# Top New Features of
# DB2 Version 8

*For Developer's Only*


**Craig S. Mullins**

**Mullins Consulting, Inc.**

http://www.CraigSMullins.com

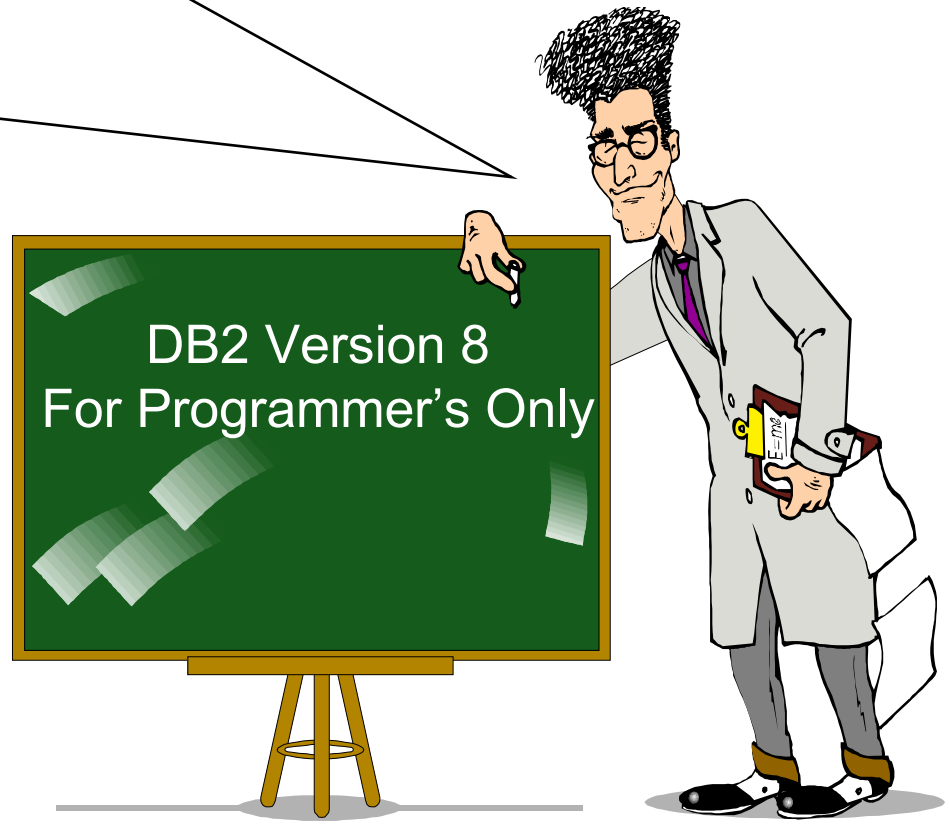

**Sponsored And Hosted By**

SoftBase

http://www.SoftBase.com

# Agenda

In this presentation we will go over the highlights of DB2 V8 from the programmer's perspective. …that is, we will look at the most useful and interesting aspects of the latest and greatest version of DB2 in terms of what a programmer, coder, or developer cares about instead of what a DBA cares most about… Though there is some overlap.



# Up to 225 Tables per Query

- **Previous limit (V7) was 15 tables joined in a query**
- **As of DB2 V8, the limit becomes 225 tables**
- **When more than 15 tables are joined the optimizer operates somewhat differently**
  - **Limits are set on elapsed time, storage, and CPU**
  - **Will only examine a limited number of access path choices**
  - **After a limit is reached:**
    - Not every access path is analyzed due to resource constraints
    - The optimizer will make a decision
    - Goal for access plan changes from "**optimal**" to "**reasonable**"

# Larger SQL Statements

- Previous limit (V7) for SQL statements was 32K
  - Very complex requests could not be done using SQL alone
- Maximum predicate operand length is now 32,704 (matches maximum VARCHAR)
- As of DB2 V8, SQL statements can be up to 2MB
  - More work can be put into SQL
  - Support for Long Names *(next slide)*
  - Support for up to 225 table joins *(previous slide)*
  - Reduction in -101 SQLCODEs
  - Useful when writing SQL Procedure Language code

# Long Names


- Limits for DB2 object names raised in V8
- Up to 128 byte names for:
  - Table, Index, Creator, View, Synonym, Stogroup, Authid
- Column name is limited to 30 bytes (was 18)
- Some things don't change – still same size
  - Database, Plan, Program, Tablespace all still 8 bytes
- Index key limit was 255 bytes, now 2000 (includes nulls, lengths)
  - Partitioning key limit still 255
- Pertinent SQLCODE:
  - -107 THE NAME *name* IS TOO LONG. MAXIMUM ALLOWABLE SIZE IS *size*

# Multi-Row FETCH and INSERT

- Allows multi-row result set with one execution
  - One trip across the network
- Eases cross-platform browsing operations
- Can significantly improve performance
  - When large amounts of data are either retrieved or inserted
  - Distributed Block Fetch mitigated some FETCH, but not INSERT
- Uses arrays in application programs
  - FETCH into the array, or;
  - INSERT from the array
- Rowset Positioning used to ‘navigate’

# What is a ROWSET?

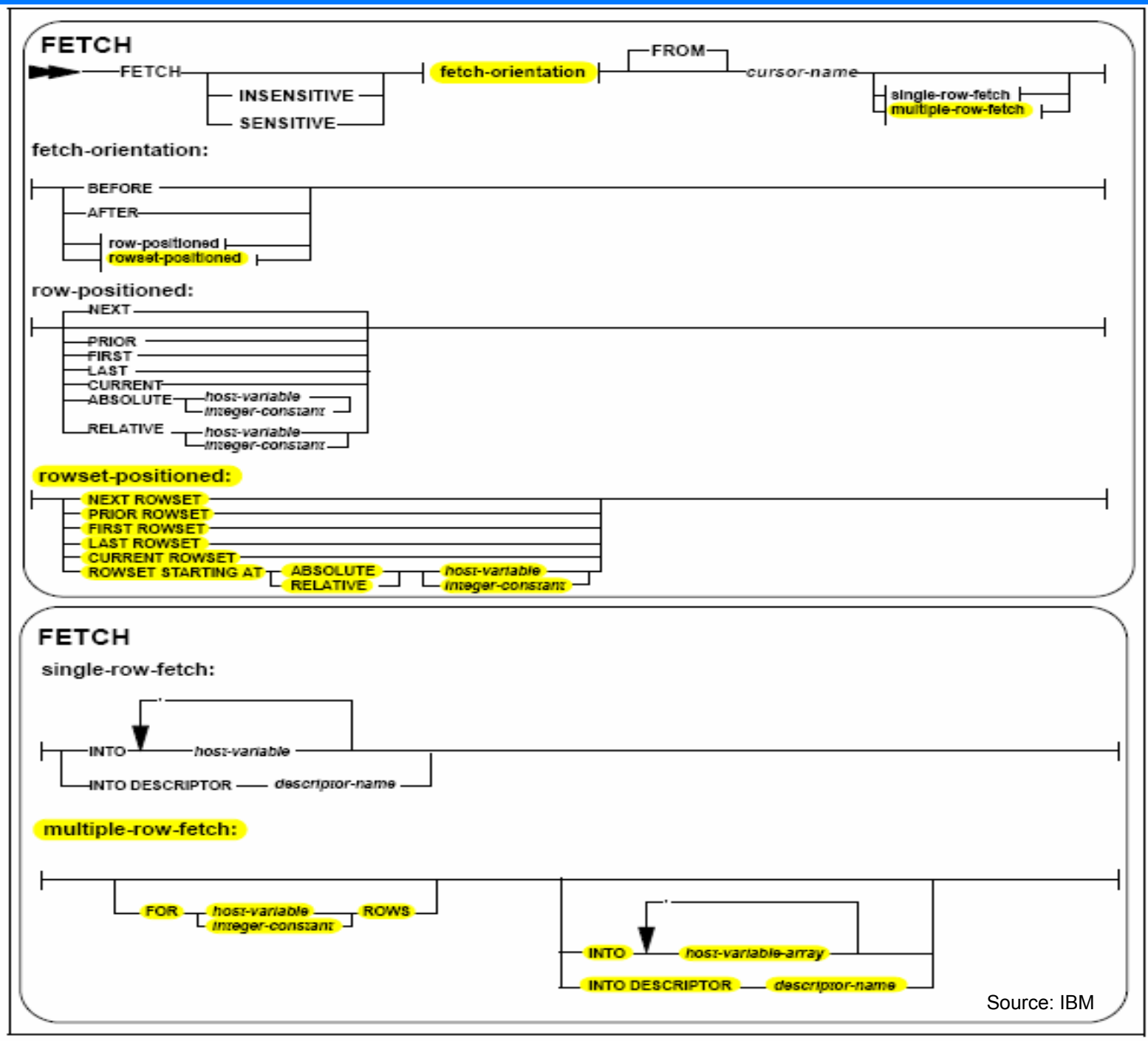
- A ROWSET is a group of rows for the result set of a query that are returned by a single FETCH statement.
- Your program controls the size of the ROWSET
  - …that is, how many rows are returned by each FETCH
- ROWSET size is specified on the FETCH statement
  - The maximum rowset size is 32,767
- You can mix single row and multiple row fetches for a multi-fetch cursor

# Multi-Row FETCH

- Multi-row FETCH: a single FETCH statement can retrieve multiple rows of data from the result table of a query as a rowset.
- Must DECLARE the cursor for rowset positioning, for example:

```
EXEC SQL
  DECLARE C1 CURSOR
  WITH ROWSET POSITIONING
  FOR SELECT * FROM EMP;
```

- Then you can FETCH multiple rows-at-a-time from the cursor.


FETCH
FETCH
INSENSITIVE
SENSITIVE
fetch-orientation
FROM
cursor-name
single-row-fetch
multiple-row-fetch
fetch-orientation:
BEFORE
AFTER
row-positioned
rowset-positioned
row-positioned:
NEXT
PRIOR
FIRST
LAST
CURRENT
ABSOLUTE
host-variable
integer-constant
RELATIVE
host-variable
integer-constant
rowset-positioned:
NEXT ROWSET
PRIOR ROWSET
FIRST ROWSET
LAST ROWSET
CURRENT ROWSET
ROWSET STARTING AT
ABSOLUTE
RELATIVE
host-variable
integer-constant
FETCH
single-row-fetch:
INTO
host-variable
INTO DESCRIPTOR
descriptor-name
multiple-row-fetch:
FOR
host-variable
integer-constant
ROWS
INTO
host-variable-array
INTO DESCRIPTOR
descriptor-name

Source: IBM

# Multi-Row FETCH – Examples (1)

- Fetch the first x rows and leave the cursor positioned on that rowset at the completion of the fetch.

```
FETCH FIRST ROWSET FROM C1
FOR x ROWS INTO ...
```

- Fetch the next 5 rows starting at the current position of the cursor, and cause the cursor to be positioned on that rowset at the completion of the fetch.

```
FETCH ROWSET FROM C1
FOR 5 ROWS INTO ...
```

# Multi-Row FETCH – Examples (2)

- Fetch the previous ROWSET and have the cursor positioned on that rowset.

```
FETCH PRIOR ROWSET FROM C1
FOR x ROWS INTO ...
```

- Fetch 5 rows starting with row 10 regardless of the current position of the cursor, and cause the cursor to be positioned on that rowset at the completion of the fetch.

```
FETCH ROWSET
STARTING AT ABSOLUTE 10 FROM C1
FOR 5 ROWS INTO ...
```

# ROWSET Size

If FOR n ROWS is NOT specified and the cursor is declared for ROWSET positioning..

