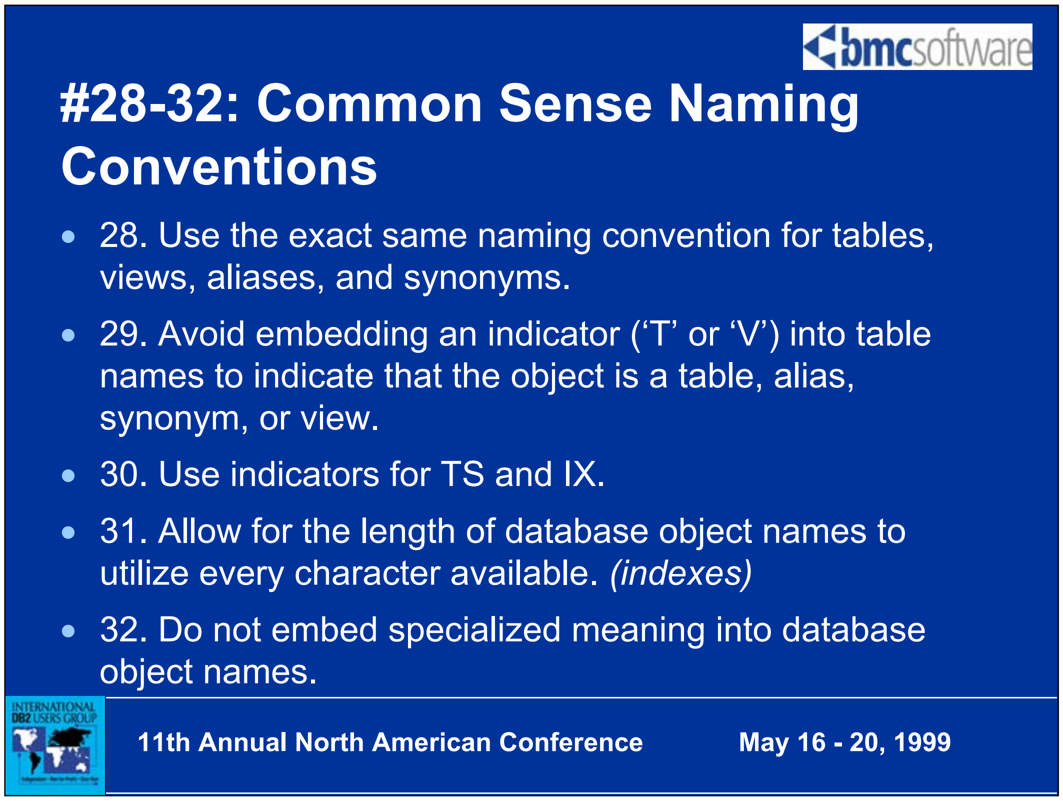
```
SELECT EMPNO, LASTNAME
FROM DAN8510.EMP
WHERE SALARY > :HV-SALARY
```

#26: Use Multiple Bufferpools

- IBM provides 80 bufferpools for a reason ***(as of V6)***
- Using them effectively optimizes performance
- Ideas:
  - isolate the catalog in BP0
  - separate indexes from tablespaces
  - isolate heavily hit data
  - isolate sort work area
  - optimize your strategy for your data and application processing mix - there is no “silver bullet” approach

#27: Consider Global Temporary Tables

- No locking, no logging - but very useful
- Increased efficiency : optimize recurring expressions
  - Subquery in multiple UPDATE statements
  - Cost = tablespace scan to fill a workfile
  - Compare the CPU consumption of scanning the workfile versus processing the subquery
  - A normal DB2 table would require management
- But, remember, the access path for GTT is TS scan!



#28-32: Common Sense Naming Conventions

- 28. Use the exact same naming convention for tables, views, aliases, and synonyms.
- 29. Avoid embedding an indicator (‘T’ or ‘V’) into table names to indicate that the object is a table, alias, synonym, or view.
- 30. Use indicators for TS and IX.
- 31. Allow for the length of database object names to utilize every character available. *(indexes)*
- 32. Do not embed specialized meaning into database object names.

