DanL
Database Consulting

Db2 For z/OS Ultra High Performance and Tuning

Daniel L Luksetich

DanL Database Consulting

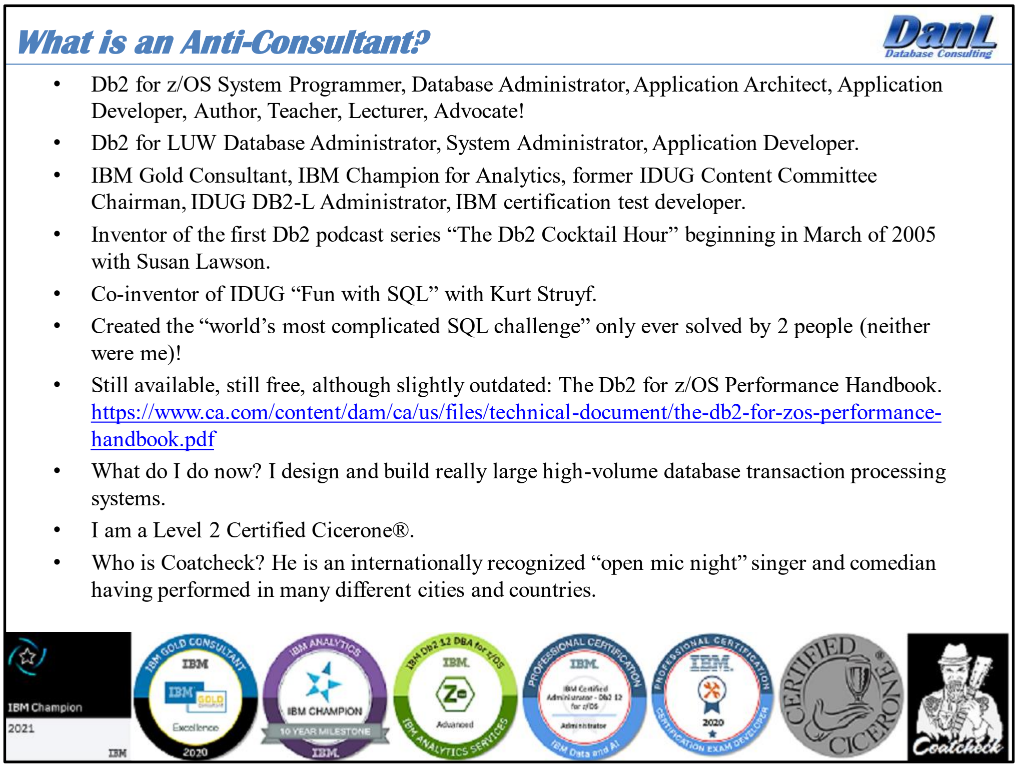
danl@db2expert.com

Dan Luksetich is a Db2 DBA consultant. He works as a DBA, application architect, presenter, author, and teacher. Dan has been in the information technology business for over 36 years, and has worked with Db2 for over 31 years. He has been an application programmer, Db2 system programmer, Db2 DBA, and Db2 application architect. His experience includes major implementations on z/OS, AIX, i Series, Windows, and Linux environments. His industry experience includes retail, banking, fraud detection, analytics, government, and utility. He specializes in highly available, high-volume transaction processing against very large database systems.

Dan's experience includes, but is not limited to:

- Application design and architecture
- Business analytics
- SQL consulting and education
- SQL coding and application programming
- Database administration
- Complex SQL
- SQL tuning
- Db2 performance audits
- Replication
- Disaster recovery
- Stored procedures, user-defined functions, and triggers
- Db2 REST services

Dan likes beer and is a Certified Cicerone!


DanL
Database Consulting
IBM Champion
2021
IBM Gold Consultant
Excellence 2020
IBM Analytics
IBM Champion
10 Year Milestone
Db2 12 DBA for z/OS
Advanced
IBM Analytics Services
Professional Certification
IBM Certified Administrator - Db2 12 for z/OS
IBM Data and AI
Professional Certification
2020
Certification Exam Developer
Certified Cicerone®
Coatcheck

What is an Anti-Consultant?

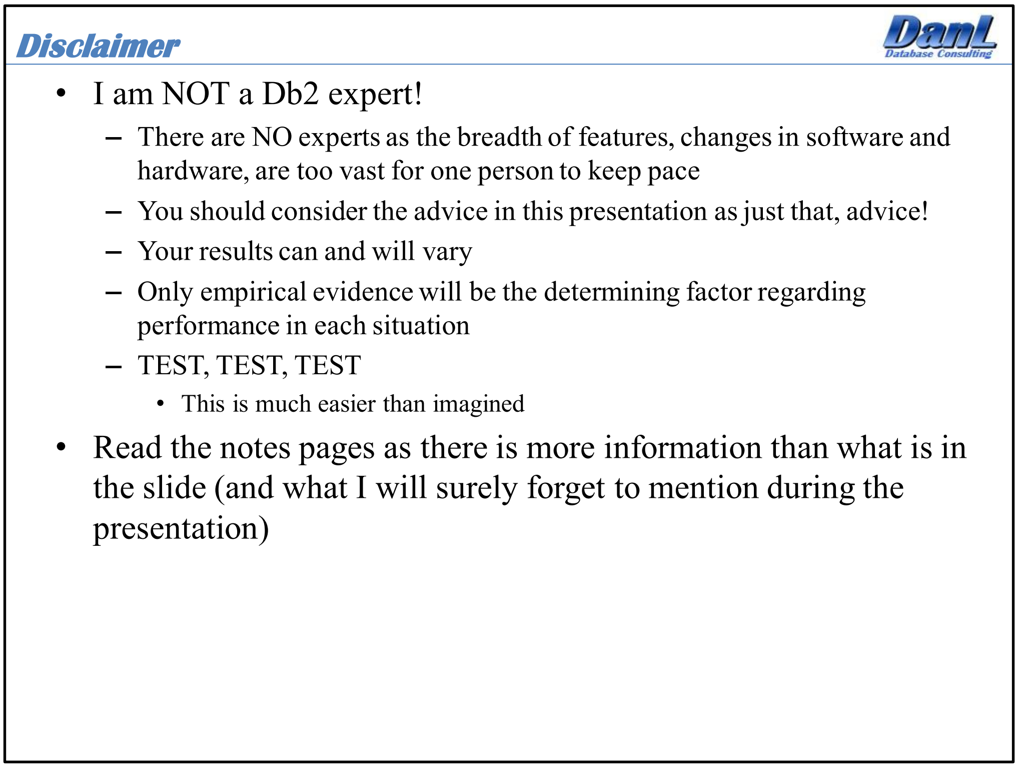
- Db2 for z/OS System Programmer, Database Administrator, Application Architect, Application Developer, Author, Teacher, Lecturer, Advocate!
- Db2 for LUW Database Administrator, System Administrator, Application Developer.
- IBM Gold Consultant, IBM Champion for Analytics, former IDUG Content Committee Chairman, IDUG DB2-L Administrator, IBM certification test developer.
- Inventor of the first Db2 podcast series "The Db2 Cocktail Hour" beginning in March of 2005 with Susan Lawson.
- Co-inventor of IDUG "Fun with SQL" with Kurt Struyf.
- Created the "world's most complicated SQL challenge" only ever solved by 2 people (neither were me)!
- Still available, still free, although slightly outdated: The Db2 for z/OS Performance Handbook. https://www.ca.com/content/dam/ca/us/files/technical-document/the-db2-for-zos-performance-handbook.pdf
- What do I do now? I design and build really large high-volume database transaction processing systems.
- I am a Level 2 Certified Cicerone®.
- Who is Coatcheck? He is an internationally recognized "open mic night" singer and comedian having performed in many different cities and countries.

When I has a child, I wanted to be a meteorologist. When I was a teenager, I wanted to be a rock star. When I was in college, I wanted to be a computer programmer.

Now, I want to be a brewer…or a short order cook…or maybe a BBQ pitmaster!

The word Cicerone (sis-uh-rohn) designates a trained professional, working in the hospitality and alcoholic beverage industry, who specializes in the service and knowledge of beer. The knowledge required for certification includes an understanding of styles, brewing, ingredients, history of beer and brewing, glassware, beer service, draught systems, beer tasting, and food pairings. To claim the title of Cicerone, one must earn the trademarked title of Certified Cicerone® or hold a higher certification. Those with a basic level of expertise gain recognition by earning the first-level title Certified Beer Server. Only those who have passed the requisite test of knowledge and tasting skill can call themselves a Cicerone.

And just for the record an anti-consultant is someone that primarily wants to solve problems, with money only being an added bonus to the effort. An anti-consultant's main objective is to solve your problems, and give you the knowledge such that you never need his or her service again. To summarize, an anti-consultant is always trying to put themselves out of work. I've failed in that aspect for the last 25 years.



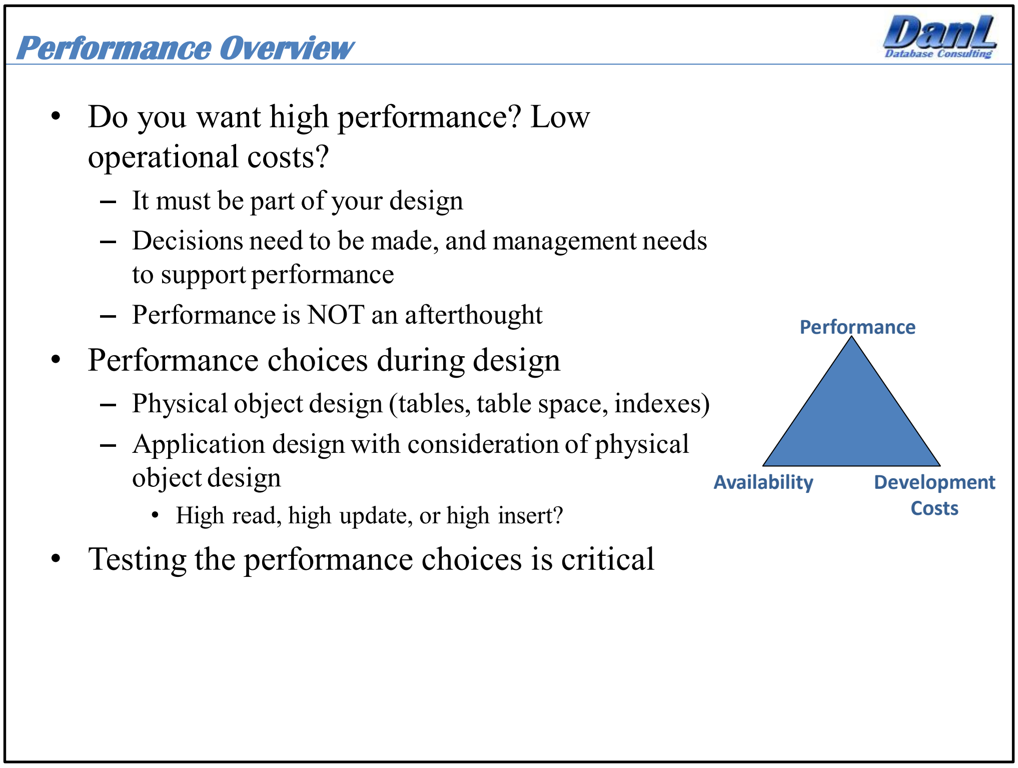
Disclaimer

- I am NOT a Db2 expert!
  - There are NO experts as the breadth of features, changes in software and hardware, are too vast for one person to keep pace
  - You should consider the advice in this presentation as just that, advice!
  - Your results can and will vary
  - Only empirical evidence will be the determining factor regarding performance in each situation
  - TEST, TEST, TEST
    - This is much easier than imagined
- Read the notes pages as there is more information than what is in the slide (and what I will surely forget to mention during the presentation)

I worked for 10 years for an electric company in Chicago. For 5 years I was an Assembler and COBOL programmer, and for another 5 years I was a Db2 System Administration and Database Administrator with a little IMS thrown in. I did a significant amount of tuning, but could not convince management to do more of it, until…

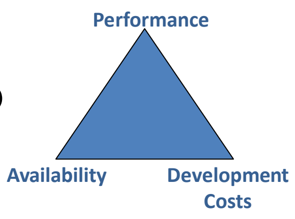
“Big Name Consulting” was brought in to “modernize” our operations. They assigned a “Db2 expert consultant” to work with me on my long requested tuning project. I was introduced and the first thing the kid said to me was “what is Db2?” I worked with him teaching him as I tuned a database and application, and after success he went away. Then my boss told me that “Big Name Consulting” was going to bring in another “Db2 expert consultant” to work with me on a second tuning project…

I went to my desk and typed a letter of resignation. Thus began my career as an anti-consultant.



Performance Overview

- Do you want high performance? Low operational costs?
  - It must be part of your design
  - Decisions need to be made, and management needs to support performance
  - Performance is NOT an afterthought
- Performance choices during design
  - Physical object design (tables, table space, indexes)
  - Application design with consideration of physical object design
    - High read, high update, or high insert?
- Testing the performance choices is critical



There are a myriad of tuning possibilities when it comes to the database. Before jumping onto any of the touted performance features of Db2 it is critical to first understand the problem that needs to be solved. If you want to partition a table, are you actually hoping that the partitioning results in a performance improvement or is it for data management, availability or flexibility? Clustering of data is critical, but clustering has to reflect potential sequential access for single table, and especially multi-table access. Generic table keys are OK, but clustering is meaningless unless parent keys are used for clustering child table data when processing follows a key-based sequential pattern or when multiple tables are being joined. Natural keys may be preferred, unless compound keys become exceedingly large. There are a number of choices of page sizes, and this can be especially important for indexes where page splitting may be a concern. Tables can also be designed specifically for high volume inserts with such features as “append only” and “member cluster”.

There is no reason to sit in meeting rooms for hours or days trying to come up with a database design that you “think” will perform properly. I personally let the database itself influence the design. This involves setting up test database designs and test conditions that will mock the predicted application and database design. This usually involves generated data and statements that are run through SPUFI, REXX programs, db2batch, or shell scripts, all while performance information is gathered from monitors and compared. Sound complicated? It really isn’t, and most proof-of-concept tests I’ve been involved with have not lasted more than a week or two. The effort is minimal, but the rewards can be dramatic.

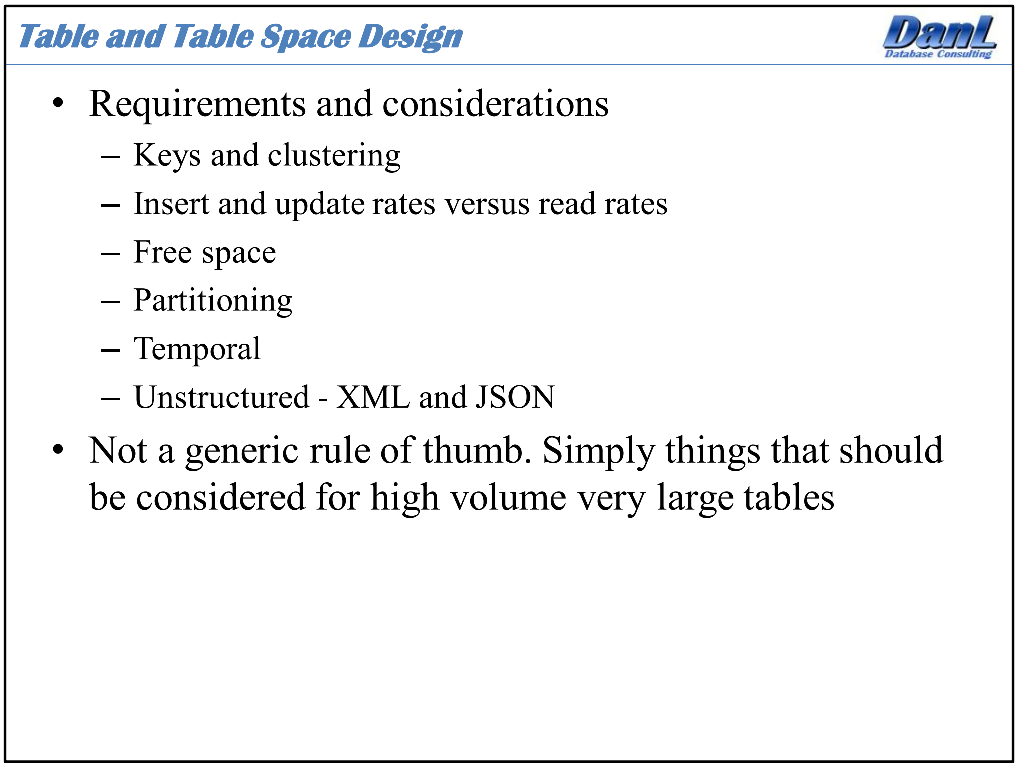


Table and Table Space Design

DanL Database Consulting

- Requirements and considerations
  - Keys and clustering
  - Insert and update rates versus read rates
  - Free space
  - Partitioning
  - Temporal
  - Unstructured - XML and JSON
- Not a generic rule of thumb. Simply things that should be considered for high volume very large tables

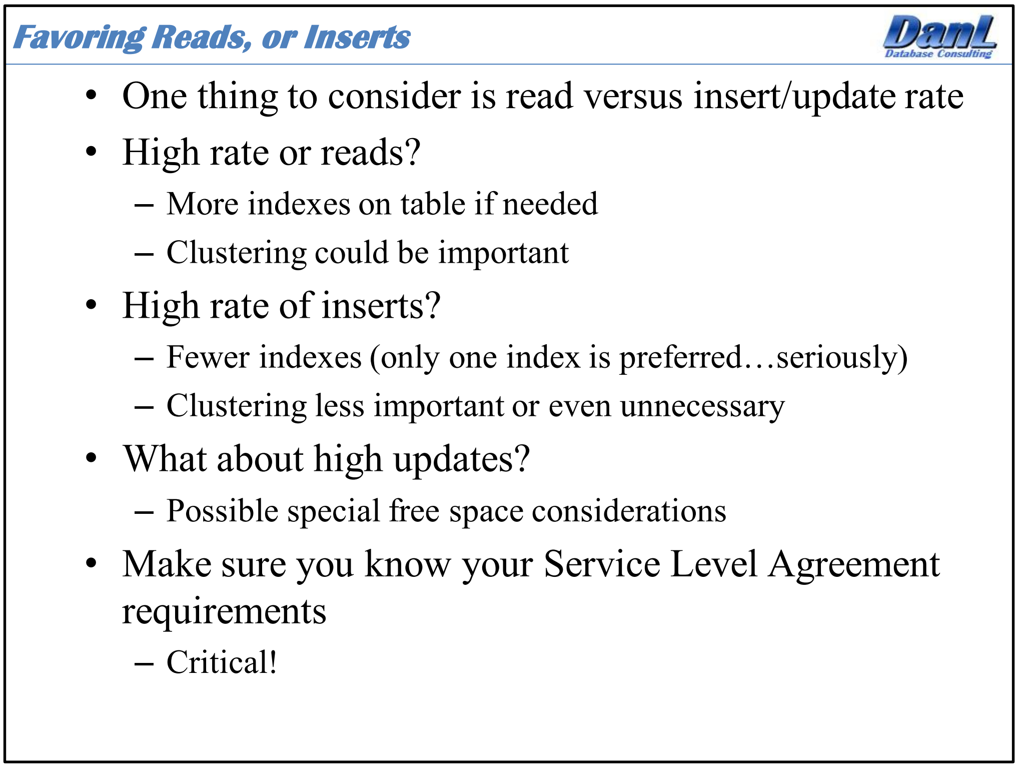
There are two basic rules to table space design:

1) Just implement the design envisioned by the programmer, application architect, or logical data modeler.
2) Figure out how the table will be used, and frequency of use, and implement a design suitable for the application.

In the first case the design can be implemented quickly and all may seem well until production implementation, or during stress testing. In those cases you may find yourself then “fire fighting” in an attempt to isolate the performance issue, and repair it if possible. You may win, or you may lose. If you win you have the benefit of being a “hero”.

In the later case there may be a delay in the implementation of the design (I would argue just a matter of days in most cases), but you can quickly determine, given the correct information, the proper design for the table in question. Of course, you will never be a “hero”.

