Structure and Format Enhancements in DB2 9 for z/OS:
UTS & RRF

Willie Favero
Senior Certified IT Specialist
DB2 for z/OS Software Sales Specialist
IBM Sales and Distribution
West Region, Americas
713-9401132
wfavero@attglobal.net

Agenda

- Today's table spaces (review)
- Universal table spaces
  - ✓Partition by growth
  - ✓Partition by range
  - ✓Conclusion
- Automatic Creation of Database Objects
- Reordered Row Format

Today's Table Spaces
- A review -

In the Beginning…

- There were simple table spaces
  - ✓Multiple tables in the same table space
  - ✓Multiple tables could occupy the same page
  - ✓Simplistic space map
  - ✓64GB size limitation
  - ✓Depreciated in DB2 9 for z/OS
  - ✓DB2 Catalog still uses them
  - ✓Customer really shouldn't be creating simple table spaces anymore in any DB2 Version

And There Were Also…

- Partitioned table spaces
  - ✓Only one table allowed per table space
  - ✓Table space is divided into multiple partitions, data sets
  - ✓Requires a partitioning column
  - ✓SQL and utilities have partition independence
  - ✓Could be up to 128Tb
  - ✓Customer must pick number of partitions

Then DB2 V2.1 Added…

- Segmented table spaces
  - ✓Multiple tables per table space
  - ✓Pages are organized into segments
  - ✓Only one table per segment
  - ✓Still 64GB size limitation
  - ✓Better space maps, better DELETEs
  - ✓And better INSERT processing
  - ✓Customer choice, either segmented or partitioned
  - ✓Of the three types, best performance in most cases

Later, DB2 V6 Introduced…

- Large Object (LOB) table spaces

And Introduced with DB2 9…

- Universal Table Space (UTS)
- XML table spaces (discussion for another time)

Universal Table Space (UTS)

What Was Needed

- A table space needs both partitioned and segmented organization:
  - ✓ It need to be larger than 64GB
  - ✓ Inter-partition parallelism or independent processing is necessary
  - ✓ Partition scope operations (ADD,ROTATE) apply
  - ✓ Rows are variable in length and a fast insert is required
  - ✓ Mass delete operations should be fast

The Solution…

- DB2 9 universal table space
  - ✓The very best of segmented and partitioned table spaces delivered in one object

Today's Pain Point

- A table's growth is unpredictable (it could exceed 64GB) and there is no convenient key for range partitioning.
- Partitioning by a ROWID column introduces additional table space administration overhead:
  - ✓estimating optimal number of partitions
  - ✓ADDing partitions if necessary
  - ✓less then optimal space utilization

What is a Universal Table Space?

- All the best features from
  - ✓ Segmented table spaces and partitioned table spaces
    - Hybrid
  - ✓ Extra space maps and space map information
  - ✓ Multiple data sets (partitions)
  - ✓ Segmentation
- Plus a bunch of really cool new stuff
  - ✓ Better space management means less REORG
  - ✓ SQL TRUNCATE supported
  - ✓ ALTER TABLE ROTATE PARTITION supported
  - ✓ CLONE table supported
  - ✓ Improved insert performance

Things to Remember

- Only available AFTER you upgrade to DB2 9 new function mode (NFM)
- Only one table per table space allowed
- Reordered Row Format (RRF) only
  - Next section
- Partition independence
- No longer has a 64GB limitation
  - ✓ Depending on DSSIZE and the number of partitions, the table space could grow up to 128 TB
- Incompatible with MEMBER CLUSTER

Two Flavors are Available

- Universal table spaces are available in two flavors
  - ✓Partition by growth (PBG)
    - All the features of classic partitioning
    - Table controlled partitioning only
    - Using partition column
    - Partitioned and segmented
  - ✓Partition by range (PBR)
    - Partitions added as space is needed
    - No partitioning key
    - Partitioned and segmented