- The ROWSET will be the same as the previous ROWSET FETCH if…
  - It was the previous FETCH for this cursor…
  - Or the previous FETCH was a FETCH BEFORE or FETCH AFTER and the FETCH before that was a ROWSET FETCH.
  - Otherwise the ROWSET defaults to 1

# Host Variable Arrays

Your program needs to be written to include arrays that will work with multi-row SQL operations.

- A host variable array is an array in which each element of the array contains a value for the same column
- Can only be referenced in multi-row fetch or insert
- In general, arrays may not be arrays of structures

# COBOL w/Host Variable Arrays

```
01 OUTPUT-VARS.
   05 LASTNME OCCURS 10 TIMES.
      49 LASTNME-LGTH   PIC S9(4) USAGE COMP.
      49 LASTNME-DATA   PIC X(50).
   05 EMPNO             PIC S9(9) COMP-4 OCCURS 10 TIMES.

PROCEDURE DIVISION.

EXEC SQL
  DECLARE C1 CURSOR WITH ROWSET POSITIONING FOR
     SELECT LASTNME, EMPNO
     FROM   DSN8810.EMP
END-EXEC.

EXEC SQL
  OPEN C1
END-EXEC.

EXEC SQL
  FETCH FIRST ROWSET FROM C1 FOR 10 ROWS
                INTO :LASTNME,
                     :EMPNO
END-EXEC.
```

# Positioned UPDATE

The new syntax allows you to specify in the UPDATE statement the following clause to update a specified row in the rowset:

```
UPDATE T1
   SET C1 = 5
WHERE CURRENT OF cursor-name
FOR ROW row-number OF ROWSET
```

Instead, if you specify the existing WHERE CURRENT OF cursor-name, all the rows in the rowset are updated. For example:

- Update all the rows in the rowset that cursor CSR1 is positioned on.

```
UPDATE T1
   SET C1 = 5
WHERE CURRENT OF CSR1
```

# Positioned DELETE

The new syntax allows you to specify in the DELETE statement the following clause to delete a specified row in the rowset:

```
DELETE FROM T1
WHERE CURRENT OF cursor-name
FOR ROW row-number OF ROWSET
```

Instead, if you specify the existing WHERE CURRENT OF cursor-name, all the rows in the rowset are deleted. The following example deletes the fourth row in the rowset that cursor CSR1 is positioned on:

```
DELETE FROM T1
WHERE CURRENT OF CSR1
FOR ROW 4 OF ROWSET
```

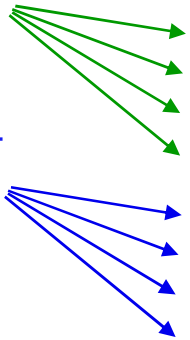
# Know Your Cursor Position

## After a multi-row FETCH, the cursor is positioned on ALL rows in current ROWSET

FETCH FIRST ROWSET
FOR 4 ROWS

FETCH NEXT ROWSET
FOR 4 ROWS

FETCH NEXT ROWSET
FOR 2 ROWS



| EXP_DATE | CUSTNO | CUSTNAME … |
|---|---|---|
| 07-11-2004 | 500 | ACME RENTAL |
| 07-19-2004 | 100 | MULLINS CONSULTING |
| 07-25-2004 | 600 | SOFTBASE SYSTEMS |
| 07-31-2004 | 175 | SUPER BANK |
| 08-01-2004 | 400 | SPUMCO |
| 08-16-2004 | 333 | SIGH BASE CORP. |
| 08-27-2004 | 200 | BETH-ANN INC. |
| 08-28-2004 | 900 | DIAL-A-DBA |
| 09-01-2004 | 800 | VAN THE MAN |
| 09-02-2004 | 715 | DBAZINE.com |
| 09-04-2004 | 300 | RENT-IT, INC. |
| 09-10-2004 | 950 | VANDALAY INDUST. |

# End of Multi-Row FETCH

**If you try to fetch beyond the result set you will receive an end of data condition.**

- Consider, for example, that there are only 10 rows left in the result table and you issue the following FETCH request:

```
FETCH NEXT ROWSET FOR 25 ROWS
```

- The last 10 rows will be returned along with an SQLCODE +100
- This is also the case when FETCH FIRST n ROWS ONLY has been specified

# Performance Considerations

**Testing has shown up to a 50% CPU time reduction when using multi-row FETCH**

- **Due to avoiding the reduction in API calls**
- **The percentage improvement will be lower the fewer rows fetched per call (also, the more columns fetched, the less the improvement will be)**

**The current "sweet spot" for multi-row FETCH is at about 100 rows.**

- **Performance gains start with 10 rows; if you are going to retrieve less than that it may not make a lot of sense to code multi-row FETCHing**

# Multi-Row FETCH SQLCODEs

**-227 FETCH *fetch-orientation* IS NOT ALLOWED, BECAUSE CURSOR *cursor-name* HAS AN UNKNOWN POSITION**

**-246 STATEMENT USING CURSOR *cursor-name* SPECIFIED NUMBER OF ROWS *num-rows* WHICH IS NOT VALID WITH *dimension***

**-248 A POSITIONED DELETE OR UPDATE FOR CURSOR *cursor-name* SPECIFIED ROW *n* OF ROWSET, BUT THE ROW IS NOT CONTAINED WITHIN THE CURRENT ROWSET**

# Multi-Row FETCH SQLCODEs (2)

**-249 DEFINITION OF ROWSET ACCESS FOR CURSOR *cursor-name* IS INCONSISTENT WITH THE FETCH ORIENTATION CLAUSE *clause* SPECIFIED**

**-253 A NON-ATOMIC *statement* STATEMENT SUCCESSFULLY COMPLETED FOR SOME OF THE REQUESTED ROWS, POSSIBLY WITH WARNINGS, AND ONE OR MORE ERRORS**

**-254 A NON-ATOMIC *statement* STATEMENT ATTEMPTED TO PROCESS MULTIPLE ROWS OF DATA, BUT ERRORS OCCURRED**

# Multi-Row FETCH SQLCODEs (3)

**-353 FETCH IS NOT ALLOWED, BECAUSE CURSOR *cursor-name* HAS AN UNKNOWN POSITION**

**-508 THE CURSOR IDENTIFIED IN THE UPDATE OR DELETE STATEMENT IS NOT POSITIONED ON A ROW OR ROWSET THAT CAN BE UPDATED OR DELETED**

**-589 A POSITIONED DELETE OR UPDATE STATEMENT FOR CURSOR *cursor-name* SPECIFIED A ROW OF A ROWSET, BUT THE CURSOR IS NOT POSITIONED ON A ROWSET**

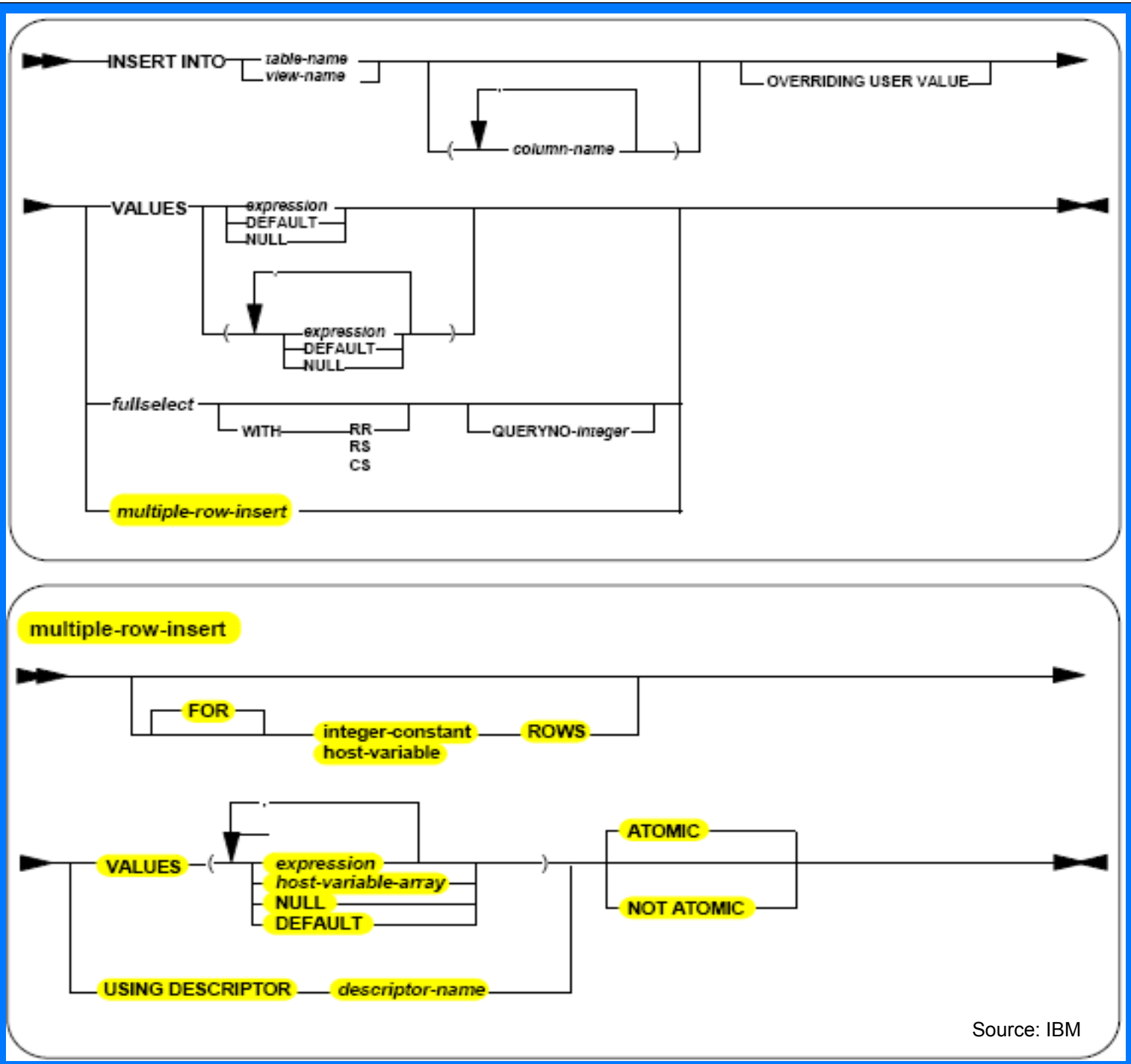
# Multi-Row FETCH SQLCODEs (4)

**-5012 HOST VARIABLE *host-variable* IS NOT EXACT NUMERIC WITH SCALE ZERO**

**-20185 CURSOR *cursor-name* IS NOT DEFINED TO ACCESS ROWSETS, BUT A CLAUSE WAS SPECIFIED THAT IS VALID ONLY WITH ROWSET ACCESS**