“If you do things right, people won’t be sure you’ve done anything at all” – quote from Futurama, “Godfellas” S3E20



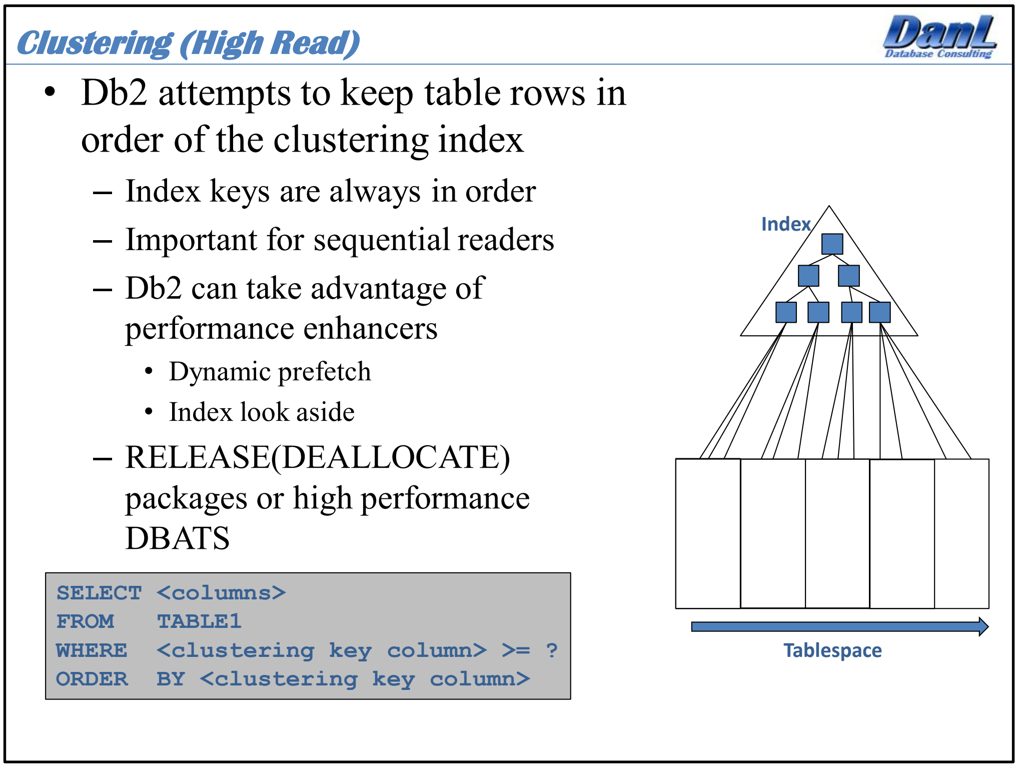
Favoring Reads, or Inserts

- One thing to consider is read versus insert/update rate
- High rate or reads?
  - More indexes on table if needed
  - Clustering could be important
- High rate of inserts?
  - Fewer indexes (only one index is preferred…seriously)
  - Clustering less important or even unnecessary
- What about high updates?
  - Possible special free space considerations
- Make sure you know your Service Level Agreement requirements
  - Critical!

When designing a table for a high performance application there are many things to consider. However, a simple question should be answered when designing a table or set of tables; Is this a high insert table or a high read table? To put it another way are you designing for fast insert or optimal read performance? Once this is answered then the design path can be chosen for the table in question. You must get confirmation from the development team and management. If the boss says she wants the fast insert option along with the fast read option make sure she understands that it is only a dream. She must favor one or the other. Get a service level agreement for performance.

I once asked a manager how fast he wanted the application to run. His response "I want it to be as fast as possible". My response was "Oh great that's easy, buy an infinite number of machines!" After the initial shock he then gave me a real number!

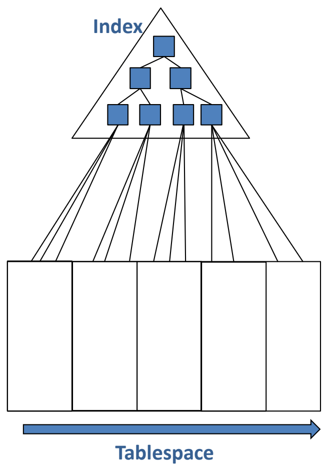
e.g. 90% of all transactions with a response time under 500ms.



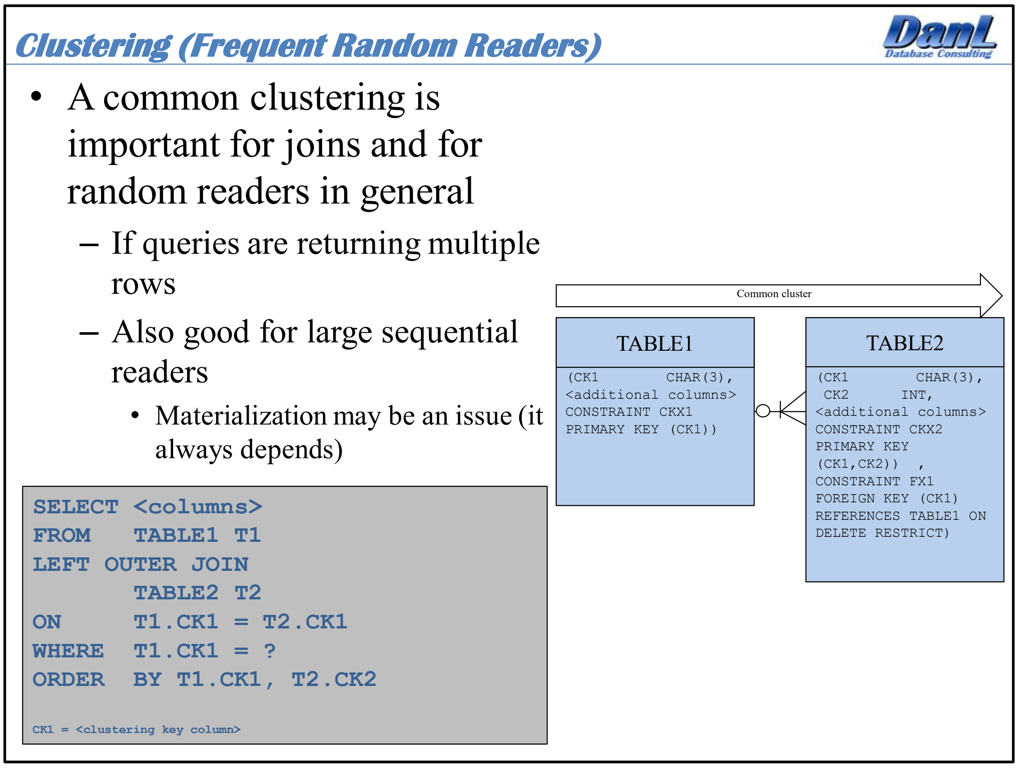
Clustering (High Read)

- Db2 attempts to keep table rows in order of the clustering index
  - Index keys are always in order
  - Important for sequential readers
  - Db2 can take advantage of performance enhancers
    - Dynamic prefetch
    - Index look aside
  - RELEASE(DEALLOCATE) packages or high performance DBATS

```
SELECT <columns>
FROM   TABLE1
WHERE  <clustering key column> >= ?
ORDER  BY <clustering key column>
```



Db2 can achieve incredibly high read rates in support of batch processing when a sequential reader (usually the driving cursor for a process) is reading from a table in the order of the clustering key and the table is reasonably well organized. Depending upon insert and update activity and available free space in the table space, REORGs may be required at certain intervals in order to maintain this high efficiency. The cursor fetching the data should use the clustering key in the predicate, and in the ORDER BY clause, to ensure the best performance.



Clustering (Frequent Random Readers)

- A common clustering is important for joins and for random readers in general
  - If queries are returning multiple rows
  - Also good for large sequential readers
    - Materialization may be an issue (it always depends)

```
SELECT <columns>
FROM   TABLE1 T1
LEFT OUTER JOIN
       TABLE2 T2
ON     T1.CK1 = T2.CK1
WHERE  T1.CK1 = ?
ORDER  BY T1.CK1, T2.CK2

CK1 = <clustering key column>
```

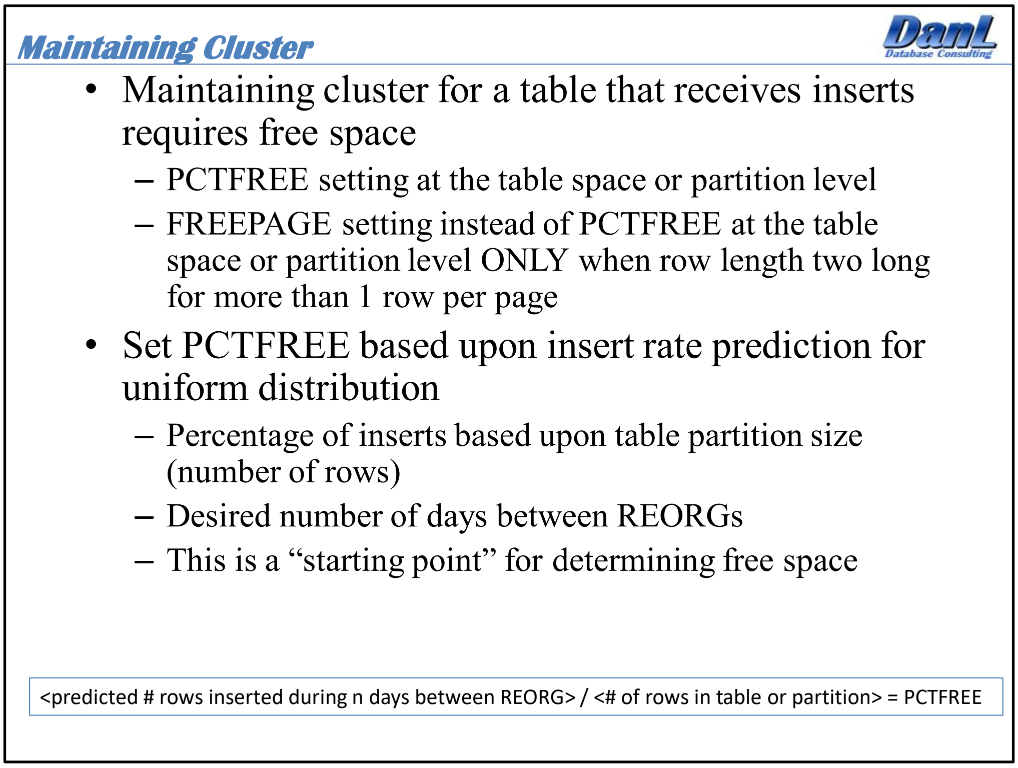
Common cluster
TABLE1
TABLE2

```
(CK1         CHAR(3),
<additional columns>
CONSTRAINT CKX1
PRIMARY KEY (CK1))
```

```
(CK1         CHAR(3),
 CK2       INT,
<additional columns>
CONSTRAINT CKX2
PRIMARY KEY
(CK1,CK2))  ,
CONSTRAINT FX1
FOREIGN KEY (CK1)
REFERENCES TABLE1 ON
DELETE RESTRICT)
```

A common clustering between two related tables can be beneficial for random readers as well as most sequential readers. A good example is a table relationship between a parent table and child table that is identifying. Identifying relationships exist when the primary key of the parent entity is included in the primary key of the child entity. On the other hand, a non-identifying relationship exists when the primary key of the parent entity is included in the child entity but not as part of the child entity's primary key.

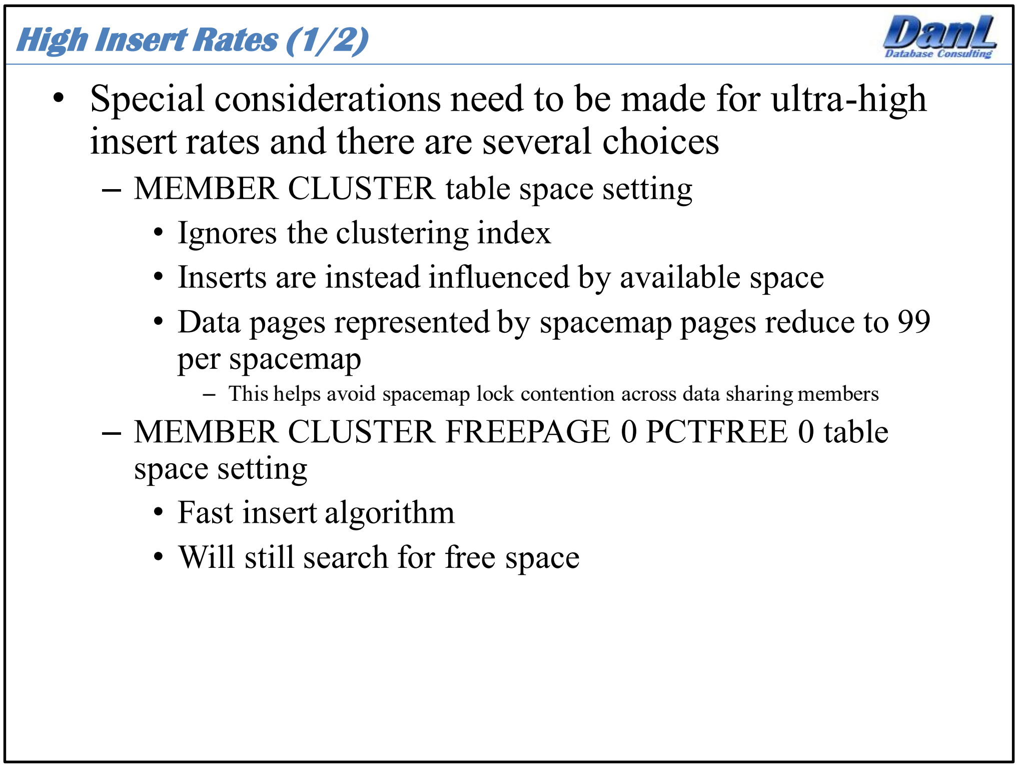
When traversing the index of the inner table (TABLE2 in this example) via the primary key index, key values for multiple rows will be co-located. If TABLE2 is well organized then all of the rows related to TABLE1 will be clustered together. This will minimize the number of random read operations on behalf of the query. Certain things, such as the bind parameter RELEASE(DEALLOCATE), including for high performance DBATS, can also enable the built-in performance enhancers sequential detection and index look aside.


Maintaining Cluster

- Maintaining cluster for a table that receives inserts requires free space
  - PCTFREE setting at the table space or partition level
  - FREEPAGE setting instead of PCTFREE at the table space or partition level ONLY when row length two long for more than 1 row per page
- Set PCTFREE based upon insert rate prediction for uniform distribution
  - Percentage of inserts based upon table partition size (number of rows)
  - Desired number of days between REORGs
  - This is a "starting point" for determining free space

<predicted # rows inserted during n days between REORG> / <# of rows in table or partition> = PCTFREE

If clustered data is desired or required to support sequential processing of table data, and the table receives frequent inserts then proper free space is required to maintain a certain level of cluster. Assuming an even distribution of data in the table space relative to the clustering order then a simple formula can be used to calculate an appropriate PCTFREE setting for the table space and/or table space partition. By using the insert rate and current table size in number of rows a simple formula can be used to determine the PCTFREE setting in order to accommodate the new data in between REORGs. If the distribution of data is skewed then adjustments will have to be made depending upon that skew.

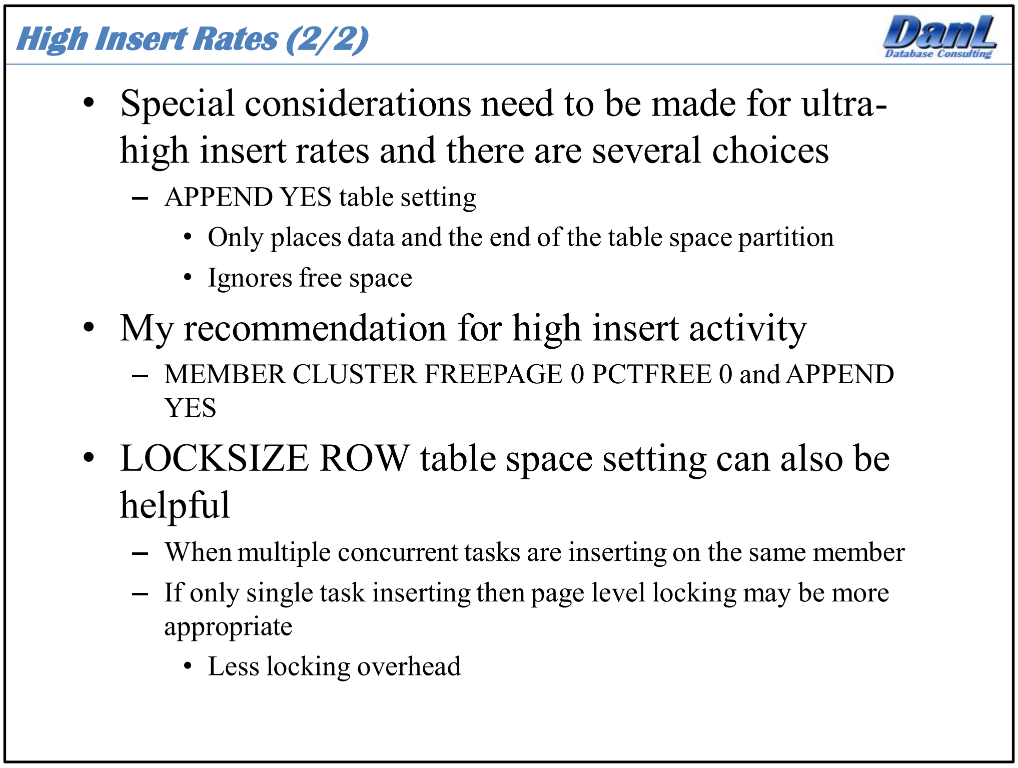


High Insert Rates (1/2)

- Special considerations need to be made for ultra-high insert rates and there are several choices
  - MEMBER CLUSTER table space setting
    - Ignores the clustering index
    - Inserts are instead influenced by available space
    - Data pages represented by spacemap pages reduce to 99 per spacemap
      - This helps avoid spacemap lock contention across data sharing members
  - MEMBER CLUSTER FREEPAGE 0 PCTFREE 0 table space setting
    - Fast insert algorithm
    - Will still search for free space

The MEMBER CLUSTER table space setting is specifically designed for high performance and availability table spaces that are subject to very high insert activity. Setting MEMBER CLUSTER together with PCTFREE 0 and FREEPAGE 0 for a table space does two things: it establishes an alternate insert algorithm by placing the majority of inserts to the end of the table space or target table space partition; and it also sets one space map page for every 99 data pages and reserves the space map pages exclusively for every data sharing member acting against the table space or table space partition. This space map feature can significantly improve the performance and concurrency of high insert applications that are distributed across data sharing members because the unshared space map pages are not a source of contention in the coupling facility. Setting MEMBER CLUSTER, like APPEND ON at the table level, will ignore clustering and place inserts at the end. However, unlike APPEND ON, it will be more aggressive in searching for available free space before extending a table space.

Since these table and table space settings do not respect table clustering table space REORGs may be needed to reinstate the table cluster if readers rely on clustering.



