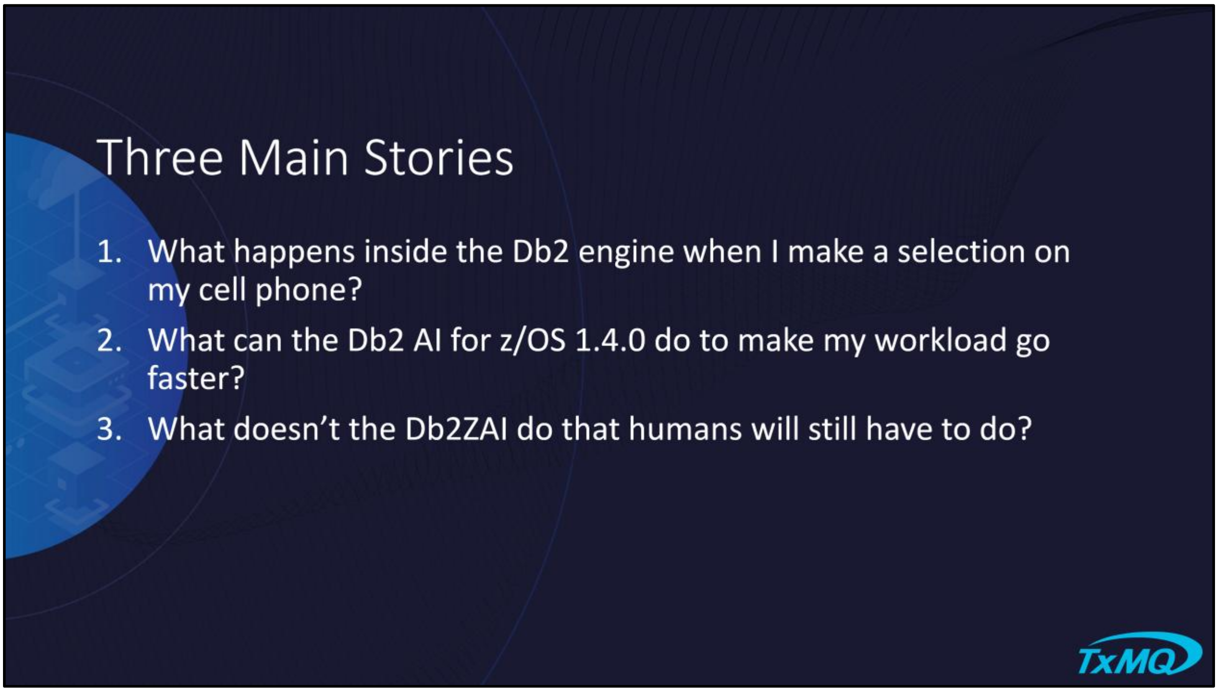
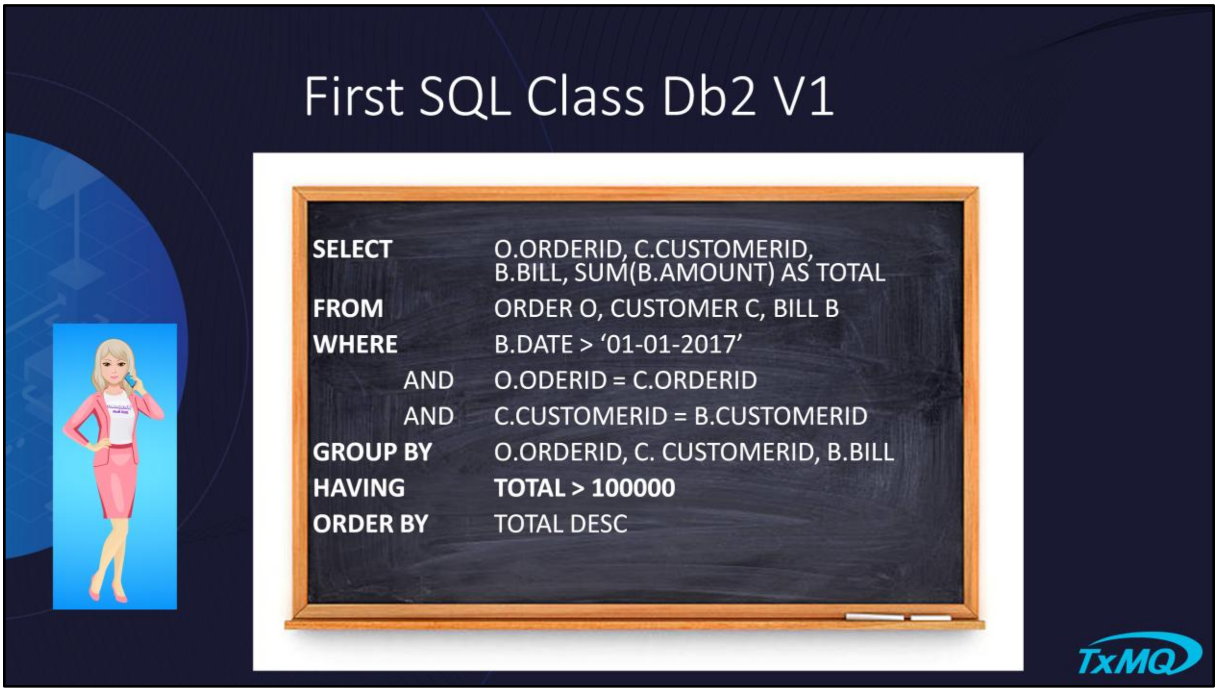
TxMQ works with companies across the globe to innovate and transform their businesses. To excel in their industries through the optimal use of people, process and technologies. **We do this leveraging subject matter experts in integration, cloud architecture, process and analytics. I am honored to join this brilliant team as the lead in their Db2 Consulting service.**

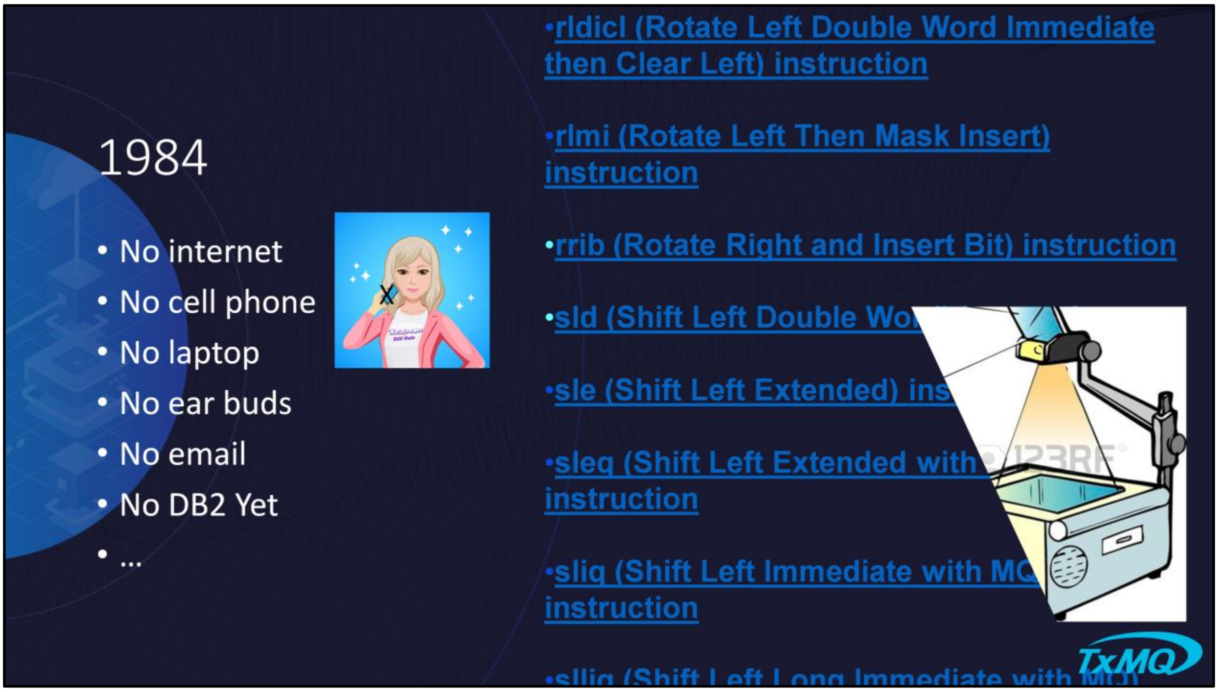
Sheryl is the Senior Db2 Z Consultant at TxMQ. Previously she worked for, BMC, IBM, Sheryl M. Larsen, Inc., and Platinum Technology (now Broadcom). She is known for her extensive expertise in SQL Tuning and has performed detailed Db2 Performance Health Checks for many Fortune 500 clients. Sheryl has over 30 years' experience in Db2, has published many articles, white papers and co-authored a book, Db2 Answers, Osborne-McGraw-Hill, 1999. Currently she is the President of the Midwest Db2 User Group, an IBM Z Champion, and a member of the Northern Illinois University Computer Science Alumni Council.



I hope to increase your SQL tuning confidence. If not, rinse and repeat (maybe a little slower the second time). For English is a second language attendees, please stop to take notes and translate when needed. All attendees should view the NotesPages as they watch the video to gain a broader knowledge on the subject.

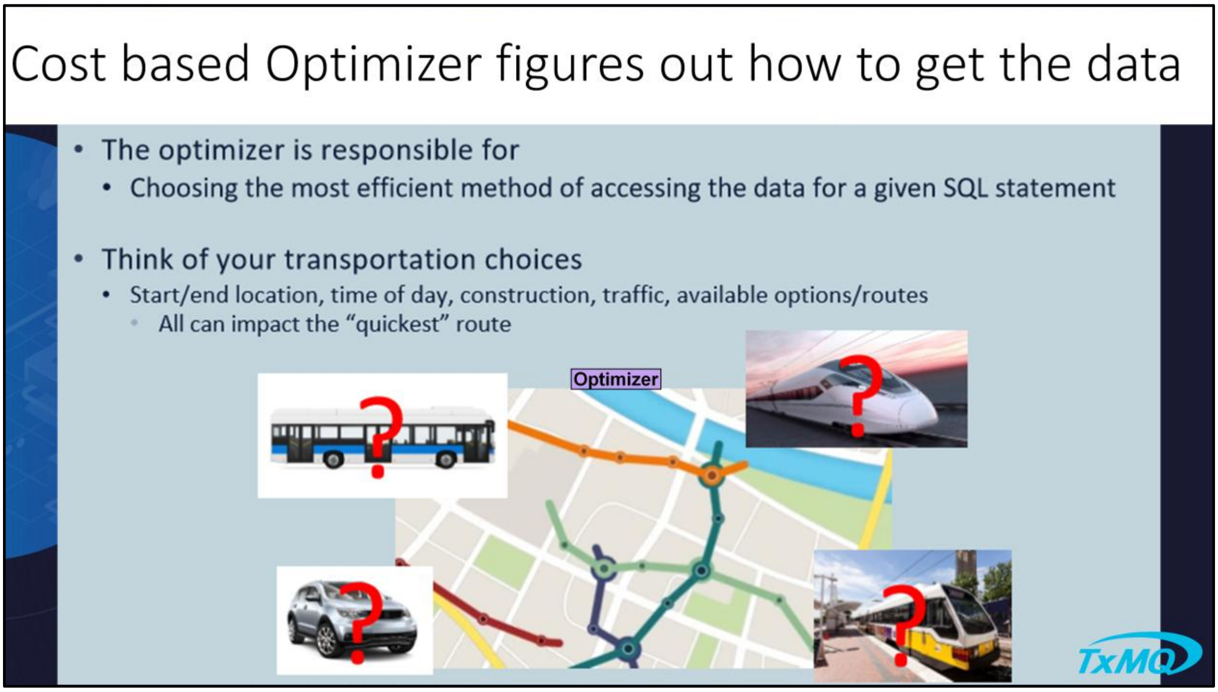


Gone are the days of manually tuning queries. NOT SO FAST! As smart as the internal Virtual Data Scientist, inside Db2ZAI is, there are many techniques that it cannot apply. Query re-write lives on in an Agile work environment! Human eyes are still required on projects demanding *only* high-performance SQL moves on into production. Sheryl will cover all kinds of techniques that are still best practice rules to follow. Perfect for newbies to Db2 for z/OS due to all the pictures used to demonstrate techniques, as well as oldies, who need a refresher.

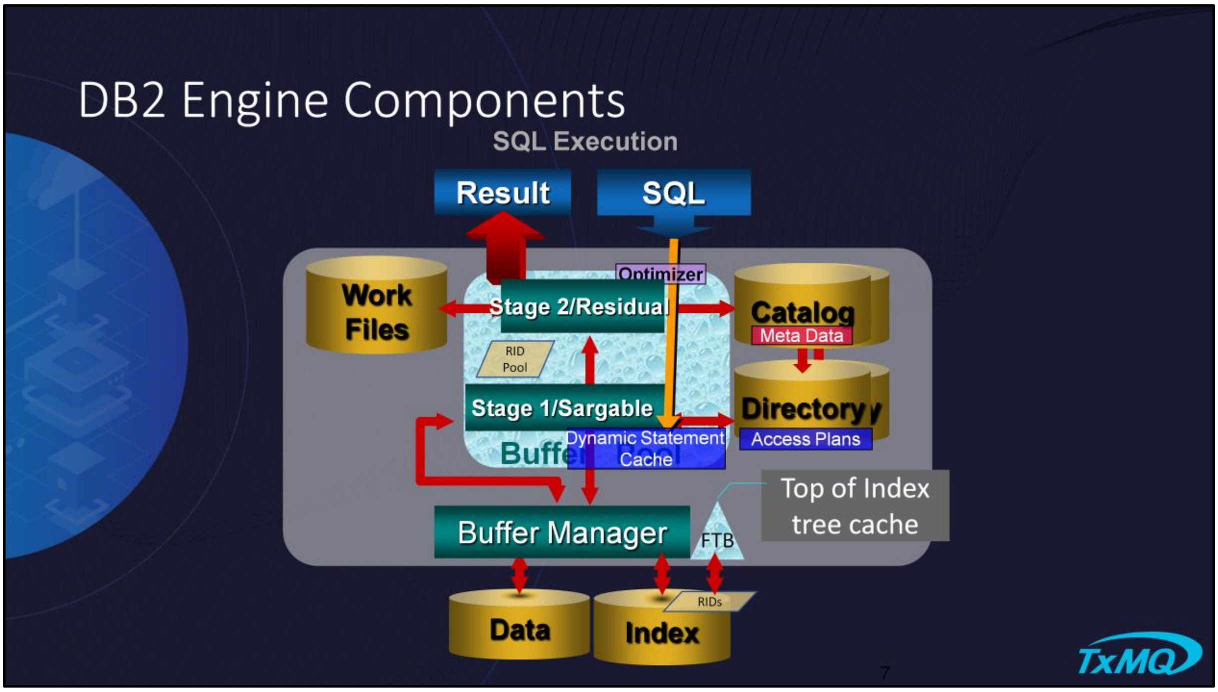


DB2 Beta came out end of 1984. DB2 Version 1 Release 1 for MVS became generally available (GA) in **April 1985** and DB2 Version 1 Release 2 became GA in **March 1986** -- only a month after it was announced.

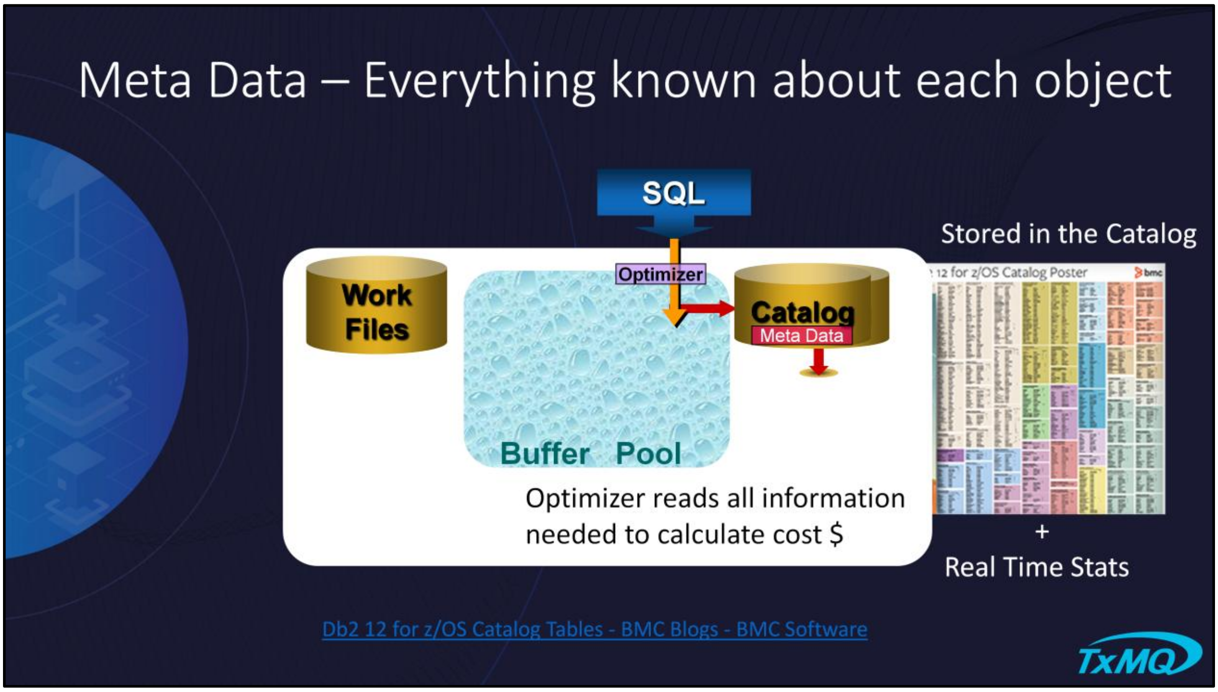
Check current IBM SQL Reference for a complete list:
[Db2 12 for z/OS: SQL Reference (ibm.com)]



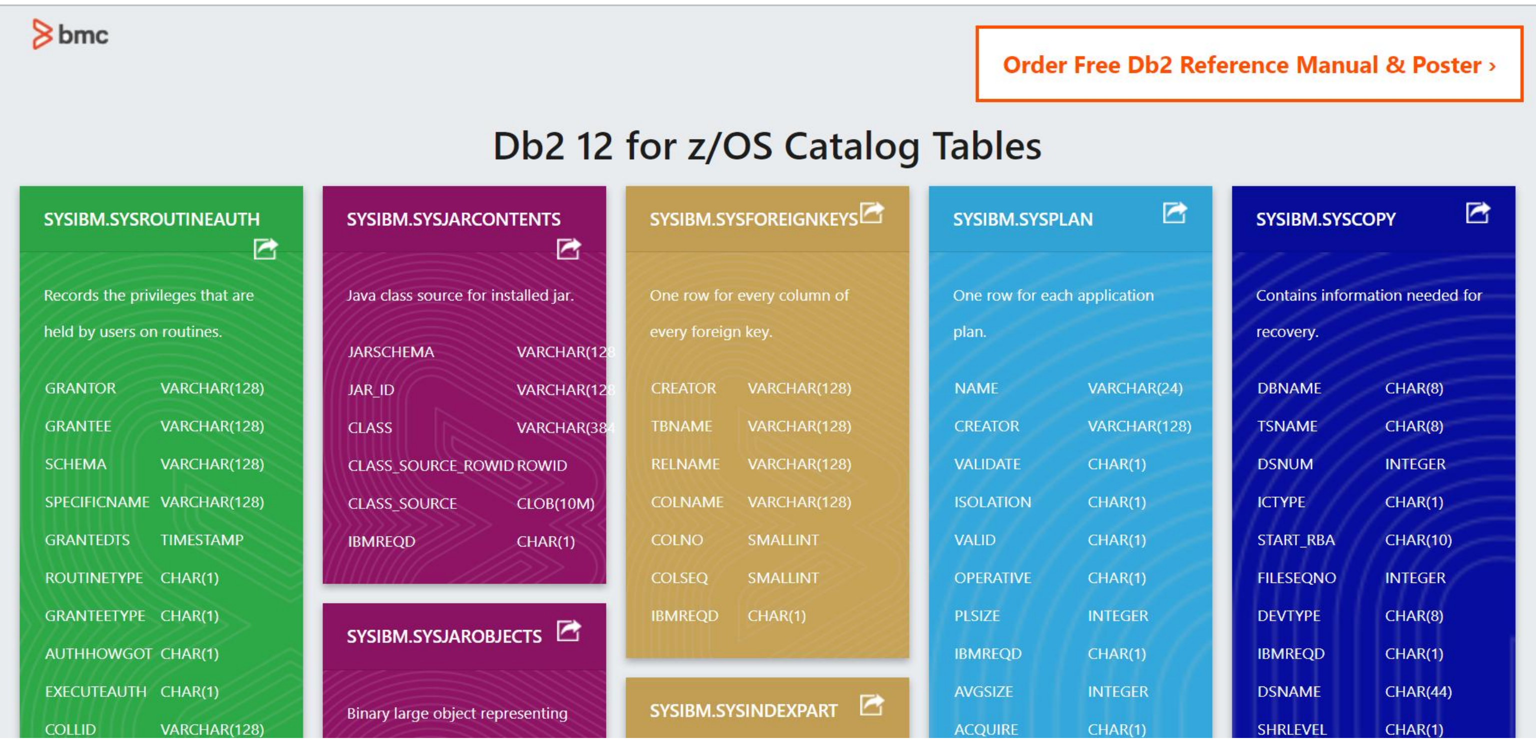
The DB2 Optimizer is a cost-based optimizer, which means it calculates the cost of multiple access paths (in a unit called Timerons), and then chooses the cheapest access path. One of the advantages of the SQL language is that it means we don't have to know where a particular piece of data lives on disk or memory.

