

## DB2 9 for z/OS New Features Refresher Sprinkled with User Experience

Diwakar Rao  
Fidelity Investments

# Agenda

NEB20C

June  
2010

- UNIVERSAL TABLESPACES
  - CLONE TABLES
  - ROW CHANGED TIMESTAMP
  - APPEND OPTION ON TABLE
  - IMPLICITLY HIDDEN COLUMNS
  - SKIP LOCKED DATA
  - REORDERED ROW FORMAT
  - PLAN/PACKAGE STABILITY
  - INDEX CHANGES
  - REALTIME STATISTICS
- INSTEAD OF TRIGGER
  - TRUNCATE STATEMENT
  - MERGE
  - SELECT FROM UPDATE, DELETE, MERGE
  - INTERSECT & EXCEPT
  - FETCH FIRST AND ORDER BY IN SUB SELECT AND FULL SELECT
  - NEW BUILT-IN FUNCTIONS
  - NEW OLAP FUNCTIONS

**Universal Table Space (UTS)** NEDB2UG  
June  
2010

- Simple TS deprecated in DB2 9
- A new table space that is both segmented and partitioned is called an UTS.
- Allows a segmented table space grow beyond 64 GB.
- And, Allows partitioned table spaces to be defined like segmented structure.
- Classic Partitioned TS will be deprecated in DB2 10

3

In V9 we lose the ability to create simple tablespace but we get the new UTS.

Why did we get a new tablespace ?

So with segmented tablespace structures the tablespace size is restricted to 64GB, so if we want to create a segmented table space and we expect it to grow beyond 64gb, we are forced to create a partition table space, and to do this we have to invent artificial partitioning keys.

Classic Partitioned table spaces are great, but they lack the performance advantage of better space management provided by segmented structures.

With UTS you can have the best of both the worlds, you can have a segmented TS that can grow beyond 64gb (without inventing partitioning keys) and you can create a partitioned table space with segmented structures for better DML performance.

As you see classic partitioned tablespaces are deprecated in v10,

UTS could be the table space that could replace every other table space we have and be the only choice in the future.

UTSs comes in two flavors.

- Partition By Range (PBR) or range-partitioned
  - Is a regular (pre V9) table/index controlled partitioned table space.
  - + With segmented structures.
- Partition By Growth (PBG)
  - Is a regular (pre V9) segmented table space.
  - + Can grow beyond 64 GB.
  - + Auto partitioned with out partitioning key.

4

UTSs comes in two flavors.... PBR or range-partitioned and PBG...

Partition By Range UTS is nothing but a Table controlled partitioned table (can not be index controlled partition table) but the distinction is that the data pages are segmented structures.

Partition By Growth UTS is nothing but a Segmented tablespace that can grow beyond 64 GB .

Has anyone noticed the importance of segmented table spaces associated with Universal Table Spaces, in the last 2 slides, the word "SEGMENTED" is pretty much on all bullet points.

- Segmented organization is better because
  - Space-map page on segmented structure has more details about free space for each page in it's range.
  - Improved Insert and update performance.
  - Deletes produces less logging, and inserts can reuse deleted space immediately before REORG.
  - Improved incremental copy performance.

So why is the segmented tablespace so much better than the partitioned tablespace and the now un-supported simple tablespace.

It's all about the space map pages!

The space map pages in an segmented structure has more detailed information about free space on the data pages that follow. Each space map entry in a segmented table space is represented by 4 bits while that of a non-segmented table space is represented by 2 bits.

Since DB2 checks the space map page to find if space is available on a page to insert a row or update a row and make it longer. The additional information on segmented tablespace space map pages is a primary reason that segmented Tablespaces have advantages over non-segmented Tablespaces.

With segmented you get one space map page for every 5,000 data pages, but with partitioned just one for every 10,000 pages .

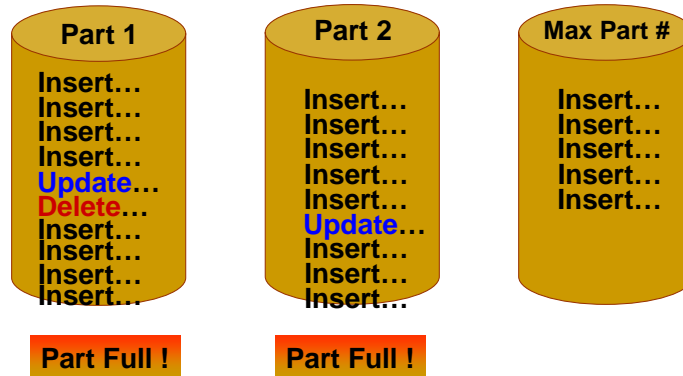
Since only the space map page is updated to mark mass deletes, it results in less logging and making it efficient for mass delete operations and INSERTS can reuse the deleted space immediately.

Better space management = Less REORG.

## Partition By Growth

NEB820G  
June  
2010

- A PBG table space starts with a single partition table space.
- Automatically adds new partitions as required until maxpartitions.



6

DB2 starts out with one partition. When a partition is filled, DB2 automatically defines the next partition and starts using it .

So this table space acts like a non-partitioned table space but with the added benefit of being able to grow beyond current limitation of 64GB to a maximum of 16TB for 4KB page size and up to 128 TB in size (for 32 KB page size).

Since partition-by-growth table spaces are segmented partitioned, they offer almost all capabilities of partition-level operation.

- Create a table specifying the table space name created using **MAXPARTITIONS**.

```
CREATE TABLESPACE TS001 IN DB001  
MAXPARTITIONS 60  
SEGSIZE 32  
DSSIZE 8G;
```

Maximum size for Each partition

Makes Table Space PBG

```
CREATE TABLE table_name IN DB001.TS001  
PARTITION BY SIZE EVERY 8G;
```

For Partition by Growth, You start by creating a tablespace with MAXPARTITIONS keyword, and then create the table in that tablespace.