11th Annual North American Conference May 16 - 20, 1999

The first revision to the common wisdom for database naming standards. Use the exact same naming convention for tables, views, aliases, and synonyms. These four objects all logically refer to the same thing in the relational model - a representation of data in terms of columns and rows. It is common for many shops to implement different naming conventions for each of these objects. Does it really make sense? Why put a “T” in the table name or a “V” in the view name? The name of a DB2 table should accurately and succinctly convey the contents of the data it contains (not the type of object it is - that is in the system catalog).

Use all 18 characters of those table names. Make them as descriptive as possible. Same for column names, etc. etc. Exception is index names because DB2 allows 18 bytes, but uses a formula to convert 18 bytes to 8 bytes to create the underlying VSAM data set name. Limit indexes to 8 bytes for this reason.

Some shops enforce DB2 index naming conventions where the type of index is embedded in the index name. For example, to indicate clustering, PK, FK, and/or uniqueness. This is not wise. Attributes can change and the list is not orthogonal. An index can be both unique and clustering - what do you do then? For tablespaces and indexes you want special indicator characters (e.g. S for TS and X for index) to differentiate them. Indicator characters are helpful to ensure that tablespaces and indexes are never named the same. For a complete discussion of DB2 naming conventions check out the article at:

http://www.xephon.com/archives/b023a04.txt

#33-35: Common Sense DBA Rules

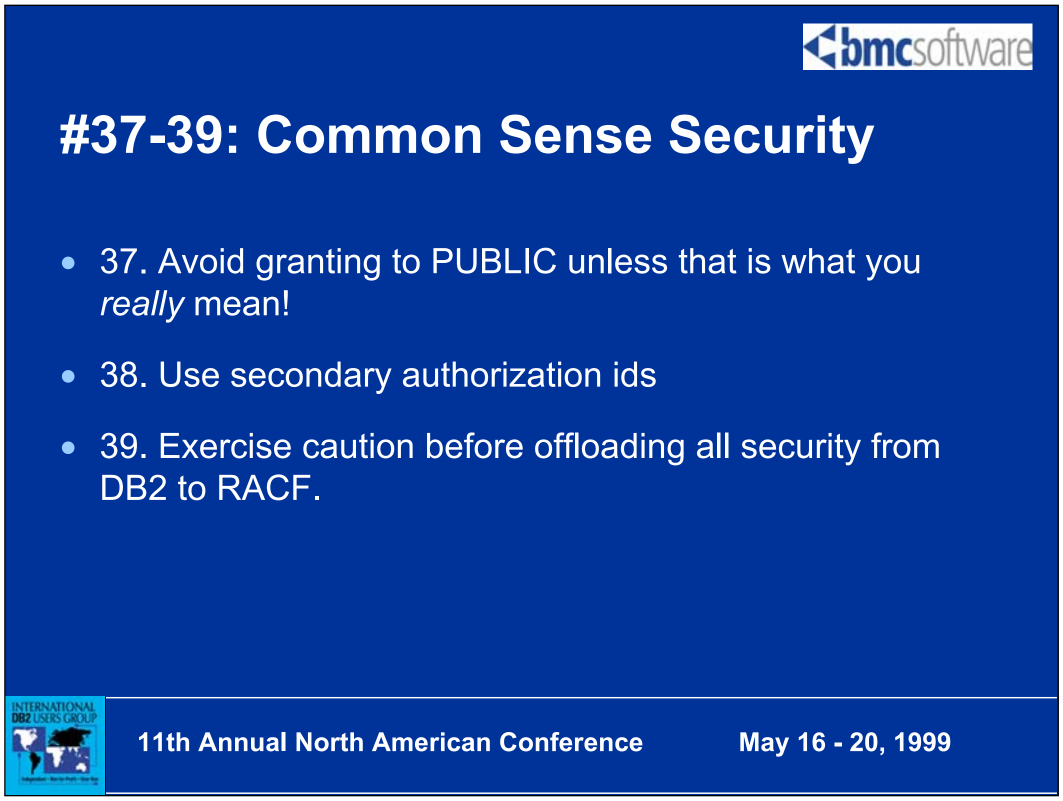
- 33. Proliferation Avoidance - don’t create unnecessary objects or leave unused object “hanging around”
- 34. Know Your Data - understand your environment and its data usage and processing to optimize performance
- 35. Review everything before it goes into production - SQL, application code, DDL, subsystem changes, DB2 PTFs, etc. etc. etc.