# Multi-Row INSERT

- Third type of INSERT now available in V8
  - List of Values ⟶ `INSERT … VALUES`
  - From a SELECT ⟶ `INSERT … SELECT …`
  - From an array ⟶ `INSERT … FOR x ROWS`
- Multi-Row INSERT: a single INSERT statement can add multiple rows of data from an array.
  - For static and dynamic SQL (examples upcoming)
  - FOR n ROWS
    - Maximum n is 32,767


INSERT INTO
table-name
view-name
OVERRIDING USER VALUE
column-name
VALUES
expression
DEFAULT
NULL
fullselect
WITH
RR
RS
CS
QUERYNO-integer
multiple-row-insert
multiple-row-insert
FOR
integer-constant
host-variable
ROWS
VALUES
expression
host-variable-array
NULL
DEFAULT
ATOMIC
NOT ATOMIC
USING DESCRIPTOR
descriptor-name

Source: IBM

# ATOMIC versus NON-ATOMIC

- ATOMIC – if any single row insert fails, all fail
- NOT ATOMIC – rows inserted independently
  - If errors occur during execution of INSERT, processing continues
  - Diagnostics are available for each failed row through GET DIAGNOSTICS *(more on GET DIAGNOSTICS later)*
  - SQLCODE will indicate if all failed, all were successful or at least one failed

# Multi-Row INSERT – Examples

- Static SQL INSERT with host variable arrays.

```
INSERT INTO T FOR :x ROWS
VALUES(:hva1, :hva2) ATOMIC
```

- Dynamic SQL INSERT with host variable arrays.

```
stmt:     'INSERT INTO T VALUES(?, ?)'

attrvar:  'FOR MULTIPLE ROWS ATOMIC'

PREPARE my_isrt ATTRIBUTES :attrvar FROM :stmt

EXECUTE my_isrt FOR :hv ROWS USING (:hva1, :hva2)
```

# Performance Considerations

**Testing has shown up to a 40% CPU time reduction when using multi-row INSERT**

- **Due to avoiding the reduction in API calls**
- **The percentage improvement will be lower as the number of indexes goes up, number of rows inserted per call goes down, and/or the number of columns goes up**

# Multi-row INSERT SQLCODEs

**-30106 INVALID INPUT DATA DETECTED FOR A MULTIPLE ROW INSERT OPERATION. INSERT PROCESSING IS TERMINATED**

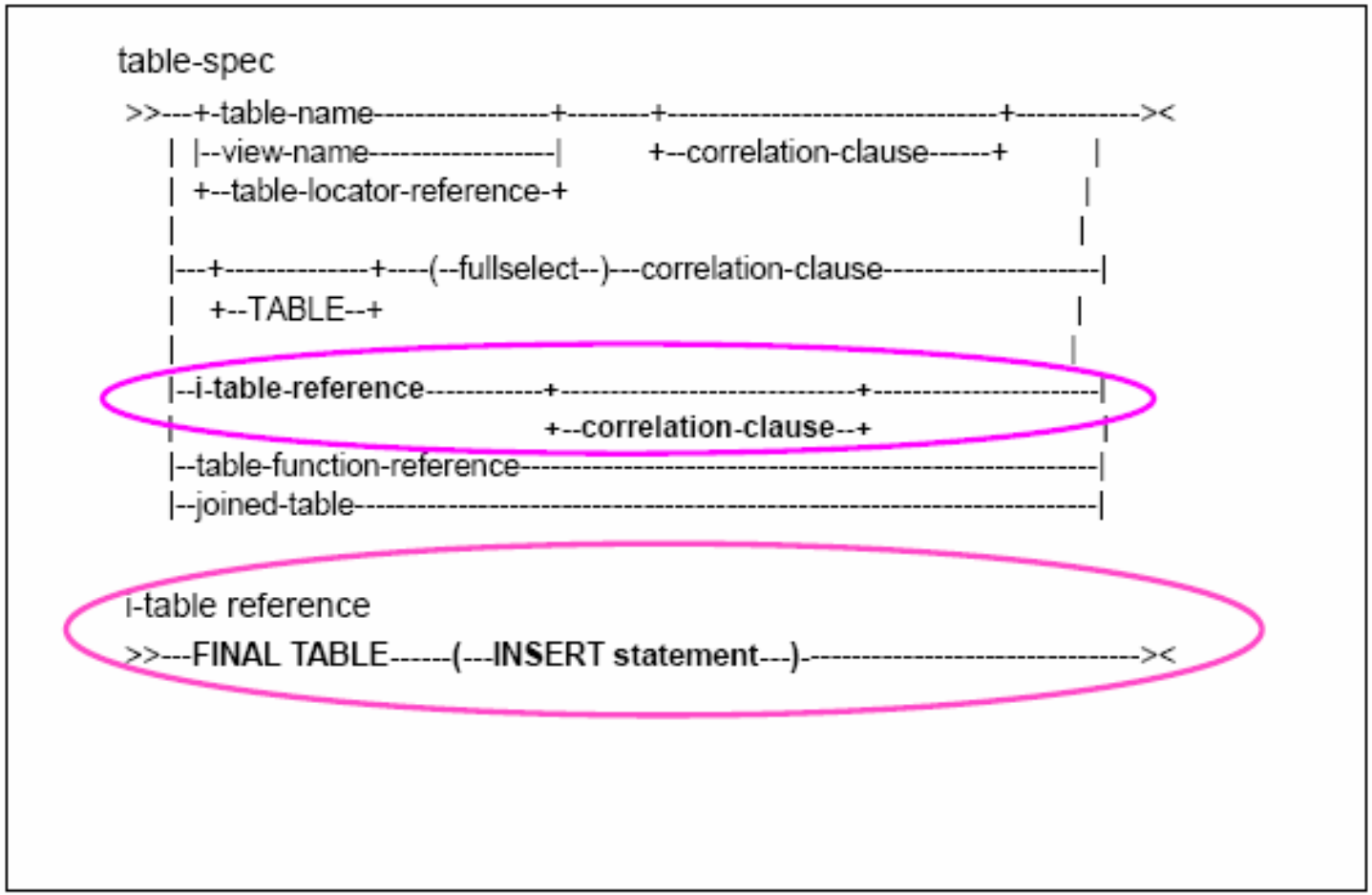
# SELECT from INSERT

**You can combine a SELECT and INSERT statement.**

**Why would you want to do that?**

- **Users can automatically retrieve column values created by DB2**
  - Enables a program to immediately determine values inserted into tables (identity, sequence, defaults, etc.) and BEFORE triggers
  - Fewer SQL statements can minimize network cost for applications
  - Less procedural logic may be needed in stored procedures

# Syntax




Source: IBM

# Example

```
SELECT FNAME, LNAME, DATE_ADDED
  INTO :fname_hv,
       :lname_hv,
       :dateadd_hv
FROM FINAL TABLE
  (INSERT
   INTO EMP (FNAME, LNAME, SALARY, LEVEL)
         VALUES('CRAIG',
                'MULLINS',
                2000000.00,
                'DBA')
   );
```

DATE_ADDED Defaults to Current DATE

# SELECT from INSERT Considerations

**INSERT statement is not allowed in the FROM clause of a:**

- **Select statement that is a subselect**
- **SELECT INTO statement**

# SELECT from INSERT SQLCODEs

**-20165 INSERT STATEMENT WITHIN A SELECT IS NOT ALLOWED IN THE CONTEXT IN WHICH IT WAS SPECIFIED**

**-20166 INSERT STATEMENT WITHIN A SELECT SPECIFIED A VIEW *view-name* WHICH IS NOT A SYMMETRIC VIEW**

# IS NOT DISTINCT FROM

Dealing with NULLs can be tricky.

- Two columns are not equal if both are NULL, but sometimes you want to know that:

```
WHERE COL1 = COL2
OR COL1 IS NULL AND COL2 IS NULL
```

- DB2 V8 offers IS NOT DISTINCT FROM to help.
- The following SQL is logically equivalent to that above, but perhaps simpler to code and understand:

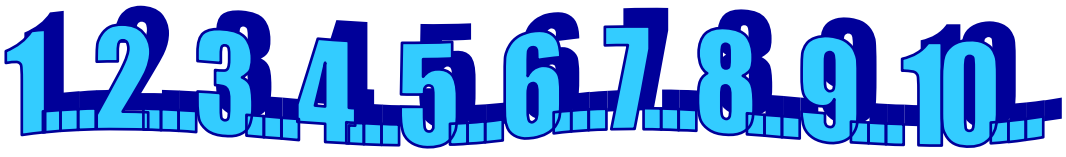
```
WHERE COL1 IS NOT DISTINCT FROM COL2
```

- Also, IS NOT DISTINCT FROM is a Stage 1 predicate

# SEQUENCE Objects

- A sequence is a standalone object which maintains an ordered set of values
  - e.g.) Start with 1, increment by 1
  - Define starting value, increment, maximum value
  - Can CYCLE sequence values

```
CREATE SEQUENCE <sequence-name>
AS  < data type >
START WITH  <numeric value>
INCREMENT BY <numeric value>
NO MINVALUE   /  MINVALUE   <numeric value>
NO MAXVALUE  /  MAXVALUE  <numeric value>
NO CYCLE   / CYCLE
NO CACHE  / CACHE <integer value>
```



# Using SEQUENCEs

- **SEQUENCE Values are Retrieved Using**
  - **NEXT VALUE FOR sequence-name**
    - Generates and returns the next value
  - **PREV VALUE FOR sequence-name**
    - Generates and returns the previous value

- **SEQUENCE Values can be used with:**
  - **SELECT and SELECT INTO**
    - As long as it does not use a DISTINCT keyword, GROUP BY, ORDER BY or UNION
  - **INSERT statement within SELECT clause of fullselect**
  - **UPDATE statement within the SET clause**
    - Searched or positioned
  - **SET host-variable**
    - Except within SELECT of fullselect of expression
  - **VALUES or VALUES INTO**
  - **CREATE PROCEDURE, FUNCTION, TRIGGER**

# Examples Using SEQUENCEs

```
CREATE SEQUENCE order_seq
   START WITH 1
   INCREMENT BY 1
   NOMAXVALUE
   NOCYCLE
   CACHE 20;

INSERT INTO orders (orderno, custno)
VALUES (NEXT VALUE FOR order_seq, 10);

UPDATE orders
SET orderno = NEXT VALUE FOR order_seq
WHERE custno = 10;

SELECT NEXT VALUE FOR order_seq INTO :hv_seq from orders;
```

# IDENTITY Column Enhancements

- **As of V8, IDENTITY columns can be ALTERed**

```
ALTER  TABLE ALTER COLUMN <identity-column-name>
SET GENERATED  ALWAYS / BY DEFAULT
SET INCREMENT BY <numeric value>
SET NO MINVALUE   /  MINVALUE   <numeric value>
SET NO MAXVALUE  /  MAXVALUE  <numeric value>
SET NO CYCLE   / CYCLE
SET NO CACHE  / CACHE <integer value>
RESTART   /  RESTART WITH  <numeric value>
```