MAXPARTITIONS keyword specify the maximum number of partitions to which the partition-by-growth table space may grow.

Max partitions depends upon the page size and DSSIZE.

- No DPSIs and no PIs, only NPIs.
- Load part level not allowed.
- Alter ADD and ROTATE part not allowed.
- No Parallelism
- No REORG REBALANCE

**Think of PBG table as segmented table on steroids!, It can grow on demand, grow > 64G, allow partition level operation for some utilities.**

Some of the limitations of partition by growth table spaces are..

Allows only one table per table space, just like a regular partitioned table space.

Since you are not allowed to define partition key ranges, you can not define Partitioned Indexes or DPSIs.

All utilities can operate at the partition level except LOAD utility.

Compression dictionary will be copied from previous partition to the new partition.

- Create a table controlled partitioned table space with **SEGSIZE** clause.

```
CREATE TABLESPACE TS001 IN DB001  
  NUMPARTS 60  
  SEGSIZE 32  
  DSSIZE 8G;
```

Makes Table  
Space PBR

- Segmented table space structure provides better performance than classic partitioned tables.
- Must be an UTS when CLONE table is required.
- MEMBER CLUSTER not allowed.

9

Partition By Range table space is a regular partitioned table space with segmented space structure.

When you specify SEGSIZE with NUMPARTS, you get a Partition By Range UTS.

Better space management

Better mass delete performance,.

UTS is required if you want to CLONE the table.. Another new feature in V9!

## CLONE TABLE RATIONALE

WED820G  
June  
2010

- “. . . many of our processes involve completely emptying the table and then reloading it in its entirety. In order to accomplish this the table space must be offline which does impact the availability of the business application . . . “
- “. . . It is increasingly difficult to do LOAD REPLACES of data which is required for business as their business becomes more and more 24x7x365 with online transactions 24 hrs a day.”

10

Do you have tables that gets completely emptied and reloaded in its entirety frequently and impact availability of business application?

Clone table support in V9 can help alleviate some of the pain associated with such processes.

## CLONE TABLE use sequence

NEB820G

June  
2010

- ALTER TABLE *emp* ADD CLONE *emp\_c* (*emp* is the base table)
  - DB2 creates a clone table called *emp\_c*
  - Clone *emp\_c* looks and feels like *emp*, but has NO data rows
  - Transactions/SQL are still reading *emp* at this point, not the clone
- INSERT or LOAD data into the clone table *emp\_c*
  - Transactions are still reading *emp*, and the clone *emp\_c* has data
- EXCHANGE DATA BETWEEN TABLE *emp* AND *emp\_c*
  - DB2 switches gears and diverts transactions/SQL traffic from *emp* to *emp\_c*
  - Transactions/SQL referring to *emp* table will now read data from the clone table *emp\_c*
  - No bind/rebind required, static bound packages and dynamic SQL will point to the clone table after EXCHANGE.
- EXCHANGE DATA BETWEEN TABLE *emp* AND *emp\_c*
  - Will switch transactions/SQL access back to *emp* table.
- ALTER TABLE *emp* DROP CLONE : will drop the clone table

11

Lets quickly walk thru the sequence of events when we use CLONE tables..

On the next few slides, we will look at how db2 does this trick !

NED820G  
June  
2010

## ADD CLONE

DB2xV...TS00001. I0001.A001

**MKTDBO1.T\_SALE...**

ALTER TABLE  
**MKTDBO1.T\_SALE...**  
ADD CLONE  
**CLONEDBO.C\_SALE**

DB2xV...TS00001. I0002.A001

**CLONEDBO.C\_SALE...**

SYSTABLES		
Name	Owner	Type
T_SALE	MKTDBO1	T

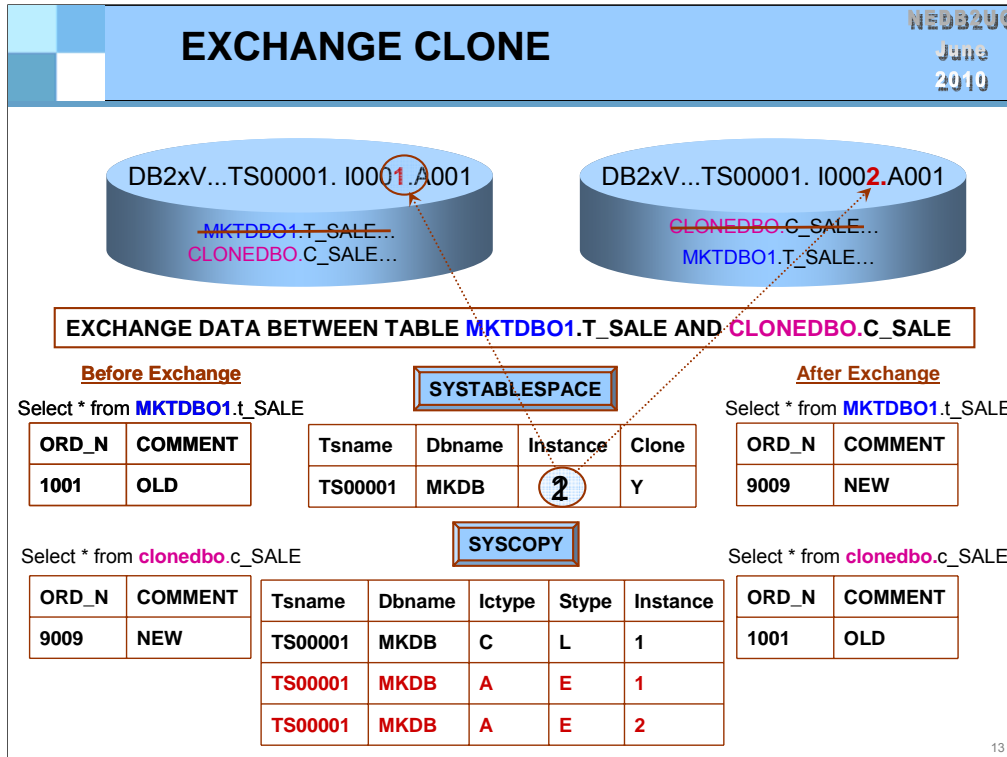
SYSTABLESPACE			
Tsname	Dbname	Instance	Clone
TS00001	MKDB	1	N

