It seems to be a constant battle to ensure that DB2 applications are running at their optimum efficiency and speed, but what are the factors that affect this efficiency. This presentation looks briefly at those aspects of application coding, development and execution that have a bearing on the overall performance.

But what if it's really not the applications fault? How do you determine, before embarking on lengthy analysis, that the problem really IS one you can fix from a DB2 perspective. We'll also look at why it is important to build a performance database that easily allows comparisons between "now" and "then".

The performance database can also prove invaluable in determining trends in performance – perhaps even allowing problems to be highlighted and resolved BEFORE they cause major impacts

- Objective 1:What affects performance - Let's do an A-Z
- Objective 2:What do we have control over
- Objective 3:What do we WISH we had control over
- Objective 4:See - I told you tuning was easy
- Objective 5:But we mustn't forget the Performance Database

Todays Presentation

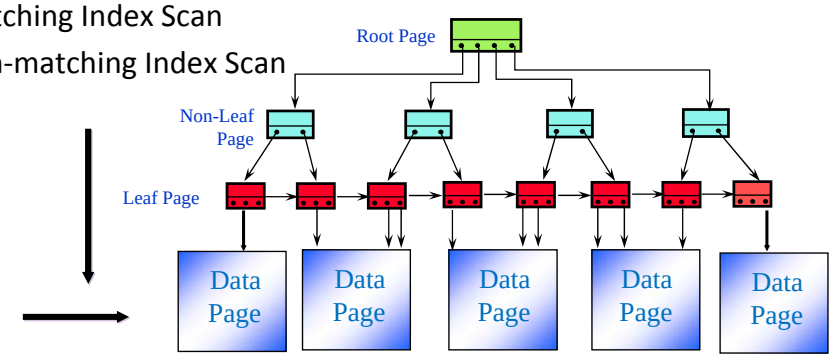
- Application performance challenges
  - What can application developers do to ensure optimum application performance
- What is a “performance database”
  - It sounds very grand, but can be very simple
- Tracking, Truthfulness and Trending
  - Three VERY useful things you can do with a performance database!
- And this presentation has been tuned
  - So it goes VERY fast !

The Challenge

- Applications accessing DB2 will, in general, do one (and only one) of the following at one time:
  - Use CPU
  - Wait for something
    - I/O
    - CPU
    - Locked DB2 Object
    - Latch, drain, dataset operation etc.
- The aim is more (1) and less (2)!

Single Table Access Paths

- Non-segmented Tablespace Scan
- Segmented Tablespace Scan
- Partition Scan
- Matching Index Scan
- Non-matching Index Scan
- Multiple Index Access
- Index Only
- One-Fetch Index

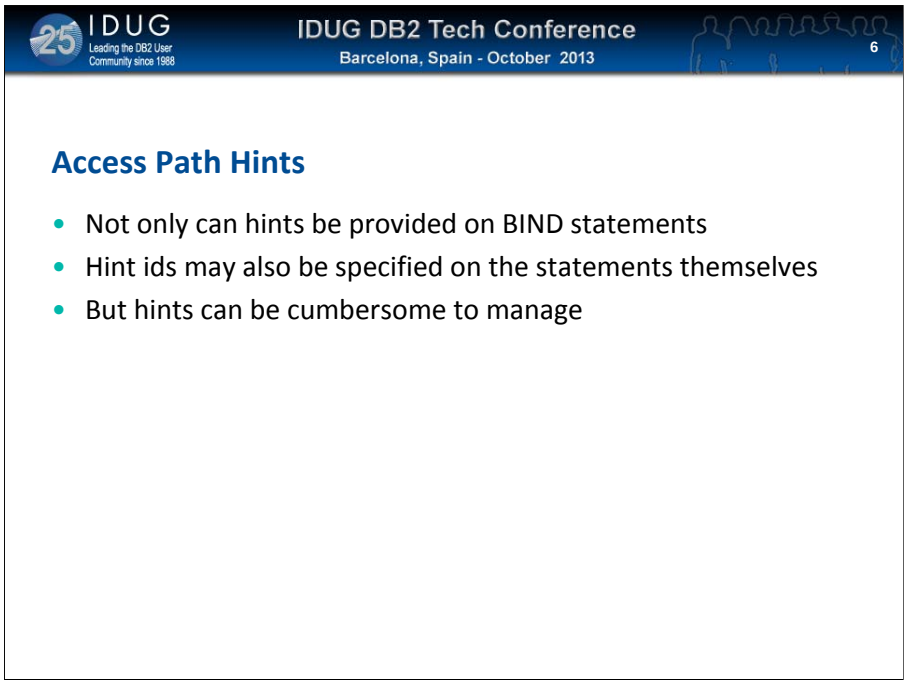


Access Paths are DB2s instructions on how to get the data your application wants

These can range from a Tablespace Scan (look at EVERY row to see if you want it) to a 1-Fetch Index Access (giving you ONE key from an index)

There is a wide range of others, all with their own characteristic performance issues

It is NOT true to say that a “tablespace scan is always bad” – if you want to process more than a small proportion of the rows in a table, it might actually be preferable to scan them all. You will, at least, get the benefits of prefetch



You can provide HINTS at BIND time, or by adding the hint name to the SQL statement itself from v8 onwards

Access Path stability

- Introduced with PK52523
  - AKA Plan Stability
- Allows rebinding to take place whilst retaining old access paths for regression
- Removes the major fear of rebinding
  "If I rebind and get an access path I don't like, there is no way back to where I was"
- Rebind now has "fallback"

Access Path Stability allows a rebind to take place BUT also to save the current access path.