# Sequences or Identity Columns?

| Identity Columns | Sequence Objects |
|---|---|
| Internal objects generated and maintained by DB2 | Stand-alone objects created by the DBA |
| Associated with a single table – and only one per each table. | Not associated with any table; can use multiple per table |
| IDENTITY_VAL_LOCAL() to get last value assigned | PREVIOUS VALUE expr to get last value assigned |
| N/A – DB2 handles assigning next value | NEXT VALUE expression gets next value to be assigned |
| Add/change using ALTER TABLE *(V8+ only)* | Administer using ALTER SEQUENCE, DROP, GRANT, REVOKE, COMMENT. |
| CYCLE can encounter problems w/ a unique index if duplicates are created | CYCLE will wrap around and repeat with no uniqueness consideration |
| Available as of V6 refresh | Available only as of V8 |

# SEQUENCE Object SQLCODEs

**-348** *sequence-expression* **CANNOT BE SPECIFIED IN THIS CONTEXT**

**-359 THE RANGE OF VALUES FOR THE IDENTITY COLUMN OR SEQUENCE IS EXHAUSTED**

**-372 ONLY ONE ROWID, IDENTITY, OR SECURITY LABEL COLUMN IS ALLOWED IN A TABLE**

**-373 DEFAULT CANNOT BE SPECIFIED FOR IDENTITY COLUMN OR SECURITY LABEL COLUMN***columnname-seclabel*

# Multiple DISTINCT

- The DISTINCT keyword can be used at the statement level, as in:

```
SELECT DISTINCT C1,C2,C3
FROM T1;
```

- Or, it can be used at the column level, as in:

```
SELECT AVG(C1),COUNT(DISTINCT C2)
FROM T1;
```

- With DB2 V7 you can also have multiple DISTINCT on the same column only, as in:

```
SELECT COUNT(DISTINCT(A1)),
       SUM(DISTINCT A1)
FROM T1;
```

- However, you cannot specify multiple DISTINCT on different columns. If you tried, you received a SQLCODE -127

# Multiple DISTINCT in V8

- Now, in V8, the prior restrictions are removed.
- All of the following are now legal:

```
SELECT DISTINCT COUNT(DISTINCT(A1)), COUNT(A2) FROM T1
SELECT DISTINCT SUM(DISTINCT(A1)), COUNT(A2) FROM T1
SELECT DISTINCT AVG(DISTINCT(A1)), COUNT(A2) FROM T1
SELECT COUNT(DISTINCT(A1)), COUNT(DISTINCT(A2)) FROM T1
SELECT COUNT(DISTINCT(A1)), AVG(DISTINCT(A2)) FROM T1
SELECT DISTINCT COUNT(DISTINCT(A1)), COUNT(A2) FROM T1 GROUP BY A3
SELECT COUNT(DISTINCT(A1)), SUM(DISTINCT(A2)) FROM T1 GROUP BY A3
SELECT COUNT(DISTINCT(A1)) FROM T1 HAVING AVG(DISTINCT(A4)) > 1
SELECT COUNT(DISTINCT(A1)) FROM T1 WHERE A3 > 0 GROUP BY A2 HAVING AVG(DISTINCT(A4)) > 1
```

# Multiple DISTINCT SQLCODE

**The following SQLCODE will no longer apply when you are in DB2 V8 NFM, but it still can be issued prior to NFM:**

- **-127 DISTINCT IS SPECIFIED MORE THAN ONCE IN A SUBSELECT**

# Padded Index Keys

- **Before V8, a VARCHAR in an index was padded to its full length**
  - **Now, true variable length keys are supported**
  - **Syntax :**

    **CREATE/ALTER INDEX …**
    **[ NOT ] PADDED**

  - **Specify either PADDED or NOT PADDED**
    - NOT PADDED is default in NFM
  - **Allows index only access to VARCHAR indexes**
    - Previously DB2 always had to access the table data to get the length information

# Backward Index Scan

- **DB2 can use an ASCending index and use a backward index scan to avoid sorting into DESCending order**
  - **No longer need to create the same index as both ASC and DESC**
  - **ACCESSTYPE in PLAN_TABLE is 'IR'**
- **To be able to use a Backward Index Scan:**
  - Index must be defined on the same columns as ORDER BY
  - Ordering must be exactly opposite of what is requested in ORDER BY
  - For example: index on EMPNO DESC, LASTNME ASC
    - Forward scan for ORDER BY EMPNO DESC, LASTNME ASC
    - Backward scan for ORDER BY EMPNO ASC, LASTNME DESC

# Scalar Fullselect

**A scalar fullselect is when a SQL SELECT statement is used in the place of a value**

**For example:**

```
SELECT EMPNO, SALARY
FROM   EMP E
WHERE PRICE BETWEEN 2 * (SELECT MIN(SALARY) FROM EMP)
AND                .5 * (SELECT MAX(SALARY) FROM EMP);
```

# More on Scalar Fullselect

Rules for naming the result of a scalar-fullselect:

- If the AS clause is specified, the name of the result column is the name specified on the AS clause.
- If the AS clause is not specified and the result column is derived from a column, the result column name is the unqualified name of that column.
- All other result column names are unnamed.

A scalar fullselect cannot be used in the following instances:

- A CHECK constraint in CREATE TABLE and ALTER TABLE statements
- A grouping expression, which is limited to a list of column names
- A view definition that has a WITH CHECK OPTION
- A CREATE FUNCTION (SQL) statement (subselect already restricted from the expression in the RETURN clause)
- A column function
- An ORDER BY clause
- A join-condition of the ON clause for INNER and OUTER JOINs

# Stage 1 for Unlike Data Types

**Prior to V8:**

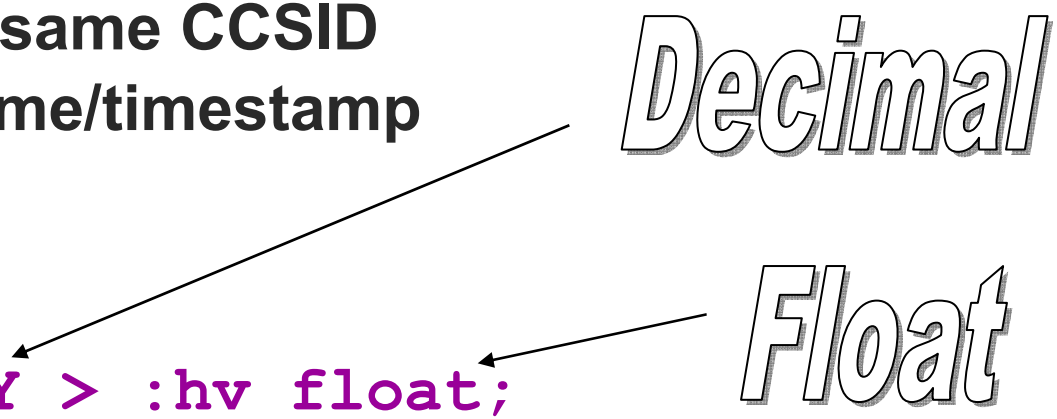
- **if the data types and length of predicate operands did not match, the predicate was considered Stage 2**

**As of V8:**

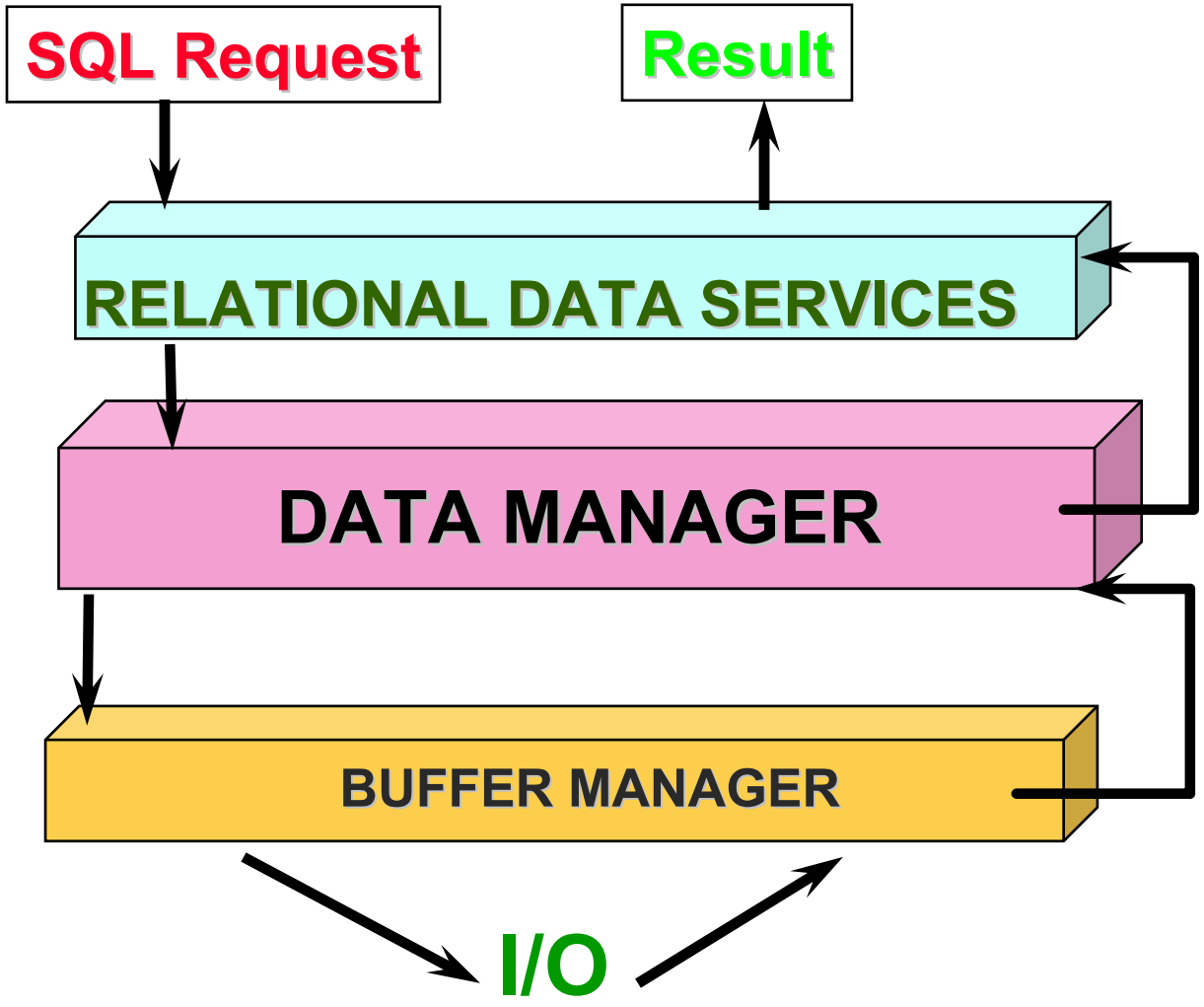
- **if the data types and lengths do not match (within data type family) the predicates can be processed at Stage 1 (and can possibly use an index)**
- **Why?**
  - C does not have a DECIMAL data type, but you might want to access existing tables with DECIMAL columns
  - Java does not have a fixed length character string data type; every string is variable length.
  - It is a common mistake for programmers to mismatch data type and/or length