#36: To Wait or To Boldly Go...

Version 5 Changes (SUPs)

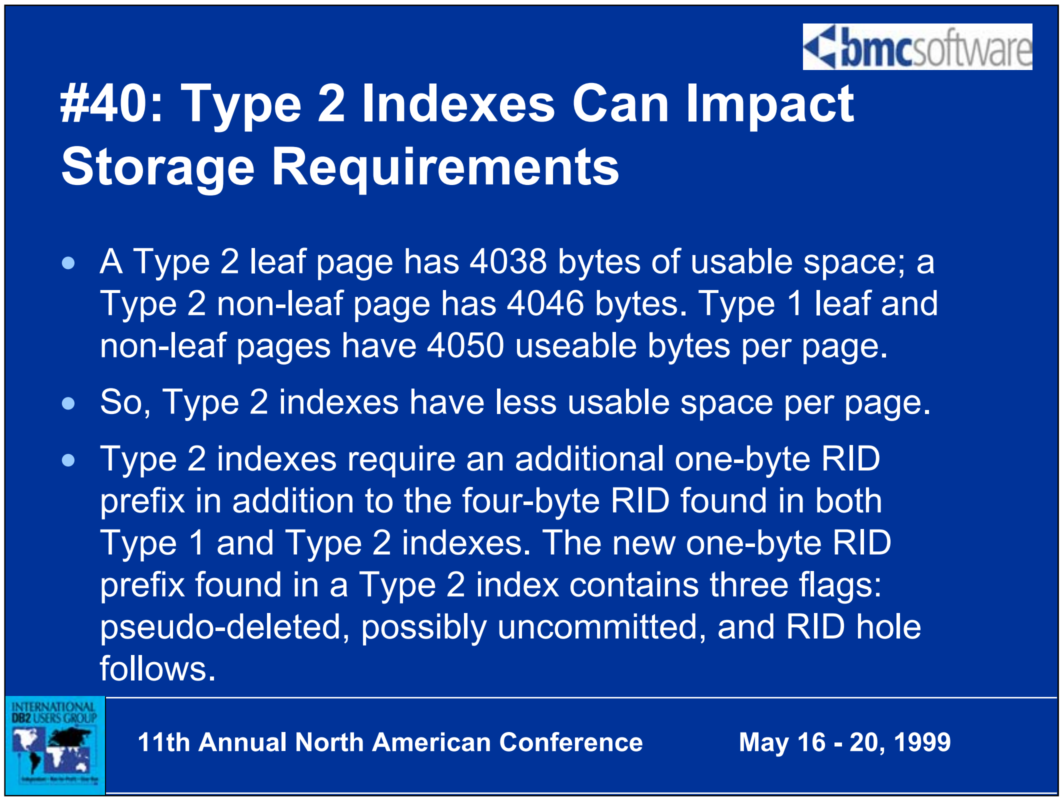
Dynamic statement cache improvements continue in APARs PQ14893, PQ14941, and PQ17905.
Index screening for list prefetch is PQ15670
Parallel data set allocation is PQ02028, PQ05287 and PQ08269.
Over 10,000 allocations are PQ18543 and OS/390 V2R6
IRLM is improved in PQ12126, PQ12390, PQ15854 and PQ09947.
REORG REUSE is PQ19077
Delete and unload external for reorg is PQ19897
GBP duplex is PQ17797
ALTER space without stop is PQ04053.
RACF access control comes in OS/390 V2R4 Security Server.
SQLJ comes in PQ19814
RRS was enhanced in PQ06950 and PQ11045.
The ODBC thread model of multiple uncoordinated connections and thread- safe ODBC support came in PQ09901.
Improvements for the ability to identify end users and end user applications are in APAR PQ07043.
REBUILD Index in Recover is PQ09842.
Utility stored procedures PQ15682-5.

http://www.software.ibm.com/data/db2/os390/v5apar.html



#37-39: Common Sense Security

- 37. Avoid granting to PUBLIC unless that is what you *really* mean!
- 38. Use secondary authorization ids
- 39. Exercise caution before offloading all security from DB2 to RACF.