If access path regression has occurred, a further REBIND with the new SWITCH(PREVIOUS/ORIGINAL) option can revert the access path back to what it was (or even right back to it's original)

Rebind now has a fallback option that was sorely missed by many

Access Path stability

- REBIND PACKAGE ….
  PLANMGMT(OFF, BASIC or EXTENDED)
  - Default defined in DSNZPARM
  - Note, not ALL package options can be changed during this type of rebind though
  - But great for a REBIND to get a new access path
  - Also applies to trigger packages

To retain old access paths, it is necessary to rebind with the BASIC or EXTENDED plan management options. The default for this option can also be specified in dsnzparm ensuring that plan management will always be in effect

This access path management is also allowed for rebinding of trigger packages

Access Path stability

- PLANMGMT(OFF)
  - Rebinds package without affecting other package copies
  - Other copies ARE retained

PLANMGMT(OFF) enables a rebind to take place WITHOUT the old access path information being overwritten

PLANMGMT(BASIC) keeps ONE copy of an old access path – the one that was in effect immediately prior to the rebind

PLANMGMT(EXTENDED) keeps ONE copy of an old access path – the one that was in effect immediately prior to the rebind AND keeps the original copy that was in effect the first time plan management was used

Access Path stability

- PLANMGMT(BASIC)
  - Copies current access plan to PREVIOUS
  - Rebinds new package
- PLANMGMT(EXTENDED)
  - Copies current access plan to PREVIOUS
  - IF an ORIGINAL copy does not exist, copies current access plan to ORIGINAL
  - Rebinds new package

Access Path stability

- Now a REBIND can specify SWITCH(ORIGINAL) or SWITCH(PREVIOUS)
  - To revert to an earlier access path selection
  - Remember, though, an EXPLAIN is NOT done at this time
    - Explain was done when the original rebind was done
  - So EXPLAIN for the current access path may NOT be the latest one in the PLAN_TABLE
  - You will need to check bind & explain timestamps

To go with this ability to save old access paths, comes the option to REBIND and switch BACK to one of them

SWITCH(PREVIOUS) switches back to the last saved access path

SWITCH(ORIGINAL) switches to the original saved one

Note though that NO explaining is performed on a SWITCH – so, the explain information for the current access path may NOT be the latest set of rows in the PLAN_TABLE. Only examination of the bind and explain time stamps will help

Bufferpools - A Strategy

- Separating of production DB2 objects across bufferpools can offer several advantages:
  - This Object/Bufferpool separation delivers isolation
  - DB2 performance improves as objects can be tuned based on processing
- The Basic Plan
  - BP0 - Catalog & Directory
  - BP1 - Indexes
  - BP5 - Tablespaces
  - BP7 - DSNDB07(Work Files)
- Expand by separating like-processed objects
  - Random ones
  - Sequential ones

IBM used to say “One bufferpool can be managed better by DB2 than multiple ones can by DBAs”

Whilst this is still largely true, it can be of benefit to put differently ACCESSED objects into different bufferpools and further tune those bufferpools to be optimal for a particular type of access (sequential, random etc)

Clustering

- Clustering (and clusterratio) describes whether the data is held in the same sequence as an index
  - Usually the CLUSTERING index
- Ensures that any sequential access by this key will be efficient

Clustering

- BUT if the data is NEVER accessed by this clustering key
- Why bother keeping the data clustered?
  - And why reorg “because clusterratio is low”?
- DB2 10 provides RTS REORGCLUSTERSENS
  - How often a table is accessed IN THE CLUSTERING KEY SEQUENCE since a reorg
  - If low (or zero) – why bother!
  - Or did you choose the wrong clustering key completely?

Direct Row Access

- ROWID Data Type
- PRIMARY_ACCESSTYPE = 'D'
  - Will use ACCESSTYPE if needed
- Could use index on ROWID
- ROWID also used for LOB support

- BUT Look out for Hash Access in DB2 10
  - This is optimised for random I/O

The Direct Row Access path is a specialty that (so far as I can see) has no real use

If you know a ROWID, then you can fetch a row directly as the ROWID itself contains information related to a RID. The snag is, you can only know a ROWID by reading a row with a ROWID datatype column in it

An neither can you do a BETWEEN for a range of ROWIDs

Dynamic SQL

- Dynamic SQL Caching
  - Reuse and sharing of Prepared SQL
  - Parameter CACHEDYN = Yes/No
  - KEEPDYNAMIC(YES)
  - MAX KEPT DYN STMTS is the maximum number of prepared dynamic statements saved

Dynamic QL used to be BAD (REALLY bad!)

Now, with dynamic SQL caching it is almost as good as static – indeed, in many cases it can be better

There are some tradeoffs though. To really get the best access paths EVERY time, don't use parameter markers but use literals instead

To get the best cache reuse, use parameter markers not literals – see the conflict!

Dynamic Statement Cache - Sizing

- DSNZPARM parameters now allow independent sizing of EDMPOOL items:
- EDMPOOL
  - Size of EDMPOOL itself
  - NOT including the following two areas
- EDMSTMTC
  - How space for dynamic statement caching
  - Default = EDMPOOL
- EDMDBDC
  - How much space for DBD caching
  - Default = EDMPOOL