SYSTABLES		
Name	Owner	Type
T_SALE	MKTDBO1	T
C_SALE	CLONEDBO	C

SYSTABLESPACE			
Tsname	Dbname	Instance	Clone
TS00001	MKDB	1	Y

12

- Here you see for tablespace TS00001 the dataset name, note that the instance number is 1
- SYSTABLES catalog table has one entry for the SALE table, note the table type as “T” for table
- SYSTABLESPACE catalog table has one entry for the tablespace TS00001 , note the instance is “1” and says “N” suggesting no clone has be defined yet on this table.
- We ALTER SALE table and ADD C\_SALE clone table.
- DB2 Create a new table with the same TABLESPACE name TS00001, but with a different instance number “2”
- Systable gets a new row for the clone with type “C” for clone
- Systablepsace row is updates to “Y” on the clone column.
- At this point applications are still reading from the base table (not the clone) ..
  
- Lets move on to the next slide..



- Now after the ADD, we have the base table and the clone table,
- base with instance #1 and the clone has 2
- Note the systablespace is pointing to instance number "1",
- We now execute the EXCHANGE DATA
- Note the instance number in systablespace changed to 2, from this point on, all transaction/SQL will read the mirror table.
- For all applications/SQL the clone becomes the base table.
- DB2 must drain all access before it lets the EXCHANGE complete.
- SYSCOPY get couple of rows to show that the EXCHANGE was executed, (if you ever want to find out when the exchange was executed)
- Lets take a look at how data looked like before exchange
- And After Exchange.

- Exchange drains both base and clone objects.
- EXCHANGE will take a DBD lock, prevent DDL from executing in the database.
- Exchange will invalidate RTS stats for the base table.
- When an EXCHANGE statement is running, none of the packages that are dependent on the target table involved can be executed.

- Cloned table support provides the ability to generate a table with the exact same attributes as a table that already exists .
  - Same structure as base table
  - Same column names, data types, null attributes, check constraints, indexes & before triggers.
- Not cloned ?
  - Views, authorization
- Cannot clone
  - Tables with RI and Tables with After triggers
  - No clones allowed on MQTs
  - Tables with a clone table!
- Table **MUST** be in a UTS to create a clone

With clone table support, you can generate a copy of a current table, in the same table space, that has the same attributes and structure as the original table.

After you create a clone table, you can insert or load data into the clone table and exchange the clone table name with the current table name.

In a nutshell, “clone” tables provide faster replacement of one table with another.

## Row Changed Timestamp & Token

NED82UG  
June  
2010

- A row changed timestamp column may be added to a table that will automatically track the last time a row was updated.
- Useful for auditing and for implementing optimistic locking.

```
CREATE TABLE EMP (EMPNO INTEGER, F_NAME CHAR(30),...  
LAST_UPD_TS      TIMESTAMP NOT NULL  
                  GENERATED ALWAYS FOR EACH ROW  
                  ON UPDATE AS ROW CHANGE TIMESTAMP)
```

- You can use the new expression on any table. (even on tables with NO timestamp column)
  - ROW CHANGE TIMESTAMP FOR <corr\_id> or
  - ROW CHANGE TOKEN FOR <corr\_id>

```
SELECT EMPNO,F_NAME, ROW CHANGE TIMESTAMP FOR T1 FROM EMP T1;  
SELECT EMPNO,F_NAME, ROW CHANGE TOKEN FOR T1 FROM EMP T1;  
– For table with NO timestamp column defined, DB2 will get the timestamp from the data page.
```

16

You can now create a true last update timestamp column that's DB2 updates on new INSERTS and when any column of the table is UPDATED!

No need to trust the application, or maintain a After update trigger to keep the last update timestamp updated!

We get 2 new expressions, these can be used on any table, even when the table has no ROW CHANGE TIMESTAMP column defined.

When the table has no ROW CHANGE TIMESTAMP column defined, DB2 will get the timestamp from the data page, so all rows on a data page will return the same timestamp and token.

If the table has the ROW CHANGE TIMESTAMP defined, these expressions return unique timestamp OR token for each changed row.

- OCC is an alternative to database locking for concurrent access. (Not a new feature!)
- Used to minimize the time for which a given resource is unavailable for use by other transactions
- Locks for Reads
  - Obtained immediately before a read operation and released
- Locks for Updates
  - Obtained immediately before an update operation and held until commit
- Allows simple timestamp and RID predicate to validate that row has not changed since last access
  - The new RID() function returns ROW ID, for direct row access.
- It's not automatic, you still have to program for it.

```
DECLARE CURSOR1 CURSOR FOR
SELECT Col1, Col2, Col3, RID(T1), ROW CHANGE
TIMESTAMP FOR T1 FROM TableA T1 WHERE ... FOR FETCH
ONLY (or) WITH UR;

UPDATE TableA T1 SET .....
WHERE RID(T1) = :RID-HV AND ROW CHANGE TIMESTAMP
FOR T1 = :UPD-TS-HV;
```

Optimistic concurrency control, also known as *optimistic locking*, represents a faster, more scalable locking alternative to database locking for concurrent data access.

It minimizes the time for which a given resource is unavailable for use by other transactions.

When an application uses optimistic concurrency control, locks are obtained immediately before a read operation and released immediately.

Update locks are obtained immediately before an update operation and held until the end of the transaction.