Common UTS Function limitations

- **Not** for the WORKFILE database.
- **No** MEMBER CLUSTER (not supported for segmented)
- **No** LOCKSIZE TABLE (uses partitioned table space locking scheme)
- **No** ALTER SEGSIZE/DSSIZE
  - ✓So get SEGSIZE/DSSIZE right or it is a DROP/CREATE
- **No** easy way to convert current type of table space to UTS
  - ✓Required to use DROP/CREATE

Partition by Growth (PBG)
Universal Table Space (UTS)

Partition by Growth Table Space

CREATE TABLESPACE …
MAXPARTITIONS integer

explicit specification

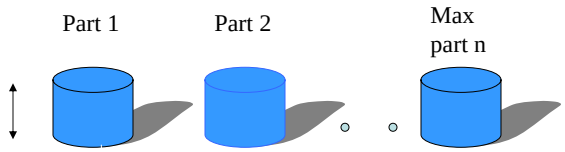
CREATE TABLE …
PARTITIONED BY SIZE EVERY integer G

implicit specification

- Associated SYSTABLESPACES columns
  - ✓ MAXPARTITIONS =max number of partitions
  - ✓ PARTITIONS =actual number of partitions
  - ✓ TYPE =G
- Only single-table table space
- Universal table space organization: although the table space is partitioned, the data within each partition is organized according to segmented architecture
- Incompatible with MEMBER CLUSTER, ADD PARTITION, ROTATE PARTITION

How Partition By Growth Works

- The table space starts with one partition, additional partitions will be added on demand until the maximum partition is reached.



Partitioned Table Space (parts added on demand)

Partition By Growth CREATE

- SQL CREATE TABLESPACE statement for PBG

```
CREATE TABLESPACE TS1 IN DB1
  MAXPARTITIONS  55
  SEGSIZE 64
  DSSIZE 2G
  LOCKSIZE ANY;
```

Makes PBG

Partition size

- ✓ A new key word MAXPARTITIONS - specifies the maximum # of partition for a table space.
- ✓ Maxpartitions can be changed by ALTER TABLESPACE .
  - Keep in mind that ALTER MAXPARTITIONS may require down time because it needs to physically close the datasets

Partition By Growth Create

- SQL CREATE TABLE statement for PBG

```
CREATE TABLE Mytable
    PARTITION BY SIZE EVERY integer G;
                                  where integer ≤ 64
```

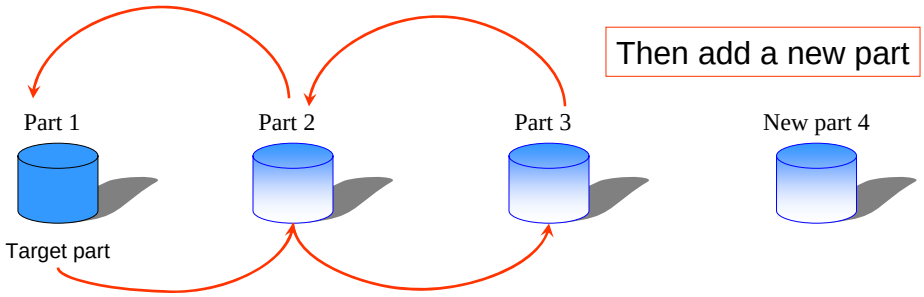
- Only available when you don't specify a table space name on the CREATE TABLE
- Table space is implicitly created
- mG specifies DSSIZE of the table space

More on Implicitly Created PBG

- Implicitly created table space defaults to PBG
- It defaults to row locking
- The LOCKMAX defaults to SYSTEM
- Default value for Maxpartitions = 256
- Default SEGSIZE = 4 if not specified on DDL
- Default DSSIZE = 4G if not specified on DDL
  - ✓ Note: DSSIZE and SEGSIZE require a DROP to change, no ALTER option

PBG Space Search

- No more space in the partition…
  - ✓ Search forward to next partition if there is one
  - ✓ Search backward to previous partitions



Note: If there is any restricted DBET state of any part during the backward space search. New part will not be added.

