

**IDUG**  
**VIRTUAL**  
2021 EMEA Db2 Tech Conference

**IDUG**

**TxMQ**

**SQL Tuning that Still Works!**

Sheryl M. Larsen

TxMQ

Even with the Intelligence of Db2ZAI

#IDUGdb2

Platform: z/OS

***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.

## Three Main Stories

1. What happens inside the Db2 engine when I make a selection on my cell phone?
2. What can the Db2 AI for z/OS 1.4.0 do to make my workload go faster?
3. What doesn't the Db2ZAI do that humans will still have to do?



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.

# First SQL Class Db2 V1



```
SELECT      O.ORDERID, C.CUSTOMERID,
            B.BILL, SUM(B.AMOUNT) AS TOTAL
FROM        ORDER O, CUSTOMER C, BILL B
WHERE       B.DATE > '01-01-2017'
           AND O.ODERID = C.ORDERID
           AND C.CUSTOMERID = B.CUSTOMERID
GROUP BY    O.ORDERID, C. CUSTOMERID, B.BILL
HAVING      TOTAL > 100000
ORDER BY    TOTAL DESC
```



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.

1984

- No internet
- No cell phone
- No laptop
- No ear buds
- No email
- No DB2 Yet
- ...



• [rldicl \(Rotate Left Double Word Immediate then Clear Left\) instruction](#)

• [rlmi \(Rotate Left Then Mask Insert\) instruction](#)

• [rrib \(Rotate Right and Insert Bit\) instruction](#)

• [sld \(Shift Left Double Word\)](#)

• [sle \(Shift Left Extended\) instruction](#)

• [sleg \(Shift Left Extended with MQ\)](#)  
[instruction](#)

• [sliq \(Shift Left Immediate with MQ\)](#)  
[instruction](#)

• [slliq \(Shift Left Long Immediate with MQ\)](#)



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.

# Fast Forward



[Db2 12 for z/OS: SQL Reference \(ibm.com\)](#)

37  
Y  
E  
A  
R  
S

1984  
1989  
2003  
2010  
2015  
2019  
2020  
2021

**Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, ORDER BY,** Complex Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions including SQL/XML, Limited Fetch, Insensitive Scroll Cursors, UNION Everywhere, MIN/MAX Single Index, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions, 2M Statement Length, GROUP BY Expression, Sequences, Scalar Fullselect, Materialized Query Tables, Common Table Expressions, Recursive SQL, CURRENT PACKAGE PATH, VOLATILE Tables, Star Join, Sparse Index, Qualified Column names, Multiple DISTINCT clauses, ON COMMIT DROP, Transparent ROWID Column, Call from trigger, statement isolation, FOR READ ONLY KEEP UPDATE LOCKS, SET CURRENT SCHEMA, Client special registers, long SQL object names, SELECT from INSERT, UPDATE or DELETE, INSTEAD OF TRIGGER, SQL PL in routines, BIGINT, file reference variables, XML, FETCH FIRST & ORDER BY in subselect & fullselect, caseless comparisons, INTERSECT, EXCEPT, MERGE not logged tables, OmniFind, spatial, range partitions, data compression, DECFLOAT, optimistic locking, ROLE, TRUNCATE, index & XML compression, created temps, inline LOB, administrative privileges, implicit cast, increased timestamp precision, currently committed, moving sum & average, index include columns, row and column access controls, time travel query, GROUPING SETS, ROLLUP, CUBE, global variables, Text Search functions, accelerated tables, DROP COLUMN, array data type, XML enhancements, moving SUM/AVG, Array variables, ARRAY\_EXISTS, COUNTBIG, SELECT INTO statements with UNION or UNION ALL allowed, SELECT INTO statements with UNION or UNION ALL disallowed, OFFSET, 11 global variables, LISTAGG + 33 more BIFs, LIMIT fetch row count1

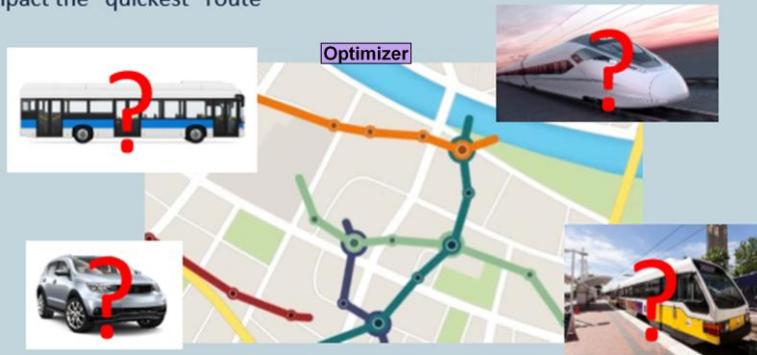


Check current IBM SQL Reference for a complete list:

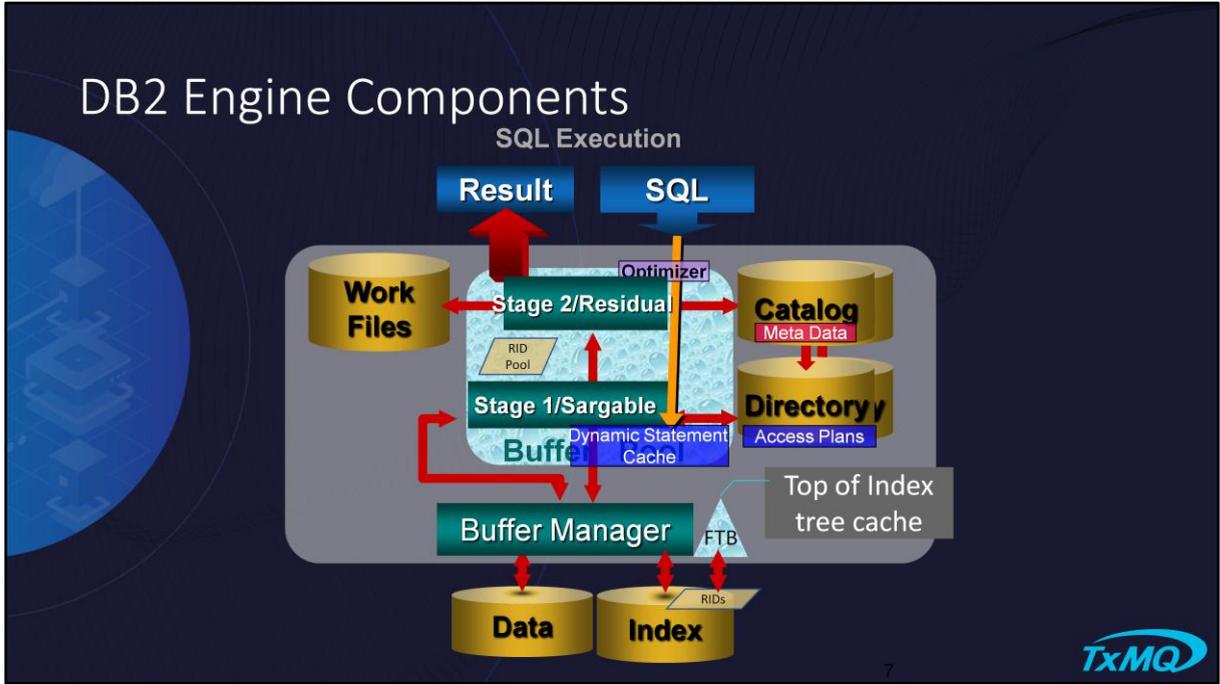
[Db2 12 for z/OS: SQL Reference \(ibm.com\)](#)

## Cost based Optimizer figures out how to get the data

- The optimizer is responsible for
  - Choosing the most efficient method of accessing the data for a given SQL statement
- Think of your transportation choices
  - Start/end location, time of day, construction, traffic, available options/routes
    - All can impact the “quickest” route

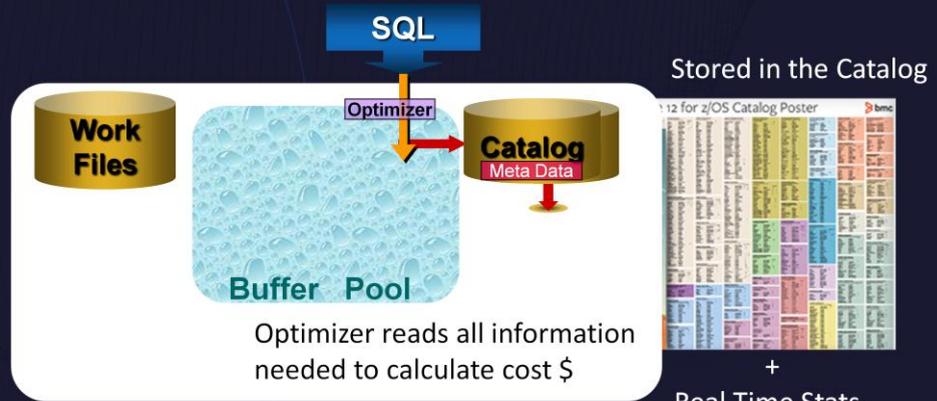


The DB2 Optimizer is a cost-based optimizer, which means it calculates the cost of multiple access paths (in a unit called Timerons), and then chooses the cheapest access path. One of the advantages of the SQL language is that it means we don't have to know where a particular piece of data lives on disk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