Optimistic concurrency control uses the RID (for direct row access) and a row change token to test whether data has been changed by another transaction since the last read operation.

Note: RID can change if a row gets moved.. Maybe due to REORG or update...

## Implicitly Hidden Columns

- Specify “IMPLICITLY HIDDEN” when you create table OR alter add column.
- Hidden column is not visible in the results of SELECT \*
- These columns can be explicitly SELECTed
- DCLGEN / DESCRIBE do not include the column
- NOT NULL & IMPLICITLY HIDDEN are compatible keywords but can lead to awkward behavior

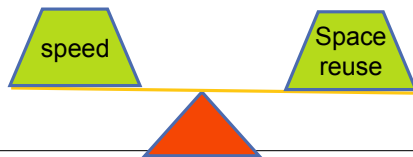
18

IMPLICITLY HIDDEN causes the column to not appear in DCLGEN, Describe, or SELECT \*.  
It can be explicitly referenced of your SELECT

Not appearing in the DCLGEN / Described, if these columns are also defined as NOT NULL;  
INSERT statements may be awkward as a column not visible in the DCLGEN / Describe if not  
provided and yet is needed.

NOT NULL WITH DEFAULT may be a better choice for IMPLICITLY HIDDEN columns.

- Balancing qualities of:
  - Storing the data quickly at the cost of space & certain queries
  - More effective use of space at the cost of searching for space
- CREATE / ALTER Table APPEND option:
  - Maximizes performance for “INSERT at end”
  - Avoids overhead of attempting to preserve clustering sequence
  - REORG after bulk insert to establish clustering
- Useful for:
  - Cases where cluster is not important
  - Large batch INSERTs following by a REORG



### Table Append option

The Table Append option offers increased performance for inserting data into the end of a table. It reduces the instructions used to target the locations for new rows.

If your application is keyed access, you will not see performance issues for our queries, queries with sequential access will suffer due to lack of clustering.

# APPEND TEST

NED82UG  
June  
2010

One job Insert 1.3m rows. With 68% clustered data						
Test case	CPU seconds		Elapsed seconds		Getpages	
Baseline segmented TS	162		349		26m	
APPEND + seg TS	158	-2.47%	342	-2.01%	24m	-6.35%
APPEND + Partiton TS and Member Cluster	161	-0.62%	320	-8.31%	23m	-8.91%

Three concurrent jobs run all on the same CPU/member insert about 1.8m rows in total.						
Test case	CPU seconds		Elapsed seconds		Getpages	
Baseline segmented TS	735		304		37m	
APPEND + seg TS	879	19.59%	312	2.63%	37m	0.85%
APPEND + Partiton TS and Member Cluster	936	27.35%	348	14.47%	35m	-4.11%

- Mileage varies! Test before you implement.

- Sort avoidance for GROUP BY
  - Order of GROUP BY columns re-arranged to match index
    - Data may be returned in a different order without an ORDER BY in V9. (order is NOT guaranteed without ORDER BY)



```
SELECT C2, C1  
FROM T1  
GROUP BY C2, C1
```

← required sort in V8 , and used to return in order.

Applications depending on GROUP BY to return data in order must be changed to add ORDER

BY.

Index 1 (C1, C2)

- Improved Sort avoidance for DISTINCT
  - From V9, DISTINCT can avoid sort using duplicate index
    - DISTINCT required unique index to avoid sort

In V8, grouping is done after sort input processing. In DB29 DB2 can avoid sort when a support index exists.

This is good from a performance point of view that DB2 is not doing the sort, this could also be a problem for existing applications that might be getting the correct order even with the order by, this might change when we get to v9.

So, it's really important to remember if you have situations in your application where you require the data in certain order and don't have an ORDER BY clause to match the required order, this is the time to find those and fix it!

- Rows with incompatible locks by other transactions are skipped
- You can SKIP LOCKED DATA
  - On SELECT, SELECT INTO, PREPARE, searched UPDATE, searched DELETE (or) UNLOAD utility
  - Only works with CS or RS isolation level
    - Otherwise it is ignored
  - Only works with locks at page (or) row level
    - Otherwise it is ignored
- No warning issued when rows skipped. (partial truth!)

#### Logic / Scenario

- When a transaction needs to find work to do, regardless of order.
- Messaging applications without strict ordering requirements expect to be able to skip over records that are locked by other transactions



Valuable for applications that perform queuing or messaging, SKIP LOCKED DATA might be a good match. This allows DB2 to skip past the rows of data that have an incompatible lock type rather than generate lock time.

## SKIP LOCKED DATA vs. WITH UR

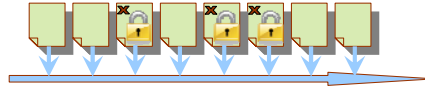
NED820G  
June  
2010

### • SKIP LOCKED DATA



- Skip locked data rows/pages
- Only committed rows show up, not all committed rows show up
- Apply to SELECT, UPDATE & DELETE
- Can specify only at statement level
- May return partial truth!

### • WITH UR (Uncommitted Read)



- Read thru locked/uncommitted data
- Uncommitted rows can show up
- Apply to read-only SELECT statements
- Can specify at statement and at package level as bind option
- Could be an outright lie!

23

Difference between SKIP LOCKED DATA vs. WITH UR

## Reordered Row Format (RRF)

- Automatic repositioning of Variable columns to end of row
- Any table space created in DB2 9 NFM
- To Convert:
  - REORG or LOAD REPLACE a table space or partition
  - ADD PARTITION
- Catalog / Directory remains in Basic Row Format (BRF)
- SPRMRRF=ENABLE|DISABLE controls row format for new tablespaces

Prefix	Fixed Length Cols	Varchar Indicators	Varying Length Cols
--------	-------------------	--------------------	---------------------

- Ability to backup your static SQL packages access path.
- At REBIND
  - Save old copies of packages in Catalog/Directory
  - Switch back to previous or original version
- Two flavors
  - BASIC
    - 2 copies: Current and Previous
  - EXTENDED
    - 3 copies: Current, Previous, Original
  - Default can be controlled by a ZPARM
  - Also supported as REBIND options

One of the concerns we have when we migrate to a new version of DB2 is the performance of some of their thousands of SQL queries may regress. Usually IBM recommends customers to rebind all of our static SQL to take advantages of the new optimization enhancements delivered in the new release.