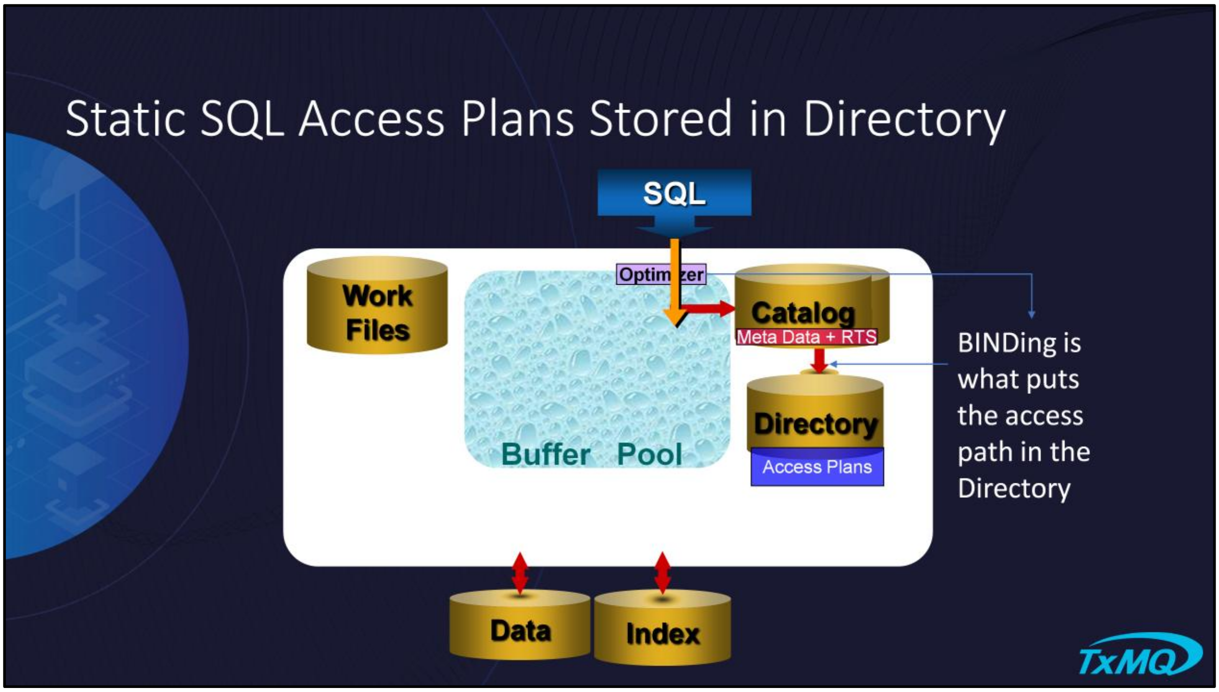


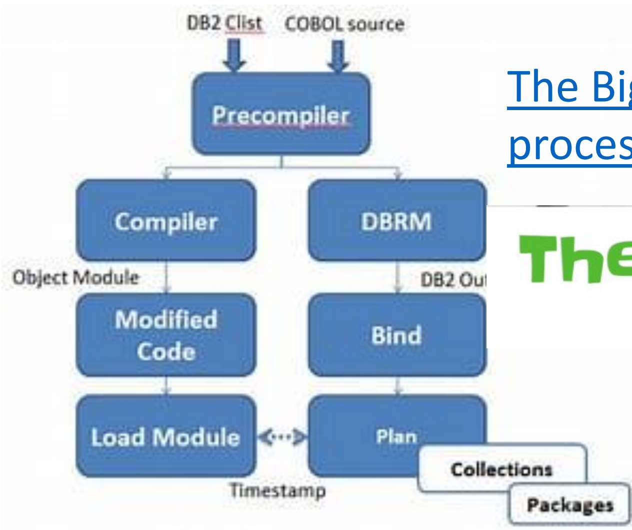
This is the flow of the Db2 Engine components. The following slides go through each step, one at a time



A new interactive Catalog application from BMC:
[Db2 12 for z/OS Catalog Tables - BMC Blogs - BMC Software]



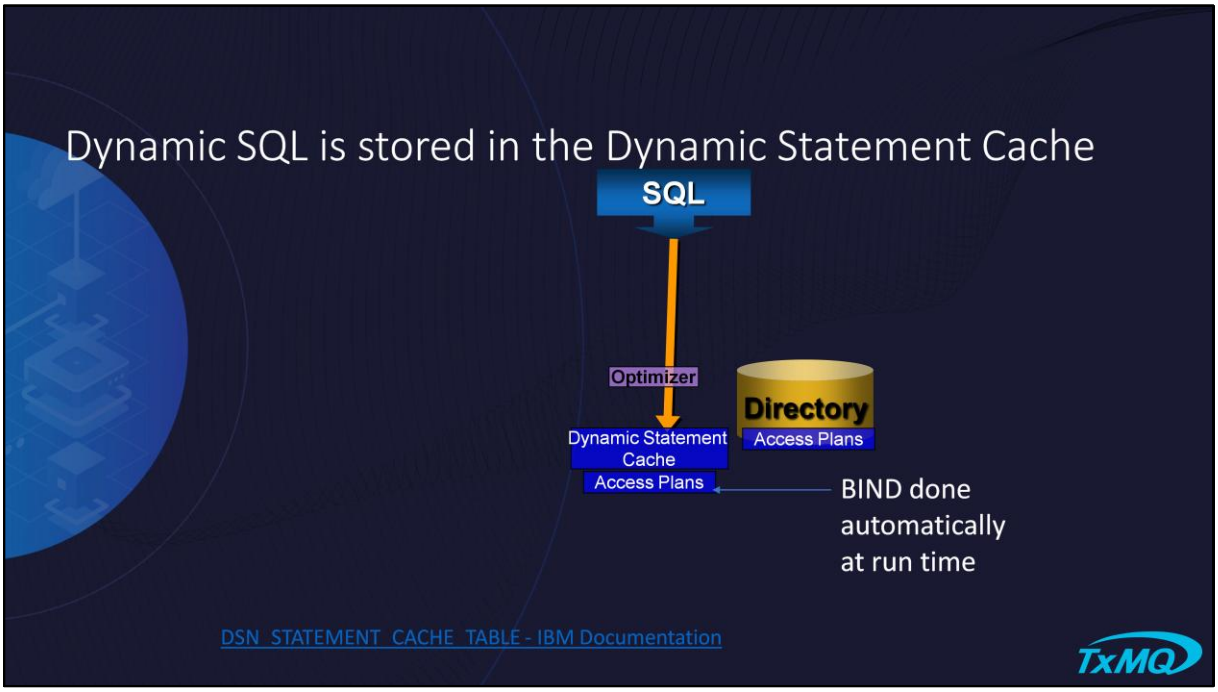




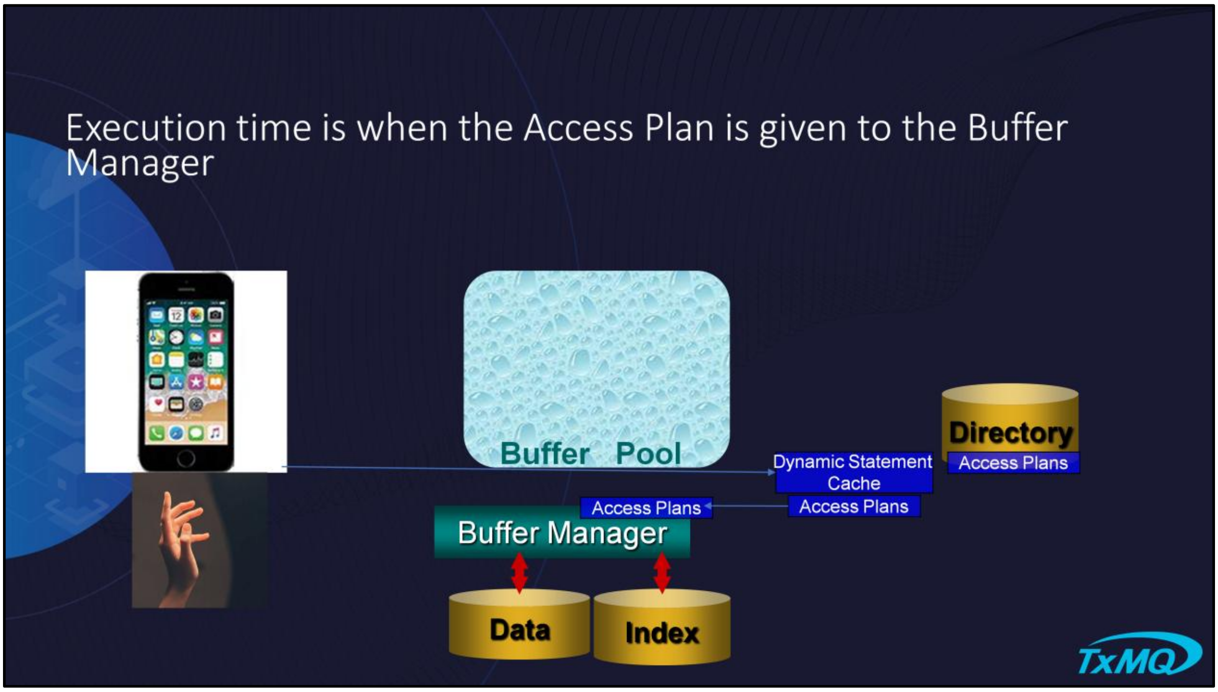
[The Big Old Mainframe: DB2 Bind process]

•Bind Process read DBRM which is created in precomplier stage and creates access path to read data.
•Access path along with consistency token is stored in DB2 catalog tables as a package.
•Every package is bound into package list or collection
•Collection name is specified by package parameter.
•A Collection is a group of Packages that are included in one or more Plans. The QUALIFIER parameter of the bind is used to direct the SQL to the specific set of DB2 objects (tables, views, aliases or synonyms) qualified by this name.
•Apart from building plans and packages, bind also validates:

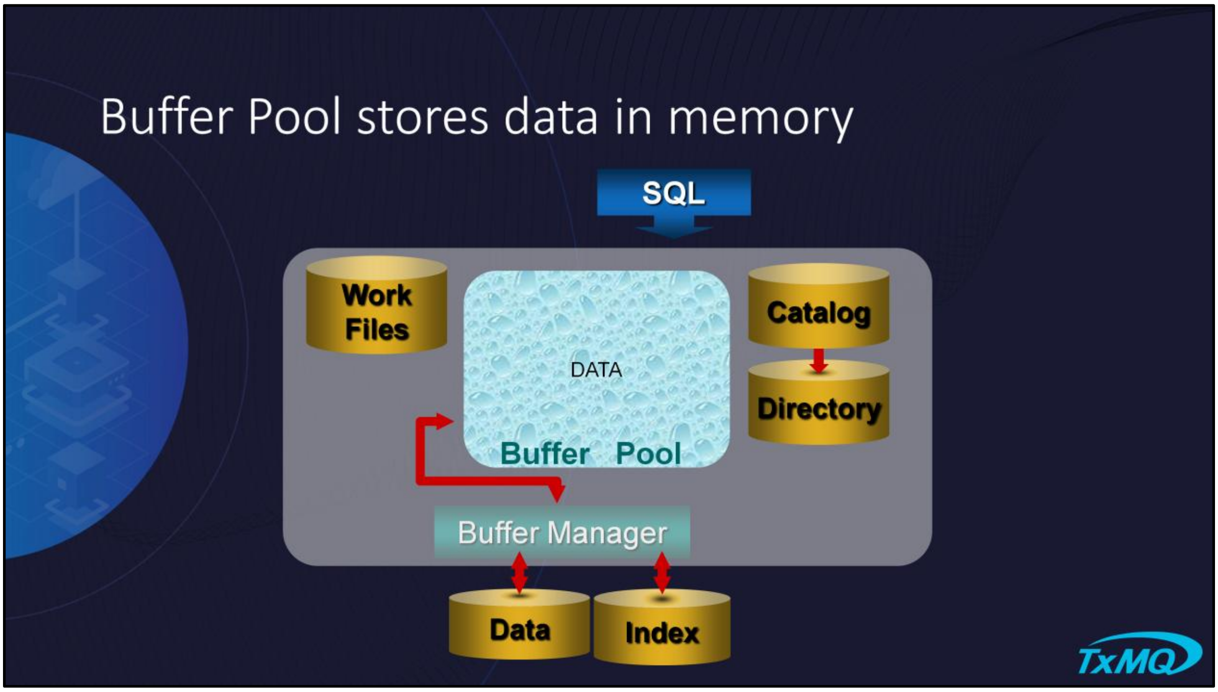
1. SQL statements using DB2 Catalog
2. Validates authorization id that if owner is allowed to perform bind process
3. Selects access path depending upon availability of indexes, table size etc.



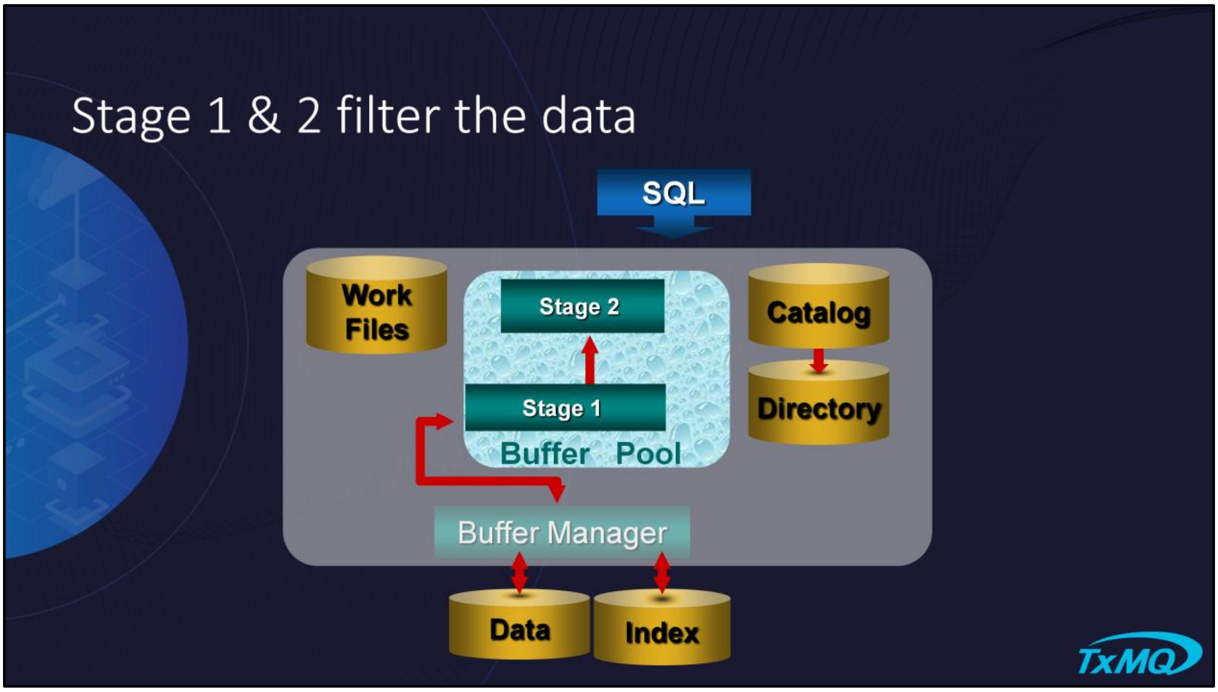
[DSN_STATEMENT_CACHE_TABLE - IBM Documentation]



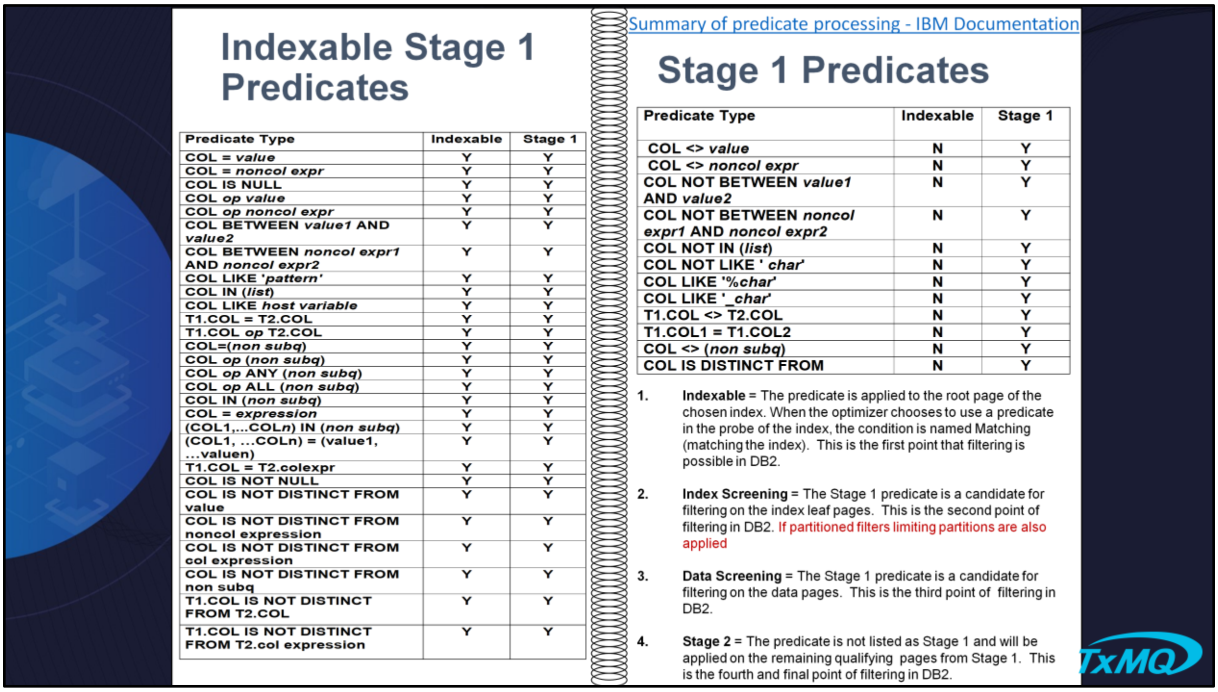
From an app on your phone, to the mainframe storing all the transaction data, your Access Plan is pulled into the Buffer Manager. This component follows the instructions to pull data from index, MQT, data into to Buffer Pool that is not already there to meet your query needs.



IBM® DB2® buffer pools are still a key resource for ensuring good performance. This has become increasingly important as the difference between processor speed and disk response time for a random access I/O widens in each new generation of processor. An IBM System z® processor can be configured with large amounts of storage, which if used wisely, can help compensate by using storage to avoid synchronous I/O.



Rows retrieved for a query go through two stages of processing. Certain predicates can be applied during the first stage of processing, whereas other cannot be applied until the second stage of processing. You can improve the performance of your queries by using predicates that can be applied during the first stage whenever possible.



[Summary of predicate processing - IBM Documentation]

Indexable Stage 1 Predicates

Predicate Type	Indexable	Stage 1
COL = *value*	Y	Y
COL = *noncol expr*	Y	Y
COL IS NULL	Y	Y
COL *op value*	Y	Y
COL *op noncol expr*	Y	Y
COL BETWEEN *value1* AND *value2*	Y	Y
COL BETWEEN *noncol expr1* AND *noncol expr2*	Y	Y
COL LIKE '*pattern*'	Y	Y
COL IN (*list*)	Y	Y
COL LIKE *host variable*	Y	Y
T1.COL = T2.COL	Y	Y
T1.COL *op* T2.COL	Y	Y
COL=(*non subq*)	Y	Y
COL *op* (*non subq*)	Y	Y
COL *op* ANY (*non subq*)	Y	Y
COL *op* ALL (*non subq*)	Y	Y
COL IN (*non subq*)	Y	Y
COL = *expression*	Y	Y
(COL1,...COL*n*) IN (*non subq*)	Y	Y
(COL1, ...COLn) = (value1, ...valuen)	Y	Y
T1.COL = T2.colexpr	Y	Y
COL IS NOT NULL	Y	Y
COL IS NOT DISTINCT FROM value	Y	Y
COL IS NOT DISTINCT FROM noncol expression	Y	Y
COL IS NOT DISTINCT FROM col expression	Y	Y
COL IS NOT DISTINCT FROM non subq	Y	Y
T1.COL IS NOT DISTINCT FROM T2.COL	Y	Y
T1.COL IS NOT DISTINCT FROM T2.col expression	Y	Y

Stage 1 Predicates

Predicate Type	Indexable	Stage 1
COL <> *value*	N	Y
COL <> *noncol expr*	N	Y
COL NOT BETWEEN *value1* AND *value2*	N	Y
COL NOT BETWEEN *noncol expr1* AND *noncol expr2*	N	Y
COL NOT IN (*list*)	N	Y
COL NOT LIKE ' *char*'	N	Y
COL LIKE '%*char*'	N	Y
COL LIKE '_*char*'	N	Y
T1.COL <> T2.COL	N	Y
T1.COL1 = T1.COL2	N	Y
COL <> (*non subq*)	N	Y
COL IS DISTINCT FROM	N	Y

1. **Indexable** = The predicate is applied to the root page of the chosen index. When the optimizer chooses to use a predicate in the probe of the index, the condition is named Matching (matching the index). This is the first point that filtering is possible in DB2.
2. **Index Screening** = The Stage 1 predicate is a candidate for filtering on the index leaf pages. This is the second point of filtering in DB2. If partitioned filters limiting partitions are also applied
3. **Data Screening** = The Stage 1 predicate is a candidate for filtering on the data pages. This is the third point of filtering in DB2.
4. **Stage 2** = The predicate is not listed as Stage 1 and will be applied on the remaining qualifying pages from Stage 1. This is the fourth and final point of filtering in DB2.

There are total of 54 with 33 notes associated to the current list. Find the updates here:
[Summary of predicate processing - IBM Documentation]

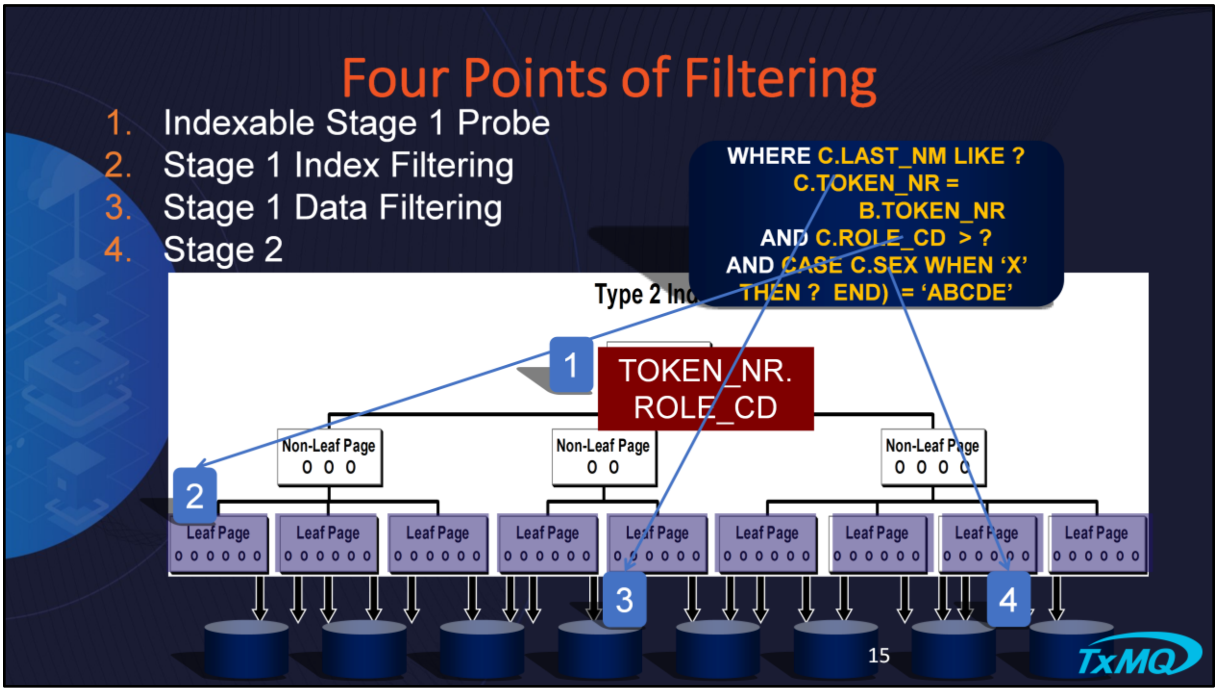
Indexable and stage 1 predicates [31]The following predicates might be evaluated by matching index access, during index screening, or after data page access during stage

There are 42 of them shown in slide 18 notes.