There are now THREE DIFFERENT EDMPOOL areas

•Dynamic Statement Cache storage

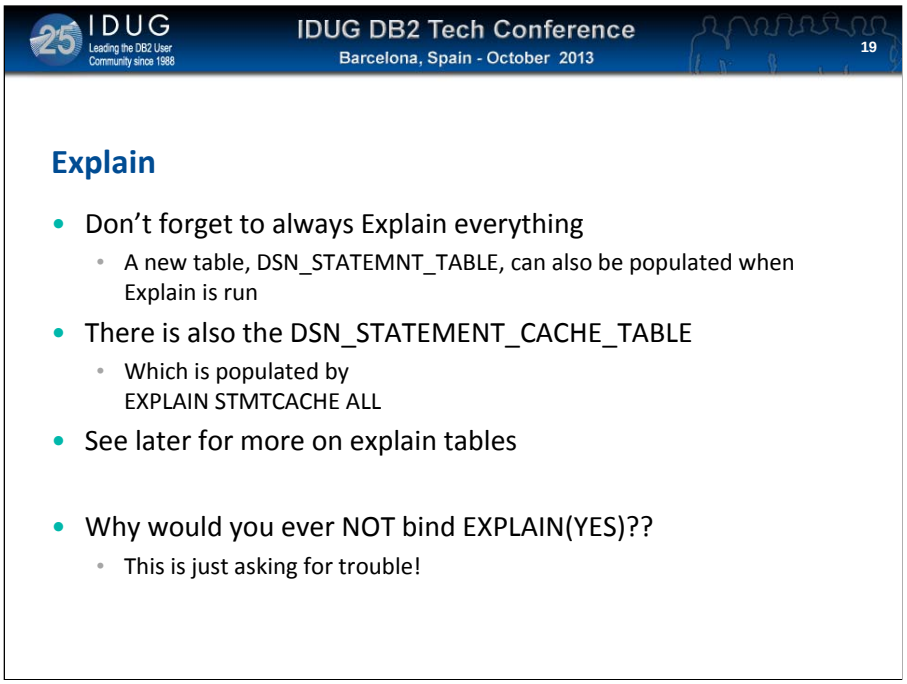
•DBD Cache

•Everything else Cache

So you can size them all independently

Dynamic Statement Cache – Literals

- In DB2 9 (and earlier) SQL with literals caches very badly
  - Each unique literal is a "new" statement
  - Very unlikely to get cache hits on subsequent SQL
- DB2 10 replaces literals with special "&" markers
  - Looks up statement with literals – if not found ..
  - Looks up statement with literals replaced – if not found ..
  - Prepares statement as before
- Still better reuse if using literals
  - But not quite so bad if you don't



Many access path questions would be a LOT easier to answer if only there was a matching EXPLAIN to look at

Make it a standard that EVERYTHING is Explained when it is moved into production

If you have a problem SQL, it is no use running an Explain on it after it starts executing – that does not guarantee an accurate access path analysis (especially not for static SQL)

On v8 it is now possible to Explain the contents of the Dynamic Statement Cache (AND this externalises the SQL statement text as well)

Filter Factors & Host Variables

- Proportion of rows for predicate is true
  - COL1 = Literal → FF = 1/25 or .04
- Filter Factor for Uniform Distribution
  - Positive Value in CARDF in SYSCOLUMNS
- Filter Factors for Non-Uniform Distribution/Column Correlations
  - Columns in SYSCOLDIST & SYSCOLDISTSTATS
  - And Histogram statistics
- Reoptimization at Run Time - REOPT(VARS)
  - Addresses impact of Host Variables and Filter Factors
  - Reoptimization occurs when Cursor is opened
  - Look for big differences in data distribution

Filter Factors are DB2s way of determining whether your predicates will result in a reduction in qualifying rows.

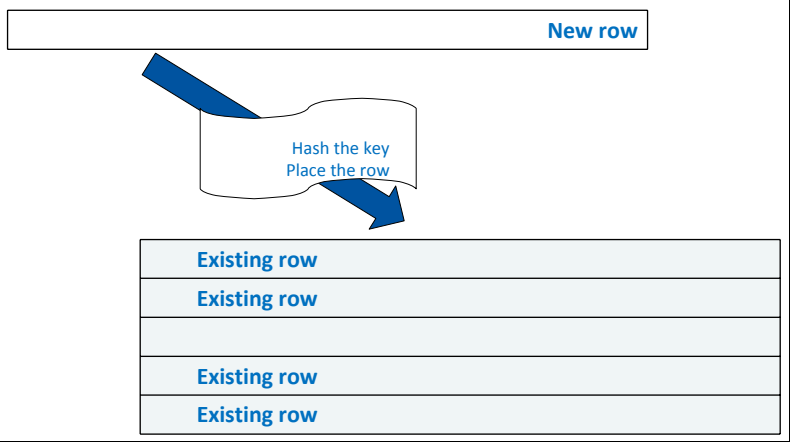
It uses a lot of defaults (in the absence of real data) but you can help DB2 out by collecting distribution stats

By default (for example) WHERE GENDER = “M” gets a filter factor of 1/25 (see slide) whereas it ought to be 1/2!

Use of REOPT(VARS) can allow DB2 to reoptimize statements based on host variable content at run time

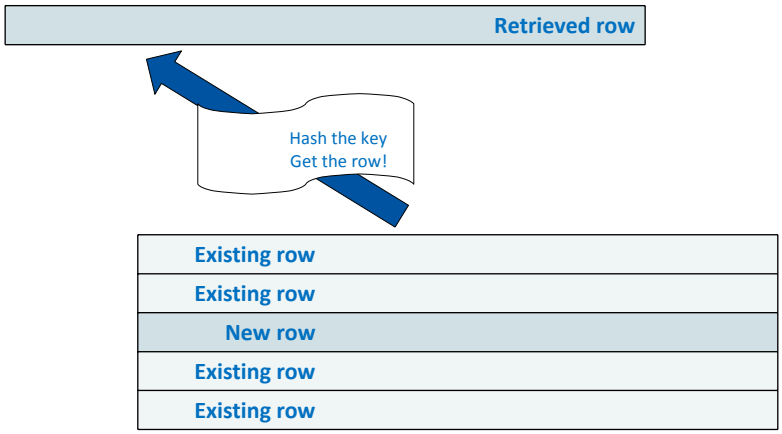
Hash access to data

- Hashed row location



Hash access to data

- Hashed row location



Indexes

- An Index causes 4 - 10 percent overhead on Delete and Insert processing
- Utility Impact of Indexes
- Matching Columns VS Index Screening
  - List Prefetch can use Index Screening