•The idea of plan stability is to provide the functionality of backing up the previous plans. It makes it possible to recover when the performance of the new access plan proves to be less efficient than the previous plan.

•There are two flavors, BASIC and EXTENDED, where in BASIC you can store 2 copies and in EXTENDED you can store 3 copies. The default is controlled by a ZPARM and it is also supported as REBIND options.

- Larger index pages allow for more efficient use of storage
  - Fewer page splits
  - More key values per page
- Define RANDOM index keys to avoid hot spots with multiple processes inserting sequential keys
- Index compression provides page-level compression
  - Data is compressed to 4K pages on disk (not in buffer pool)
  - 32K/16K/8K pages results in up to 8x/4x/2x disk savings
  - No compression dictionaries
    - Compression on the fly
    - No LOAD or REORG required



26

Indexing improvements contribute to the overall improvements in query performance.

- Specific improvements include **large index pages**, you can now have index pages that are 8, 16 or 32k,

- Larger index pages means fewer splits (just by going to a 8k you will reduce your splits by 50%) better performance for high insert applications

- And larger pages also helps with Index compression.

- **Index key randomization**, Multiple processes inserting sequential keys can create hotspots on indexes. Randomized index keys avoid hot spots.

Application insert throughput improved via avoidance of locking conflicts, but retrieval of sequential rows is likely to be slower.

- **Compression**, In data warehouse type applications, it is common to have many very large indexes defined on large tables. It is also possible to have index spaces that are much larger than the tables they are based upon.

- Index pages are stored on disk in their compressed format (physical 4 KB index page on disk)

- Simple indexes can contain concatenated columns

Create index empix01 on  
EMP(salary, bonus)

- Index on expression
  - Value of the index has been transformed
  - May not be the value of any of the columns that it is derived from
  - Optimizer can use this index

Create index empix02 on  
EMP(salary+bonus)

EMP

name	salary	bonus
Raisa	80,000	6,500
Paul	40,000	5,000
Sally	400,000	10,000
Rex	40,000	2,000
Dave	200,000	20,000

Select name from emp  
Where salary+bonus > 100000

27

## From “What’s New”

Support for a new index type, index on expression, lets you create an index on a general expression. Query performance can be enhanced if the optimizer chooses that index. When you use an index on an expression, the results of the expressions are evaluated during insertion time or during an index rebuild and are kept in the index. If the optimizer chooses to use that index, the predicate is evaluated against the values that are stored in the index. As a result, run-time performance overhead is eliminated.

In the example, we are satisfying a query that is searching for the “name” with the highest gross income (calculated as salary+bonus).

- CREATE INDEX ...  
(SUBSTR(SSN,6,4))
- CREATE INDEX ...  
(UPPER(LASTNAME),  
UPPER(FIRSTNAME))
- CREATE INDEX ...  
(COL1 concat COL2)
- Can be defined UNIQUE
- Can be only in ASC order
- Cannot be clustering index
- Must reference to at least  
one table column
- Many more restrictions....

Where in the catalog can you find the expression ?

```
Select KEYSEQ,DERIVED_FROM from SYSKEYTARGETS  
WHERE IXNAME = ?
```

More example on INDEX on Expressions, and listed some of the restrictions on the right..

- Real Time Statistics (RTS) have been collected automatically since V7
- With RTS, DB2 gathers performance and maintenance statistics about database objects “Real-Time”!
- Prior to V9, users were responsible for creating and managing these tables manually outside the scope of the DB2 catalog.
- DB2 9 moves RTS into the Catalog & externalizes the information automatically
  - DSNDB06.SYSRTSTS
    - SYSIBM.SYSTABLESPACESTATS
    - SYSIBM.SYSINDEXSPACESTATS



Real time statistics are always (since V7) collected by DB2 and stored in-memory. DB2 just does not externalize the statistics to the RTS tables unless RTS is enabled (prior to V9 ENFM) by the user.

Now in V9 db2 moves the RTS into the catalog and automatically externalizes the information.

- RTS records the index last used date.
  - SYSINDEXSPACESTATS.LASTUSED
- "Used", as defined by DB2 as:
  - As an access path for query or fetch.
  - For searched UPDATE / DELETE SQL statement.
  - As a primary index for referential integrity.
  - To support foreign key access

New feature in RTS, able to identify unused indexes, maybe now we can drop all those unused indexes sitting there and costing in application performance and wasting disk space.

## INSTEAD OF Triggers

- INSTEAD OF triggers can only be defined on views.
- Typically, a view that consists of multiple base tables canNOT be updated.
- INSTEAD OF triggers enable views that would not otherwise be updatable to support updates.
- With an INSTEAD OF trigger you can code logic to direct inserts, updates and deletes to the appropriate underlying tables that comprise the view.
- Application still believes all operations are performed against the view
- Applicable even for updatable views

31

INSTEAD OF triggers work only on Views, and we don't have much use for this feature because we don't have many views.

And we prefer updating the base table when required.

## INSTEAD OF Trigger - Notes

NED820G  
June  
2010

- Only 1 INSTEAD OF INSERT, UPDATE, DELETE per view
- No WITH CASCADE CHECK OPTION
- Only has row granularity
- No AFTER or WHEN clauses
- No LOB, XML, or Field Procs
- Cannot specify UPDATE OF column list
- Cannot change transition variables
- Does not work with position UPDATE / DELETE
- CCSID View must be the same
- View cannot have a dependent view
- SELECT FROM UPDATE/DELETE/INSERT not supported
- MERGE into a view with INSTEAD OF trigger is not supported

32

Other details and restrictions ...