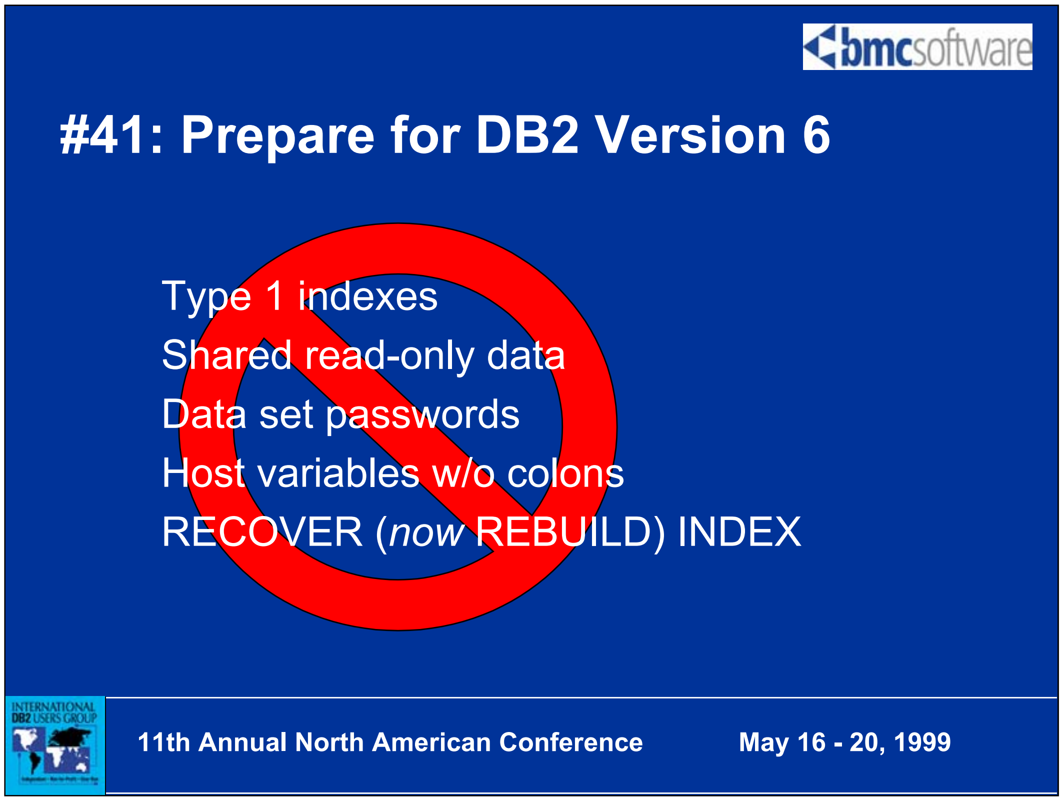


#40: Type 2 Indexes Can Impact Storage Requirements

- A Type 2 leaf page has 4038 bytes of usable space; a Type 2 non-leaf page has 4046 bytes. Type 1 leaf and non-leaf pages have 4050 useable bytes per page.
- So, Type 2 indexes have less usable space per page.
- Type 2 indexes require an additional one-byte RID prefix in addition to the four-byte RID found in both Type 1 and Type 2 indexes. The new one-byte RID prefix found in a Type 2 index contains three flags: pseudo-deleted, possibly uncommitted, and RID hole follows.

For in-depth coverage of the storage impact of type 2 indexes refer to the article at the following URL:

http://www.craigsmullins.com/db2_type.htm



#41: Prepare for DB2 Version 6

Type 1 indexes
Shared read-only data
Data set passwords
Host variables w/o colons
RECOVER (*now* REBUILD) INDEX