INDEXA

COL1,
COL2,
COL3,
COL4

EXAMPLE # 1

```
SELECT COL5, COL6
     FROM TABLE_1
 WHERE COL1 = :HV
   AND COL2 = :HV
   AND COL3 = :HV
   AND COL4 = :HV
```

EXAMPLE #2

```
SELECT COL5, COL6
     FROM TABLE_1
 WHERE COL1 = :HV
   AND COL3 = :HV
   AND COL4 = :HV
```

Indexes are the only way to really speed up retrieval (assuming you have the right ones, of course!)

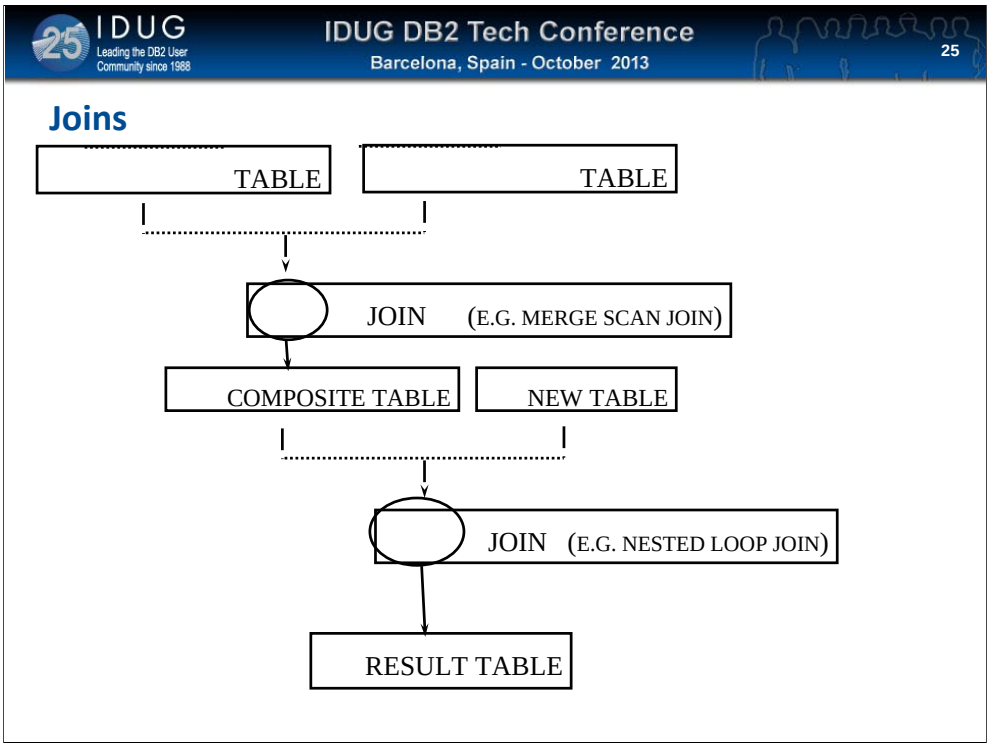
However, Indexes are also a COST overhead for Insert, Delete and any Update that changes key columns

Even if you don't actually MATCH on all columns, it is still possible to get use of additional columns in the index

In the example above, Index Screening will check the values of col3 and col4 even though a value for col2 is not specified (so we get MATCHCOLS = 1)

Indexes

- Foreign keys do not HAVE to have indexes
- But really ought to
  - Without them DELETE CASCADE/RESTRICT may need to do a table space scan
- Acceptable foreign key indexes include those with trailing key columns
  - Provided the foreign key columns are at the beginning of the index key
- Use (or not) of foreign key indexes by RI is NOT externalised by EXPLAIN!



Remember, regardless of how clever the join techniques are, DB2 will only EVER join two tables at a time

The more tables in a join, the more options DB2 has to join them and the longer the optimization process will take (regardless of how long the actual join itself takes after it's been optimized!)

And, of course, there are three different join types as well (NESTED LOOP, MERGE SCAN and HYBRID)

Joins

- Join and Transitive Closure
  - If TAB1.COL1 = TAB2.COL1
    AND TAB2.COL1 = 'ABC'
    - then TAB1.COL1 = 'ABC' as well
    - Don't keep "secrets" from DB2
    - Transitive closure still doesn't happen everywhere
- Beware of "too many" tables
  - V8 "most reasonable" vs "best" access path

Also be aware that access paths (including those involving join) may well change from one release of DB2 to another – here is an example