- Current issues when emptying out table:
  - Mass delete via SQL – trigger will fire
  - Load replace – operates at tablespace level instead of table.
- DELETE triggers will **NOT** be fired by default.

```
TRUNCATE TABLE EMP  
RESTRICT WHEN DELETE TRIGGERS  
IMMEDIATE; (ROLLBACK cannot undo when IMMEDIATE used)
```

- TRUNCATE not allowed on **parent table.** (even when child table is empty)
- Performance same as mass DELETE , advantage is TRUNCATE can ignore DELETE triggers. But!
- The **ALTER table privilege** is needed for IGNORE DELETE TRIGGERS ☹

33

DB2 might be the last DBMS to get this functionality!  
Easy statement to use to delete all rows in a table.

The advantage with TRUNCATE compared with mass deletes is the option to IGNORE after delete triggers.

Cant truncate parents!, that rules out lots of tables!

Performance is the same as mass deletes.

Truncate also needs ALTER table privilege, I don't see DBA handing out that privilege anytime soon!, so really not much of a developer tool!.

The best way to clear a table fast and without firing the triggers is to do a dummy load replace utility with LOG NO option.

### Retrieve identity column values.

Find value of Identity Column before insert (assume table created with identity column on ACCT\_ID generated always)

```
SELECT ACCT_ID
FROM FINAL TABLE (INSERT INTO RTD1DBO.ACCOUNT
(NAME, TYPE, BALANCE)
VALUES ('Master Card', 'Credit', 50000) )
```

### Retrieve multiple rows from a multi row insert.

```
DECLARE CS1 CURSOR WITH ROWSET POSITIONING FOR
SELECT COL_A, COL_B FROM FINAL TABLE
(INSERT INTO T1 VALUES (:hva, :hvb) FOR :hvn ROWS)
WHERE C1 > :hv1 AND C2 < :hv2;
OPEN CS1;
FETCH NEXT ROWSET FROM CS1 FOR :hvn ROWS
INTO :ARRAY1, :ARRAY2;
```

Before talking about SELECT from UPDATE and DELETE, here is a refresher from DB2 V8!, most of you might have used this pretty sleek feature in V8 already.

The most common use of this will likely be to retrieve automatically generated values such as timestamps and identity columns.

The idea was to reduce the number trips between the application and the database.

## SELECT FROM UPDATE/DELETE

NEB820G  
June  
2010

DB2 V9 allows the FROM clause of a SELECT statement to contain a searched UPDATE, a searched DELETE

```
SELECT SUM(SALARY) INTO :SAL-HV FROM  
FINAL TABLE (UPDATE EMP SET SALARY = SALARY *  
1.22 WHERE JOB = 'DB2DBA');
```

Also SELECT FROM  
"OLD TABLE" for before values

```
DECLARE DEL_CSR CURSOR FOR  
SELECT NAME,SALARY INTO NAME-HV,;SAL-HV  
FROM OLD TABLE (DELETE FROM EMP  
WHERE JOB = 'MANAGER');
```

35

Same idea here, to reduce the trips between the application and the database..

In this example ..we are giving the DBA's a 22% raise! I'm allowed to dream! And we are selecting the sum(salary) in a singleton sql.

And in the second example we are deleting all MANAGER JOBS, oops.. That was a typo! Here we are listing all name with the corresponding salary.

FINAL TABLE and the OLD TABLE are KEYWORDS here! For select from update you will have the before values and the after values, OLD TABLE has the before values, and FINAL TABLE has the final/after values!

## INCLUDE COLUMN

NEB82UC  
June  
2010

- The include-column allows you to specify one or a list of additional columns on the select list.
- The included columns are appended **to the end** of the list of columns that is identified by target table.
- The INCLUDE can be specified only if the DELETE/INSERT/UPDATE/MERGE statement is nested in the FROM clause of a SELECT statement.

```
SELECT SUM(SAL), SUM(BEF_SAL) , SUM(RAISE) FROM  
INTO :SAL-HV, :BEF-SAL-HV FROM FINAL TABLE (  
UPDATE EMP INCLUDE(BEF_SAL DEC(9,2), RAISE DEC(9,2))  
SET SAL = SAL * 1.22,  
BEF_SAL = SAL, RAISE = (SAL * 1.22) - SAL  
WHERE JOB = 'DB2DBA');
```

36

So for the example from the previous slide, lets say you want to select the sum(before salary) and the sum(raise)

You can do that by using the new INCLUDE keyword and include 2 columns specify datatypes,

And then derive values for the new columns, as it's done here.

## MERGE (UPdate or InSERT = UPSERT!)

WED820G

June  
2010

- MERGE statement UPDATES if the row exists (or) INSERTs when not found all in one single SQL statement.
  - Inserted or updated row is immediately available for more updates in the same statement. (example in next slide)
  - NOT ATOMIC CONTINUE ON SQLEXCEPTION is the ONLY option if you are trying to MERGE > 1 row.
  - Can use GET DIAGNOSTICS to determine specific row that had an error, as with any multi-row operation.
- IBM Reported improvements : 13% for static and 20% for dynamic SQL.

37

Merge statement, the way you think about this is ,if you are a developer you might have been hit with this situation, where you have given some input, where you need to look at the input and decide if the row exists in the table already, I want to UPDATE it , if it doesn't exists, I want to INSERT it. So up thru v9 the only way we were able to do that is we either do a select to check, or do a insert first, and if that fails, do an update, back and forth with the application many times wasting CPU cycles..

