SHARE in Orlando 2015

17702 - DB2 V11 Migration Experiences

Hal Steiner
Bank of America / Merrill Lynch

Monday, August 10, 2015: 11:15 AM-12:15 PM
Southern Hemisphere 4
Walt Disney World Dolphin

#SHAREorg

SHARE is an independent volunteer-run information technology association that provides **education**, professional **networking** and industry **influence.**

Presentation Outline

- In this presentation, Hal Steiner will discuss the DB2 V10 to V11 Migration from a customer's perspective, especially issues of concern to application DBAs. Hal is one of the key players working on this transition for Bank of America.

- Hal will focus during the first part of the presentation on the pre-migration remediation efforts which are necessary from the application perspective to get ready for V11. Hal is especially passionate about converting simple tablespaces and will propose a methodology for migrating to PBG. In the second part of the presentation, Hal will highlight the DB2 V11 new features his team is most anxious to adopt.

- The session provides several opportunities for Q & A plus general comments from the audience.

Big Picture DB2 V11 Migration Comments

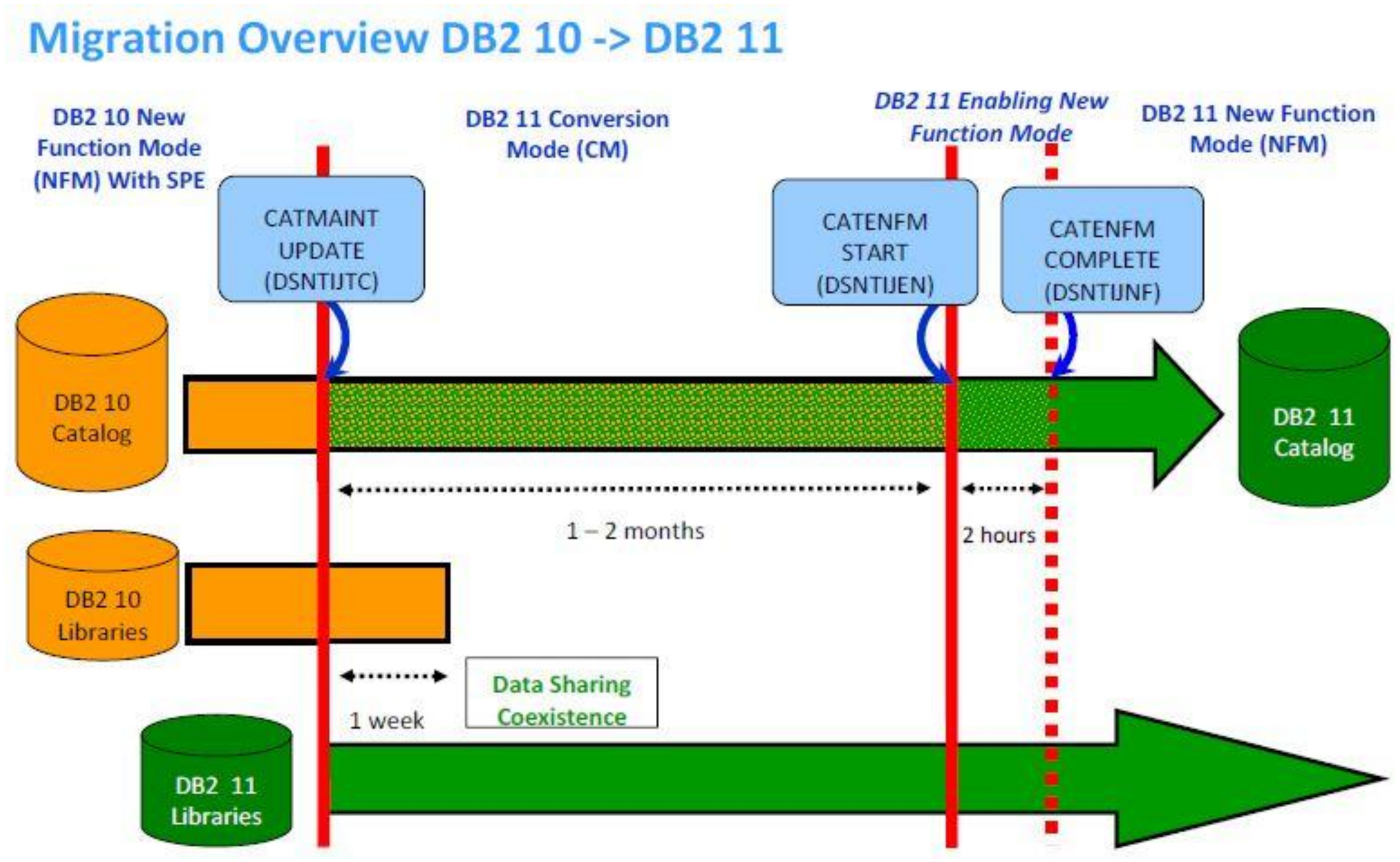
- So far, it's not painful migrating from a DB2 V10 NFM baseline. It's evolutionary, not revolutionary. And the word on the street is very positive regarding out of the box CPU performance improvements, especially.