# Stage 1 Predicates

- Following comparisons are now 'Stage 1' (and can possibly use an index)
  - Unlike Data Types
  - All numeric
  - All strings with same CCSID
  - String to date/time/timestamp
- Example:
  - ```
    SELECT *
    FROM   EMP
    WHERE  SALARY > :hv_float;
    ```
    - Prior to V8 – Stage 2, tablespace scan
    - As of V8 – Stage 1, use index on salary if available


    ```

# Stage 1 versus Stage 2




STAGE 2 - Evaluated after data retrieval (non-sargable) via the RDS (Relational Data Services) which is more expensive than the Data Manager.

STAGE 1 - Evaluated at the time the data rows are retrieved (sargable). There is a performance advantage to using Stage 1 predicates because fewer rows are passed to Stage 2 via the Data Manager

# Data Type Details (1)

**String Data**

- All predicates comparing string types with the same CCSID are stage 1 and indexable except the following:

  graphic/vargraphic -> char/varchar

- In general, predicates comparing graphic/vargraphic to char/varchar are *not* indexable. However, if the char/varchar is Unicode mixed and the predicate is an '=' predicate, then the predicate is indexable.

  char/varchar(n1) -> char/varchar(n2)

  graphic/vargraphic(n1) -> graphic/vargraphic(n2)

  char/varchar(n1) -> graphic/vargraphic(n2)

  …for all of above (n1 > n2 and not '=' pred)

# Data Type Details (2)

## Numeric

- All numeric comparisons are stage 1 and indexable except the following:

  REAL -> DEC(p,s) where p > 15

  FLOAT -> DEC(p,s) where p > 15

# Data Type Details (3)

## Date/Time

- All date, time, and timestamp comparisons are Stage 1 and indexable.
- When comparing date, time, and timestamp data to strings take the following into account:

date/time/timestamp -> string column

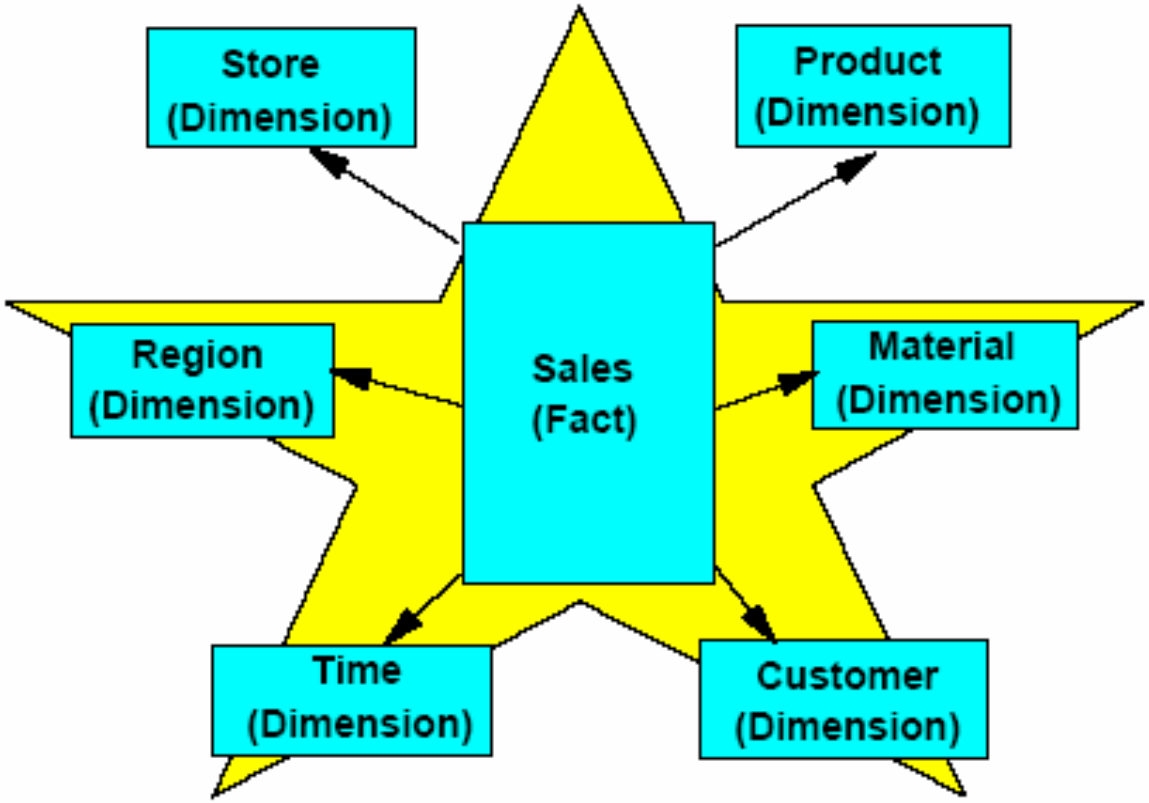
This is stage 2 if the string column is on the inner table or the “column side” of the predicate. If the comparison is the other way around, the predicate is stage 1 and indexable:

string -> date/time/timestamp

# Stage 1 for Unlike Data Types


**More efficient SQL in more applications.**

# Star Join Improvements




Source: IBM

# Sample Star Join Query

```
SELECT *
FROM SALES S,
     TIME T,
     REGION L,
     PRODUCT P
WHERE S.TIME = T.ID
AND   S.REGION = L.ID
AND   S.PRODUCT = P.ID
AND   T.YEAR = 2002
AND   T.QTR = 1
AND   L.CITY IN ('Boston', 'Seattle')
AND   P.ITEM = 'stereo';
```

Fact Table

Dimension Tables

Dimension Tables

Source: IBM

# Sparse Index and Data Caching

## Star Join is more efficient in DB2 V8

- A Star Join requires lots of unindexed ‘work files’ to be materialized…
- DB2 V8 can create a ‘virtual’ index on each work file.
  - Sparse index on the join key for the materialized work files
  - A sparse index is a dynamically built index, pointing to a single value or a range of values, depending on the number of keys that can be stored in a pre-allocated space.
- Work files can be cached in memory
- The Optimizer makes the decisions

# Common Table Expressions

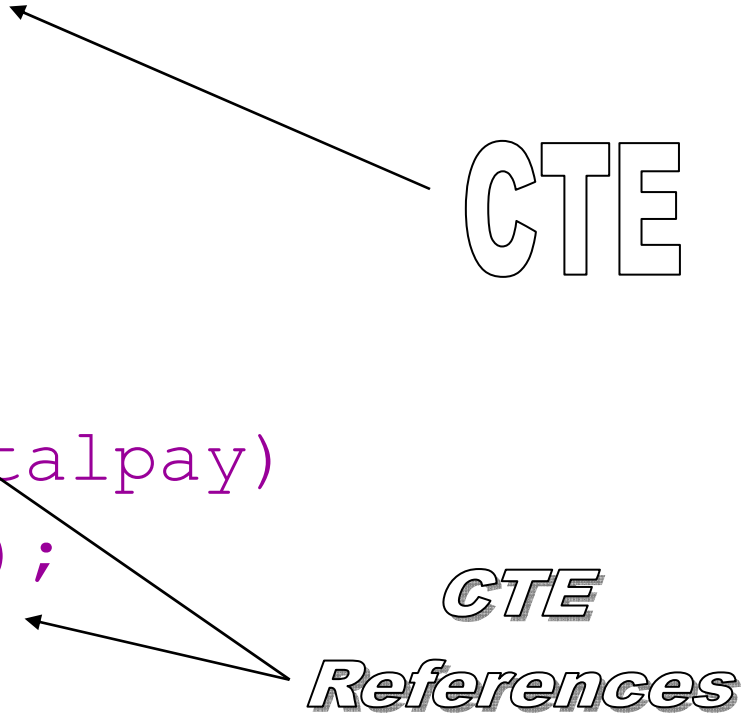
A common table expression, or CTE, allows a SQL statement to be defined using the WITH clause, and then referenced as a table within the rest of the SQL statement.

- Contrast with a nested table expression, or NTE, which is defined in the FROM clause of an SQL statement

```
WITH table-name (column-list) AS fullselect
```

# CTE Example

```
WITH DTOTAL (deptno, totalpay) AS
    (SELECT deptno, sum(salary+bonus)
     FROM   employee
     GROUP BY deptno)
SELECT deptno
FROM   DTOTAL
WHERE totalpay =
      (SELECT max(totalpay)
       FROM   DTOTAL);
```



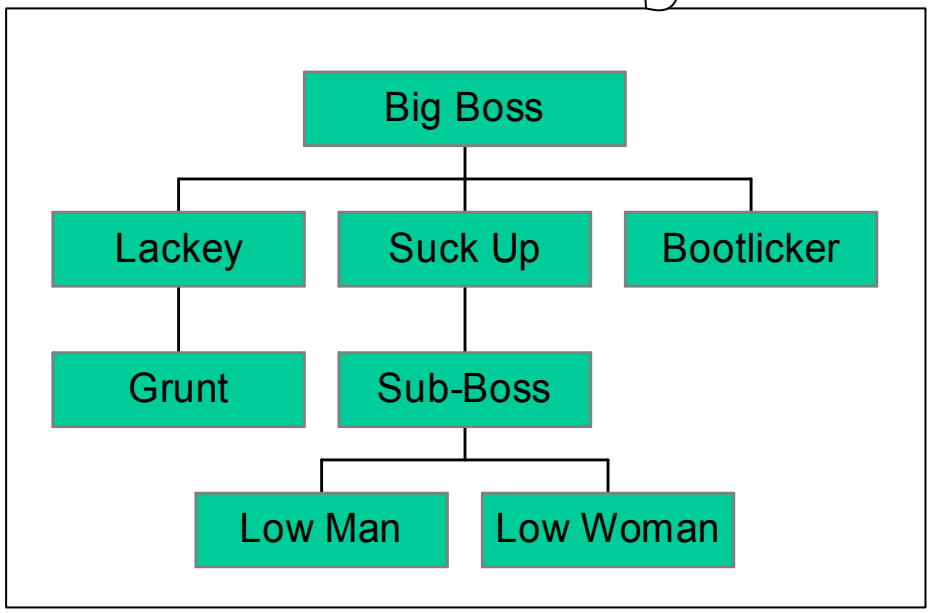
# Recursive SQL

CTEs also enable us to write recursive SQL.

- Fullselect of common table expression refers to itself in the FROM clause
- Can be quite useful for bill of materials and organization hierarchy explosions
  - A single SQL statement can be used to traverse hierarchies
- Be sure to add controls to stop

# Recursive SQL Setup

## Hierarchy




## Data

| MGR_ID | EMP_ID | EMP_NAME |
|---|---|---|
| -1 | 1 | BIG BOSS |
| 1 | 2 | LACKEY |
| 1 | 3 | SUCKUP |
| 1 | 4 | BOOTLICKER |
| 2 | 5 | GRUNT |
| 3 | 6 | SUB-BOSS |
| 6 | 7 | LOW MAN |
| 6 | 8 | LOW WOMAN |

## DDL

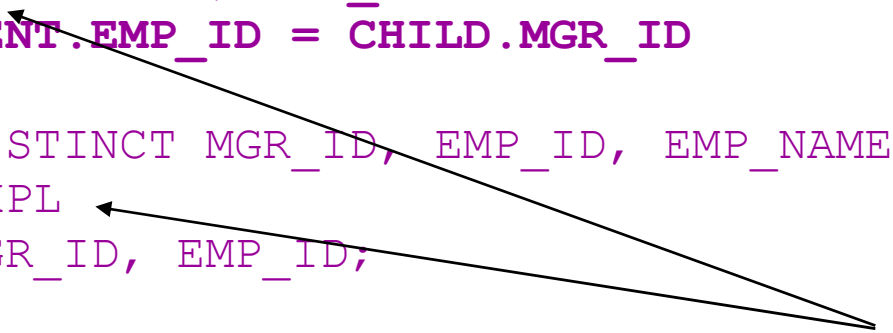
```
CREATE TABLE ORG_CHART
  (MGR_ID        SMALLINT,
   EMP_ID        SMALLINT,
   EMP_NAME      CHAR(20));
```

# Recursive SQL Example