PBG UTS and Catalog Table

Catalog table SYSTABLESPACE

- TYPE column value
  - ✓ "**R**" - Range-partitioned universal table space
  - ✓ "**P**" - Implicit table space created for XML columns.
  - ✓ "**O**" - Table space is a LOB table space
  - ✓ "**G**" - Partitioned-by-growth table space
- MAXPARTITIONS (new column)
  - ✓ Maximum number of partitions
- PARTITIONS
  - ✓ Column contains the number of physical partition (dataset) that currently exist

PBG UTS and Catalog Table

Catalog table SYSTABLEPART

- When table space created, one partition created and one row inserted to SYSTABLEPART (assuming created with DEFINE YES)
- Additional row added for each new partition required

Additional Characteristics of PBG

- It is partitioned according to space requirements
  - ✓ A partition is allocated when one is needed due to growth
- Each partition has one-to-one correspondence to a VSAM data set and must be DB2-managed
- No partitioning key to bound the data within a table space, so no PI index
- Only NPI indexes can be created

Additional Characteristics of PBG

- When partition fills and MAXPARTITIONS not reached, new partition created and catalog is updated
  - ✓ Even if unit of work adding a partition issues a rollback, new partition remains
- Compression dictionary will be copied from previous partition to the new partition
- Freespace, caching, define, logging and trackmod attributes are same for each partition

Additional Characteristics of PBG

- Drains and Claims of new partition are inherited from prior partition
- Some DBET states are also inherited by the new partition from the previous partition
  - ✓ (RO*, UTUT, UTRO for table space, PSRBD, ICOPY for NPI)
- CLONE table can be created
  - ✓ Both CLONE and base table grow at the same time
- All utilities can operate at the partition level except LOAD utility

Partition by Growth Limitations

- Single table only, can not totally replace segmented table space
- Only NPI allowed

PBG – Additional Function Limitations

- No partition key range can be defined
- No ALTER ADD PART
- No ALTER ROTATE PART
- No ALTER Stogroup
- No LOAD PART
- No user-directed define partition
  - ✓ Required to use UNLOAD/LOAD instead of DSN1COPY for copying data between table space if source table space has more than 1 partition

Practical applications for PBG

- When no obvious partitioning column exists
- When a table requiring > 64G
  - ✓Lift 64G size limitation of segmented table space
  - ✓Increase overall size of table space on demand
- Space on Demand
- Large table space and manage utilities at a data subset is needed
  - ✓Partition level utility
- There's a need for CLONE table

Partition By Growth and REORG

- Reorganize the data, could result in more or less partitions needed to hold data
- If n # of parts to start with will be n or more # of parts at the end of the REORG
- No delete of existing partitions
- If MAXPARTITIONS reached, REORG fails
- If new partition is added, dictionary pages are copied from the previous partition into the new partition

PBG, REORG and LOB

- **Table has LOB column**,
  - ✓ The data in base table will not move between parts even it is table space level operation.
  - ✓ Data must fit back to original part. If it is not due to the partition space attributes (PCTFREE or PGFREE), REORG adjusts the space attributes in order to fit original data back
  - ✓ Data within each partition is organized

PBG, REORG and no LOB

- **Table does not have LOB column**,
  - ✓ Holes within each partition will be eliminated
  - ✓ If REORG at the table space level, could result in empty partitions at the end of table space
  - ✓ If REORG at the part range level, the data on one part can overflow to the other
  - ✓ If REORG at single part level, data must be able to fit back into the partition
  - ✓ REORG SHRLEVEL CHANGE – new partition is added to both shadow and base(I and J data set)

PBG and REORG limitations

- **No** parallelism to ensure data reduced to minimum number of partitions
- **No** REBALANCE
- **No** shrinking of partitions even if there are only empty partitions at the end of table space
  - ✓ The empty partitions could have header, space map page, dictionary page and system pages