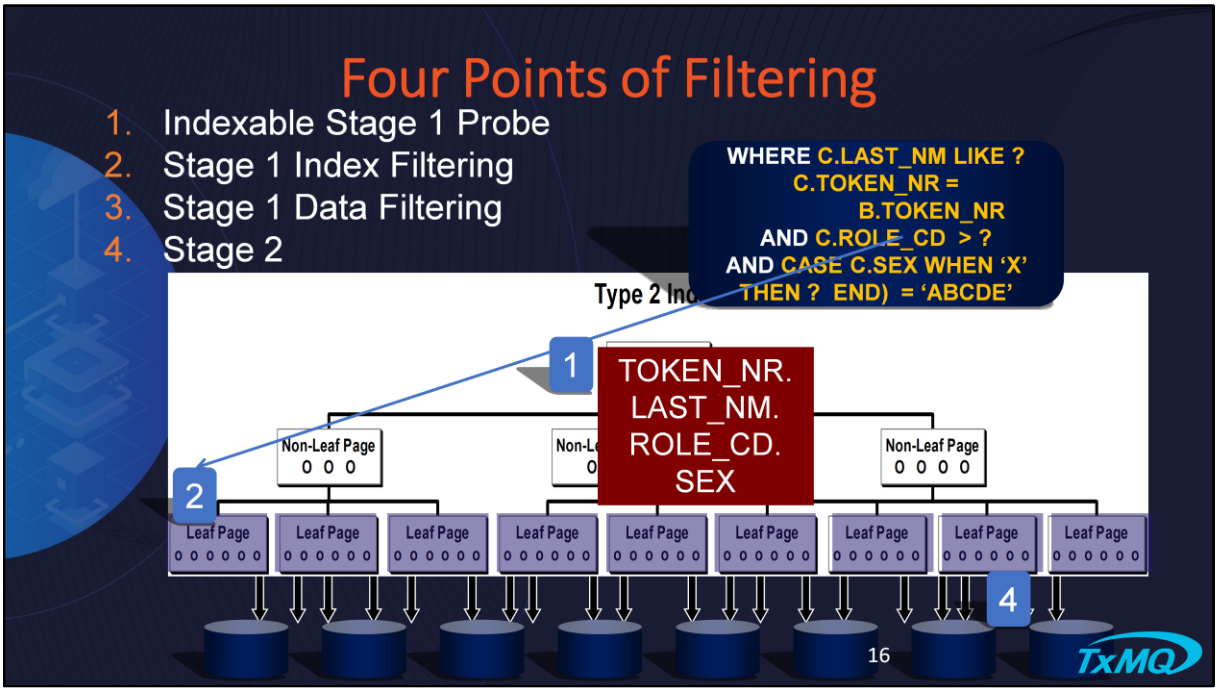
Stage 1 not indexable predicates [31]The following predicates might be evaluated during stage 1 processing, during index screening, or after data page access

There are 12 of them shown in slide 19 notes.

Stage 2 partial list is shown in slide 20 notes

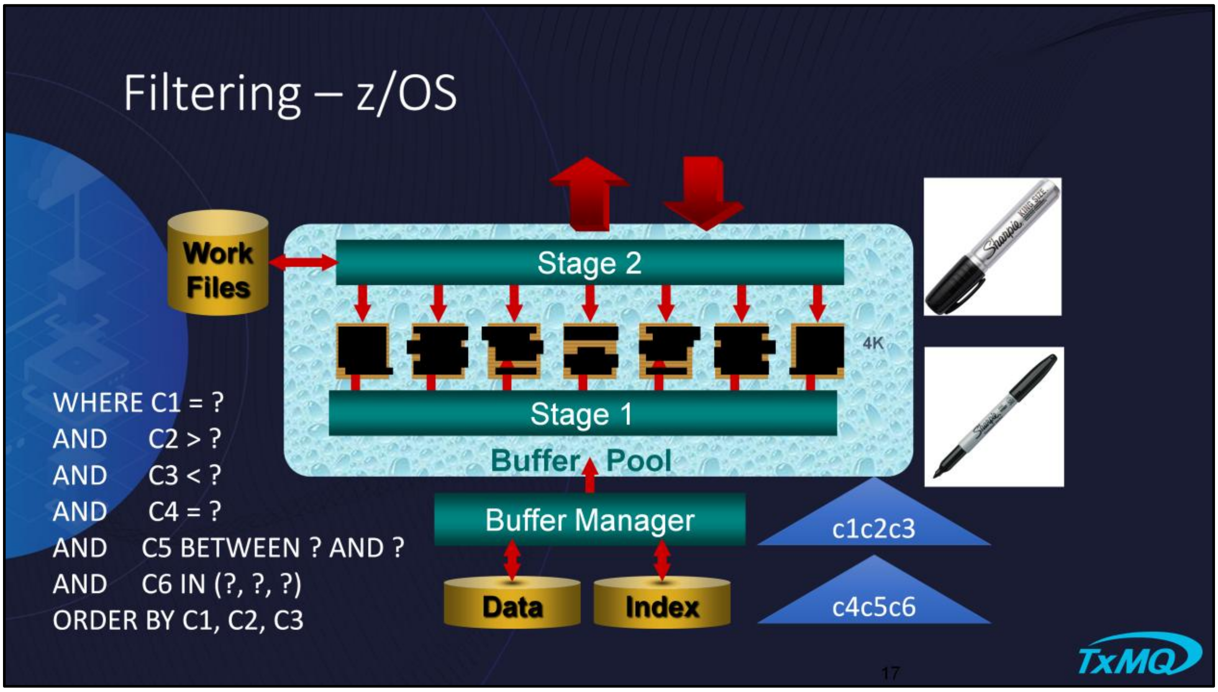


This is an example of a non-optimal index. The query has one join predicate and 3 local filters on the C table. Trouble is there are only two columns in the index leaving the filtering pushed back until steps 3 and 4. A better index would be TOKEN_NR.LAST_NM.ROLEC_CD.SEX. This moves all the filtering to step 2, greatly reducing the I/O necessary for the query.



A better index would be TOKEN_NR.LAST_NM.ROLEC_CD.SEX. This moves all the filtering to steps 1 and 2, greatly reducing the I/O necessary for the query.

1. skipped
2. Stage 1 Index Filtering - If there is no predicate involving the first column of the index, tree navigation is not allowed (0 matching). Any Stage 1 predicate (all 54) can be applied on the leaf page. This point of filtering is called index screening. Stage 2 conditions can also be applied after the Stage 1 conditions are applied (if this is index-only access *and* the Stage 2 column is included in the index - like the column SEX above.



Stage 1 Sharpie pen is much thinner that Stage 2 because Stage 1 only has 54 filters and Stage 2 has an almost infinite list (what ever is not Stage 1 is Stage 2)

Partial list of Stge2:

COL BETWEEN COL1 AND COL2 [10]

•*value* NOT BETWEEN COL1 AND COL2

•*value* BETWEEN *col expr* and *col expr*[32]

•T1.COL <> T2.COL

•T1.COL1 = T1.COL2 [3,25]

•T1.COL1 *op* T1.COL2 [3]

•T1.COL1 <> T1.COL2 [3]

•COL = ALL (*noncor subq*)

•COL <> (*noncor subq*) [22]

•COL <> ALL (*noncor subq*)

•COL NOT IN (*noncor subq*)

•COL = (*cor subq*) [5]

•COL = ALL (*cor subq*)

•COL *op* (*cor subq*) [5]

•COL *op* ANY (*cor subq*) [22]

•COL *op* ALL (*cor subq*)

•COL <> (*cor subq*) [5]

•COL <> ANY (*cor subq*) [19]

•(COL1,...COL*n*) IN (*cor subq*)

•COL NOT IN (*cor subq*)

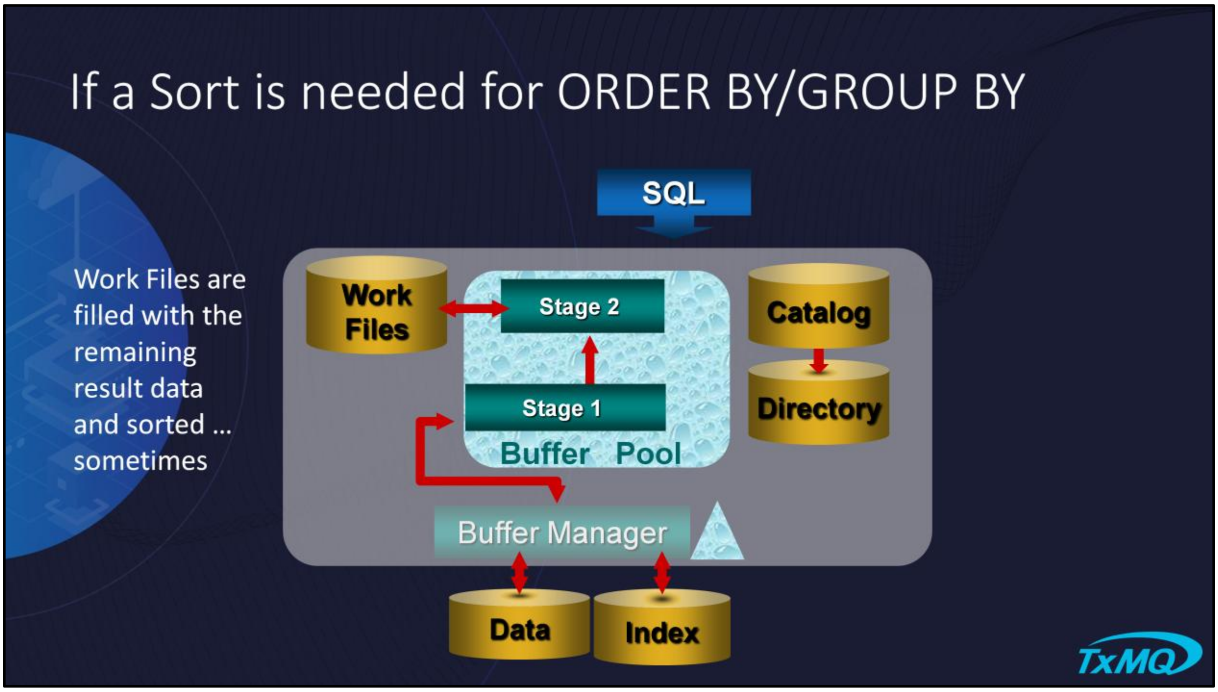
•(COL1,...COL*n*) NOT IN (*cor subq*)

•T1.COL1 IS DISTINCT FROM T2.COL2 [3]

•T1.COL1 IS DISTINCT FROM T2 *col expr* [8, 11]

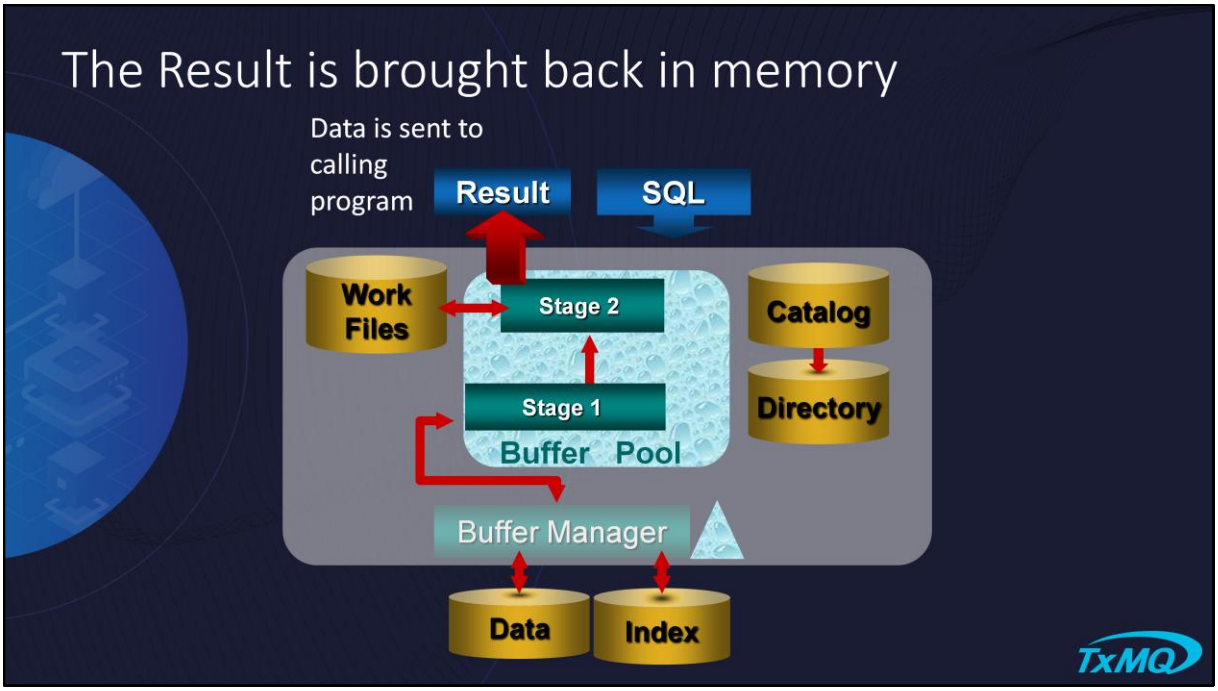
•COL IS NOT DISTINCT FROM (*cor subq*)

•EXISTS (*subq*)[19]



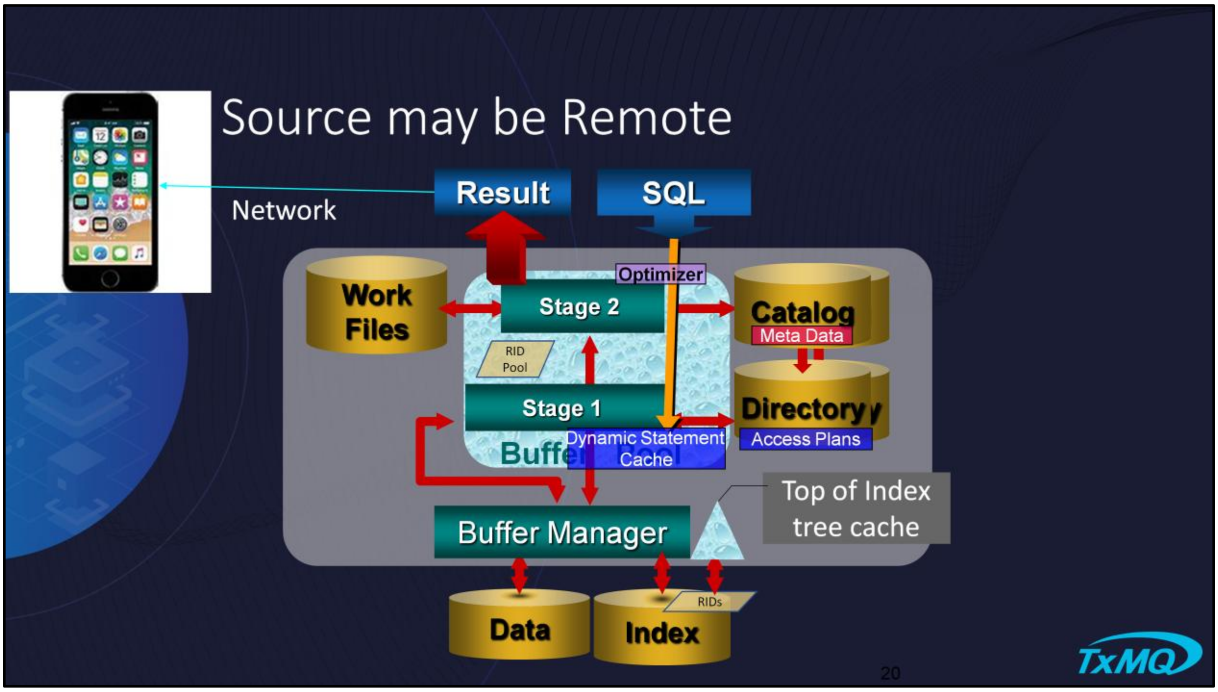
Sort rules are:

1. If you are sorting on a unique key do not include any other columns to the ORDER BY
2. Do not add redundant columns to the ORDER BY.
3. So not SELECT columns that you already know example:
   WHERE NAME = ‘SHERYL’
   Do NOT put NAME in the SELECT list



Indexable and stage 1 predicates- [31]The following predicates might be evaluated by matching index access, during index screening, or after data page access during stage 1 processing

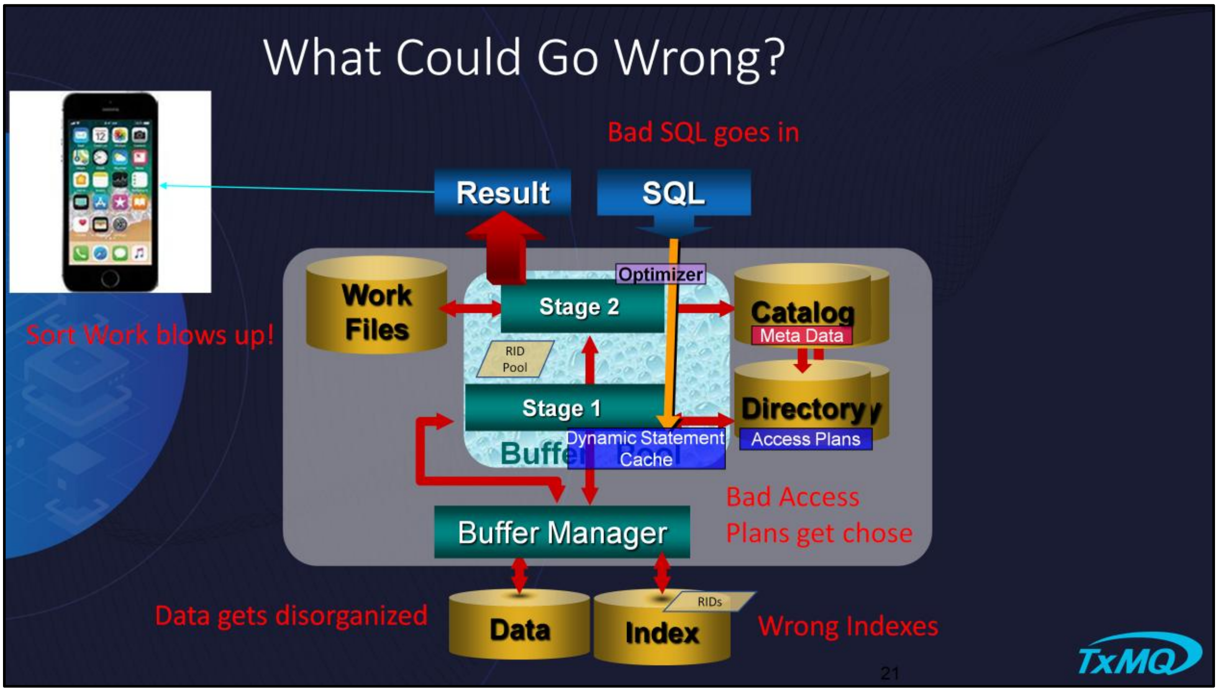
1. .COL = *value* [16, 31]
2. COL = *noncol expr* [9, 11, 12, 15, 29, 31, 32]
3. COL IS NULL [20, 21]
4. COL *op value* [13, 31]
5. COL *op noncol expr* [9, 11, 12, 13, 29, 31, 32]
6. *value* BETWEEN COL1 AND COL2 [13, 32]
7. COL BETWEEN *value1* AND *value2* [13]
8. COL BETWEEN *noncol expr 1* AND *noncol expr 2* [9, 11, 12, 13, 23, 29]
9. COL BETWEEN *expr-1* AND *expr-2* [6, 7, 11, 12, 13, 14, 15, 27, 29]
10. COL LIKE '*pattern*' [29]
11. COL IN (*list*) [17, 18]
12. COL IS NOT NULL [21]
13. COL LIKE *host variable* [2, 29]
14. COL LIKE UPPER ('*pattern*') [29]
15. COL LIKE UPPER (*host-variable*) [2, 29]
16. COL LIKE UPPER (*SQL-variable*)[2, 29]
17. COL LIKE UPPER (*global-variable*)[2, 29]
18. COL LIKE UPPER (CAST ('*pattern*' AS *data-type*))[2, 29]
19. COL LIKE UPPER (CAST (*host-variable* AS *data-type*))[2, 29]
20. COL LIKE UPPER (CAST (*SQL-variable* AS *data-type*))[2, 29]
21. COL LIKE UPPER (CAST (*global-variable* AS *data-type*))[2, 29]
22. [2, 29]
23. T1.COL = T2.COL
24. T1.COL *op* T2.COL
25. T1.COL = T2 *col expr* [6, 9, 11, 12, 14, 15, 25, 27, 29]
26. T1.COL *op* T2 *col expr* [6, 9, 11, 12, 13, 14, 15, 29]
27. COL = (*noncor subq*)
28. COL *op* (*noncor subq*) [28]
29. COL = ANY (*noncor subq*) [22, 29]
30. (COL1,...COL*n*) IN (*noncor subq*) [29]
31. COL = ANY (*cor subq*) [19, 22, 29]
32. COL IS NOT DISTINCT FROM *value* [16]
33. COL IS NOT DISTINCT FROM *noncol expr* [9, 11, 12, 15, 29]
34. T1.COL1 IS NOT DISTINCT FROM T2.COL2 [3, 4
35. T1.COL1 IS NOT DISTINCT FROM T2 *col expr* [6, 9, 11, 12, 14, 15, 29]
36. COL IS NOT DISTINCT FROM (*noncor subq*)
37. SUBSTR(COL,1,*n*) = *value*
38. SUBSTR(COL,1,*n*) *op value*
39. DATE(COL) = *value*[33]
40. DATE(COL) *op value*[33]
41. YEAR(COL) = *value*[33]
42. YEAR(COL) *op value*[33]



Stage 1 not indexable predicates [31]

The following predicates might be evaluated during stage 1 processing, during index screening, or after data page access.