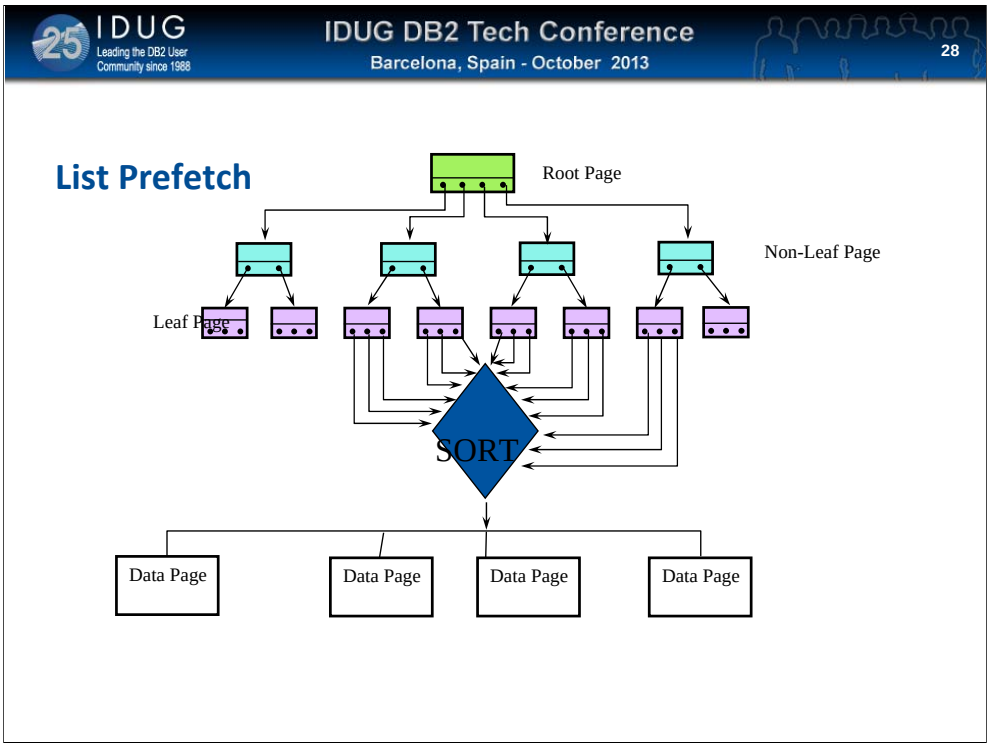
HOWEVER you will only get these benefits (they usually are better than the "old" way) if you rebind

Transitive Closure can also help, but it is also good practice to tell DB2 as much as you can about the data you want and the predicates you are providing

V8 introduces the concept of "most reasonable" access path instead of the "best" access path you were promised in Version 7 (and every version before that)

Last Committed (updaters don't block readers)

- Allows rows to be returned that are currently locked
  - DB2 10 can return data as of Last Commit
- New BIND/PREPARE option
- For a newly inserted row, this is easy
  - If it's still locked, then skip the row
- For a row being deleted, it is also easy
  - If it's still locked, then return the row
- For a row being updated, this is tricky
  - And not supported by DB2 10



List prefectch is a special type of prefetch that does “skip sequential” processing to read non-contiguous pages

The pages read do have to be in order though, so DB2 has to sort the RIDs (row pointers) into page order before the list prefetch can begin

There may also have to be another sort AFTER the list prefetch to get the data in the order you want

It is hoped that these two sorts cost less than the savings in I/O – if not, list prefetch will hurt and not help

It is possible to run out of storage to sort RIDS – if this happens, DB2 falls back to an “alternative” access path (usually a tablespace scan!)

Locking

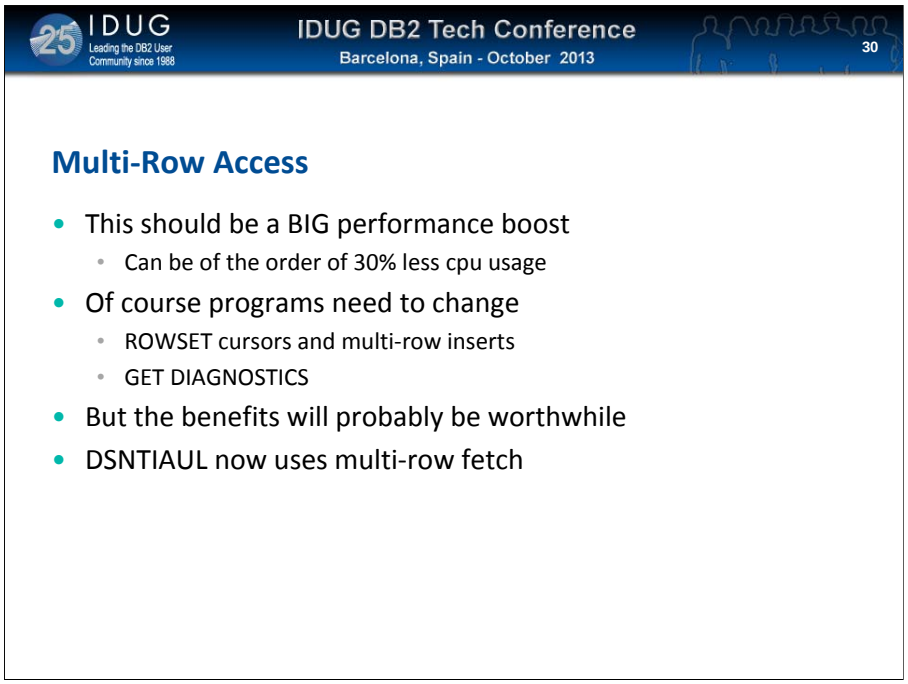
- Watch IRLM Parm of PC=NO
  - Use of ECSA
  - Data Sharing might need to use PC=YES
    - More CPU required
    - ONLY option in v8
- MAXROWS
- PAGESIZE
- LOCK SIZE
- CURRENTDATA YES/NO
- UNCOMMITTED READ (DB2 V8)
- SKIP LOCKED ROWS (DB2 9)
- LAST UPDATED (DB2 10)

ANY locking conflict will cause performance problems

There is not much you can do to speed up the actual locking process itself, but…

Looking at MAXROWS, page sizes, lock sizes and other options can affect how many OTHER processes YOUR locks are causing problems for

You could also consider using UR reads where absolutely 100% accurate data is not required