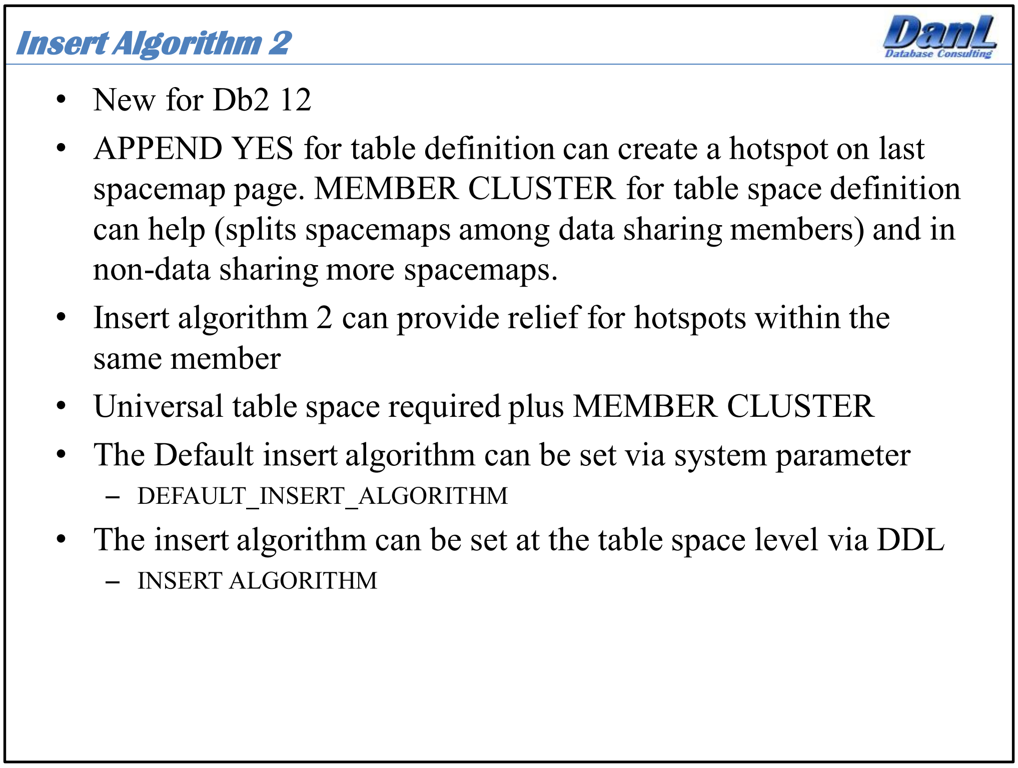
High Insert Rates (2/2)

- Special considerations need to be made for ultra-high insert rates and there are several choices
  - APPEND YES table setting
    - Only places data and the end of the table space partition
    - Ignores free space
- My recommendation for high insert activity
  - MEMBER CLUSTER FREEPAGE 0 PCTFREE 0 and APPEND YES
- LOCKSIZE ROW table space setting can also be helpful
  - When multiple concurrent tasks are inserting on the same member
  - If only single task inserting then page level locking may be more appropriate
    - Less locking overhead

The APPEND table setting affects the behavior of inserts against a particular table. By default the setting is APPEND NO, and any rows inserted into the table are based upon the clustering index (except if MEMBER CLUSTER is set for the table space). Setting APPEND YES tells Db2 that any inserts into the table will ignore clustering and free space completely and place the new data at the end of the table. This is definitely a setting to consider for a high performance insert configuration, and can be an option to set alone or use in conjunction with the MEMBER CLUSTER table space setting. Keep in mind that by setting APPEND YES you are telling DB2 to only put inserted rows at the end of the table and ignore any space that may have been acquired by deletes. Therefore, the table space may need frequent REORGs if clustering is important for efficient query processing against the table. This also works a bit differently than MEMBER CLUSTER.

The recommendation for high insert rate tables is to define the table space with MEMBER CLUSTER PCTFREE 0 FREEPAGE 0 and the table with APPEND YES. This will establish a fast insert algorithm, and if there are deletes and updates it will not reclaim free space in the table space. In addition, row level locking can help if there are multiple concurrent processes inserting. If only a single process will be inserting data then perhaps a page level locking strategy is more beneficial.

Keeping the number of indexes to a minimum will also significantly help insert performance! More on this in upcoming slides.



Insert Algorithm 2

- New for Db2 12
- APPEND YES for table definition can create a hotspot on last spacemap page. MEMBER CLUSTER for table space definition can help (splits spacemaps among data sharing members) and in non-data sharing more spacemaps.
- Insert algorithm 2 can provide relief for hotspots within the same member
- Universal table space required plus MEMBER CLUSTER
- The Default insert algorithm can be set via system parameter
  - DEFAULT_INSERT_ALGORITHM
- The insert algorithm can be set at the table space level via DDL
  - INSERT ALGORITHM

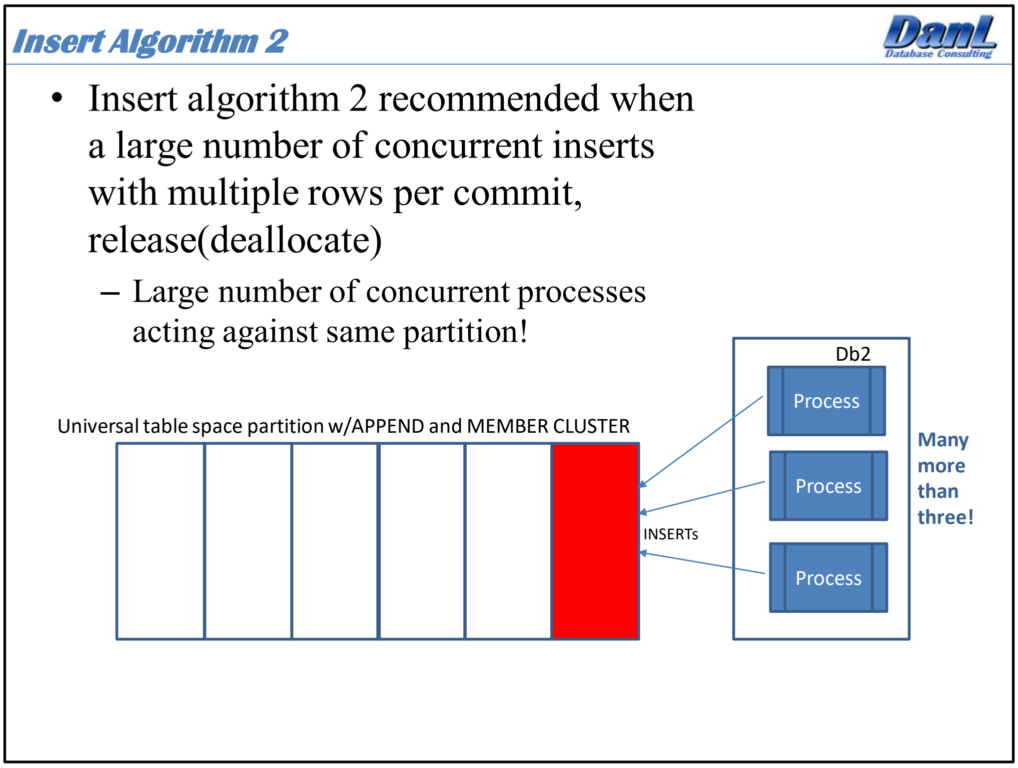
Insert algorithm 2 was introduced in Db2 12 for z/OS and is available once new function is enabled (FL500).

It can be set as the system default, but I don't recommended choosing IA2 as the overall default. This new algorithm is specifically designed for assistance where there are a large number of concurrent processes across a single data sharing member, or multiple members where all of the inserts are going to the end of a table space partition. This would be in situations where clustering has been turned off for the table space via the MEMBER CLUSTER table space option. These are situations where there may be an ascending key (via a key that is a sequence value or ascending UUID primary key), but no actions have yet been taken to improve a perceived performance problem. That is, a performance problem has been detected in that insert performance is slower than expected, and there is a large amount of contention:

High latch times
High amount of logging

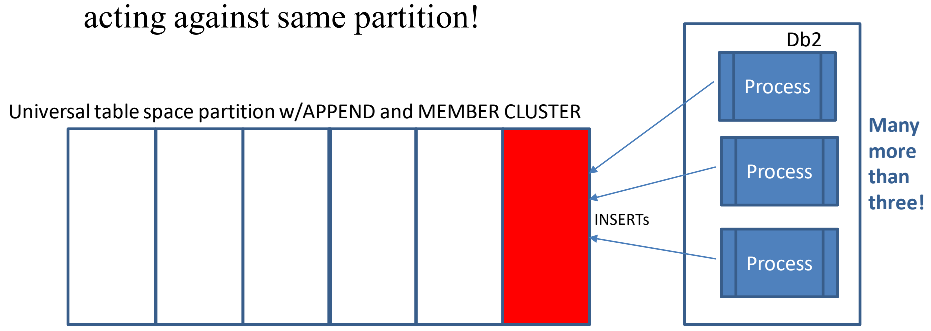
It takes a large amount of monitoring to determine if this is a factor affecting performance.

If, during design, there is the ability to test high volume concurrent insert processing then a head to head comparison could be performed using the two different insert algorithms.



Insert Algorithm 2

- Insert algorithm 2 recommended when a large number of concurrent inserts with multiple rows per commit, release(deallocate)
  - Large number of concurrent processes acting against same partition!

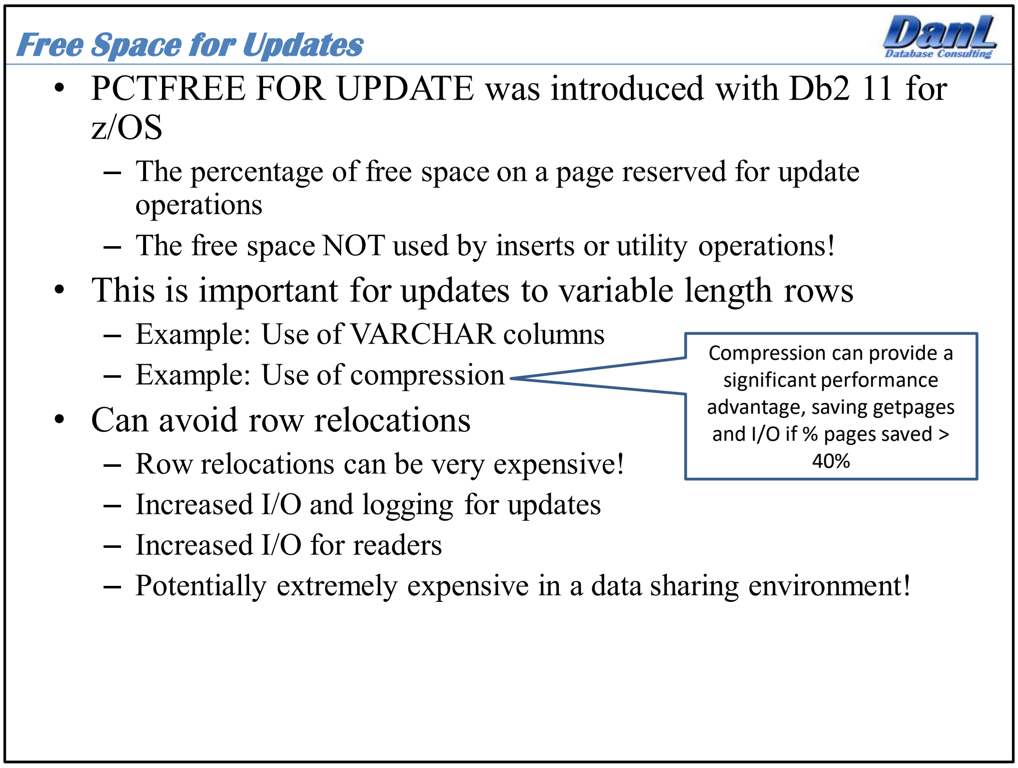


Locksize row + proper page size + member cluster are also important in this design as well!

This slide shows three concurrent processes, but we are really talking about many processes inserting millions of rows concurrently across one or multiple data sharing members.

IBM recommendation:

To determine the group buffer pool size when insert algorithm 2 is used for concurrent INSERT operations from multiple data sharing members, allocate a minimum of 2000 * *n* directory and data entries (where *n* is the number of members).



Free Space for Updates

- PCTFREE FOR UPDATE was introduced with Db2 11 for z/OS
  - The percentage of free space on a page reserved for update operations
  - The free space NOT used by inserts or utility operations!
- This is important for updates to variable length rows
  - Example: Use of VARCHAR columns
  - Example: Use of compression
- Can avoid row relocations
  - Row relocations can be very expensive!
  - Increased I/O and logging for updates
  - Increased I/O for readers
  - Potentially extremely expensive in a data sharing environment!

Compression can provide a significant performance advantage, saving getpages and I/O if % pages saved > 40%

PCTFREE FOR UPDATE was added in Db2 11 for z/OS in order to provide relief for high volume update applications in which the data stored in a Db2 table is variable length and/or compressed. In these situations an update to a row in the table can result in the length of the row increasing. In this situation the row may not fit in the spot in the table from which it was retrieved. Db2 can attempt to compact the data on the page, spontaneously relocating rows of data on the same data page in order to reclaim space in gaps between rows. If Db2 cannot fit the increased row into the data page from which it came, to will then look for another page with free space to place the updated row.

Both page compaction and row relation can result in excessive synchronous I/O and synchronous log writes. This is especially true in a data sharing environment where the data pages are group buffer pool dependent. This can potentially be very expensive. We are talking the difference between minutes and hours of processing time!

Free Space for Updates

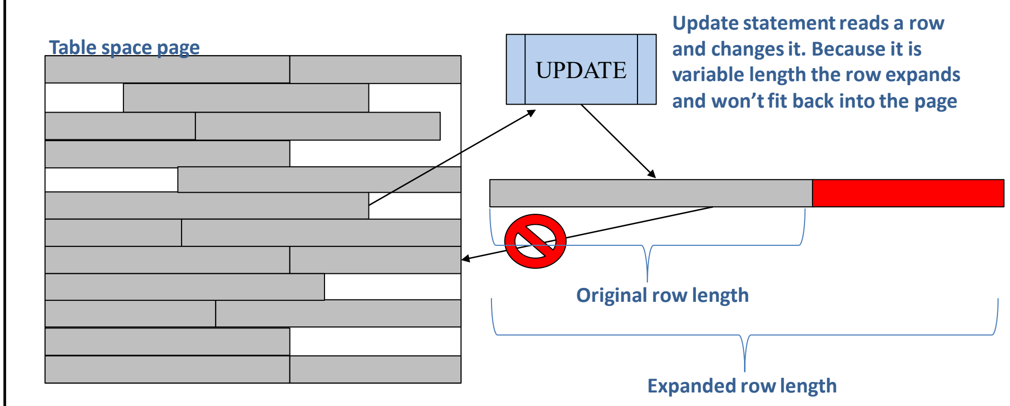
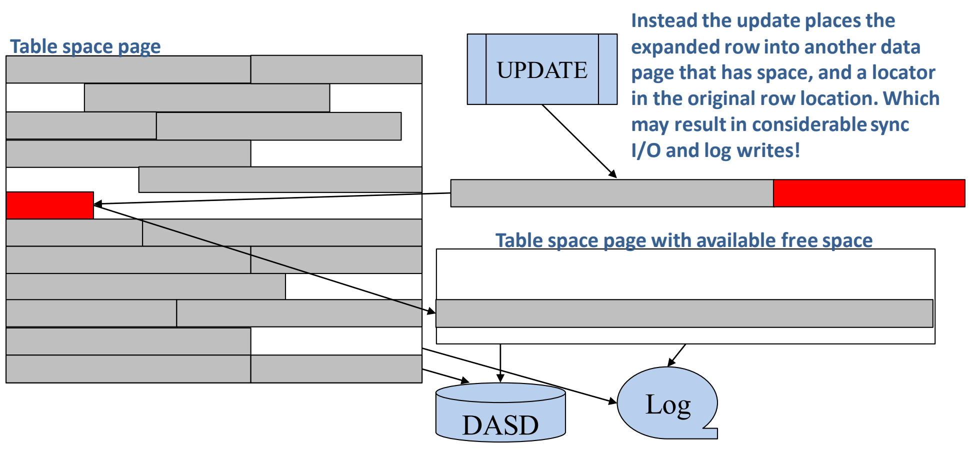


Table space page
UPDATE
Update statement reads a row and changes it. Because it is variable length the row expands and won't fit back into the page
Original row length
Expanded row length

This illustrates the result of an update operation where the length of a row increases and will not fit back on to the data page from where it came. Db2 will attempt to compact the page by consolidating all of the free space on the page to see if the expanded row will fit. If it will not fit then Db2 will have to relocate the row.

Free space for Updates

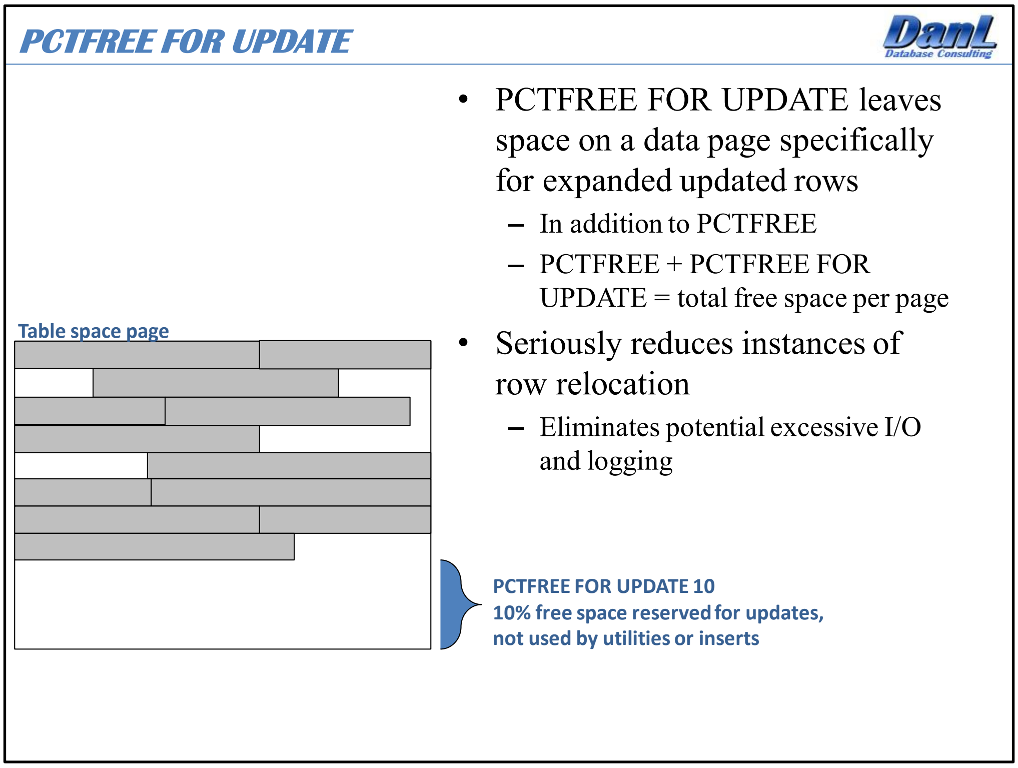


Instead the update places the expanded row into another data page that has space, and a locator in the original row location. Which may result in considerable sync I/O and log writes!

This represents the resulting operation where the updated row does not fit back onto the page from which it came, and Db2 does not compact the page, or Db2 compacts the page and the enlarged row will still not fit on the page. What Db2 will then do is search for free space in another data page that will fit the row. Once it places the expanded row in a new page it has to go back to the original page that it came from and place a locator in the previous row location that points to the updated relocated row in the new data page.

This operation could result in excessive synchronous I/O and logging!

The Catalog table SYSIBM.SYSTABLESPACESTATS can be monitored to determine the rate in which a tablespace partition is experiencing row relocation.