- Most of the work is for systems DBAs not application DBAs, with a few notable exceptions.

- Not only can you as usual delineate between Conversion Mode (CM) and New Function Mode (NFM) but with the new APPLCOMPAT ZPARM (overrideable at BIND time at the package level) you can also determine when new/modified V11 SQL Features will be enabled at a later date. This effectively separates system migration from application migration.

- You can also defer Extended LRSN implementation to post NFM (note REORGs are necessary to migrate objects to Extended LRSN). This involves converting the BSDS as well as REORGing the DB2 Catalog and Directory tables to support 10 byte RBA format.

DB2 V11 Migration Overview


Migration Overview DB2 10 -> DB2 11
DB2 10 New Function Mode (NFM) With SPE
DB2 11 Conversion Mode (CM)
DB2 11 Enabling New Function Mode
DB2 11 New Function Mode (NFM)
CATMAINT UPDATE (DSNTIJTC)
CATENFM START (DSNTIJEN)
CATENFM COMPLETE (DSNTIJNF)
DB2 10 Catalog
DB2 11 Catalog
1 – 2 months
2 hours
DB2 10 Libraries
1 week
Data Sharing Coexistence
DB2 11 Libraries

Complete your session evaluations online at www.SHARE.org/Orlando-Eval

Performance Benefits in V11 Without REBIND

- DDF and Data Sharing performance improvements
- zIIP enablement for most all SRB-mode DB2 system agents
- Reduced cross-memory overhead for writing to DB2 Log
- Improvements in Data COMPRESSion
- Multiple Improvements regarding INSERTs
- Internal DB2 Procs (xProcs) moved above the bar.
- Automatic index pseudo delete cleanup
- ODBC/JDBC type 2 driver performance improvements
- Sort performance improvements
- DPSI and DGTT performance improvements
- Performance improvements with large # of partitions
- XML improvements
- Optimized RELEASE(DEALLOCATE) now preferred over RELEASE(COMMIT)
- Various Utility improvements, especially REORG.

Performance Benefits in V11 With REBIND

- Most improved "in-memory" techniques (i.e. less workfile I/0)
- Improvements to complex SQL, including non-correlated subqueries with mismatched lengths.
- Column processing improvements (i.e. generated machine code for output column processing written in memory above the bar).
- RID overflow improvements
- DECFLOAT improvements (inc. 50%+ faster INSERT & FETCH)
- Query transformation improvements (for example will convert YEAR(colx) = ? To a colx BETWEEN 'start' AND 'end')
- Improved algorithm for duplicate removal (e.g. SELECT DISTINCT or GROUP BY)
- Improvements to DPSI and page range screening
- Optimizer performance improvements esp. regarding CPU and I/O cost balancing

To REBIND or NOT TO REBIND?

- You *MUST* rebind pre-migration anything bound prior to DB2 V9.1 Previous or Original PLANMGMT copies prior to V9 cannot be used.

- You don't have to rebind anything else, even once you're in DB2 V11 CM, but you might be missing out on CPU improvements; however, if you rebind access paths there is always the risk of a few outliers which actually run slower than before, sometimes dramatically slower.

