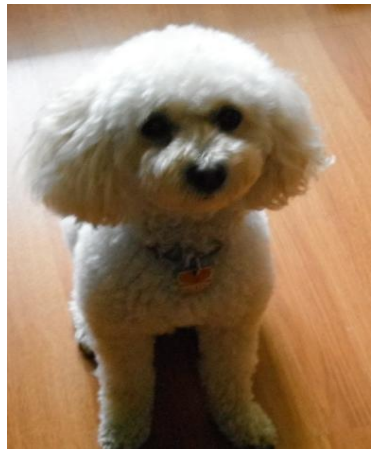
My Favorite Things in DB2 11 for z/OS

Martin Hubel
+ 1 905-764-7498
martin@mhubel.com
www.mhubel.com

Frame of Reference

- I have worked with DB2 for z/OS since 1985
  - I started with DB2 for LUW in 1993
  - I wrote my 1st New Features class for DB2 V2R3 in 1992
- The “big ticket” items get the most press
- There are many cool features that you can discover in each DB2 version
  - This is my list of less-discussed items in DB2 11



Agenda

- A brief history of DB2 time
  - New archive tables added in DB2 11
- Smoke ‘em if you got ‘em
  - Cube and rollup grouping functions
- The line forms to the left
  - Arrays and new functions to support them

Transparent archiving of temporal data

- DB2 11 archiving uses two table approach
- Current data is in archive-enabled table
  - Pre-existing rows in archive table
  - Application can design its own way to archive data
  - Or DB2 can automatically move rows deleted from archive-enabled table to associated archive table
    - Retrieval of data from base table plus its associated archive table controlled by setting built-in system defined global variable without changing SQL
- New global variable controls whether DB2 automatically moves deleted rows to the archive table
  - Can also use LOAD utility with resume behavior to archive data

Archiving of temporal data

- ALTER TABLE extended with ENABLE ARCHIVE
  - Changes an existing table into an archive-enabled table with an associated archive table
  - You can use the table as the archive table is specified in the USE clause.
  - Defining a table as an archive-enabled table results in package invalidation
- For archive-enabled tables, you do not need to change the application
  - No DBA intervention to recall data required
- NOTE: A table cannot be defined as both an archive-enabled table and a system-period temporal table
- Two global variables used to control archiving:
  - SYSIBMADM.GET_ARCHIVE
  - SYSIBMADM.MOVE_TO_ARCHIVE

Archiving of temporal data

- After a table is defined as an archive-enabled table:
  - ALTER TABLE with ADD COLUMN clause implicitly adds column to associated archive table
  - If SYSIBMADM.GET_ARCHIVE is set to Y:
    - Data is retrieved from the archive table when an archive-enabled table is referenced in a table-reference
    - Access of historical data in the archive table is “transparent” to the application
    - Allows application to see active and archive data without modifying SQL statements in multiple packages
    - DB2 rewrites the query with UNION ALL operator

Archiving of temporal data

- If SYSIBMADM.MOVE_TO_ARCHIVE is Y
  - Historical data stored in archive table when a row is deleted in an archive-enabled table.
  - Storing of a row of historical data in archive table is “transparent” to the application
  - When the global variable is set to Y, an update operation will return an error

Archiving of temporal data

- Any INSERT, UPDATE, DELETE, or MERGE will not include rows of the associated archive table
- Cannot reference temporal tables and archive-enabled tables in same DML
  - Neither system-period or application-period tables
  - Both tables are considered for transparent archive transformations

Controls of archive transparency

- Bind/routine options added that control SYSIBMADM.GET_ARCHIVE global variable:
  - ARCHIVESENSITIVE (default YES)
    - BIND/ REBIND/ (TRIGGER) PACKAGE
    - CREATE TRIGGER (implicit trigger package)
  - ARCHIVE SENSITIVE (default YES)
    - CREATE/ALTER FUNCTION (SQL scalar) or PROCEDURE (SQL native)

Sample for enabling archive transparency

- To use archive transparency, need two tables, issue ALTER TABLE to define relationship

```
-- Main table which will be archive enabled
CREATE TABLE POLICY_INFO_AET
(POLICY_ID CHAR(10) NOT NULL,
COVERAGE INT NOT NULL);

-- Archive table to store archive data
CREATE TABLE POLICY_INFO_ARC
(POLICY_ID CHAR(10) NOT NULL,
COVERAGE INT NOT NULL);

-- ARCHIVE ENABLE -- ALTER table to enable archive transparency
ALTER TABLE POLICY_INFO_AET ENABLE ARCHIVE USE POLICY_INFO_ARC;
```