Partition By Growth and COPY

- Copies can be made at the part level or the table space level
- Will copy empty partition
- Will also pick up new partition added during COPY for COPY SHRLEVEL CHANGE at the table space level
  - ✓ Remember this is a fuzzy copy. It is not recommend to be used for RECOVER TOCOPY

Partition By Growth and RECOVER

- RECOVER to currency with image copy
  - ✓ Pick up new added parts since the copy via log apply.
- RECOVER to image copy, PIT or NOT LOGGED TB
  - ✓ The excess partitions (in base, LOB or XML) will be empty (header/space map/system pages).

Partition By Growth and RECOVER

- For PIT recovery
  - ✓ It is suggested to RECOVER base, LOB and XML table space in the same utility statement
    - – Example of RECOVRE Statement

```
LISTDEF MYSET
  INCLUDE TABLESPACE DB.PBGTS
  INCLUDE TABLESPACE DB.PBGTS  LOB
  INCLUDE TABLESPACE DB.PBGTS  XML

RECOVER LIST  MYSET TOLOGPOINT X‘lrsn-value‘
```

  - ✓ AUX Check Pending (ACHKP) is set on base table space if RECOVER base, LOB and XML table space are not in the same utility statement

Partition By Growth and LOAD

- Only support table space level operation
- No parallelism for Load Utility
- Can accommodate growth of table space
- Copy dictionary from previous partition to the new partition
- Excess partitions remain empty

Partition By Growth and Other Utilities

- **CHECK INDEX SHRLEVEL CHANGE**
  - ✓ Partition added during the course of the CHECK INDEX utility is not checked
- **REBUILD INDEX SHRLEVEL CHANGE**
  - ✓ Index for record inserted into new added partition during the course of the REBUILD is reflected in the index page set via log apply

Partition By Growth and DSN1COPY

- Partition # may be inconsistent between DSN1COPY and the target table space
  - ✓ If part # of target TS > part # of source TS :
    - Use TRUNCATE TABLE on the target table before DSN1COPY to make sure the target table is empty
  - ✓ If part # of target TS < part # of source TS:
    - DSN1COPY can not be used
    - Unload/Load may be used

Partition by Range (PBR)
Universal Table Space (UTS)

Create Partition By Range UTS

- **SQL CREATE statement**

```
CREATE TABLESPACE PRB_TS1 IN UTS_DB1
    NUMPARTS 3
    SEGSIZE 64
    LOCKSIZE ANY;
```

Makes PBR

- Create a partitioned table space and just add the SEGSIZE clause = PBR

Create Table in PBR UTS

```
CREATE TABLE MyTable
        ( C1  CHAR(4),
          C2   VARCHAR(20),
          C3   INTEGER )
    PARTITION BY (C1)
        ( PARTITION 1 ENDING AT ('DDDD'),
          PARTITION 2 ENDING AT ('HHHH'),
          PARTITION 3 ENDING AT ('ZZZZ') )
                    IN UTS_DB1.PRB_TS1 ;
```

- Must use table-controlled partitioning

PBR UTS and Catalog Table

Catalog table SYSTABLESPACE

- TYPE column value
  - ✓"**R**" - Range-partitioned universal table space
  - ✓"**P**" - Implicit table space created for XML columns.
  - ✓"**O**" - Table space is a LOB table space
  - ✓"**G**" - Partitioned-by-growth table space

PBR's Aux Table Space

- LOB table space
  - ✓Can be user defined via SQLRULES
- XML AUX table space
  - ✓Its table space type is also PBR UTS
  - ✓XML rows are in the same part as base row