- RECOMMENDATION: Rebind first in lower lanes and use a tool to evaluate altered access paths which might be suboptimal.

- Remember going forward in V11 you can specify APPLCOMPAT(V10R1 or V11R1) at BIND time.

Other Pre-Migration Application Activities

- Run Job DSNTIJPB which produces a series of over two dozen reports of remediation activities. Most of these relate to the DB2 System Catalog or Directory and must be run by System DBAs. Here are the main issues affecting application DBAs:

- REPORT #4 – Plans bound prior to DB2 V9.1 – to avoid auto rebind being triggered these plans should be rebound prior to migration to CM.

- REPORT #5 – Packages bound prior to DB2 V9.1 that are also autobind candidates in V11

- REPORT #6 – Explain tables (incl. DSN_*) not in current release (V10) acceptable format.

- REPORT #11 – Simple Tablespaces (see following pages) -

- REPORT #12 – Trigger Packages with negative SECTNOI value must be dropped and re-created.
- REPORT #13 – Views that have Period specifications will need to be dropped
- REPORT #14 – MQTS that have Period specification will need to be dropped
- REPORT #15 – SQL functions that have Period specifications will need to be dropped

Simple Tablespaces: The facts

1. DB2 Simple Tablespaces were introduced in DB2 V1.1 as the original default tablespace structure. Initially the only alternative was Classic Index Based Partitioned. Over the years, better alternatives have emerged. For example, in V2.1 Segmented tablespaces became a third option. In DB2 V9, the introduction of Universal Tablespaces added 2 more options (Partition by Growth and Partition by Range), hereafter PBG and PBR.

2. Simple Tablespaces were deprecated by IBM in DB2 V9. They were replaced in the DB2 Catalog in DB2 V10. And in DB2 V11, they can no longer be created, although their existence is tolerated. So if you deliberately (or accidentally) DROP a simple tablespace in V11 you won't be able to re-create it as a simple tablespace. IBM recommends you migrate all simple tablespace pre-V11 to a "preferred" table type.

3. Consider the problem if we transition to V11 without migrating simple tablespaces – what if any of these tables need to be dropped and re-created? At that point the DBA could be faced with a production recovery situation in which they would have to convert to another structure without the benefit of careful planning and testing. Packages would be invalidated and need to be rebound as part of the recovery.

What are the alternatives?

What Else is Wrong with Simple Tablespaces?

- Less granular – more primitive space map – made poor insert/update/delete decisions
- Tablespace scans must touch every physical page even if unused
- No “Mass Delete” functionality due to original space map design
- Can’t exceed 64 GB table size limit.
- New cool UTS features not available (see list on next page)

What About Segmented Tablespaces?

- More granular space map -- accurate insert/update/delete decisions.
- Tablespace scans only touch every occupied physical page
- Rapid “Mass Delete” functionality due to improved space map design
- Also can’t exceed 64 GB table size limit.
- Likely to be deprecated and have supported dropped in next release or two.
- New cool UTS features not available (see list on next page)

One big advantage of UTS

What new DB2 features require Universal Tablespaces?

- Clone tables (DB2 9)
- Hash-organized tables (DB2 10)
- "Currently committed" locking behavior (DB2 10 -- a means of reducing lock contention)
- Pending DDL (DB2 10)
- LOB inlining (DB2 10)
- XML multi-versioning (DB2 10 -- required for a number of XML-related enhancements)
- ALTER TABLE with DROP COLUMN (DB2 11)

UTS- Partitioned by Growth (PBG)

- This is now the default tablespace type in DB2 V10 – it is the combination of best features of both partitioned and segmented (hybrid).
- Extra enhanced space maps, super improved insert /delete logic .
- Can automatically extend beyond 64 GB up to 128TB based on DSSIZE and MAXPARTITIONS. Logical partitions are added sequentially as needed up to the limit of MAXPARTITIONS.
- Allocation is determined by PRIQTY, SECQTY and usage
- Increasing MAXPARTITIONS is possible via ALTER TABLESPACE; however, that process will need to close all VSAM datasets to successfully synchronize, so it might require downtime.
- There is an agent memory penalty paid for over-specifying MAXPARTITIONS.
- Only one table per tablespace allowed (not much of a prohibition anymore)
- With PBG, no partitioning key will be defined
- With PBG, you cannot create DPSIs. Only Non-Partitioned Indexes

UTS- Partitioned by Range (PBR)

- This is the second flavor of Universal Table Space. Both PBG and PBR are Universal Table Spaces and share many of the same advantages. The big difference is the definition of the partitioning key (in the CREATE TABLE) and having a definite size in terms of number of partitions (NUMPARTS) -- not a gradually expanding one.