1. COL <> *value* [8, 11]
2. COL <> *noncol expr* [8, 11, 29]
3. COL NOT BETWEEN *value1* AND *value2*
4. COL NOT IN (*list*)
5. COL NOT LIKE ' *char*' [29]
6. COL LIKE '%*char*' [1, 29]
7. COL LIKE '_*char*' [1, 29]
8. T1.COL <> T2 *col expr* [8, 11, 27, 29]
9. COL *op* ANY (*noncor subq*) [22]
10. COL *op* ALL (*noncor subq*)
11. COL IS DISTINCT FROM *value* [8, 11]
12. COL IS DISTINCT FROM (*noncor subq*)



Stage 2 predicates- The following predicates must be processed during stage 2, after the data is returned.

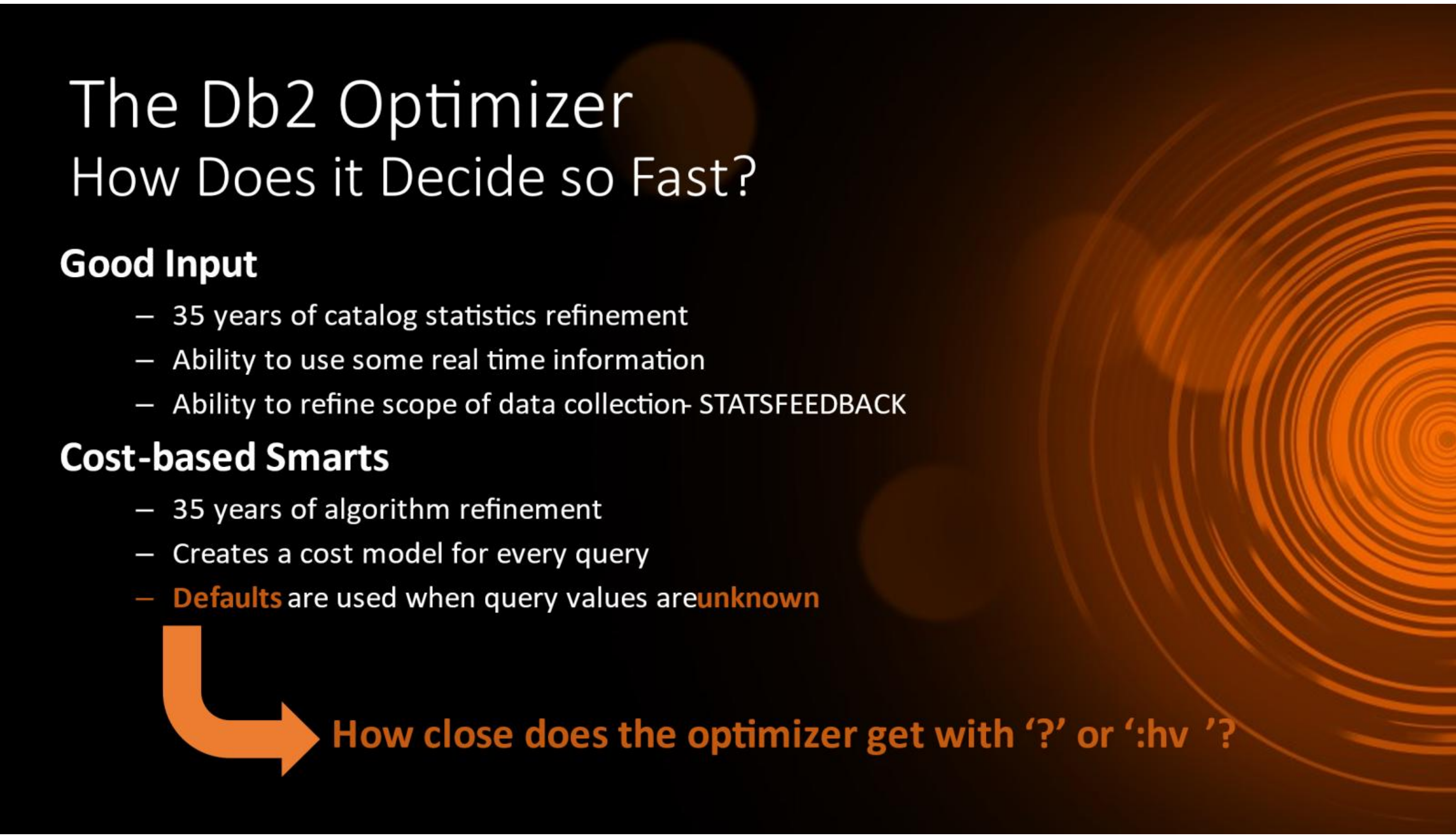
The list is not complete due to any predicate not Stage 1 Indexable or Stage 1 index/data screening is State 2.

COL BETWEEN COL1 AND COL2 [10]
•*value* NOT BETWEEN COL1 AND COL2
•*value* BETWEEN *col expr* and *col expr*[32]
•T1.COL <> T2.COL
•T1.COL1 = T1.COL2 [3,25]
•T1.COL1 *op* T1.COL2 [3]
•T1.COL1 <> T1.COL2 [3]
•COL = ALL (*noncor subq*)
•COL <> (*noncor subq*) [22]
•COL <> ALL (*noncor subq*)
•COL NOT IN (*noncor subq*)
•COL = (*cor subq*) [5]
•COL = ALL (*cor subq*)
•COL *op* (*cor subq*) [5]
•COL *op* ANY (*cor subq*) [22]
•COL *op* ALL (*cor subq*)
•COL <> (*cor subq*) [5]
•COL <> ANY (*cor subq*) [19]
•(COL1,...COL*n*) IN (*cor subq*)
•COL NOT IN (*cor subq*)
•(COL1,...COL*n*) NOT IN (*cor subq*)
•T1.COL1 IS DISTINCT FROM T2.COL2 [3]
•T1.COL1 IS DISTINCT FROM T2 *col expr* [8, 11]
•COL IS NOT DISTINCT FROM (*cor subq*)
•EXISTS (*subq*)[19]
•*expression* = *value* [27, 32]
•*expression* <> *value* [27]
•*expression op value* [27, 32]
•*expression op* (*subq*)
•NOT XMLEXISTS
•CASE *expression* WHEN *expression* ELSE *expression* END = *value* [32]
....

Indexable but not stage 1 predicates The following predicates can be processed during index access, but cannot be processed during stage 1
XMLEXISTS [26]

Remaining Agenda

- Review of what IBM's Db2ZAI can and cannot do
- How to change the optimizers mind
  - Case Studies Using a Proven Method
  - Extreme Tuning
  - How to put a query on a diet



DB2 12 for z Optimizer (ibm.com)

DB2 12 for z Optimizer (ibm.com)

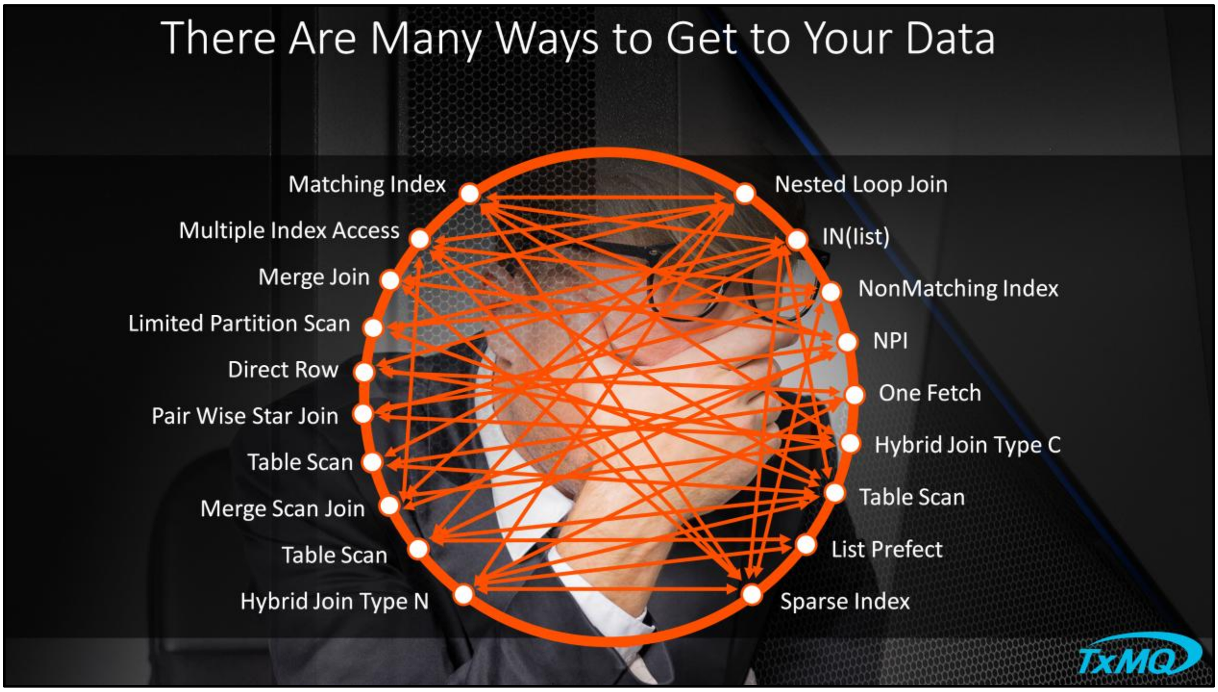
- **30% - 90% reduction in ET and CPU**
  - Complex OJs, UNION ALL, UDFs & Table UDFs
  - Combinations of Table Expressions, Views and Outer Joins
  - VARGRAPHIC data type
  - Disorganized data, poor CR indexes
  - ***Nearly 100% NEW Access Paths vs. DB2 11***

Figure 3 Workload characteristics of workloads in the 30–90% CPU reduction range

Open ended ranges, '>' etc. are larger 1/1000 vs. closed ended ranges, BETWEEN or LIKE are smaller. 3/10,000. This is the default % of rows the optimizer is estimating will come back from your WHERE clause. Notice that it screams, "Use and Index!"

This may be far from reality.

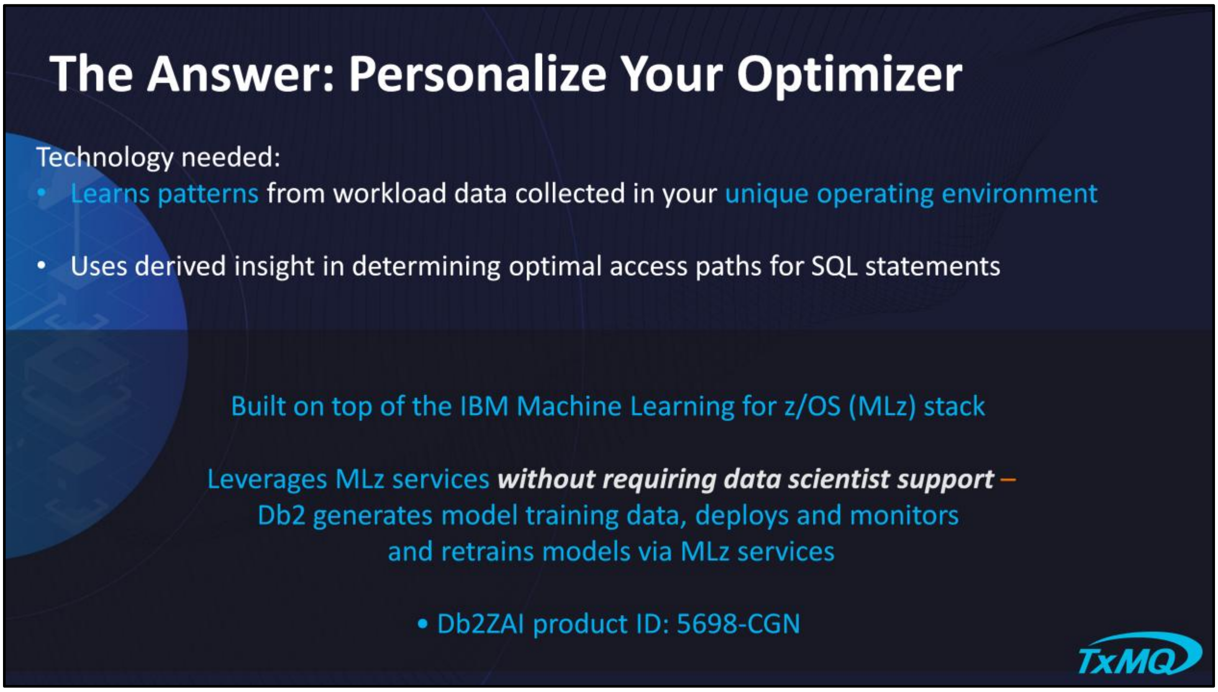
Use machine learning to improve the quality and effectiveness of inputs to the optimizer cost model



It wasn't always like this, but IBM keeps adding new cool access techniques.

From Terry Purcell's RedPaper: [DB2 12 for z Optimizer (ibm.com)]

"nearly 100% new access paths vs. Db2 11" .

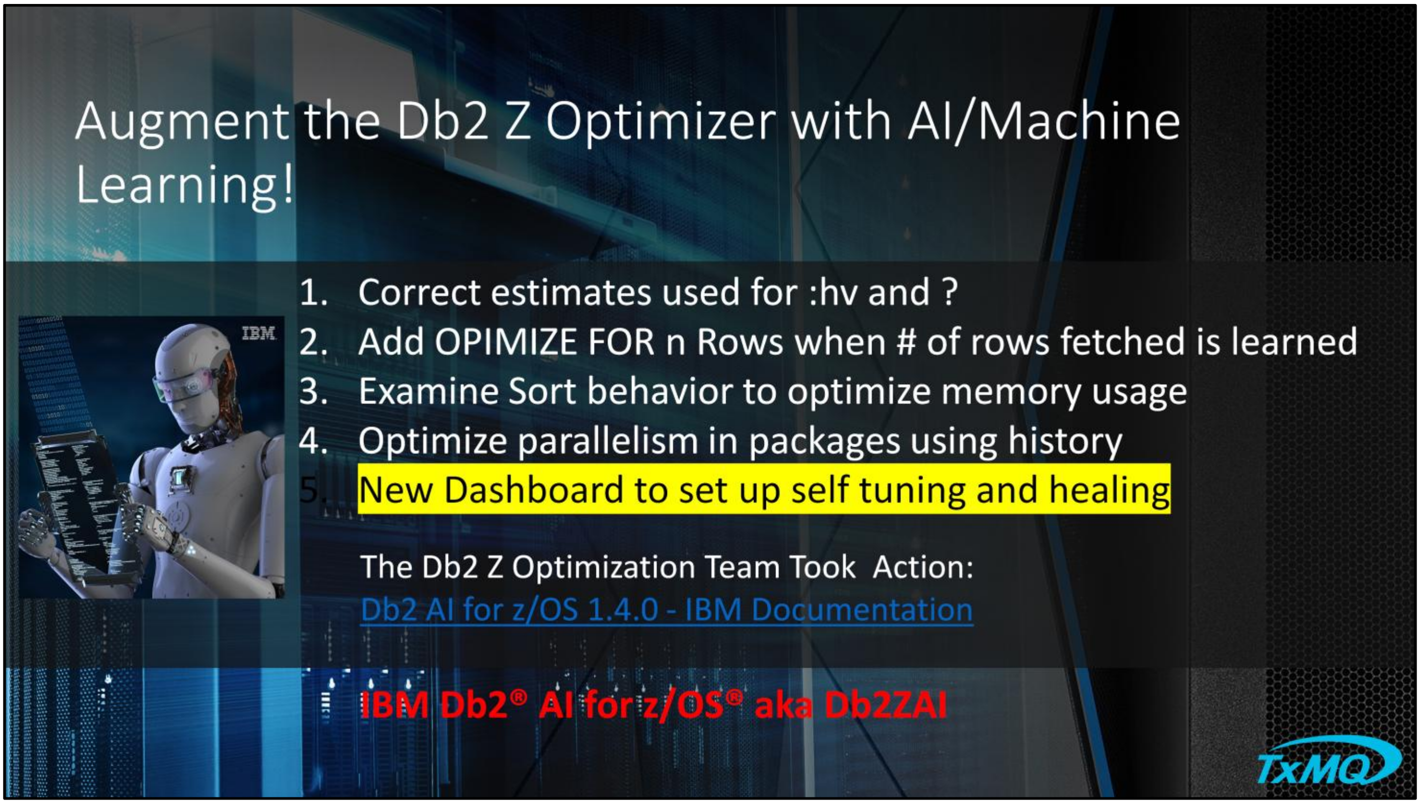


From Overview of IBM Db2 AI for z/OS
Db2® AI for z/OS® ("Db2ZAI") empowers the optimizer in your Db2 for z/OS engine to determine the best-performing query access paths, based on your workload characteristics. In addition, Db2ZAI detects Db2 system performance exceptions and provides recommended actions for tuning which are based on your environment.
The optimizer consists of Relational Data Services ("RDS") components that govern query transformation, access path selection, run time, and parallelism for every SQL statement in your system. The access path for an SQL statement essentially dictates how Db2 accesses the data that the query specifies. It determines the indexes and tables that are accessed, the access methods that are used, and the order in which objects are accessed.

Leveraging machine learning technology, Db2ZAI collects data from the Db2 for z/OS optimizer and the query execution history, which are derived from workloads in your unique operating environment. As part of the model training process, Db2ZAI then finds patterns from this data and learns the optimal access paths for queries entering Db2 for z/OS.

Once trained, the model is ready to be deployed into production, providing insights to the optimizer's access path selection. These insights are in addition to what the optimizer uses today in the selection of the best query path. The information is unique to your environment, and currently unknown to the traditional query optimizer. With the new intelligence on the insights gained from these models, the query optimizer is better able to identify the optimal access paths for SQL statements

Virtual Data Scientist! Has the data, knows which algorithm to use, learns from modeling and scoring, provides solutions, and cleans up after itself.



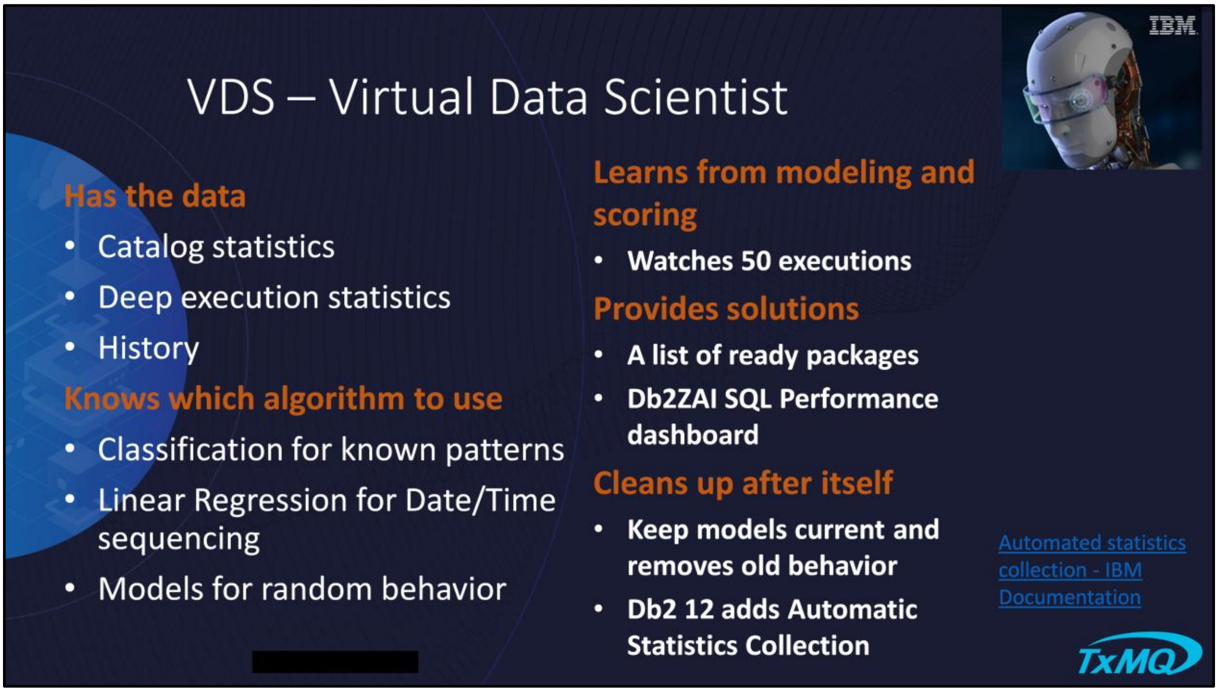
Data Tech Summit Presented Live Demo but recorded from Silicon Valley Lab

October 5-7, 2021

Join session in Channel 1 - Many considerations can be made regarding design choices when building and maintaining Db2 for z/OS indexes with regard to high performance. This presentation is going to attempt to address many of those choices, but it is the combination of knowing how an application is going to use the database as well as adequate testing to make the appropriate design decisions

Tom Beavin, Db2 Developer, IBM

Tom Ramey, Director, WW Z Data and AI, IBM



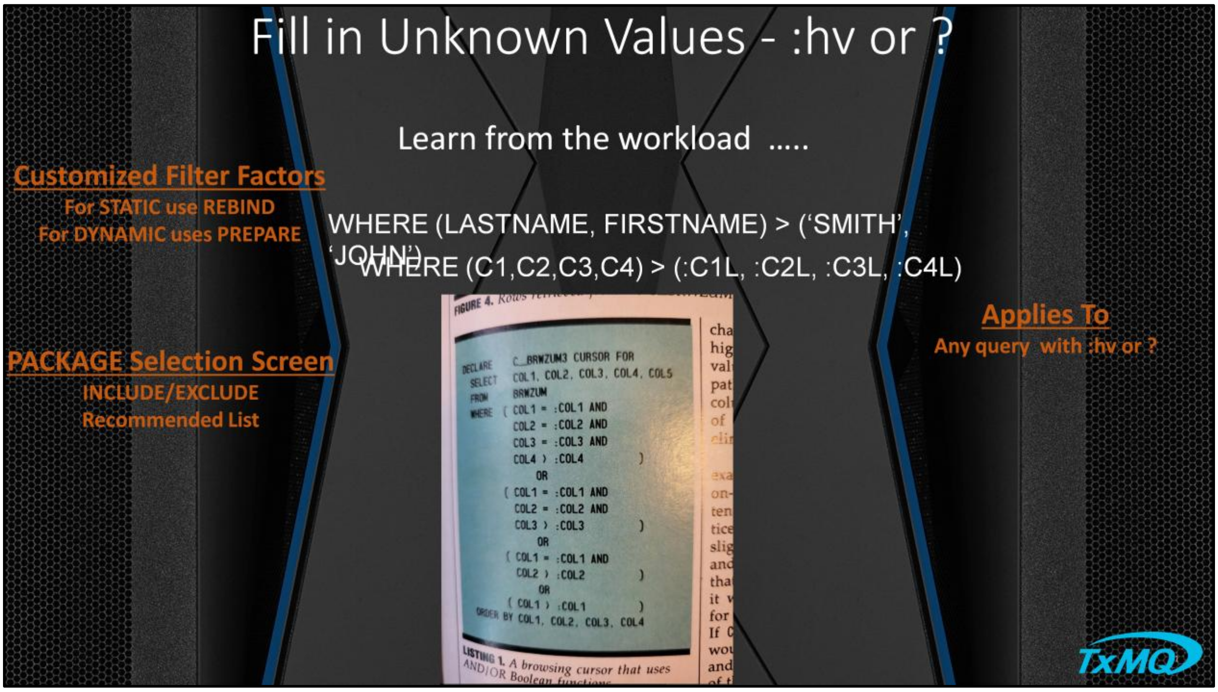
A type of supervised machine learning: classification

Classification Predictive Modeling. In machine learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data.