PCTFREE FOR UPDATE

- PCTFREE FOR UPDATE leaves space on a data page specifically for expanded updated rows
  - In addition to PCTFREE
  - PCTFREE + PCTFREE FOR UPDATE = total free space per page
- Seriously reduces instances of row relocation
  - Eliminates potential excessive I/O and logging

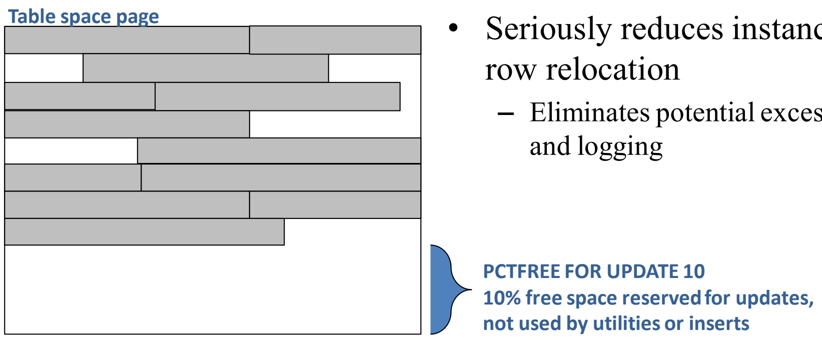
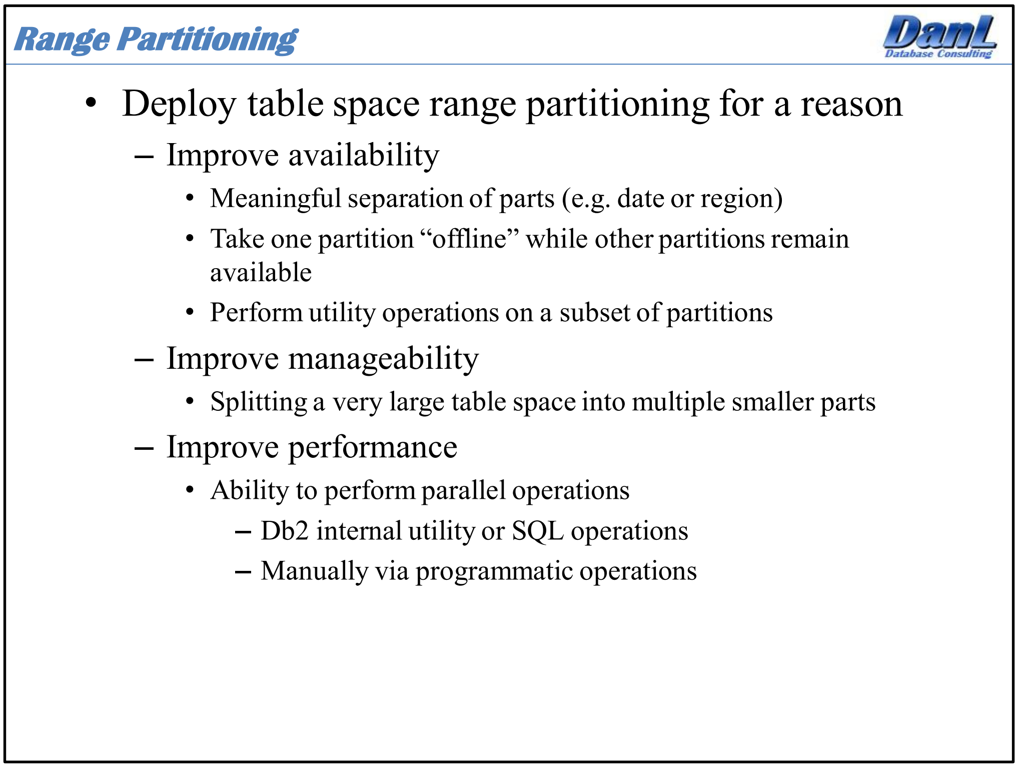


Table space page
PCTFREE FOR UPDATE 10
10% free space reserved for updates, not used by utilities or inserts

What PCTFREE FOR UPDATE does is reserve free space on a data page just for updated rows that increase in length. It will not be used by insert operations. It will be respected, along with PCTFREE, during LOAD and REORG utility executions. This additional free space is really useful for high update applications where row expansion is a possibility, and therefore could provide significant synchronous I/O relief in high volume environments.

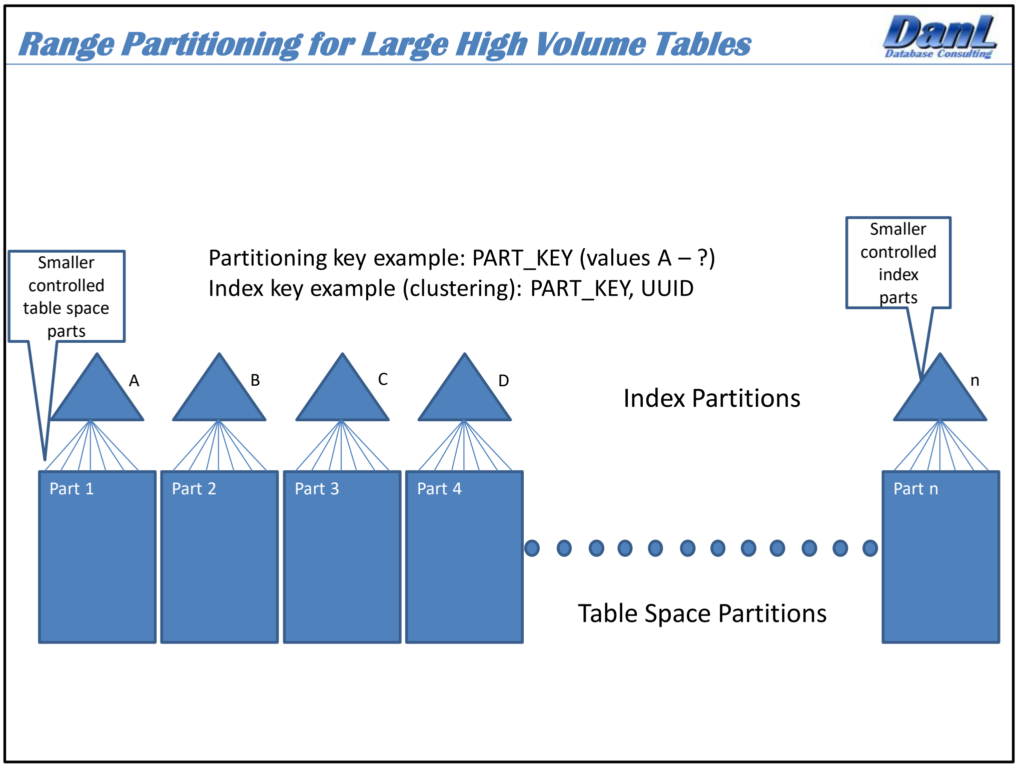


Range Partitioning

- Deploy table space range partitioning for a reason
  - Improve availability
    - Meaningful separation of parts (e.g. date or region)
    - Take one partition “offline” while other partitions remain available
    - Perform utility operations on a subset of partitions
  - Improve manageability
    - Splitting a very large table space into multiple smaller parts
  - Improve performance
    - Ability to perform parallel operations
      - Db2 internal utility or SQL operations
      - Manually via programmatic operations

There are two choices for partitioning; partition-by-growth (PBG) or partition-by-range (PBR). Typically a PBG table space is used to store relatively smaller and more stable table data. For really large tables and/or tables with high volume processing the preference is to use PBR table spaces. While a PBG can support partition level utility operations and parallelism, the DBA has little control over the distribution of the data in the partitions. In addition, a partitioned index cannot be created for a PBG table space. For a PBR table space, if a meaningful partitioning key can be defined, then there are definitely advantages to using a PBR table space for large high volume tables. Some advantages include:

- Meaningful partitioning keys
- Query parallelism
- Ability to control application parallelism based upon partitioning key values
- Partitioned indexes



Range Partitioning for Large High Volume Tables

DanL Database Consulting

Partitioning key example: PART_KEY (values A – ?)
Index key example (clustering): PART_KEY, UUID

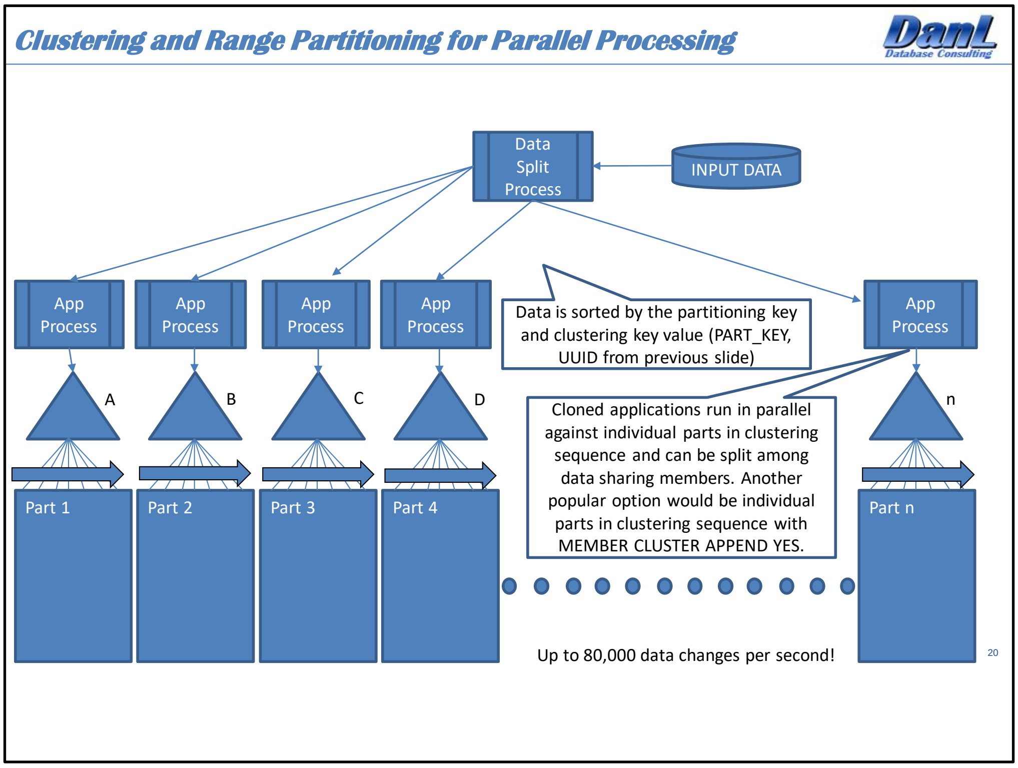
Smaller controlled table space parts
Smaller controlled index parts
A
B
C
D
n
Index Partitions
Part 1
Part 2
Part 3
Part 4
Part n
Table Space Partitions

There are several reasons for choosing range-partitioning:

- Rotating and aging data
- Separating old and new data
- Separating data logically for region or business segment based availability
- Spreading data out to distribute access
- Logically organizing data to match processes for high concurrency

One of the performance advantages to range-partitioning is in distributing the data to spread out overall access to the table space. What this means is that index b-tree levels can potentially be reduced due to reduced entry count per partition and that can reduce I/O's and getpages to indexes. In addition, free space searches could be more efficient in a partitioned situation since the data is broken up and the searches are on a partition-by-partition basis. The same is true for any partitioned indexes defined against the table space.

From a performance perspective, range-partitioning is useful to enable a high degree of parallelism for queries and utilities. In addition, queries coded against range-partitioned tables can take advantage of partition elimination, also sometimes called page range screening, when predicates are coded against partitioning columns. These predicates can contain host variable or parameter markers, literal values, and, beginning with DB2 11 for z/OS, joined columns.

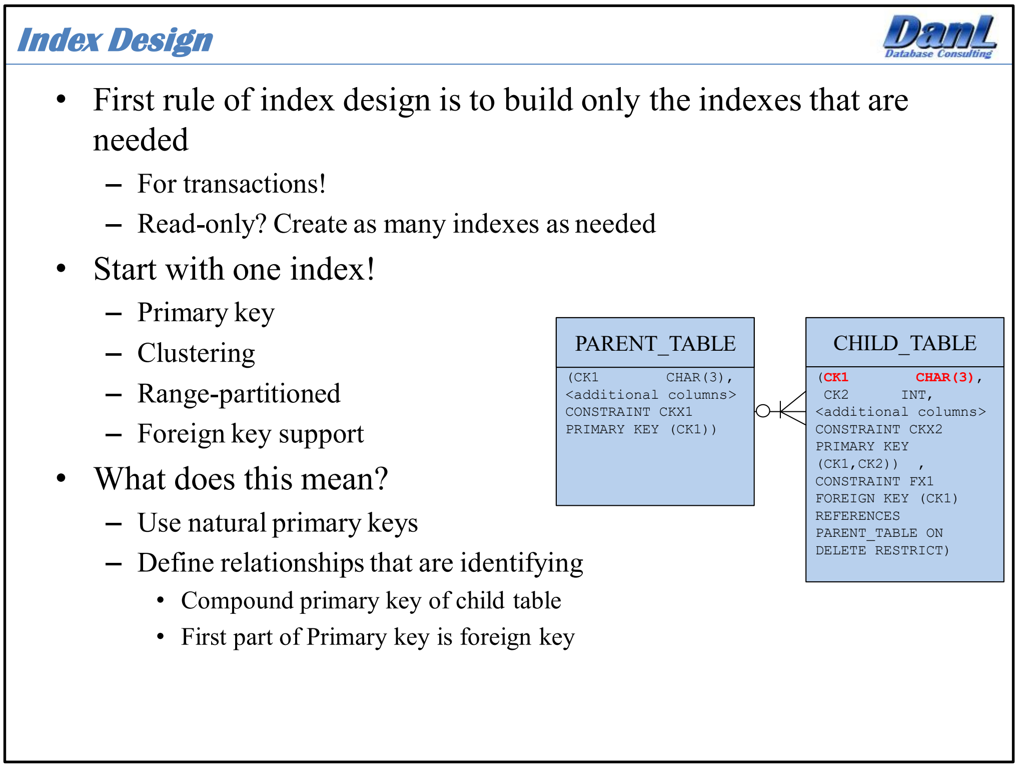


Clustering and Range Partitioning for Parallel Processing

DanL Database Consulting
Data Split Process
INPUT DATA
App Process
App Process
App Process
App Process
App Process
Data is sorted by the partitioning key and clustering key value (PART_KEY, UUID from previous slide)
A
B
C
D
n
Cloned applications run in parallel against individual parts in clustering sequence and can be split among data sharing members. Another popular option would be individual parts in clustering sequence with MEMBER CLUSTER APPEND YES.
Part 1
Part 2
Part 3
Part 4
Part n
Up to 80,000 data changes per second!
20

For the highest level of partitioned table access possible an effective design technique to use is called "application controlled parallelism". This is a technique that is useful when there exists high volume batch processes acting against very large OLTP data stores. In these situations, it is a careful balance between partitioning, clustering (or intentionally not clustering), and distribution of transactions in a meaningful way across multiple application processes. What I mean by a meaningful way is that the distribution of inbound transactions has to match the partitioning and/or clustering of the data in the table space partitions accessed during the processing. In this way, each independent (or sometimes dependent) process operates against its own set of partitions. This will eliminate locking contention and possibly allow for a lock size of partition and use of MEMBER CLUSTER to further improve performance of ultra-high insert/update tables.

When the input is clustered in the same sequence as the data is stored, or you are using MEMBER CLUSTER or APPEND ON to place data at the end of a table space partition, Db2 can take advantage of sequential detection to further improve the performance of these types of processing. In these situations (real world) the business transaction rates have been as high as 21,000 per second, single table insert rates as high as 30,000 per second, and application wide insert rates as high as 60,000 per second with overall data change rates at 79,000 per second. These are not test cases but real world results.



Index Design

DanL Database Consulting

- First rule of index design is to build only the indexes that are needed
  - For transactions!
  - Read-only? Create as many indexes as needed
- Start with one index!
  - Primary key
  - Clustering
  - Range-partitioned
  - Foreign key support
- What does this mean?
  - Use natural primary keys
  - Define relationships that are identifying
    - Compound primary key of child table
    - First part of Primary key is foreign key

PARENT_TABLE

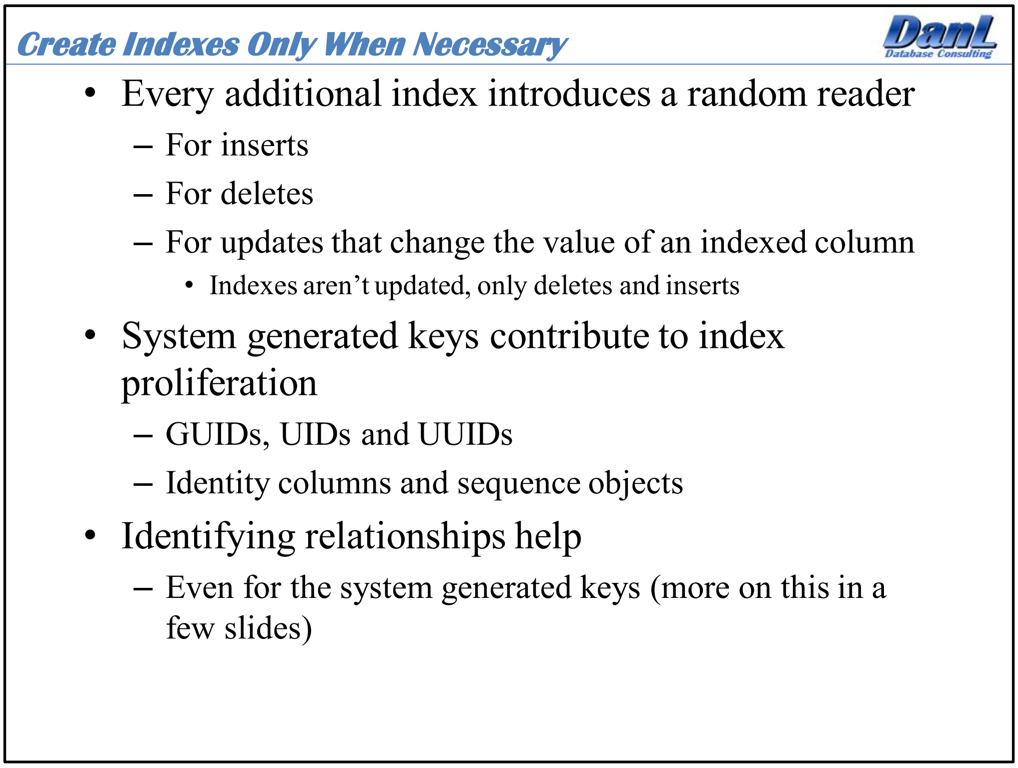
```
(CK1          CHAR(3),
<additional columns>
CONSTRAINT CKX1
PRIMARY KEY (CK1))
```

CHILD_TABLE

```
(CK1          CHAR(3),
 CK2          INT,
<additional columns>
CONSTRAINT CKX2
PRIMARY KEY
(CK1,CK2))  ,
CONSTRAINT FX1
FOREIGN KEY (CK1)
REFERENCES
PARENT_TABLE ON
DELETE RESTRICT)
```

If a table is read-only then there is no need to worry about indexes as no number of indexes will create a negative performance situation. However, most tables are not read-only. A common rule of thumb might be 80% reads and 20% data changes, however in this day of mass ingest those percentages could be highly inaccurate. It is important to note that indexes do not get updates, and an update to an index is a delete and insert of the index entry. It is also very important to know the data change rate for tables, as well as the common access paths into those tables. Then a decision can be made about the number of indexes on the table. However, a very good rule of thumb is to start with a design that utilizes only one index per table! The goal is to have this index serve all purposes, whether it be clustering, partitioning, access path, etc.

Use identifying relationships when setting up tables. In database terms, relationships between two entities may be classified as being either identifying or non-identifying. Identifying relationships exist when the primary key of the parent entity is included in the primary key of the child entity as the initial columns of that key. On the other hand, a non-identifying relationship exists when the primary key of the parent entity is included in the child entity but not as part of the child entity’s primary key. A significant number of modern day developers will object identifying relationships as they prefer a single column system-generated unique key for each table. So, not every table needs to be designed in this manner, but perhaps a compromise can be reached with only the highest volume data changes tables getting the 1-index design.

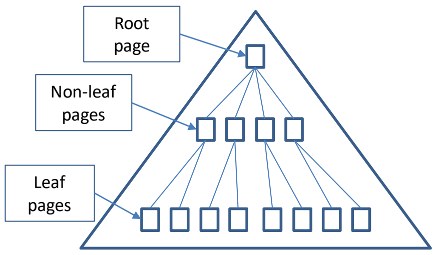


Create Indexes Only When Necessary

- Every additional index introduces a random reader
  - For inserts
  - For deletes
  - For updates that change the value of an indexed column
    - Indexes aren't updated, only deletes and inserts
- System generated keys contribute to index proliferation
  - GUIDs, UIDs and UUIDs
  - Identity columns and sequence objects
- Identifying relationships help
  - Even for the system generated keys (more on this in a few slides)

Be careful in creating additional indexes on a table, especially for tables with a high insert or delete rate, and also for tables in which the indexes keys would be heavily updated. For partitioned tables, non-partitioned secondary indexes and data-partitioned secondary indexes can provide further challenges in the form of large size and/or access path challenges.