- The PBR Tablespace must use table-controlled partitioning.

- Some applications will run better due to queries minimizing which partitions need to be accessed (e.g. partition independence by date range).

- Best candidates for PBR will be large tables with high insert, update, delete volumes.

Also Consider …

- Perhaps some of these tablespaces haven't ever been modernized until now because no one is using them. Check LASTUSED reference dates for situations were perhaps tables can be dropped. This is especially likely in development environments.

- Some simple tablespaces are just very old and have other glaring inefficiencies due to being antiquated, such as COMPRESS NO , or TRACKMOD NO, or they aren't taking advantage of MEMBER CLUSTER. Now might be a great opportunity to add them in.

- Some migration paths require a DROP and CREATE, while others can be done via a non-disruptive ALTER followed by a REORG. The ones that are ALTER-able (assuming only 1 table per tablespace) are:

    - SIMPLE TO PBG
    - SEGMENTED TO PBG
    - PARTITIONED TO PBR

How to convert Simple to PBG, part 1

```
ALTER TABLESPACE DBxxx001.TSxxxyyy
    MAXPARTITIONS 1
    MEMBER CLUSTER YES;
```

NOTE: While MAXPARTITIONS 1 will meet most people's needs, it can be coded as high as 4096, when needed and/or desirable. You can start it at 1, or a small positive integer, and always increase it later via ALTER as you approach running out of space.

```
ALTER TABLESPACE DBxxx001.TSxxxyyy
    SEGSIZE 64;          -- otherwise it can default to a low value - as always          --
    use your best judgment
```

NOTE: Both of these are deferred changes and have to be issued separately. The tablespace will be placed in Advisory REORG Pending (SQLCODE +610).

How to convert Simple to PBG, part 2

ALTER TABLESPACE DBxxx001.TSxxxyyy
DSSIZE 64G;

NOTE: This requires a STOGROUP with Extended Addressability DASD STOGROUPs

REORG TABLESPACE

RUNSTATS (there are extra stats for UTS) Be careful with -1 converting to 0 rows – consider STATISTICS UPDATE NONE REPORT YES

REBIND PACKAGES (all packages that reference altered tablespace will be invalidated)

HINT: Consider using an automated tool such as BMC Change Manager.

New Features in DB2 V11 to leverage

RELEASE DEALLOCATE improvement Persistent Thread Break-In

Extended LRSN (anticipated 6% CPU gain)

Cleanup of Pseudo-Empty Index Pages

Native SQL Procedure improvements, including Autonomous Procedures

XML enhancements

REORG Improvements

PCTFREE FOR UPDATE enhancement

Predicate Selectivity Overrides

RELEASE(DEALLOCATE) Improvement

- Version 11 makes improvements to the RELEASE(DEALLOCATE) bind option, making it the best choice in more situations.

- A serious past limitation of this choice was that it wasn't possible for DBA jobs such as DDL or binds to "break-in" to long running applications. This is sometimes referred to as the persistent thread problem.

- In V11 there are two improvements: break-in for active threads, and break-in for idle threads.

- A new zparm, PKGREL_COMMIT can allow a persistent DB2 thread to release a package that is active on that thread *IF* certain DB2 operations are waiting for exclusive access to the package (such as Bind or online schema changes).

- Break-in for idle threads – if a package lock is likely to time out, DB2 recycles any idle threads that use local attach facilities (e.g. CICS). When a thread is eligible for recycling, use of the thread is temporarily delayed until the recycle process is complete, allowing the DB2 operation to "break-in".