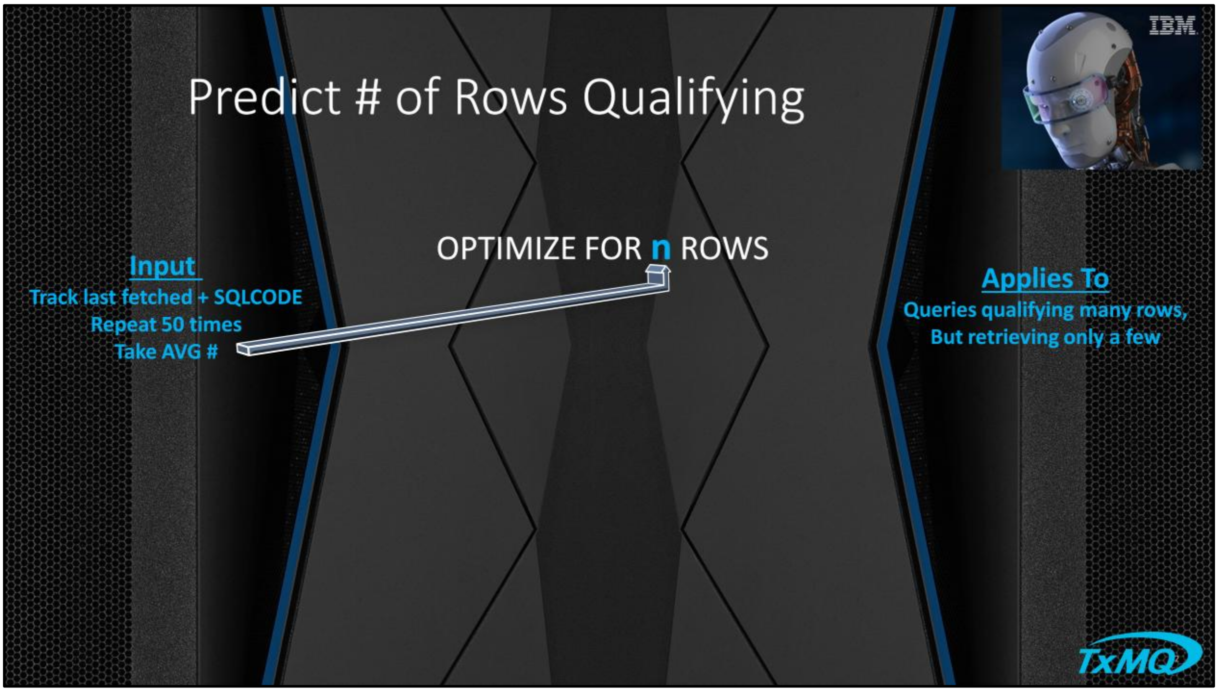
Another type of supervised machine learning: regression

Regression is a supervised machine learning technique which is used to **predict continuous values**. The ultimate goal of the regression algorithm is to plot a best-fit line or a curve between the data. The three main metrics that are used for evaluating the trained regression model are variance, bias and error.

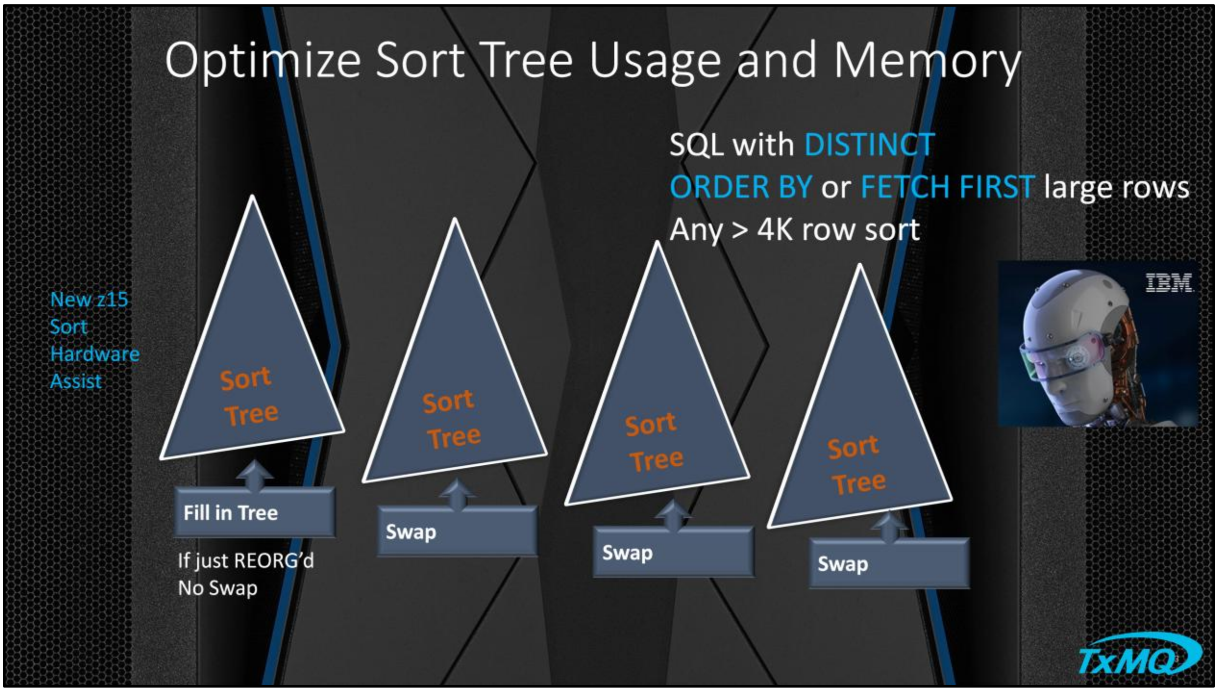
[Automated statistics collection - IBM Documentation]



This is the game changer for static SQL. Think BIG static Batch runs and Dynamic Queries too. This alone can flip table join sequences.

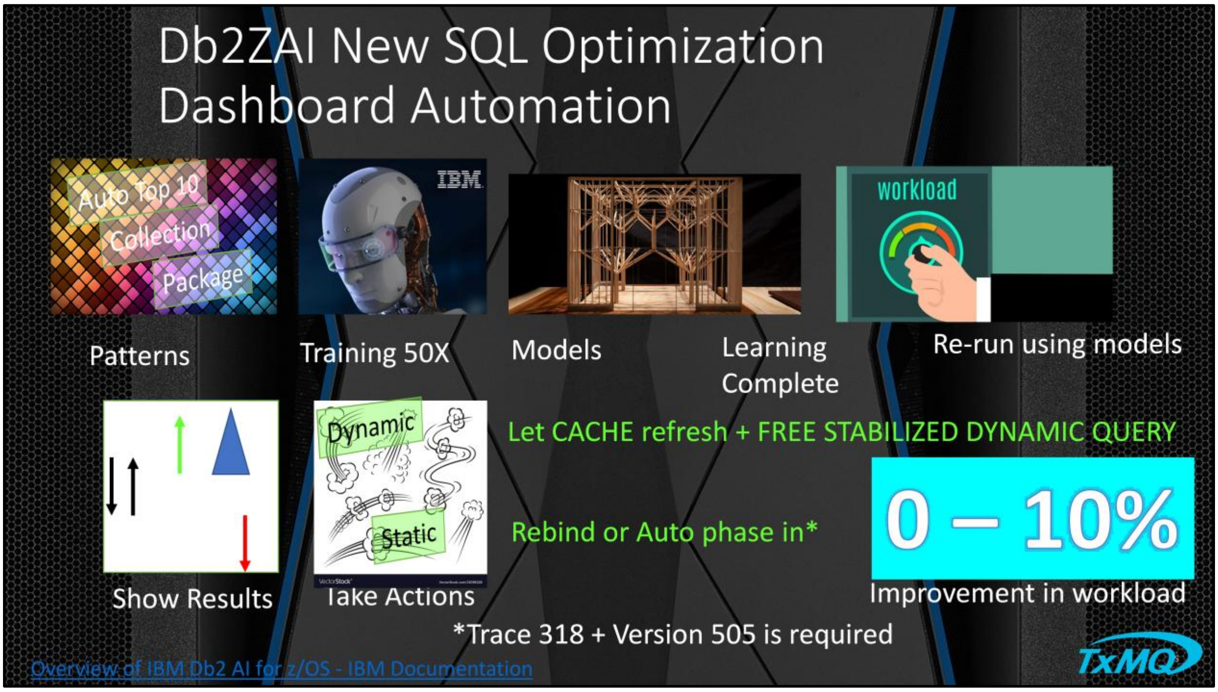


Most **DB2** predicates are based on the columns of a table. They either **qualify rows** (through an index) or reject **rows** (returned by a scan) when the table is accessed.



Db2 AI for z/OS will feel the organization of your data during training and adjust its algorithm to use less memory over time. For z15, sort hardware assist may be available.

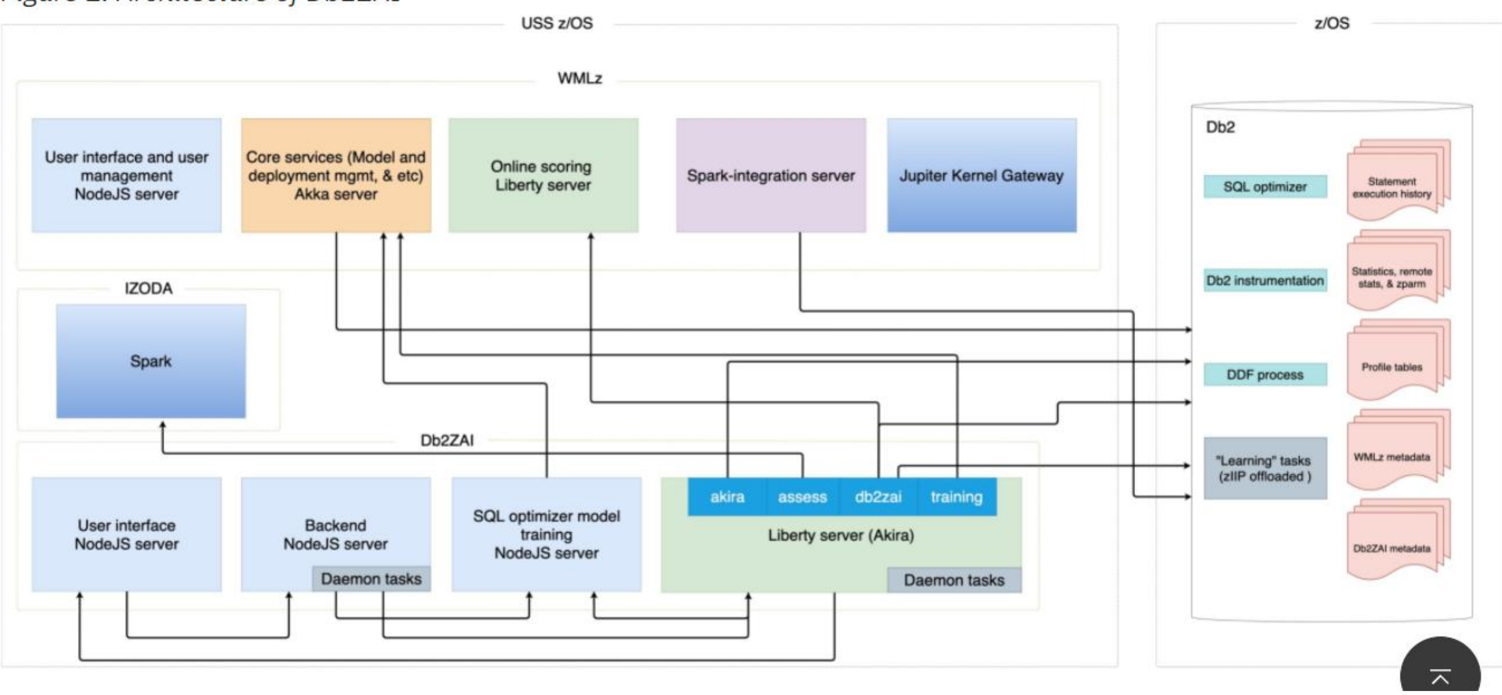
DB2 Version 6 had two releases because parallelism was broken and gave bad results. This caused EVERYONE to back off from using it. That was so long ago it no longer applies. The current Db2 Optimizer is cautious of taking queries parallel in general. Db2ZAI turbo charges its appetite for going more aggressive into parallelism.

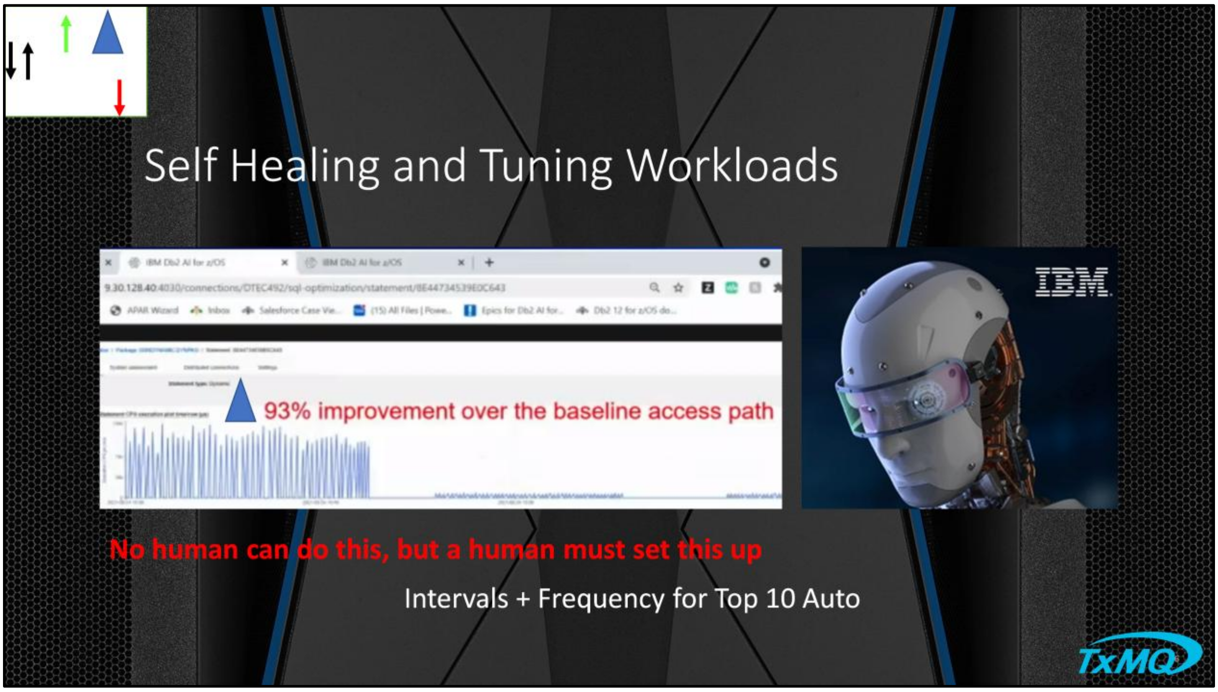


From IBM's Overview of V1.4.0:
[Overview of IBM Db2 AI for z/OS - IBM Documentation]

Db2ZAI now offers a simplified architecture, eliminating the need for a Linux environment.

Figure 1. Architecture of Db2ZAI



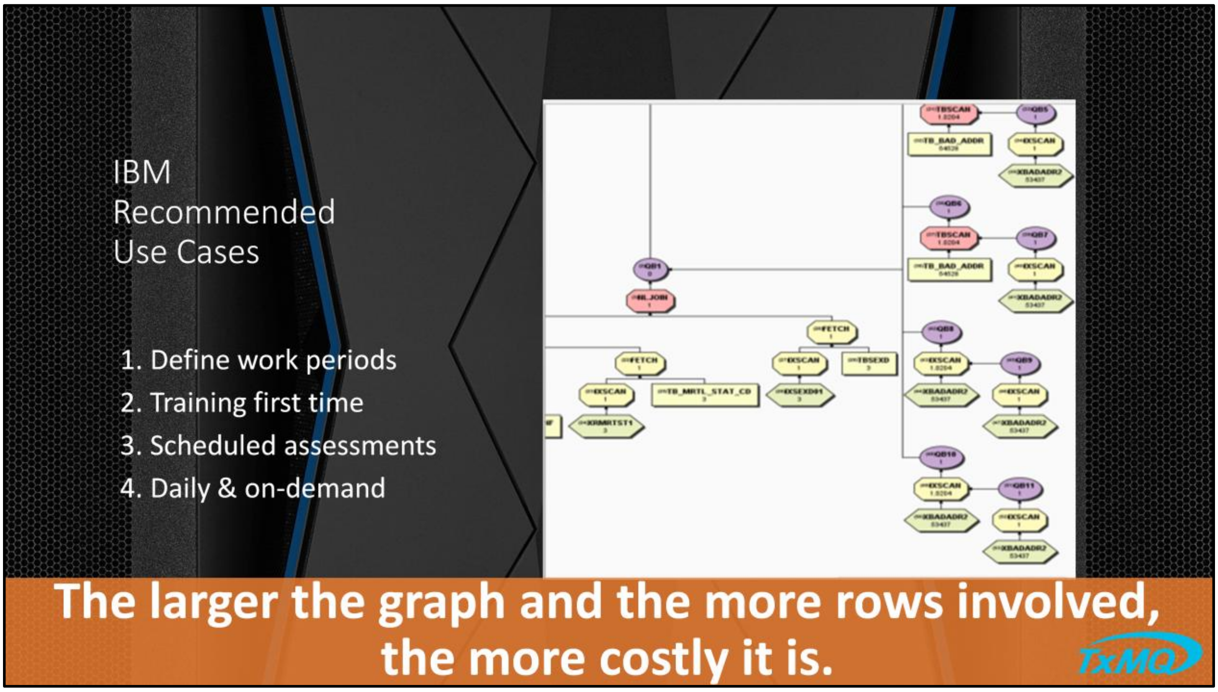


[What's new in IBM Db2 AI for z/OS 1.4.0 - IBM Documentation]

What's new in version 1.4.0?

•Improved automated SQL regression detection and resolution, enabling regressions to be detected and resolved more quickly, and in some cases in real-time, even before the query has completed execution.
•Sort optimization enabled for OLTP queries, which can result in savings of CPU resources.
•Improved sort optimization when combined with the IBM® z15™ hardware sort assist for SQL, whereby learning can improve exploitation of the z15 sort feature.
•Improved SQL optimization user interface, which clearly shows the benefits that Db2ZAI is providing by highlighting progress, benefits, and actions to be taken:

- Db2ZAI now shows the progress it has made in learning about static SQL packages and dynamic SQL statements.
- The improved SQL optimization dashboard shows the benefits provided by Db2ZAI in terms of SQL statements improved, average CPU improvement, and access path regressions resolved.
- The improved user interface clearly indicates the recommended actions to be taken to improve performance.