- ALTER statement to add a column or multiple columns on an archive-enabled table also adds the same column or columns to an archive table

Inserting rows into archive enabled table

- The INSERT, UPDATE, and MERGE statements are all blocked in archive mode. These statements use the following options:
  - If SYSIBMADM.MOVE_TO_ARCHIVE = ‘Y’
    - INSERT, UPDATE, and MERGE statements fail - assumed in archive mode, you only DELETE
  - If SYSIBMADM.MOVE_TO_ARCHIVE = ‘N’
    - No archive mode failure occurs (that is, it is business as usual)
  - If SYSIBMADM.MOVE_TO_ARCHIVE = ‘E’
    - Similar to Y, but with flexibility
    - No restriction on data change statements
    - Users that favor the restriction can set the global variable to Y, and users that do not want the restriction can set the global variable to E

Inserting rows into archive enabled table

```
SET SYSIBMADM.MOVE_TO_ARCHIVE = 'N';
INSERT INTO POLICY_INFO_AET
(SELECT J_POLICY, COVERAGE_NBR, CHG_TIMESTAMP
FROM DSN81110.TEMP)
```

- INSERT statement attempted on an archive enabled table with SET SYSIBMADM.MOVE_TO_ARCHIVE = 'Y'.

```
DSNT408I SQLCODE = -20555, ERROR: AN ARCHIVE-ENABLED
   TABLE IS NOT ALLOWED IN THE SPECIFIED CONTEXT.
   REASON CODE 2
```

- A single DELETE statement can trigger transparent archive when MOVE_TO_ARCHIVE is on
  - No additional privilege is required on an archive table
  - Only the privileges on the delete of an archive-enabled table are needed

Deleting rows into archive enabled table

- Given an SQL DELETE from an archive enabled table, and ARCHIVESENSITIVE option YES or NO:
  - If SYSIBMADM.MOVE_TO_ARCHIVE = ‘Y’, for each row deleted, DB2 inserts into corresponding archive table
  - If SYSIBMADM.MOVE_TO_ARCHIVE = ‘N’ (default), DB2 does no data propagation
- Basically an SQL performance improvement to help archiving data with a single DELETE
  - No change on the data-change SQL statements (that is, transparent to the application)

```
SET SYSIBMADM.MOVE_TO_ARCHIVE = 'Y';
DELETE FROM POLICY_INFO_AET
WHERE UPDATE_TS < CURRENT TIMESTAMP - 7 YEARS;
```

Querying archive enabled table

- No need to change existing SQL statements to get current data only or to get both current and archive data
  - Setting of SYSIBMADM.GET_ARCHIVE built-in global variable offers transparent access to archive data
- If the application wants to get the result from both base and archive table, use the following option:
  - SET SYSIBMADM.GET_ARCHIVE = ‘Y’ ;
- If the application just wants to access the active data, use the following default option:
  - SET SYSIBMADM.GET_ARCHIVE = ‘N’ ;

```
SET SYSIBMADM.GET_ARCHIVE = 'Y' ;
SELECT * FROM POLICY_INFO_AET
```

DISABLE ARCHIVE

- **ALTER TABLE with DISABLE ARCHIVE clause**
  - **Table is no longer treated as an archive-enabled table**
  - **Data in both tables is unaffected**
  - **The archive table is not dropped - only the relationship between the two tables is removed**
  - **Subsequent queries will not consider rows in the archive table and deleted rows will not be moved to the archive table**

```
-- ALTER table to disable archive transparency
ALTER TABLE POLICY_INFO_AET DISABLE ARCHIVE;
```

- **After ALTER TABLE ... DISABLE ARCHIVE, the packages and statements in dynamic statement cache are invalidated**

Archiving Summary

- Archiving provides a way to save data elsewhere for archival purposes
  - Likely a stand alone operation
  - Periodically done
- System-period temporal tables provide more of an audit functionality
  - Rows images stored