EXTENDED LRSN

- When DB2 was first introduced, it internally used 6 byte relative byte addresses (RBA) and Log Record Sequence number (LRSN). These values reached a maximum at 2 to the 48th power, which is roughly 140 trillion. I'm sure at the time all the best brains in the world thought we'd never reach 140 trillion bytes of DB2 log information on one subsystem. But they were wrong.

- If a DB2 subsystem reaches the soft then the hard logging limits, it will need to be recycled in some manner.

- Version 11 optionally allows you to extend these values to a 10 byte format, which is 2 to the 80th power. There is a name for this – a yottabyte. It's a trillion terabytes.

- In addition to avoiding the need to recycle a DB2 subsystem for at least the foreseeable future, it offers several performance enhancements which net out to an approximately 6% gain, including LRSN spin reduction

- Some DB2 objects will be put into this mode automatically (e.g. workfiles), when it's time to convert everything the systems team runs several jobs and the net effect is that new table spaces and indexes are created in 10byte extended format for RBA and LRSN, and existing ones are modified on next REORG/REBUILD.

CLEANUP OF PSEUDO-EMPTY INDEX PAGES

- In V11, DB2 does more than ever to automatically clean up both pseudo-deleted index entries and pseudo-empty index pages. This improves performance by making the index more compact while reducing the need to run REORG frequently.

- Only in some cases are index entries physically removed at DELETE time. Often, they are instead marked with a flag in a Delete Byte in the index entry itself for subsequent removal. Of course those index entries remain physically in the index and might be searched/accessed although skipped over due to the delete flag. As more and more of these occur, performance degrades.

- In V10, DB2 attempted to clean up pseudo-empty index pages as part of SQL DELETE processing only – this involves index pages which are entirely empty *AND* in which all the deletes entries on that page had already been COMMITTed. This would miss many situations, leaving the page in the index effectively empty until the next REORG.

- You enable this feature through a zparm INDEX_CLEANUP_THREADS when non-zero. You can exempt certain indexes and you can specify certain blackout time periods when it won't run. The process is ZIIP-enabled and will run with minimal disruption. For example, index cleanup will never cause you to deadlock – the index cleanup task would always be the loser. When enabled, index cleanup will remove both pseudo-deleted index entries and pseudo-empty index pages.

NSP IMPROVEMENTS

- In V11, there are a number of improvements which address Native SQL Stored Procedures, especially for distributed callers. For example with ODBC applications, DB2 V11 optimizes result set processing with a progressive streaming logic feature known as “limited block fetch”. This can be turned off with the LIMITEDBLOCKFETCH keyword. You can also enable limited block fetch for LOBs and XML processing, or turn it off with the STREAMBUFFERSIZE keyword.

- V11 introduces autonomous NSP procedures (similar to the ATTACH macro) which can execute under their own units of work, separate from the calling program. The follow the rules of the COMMIT ON RETURN YES option when they finish, *WITHOUT* committing the work of the calling program. The calling program is in control of its own updates and can COMMIT or ROLLBACK independently.

- There are a few restrictions for autonomous procedures: DYNAMIC RESULT SETS must be 0. Parameters to the procedure can’t be LOB data type or XML. Autonomous procedures cannot “see” uncommitted changes from the calling application. Locks are not shared being the called and calling program so they are contending potentially and could cause timeouts/deadlocks. Careful design is necessary.

- You can now define ARRAYs in IBM SQL PL for z/OS similar to the way it works on distributed platforms. This is for use as parameters and variables only, not for DB2 column definitions. Array support will allow you to define arrays as parameters and variables for SQL routines, pass arrays from one procedure to another as arguments for input or output parameters, pass arrays to functions as parameters or from functions as return values, manipulate arrays, transform arrays to tables, and transform tables to arrays by using new built-in functions. Examples:

```
SET RECENT_CALLS = ARRAY[9055553907, 4165554213, 4085553678];

SET DEPT_PHONES = ARRAY[SELECT DECIMAL(AREA_CODE CONCAT '555' CONCAT EXTENSION,16)
                        FROM DEPARTMENT_INFO WHERE DEPTID = 624];
```

XML ENHANCEMENTS

- In V11, there are a number of improvements which address XML.
- Document node types are implicitly defined by INSERT, UPDATE, or XMLDOCUMENT statements.
- LOAD utility performance improved when loading binary XML pre-validated data.
- XML support for the cross-loader function (LOAD with INCURSOR)
- A new UDF XSLTRANSFORM provides support for transforming an XML document based on XSL (XML Style Sheet) specifications.