This is part of a picture of the most expensive query I have ever tuned

The 5th Use Case – 24/7 Security

STMT_ID	CPU	EXECS	ELAPSED	GETPAGES	EXAMINED ROWS	PROCESSED	STAT_SORT	STAT_INDX	STAT_RSCN
1022787	**4705.7**	**5258**	4722.46	**729,475,051**	298,690,230	2629	0	283543230	5606
996083	214.98	112873	179.02	11902513	68238	112873	0	112873	0
800016	214.56	113783	190.44	11947215	0	113783	0	113783	0

Math:
729,457,051/5258 =
138,731 Getpages per Exec

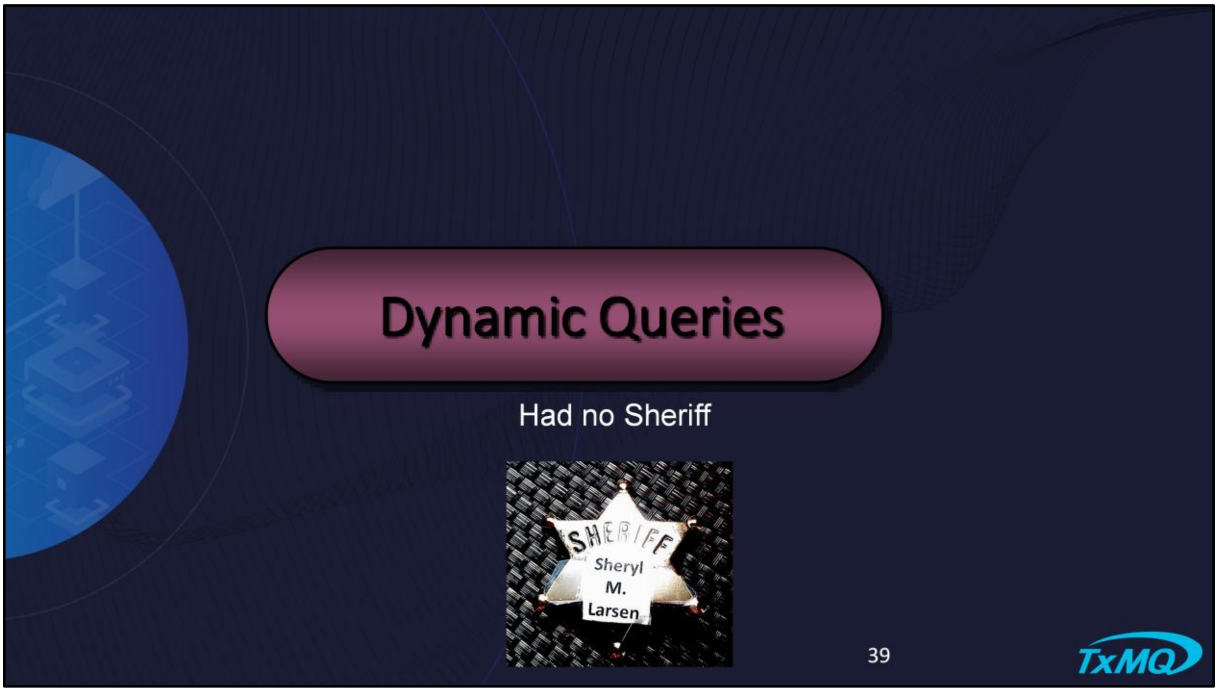
In Procedure SQL:

Exec MONSTER query

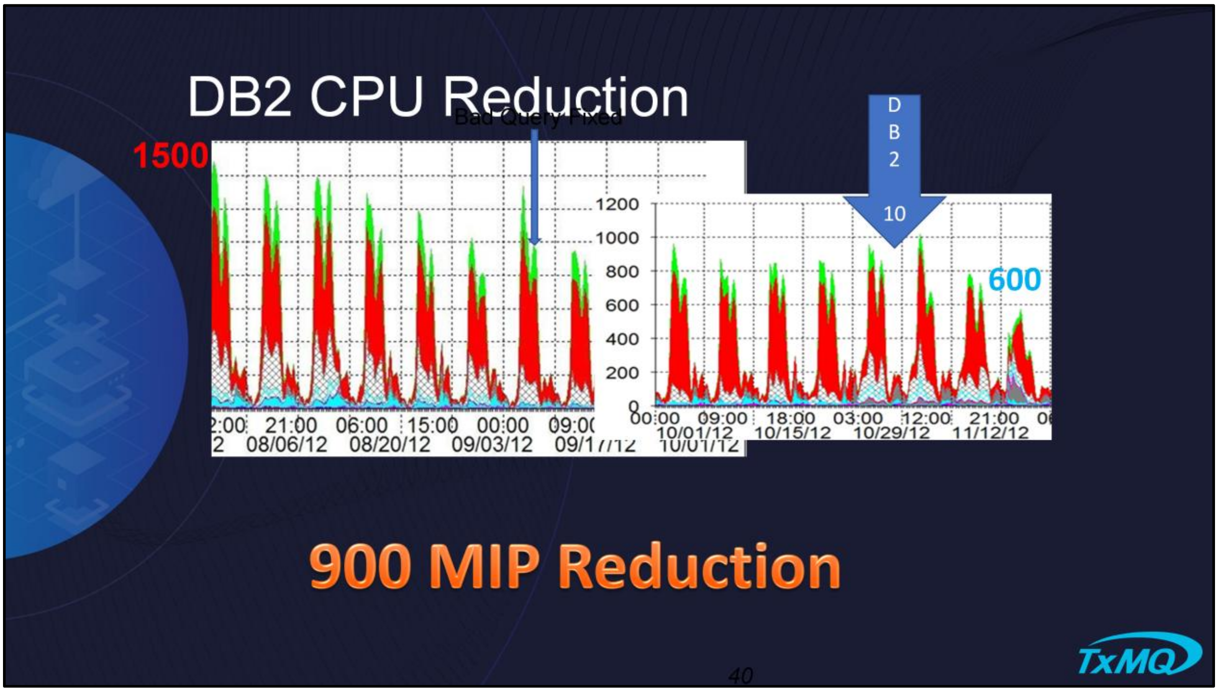
```
IF SQLCODE = 100
   SET BADADDRFLAG = 'N'
ELSE
   CONTINUE
```

TxMQ

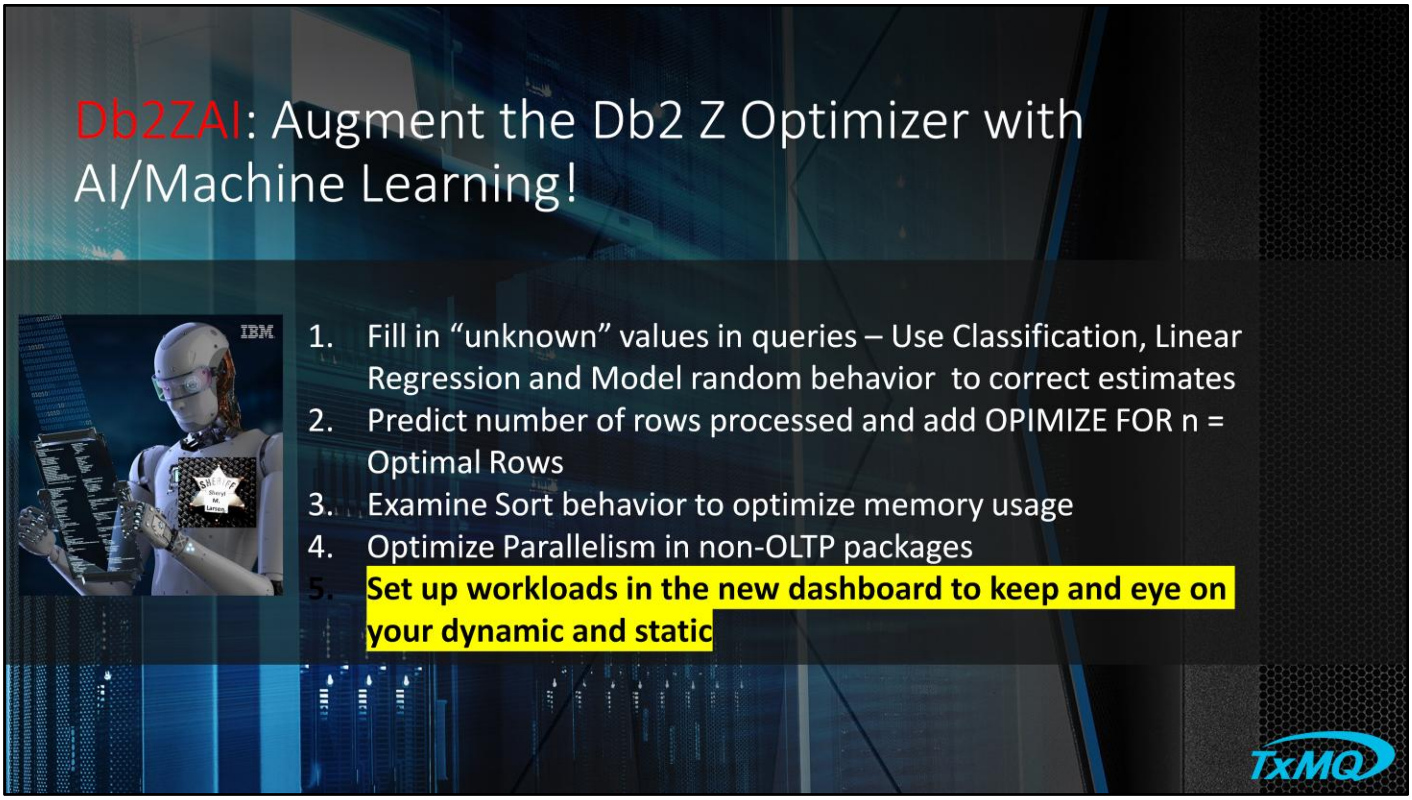
¾ billion getpages with traced turned on only for 3 days. Orders of magnitude more expensive that anything else. This is not SQL injection. This is SQL explicitly forced to execute and read the entire result.



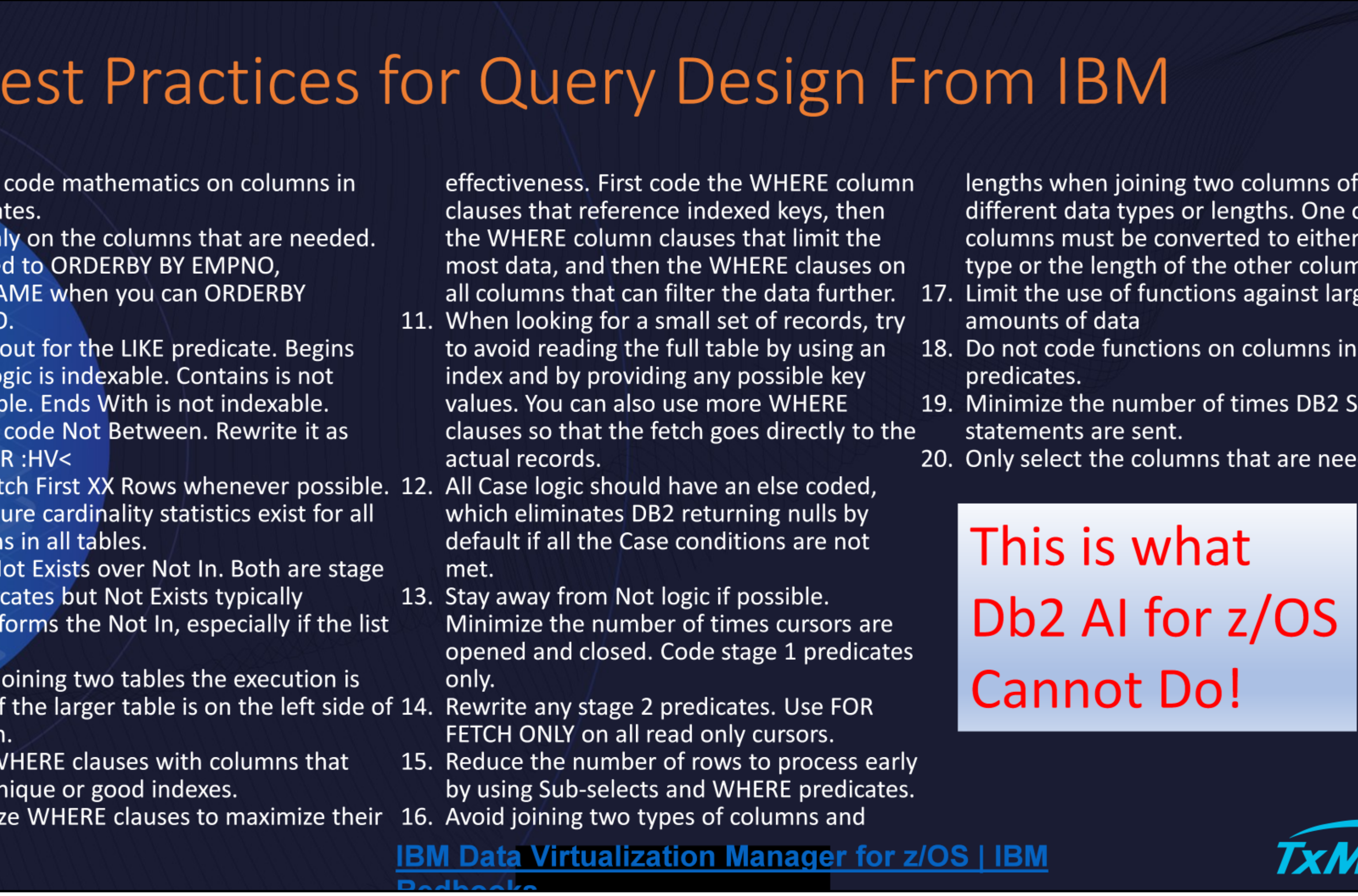
SQL statements can sneak into your shop. The query re-write took the getpages down to 6 per execution. Just a correlated EXISTS check returning no data. Back track to the actual business question being asked to see if the query answers correctly.



The tuned expensive query brought down IBM MIPS used 300 points in one day. The entire manual tuning effort to seven months to complete starting at a near $5million looming CPU upgrade 1600 MIPS and reducing to 600 MIPS. No upgrade was needed for the next five years!



Hire a full time Sherriff to keep an eye on your workload and your statistics!



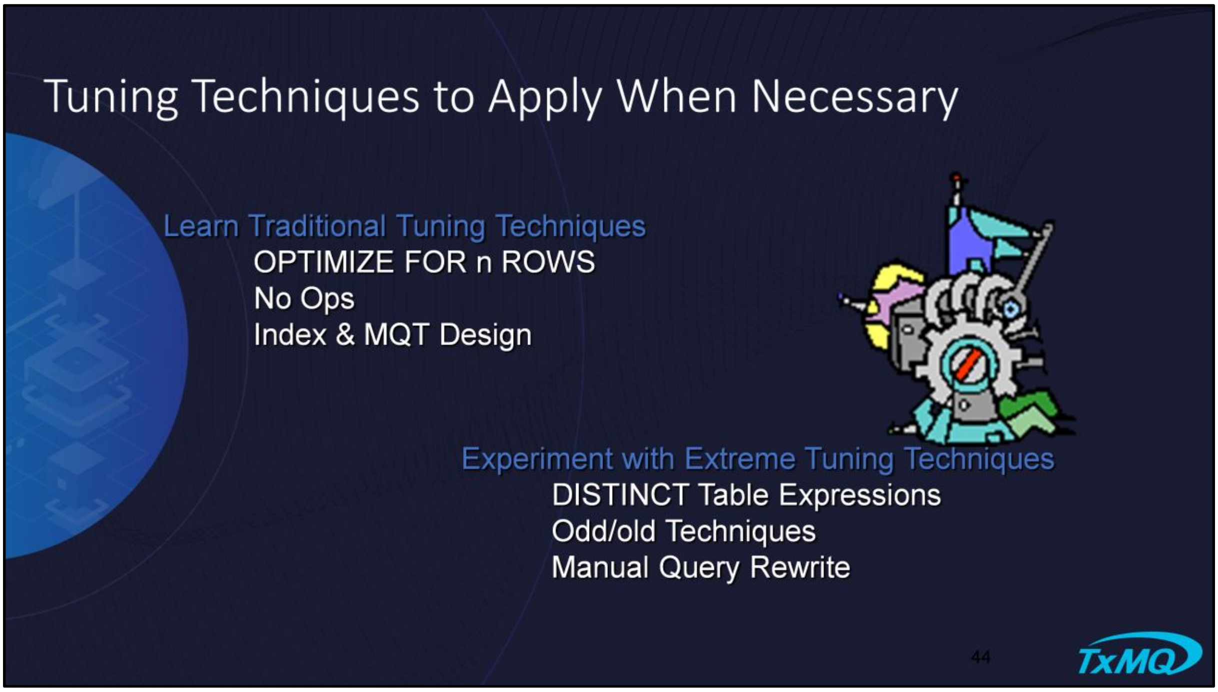
Best practices for query design from IBM Efficient SQL list: 8.8.1 SQL best practices https://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg248514.html?Open

1. Do not code mathematics on columns in predicates.
2. Sort only on the columns that are needed. No need to ORDERBY BY EMPNO, LASTNAME when you can ORDERBY EMPNO.
3. Watch out for the LIKE predicate. Begins With logic is indexable. Contains is not indexable. Ends With is not indexable.
4. Do not code Not Between. Rewrite it as >:HV OR :HV<
5. Use Fetch First XX Rows whenever possible.
6. Make sure cardinality statistics exist for all columns in all tables.
7. Code Not Exists over Not In. Both are stage 2 predicates but Not Exists typically outperforms the Not In, especially if the list is long.
8. When joining two tables the execution is faster if the larger table is on the left side of the join.
9. Code WHERE clauses with columns that have unique or good indexes.
10. Prioritize WHERE clauses to maximize their effectiveness. First code the WHERE column clauses that reference indexed keys, then the WHERE column clauses that limit the most data, and then the WHERE clauses on all columns that can filter the data further.
11. When looking for a small set of records, try to avoid reading the full table by using an index and by providing any possible key values. You can also use more WHERE clauses so that the fetch goes directly to the actual records.
12. All Case logic should have an else coded, which eliminates DB2 returning nulls by default if all the Case conditions are not met.
13. Stay away from Not logic if possible. Minimize the number of times cursors are opened and closed. Code stage 1 predicates onl
14. Rewrite any stage 2 predicates. Use FOR FETCH ONLY on all read only cursors.
15. Reduce the number of rows to process early by using Sub-selects and WHERE predicates.
16. Avoid joining two types of columns and lengths when joining two columns of different data types or lengths. One of the colum must be converted to either the type or the length of the other column.
17. Limit the use of functions against large amounts of data
18. Do not code functions on columns in predicates.
19. Minimize the number of times DB2 SQL statements are sent.
20. Only select the columns that are needed.

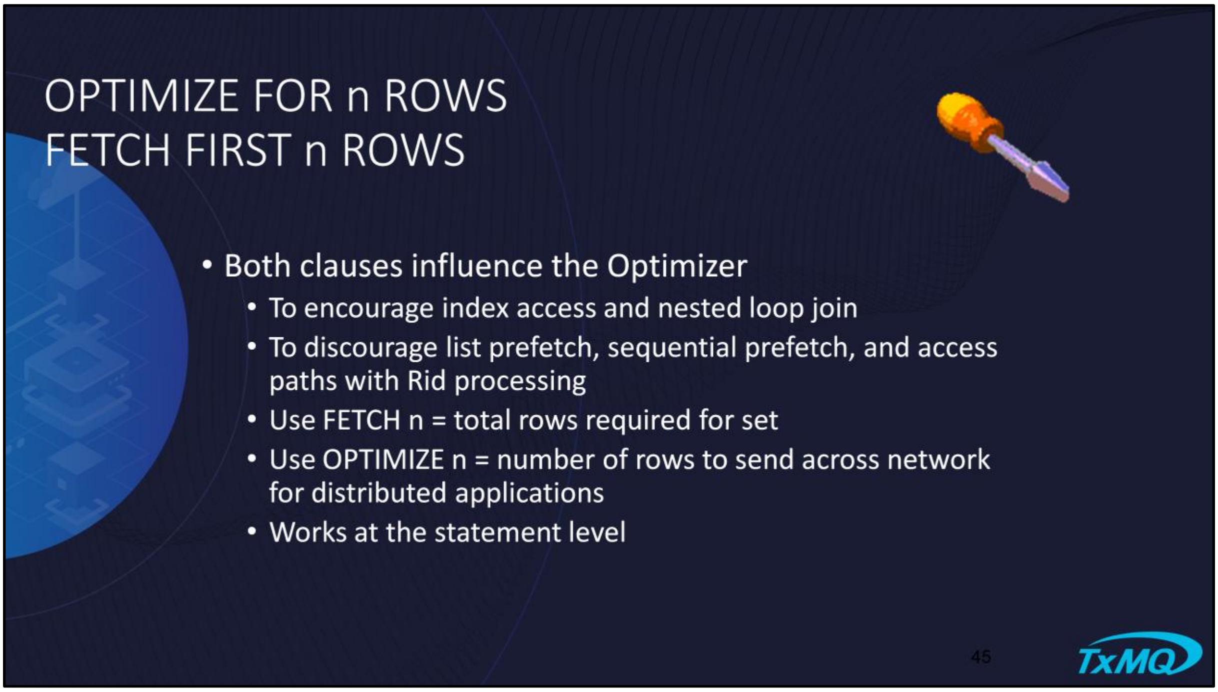
My SQL Review Checklist

1. Examine Program logic
2. Examine FROM clause
3. Verify Join conditions
4. Promote Stage 2's and Stage 1 NOTs
5. Prune SELECT lists
6. Verify local filtering sequence
7. Analyze Access Paths
8. Tune if necessary

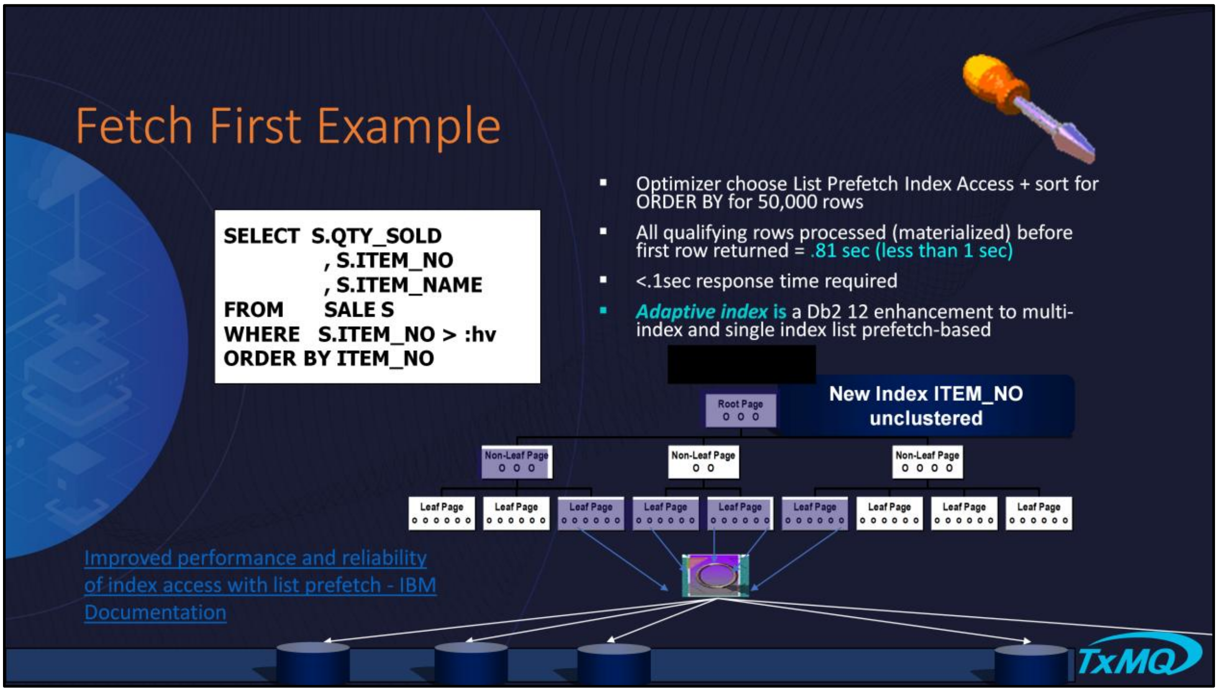
1. Examine Program logic – check for program filtering and joining. Move work into the query.
2. Examine FROM clause – order of tables insignificant unless > 9 table joins. List preferred join sequence for this and OUTER JOINs
3. Verify Join conditions – make sure every table is hooked up correctly to avoid cartesian joins
4. Promote Stage 2's/Residuals and Stage 1's if possible – promotions can change access paths
5. Verify data type matches – mismatched numeric and date/time will cause delays in filtering and alter the access path
6. Prune SELECT lists – remove columns with values determined to be static by WHERE clause filtering. Remove columns used in the ORDER BY or GROUP BY sequencing but not needed for the display.
7. Verify local filtering sequence – If host variables are used, add parenthesis to override the predetermined filtering sequence when necessary. This reduces the CPU required to disqualify rows
8. Analyze Access Paths – Only check the access path of the FINAL query, after query rewrite, bound with production statistics in a subsystem that resembles the production thresholds as closely as possible.
9. Tune if necessary – A topic for today!



You can use multiple techniques but should add them one at a time.



The IBM Db2 AI for z/OS product adds Optimize for n Rows if you forget.