This is one of the best performance boosters to come from DB2 for some time

The ability to FETCH (and INSERT) using hot language arrays to handle more than one row per SQL call

As the slide says, the savings can be considerable

BUT your programs will have to change to support multi-row cursors

Also, the new GET DIAGNOSTICS error management will need to be used instead of the SQLCA and SQLCODE

OR Predicates

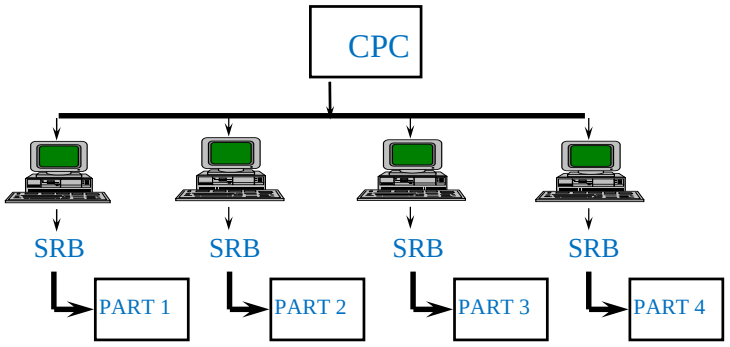
- OR predicates have typically always been non-indexable
  - DB2 couldn't use an index to determine whether something OR something else was true
- DB2 10 is smarter though

OR Predicates

- Consider an index on FAMILY_NAME
- A SQL statement like
- SELECT * FROM my_table
- WHERE FAMILY_NAME = 'GRAINGER'
- OR FAMILY_NAME = 'GRANGER'
- Can now use that index by utilising TWO probes to see if one or the other (or both) is true
- Can also be useful for "pagination restart" queries
  - Where you need to restart a result set after a known point

Parallelism

- PARAMDEG - Upper DEGREE limit
  - Limits Too Many Parallel Tasks



Don’t assume that DEGREE(ANY) is always bad

It’s had some bad history (runaway parallel streams of ludicrously high degrees)

However, for some time there has been a ‘zparm parameter (PARAMDEG) which limits the maximum number of degrees that DB2 will use

PLAN_TABLE

- Understand steps within a SQL statement
- Retrieve rows in proper sequence
- Keep your PLAN_TABLE definitions up to date
  - This becomes MANDATORY from DB2 10
  - DB2 10 also insists on a UNICODE PLAN_TABLE

```
SELECT * FROM PLAN_TABLE
WHERE APPLNAME LIKE 'PROG%'
ORDER BY TIMESTAMP, QUERYNO,QBLOCKNO,
PLANNO, MIXOPSEQ
```

Performing EXPLAIN is all well and good, but…

Did you know, the PLAN_TABLE structure tends to get new columns with every new release of DB2. These columns are optional, but if they are not there then you will obviously be missing some EXPLAIN information

Also, when selecting rows from the PLAN_TABLE, don't forget to use the correct ORDER BY

PLAN_TABLE

- Don't forget all the other EXPLAIN tables
- They contain a lot of VERY valuable information
- As well as PLAN_TABLE, how many others are there?

DSN_FUNCTION_TABLE
DSN_STATEMNT_TABLE
DSN_STATEMENT_CACHE_TABLE
DSN_DETCOST_TABLE
DSN_FILTER_TABLE
DSN_PGRANGE_TABLE
DSN_PGROUP_TABLE
DSN_PREDICAT_TABLE
DSN_PTASK_TABLE
DSN_QUERY_TABLE
DSN_SORTKEY_TABLE
DSN_SORT_TABLE
DSN_STRUCT_TABLE
DSN_VIEWREF_TABLE

14!

They are all documented in the Performance Monitoring and Tuning Guide

Predicates

- Indexable predicates allow the optimizer to consider using an index when available
- STAGE 1 predicates are evaluated at the time the data rows are retrieved (SARGABLE). Performance advantage in using STAGE 1 PREDICATES because this stage eliminates ROWS passed to STAGE 2 via the Data Manager.
- STAGE 2 predicates are evaluated after data retrieval via the relational (NON-SARGABLE) data services which is more expensive than the Data Manager
- Please don't use "STAGE 3" predicates!
  - These are "predicates" that are evaluated in the application and result in non-qualifying rows being discarded after all

The best type of predicate to use is a "stage 1, indexable" predicate, but what is one of those?

Well, the different predicate types are documented in the DB2 manuals, but the actual type of individual predicates can change between releases of DB2. For example, DB2 v8 changes MANY formerly stage 2 predicates to stage 1 – this is a GOOD thing

A so-called "stage 3" predicate is one that is evaluated in the application program and not in the SQL. For every row that fails a stage 3 predicate evaluation, you have wasted all the resources that DB2 expended returning that row to you. Much better to move the stage 3 predicate into the SQL statement if at all possible (even if it is "only" a stage 2 predicate)

REBIND

- Rebind to ensure best possible access path
- Sometimes needed after DB2 maintenance
- Almost always recommended after DB2 version upgrade
  - See Access Path Stability for protection against access path regression

RUNSTATS
Histogram Statistics

- When collecting histogram stats for a group of columns, a number of "intervals" is specified
  - These are called QUANTILES
- RUNSTATS then groups key values together such that each quantile contains the same proportion
  - For 100 quantiles, each contains 1% of the rows
- In the catalog, DB2 stores, for each quantile, the low and high key range
  - As well as the number of unique values

When collecting histogram statistics, a number of required intervals is specified – these are known as "quantiles"

RUNSTATS then groups keys together such that each quantile contains approximately the same number of rows (for example, specifying 100 quantiles would result in 1% of the keys in each one)

Then, RUNSTATS stores (in the catalog) the high and low key for EACH quantile as well as the number of unique keys in each quantile

RUNSTATS
Histogram statistics

- Keyword NUMQUANTILES on RUNSTATS
  - Number of quantiles should be < than number of unique values

Specify NUMQUANTILES on RUNSTATS (the Technical Preview says that "ideally, there should be more quantiles than unique values" but I am not sure how helpful that is for unique data!)