```
WITH EXPL (MGR_ID, EMP_ID, EMP_NAME) AS
(
 SELECT ROOT.MGR_ID, ROOT.EMP_ID, ROOT.EMP_NAME
 FROM   ORG_CHART   ROOT
 WHERE  ROOT.MGR_ID = 3

 UNION ALL

 SELECT CHILD.MGR_ID, CHILD.EMP_ID, CHILD.EMP_NAME
 FROM   EXPL PARENT, ORG_CHART CHILD
 WHERE  PARENT.EMP_ID = CHILD.MGR_ID
)
 SELECT   DISTINCT MGR_ID, EMP_ID, EMP_NAME
 FROM     EXPL
 ORDER BY MGR_ID, EMP_ID;
```

CTE References

# Results of the Query

The results of running this query would be:

| MGR ID | EMP ID | EMP NAME |
|---|---|---|
| 1 | 3 | SUCKUP |
| 3 | 6 | SUB-BOSS |
| 6 | 7 | LOW MAN |
| 6 | 8 | LOW WOMAN |

# Another Recursive Example

```
WITH NUMGEN(NBR) AS
   (VALUES(1)

    UNION ALL

    SELECT NBR+1
    FROM NUMGEN
    WHERE NBR < 100)

SELECT NBR
FROM NUMGEN;
```

**This query generates numbers from 1 to 100.**

- **Be careful though.**
- **What happens if we remove the WHERE clause?**
- **Endless Loop!**

# Recursive SQLCODEs

**+347 THE RECURSIVE COMMON TABLE EXPRESSION *name* MAY CONTAIN AN INFINITE LOOP**

***Note*: SQLCODEs that impact CTEs (which are required for recursive SQL) follow on the next slide**

# CTE SQLCODEs

**-340 THE COMMON TABLE EXPRESSION *name* HAS THE SAME IDENTIFIER AS ANOTHER OCCURRENCE OF A COMMON TABLE EXPRESSION DEFINITION WITHIN THE SAME STATEMENT**

**-341 A CYCLIC REFERENCE EXISTS BETWEEN THE COMMON TABLE EXPRESSIONS *name1* AND *name2***

**-342 THE COMMON TABLE EXPRESSION *name* MUST NOT USE SELECT DISTINCT AND MUST USE UNION ALL BECAUSE IT IS RECURSIVE**

**-343 THE COLUMN NAMES ARE REQUIRED FOR THE RECURSIVE COMMON TABLE EXPRESSION *name***

**-344 THE RECURSIVE COMMON TABLE EXPRESSION *name* HAS MISMATCHED DATA TYPES OR LENGTHS FOR COLUMN *column-***

# CTE SQLCODEs (2)

**-345 THE FULLSELECT OF THE RECURSIVE COMMON TABLE EXPRESSION *name* MUST BE THE UNION OF TWO OR MORE FULLSELECTS AND MUST NOT INCLUDE COLUMN FUNCTIONS, GROUP BY CLAUSE, HAVING CLAUSE, OR AN EXPLICIT JOIN INCLUDING AN ON CLAUSE**

**-346 AN INVALID REFERENCE TO COMMON TABLE EXPRESSION *name* OCCURS IN THE FIRST FULLSELECT, AS A SECOND OCCURRENCE IN THE SAME FROM CLAUSE, OR IN THE FROM CLAUSE OF A SUBQUERY**

# GROUP BY an Expression

**DB2 V8 allows grouping by an expression.**

- A grouping-expression is an expression used in defining grouping.
- Each column-name included in a grouping-expression must unambiguously identify a column in the result set.
- A grouping-expression cannot include a scalar-fullselect or any function that is non-deterministic or is defined to have an external action.
- Columns with a LOB data type (or distinct type for which the source type is a LOB) cannot be used in a grouping-expression.
- Correlated columns or host variables cannot be used in a grouping-expression.
- The length attribute of each grouping-expression cannot exceed 255 for a character expression or 127 for a graphic expression.

# Example

Source: IBM

```
SELECT SUBSTR(C1,LENGTH(C3), 1),
       C2 + C3,
       C3 + C2,
       C2 + C3 + 4,
       C1||CHAR(C2),
       MAX(C2)
FROM   T1
GROUP BY C1, C2,
         C2 + C3,
         SUBSTR(C1,LENGTH(C3),1)
HAVING C2 + 2 = 177.1 AND
       C2 + C3 = 277.2;
```

Grouping Expressions

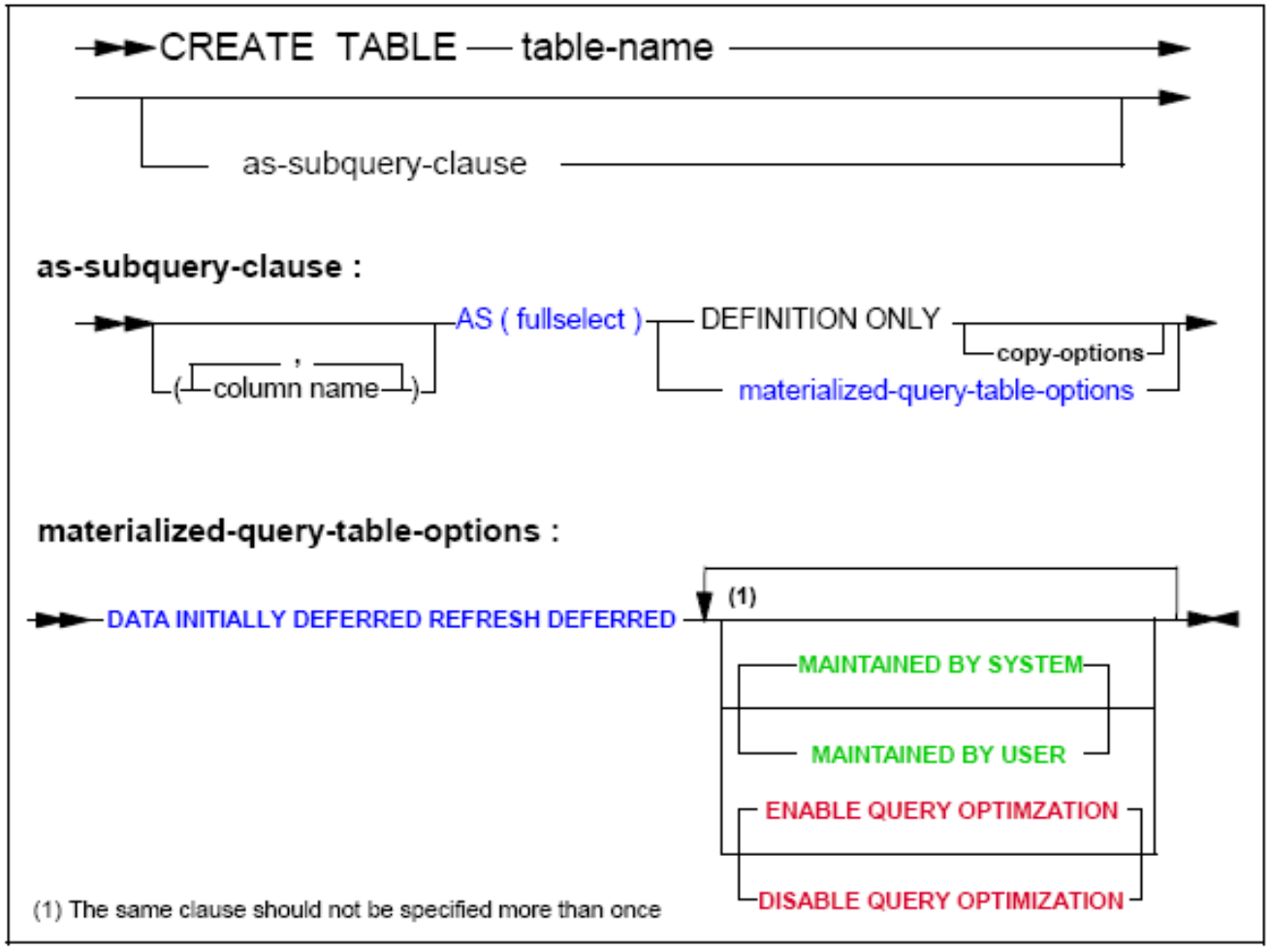
Matches Grouping Expression C2+C3

# Materialized Query Tables

**Previously known as Automatic Summary Tables (in DB2 for LUW)**

- Purpose is to speed up dynamic queries – very useful for decision support and data warehousing queries. The results are pre-computed and stored.
- Uses include:
  - Scanning or processing large amounts of data (terabytes)
  - Performing multiple joins
  - Performing complex aggregations
- An MQT is a table of derived data that is maintained by DB2
- An MQT can be thought of as a view that has been materialized – that is, a view whose data is physically stored instead of virtually accessed when needed.
  - Defined via CREATE TABLE with fullselect
  - Can populate immediately or defer
  - Can be system maintained, or user maintained
  - Can refresh with each update, or REFRESH TABLE
  - Can enable automatic query rewrite, or not
- Can run utilities on them (UNLOAD, REORG, COPY, RECOVER)
- Appear in the DB2 Catalog (SYSIBM.SYSTABLES)

# MQT Syntax



# MQT Example