REORG IMPROVEMENTS

- In V11, when the REORG utility is running in SHRLEVEL CHANGE mode, it can create its own mapping table and index instead of requiring one to have been pre-allocated and defined by the DBA. A new MAPPINGDATABASE keyword lets you specify a database name from which REORG implicitly creates the mapping table and index objects. A zparm REORG_MAPPING_DATABSASE establishes a default database, but that can be overriden by MAPPINGDATABASE for a particular REORG.

- One warning during migration: Current mapping table definitions have a column for LRSN which is CHAR(6). This is the "old" format. You want CHAR(10) even in V11 Conversion Mode. You will receive a warning message like "MAPPING TABLE IS SPECIFIED WITH A NON-EXPANDED LRSN COLUMN" until you convert.

- You can change limit keys for a partitioned table space without impacting availability of data, called an "online alter limit key". In prior versions of DB2 if you changed the limit key(s) all affected partitions were put in REORG-PENDing and were inaccessible without a REORG. In V11, any affected partitions are placed in AREO (Advisory REORG pending). The limit key changes will be materialized the next time you run REORG. This is called a "pending" definition change.

PCTFREE FOR UPDATE

- In V11, you can reserve free space in table spaces for use only by UPDATE operations. Normally, free space is used by both INSERT and UPDATE. So in addition to the usual PCTFREE and FREEPAGE parameters, you can now specify PCTFREE FOR UPDATE at CREATE or ALTER TABLESPACE time.

- The goal is to improve performance especially for applications which tend to increase the size of variable length rows when they UPDATE. Of course this happens also due to data compression, so even apparent “fixed” rows might actually be physically variable. You can maintain higher CLUSTERRATIOs for a longer time in between REORGs. You can reduce the number of NEARINDREF and FARINDREF overflow records which cause an increase in CPU cost and synchronous I/Os (DB2 does not use List prefetch to read overflow records).

- You can specify PCTFREE FOR UPDATE -1 and DB2 will use Real Time Statistics to calculate an optimal value.

- There is a zparm PCTFREE_UPD to set an installation default value for PCTFREE FOR UPDATE.

- If you specify both PCTFREE and PCTFREE FOR UPDATE it is additive – the total free space is the sum of the two values, and both REORG and LOAD will respect that.

PREDICATE SELECTIVITY OVERRIDES

- This is a new feature in V11 which gives the DBA another tool in the arsenal of SQL tuning. It expands the progress made in the past several releases of DB2 towards allowing the SQL coder or DBA to issue "hints" to the optimizer, available in one form or another since V6 and still evolving.

- In V11, you can provide information to DB2 about the expected data distribution for a query which the optimizer can use for access path determination. Some predicates are very difficult for the optimizer to estimate at BIND time, such as BETWEEN :host variable1 and :hostvariable2. Rather than accept a filter factor of .1 or a 10% guesstimate, you could provide the actual percentages (or your best guess as to what they are) by creating "selectivity profiles". This requires four tables:

  – PLAN_TABLE
  – DSN_USERQUERY_TABLE
  – DSN_PREDICAT_TABLE (this is not a typo)
  – DSN_PREDICATE_SELECTIVITY

- After initially populating these tables, you run EXPLAIN which puts various rows in these tables. You then complete the profile by adding your filter factor values, then run BIND QUERY followed by REBIND PACKAGE.

- So you can tell the optimizer for example that predicate 1 will be actually true 20% of the time but predicate 2 will be true only 1% of the time instead of the optimizer defaulting to a 10% assumption for both.

THANKS

Q & A   Anyone?

Please contact the author of this presentation, Hal Steiner, with any questions, concerns, comments, or corrections.

harold.steiner_iii@bankofamerica.com

732-673-6522

@HalSteiner3