11th Annual North American Conference May 16 - 20, 1999

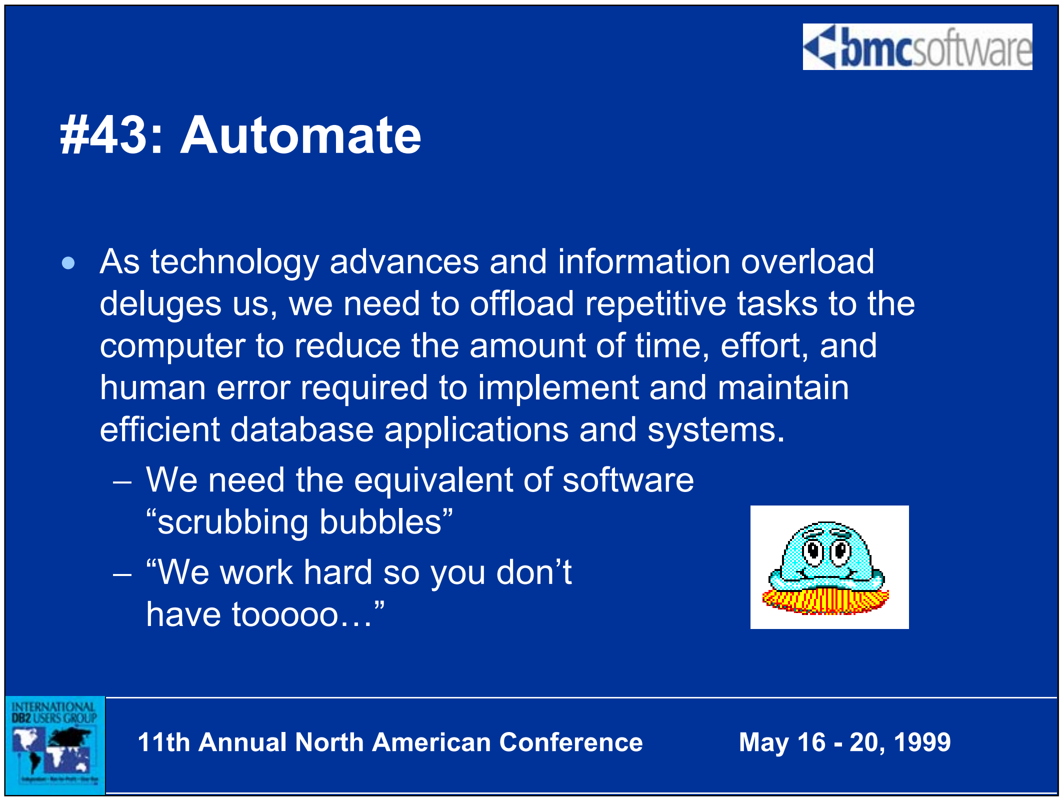
Version 6 is the first release/version of DB2 to actually remove features. The above listed items are no longer supported in DB2 Version 6. Applications and databases that utilized these features pre-V6, will need to be modified.

Be sure to do the following under V5 before moving to V6:

- convert all type 1 indexes to type 2
- migrate SROD to distributed DB2 or data sharing
- eliminate data set passwords (use native O/S security features to protect data sets if required)
- make sure all host variables, in all DB2 programs, are preceded by a colon
- convert all RECOVER INDEX jobs to REBUILD INDEX

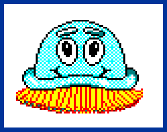
#42: Migrate Away from Private Protocol Distribution

- It appears to be “the next thing to go” from the next version of DB2 (post-V6)
- Move to DRDA distribution
  - – easier to do with DRDA now supporting three part names



#43: Automate

- As technology advances and information overload deluges us, we need to offload repetitive tasks to the computer to reduce the amount of time, effort, and human error required to implement and maintain efficient database applications and systems.
  - We need the equivalent of software "scrubbing bubbles"
  - "We work hard so you don't have tooooo…"