```
CREATE TABLE DEPT_SAL
   (DEPT, TOTAL_SALARY, TOTAL_BONUS, TOTAL_COMM,
    TOTAL_COMPENSATION, EMPLOYEES)
AS (SELECT   WORKDEPT, SUM(SALARY),
             SUM(BONUS)  SUM(COMM),
             SUM(SALARY+BONUS+COMM), COUNT(*)
    FROM     DSN8810.EMP
    GROUP BY WORKDEPT)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY SYSTEM
ENABLE QUERY OPTIMIZATION;
```

MQT Controls

# Dynamic Scrollable Cursors

- Status Cursors scrollable with DB2 V7
  - Sensitive and Insensitive to updates while open
- Dynamic Scrollable Cursor in V8
  - Allows for 'backwards' index scan and backward sequential detect to avoid sort
  - Does not materialize the result table
  - Scrolls directly on the base table
    - Therefore sensitive to all committed INSERT, UPDATEs, and DELETEs
  - Supported by Index and Table Scan access paths
  - Dynamic scrollable cursors may not be appropriate in some cases (for example, if you need to scroll backward to see data in its same prior state)

# Useful New Built-In Functions

- GETVARIABLE()
- GENERATE_UNIQUE()
- Encryption/Decryption
  - ENCRYPT(), ENCRYPT_TDES()
  - DECRYPT_BIT(), DECRYPT_CHAR(), DECRYPT_DB()
  - GETHINT()

# GETVARIABLE()

## GETVARIABLE()

- Returns string value for setting of session variable or built-in variable
- DB2-defined session variables
  - DATA_SHARING_GROUP_NAME
  - PACKAGE_NAME
  - PACKAGE_SCHEMA
  - PACKAGE_VERSION
  - PLAN_NAME
  - SECLABEL
  - SYSTEM_NAME
  - SYSTEM_ASCII_CCSID
  - SYSTEM_EBCDIC_CCSID
  - SYSTEM_UNICODE_CCSID
  - VERSION

# GENERATE_UNIQUE()

## GENERATE_UNIQUE()

- Generates a unique value across the Sysplex
  - CHAR (13) FOR BIT DATA
  - Universal Time Coordinated UTC and Sysplex member (if in Sysplex)
  - The value is not readable, but you can use the TIMESTAMP function to get the time value out of it:

```
SELECT TIMESTAMP(GENERATE_UNIQUE())
```

- Can be used to generate a distinct value for each row of a multiple row insert statement
  - Can be used instead on CURRENT TIMESTAMP for uniqueness
- Provides consistency across the DB2 Family (LUW)

# Encryption / Decryption

## Encryption – to encrypt the data for a column

```
ENCRYPT_TDES(string, password, hint)
```

- ENCRYPT_TDES [can use ENCRYPT() as a synonym for compatibility]
- Triple DES cipher block chaining (CPC) encryption algorithm
  - Not the same algorithm used by DB2 on other platforms
- 128-bit secret key derived from password using MD5 hash

```
INSERT INTO EMP (SSN)
VALUES(ENCRYPT('289-46-8832','TARZAN','? AND JANE'));
```

## Decryption – to decrypt the encrypted data for a column

```
DECRYPT_BIT(), DECRYPT_CHAR(), DECRYPT_DB()
```

- Can only decrypt expressions encrypted using ENCRYPT_TDES
  - Can have a different password for each row if needed
- Without the password, there is no way to decrypt

```
SELECT DECRYPT_BIT(SSN,'TARZAN') AS SSN FROM EMP;
```

# Decryption Details

| Function | Type of first argument | Actual type of encrypted data | Result |
|---|---|---|---|
| DECRYPT_BIT | FOR BIT DATA[1] | CHAR, VARCHAR | VARCHAR FOR BIT DATA |
| DECRYPT_BIT | FOR BIT DATA | GRAPHIC, VARGRAPHIC (UTF16) | Warning or error |
| DECRYPT_BIT | FOR BIT DATA | GRAPHIC, VARGRAPHIC (not UTF16) | Warning or error |
| DECRYPT_CHAR | FOR BIT DATA | CHAR, VARCHAR | VARCHAR(3) |
| DECRYPT_CHAR | FOR BIT DATA | GRAPHIC, VARGRAPHIC (UTF16) | VARCHAR(3) |
| DECRYPT_CHAR | FOR BIT DATA | GRAPHIC, VARGRAPHIC (not UTF16) | Warning or error |
| DECRYPT_DB | FOR BIT DATA | CHAR, VARCHAR, GRAPHIC, VARGRAPHIC | VARGRAPHIC |

**Note 1:** FOR BIT DATA means CHAR or VARCHAR FOR BIT DATA

# GETHINT()

- GETHINT() - Function that allows hint to be retrieved

```
SELECT GET_HINT(SSN) AS HINT
FROM   EMP;
```

- Hint is set when encrypting or by:
  - SET ENCRYPTION PASSWORD = ‘password’ (6-127 bytes)
    - WITH HINT = ‘hint string’ (1-32 bytes)
  - Will be used by functions if password is not provided

```
SET ENCRYPTION PASSWORD = 'TARZAN';

INSERT INTO EMP(SSN)
VALUES ENCRYPT_TDES('289-46-8832');

SELECT DECRYPT_CHAR(SSN) FROM EMP;
```

  - “ENCRYPTION PASSWORD” is not a special register
    - There is no way to retrieve its value
    - If the hint does not help you then you are out of luck

# Encryption-Related SQLCODEs

**-20143 THE ENCRYPTION OR DECRYPTION FUNCTION FAILED, BECAUSE THE ENCRYPTION PASSWORD VALUE IS NOT SET**

**-20144 THE ENCRYPTION IS INVALID BECAUSE THE LENGTH OF THE PASSWORD WAS LESS THAN 6 BYTES OR GREATER THAN 127 BYTES**

**-20146 THE DECRYPTION FAILED. THE DATA IS NOT ENCRYPTED**

**-20147 THE ENCRYPTION FUNCTION FAILED. MULTIPLE PASS ENCRYPTION IS NOT SUPPORTED**

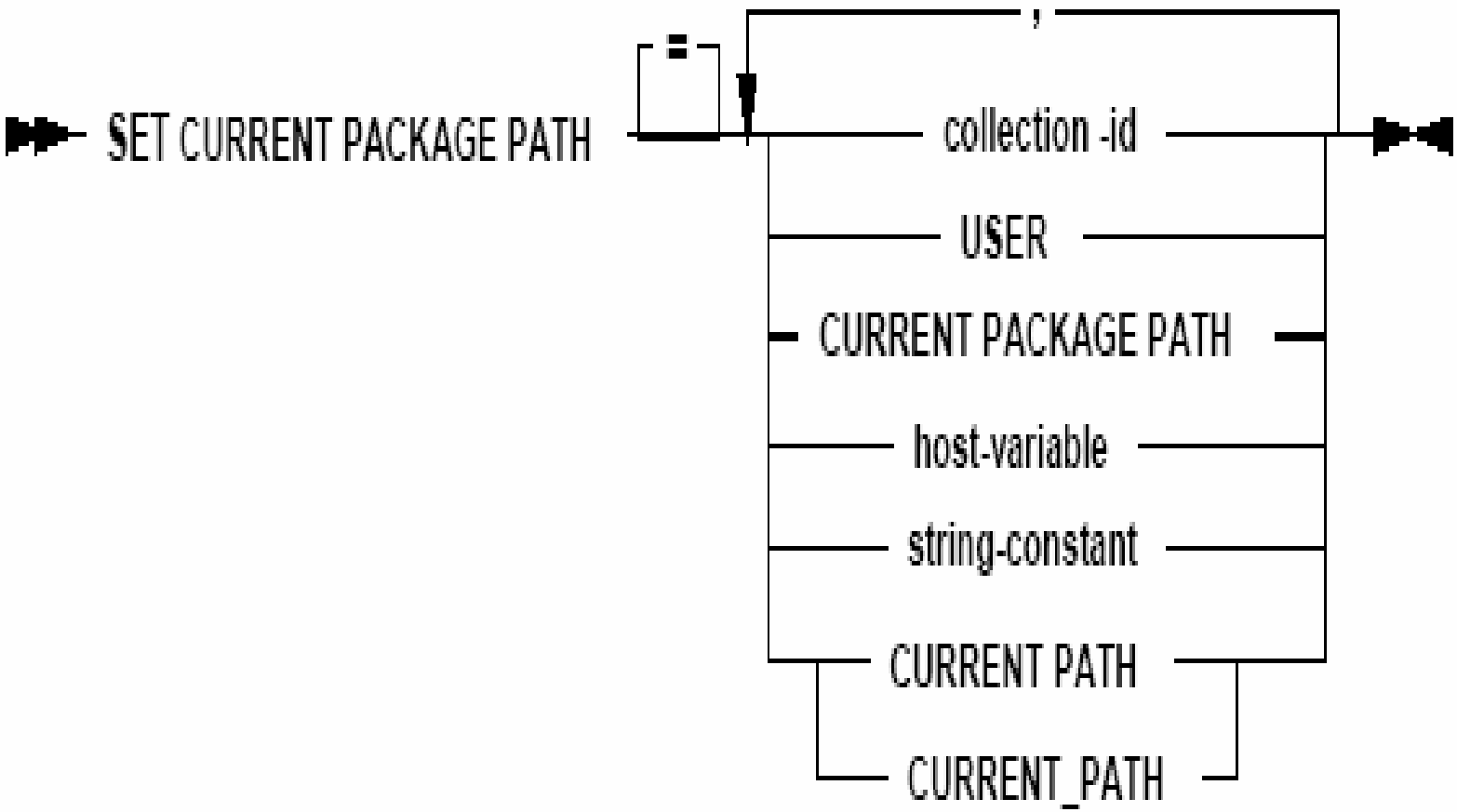
**-20223 THE ENCRYPT_TDES OR DECRYPT FUNCTION FAILED. ENCRYPTION FACILITY NOT AVAILABLE** *return-code*, *reason-code*

# CURRENT PACKAGE PATH Special Register

For resolution of the package collection:

- You can use this special register to specify a list of package collections to DB server (similar to PKLIST on BIND PLAN)
- Once specified, the Database server (instead of the application requester) searches the list and finds the package
- You can use it for applications that do not run under a DB2 plan (and hence do not have a PKLIST)

# Syntax


SET CURRENT PACKAGE PATH
=
,
collection -id
USER
CURRENT PACKAGE PATH
host-variable
string-constant
CURRENT PATH
CURRENT_PATH

# CM and ENFM SQLCODE

**Remember, although there are a lot of new features and functionality in DB2 V8, *most of it is not available until you are in New Function Mode* (NFM).**

- **If you try to use a feature in CM or ENFM that is not supported until NFM, you get:**
  - **-4700 ATTEMPT TO USE NEW FUNCTION BEFORE NEW FUNCTION MODE**

# Don't Expect to Proceed Without Re-evaluating Good Old COBOL

Older COBOL compilers are no longer supported:

- OS/VS COBOL (out of support since 1994)
- VS COBOL II (out of support since 2001)
  - An existing COBOL II load module running in LE will still work though
  - But no more compiles for OS/VS COBOL or VS COBOL II
- DB2 V8 supports:
  - IBM COBOL V2.2 *(already end of service though!)*
  - Enterprise COBOL V3.2 or later
    - But end of service for V3.2 was back in October 2005

# More on COBOL

See also the following Techdocs, available at:

- http://www.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=v8&uid=swg21164264&loc=en_US&cs=utf-8&lang=en
- http://www.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=v8&uid=swg21166881&loc=en_US&cs=utf-8&lang=en

In case you are not on any of the supported releases, you will need to migrate. In that case, be sure to review the following:

- *Enterprise COBOL for z/OS Compiler and Runtime Migration Guide* (GC27-1409).
- http://www.ibm.com/software/awdtools/cobol/zos/library/

# BIND Parameter: REOPT(ONCE)

**REOPT(ONCE) is a new BIND option that sort of combines the benefits of REOPT(VARS) and dynamic statement caching:**

- **Access paths for dynamic SQL are normally determined at PREPARE**
- **The REOPT parameter controls when an access path is obtain (reoptimized) for a statement:**

exists

- **REOPT(VARS)** - reprepares statements at run time when the input variable values are available, so that the optimizer can re-optimize the access path using the host variable values.

V8

- **REOPT(ONCE)** - reoptimizes the access path only once (using the first set of input variable values) no matter how many times the same statement is executed.
- **REOPT(ALWAYS)** - equivalent to REOPT(VARS)
- **REOPT(NONE)** - equivalent to NOREOPT(VARS)

# New Precompiler Option: NEWFUN

- NEWFUN(YES/NO) is precompiler parameter
  - NEWFUN(NO) – disables use of new SQL functionality for static SQL and creates a V7 compatible DBRM
- Default for NEWFUN is changed to YES after you go to NFM
  - If you set NEWFUN(YES) the DBRMs will be Unicode
  - If you set NEWFUN(NO) the DBRMs will be EBCDIC
  - BIND controls storage in system catalog
    - Binding Unicode DBRMs populates SYSSTMT and SYSPACKSTMT in Unicode
    - Binding EBCDIC DBRMs populates SYSSTMT and SYSPACKSTMT in EBCDIC

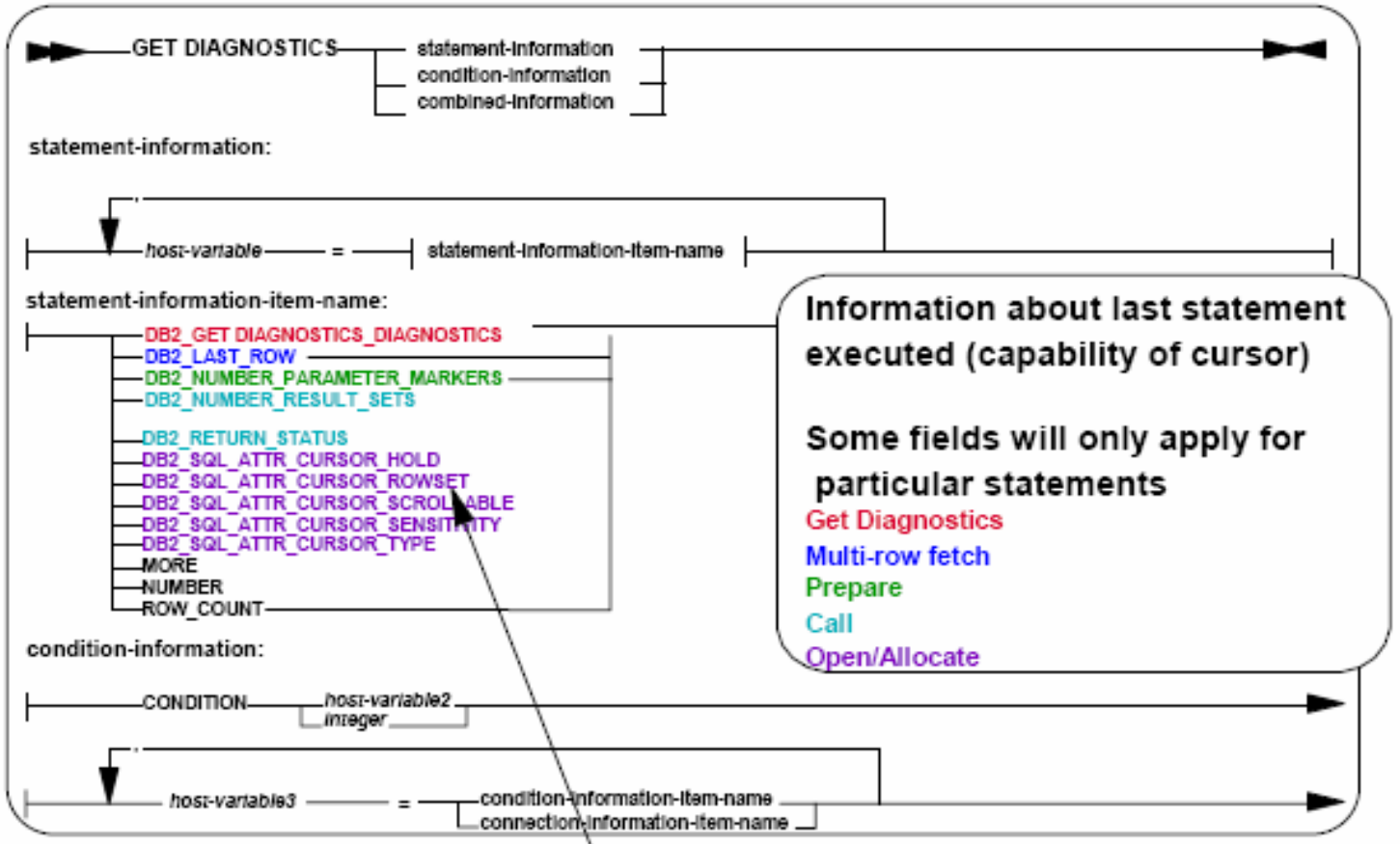
# GET DIAGNOSTICS

## GET DIAGNOSTICS statement

- Returns SQL error and diagnostic information and can be performed for
  - the entire statement
  - each condition (when multiple conditions occur)
- Enables more diagnostic information to be returned than can be contained in SQLCA
  - SQL error message tokens larger than 70 bytes are supported whereas the SQLCA cannot

# GET DIAGNOSTICS

## GET DIAGNOSTICS syntax




Indicates if cursor can be used for rowset positioned operations, i.e., multi-fetch

# Example

Here is a quick example to get the return code and error count:

```
GET DIAGNOSTICS
  :db2rc = DB2_RETURN_STATUS,
  :errcount = NUMBER;
```

But simply returning the SQLCODE may no longer be enough.

- After detecting an error you might need to retrieve error information for multiple errors that may have occurred.

# Another Example

## Looping to get all the error information:

```
MOVE 1 TO CTR.
PERFORM GETDIAG UNTIL CTR > ERRCNT.

GETDIAG.
  EXEC SQL
     GET DIAGNOSTICS CONDITION :CTR
     :errsqlcode = DB2_RETURNED_SQLCODE,
     :errsqlstate = RETURNED_SQLSTATE,
     :errmessage = MESSAGE_TEXT
  END-EXEC.

  ADD 1 TO CTR.

  IF :errsqlcode <> 0 THEN …
```

# EXPLAIN Enhancements

- **New PLAN_TABLE columns** *(next slide)*
- **Can access a PLAN_TABLE using an alias**
- **Enhanced Visual EXPLAIN**
  - Uses additional "explain" tables
  - Can now show more information, including:
    - **Single predicate filter factor estimates**
    - **Qualified row estimates**
    - **Predicate information (Matching, screening, Stage 1, Stage 2)**
    - **Limited partition scan details**
    - **Parallelism details**
    - **Sort estimation**
  - Enhanced reporting
    - **Generate HTML reports**
    - **XML output file**
    - **Generate information for IBM Service Team**

```
CREATE ALIAS yourid.PLAN_TABLE
FOR groupid.PLAN_TABLE;
```

# New PLAN_TABLE Columns

**TABLE_ENCODE CHAR(1) -** indicates the encoding scheme of the table. If the table has a single CCSID set, then the column will contain 'E' for EBCDIC 'A' for ASCII or 'U' for Unicode. If the table contains multiple CCSID sets, then the column will be set to 'M' for multiple CCSID sets.

**TABLE_SCCSID FIXED(16) -** the SBCS CCSID value of the table. If the TABLE_ENCODE column is 'M', the value is zero.

**TABLE_MCCSID FIXED(16) -** the Mixed CCSID value of the table. If the TABLE_ENCODE column is 'M', the value is zero.

**TABLE_DCCSID FIXED(16) -** the DBCS CCSID value of the table. If the TABLE_ENCODE column is 'M', the value is zero.

**ROUTINE_ID INTEGER -** used to pinpoint the table function record from SYSIBM.SYSROUTINES.

# And There is More…

**Java**

- **Universal Driver, Type 4 Drivers, SQLJ Improvements**

**Unicode**

- **SQL is converted to Unicode before it is parsed**
- **Unicode, EBCDIC and ASCII tables and host variables can be used within the same SQL statement**

**SQL Procedure Language**

- **Enhancements have been made to help better support the SQLProcedure Language**
  - New statements include RETURN, SIGNAL and RESIGNAL, SET_MESSAGE_TEXT, and GET DIAGNOSTICS
- **Statements up to 2MB**
- **Integrated debugger**
- **ITERATE statement**

**DB2 Managed Stored are no longer supported**

- **Must go to WLM**

# The End

Time for a beer…

# References

- IBM Redbook: DB2 V8 Everything You Wanted to Know
- IBM DB2 V8 Administration Guide
- IBM DB2 V8 Application and SQL Guide
- IBM DB2 V8 SQL Reference Guide
- DB2 Developer's Guide, *5th edition*

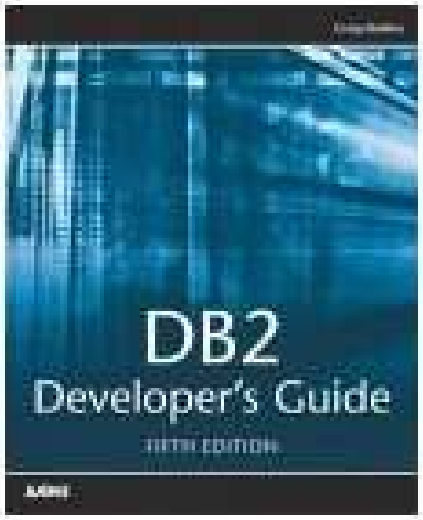
# Craig S. Mullins

**Mullins Consulting, Inc.**
**15 Coventry Court**
**Sugar Land, TX 77479**

**Craig@CraigSMullins.com**

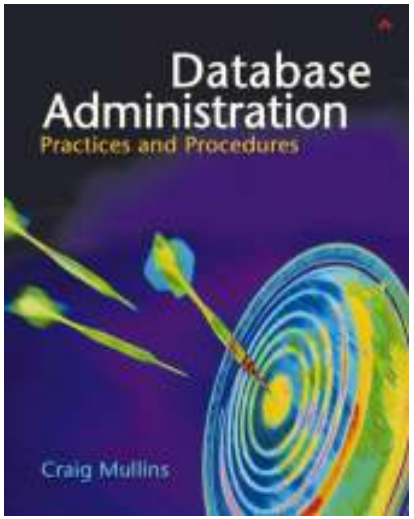
**http://www.craigsmullins.com**


## Available Now




**DB2 Developer's Guide, *5ed***

**www.craigsmullins.com/cm-book.htm**




**DBA: Practices & Procedures**

**www.craigsmullins.com/dba_book.htm**

# About SoftBase Systems

**SoftBase is a leading provider of application testing and tuning solutions for IBM's DB2 database utilizing the OS/390 and z/OS operating systems. SoftBase solutions enable its customers to build and maintain high-quality DB2 applications that run as reliably and cost-effectively as possible. SoftBase was founded in 1987 and is recognized globally for its long-term service and commitment to DB2 mainframe customers.**

**SOFTBASE SYSTEMS, INC.**
**1200 Ridgefield Blvd. Suite 290**
**Asheville, NC USA 28806**
**sales@softbase.com**
**support@softbase.com**
**828-670-9900**