Index PCTFREE and FREEPAGE



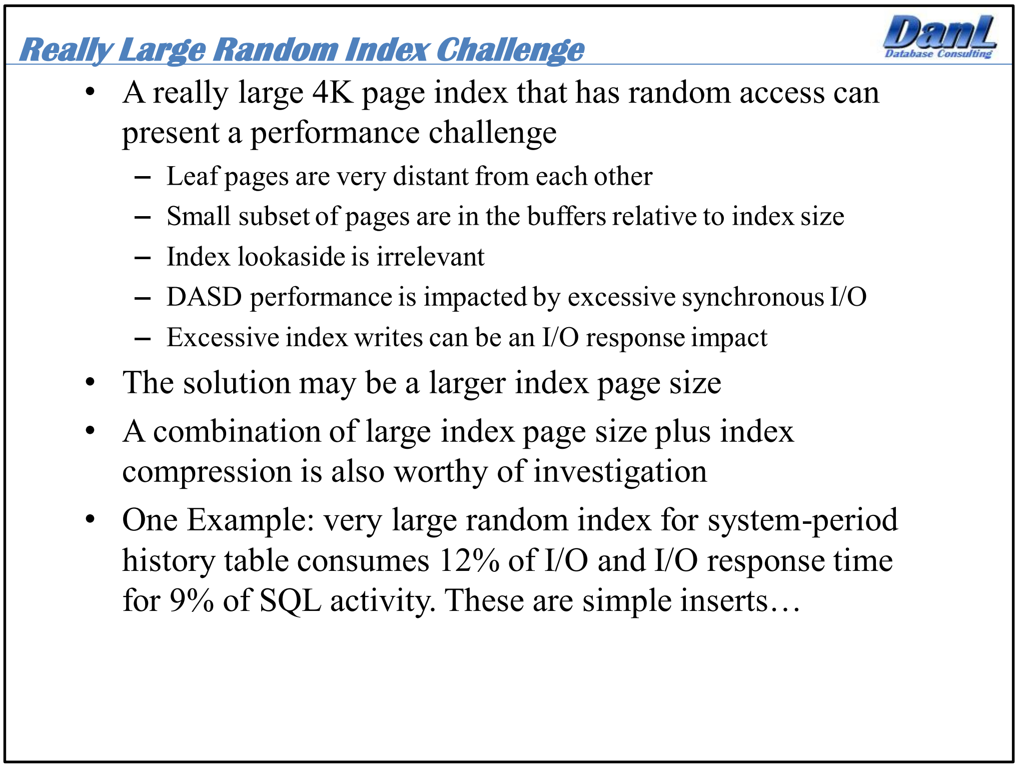
Recommendation only for natural key or random index (new keys non-ascending or descending).

- Unlike table spaces, indexes are always kept in key sequence
- PCTFREE should be utilized to avoid index splits between index REORGs
  - Calculate the number of new index entries between index REORGs divided by the total number of index entries to estimate PCTFREE. The goal is no index page splits between index REORGs.
- FREEPAGE should be utilized if index splits can't be avoided between index REORGs

When an index page fills with entries it must be split. When Db2 splits an index page it needs to put the new page somewhere, and it needs a completely empty available page to do it. Another thing that happens during an index page split is that Db2 will serialize on the root page when the split occurs. So, index page splits are a serial process, meaning one agent at a time can split a page. For highly parallelized processes where you have multiple agents acting on a single index or index partition then there could be concurrency issues. Since indexes are typically maintained in key sequence then free space should be allocated such that index page splits are avoided between index REORGs. If splits can't be avoided then a combination of PCTFREE and FREEPAGE can be set to try to reduce the overhead of splits so that if one happens Db2 can find a page nearby to use for the new page. This is most important if the index key is a natural key.

If the index key is a system-generated ascending or descending value, or if the key is non system-generated but still ascending or descending in nature (that is, new keys are assigned in a sequence), then it is perhaps best to allocate no free space and let Db2 simply tack new entries to the end of the index. In this situation if the index keys are updated then some free space might be helpful.

Indexes should be reorganized more frequently than table spaces.

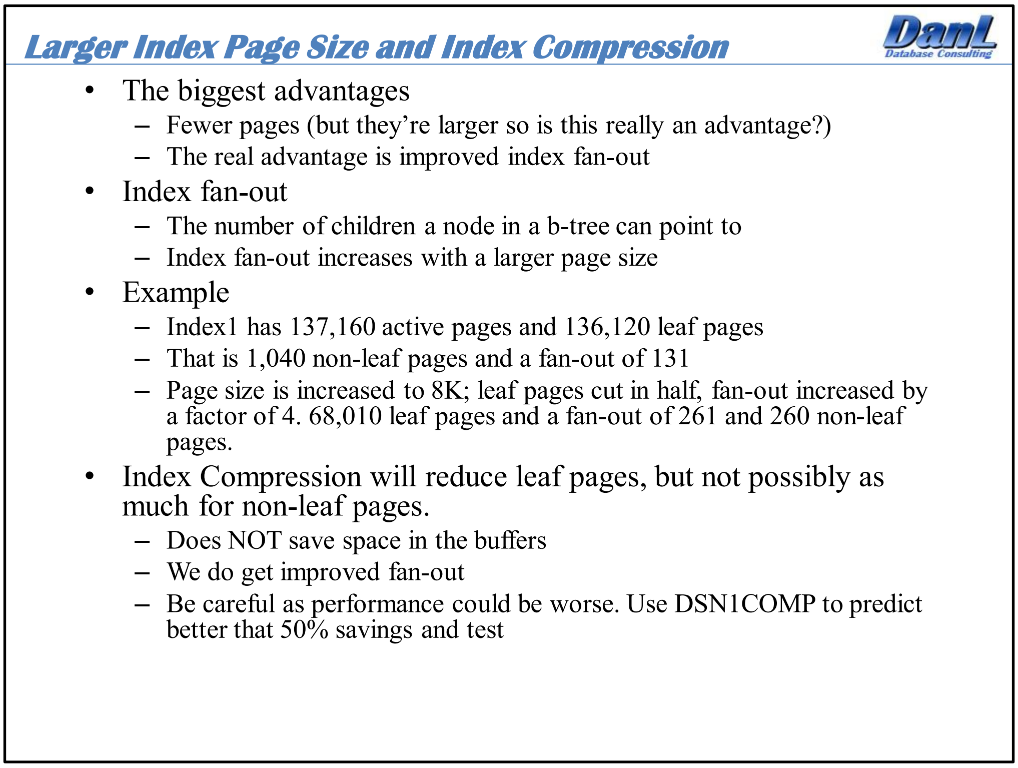


Really Large Random Index Challenge

- A really large 4K page index that has random access can present a performance challenge
  - Leaf pages are very distant from each other
  - Small subset of pages are in the buffers relative to index size
  - Index lookaside is irrelevant
  - DASD performance is impacted by excessive synchronous I/O
  - Excessive index writes can be an I/O response impact
- The solution may be a larger index page size
- A combination of large index page size plus index compression is also worthy of investigation
- One Example: very large random index for system-period history table consumes 12% of I/O and I/O response time for 9% of SQL activity. These are simple inserts…

When an index allocated with 4K pages becomes quite large, at 200+ gigabytes in size, they can become a bit of a performance impact. This is due to the fact that there could be significant random access to the index, and that could lead to a large volume of synchronous I/O, and very inefficient I/O's. These indexes are difficult to tune because as free space is added in the form of PCTFREE and FREEPAGE it increases the size of the index, and that can exacerbate the situation.

This may be the perfect situation for a larger index page size!



Larger Index Page Size and Index Compression

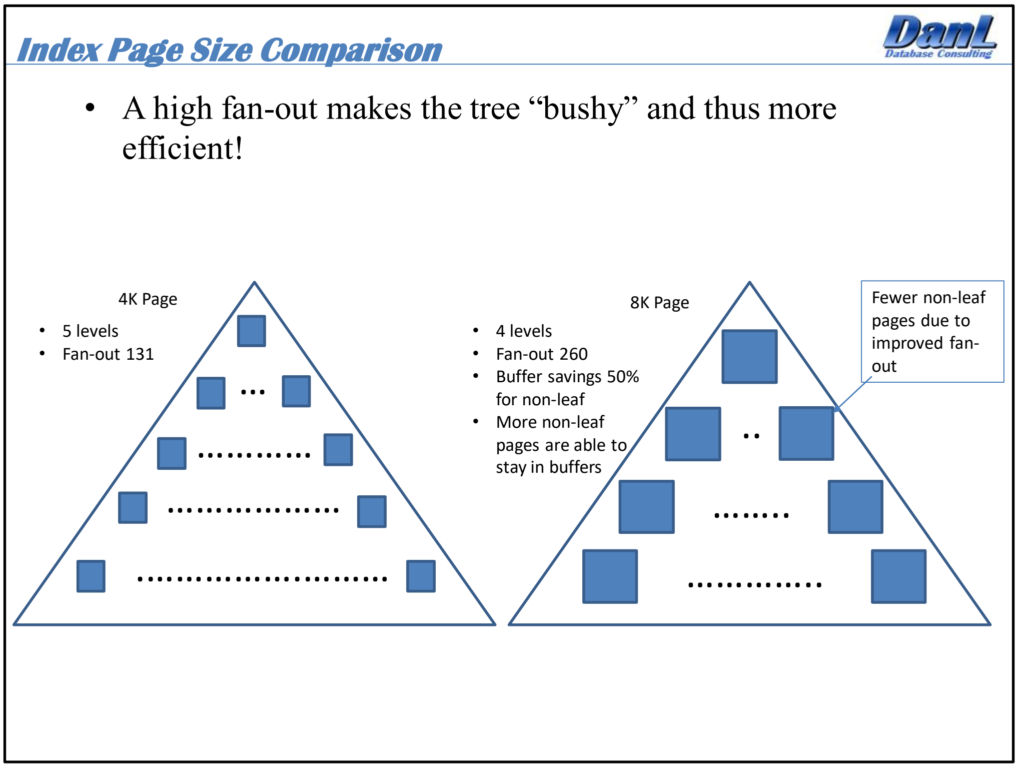
- The biggest advantages
  - Fewer pages (but they're larger so is this really an advantage?)
  - The real advantage is improved index fan-out
- Index fan-out
  - The number of children a node in a b-tree can point to
  - Index fan-out increases with a larger page size
- Example
  - Index1 has 137,160 active pages and 136,120 leaf pages
  - That is 1,040 non-leaf pages and a fan-out of 131
  - Page size is increased to 8K; leaf pages cut in half, fan-out increased by a factor of 4. 68,010 leaf pages and a fan-out of 261 and 260 non-leaf pages.
- Index Compression will reduce leaf pages, but not possibly as much for non-leaf pages.
  - Does NOT save space in the buffers
  - We do get improved fan-out
  - Be careful as performance could be worse. Use DSN1COMP to predict better that 50% savings and test

Index fan-out is defined as the number of children a node in a b-tree can point to. So, by having a larger page size, and also possibly compressing the index key, you can get a greater index fan-out. This has sort of a double effect, or maybe a quadruple effect with compression.

Let's look at one partition of one index. There are 137,160 active pages and 136,120 leaf pages. That means about 1,040 non-leaf pages, and a fan-out of 131. If we increase the page size to 8K we will cut the number of leaf pages in half, but also approximately increase the fan-out by a factor of 2 because the non-leaf pages are also larger. So, if I just use simple math 136,120/2 is 68,060 leaf pages. Now those non-leaf pages will be reduced by a factor of 4 to about 260, with a fan-out of 261.

It looks like compression will get us another 50% reduction in leaf pages to 34,030. I don't think we reduce non-leaf pages in this case.

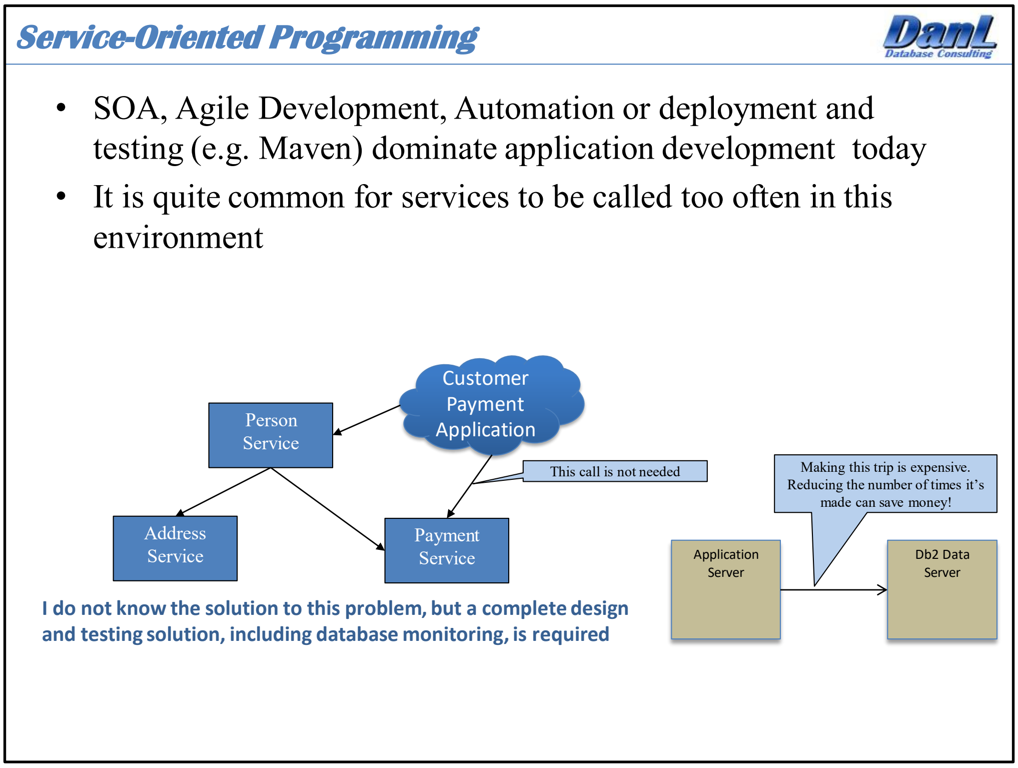
As far as the buffer pool go the compression does not save space in the buffer because the index pages are expanded in the pool. However, the large page size gets us the increased fan-out, which does reduce the amount of memory in the buffer pool required for non-leaf pages. This is due to the double effect I mentioned earlier with more entries per page and half the leaf pages for the non-leaf pages to point to. So, in this example we could possibly cut non-leaf buffer storage requirements in half, from 3GB to 1.5GB.



Index Page Size Comparison

- A high fan-out makes the tree "bushy" and thus more efficient!

The potential savings is two fold. First there could be fewer I/O's because there are less pages to read in the larger page size index. Second, because of the improved fan-out the number of non-leaf pages can be drastically reduced and thus require less actual memory inside the buffer pool. There is also one less index level to traverse.


Customer Payment Application
Person Service
Address Service
Payment Service
This call is not needed
Making this trip is expensive. Reducing the number of times it's made can save money!
Application Server
Db2 Data Server

Service-Oriented Programming

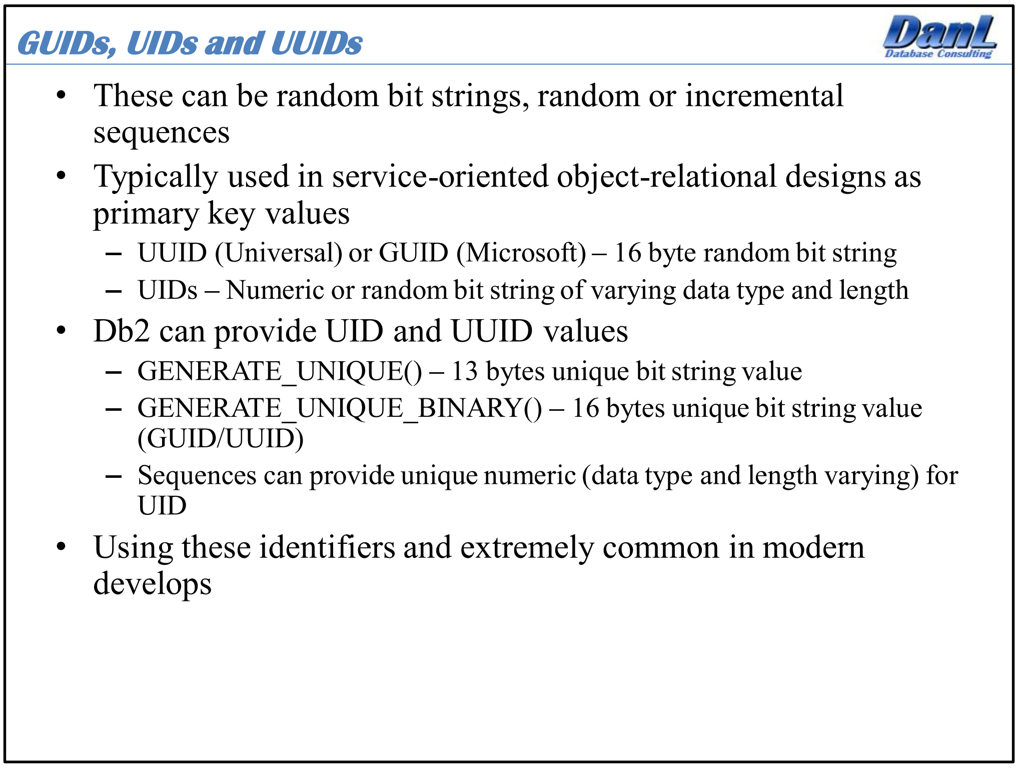
- SOA, Agile Development, Automation or deployment and testing (e.g. Maven) dominate application development today
- It is quite common for services to be called too often in this environment

I do not know the solution to this problem, but a complete design and testing solution, including database monitoring, is required

Service oriented Agile development practices using frameworks and open software components are really important for cost savings related to application development. However, if the framework used and the development techniques are not uniquely controlled (and they often aren't), then this could lead to some sloppy design. It is important that there is a proof of concept, or at least accurate testing involving the database, to be sure that the database is being called the least number of times for a given service or transaction.

At one customer site each service included an optional "test" mode, which could be invoked via a global parameter. When in "test" mode the service would log the service name and entry and exit time each time the service was called. In that way we could track the relationship between a business transaction and the various service calls that support that transaction. The result of this policy resulted in a tremendous amount of machine time savings, improved transaction response times, and no major impact to application development time. This was due to the fact that we could catch redundant service calls during testing.

Calling the database server too often is the number one waste of machine resources that I have witnessed in all my years of working with Db2.