CUBE, ROLLUP and GROUPING SETS

- Grouping-sets and super-groups are two new options under GROUP BY of SELECT
- Super-group stands for ROLLUP, CUBE or grand-total clause
  - ROLLUP is helpful in providing subtotaling along a hierarchical dimension such as time or geography
  - CUBE is helpful in queries that aggregate based on columns from multiple dimensions
- SQL coding complexity can be reduced greatly and SQL performance can be improved dramatically

Grouping Sets

- Can be considered as the union of two or more groups of rows into a single result set
  - Logically equivalent to union of multiple subselects with the group by clause in each subselect corresponding to one grouping set
- Sample:

```
SELECT WORKDEPT, EDLEVEL, SEX,
  SUM(SALARY) as SUM_SALARY,
  AVG(SALARY) as AVG_SALARY, COUNT(*) as COUNT
 FROM DSN81110.EMP WHERE SALARY > 20000
 GROUP BY GROUPING SETS (WORKDEPT,EDLEVEL,SEX)
```

- Result is logically equivalent to the union all of three subselects
  - Non-aggregate functions must be in Select list and in the Grouping Sets

Grouping sets sample output

WORKDEPT	EDLEVEL	SEX	SUM_SALARY	AVG_SALARY	COUNT
-	-	F	1201630.00	63243.68	19
-	-	M	1240895.00	53951.95	23
-	12	-	107140.00	35713.33	3
-	14	-	335320.00	47902.85	7
-	15	-	86560.00	43280.00	2
-	16	-	847855.00	60561.07	14
-	17	-	350880.00	50125.71	7
-	18	-	550020.00	78574.28	7
-	19	-	66500.00	66500.00	1
-	20	-	98250.00	98250.00	1
A00	-	-	354250.00	70850.00	5
B01	-	-	94250.00	94250.00	1
C01	-	-	308890.00	77222.50	4
D11	-	-	646620.00	58783.63	11
D21	-	-	358680.00	51240.00	7
E01	-	-	80175.00	80175.00	1
E11	-	-	317140.00	45305.71	7
E21	-	-	282520.00	47086.66	6

ROLLUP

- **ROLLUP is an extension to GROUP BY**
  - **Produces sub-total rows in addition to the "regular" grouped rows**
  - **Subtotal rows are "super-aggregate" rows**
    - **Further aggregates derived from same column functions**

Rollup sample

- Sample:

```
SELECT WORKDEPT, EDLEVEL, SEX,
   SUM(SALARY) as SUM_SALARY,
   AVG(SALARY) as AVG_SALARY, COUNT(*) as COUNT
  FROM DSN81110.EMP WHERE SALARY > 20000
  GROUP BY ROLLUP (WORKDEPT,EDLEVEL,SEX)
```

Rollup sample output

```
-- 55 rows in full answer set
-- Totals at each level, ending with grand total

WORKDEPT EDLEVEL SEX SUM_SALARY  AVG_SALARY  COUNT
-------- ------- --- ----------- ----------- -----------
-              - -    2442525.00    58155.35          42
A00            - -     354250.00    70850.00           5
B01            - -      94250.00    94250.00           1
C01            - -     308890.00    77222.50           4
...
A00           14 -      88500.00    44250.00           2
A00           18 -     199250.00    99625.00           2
A00           19 -      66500.00    66500.00           1
B01           18 -      94250.00    94250.00           1
C01           16 -      73800.00    73800.00           1
C01           18 -     136840.00    68420.00           2
C01           20 -      98250.00    98250.00           1
...
A00           14 M      88500.00    44250.00           2
A00           18 F     199250.00    99625.00           2
A00           19 M      66500.00    66500.00           1
...
```

Rollup

- **n elements of ROLLUP translate to *n*+1 grouping sets**
  - **Order of grouping-expressions is specified is significant for ROLLUP**

```
GROUP BY ROLLUP(a,b)
```

- **Is equivalent to:**

```
GROUP BY GROUPING SETS((a,b), (a), () )
```

```
GROUP BY ROLLUP(b,a)
```

- **Is the same as:**

```
GROUP BY GROUPING SETS((b,a), (b), () )
```

Rollup equivalency

- **Statement is equivalent to ROLLUP shown earlier:**

```
SELECT WORKDEPT, EDLEVEL, SEX,
    SUM(SALARY) as SUM_SALARY,
    AVG(SALARY) AS AVG_SALARY,
    count(*) as COUNT
  FROM DSN81110.EMP
  WHERE SALARY > 20000
  GROUP BY GROUPING SETS
    ((WORKDEPT, EDLEVEL, SEX),
     (WORKDEPT, EDLEVEL),
     (WORKDEPT),())
```