PBR – Additional Function Limitations

- **No** index-controlled partitioning definition

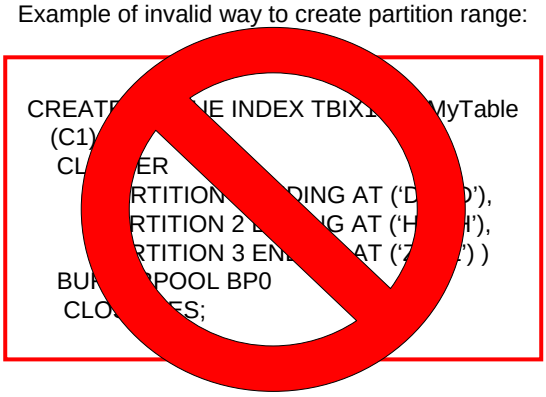
Example of invalid way to create partition range:

```
CREATE UNIQUE INDEX TBIX1 ON MyTable
  (C1)
   CLUSTER
      (PARTITION 1 ENDING AT ('DDDD'),
       PARTITION 2 ENDING AT ('HHHH'),
       PARTITION 3 ENDING AT ('ZZZZ') )
   BUFFERPOOL BP0
    CLOSE YES;
```


PBR – Additional Function Limitations

- **No** index-controlled partitioning definition

Example of invalid way to create partition range:



SQLCODE = -662

A PARTITIONED INDEX CANNOT BE CREATED ON A NON-PARTITIONED, PARTITION-BY-GROWTH, OR RANGE-PARTITIONED UNIVERSAL TABLE SPACE

Practical applications for PBR

- When a partitioned table space and a partitioning key is required
- When better performance than classic partitioned table space is required
- partition-independence and parallelism capabilities
- When a CLONE table is required

Conclusion

UTS are Very Cool EXCEPT…

- Only one table per table space
- No member cluster

- No migration
  - ✓Must **DROP** and re-CREATE

Automatic Creation of *Database Objects*

Problem

- Database objects that have different meanings for different database platforms must be transparent to database-platform agnostic applications, e.g. database and table space
- Some of the DB2 database objects must be manually created although all their attributes are implicitly known to DB2, e.g. indexes that enforce primary and unique keys

Solution

- Unless explicitly specified, DB2 will implicitly create the following database objects:
  - ✓ Database
  - ✓ Table space
  - ✓ Index that enforces primary key
  - ✓ Index that enforces unique key
  - ✓ ROWID index
    - For ROWID columns defined as GENERATED BY DEFAULT)
  - ✓ LOB table space
  - ✓ LOB auxiliary table
  - ✓ LOB auxiliary index

Automatic Creation of *Database*

- Automatically (implicitly) created databases are used when IN clause is omitted at CREATE TABLE. There are two possible outcomes:
  1. A new database is created
     - Namespace DSN00001 – DSN60000
     - Schema: SYSIBM
  2. An existing implicitly created database is used
     - Start reusing existing databases after reaching 60000
     - Wrap-around fashion
     - Therefore, implicitly created databases can contain multiple table spaces
- Explicitly created tables and table spaces in implicitly created databases are not allowed
- Implicitly created databases can be explicitly dropped

Automatic Creation of *Table Space*

If CREATE TABLE does not include explicit table space specification, a table space is automatically (implicitly) created with the following attributes:

- Partition-by-growth universal table space
  - ✓ SEGSIZE=4
  - ✓ DSSIZE=4G
  - ✓ MAXPARTITIONS=256
- Compression is controlled by DSNZPARM IMPTSCMP
  - ✓ Default is COMPRESS=NO
- Dataset creation is controlled by DSNZPARM IMPDSDEF
  - ✓ Default is DEFINE=YES
- Sliding scale for optimizing secondary extent allocation is used by default (MGEXTSZ=YES)
- The default buffer pool is determined based on the DSNZPARM settings.
  - ✓ Buffer pool page size is derived automatically using larger page size if max record size reaches 90%

Reordered Row Format (RRF)

Reordered Row Format (RRF)