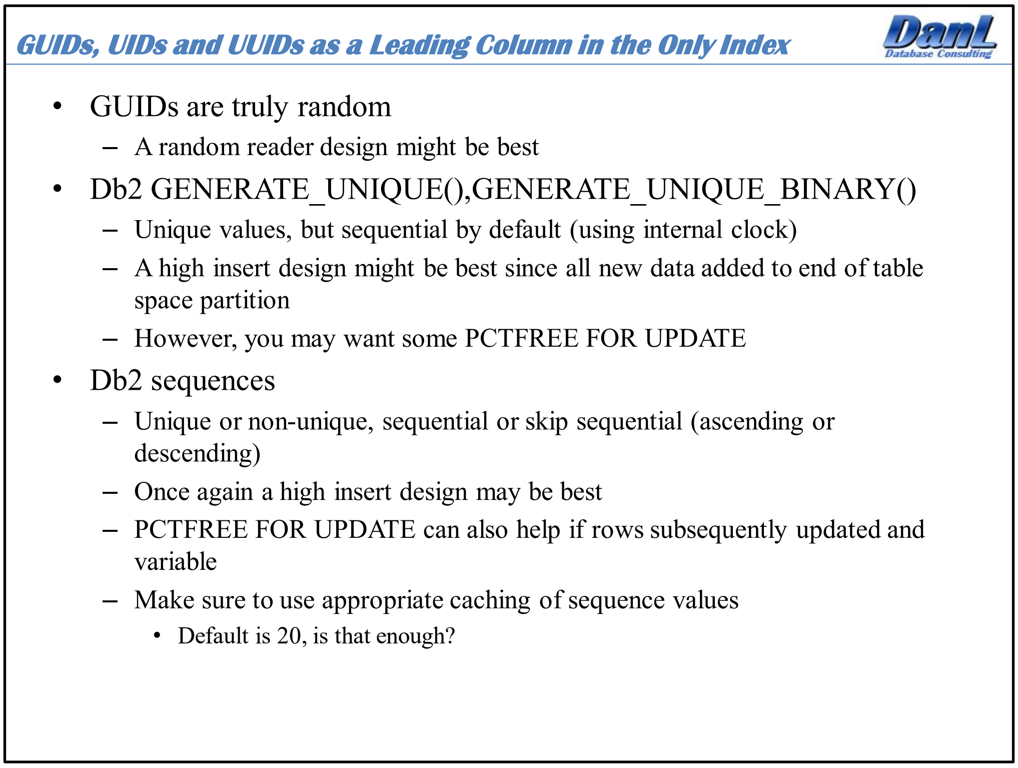
GUIDs, UIDs and UUIDs

- These can be random bit strings, random or incremental sequences
- Typically used in service-oriented object-relational designs as primary key values
  - UUID (Universal) or GUID (Microsoft) – 16 byte random bit string
  - UIDs – Numeric or random bit string of varying data type and length
- Db2 can provide UID and UUID values
  - GENERATE_UNIQUE() – 13 bytes unique bit string value
  - GENERATE_UNIQUE_BINARY() – 16 bytes unique bit string value (GUID/UUID)
  - Sequences can provide unique numeric (data type and length varying) for UID
- Using these identifiers and extremely common in modern develops

UUIDs are standard practice for many modern designs that incorporate an object-relational data stack. A standardized UUID is randomized based upon an internal time clock, but in practice it could be generated by any number of different algorithms. When UUIDs are introduced into a database design they can impact the decisions that are made relative to indexing, clustering and free space! Therefore, it is important to understand these IDs that are generated for a particular database design, and make appropriate accommodations. Perhaps it would be fruitful to address the need for natural keys for high volume tables, or possibly compound UUID keys for critical parent-child relationships.

It is possible to negotiate the natural-key “in” for a database design. Allow for clustered access based upon that natural key, and then cluster the rest of the tables by the UUID of that parent table, whether those keys be identifying or non-identifying. Remember this is something that is being done for high performance. If high performance is not an expected feature of the application then there is no need to fight the design team over primary keys.

If management tells you “we want the application to run as fast as possible” then your response should be “well great, buy an infinite number of machines”. Ask for a reasonable expectation of performance, and if the use of UUIDs won’t impede that expectation then relax, all is well. If there is fear of a performance bottleneck then special considerations should be made about the use of UUIDS, indexing, clustering, and free space!



GUIDs, UIDs and UUIDs as a Leading Column in the Only Index

- GUIDs are truly random
  - A random reader design might be best
- Db2 GENERATE_UNIQUE(),GENERATE_UNIQUE_BINARY()
  - Unique values, but sequential by default (using internal clock)
  - A high insert design might be best since all new data added to end of table space partition
  - However, you may want some PCTFREE FOR UPDATE
- Db2 sequences
  - Unique or non-unique, sequential or skip sequential (ascending or descending)
  - Once again a high insert design may be best
  - PCTFREE FOR UPDATE can also help if rows subsequently updated and variable
  - Make sure to use appropriate caching of sequence values
    - Default is 20, is that enough?

Clustering can be a challenge with a variety of generated unique id's as keys. If Db2 Generated values are used, whether they are GENERATE_UNIQUE() values or ascending sequence values then maintaining cluster based upon natural keys can be challenging without secondary indexes. However, you can create secondary indexes based upon a natural key, if one exists for the table, and cluster by that key. While the table may benefit from clustering, the indexes are another separate issue. It really depends upon the access patterns of the application queries, and which indexes they end up using.

A really common problem in Db2 is the use of sequence values without appropriate caching set. The default is 20, but if there is a high usage rate then that may be too small, especially in a data sharing environment. Setting ORDER and NO CACHE for sequence values is a massive trade-off for consistent numerical values and performance!

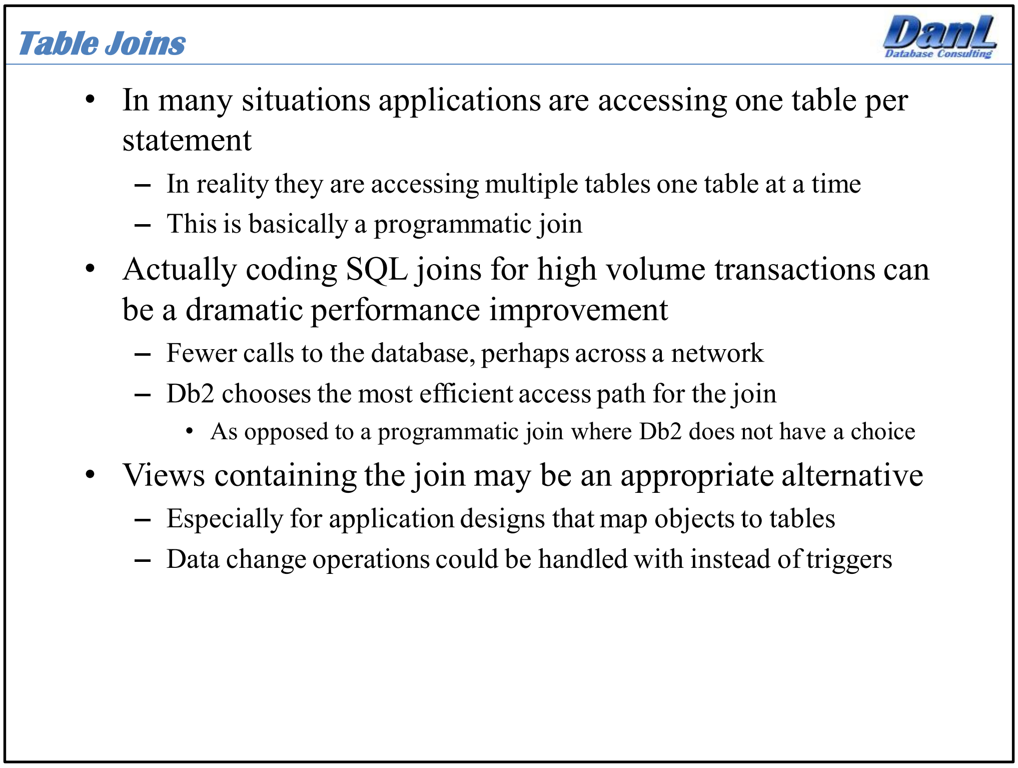
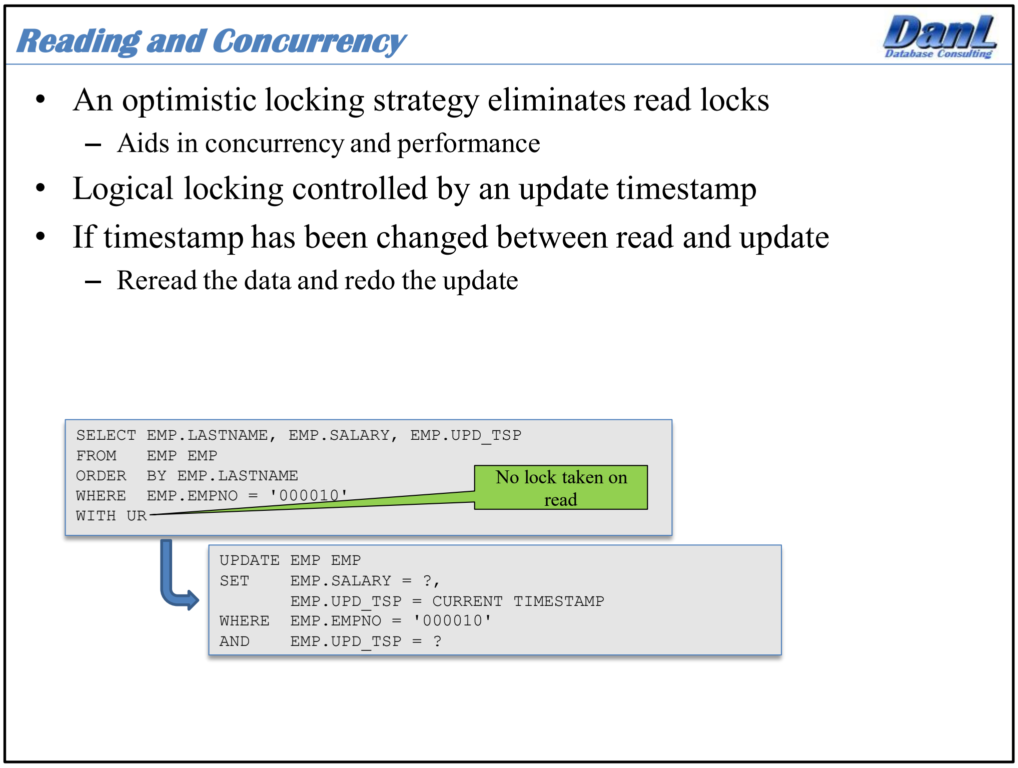


Table Joins

- In many situations applications are accessing one table per statement
  - In reality they are accessing multiple tables one table at a time
  - This is basically a programmatic join
- Actually coding SQL joins for high volume transactions can be a dramatic performance improvement
  - Fewer calls to the database, perhaps across a network
  - Db2 chooses the most efficient access path for the join
    - As opposed to a programmatic join where Db2 does not have a choice
- Views containing the join may be an appropriate alternative
  - Especially for application designs that map objects to tables
  - Data change operations could be handled with instead of triggers

This is as simple as it gets. Do you need data from two tables that are related? Code a two-table join versus a programmatic join that uses separate SQL statements. The advantage to a join versus individual SQL statements is twofold: first, it avoids excessive SQL calls, and second, DB2 chooses the most efficient access path versus the application developer forcing the access path by coding the two separate statements. That second point is extremely important to understand. If you are coding a join in a program, you are forcing the table access sequence, which is an extremely critical factor in database performance. If you code a SQL join the database optimizer makes a decision as to which table to access first. The database decision is based upon statistics and can change due to changes in tables, indexes, and table contents. The multi-table access path coded in an application program does not take any of this into account and will not change unless the developer modifies the code, which is a major disadvantage. In simple tests that I have conducted, a simple two-table join outperformed an equivalent programmatic join by 30%.



Reading and Concurrency

- An optimistic locking strategy eliminates read locks
  - Aids in concurrency and performance
- Logical locking controlled by an update timestamp
- If timestamp has been changed between read and update
  - Reread the data and redo the update

```
SELECT EMP.LASTNAME, EMP.SALARY, EMP.UPD_TSP
FROM   EMP EMP
ORDER  BY EMP.LASTNAME
WHERE  EMP.EMPNO = '000010'
WITH UR
```

No lock taken on read

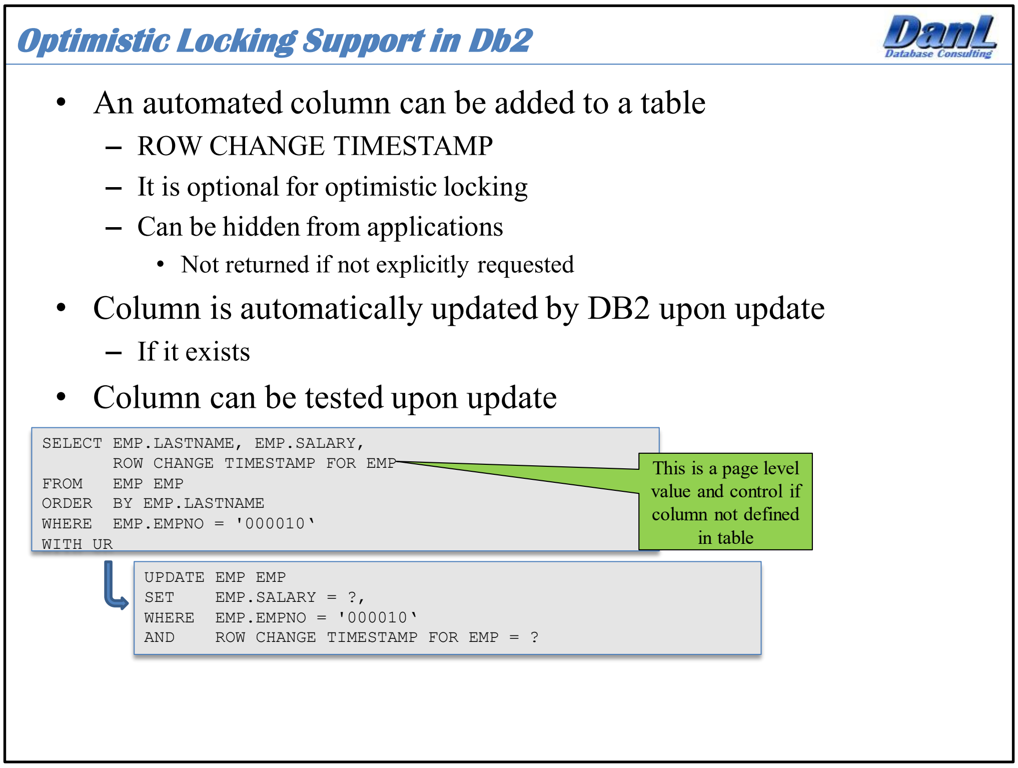
```
UPDATE EMP EMP
SET    EMP.SALARY = ?,
       EMP.UPD_TSP = CURRENT TIMESTAMP
WHERE  EMP.EMPNO = '000010'
AND    EMP.UPD_TSP = ?
```

An optimistic locking strategy is not a new technique. However it is a pretty significant technique for improving performance and concurrency for a high volume application. To support an application with an optimistic locking strategy all tables must contain a timestamp column that represents the time of last update or insert. The timestamp column in the table is the key to optimistic locking and enables concurrent processes to share data without initial concern for locking. As an application carries the data along it holds the responsibility for maintaining the update timestamp in memory and checking it against the table in the database for a given primary key before updating it. The update timestamp itself is not a part of the primary key, but is used for change detection for the row represented by the primary key.

All tables are read with isolation level uncommitted read. Using an isolation level of UR means you can potentially be reading dirty uncommitted data. However, the application will not wait for any lock that is held on a piece of data that has been updated by a concurrent in-flight transaction. The read of the data absolutely has to include, at the very least, the primary key value and the update timestamp. This update timestamp has to be retained for use in testing the validity of the data held in memory by the process.

When a process moves to update data it has previously retrieved, it has to do two things. First, it has to test the update timestamp to see if the state of the data in the table is the same as when it was initially retrieved, and second, it needs to update the update timestamp to reflect the change in the state of the data as a result of its processing. The update timestamp is tested for the value that was retrieved when the row was originally read and updated to the timestamp of the update.

This technique does a great job of improving the performance and concurrency of an application. It does, however, relieve the database engine of some of the concurrency control by using uncommitted read and puts a higher responsibility on the application to properly maintain the update timestamps and therfore the data integrity.



Optimistic Locking Support in Db2

- An automated column can be added to a table
  - ROW CHANGE TIMESTAMP
  - It is optional for optimistic locking
  - Can be hidden from applications
    - Not returned if not explicitly requested
- Column is automatically updated by DB2 upon update
  - If it exists
- Column can be tested upon update

```
SELECT EMP.LASTNAME, EMP.SALARY,
       ROW CHANGE TIMESTAMP FOR EMP
FROM   EMP EMP
ORDER  BY EMP.LASTNAME
WHERE  EMP.EMPNO = '000010'
WITH UR
```

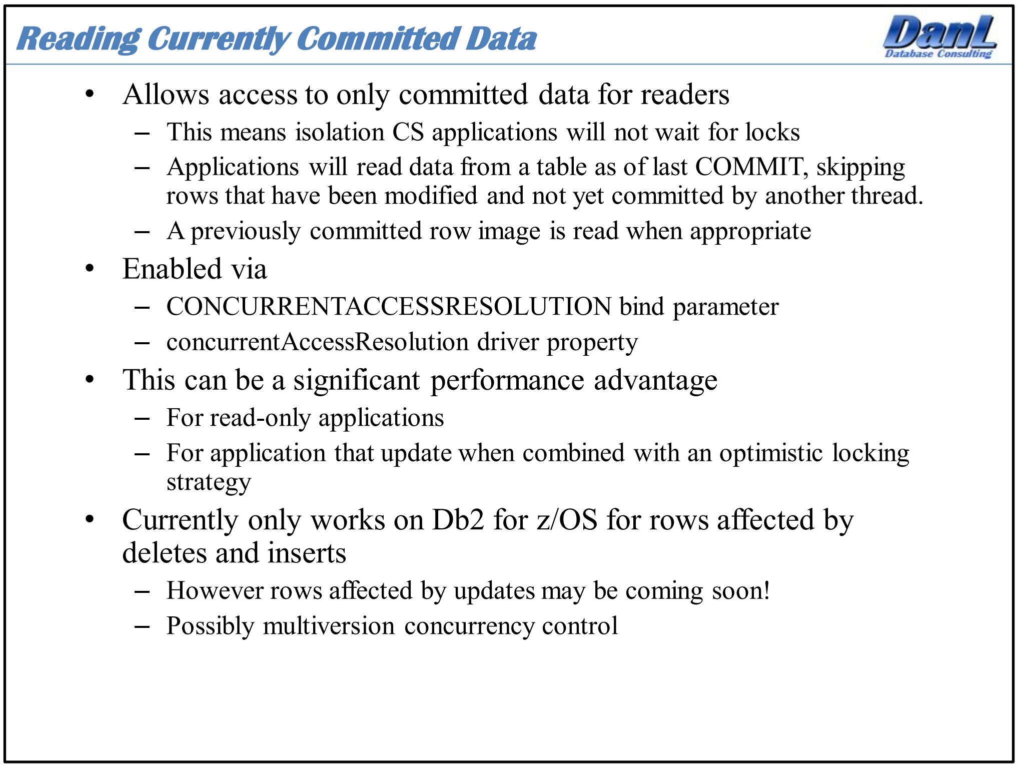
This is a page level value and control if column not defined in table

```
UPDATE EMP EMP
SET    EMP.SALARY = ?,
WHERE  EMP.EMPNO = '000010'
AND    ROW CHANGE TIMESTAMP FOR EMP = ?
```

Db2 provides support for optimistic locking and that removes some of the application responsibility in the control of optimistic locking. This support comes in the form of an automated generated column called a row change timestamp column. This column can be generated by default or always by DB2, but for optimistic locking situations it is strongly advised to force it as generated always. The advantage of this generated column is that it moves the responsibility of updating the timestamp column out of the hands of the application and into the hands of DB2, thus guaranteeing the accuracy of the timestamp value enterprise-wide. This is quite obvious if we compare the fully application-controlled optimistic locking strategy laid out in the previous slide with the same strategy that uses the built-in DB2 functionality.

The difference here is in how the update timestamp is retrieved. The row change timestamp column is not named in the SELECT but instead specified. You could also select the column by name if desired.

While the testing of the timestamp still has to be performed by the application, the updating of the timestamp happens automatically by Db2. This ensures a higher level of data integrity for optimistic locking.