- **Grand total for result of generated by ()**

Cube

- Primarily for analytics
  - Contains all the rows of a ROLLUP and also cross-tabulation rows
    - Cross-tabulation rows are additional super-aggregate rows that are not part of an aggregation with sub-totals
- Like ROLLUP, CUBE grouping can also be thought of as a series of grouping-sets
  - For CUBE, all permutations of cubed grouping-expression-list are computed with grand total
  - n CUBE elements translate to 2**n grouping-sets

Cube

- A specification of:

```
GROUP BY CUBE(a,b,c)
```

- is equivalent to:

```
GROUP BY GROUPING SETS((a,b,c),
(a,b),
(a,c),
(b,c),
(a),
(b),
(c),
() )
```

Cube

- Three elements of CUBE translate to eight grouping sets
- The order of specification of elements does not matter for CUBE

```
SELECT WORKDEPT, EDLEVEL, SEX,
    SUM(SALARY) as SUM_SALARY,
    AVG(SALARY) AS AVG_SALARY,
    count(*) as COUNT
  FROM DSN81110.EMP WHERE SALARY > 20000
  GROUP BY CUBE (WORKDEPT, EDLEVEL, SEX);
```

Cube Result

- Total of 92 rows in result

WORKDEPT	EDLEVEL	SEX	SUM_SALARY	AVG_SALARY	COUNT
-	12	F	71800.00	35900.00	2
-	12	M	35340.00	35340.00	1
-	12	-	107140.00	35713.33	3
...					
-	-	F	1201630.00	63243.68	19
-	-	M	1240895.00	53951.95	23
-	-	-	2442525.00	58155.35	42
A00	14	-	88500.00	44250.00	2
A00	18	-	199250.00	99625.00	2
A00	19	-	66500.00	66500.00	1
A00	-	-	354250.00	70850.00	5
...					
A00	-	F	199250.00	99625.00	2
A00	-	M	155000.00	51666.66	3
...					
A00	18	F	199250.00	99625.00	2
A00	14	M	88500.00	44250.00	2
A00	19	M	66500.00	66500.00	1
...					

CUBE, ROLLUP and GROUPING SETS Summary

- Grand total
  - Both CUBE and ROLLUP return a row that is the overall (grand total) aggregation,
  - Can be separately specified with empty parentheses within GROUPING SET
  - Can also be specified directly in the GROUP BY clause
    - No effect on the result of the query

Array data type

- **An array type is a user-defined data type that is an ordinary array or an associative array.**
  - **Elements of an array type are based on one of existing built-in data types.**
- **Note: array data type can only be used as one of:**
  - **An SQL variable**
  - **A parameter or RETURNS data-type of an SQL scalar function**
  - **A parameter of a native SQL procedure**
  - **Target data type for a CAST specification**

Ordinary arrays

- **An array with:**
  - **User-defined upper bound on number of elements**
  - **Referenced by their ordinal position in the array**
- **In this example:**
  - **After the execution of the assignment statement, the cardinality of mySimpleA is set to 100**
  - **The elements of array indexes with values 1 to 99 are implicitly initialized to NULL**

```
CREATE TYPE simple AS INTEGER ARRAY[];
BEGIN
SET mySimpleA[100] = 123;
END
```

Associative arrays

- **An array with no user-defined upper bound on number of elements**
  - **Ordered by and referenced by an array index value**
  - **Array index values are unique and do not have to be contiguous**
- **In this example:**
  - **After the execution of the assignment statement, the cardinality of the array is set to 1**

```
CREATE TYPE assoc AS INTEGER ARRAY[INTEGER];
BEGIN
SET myAssocA[100] = 123;
END
```

Other cool stuff

- ALTER TABLE DROP COLUMN
  - Pending definition change
  - Entry is placed in SYSIBM.SYSPENDINGDDL for pending drop column, and TS placed in advisory REORG-pending (AREOR) state
  - Many restrictions, but they make sense

My Favorite Things in DB2 11 for z/OS

Martin Hubel

Martin Hubel Consulting Inc.

martin@mhubel.com

905-764-7498