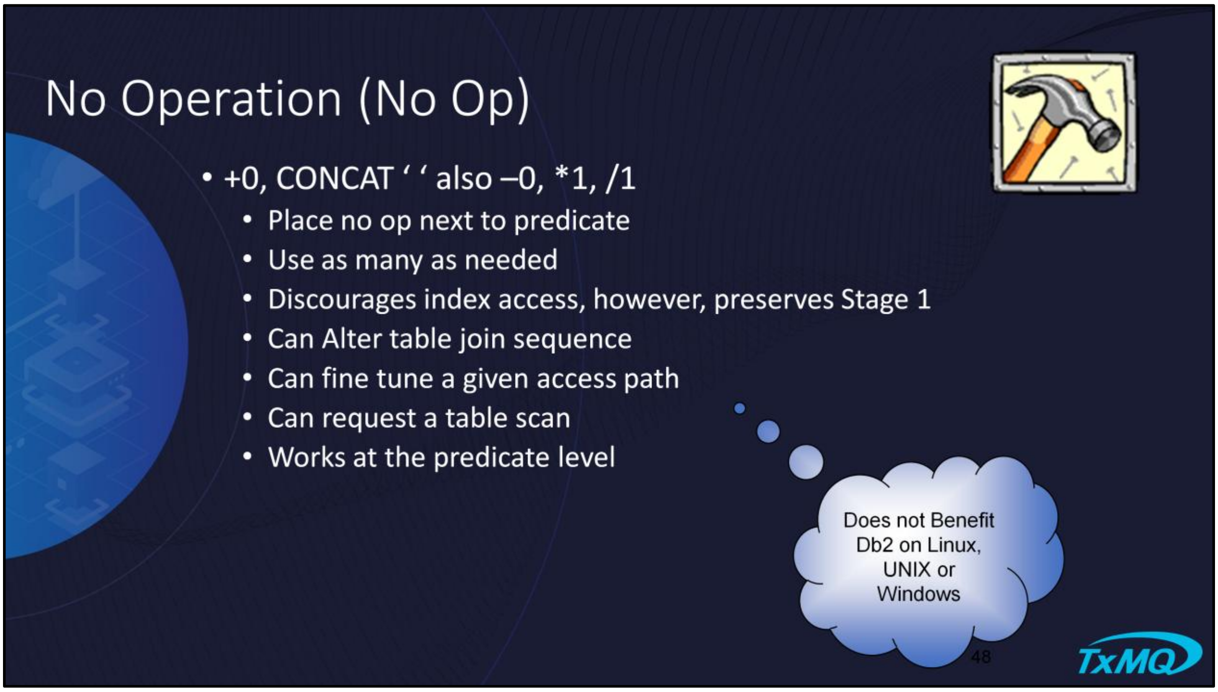
[Improved performance and reliability of index access with list prefetch - IBM Documentation]()

NEW Db212 List Prefetch enhancements

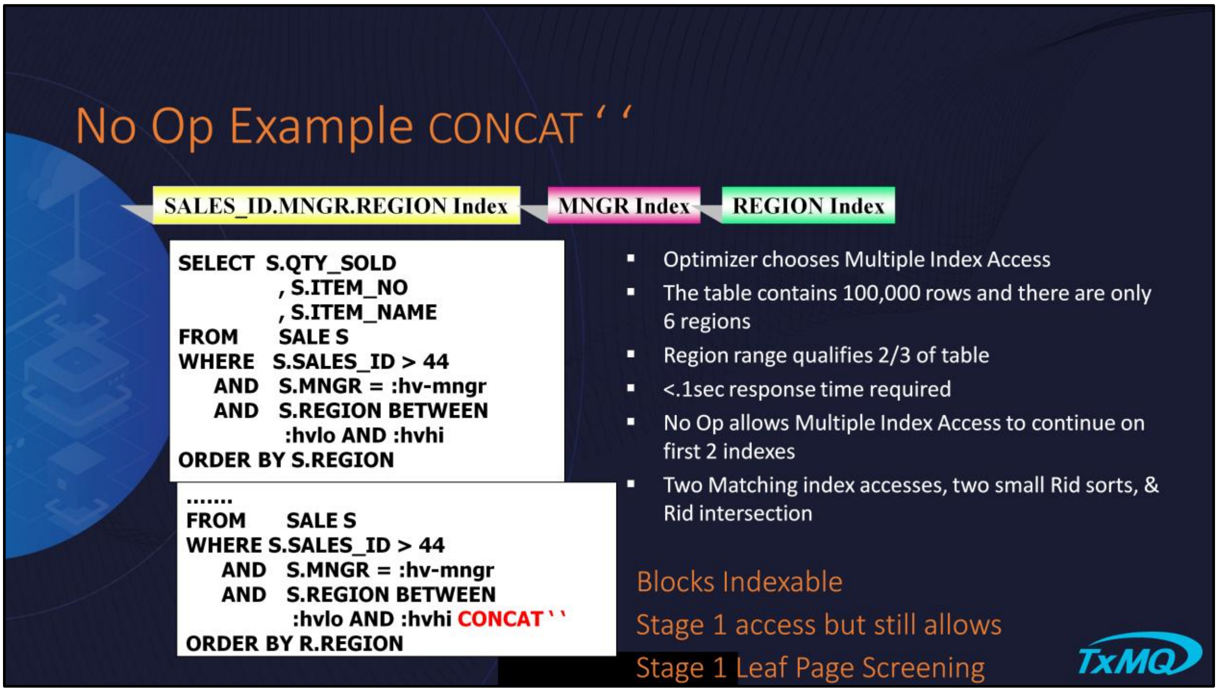
Adaptive index is a Db2 12 enhancement to multi-index and single index list prefetch-based plans that introduces logic at execution time to determine the filtering of each index to ensure the optimal execution sequence of indexes, or quicker reversion to table space scan if no filtering index exists.

This enhancement does not require any usage of REOPT bind parameters and therefore avoids any reoptimization overhead at execution time.

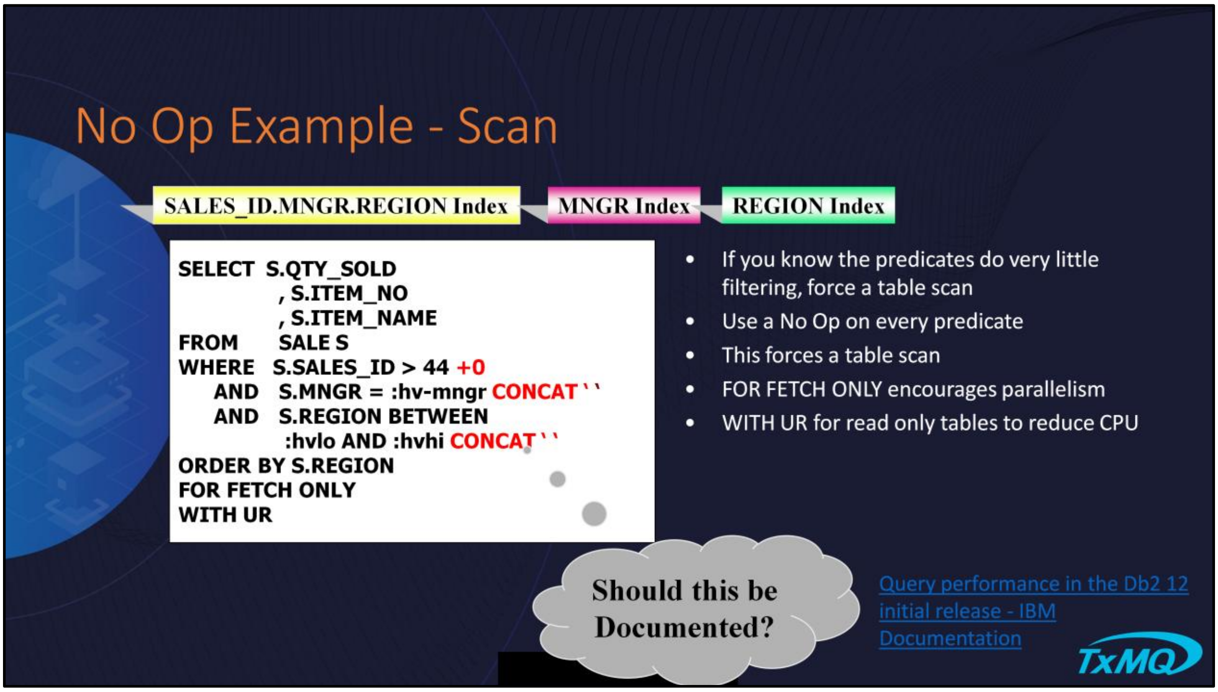
Matching index access against a non-clustered index is way more efficient than List Prefetch even with extensive random I/O.



Rarely used but very powerful so be careful.



Multiple Index Access is great when you are wanting the entire result set. If one of the legs has no chance of performing (it qualifies far more rows that the other queryblocks), block Matching Access (only from the Optimizer) but still allow all the filtering on other indexes Leafpages.



Query performance in the Db2 12 initial release - IBM Documentation

•Automated statistics collection

Db2 12 introduces several enhancements that help to automate the collection of statistics.

•Static plan stability enhancements

Db2 12 introduces improvements to the usability of static plan stability features.

•Query performance enhancements

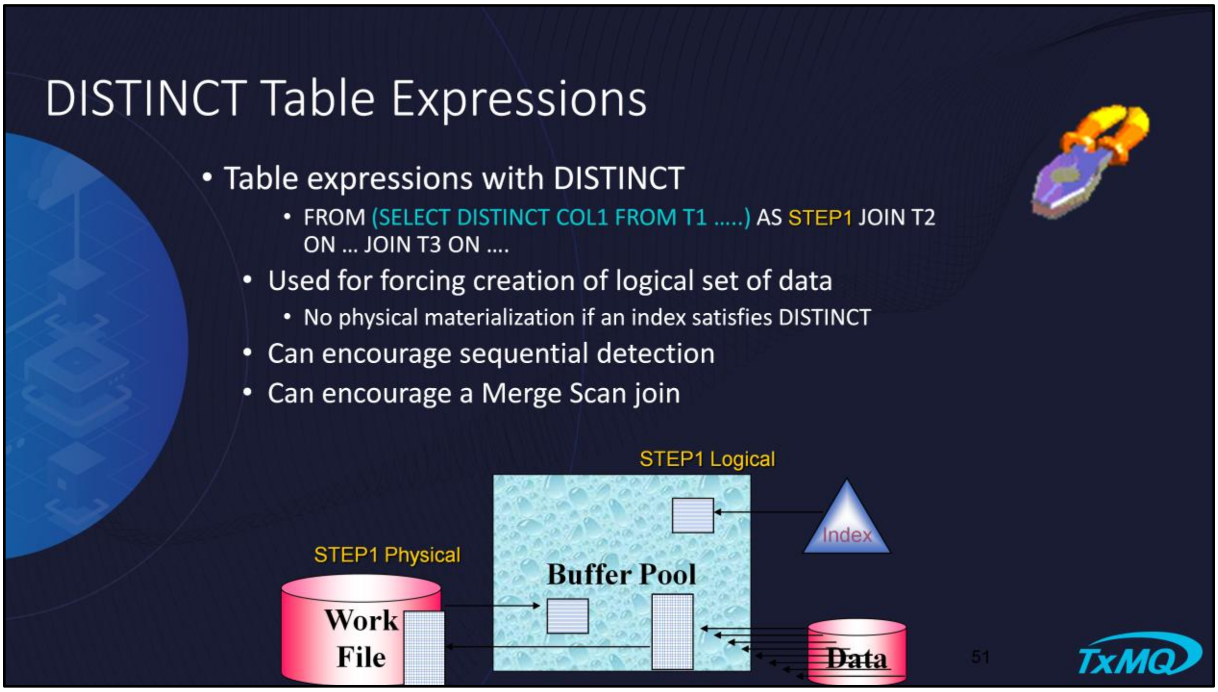
Db2 12 introduces performance enhancements for queries that use any of the following: outer joins, UNION ALL, archive transparency, system-period temporal tables.

•User-defined table function performance improvements

The merge capabilities of user-defined table functions are enhanced in Db2 12 to be similar to the capabilities of views.

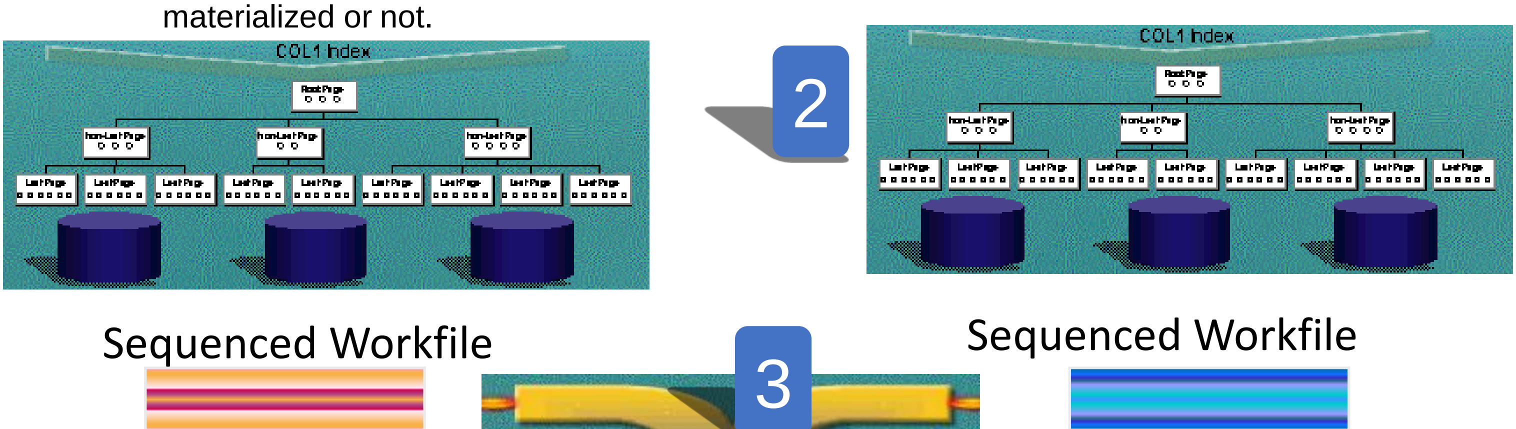
•Improved performance and reliability of index access with list prefetch

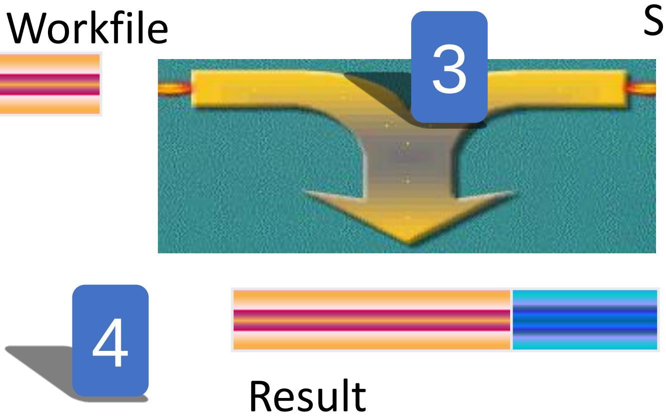
Adaptive index is a Db2 12 enhancement to multi-index and single index list prefetch-based plans that introduces logic at execution time to determine the filtering of each index to ensure the optimal execution sequence of indexes, or quicker reversion to table space scan if no filtering index exists.



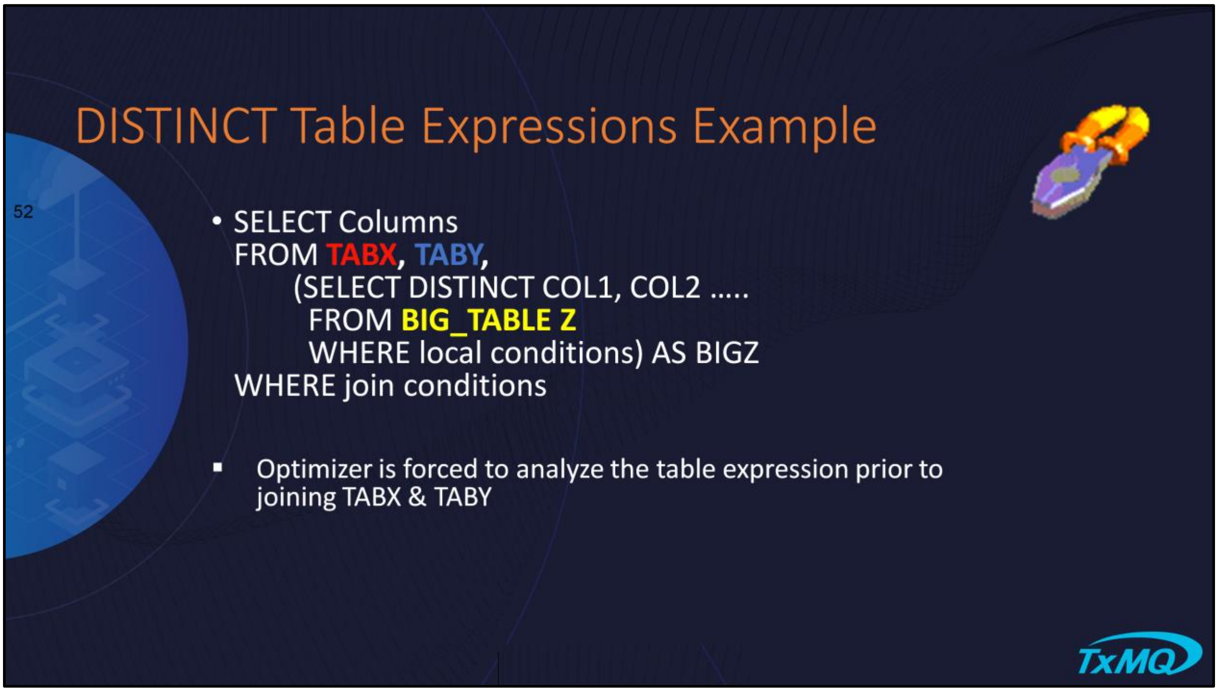
Discovered this technique when a customer called and asked how they could get their batch Merge Scan Join back into production. A migration changed it to a Nested Look Join.

Merge Scan Join favors join sequence workfiles, coming from indexes either clustered or not. The DISTINCT clause requires a sorted worklfile whether it is materialized or not.



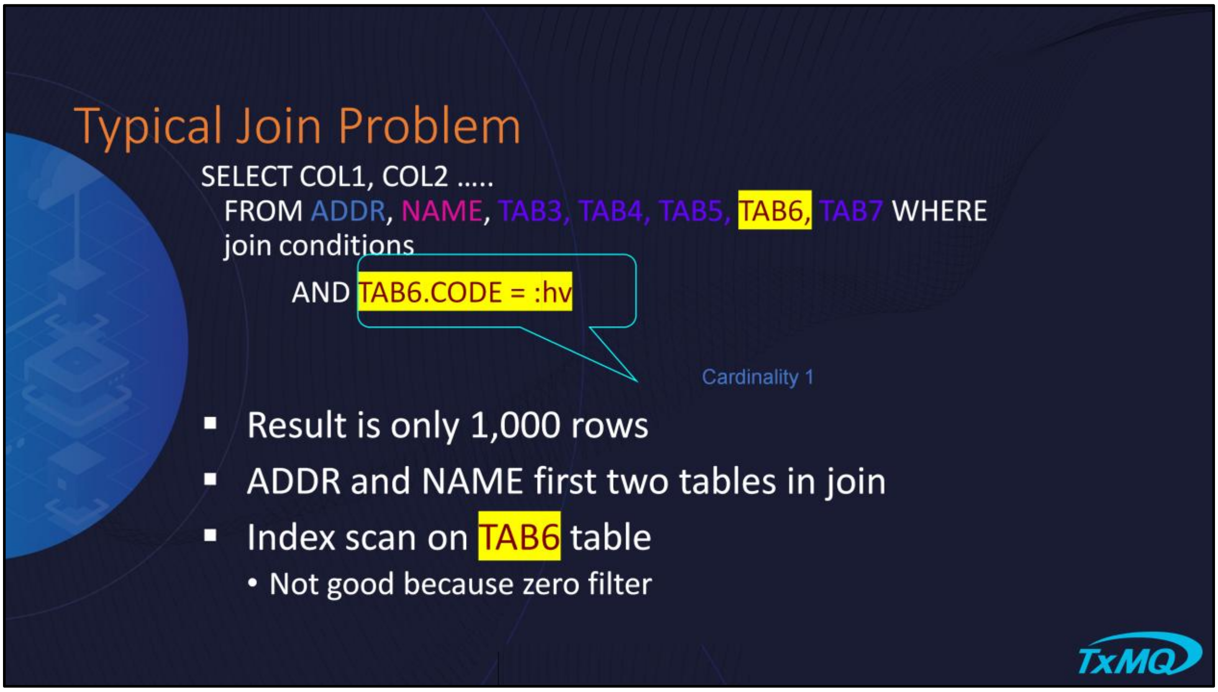


1. Access outer table using the most efficient single table access path for applying all outer table filters, a sort of these rows may be required to match the join column(s) sequence
2. Access inner table using the most efficient single table access path for applying all inner table filters, a sort of these rows may be required to match the join column(s) sequence
3. Perform match-merge check to join outer and inner table rows

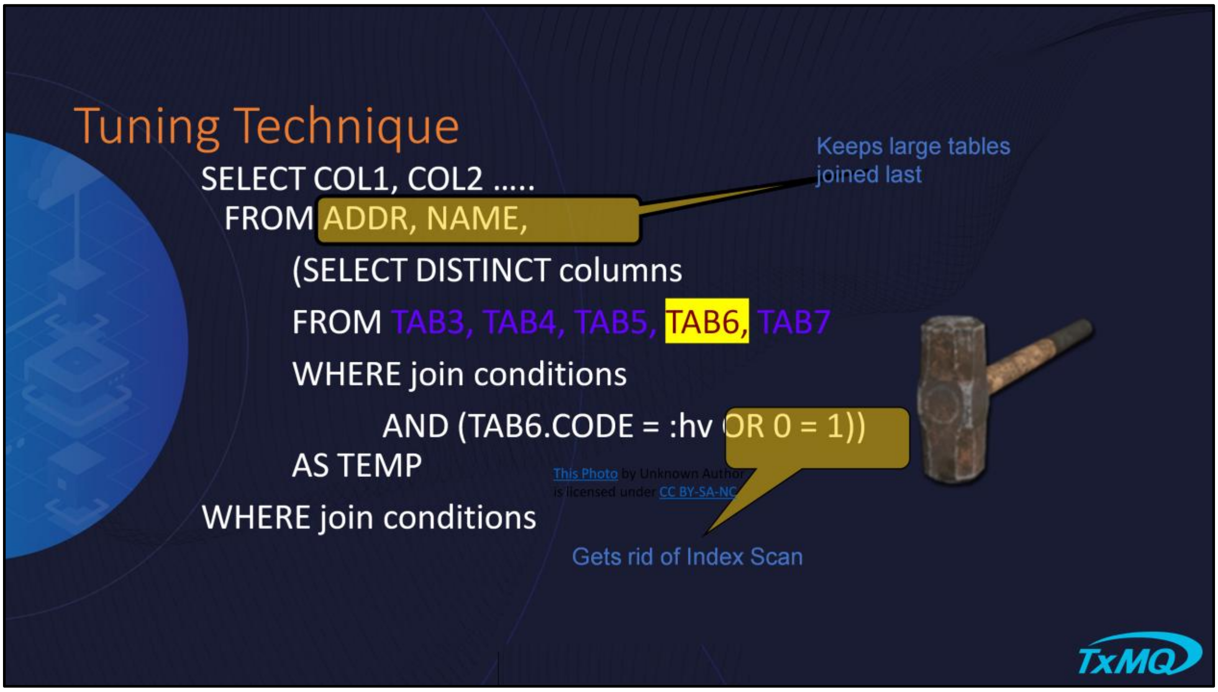


This was a batch query that had a non-optimal join sequence. The Db2 Optimizer chose to go the little tables first and then got to the 6.6B row table.