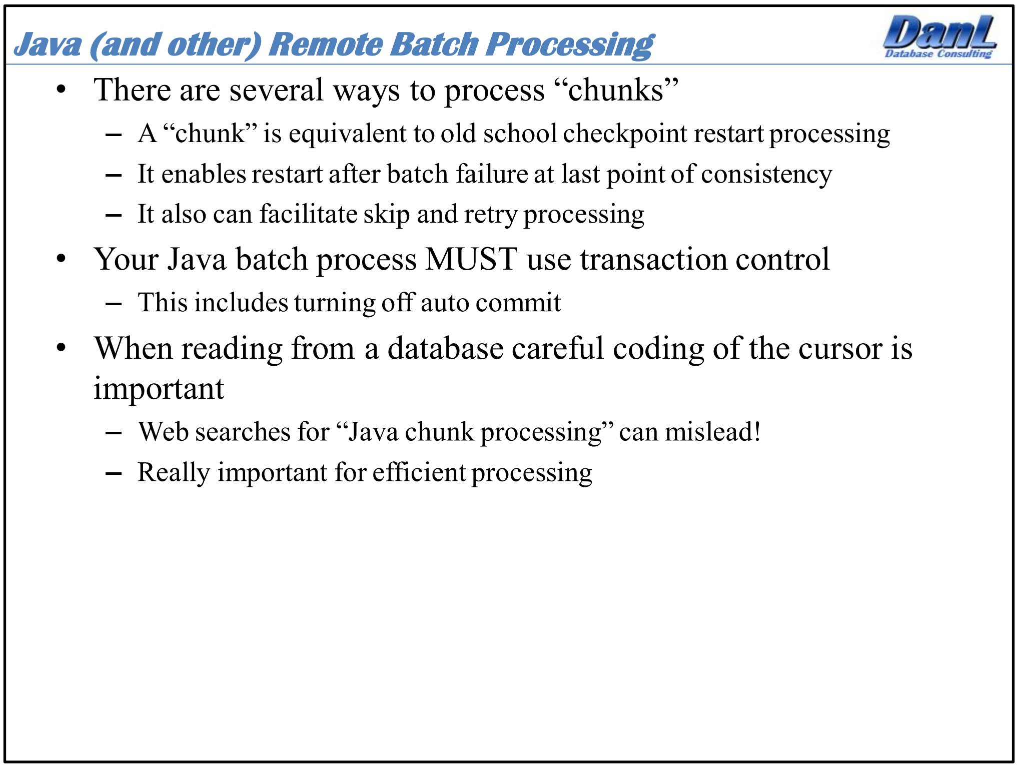
Reading Currently Committed Data

- Allows access to only committed data for readers
  - This means isolation CS applications will not wait for locks
  - Applications will read data from a table as of last COMMIT, skipping rows that have been modified and not yet committed by another thread.
  - A previously committed row image is read when appropriate
- Enabled via
  - CONCURRENTACCESSRESOLUTION bind parameter
  - concurrentAccessResolution driver property
- This can be a significant performance advantage
  - For read-only applications
  - For application that update when combined with an optimistic locking strategy
- Currently only works on Db2 for z/OS for rows affected by deletes and inserts
  - However rows affected by updates may be coming soon!
  - Possibly multiversion concurrency control

The ability to read currently committed data is enabled by the CONCURRENTACCESSRESOLUTION bind parameter for packages and the concurrentAccessResolution IBM Data Server Driver property for JCBC connections. From the manual: USECURRENTLYCOMMITTED specifies that when a read transaction requires access to a row that is locked by an INSERT or DELETE operation, the database manager can access the currently committed row, if one exists, and continue to the next row. The read transaction does not need to wait for the INSERT or DELETE operation to commit. This clause applies when the isolation level in effect is cursor stability or read stability.

This may be an interesting choice for performance. By setting this option to USECURRENTLYCOMMITTED an application that attempts to read data that is locked by an insert or a delete operation will not wait for the lock to be released, but instead access the currently committed row, if one exists, and continue the operation. This can really change the access paradigm for an application and can be considered a significant performance feature, especially when coupled with an optimistic locking strategy. However, it only works for rows impacted by insert and delete operations at this time, and doesn’t work for updates…yet. This makes its use limited unless you implement “update-less” operations for select applications, which means doing a delete and insert instead of an update.

Look forward to currently committed data expanding to cover updated data hopefully in the near future!

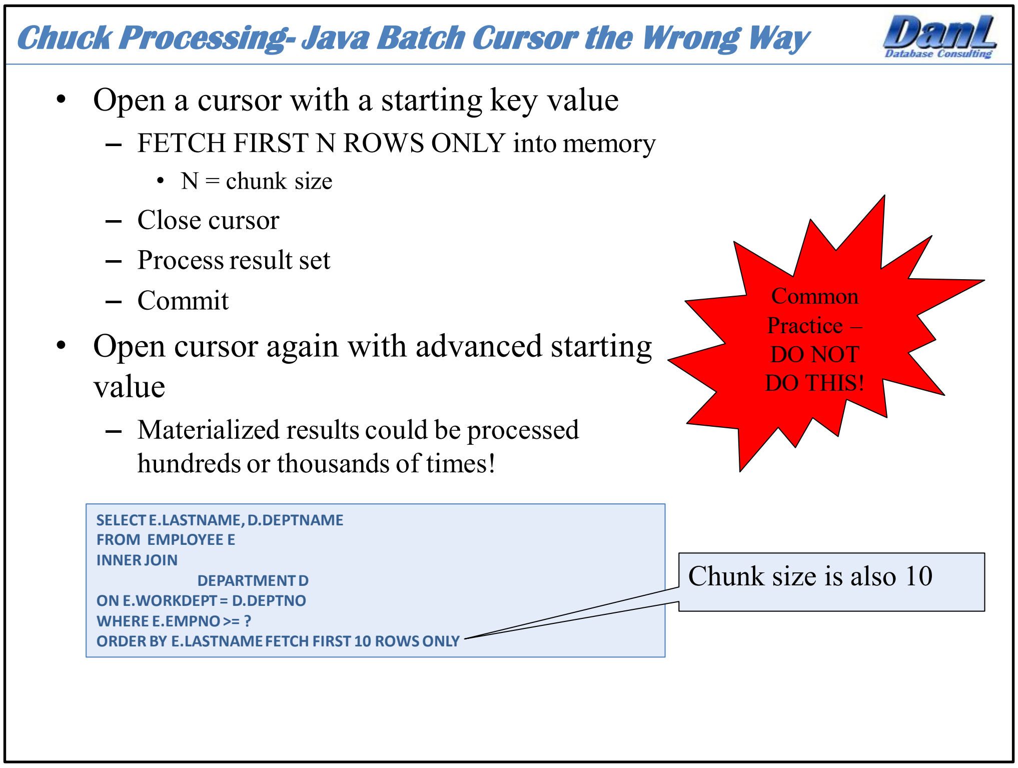


Java (and other) Remote Batch Processing

- There are several ways to process “chunks”
  - A “chunk” is equivalent to old school checkpoint restart processing
  - It enables restart after batch failure at last point of consistency
  - It also can facilitate skip and retry processing
- Your Java batch process MUST use transaction control
  - This includes turning off auto commit
- When reading from a database careful coding of the cursor is important
  - Web searches for “Java chunk processing” can mislead!
  - Really important for efficient processing

Batch processing using Java is becoming increasingly popular since large scale z/OS based batch programs are being rewritten for deployment in the cloud. It is important for application developers to understand proper processing of such things like checkpoint/restart, proper cursor positioning, proper transaction control, and efficient driving cursor design. There is a Java batch processing standard, JSR 352, which is a good starting point for understanding proper batch processing. A lot of the standard follows common mainframe batch practices that have been well established for many years.

There is the correct way and the incorrect way to do Java batch processing. Following internet recommendations can lead to trouble. Make sure that it is fully understood before programming begins. Spring Batch is a pretty good open source framework for batch processing, and with very minimal customization can be used to build very efficient batch processes (relatively speaking).



Chuck Processing- Java Batch Cursor the Wrong Way

- Open a cursor with a starting key value
  - FETCH FIRST N ROWS ONLY into memory
    - N = chunk size
  - Close cursor
  - Process result set
  - Commit
- Open cursor again with advanced starting value
  - Materialized results could be processed hundreds or thousands of times!

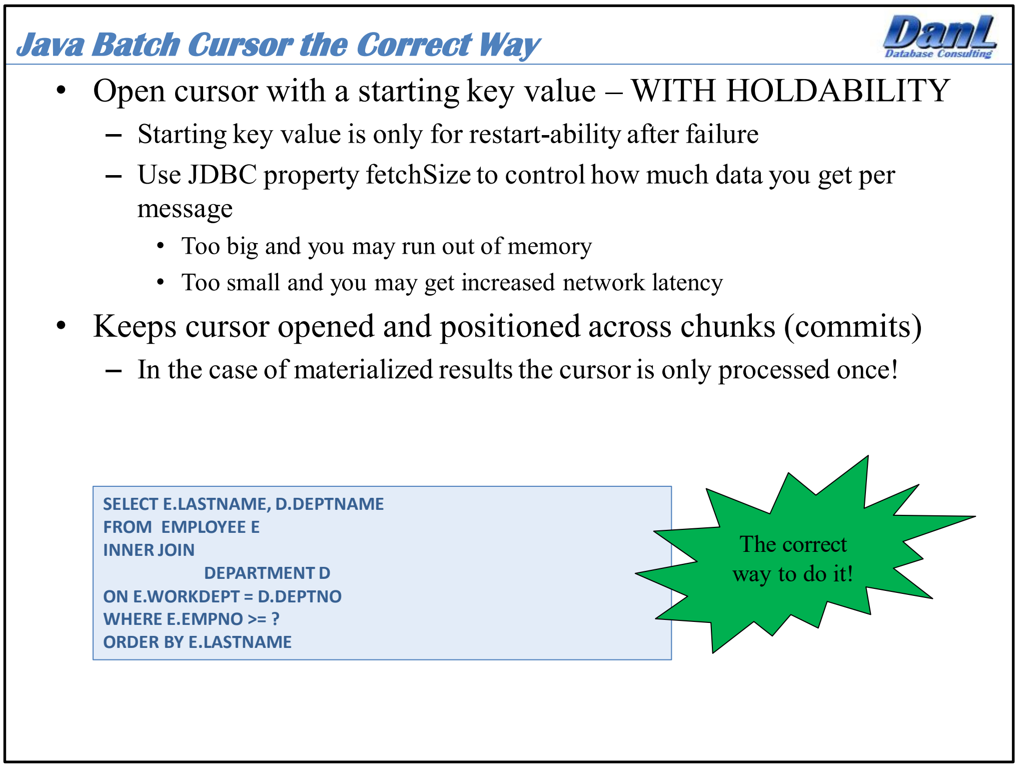
```
SELECT E.LASTNAME, D.DEPTNAME
FROM  EMPLOYEE E
INNER JOIN
                  DEPARTMENT D
ON E.WORKDEPT = D.DEPTNO
WHERE E.EMPNO >= ?
ORDER BY E.LASTNAME FETCH FIRST 10 ROWS ONLY
```

Chunk size is also 10

Common Practice – DO NOT DO THIS!

This is potentially an extremely inefficient way to do cursor driven batch processing that I have personally witnessed at several customer sites, and is also an often recommended technique promoted on the internet. In this situation the batch process is driven by a database cursor. The driving cursor result set is limited to the chunk size by utilizing a FETCH FIRST clause. The rows are processed, and then the chunk is committed. The last identifier used in the previous chunk is then used to drive the cursor for the next chunk. The process repeats until the cursor finally returns no data, and then the batch process ends. This is bad for two reasons. First, multiple cursors are being sent to Db2 when only one is really necessary. Second, if the driving cursor results in a materialized result inside Db2 then performance can be seriously compromised. In that situation Db2 is reading all of the data that qualifies, putting that into a workfile, returning only a subset of the data and throwing the rest away. Then the process commits, and the whole thing starts again. Extremely inefficient.

DO NOT DO THIS!



Java Batch Cursor the Correct Way

- Open cursor with a starting key value – WITH HOLDABILITY
  - Starting key value is only for restart-ability after failure
  - Use JDBC property fetchSize to control how much data you get per message
    - Too big and you may run out of memory
    - Too small and you may get increased network latency
- Keeps cursor opened and positioned across chunks (commits)
  - In the case of materialized results the cursor is only processed once!

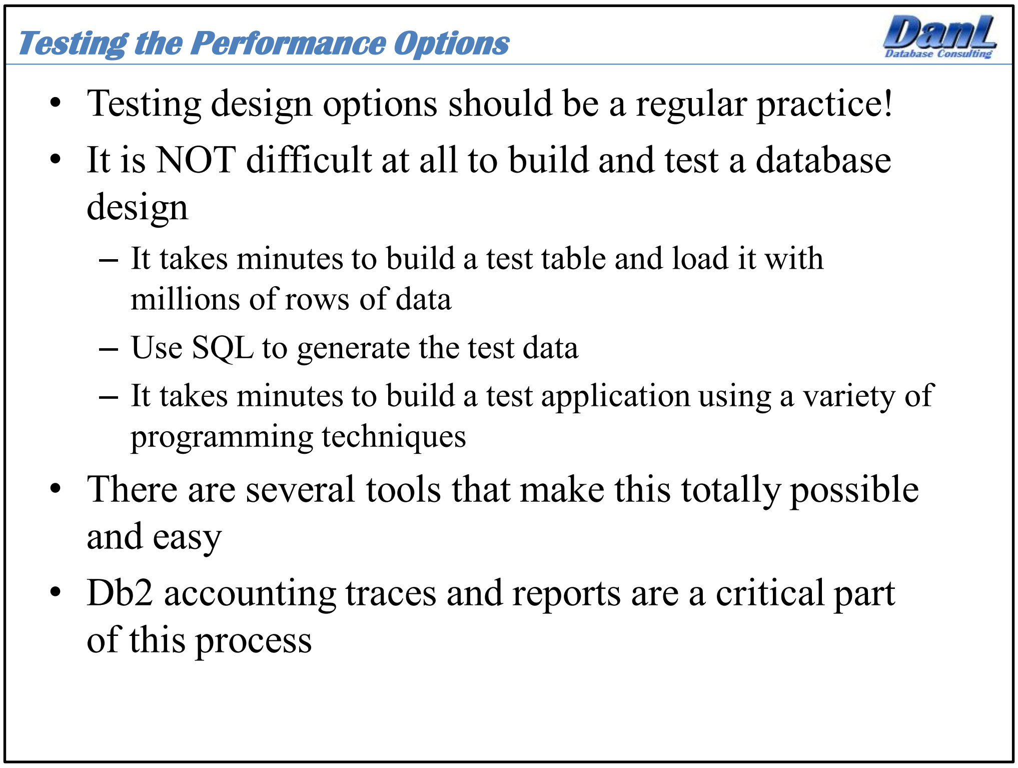
```
SELECT E.LASTNAME, D.DEPTNAME
FROM  EMPLOYEE E
INNER JOIN
              DEPARTMENT D
ON E.WORKDEPT = D.DEPTNO
WHERE E.EMPNO >= ?
ORDER BY E.LASTNAME
```

The correct way to do it!

The correct way to do Java batch processing when using a driving cursor is to keep that driving cursor open for the duration of the batch job. You can use the resultSetHoldability property of the IBM Data Server Driver to keep cursors open across a commit. In this way the driving batch cursor is only processed once during the entire batch, and chunks can be committed without closing the cursor.

Another important IBM Data Server Driver property to consider is the fetchSize, which controls how often the Db2 client goes to the server to get data by controlling the number of rows that are returned for a database fetch operation. This is different than an application fetch. An application may fetch data that is already in the client cache, and the client will only go to the server and fetch an additional number of rows when the cache is exhausted. This manages the message traffic to the database server across the network. By default the client will fetch the entire result set into memory so that may be impractical due to memory limitations on the client. So, it is more appropriate to set the fetchSize to something that balances client memory with network latency, as well as an appropriate chunk size that balances concurrency with performance. Committing too often may present performance problems and committing too infrequently may introduce locking and concurrency problems.

If using a flat file to drive a batch process it can help if that file is sorted in a sequence that matches the cluster of the tables.



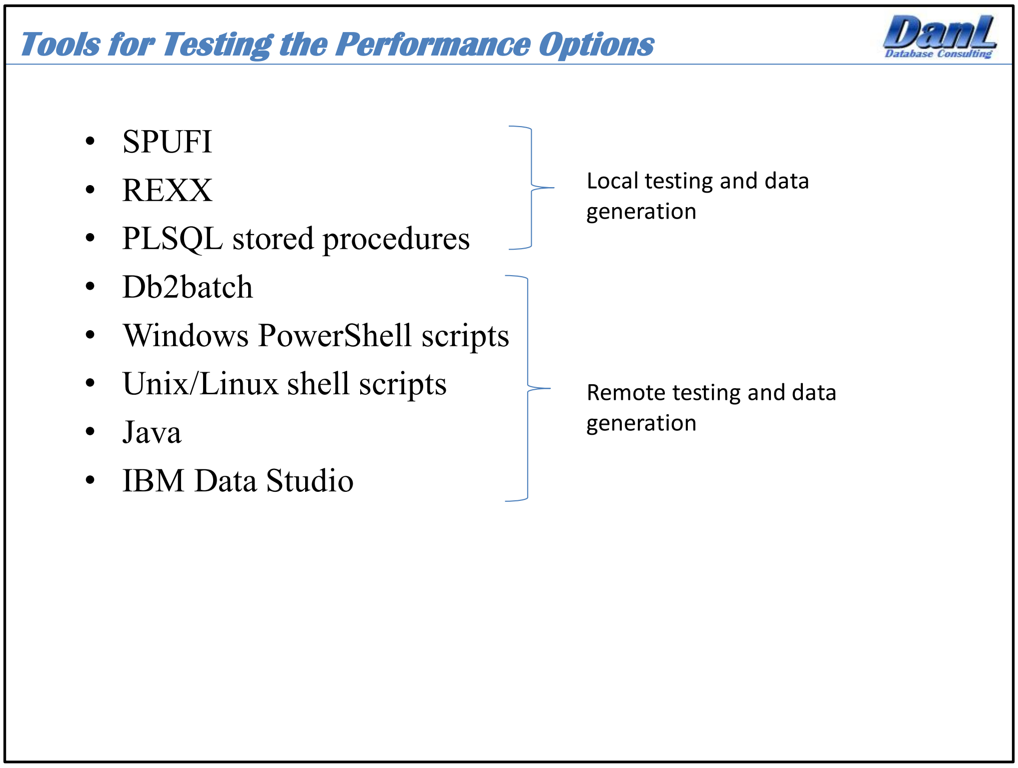
Testing the Performance Options

- Testing design options should be a regular practice!
- It is NOT difficult at all to build and test a database design
  - It takes minutes to build a test table and load it with millions of rows of data
  - Use SQL to generate the test data
  - It takes minutes to build a test application using a variety of programming techniques
- There are several tools that make this totally possible and easy
- Db2 accounting traces and reports are a critical part of this process

After 30+ years there are few things that people can do or say during my day to day DBA activities that upset me. However, there is still one thing that can get me to walk out of a meeting! That one thing is the meeting in which people sit around and debate….” I think it will work like this”, “That design will never perform properly”, “you’ll have to denormalize the initial design”. Overthinking and trying to predict Db2 performance is a slippery slope, and so rather than “thinking” about Db2 performance why not have the database tell you how it will perform. Set up some proof of concept tests!

The accounting report plays an extremely valuable role in monitoring these tests. Contrary to what might be popular belief, it takes only a modest effort to set up some tables and populate them with test data. There are a variety of testing tools at your disposal; REXX, SPUFI, DSNTEP2, db2batch, and IBM Data Studio, to name a few. Run a test, produce an accounting report, change something, repeat. No more guessing as Db2 then tells you the best design choice for your application and database. I have done these dozens, if not more, times during my career.

IBM Data Studio uses a JDBC type 4 connection to Db2. This makes it easy to test JDBC connection properties, such as currentPackageSet, clientApplcompat, ClientUser, statementConcentrator, just to name a few.