- Automatic in DB2 9 new function mode (NFM) when table space is created
  - ✓ Except if EDITPROC or VALIDPROC are used
  - ✓ DB2 catalog and directory will not be converted to the reordered row format
- Prior to DB2 9
  - ✓ Column placement when using varying length columns could be problematic
    - Where should varying length columns be placed
    - Scan for next varying length columns after VARCHAR
    - If all VARCHAR, offset calculation required
  - ✓ Table design often out of your control

Reordered Row Format

- New format order
  - ✓Fix length column first
  - ✓Followed by offsets to varying length columns
  - ✓Followed by varying length columns
- No change to any SQL reference row in reordered row format

Reordered Row Format

- Catalog table SYSTABLEPART
  - ✓FORMAT
    - Blank – basic row format – a pre-V9 or LOB table space
    - "R" – Reordered row format
- No VARCHAR – No impact
  - ✓Including performance
- Logging in most cases should not change

Reordered Row Format

- REORG or LOAD REPLACE will migrate pre-V9 table spaces to reordered row format
  - ✓Careful if using compression
    - Dictionaries automatically converted
    - Even if KEEPDICTIONARY is specified (APAR PK41156)
    - New REORG keyword available - HONOR_KEEPDICTIONARY
- Careful of DSN1COPY

The Setup… CREATE / INSERT / DSN1PRNT

CREATE TABLE RRF
(C1 CHAR(10), C2 VARCHAR(20), C3 VARCHAR(15), C4 CHAR(10));

INSERT INTO RRF
VALUES('A11111111A','A2222222222222A','A33333333A','A44444444A');

DSN1PRNT

```
PAGE: # 00000002 ------------------------------------------------------------------------------
      -
DATA PAGE: PGCOMB='10'X PGLOGRBA='00002C3C3D8A'X PGNUM='00000002'X PGFLAGS='00'X PGFREE=3952
           PGFREE='0F70'X PGFREEP=207 PGFREEP='00CF'X PGHOLE1='0014'X PGMAXID='02'X PGNANCH=1
           PGTAIL: PGIDFREE='00'X PGEND='N'

ID-MAP FOLLOWS:
01 009E 0059

RECORD: XOFFSET='00CF'X PGSFLAGS='02'X PGSLTH=59 PGSLTH='003B'X PGSOBD='0000'X PGSBID='01'X

00C1F1F1 F1F1F1F1 F1F1C100 C1F4F4F4 F4F4F4F4 F4C1001A 002A00C1 F2F2F2F2 .A11111111A.A44444444A.....A2222
F2F2F2F2 F2F2F2F2 F2C100C1 F3F3F3F3 F3F3F3F3 C1                         222222222A.A33333333A


RECORD: XOFFSET='010A'X PGSFLAGS='02'X PGSLTH=49 PGSLTH='0031'X PGSOBD='0000'X PGSBID='02'X

00C2F1F1 F1F1F1F1 F1F1C2FF 00000000 00000000 0000001A 001BFF00 C2F3F3F3 .B11111111B.................B333
F3F3F3F3 F3F3F3F3 F3F3C2                                                 3333333333B

DSN1994I DSN1PRNT COMPLETED SUCCESSFULLY, 00000003 PAGES PROCESSED
```

Next page

A Picture's Worth a 1,000 Words

```
0
00C1F1F1  F1F1F1F1  F1F1C100
C1F4F4F4  F4F4F4F4  F4C1001A
002A00C1  F2F2F2F2  F2F2F2F2
26                  42
F2F2F2F2  F2C100C1  F3F3F3F3
F3F3F3F3  C1
```

```
INSERT INTO RRF
   VALUES  ( 'A11111111A',  <-- CHAR(10),
             'A2222222222222A',  <-- VARCHAR(20),
             'A33333333A',  <-- VARCHAR(15),
             'A44444444A');  <-- CHAR(10));
```