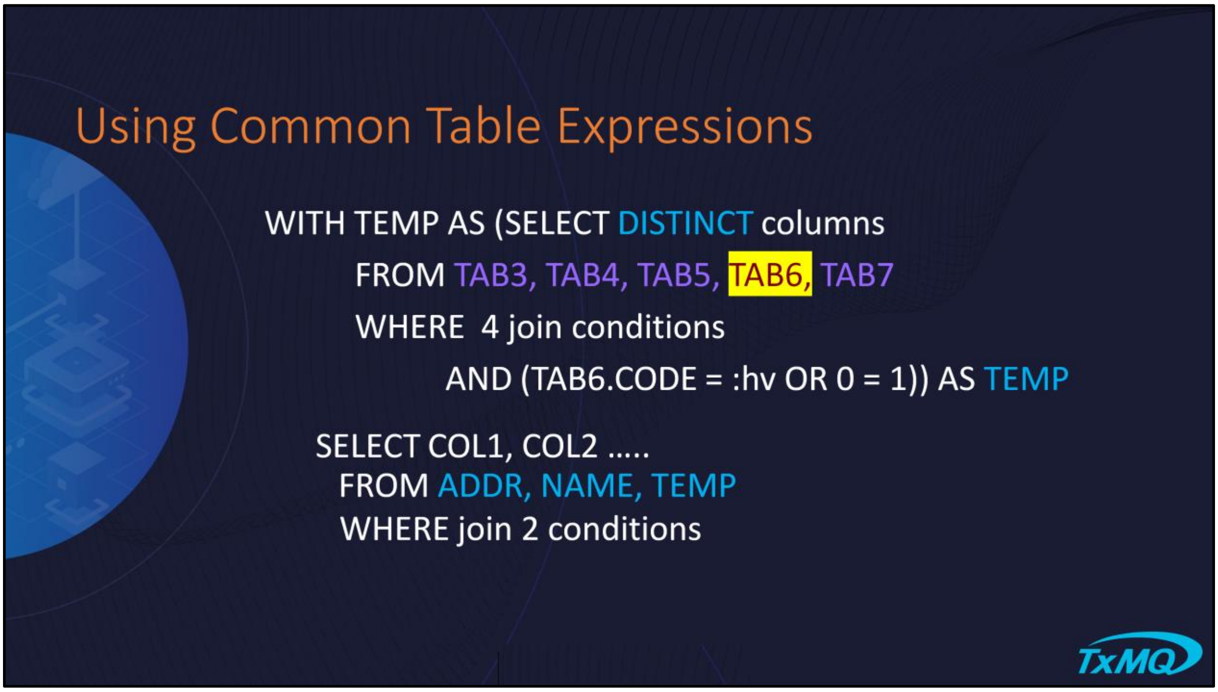
- BIG_TABLE is accessed first
  - Possibly results in materialized and sorted BIGZ workfile if DISTINCT cannot be satisfied using an index
- Great for tuning dynamic queries!



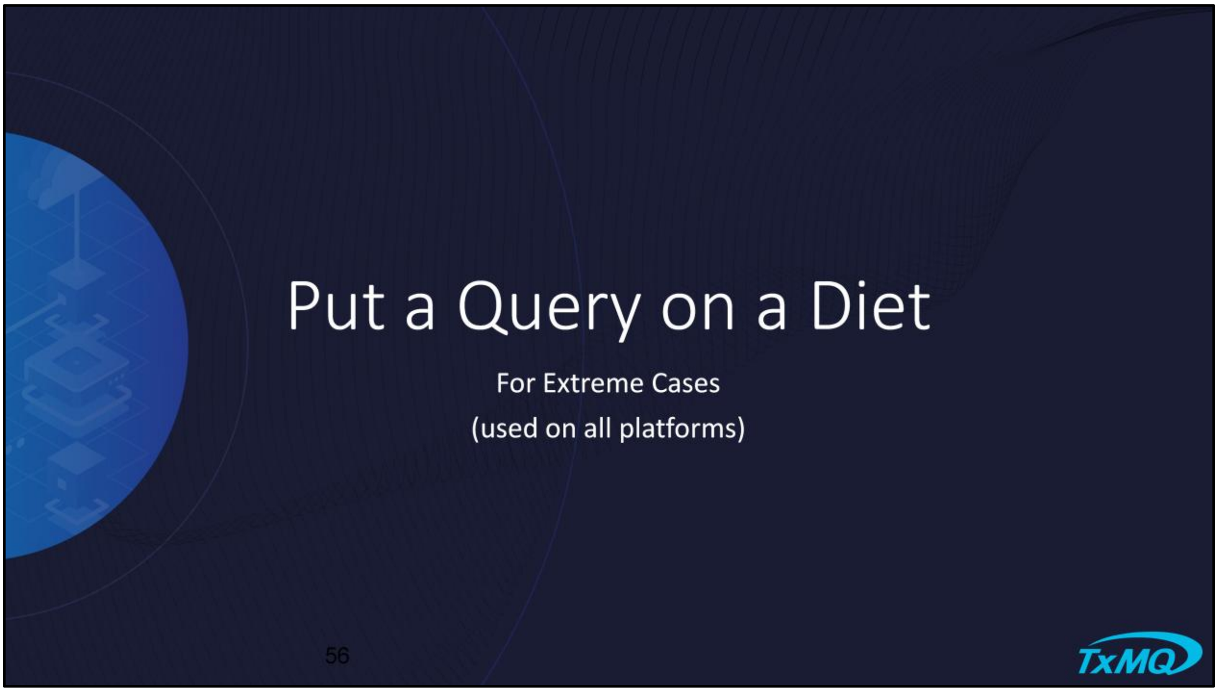
Advanced SQL Tuning class assignment. Students got really creative….

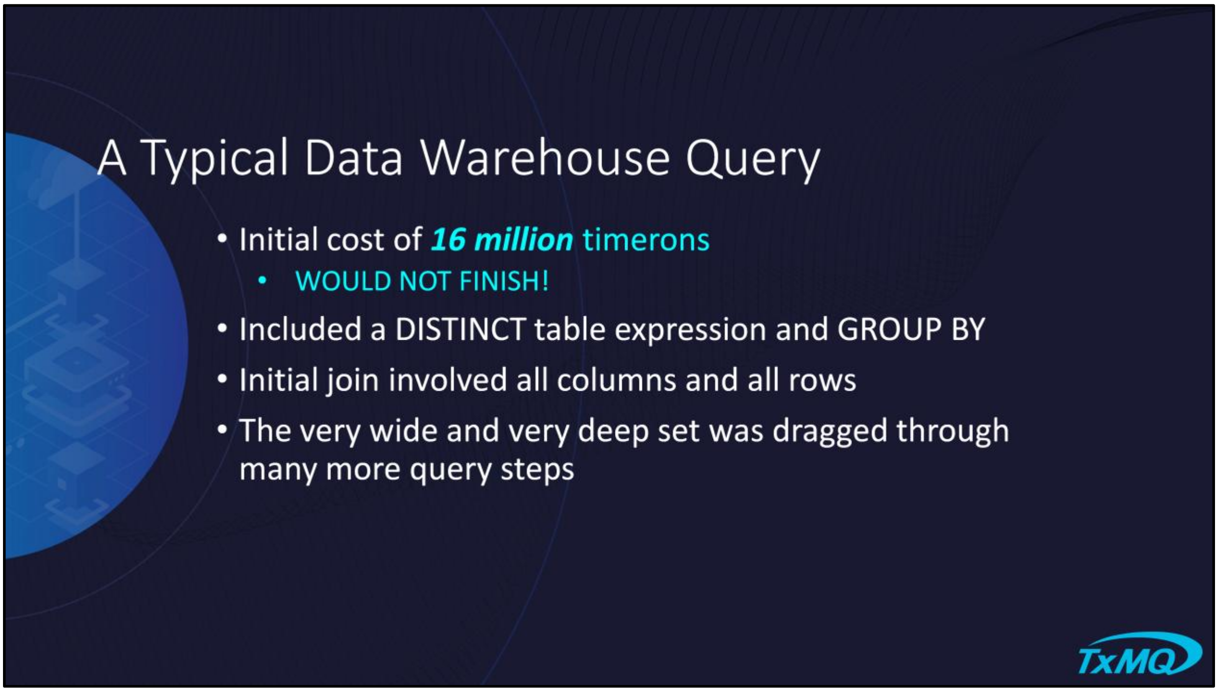


BIG tables held to be last involved in the Join

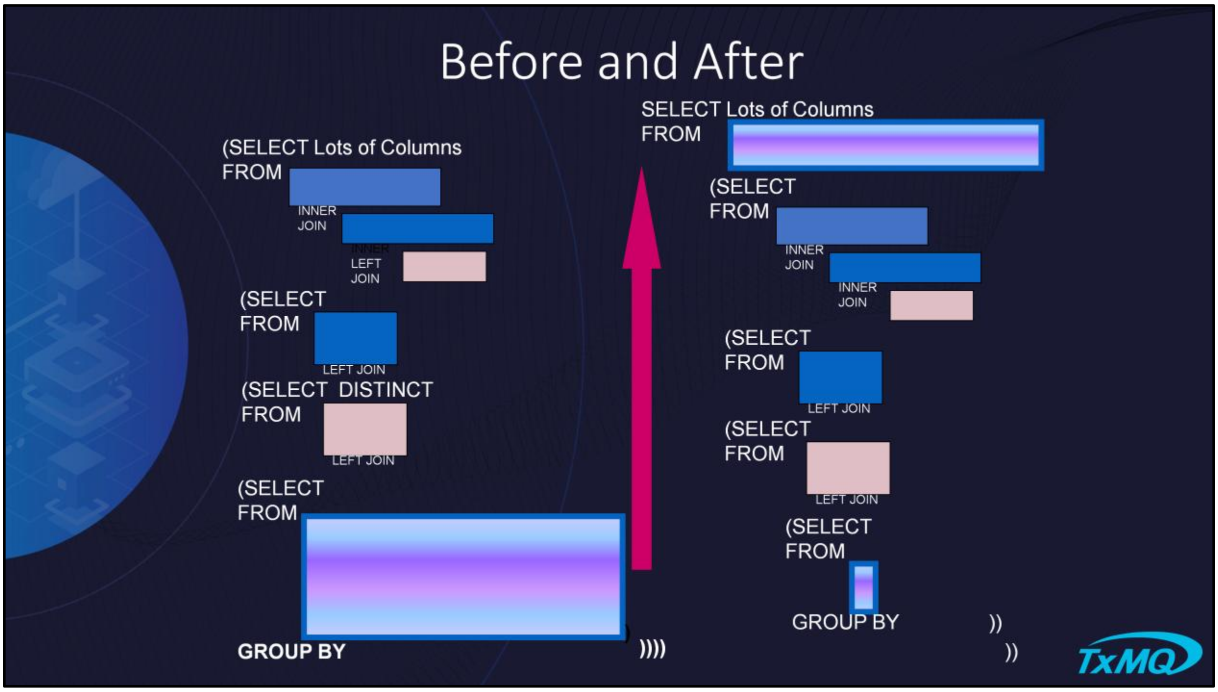


For the newbies.


Put a Query on a Diet
For Extreme Cases
(used on all platforms)
56
TxMQ

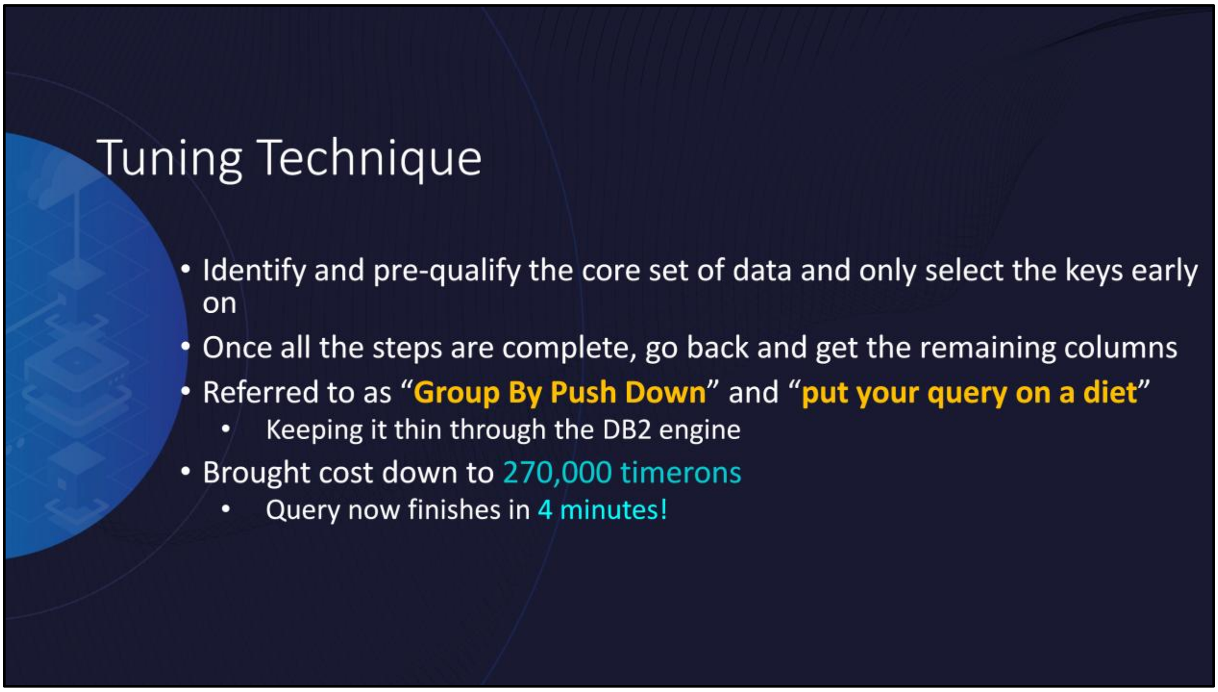


As queries get more complex, intra query optimization becomes necessary. Cross queryblock knowledge can greatly assist the optimizer in query rewrite. Right now, this is a manual rewrite process. One example is a statement that contained multiple UNION ALL subselects with the initial subselect involving and join requesting most rows all columns. This very wide and very deep set was dragged through many query steps.

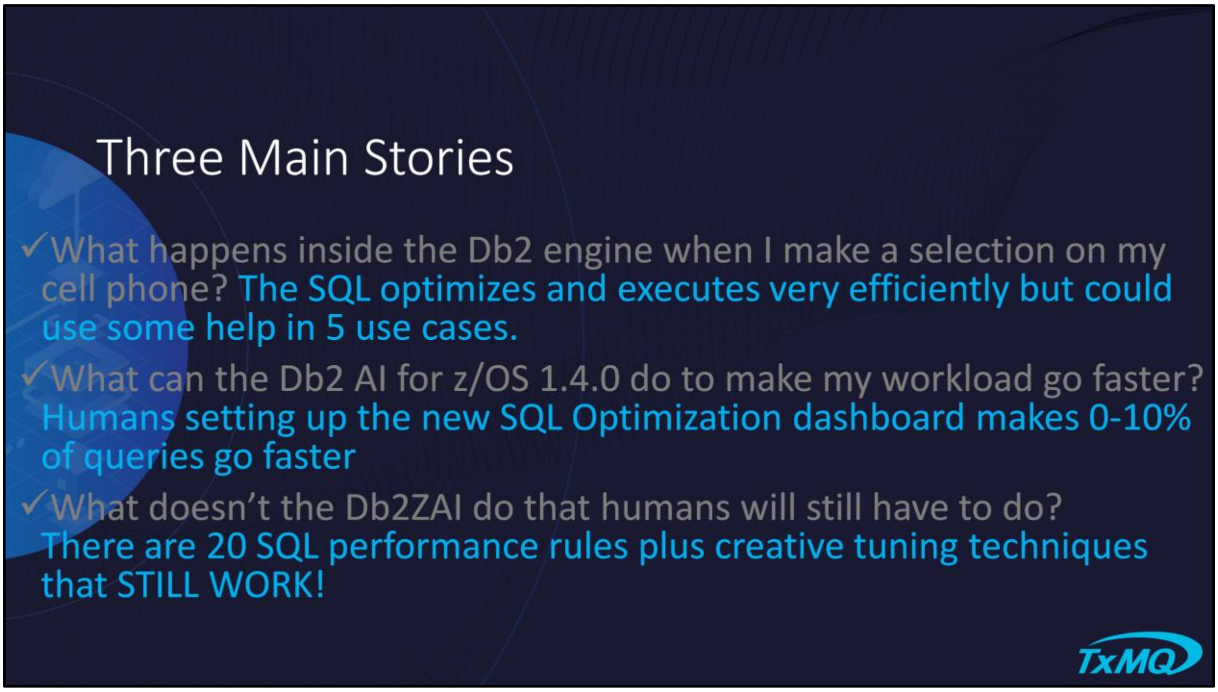


The outermost query block, the last step, requested a GROUP BY. This query would not even finish and the timeron value was over 16 million. To manually rewrite this query, the largest block was analyzed for the columns required for GROUP BY and remaining LEFT JOINs. The initial SELECT list for that really wide table expression was then manually pruned down to SELECT only critical columns, but all rows. This essentially put the subselect on a diet so that the next 5 join steps were much narrower. The final step was then rewritten to join back to the tables to get the remaining SELECT list columns. This did increase the number of times that main set was accessed but the savings from the wide joins more than offset the cost.

Further analysis was done on the GROUP BY columns. It was determined that the only columns needed in the GROUP BY calculation were from the main set. The GROUP BY operation was moved from the outer most step and pushed into the first table expression. This greatly reduced the cost of the GROUP BY operation since it did not involve many columns.

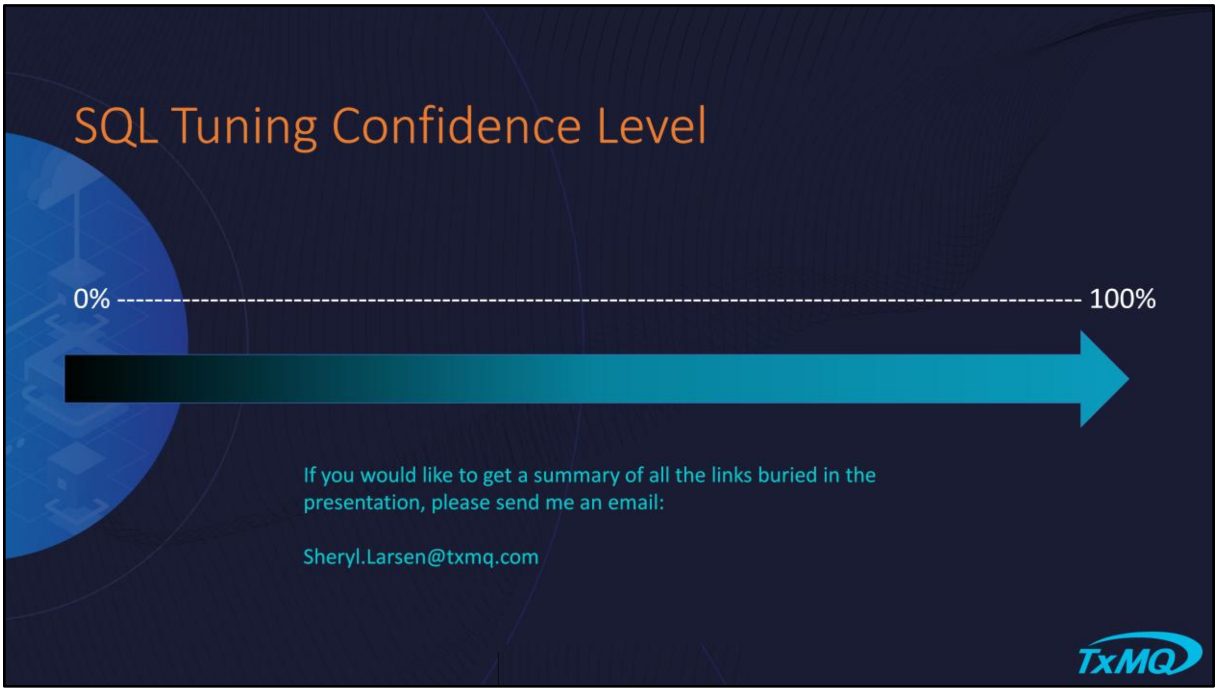


The query now finishes and the timerons were reduced to ***.27 million***. This technique of keeping the query thin through the DB2 engine has to be accomplished through manual query rewrite for now. Start by identifying the core set of data and only select the keys and grouping columns early on. Once all the step are complete, go back and get the remaining columns.



If you would like to get all the summary of all links buried in the presentation, please send me an email

Sheryl.Larsen@txmq.com



I hope you increased your SQL tuning confidence. If not, rinse and repeat (maybe a little slower the second time)

Sheryl M. Larsen
Senior DB2 Consultant

Mobile: (630) - 780 - 7641
sheryl.larsen@txmq.com
LinkedIn • Twitter • Facebook

TxMQ, Inc.
Imagine, Transform, Engage
TxMQ.com • TxMQStaffing.comTxMQ is a Premier IBM Business Partner

Sign up for our TxMQ Newsletter to stay on top of the latest in Technology, Services and Staffing!

Db2 12 for z/OS Catalog Tables - BMC Blogs - BMC Software
A new interactive Catalog application from BMC:

DB2 12 for z Optimizer (ibm.com)

Db2 AI for z/OS 1.4.0 - IBM Documentation

The Big Old Mainframe: DB2 Bind process

Overview of IBM Db2 AI for z/OS - IBM Documentation

Db2 12 for z/OS: SQL Reference (ibm.com)

DSN_STATEMENT_CACHE_TABLE - IBM Documentation

Summary of predicate processing - IBM Documentation

Automated statistics collection - IBM Documentation

IBM Data Virtualization Manager for z/OS | IBM Redbooks

Improved performance and reliability of index access with list prefetch - IBM Documentation

Query performance in the Db2 12 initial release - IBM Documentation

Data Tech Summit Presented Live Demo but recorded from Silicon Valley Lab
October 5-7, 2021
Join session in Channel 1