RUNSTATS
Histogram statistics

- From the Technical Preview:
-

QUANTILENO	LOWVALUE	HIGHVALUE	CARDF	FREQUENCYF
1	0	3	4	14%
2	4	15	8	14%
3	18	24	7	12%
4	25	25	1	15%
5	26	26	1	15%
6	27	30	4	16%
7	35	40	6	14%

- Note that there can be gaps in the key ranges
  - No rows exist with values 31-34
- There is a LOT of information here for the optimizer

Here is what a histogram of 7 quantiles might look like

There can be gaps in the key ranges – in this example, there are NO keys from 31-34 inclusive. This is also useful information for the optimizer!

RUNSTATS Histogram statistics

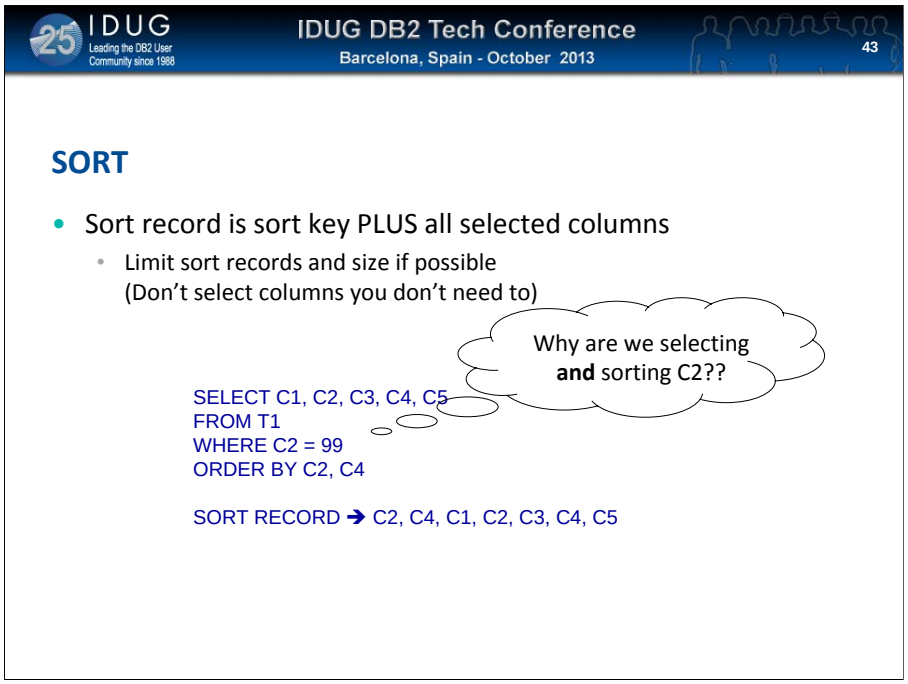
- STATE/ZIPCODE correlation?
- Histogram stats now make DB2 aware of this link

- Also, situations like DATE DECIMAL(8,0)
  - Does NOT have an infinite range from 00000000 to 99999999

- Don't underestimate the value of these histograms
- BUT don't underestimate the cost of collecting them either
  - YMMV (as always)

Skip Locked Data

- Not Uncommitted Read (UR)
  - See later
- Skip Locked Data will skip over any locked data
  - Rows or pages
- Application will not have to wait for data to be unlocked
  - BUT valid data may well be bypassed



Beware of sorting too much. Not just rows, but columns too as the sort key in DSNDB07 is the concatenation of the columns you are sorting by PLUS ALL OF THE COLUMNS YOU ARE SELECTING!

There are a few sort statistics that are worth looking out for

Sort Work Database

- DSNDB07 or Named DB for Data Sharing
- Minimum of 5 or 1/5 Max # of Partitions
- Placement of Datasets
  - Across Channels and Control Units
- Own Bufferpool
  - VPSEQT Set to 95 to 98
  - Watch DWQT and VDWQT
- New ZPARM SPRMOPT
  - Influences Optimizer to choose Sort
- Big change in DB2 9
  - From favouring 4k page sets to favouring 32k ones!

The Sort Work Database (either DSNDB07 or a named database in a datasharing member) has such different processing to all other data that this is why it is often better to have the sort work files in their own bufferpool

Do NOT place multiple sort work tablespaces on the same DASD volume. When DB2 tries to balance I/O across multiple tablespaces, the assumption is that they are all on different devices. One big tablespace on a volume is better than 2 (or more) smaller ones sharing the same volume

Also, set bufferpool thresholds to favour parallel processing

Sort Work Database

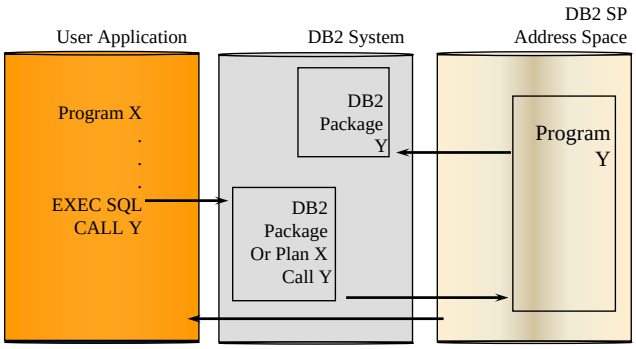
- PK70060
  - Now that work files and DGTTs/static scrollable cursors share temp space
  - Do you allocate a secondary amount or not?
    - You might want DGTTs to extend but not sort work files
  - With this APAR:
    - Work files with SECQTY = 0 will be preferred for sort work data
    - Work files with SECQTY > 0 will be preferred for DGTTs and static scrollable cursors