Nulls are allowed of all columns

A Picture's Worth a 1,000 Words

```
0
00C1F1F1  F1F1F1F1  F1F1C100
C1F4F4F4  F4F4F4F4  F4C1001A
002A00C1  F2F2F2F2  F2F2F2F2
26                  42
F2F2F2F2  F2C100C1  F3F3F3F3
F3F3F3F3  C1
```

```
INSERT INTO RRF
   VALUES  ( 'A11111111A',  <-- CHAR(10),
             'A2222222222222A',  <-- VARCHAR(20),
             'A33333333A',  <-- VARCHAR(15),
             'A44444444A');  <-- CHAR(10));
```

Nulls are allowed of all columns

A Picture's Worth a 1,000 Words

```
0
00C1F1F1  F1F1F1F1  F1F1C100
C1F4F4F4  F4F4F4F4  F4C1001A
26
002A00C1  F2F2F2F2  F2F2F2F2
42
F2F2F2F2  F2C100C1  F3F3F3F3
F3F3F3F3  C1
```

```
INSERT INTO RRF
   VALUES  ( 'A11111111A',  <-- CHAR(10),
             'A2222222222222A',  <-- VARCHAR(20),
             'A33333333A',  <-- VARCHAR(15),
             'A44444444A');  <-- CHAR(10));
```

Nulls are allowed of all columns

A Picture's Worth a 1,000 Words

```
0
00C1F1F1  F1F1F1F1  F1F1C100
C1F4F4F4  F4F4F4F4  F4C1001A
26
002A00C1  F2F2F2F2  F2F2F2F2
42
F2F2F2F2  F2C100C1  F3F3F3F3
F3F3F3F3  C1
```

```
INSERT INTO RRF
   VALUES  ( 'A11111111A',  <-- CHAR(10),
             'A2222222222222A',  <-- VARCHAR(20),
             'A33333333A',  <-- VARCHAR(15),
             'A44444444A');  <-- CHAR(10));
```

Nulls are allowed of all columns

A Picture's Worth a 1,000 Words

```
0
00C1F1F1  F1F1F1F1  F1F1C100
C1F4F4F4  F4F4F4F4  F4C1001A
002A00C1  F2F2F2F2  F2F2F2F2
26                  42
F2F2F2F2  F2C100C1  F3F3F3F3
F3F3F3F3  C1
```

```
INSERT INTO RRF
   VALUES  ( 'A11111111A',  <-- CHAR(10),
             'A2222222222222A',  <-- VARCHAR(20),
             'A33333333A',  <-- VARCHAR(15),
             'A44444444A');  <-- CHAR(10));
```

Nulls are allowed of all columns

A Picture's Worth a 1,000 Words

```
0
00C1F1F1  F1F1F1F1  F1F1C100
C1F4F4F4  F4F4F4F4  F4C1001A
002A00C1  F2F2F2F2  F2F2F2F2
26                  42
F2F2F2F2  F2C100C1  F3F3F3F3
F3F3F3F3  C1
```

```
INSERT INTO RRF
   VALUES  ( 'A11111111A',  <-- CHAR(10),
             'A2222222222222A',  <-- VARCHAR(20),
             'A33333333A',  <-- VARCHAR(15),
             'A44444444A');  <-- CHAR(10));
```