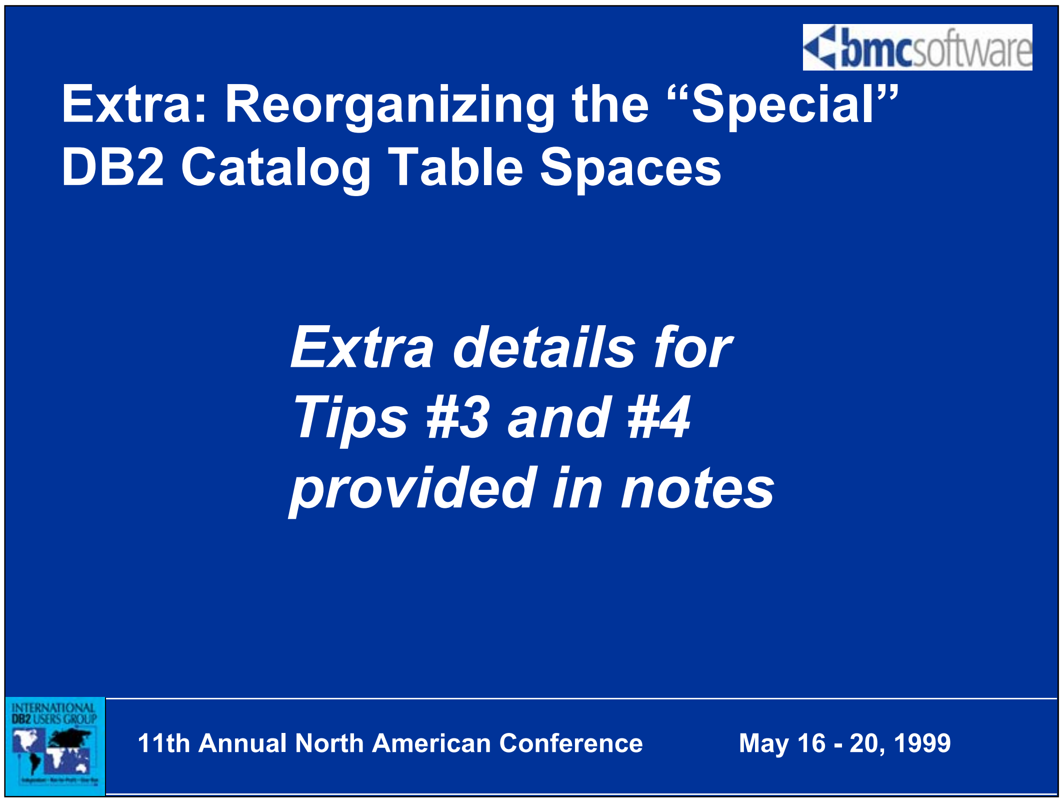


11th Annual North American Conference May 16 - 20, 1999

For more on this topic consult the article "Software Scrubbing Bubbles" at: http://www.craigsmullins.com/cnr_ssb.htm

#44: Think About The Future

- Spend time thinking about the future and the state of the industry.
- Do not get so mired in the details of your day-to-day job that you lose sight of the trends that will impact your job in the near- and long-term future.
- Hey, maybe we could have avoided this Year 2000 mess if more of us did this!



Extra: Reorganizing the “Special” DB2 Catalog Table Spaces

Extra details for Tips #3 and #4 provided in notes

11th Annual North American Conference May 16 - 20, 1999

Steps to REORG the Six “Special” Table Spaces

The following steps should be used when reorganizing the six “different” table spaces (refer to Tips #3 and 4):

1. Calculate size of unload data set (SYSREC)

   The SYSREC data set for the “special” table spaces has a different format than the other table spaces. This causes a special calculation to be required to determine its size. The equation to use is:

   DATA SET SIZE IN BYTES = (28 + LONGROW) * NUMROWS

   NUMROWS is the number of rows to be contained in the data set and LONGROW is the length of the longest in the table space. The value for LONGROW can be determined by running the following SQL statement:

   ```
   SELECT MAX(RECLENGTH)
   FROM    SYSIBM.SYSTABLES
   WHERE  DBNAME = ‘DSNDB06’
   AND       TSNAME = ‘name of table space to REORG’
   AND       CREATOR = ‘SYSIBM’;
   ```

2. Ensure incompatible operations are not executing
3. Start database DSNDB01 and DSNDB06 for read only access
4. Run QUIESCE and DSN1CHKR utilities
5. Take full image copy of entire DB2 catalog & directory table spaces
6. Start DSNDB01 and DSNDB06 for utility access
7. Execute REORG utility
8. Take full image copy of entire DB2 catalog and directory table spaces
9. Start table space and associated indexes for read/write access

44 Tips In 66 Minutes

http://www.bmc.com

Craig S. Mullins
BMC Software
Director, DB2 Technology Planning
Craig_Mullins@BMC.com
cmullins@compuserve.com
http://www.craigmullins.com

www.craigsmullins.com/cm-book.htm