When you move into v9, you get the Merge statement, what the MERGE statement does is, combined the UPDATE and Insert into one statement. Some other DBMS have it as UPSERT.

A typical use will have many rows to merge. As with multi-row insert, we use arrays to provide the multiple rows of data to merge into the table.

A common requirement is to present data using a spreadsheet metaphor, where rows can be inserted or modified on the screen.

## MERGE Example

NED820G  
June  
2010

```
MERGE INTO account AS T
USING (VALUES (:hv_id, :hv_amt) FOR 5 ROWS)
AS S (id,amt)
ON (T.id = S.id)
WHEN MATCHED THEN

UPDATE SET balance = T.balance + S.amt

WHEN NOT MATCHED THEN

INSERT (id, balance) VALUES (S.id, S.amt)
NOT ATOMIC CONTINUE ON SQLEXCEPTION
```

The diagram shows a SQL MERGE statement with several callout boxes pointing to specific parts of the code:

- Target Table**: Points to 'account AS T'.
- Input :HV (or) :HV-array**: Points to '(VALUES (:hv\_id, :hv\_amt) FOR 5 ROWS)'.
- Source Table**: Points to 'AS S (id,amt)'.
- Search/Join Condition**: Points to 'ON (T.id = S.id)'.
- When found UPDATE**: Points to 'UPDATE SET balance = T.balance + S.amt'.
- ELSE INSERT**: Points to 'INSERT (id, balance) VALUES (S.id, S.amt)'.

38

- In this example we are MERGEing into the ACCOUNT table, this is our target table
- USING VALUES is the input values we are trying to MERGE, this can be a host variable array.
- On AS S , you name the Source table (this is the name you provide to your input data)
- The ON condition is your match condition, usually all primary key values.
- When found UPDATE, else INSERT.

## MERGE Example

NEB82UG  
June  
2010

Account (Target table)

id	balance
100	1000
200	10
300	500
400	444
500	557
600	334
700	223

Transaction (Source Rows)

id	amt
→ 100	40
→ 200	10
→ 300	60
→ 200	20
→ 100	70

39

Lets walk thru a MERGE example in action. Results of a MERGE operation.

So, in the future if you have requirements like this, you may try to use the MERGE.

## SELECT from MERGE Example.

NEB820C

June  
2010

```

SELECT id, balance, status FROM FINAL TABLE (
MERGE INTO account AS T INCLUDE (status, char(3))
USING VALUES (:hv_id, :hv_amt) FOR 5 ROWS AS S (id,amt)
ON T.id = S.id WHEN MATCHED THEN
UPDATE SET balance = T.balance + S.amt, status = 'upd'
WHEN NOT MATCHED THEN
INSERT (id, balance,status) VALUES (S.id, S.amt, 'ins')
NOT ATOMIC CONTINUE ON SQLEXCEPTION )
    
```

Target table before		Source/ input		Select result			Target table after	
id	balance	id	amt	id	balance	status	id	balance
100	1000	100	40	100	1040	upd	100	1110
300	500	200	10	200	10	ins	200	30
400	444	300	60	300	560	upd	300	560
500	557	200	20	200	30	upd	400	444
600	334	100	70	100	1110	upd	500	557
700	223						600	334
							700	223

So you got SELECT from INSERT, UPDATE, DELETE, IBM decide to add SELECT from MERGE in this release and not wait for V10.

With MERGE it could get confusing as to what rows got updated and what rows got inserted.

With the help of SELECT from MERGE and the new INCLUDE keyword, we can device a SQL that can list if the rows were either updated or inserted.

We INCLUDE a column called STATUS, and on UPDATE we set it to UPD and on INSERT we set it to INS (pretty straight forward)

## INTERSECT and EXCEPT

NED820G

June  
2010

- Intersect (distinct) or Intersect all.
- Except (distinct) or Except all.
- V9 Intersect Distinct = V8 WHERE EXISTS.
- V9 Except Distinct = V8 NOT EXISTS.
- V9 Intersect All and Except All – no counterpart in V8.
- Overall better performance than V8 counterpart when available.
- When deciding which rows are duplicates of each other, values in each column must be identical.
- EXCEPT is positional in that A EXCEPT B is not the same as B EXCEPT A.
- Identical column structure on both sets is required.

41

We also get new SET operations in V9, just like the UNION operator, you now get 2 new operators, the EXCEPT AND INTERSECT.

Similar to UNION, You are familiar to what UNION does right.

Lets take a look at some examples to make sense of this new set operations..

## INTERSECT and EXCEPT

NED820G  
June  
2010

```
SELECT NAME FROM EMP  
EXCEPT ALL  
SELECT NAME FROM  
EMP_O ORDER BY 1
```

```
SELECT NAME FROM EMP  
EXCEPT DISTINCT  
SELECT NAME FROM  
EMP_O ORDER BY 1
```

```
SELECT NAME FROM EMP  
INTERSECT ALL  
SELECT NAME FROM  
EMP_O ORDER BY 1
```

```
SELECT NAME FROM EMP  
INTERSECT DISTINCT  
SELECT NAME FROM  
EMP_O ORDER BY 1
```

42

SQL used in the example.

We can do similar things with INTERSECT and EXCEPT like the UNION

NED820G  
June  
2010

## INTERSECT and EXCEPT

EMP	EMP_O	UNION ALL	UNION (DISTINCT)	EXCEPT ALL	EXCEPT (DISTINCT)	INTERSECT ALL	INTERSECT (DISTINCT)
Bruce	Bruce	Bruce	Bruce	Bruce	David	Bruce	Bruce
Bruce	Bruce	Bruce	David	David	Tom	Bruce	Dian
Bruce	Dian	Bruce	Dian	David		Dian	Sally
David	Dian	Bruce	Sally	David		Sally	
David	Dian	Bruce	Tom	Sally			
David	Dian	David		Tom			
Dian	Sally	David					
Sally		David					
Sally		Dian					
Tom		Dian					
		Dian					
		Dian					
		Dian					
		Sally					
		Sally					
		Sally					
		Tom					

UNION

EXCEPT

INTERSECT

An example showing the results from each of these set operations.