Nulls are allowed of all columns

Questions

Shameless Self promotion

My DB2 for z/OS blog…

http://blogs.ittoolbox.com/database/db2zos

References

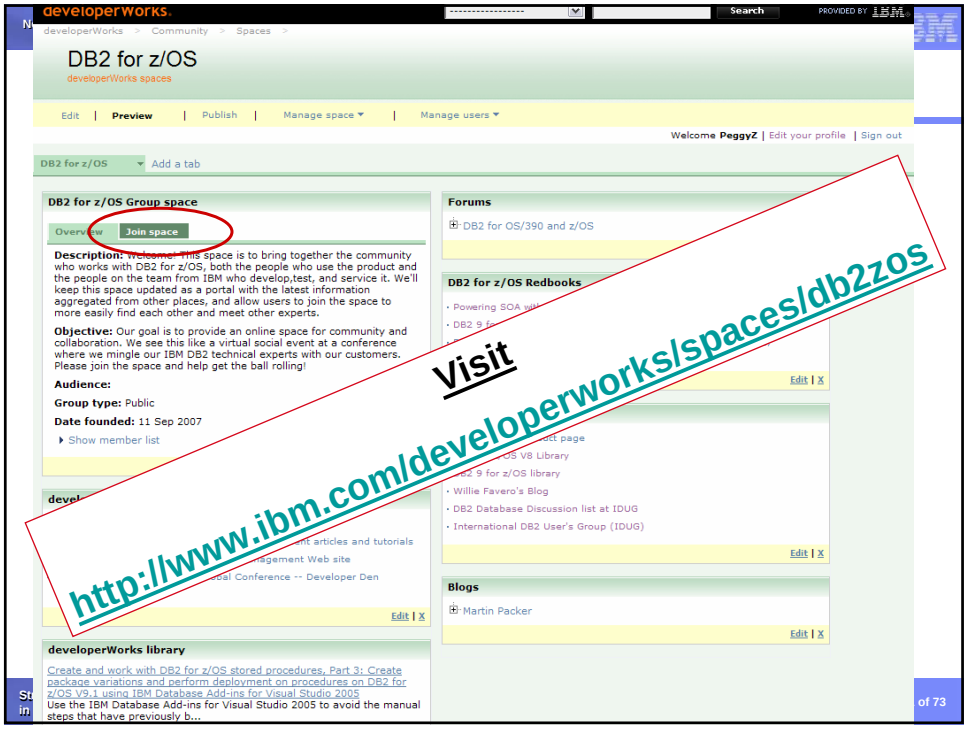
Presentations

"DB2 V9 Universal Table Spaces for z/OS",
Frances Villafuerte,
IBM DB2 Development,
IDUG NA 2007

Redbooks

SG24-7330 - DB2 9 for z/OS Technical Overview
SG24-7473 - DB2 9 for z/OS Performance Topics

http://www.ibm.com/redbooks


developerWorks
Search
DB2 for z/OS
developerWorks spaces
Edit | Preview | Publish | Manage space | Manage users
Welcome PeggyZ | Edit your profile | Sign out
DB2 for z/OS
Add a tab
DB2 for z/OS Group space
Overview
Join space
Description: Welcome! This space is to bring together the community who works with DB2 for z/OS, both the people who use the product and the people on the team from IBM who develop,test, and service it. We'll keep this space updated as a portal with the latest information aggregated from other places, and allow users to join the space to more easily find each other and meet other experts.
Objective: Our goal is to provide an online space for community and collaboration. We see this like a virtual social event at a conference where we mingle our IBM DB2 technical experts with our customers. Please join the space and help get the ball rolling!
Audience:
Group type: Public
Date founded: 11 Sep 2007
Show member list
Forums
DB2 for OS/390 and z/OS
DB2 for z/OS Redbooks
Willie Favero's Blog
DB2 Database Discussion list at IDUG
International DB2 User's Group (IDUG)
Blogs
Martin Packer
developerWorks library
Create and work with DB2 for z/OS stored procedures, Part 3: Create package variations and perform deployment on procedures on DB2 for z/OS V9.1 using IBM Database Add-ins for Visual Studio 2005
Use the IBM Database Add-ins for Visual Studio 2005 to avoid the manual steps that have previously b...
Visit
http://www.ibm.com/developerworks/spaces/db2zos
of 73


November 28, 2007
Thank You
Structure and Format Enhancements in DB2 9 for z/OS
Copyright © 2007 IBM Corporation All rights reserved
Slide 72 of 73

William (Willie) Favero

Senior Certified Consulting IT Software Specialist

DB2 for z/OS Specialty SSR

IBM Academic Initiative Ambassador for System z

IBM Certified Database Administrator - DB2 Universal Database V8.1 for z/OS

IBM Certified Database Administrator – DB2 9 for z/OS

IBM zChampion

wfavero@attglobal.net