Sort Work Database

- PM02528
  - Introduces zparm switch to CONTROL behaviour
    - Rather than it being a preference
  - WFDBSEP:
    - YES – Separation IS maintained
      - If YES, the separation is a HARD separation
      - Failures may occur as overflow is not allowed
    - NO – Separation is NOT maintained
      - If NO, the separation is a SOFT separation
      - DB2 will try and keep usage apart BUT overflow IS allowed
  - See II14587 for recommendations

Stored Procedures

- Addresses the "CLIENT/SERVER" Problem
- Statically Bound Package
  - Centrally Located, Benefits Security Issues
  - CLIENT Program Logic Simplified



Stored procedures enable SQL to be kept "safe" on the server, leaving simple calls on the client

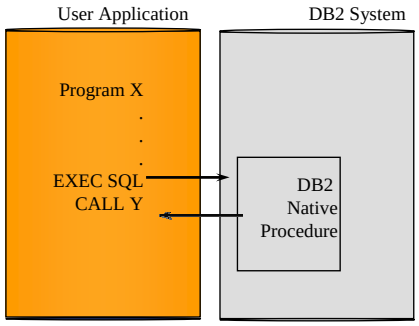
Much less opportunity for underhand activity

Also could be a much better performing option than issuing may SQL calls across a network connection

Note that from v8, all Stored Procedures must run under Workload Manager control

Stored Procedures (Native)

- Addresses the "CLIENT/SERVER" Problem
- Statically Bound Package
  - Centrally Located, Benefits Security Issues
  - CLIENT Program Logic Simplified
- Native procedures fully under DB2s control



Native Stored Procedures are executed in DB2s own address space NOT in the Stored Procedure address space – so no cross-memory movement of data is required

Uncommitted Read

- Isolation Level of UR
- Reading uncommitted data
  - Statistical Analysis
  - Sources Unknown
  - Don't Care
- WITH UR on a specific SQL
- Significant performance improvement

A great way to speed up reading of data – Uncommitted Read reads ALL rows whether they have been committed or not

In other words, it reads "through" locks (and doesn't wait)

HOWEVER

Just because a row is "there" but not committed yet, doesn't necessarily mean it WILL be committed later – it could easily be rolled back. Now your UR read has read a row "that doesn't exist".

UR is then completely unsuitable to read data when you need a definitive answer about that data, but it IS a good choice when all you need is an idea or estomate of the data

Wait Times

- IN DB2 TIME COMPARED TO IN DB2 CPU TIME
  - How much difference ?
  - What are we waiting for ?
- Service Task Wait Time Broken Down to:
  - Open / Close
  - Dataset Create/Extend/Delete
  - SYSLGRNX Downlevel Detection
  - Phase 2 Commit/Abort
- ALWAYS have accounting class 3 turned on
  - Gives much more wait time information

Wait times can be a good indication of something being wrong. Also, reducing or eliminating wait times will (of course) improve performance

With accounting Class 3 enabled, wait times are broken down into useful levels of granularity, with only Class 1 and 2, there will be a lot of “other wait time” which is less than helpful

Performance Database

- Often we hear talk of a “Performance Database”
  - But what is it?
- It’s a historical collection of performance data
- YOU choose what to store
- YOU choose how long to store it for
- YOU choose where to store it

Performance Database

- My first performance database was simply a flat file containing
  - CICS transaction ID
  - IN DB2 TIME
  - IN DB2 CPU TIME
  - TOTAL ELAPSED TIME
- For each DAY
- And I kept 3 months of data

Performance Database

- You could choose to keep much more data
- And for much longer

- A flat file is pretty simple, but also not very easy to search
- And useless for complex analysis

- Perhaps your performance database should BE a database

Performance Database

- Where you source the data from is also your choice
- Most DB2 monitors will externalise anything you ask for
- Or you could just use DB2s Accounting or Performance trace data
- Or perhaps you have a performance analysis tool for DB2 that has its own Performance Database
  - If so, the definitions should be well documented

Tracking, Truthfulness and Trending

- Tracking is performance is easy with a performance database
- It's IMPOSSIBLE without one
- Say a transaction is running slower than usual
  - Your tracking data may show it's NOT a DB2 problem (IN DB2 time is the same as usual)
  - Or it IS a DB2 problem, but IN DB2 CPU is the same as usual…..

Tracking, Truthfulness and Trending

- Has anyone ever reported something is “running slower than usual”?
- How do you know?
- Do people always tell the truth?
- Do you waste much time chasing problems that don’t exist
  - Or are not your responsibility?
- First stop – the Performance Database
- What do TODAYS figures look like compared with earlier?

Tracking, Truthfulness and Trending

- Now that you have performance data tracked over time
- It's easy to look for trends
  - Is performance staying the same?
  - Is it degrading?
    - How much by and how fast?
  - Are we running out of a critical resource?
- You can see (and fix) problems BEFORE they arise
  - And before the phone rings

Any Questions?

INTERNATIONAL DB2 USERS GROUP
Independent • Not-for-Profit • User Run
Founded 1988

Phil Grainger
BMC Software
phil_grainger@bmc.com

F02
Application Performance
Tuning, Trending and Testing

INTERNATIONAL DB2 USERS GROUP
Independent • Not-for-Profit • User Run
Founded 1988