Tools for Testing the Performance Options

- SPUFI
- REXX
- PLSQL stored procedures

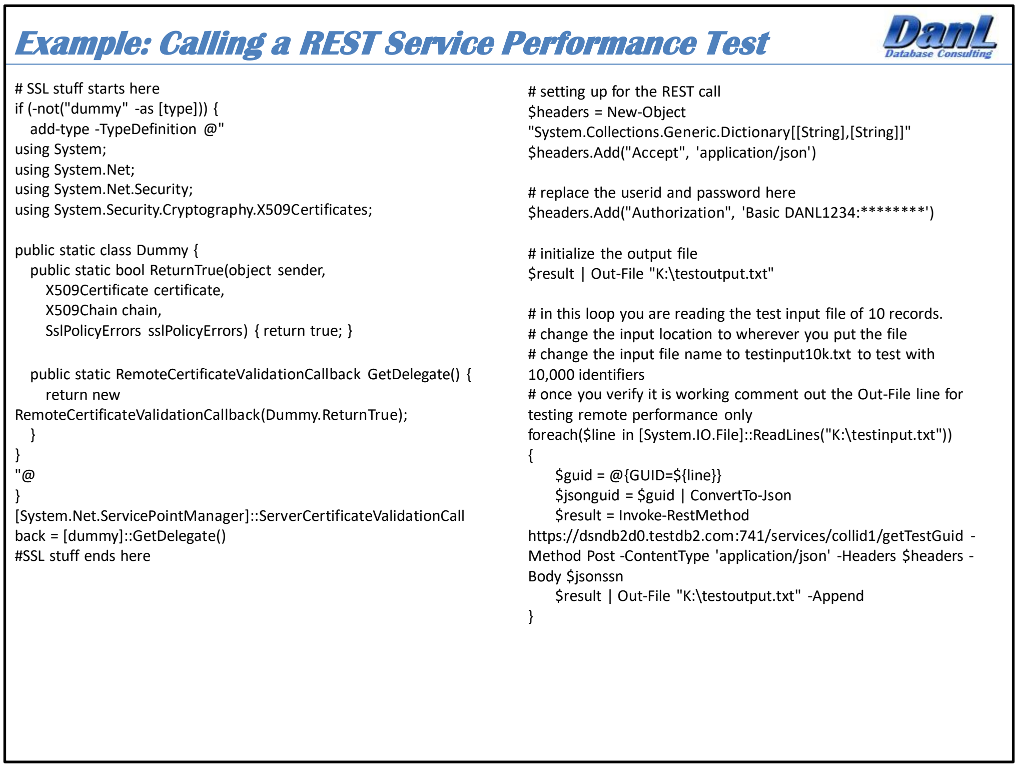
Local testing and data generation

- Db2batch
- Windows PowerShell scripts
- Unix/Linux shell scripts
- Java
- IBM Data Studio

Remote testing and data generation

Whatever the situation there is a no cost tool for the job. I have and still use every single one of these tools to test design and tuning ideas. There are plenty of examples to be found on the internet, and a decent combination of testing and monitoring will quickly point out the best design direction! I have been taking advantage of Windows PowerShell recently with great success! There is a wealth of information available online, and it is really powerful and easy to learn. Making a test table and generating data to load into the table using SQL and the RAND() function is fast and easy. Generating test data is just as easy. Consider a statement such as this:

```
SELECT CUSTNO FROM (SELECT CUSTNO, RAND() AS DANL FROM CUSTOMER_TABLE) AS
TAB1
ORDER BY DANL FETCH FIRST 10000 ROWS ONLY WITH UR;
```



Example: Calling a REST Service Performance Test

```
# SSL stuff starts here
if (-not("dummy" -as [type])) {
   add-type -TypeDefinition @"
using System;
using System.Net;
using System.Net.Security;
using System.Security.Cryptography.X509Certificates;

public static class Dummy {
   public static bool ReturnTrue(object sender,
      X509Certificate certificate,
      X509Chain chain,
      SslPolicyErrors sslPolicyErrors) { return true; }

   public static RemoteCertificateValidationCallback GetDelegate() {
      return new
RemoteCertificateValidationCallback(Dummy.ReturnTrue);
   }
}
"@
}
[System.Net.ServicePointManager]::ServerCertificateValidationCall
back = [dummy]::GetDelegate()
#SSL stuff ends here
```

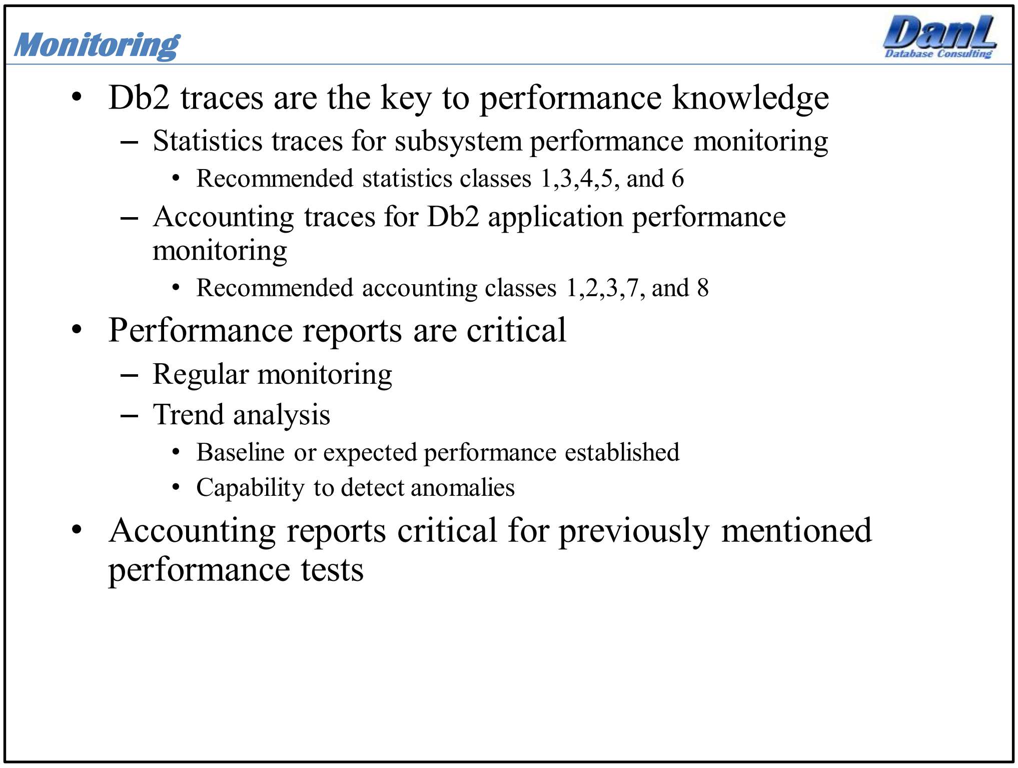
```
# setting up for the REST call
$headers = New-Object
"System.Collections.Generic.Dictionary[[String],[String]]"
$headers.Add("Accept", 'application/json')

# replace the userid and password here
$headers.Add("Authorization", 'Basic DANL1234:********')

# initialize the output file
$result | Out-File "K:\testoutput.txt"

# in this loop you are reading the test input file of 10 records.
# change the input location to wherever you put the file
# change the input file name to testinput10k.txt to test with
10,000 identifiers
# once you verify it is working comment out the Out-File line for
testing remote performance only
foreach($line in [System.IO.File]::ReadLines("K:\testinput.txt"))
{
     $guid = @{GUID=${line}}
     $jsonguid = $guid | ConvertTo-Json
     $result = Invoke-RestMethod
https://dsndb2d0.testdb2.com:741/services/collid1/getTestGuid -
Method Post -ContentType 'application/json' -Headers $headers -
Body $jsonssn
     $result | Out-File "K:\testoutput.txt" -Append
}
```

I recently coded a PowerShell script that reads in a file of random identifiers and invokes a REST service. It's purpose was to test performance. Some of it was borrowed from the internet! I had to split the script into two pieces to fit it on this slide and still have it be readable. Not shown is the SQL that generated the random key input files. However, an example of that is on the previous slide notes.



Monitoring

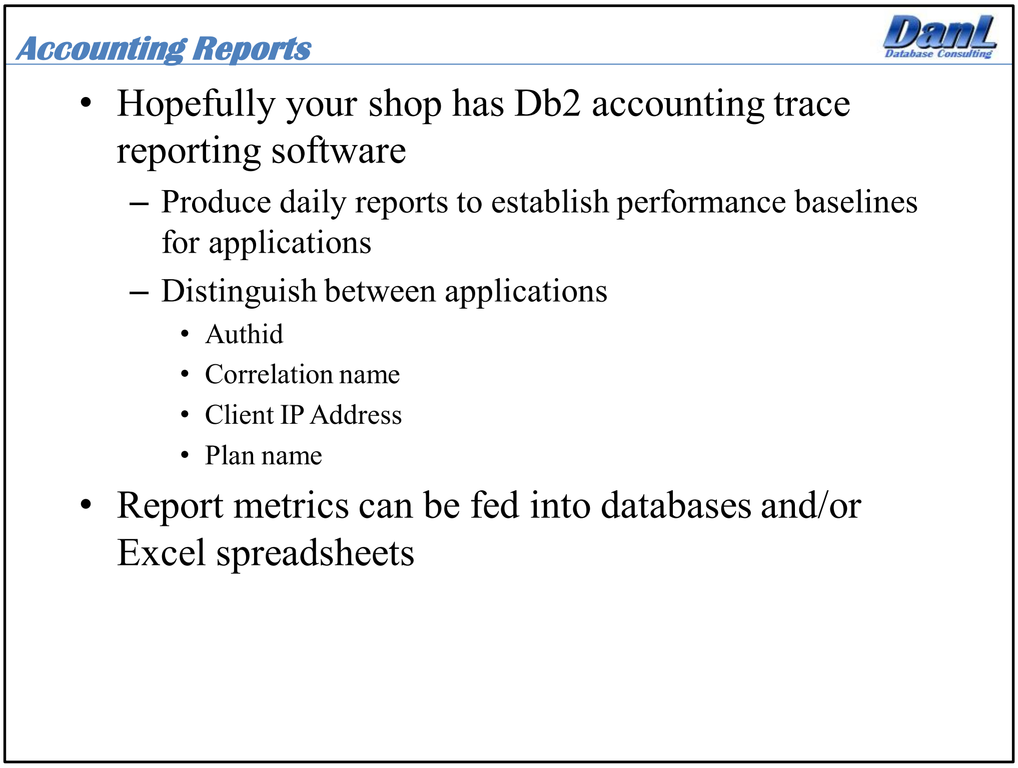
- Db2 traces are the key to performance knowledge
  - Statistics traces for subsystem performance monitoring
    - Recommended statistics classes 1,3,4,5, and 6
  - Accounting traces for Db2 application performance monitoring
    - Recommended accounting classes 1,2,3,7, and 8
- Performance reports are critical
  - Regular monitoring
  - Trend analysis
    - Baseline or expected performance established
    - Capability to detect anomalies
- Accounting reports critical for previously mentioned performance tests

Db2 has an extensive built-in tracing facility that is extremely important when analyzing the impact of various settings, as well as overall performance. There are several types of Db2 traces.

- Statistics trace, which collects performance metrics at a subsystem level.
- Accounting trace, which collects thread level performance metrics and is really an essential key to understanding and improving application performance.
- Performance trace, which is not set by default and can be used to track specific threads or sets of threads at an extremely detailed level.
- Audit trace, which collects information about Db2 security controls and can be used to ensure that data access is allowed only for authorized purposes.
- Monitor trace, which enables attached monitor programs to access Db2 trace data through calls to the instrumentation facility interface (IFI).

Traces can be set upon Db2 startup via the system installation parameters, or they can be started via Db2 commands. Typically Audit, Monitor, and Performance traces are set on demand and for specialized purposes. Statistics and accounting traces are typically set at Db2 startup. The most common settings are statistics classes 1,3,4,5, and 6 and accounting classes 1,2,3,7, and 8.

All of these traces are important and worthy of an understanding, but the focus of this article will be on the accounting trace. You can find descriptions of trace records in prefix.SDSNIVPD(DSNWMSGS).



Accounting Reports

DanL Database Consulting

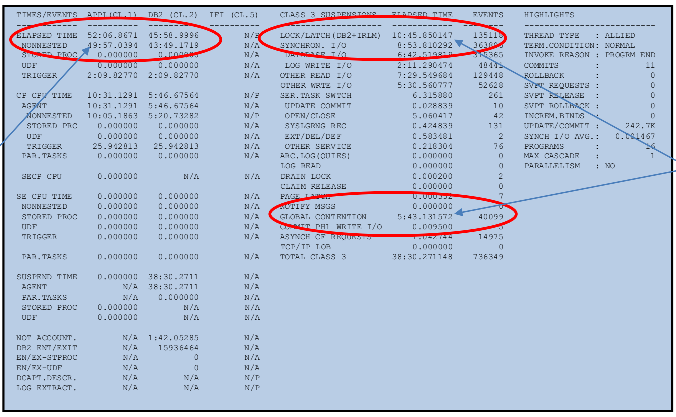
- Hopefully your shop has Db2 accounting trace reporting software
  - Produce daily reports to establish performance baselines for applications
  - Distinguish between applications
    - Authid
    - Correlation name
    - Client IP Address
    - Plan name
- Report metrics can be fed into databases and/or Excel spreadsheets

The accounting trace collects thread level performance metrics and is really an essential key to understanding and improving application performance. The accounting trace records produced are typically directed towards system management facility (SMF) datasets and hopefully your friendly system programmer is externalizing these SMF records for Db2 subsystems into separate datasets for analysis. The accounting trace data these days is typically externalized upon the termination of each individual thread. So, you get one record per thread. This means that for large scale batch jobs, CICS transactions that use protected threads, distributed threads with the ACCUMAC subsystem parameter (this parameter determines whether DB2® accounting data is to be accumulated by the user for DDF and RRSAF threads) set at something other than NO, or high performance DBATs, you can get multiple transactions bundled together into the same accounting trace record. By default, Db2 only starts accounting trace class 1, which I believe is seriously not enough to accurately diagnose and track the performance of applications. My recommended accounting trace classes to set are 1,2,3,7, and 8.

Class 1 = Total elapsed time and CPU used by a thread while connected to Db2 at the plan level.
Class 2 = Total elapsed time and CPU used by a thread within Db2 at the plan level. This is a subset of class 1. The elapsed time is broken into suspension time and CPU time.
Class 3 = Total time waiting for resources in Db2 at the plan level. This is a subset of the class 2 time and is equal to the class 2 suspension time.
Class 7 and 8 = These are, respectively, the class 2 and 3 times divided between packages utilized during thread execution.

Accounting Report

DB2 accounting report for a batch application suddenly shows excessive elapsed time way in excess of normal!



```
TIMES/EVENTS  APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)   CLASS 3 SUSPENSIONS   ELAPSED TIME    EVENTS   HIGHLIGHTS
------------  ----------  ----------  ----------   --------------------  ------------  --------   --------------------------
ELAPSED TIME  52:06.8671  45:58.9996         N/P   LOCK/LATCH(DB2+IRLM)  10:45.850147    135118   THREAD TYPE    : ALLIED
 NONNESTED    49:57.0394  43:49.1719         N/A   SYNCHRON. I/O          8:53.810292    363806   TERM.CONDITION: NORMAL
 STORED PROC    0.000000    0.000000         N/A    DATABASE I/O          6:42.519819    315365   INVOKE REASON : PROGRM END
 UDF            0.000000    0.000000         N/A    LOG WRITE I/O         2:11.290474     48441   COMMITS       :          11
 TRIGGER      2:09.82770  2:09.82770         N/A   OTHER READ I/O         7:29.549684    129448   ROLLBACK      :           0
                                                   OTHER WRTE I/O         5:30.560777     52628   SVPT REQUESTS :           0
CP CPU TIME   10:31.1291  5:46.67564         N/P   SER.TASK SWTCH            6.315880       261   SVPT RELEASE  :           0
 AGENT        10:31.1291  5:46.67564         N/A    UPDATE COMMIT            0.028839        10   SVPT ROLLBACK :           0
  NONNESTED   10:05.1863  5:20.73282         N/P    OPEN/CLOSE               5.060417        42   INCREM.BINDS  :           0
  STORED PRC    0.000000    0.000000         N/A    SYSLGRNG REC             0.424839       131   UPDATE/COMMIT :      242.7K
  UDF           0.000000    0.000000         N/A    EXT/DEL/DEF              0.583481         2   SYNCH I/O AVG.:    0.001467
  TRIGGER      25.942813   25.942813         N/A    OTHER SERVICE            0.218304        76   PROGRAMS      :          16
 PAR.TASKS      0.000000    0.000000         N/A   ARC.LOG(QUIES)            0.000000         0   MAX CASCADE   :           1
                                                   LOG READ                  0.000000         0   PARALLELISM   : NO
 SECP CPU       0.000000         N/A         N/A   DRAIN LOCK                0.000200         2
                                                   CLAIM RELEASE             0.000000         0
SE CPU TIME     0.000000    0.000000         N/A   PAGE LATCH                0.000352         7
 NONNESTED      0.000000    0.000000         N/A   NOTIFY MSGS               0.000000         0
 STORED PROC    0.000000    0.000000         N/A   GLOBAL CONTENTION      5:43.131572     40099
 UDF            0.000000    0.000000         N/A   COMMIT PH1 WRITE I/O      0.009500         5
 TRIGGER        0.000000    0.000000         N/A   ASYNCH CF REQUESTS        1.042744     14975
                                                   TCP/IP LOB                0.000000         0
 PAR.TASKS      0.000000    0.000000         N/A   TOTAL CLASS 3         38:30.271148    736349

SUSPEND TIME    0.000000  38:30.2711         N/A
 AGENT               N/A  38:30.2711         N/A
 PAR.TASKS           N/A    0.000000         N/A
 STORED PROC    0.000000         N/A         N/A
 UDF            0.000000         N/A         N/A

NOT ACCOUNT.         N/A  1:42.05285         N/A
DB2 ENT/EXIT         N/A    15936464         N/A
EN/EX-STPROC         N/A           0         N/A
EN/EX-UDF            N/A           0         N/A
DCAPT.DESCR.         N/A         N/A         N/P
LOG EXTRACT.         N/A         N/A         N/P
```

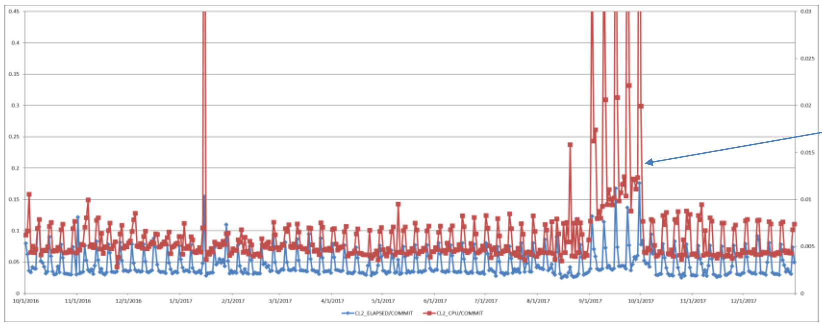
Abnormal level of lock/latch and global contention time compared to a "normal" run indicates there is a problem!

Here is an example of an accounting report that looked different than previous accounting reports for the application being monitored. The deviation from the norm manifested as excessive elapsed time, lock/latch time, and global contention time. It turns out that the cache size for a sequence object was set to the default of 20, and needed to be increased. The contention indicated in the report was actually against the Db2 system catalog as Db2 had to go there to get a next set of values for the sequence object. This was happening across a data sharing group and thus the global contention.

More on the accounting report can be found here:
https://www.db2expert.com/db2expert/the-importance-of-db2-for-z-os-accounting-traces-and-reports/

Accounting Database

- Db2 accounting trace data can be loaded into a database
- Database queries can then be used to track performance



Something went wrong here and needed to be investigated. Turns out some new untuned SQL was installed in the application.

There are several tools that let you convert SMF accounting trace data into files that can be loaded into tables. To save on storage costs I typically load the data into Db2 for LUW tables on a department server, or my own workstation. I can then run regular queries that provide health checks for applications and place the results into Microsoft Excel workbooks to create graphs. Hint: managers like graphs!

DanL
Database Consulting

Db2 For z/OS Ultra High Performance and Tuning

Daniel L Luksetich

DanL Database Consulting

danl@db2expert.com

Dan Luksetich is a Db2 DBA consultant. He works as a DBA, application architect, presenter, author, and teacher. Dan has been in the information technology business for over 36 years, and has worked with Db2 for over 31 years. He has been an application programmer, Db2 system programmer, Db2 DBA, and Db2 application architect. His experience includes major implementations on z/OS, AIX, i Series, Windows, and Linux environments. His industry experience includes retail, banking, fraud detection, analytics, government, and utility. He specializes in highly available, high-volume transaction processing against very large database systems.

Dan's experience includes, but is not limited to:

- Application design and architecture
- Business analytics
- SQL consulting and education
- SQL coding and application programming
- Database administration
- Complex SQL
- SQL tuning
- Db2 performance audits
- Replication
- Disaster recovery
- Stored procedures, user-defined functions, and triggers
- Db2 REST services

Dan likes beer and is a Certified Cicerone!