Both of these new set operations work similarly to UNION, which has existed since the beginning days of DB2.

UNION gives all rows in both Tables regardless of which they originated from.

EXCEPT, on the other hand, combines non-matching rows from two result tables. Some other DBMS implementations refer to this as the MINUS operation.

INTERSECT is used to match result sets between two tables. If the data is the same in both results sets it passes through. When INTERSECT ALL is specified, the result consists of all rows that are in both result sets.

- FETCH FIRST and ORDER BY clauses are now allowed in subselects.
- In this example both clauses are used in the subselect as a way to select the top 3 rows from table EMP as ordered by columns 4,5 for INSERT example on the left and by SALARY for the SELECT example on the right.

```
INSERT INTO EMP_OUT
(SELECT * FROM EMP
ORDER BY 4,5
FETCH FIRST 3 ROWS ONLY);
```

```
SELECT DEPTNAME FROM DEPT D
WHERE D.DEPTNO IN (SELECT
E.DEPTNO FROM EMP E
ORDER BY SALARY DESC
FETCH FIRST 3 ROWS ONLY);
```

In DB2 V8, the ORDER BY and FETCH FIRST n ROWS ONLY clauses are only allowed at the statement level as part of select-statement or a SELECT INTO statement.

You can now specify FETCH FIRST and ORDER BY in the subquery.

Why is this a problem?

- Assume a large table
- You only want the first 1000 rows sorted in a particular order
- Coding a SELECT using FETCH FIRST and ORDER BY clauses does the trick, but the sort is done before the fetch (runs a large sort for no reason).
- As a work around you could code use a temp table, but that requires a lot more work.

- **LPAD** - Returns a string that is padded on the left, with blanks (or a specific character).
- **RPAD** - Does the same but on the right.
- **OVERLAY** - Returns a string with portions of it overlaid by a specified value.
- **SOUNDEX** – Returns a 4-char code that represents the sound of the words in the argument
- **DIFFERENCE** – Returns a value from 0 to 4 that represents the difference between the sounds of 2 strings.
- **MONTHS\_BETWEEN** – Returns an estimate of the number of months between two date/timestamp

## OLAP function ROW\_NUMBER

NEB820G  
June  
2010

- Numbers the rows in a result table starting with 1.
- To ensure that the rows are in a particular order, include an ORDER BY clause after the OVER keyword.

```
SELECT EMPNO, FIRSTNAME, LASTNAME, SALARY,  
ROW_NUMBER() OVER (ORDER BY SALARY DESC)  
AS SAL_ROWNUM FROM EMP ORDER BY EMPNO
```

- If the ORDER BY clause is not specified in the window, the row numbers are assigned to the rows in an arbitrary order, as the rows are returned.

```
SELECT EMPNO, FIRSTNAME, LASTNAME, SALARY,  
ROW_NUMBER() OVER () AS ROWNUM FROM EMP
```

46

Ability to assign a row number.

Notice there are 2 order by clause, the first order (order by salary desc) by is to say how I want my row numbers assigned, and the 2<sup>nd</sup> order by is to say how I want my result set ordered.

You can also assign arbitrary row numbers OVER nothing!

- Returns a rank number for each row value.
- DB2 provides two functions (both return lowest only)
  - RANK - Rank numbers are to be skipped when duplicate row values exist
  - DENSE\_RANK - Rank numbers are **not** to be skipped when duplicate row values exist

## RANK()

```
SELECT EMPNO, FIRSTNME,  
LASTNAME, SALARY,  
RANK() OVER  
(ORDER BY SALARY DESC)  
AS SAL_RANK  
FROM EMP  
ORDER BY EMPNO
```

## DENSE\_RANK()

```
SELECT EMPNO, FIRSTNME,  
LASTNAME, SALARY,  
DENSE_RANK() OVER  
(ORDER BY SALARY DESC)  
AS SAL_DENSE_RANK  
FROM EMP  
ORDER BY EMPNO
```

RANK : Ability to rank and skip duplicates.

DENSE\_RANK : Ability to rank and NOT skip duplicates.

## ROW\_NUMBER(), RANK() and DENSE\_RANK()

NEDB2UG  
June  
2010

EMP NO	FIRSTNAME	LASTNAME	SALARY	ROW_NUMBER()	RANK()	DENSE_RANK()
101	TERENCE	REESE	1,000	7	7	4
123	BENITO	GAROZZO	2,000	4	4	3
257	DOROTHY	HAYDEN	3,000	3	3	2
420	BORIS	SCHAPIRO	2,000	5	4	3
654	OMAR	SHARIF	4,000	2	1	1
666	HUGH	KELSEY	2,000	6	4	3
712	KATHY	WEI	4,000	1	1	1

48

An example that brings all 3 functions together.

RANK : Ability to rank and skip duplicates.

DENSE\_RANK : Ability to rank and NOT skip duplicates.

RANK will skip duplicate ranks,

Skip number 2

Next one gets 3

3 people get 4

so we skip 5 & 6

and Terence gets 7

With Dense Rank


The top 2 get 1

rank is not skipped..

2 nd rank is assigned

all the 4's in rank get 3 in dense rank..

and then the last rank is 4



**WED820G**  
June  
2010

## Thank you for attending!!!

### References

- ❖ DB2 9 for z/OS - Technical Overview  
<http://www.redbooks.ibm.com/abstracts/SG247330.html>
- ❖ DB2 9 for z/OS - Performance Topics  
<http://www.redbooks.ibm.com/abstracts/SG247473.html>
- ❖ Several IBM/IDUG/IOD DB2 9 presentations!

49

Some of the many useful references.