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

# Meta Data – Everything known about each object



[Db2 12 for z/OS Catalog Tables - BMC Blogs - BMC Software](#)



A new interactive Catalog application from BMC:

[Db2 12 for z/OS Catalog Tables - BMC Blogs - BMC Software](#)

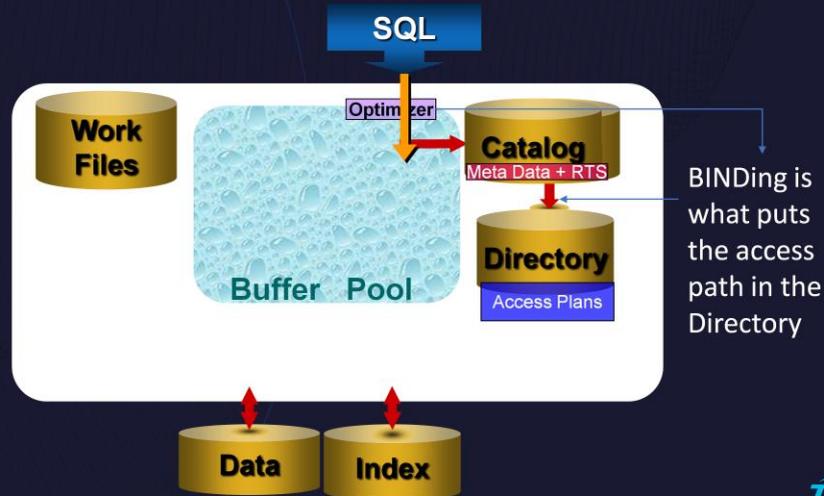


Order Free Db2 Reference Manual & Poster >

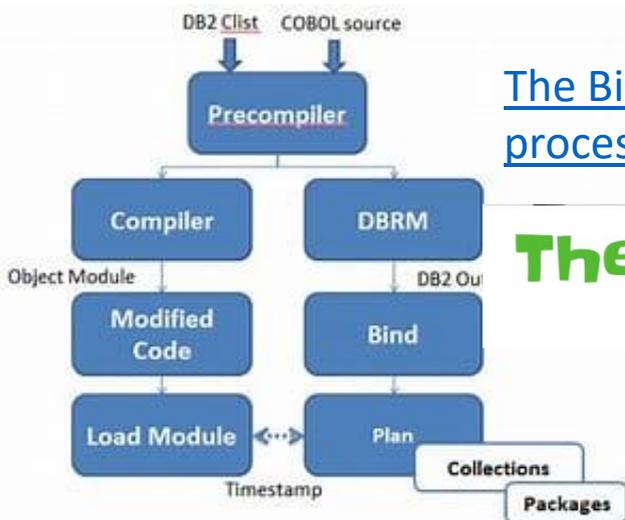
## Db2 12 for z/OS Catalog Tables

<p><b>SYSIBM.SYSROUTINEAUTH</b></p> <p>Records the privileges that are held by users on routines.</p> <table border="1"> <tr><td>GRANTOR</td><td>VARCHAR(128)</td></tr> <tr><td>GRANTEE</td><td>VARCHAR(128)</td></tr> <tr><td>SCHEMA</td><td>VARCHAR(128)</td></tr> <tr><td>SPECIFICNAME</td><td>VARCHAR(128)</td></tr> <tr><td>GRANTEDTS</td><td>TIMESTAMP</td></tr> <tr><td>ROUTINETYPE</td><td>CHAR(1)</td></tr> <tr><td>GRANTEETYPE</td><td>CHAR(1)</td></tr> <tr><td>AUTHHOWGOT</td><td>CHAR(1)</td></tr> <tr><td>EXECUTEAUTH</td><td>CHAR(1)</td></tr> <tr><td>COLLID</td><td>VARCHAR(128)</td></tr> </table>	GRANTOR	VARCHAR(128)	GRANTEE	VARCHAR(128)	SCHEMA	VARCHAR(128)	SPECIFICNAME	VARCHAR(128)	GRANTEDTS	TIMESTAMP	ROUTINETYPE	CHAR(1)	GRANTEETYPE	CHAR(1)	AUTHHOWGOT	CHAR(1)	EXECUTEAUTH	CHAR(1)	COLLID	VARCHAR(128)	<p><b>SYSIBM.SYSJARCONTENTS</b></p> <p>Java class source for installed jar.</p> <table border="1"> <tr><td>JARSCHEMA</td><td>VARCHAR(128)</td></tr> <tr><td>JAR_ID</td><td>VARCHAR(128)</td></tr> <tr><td>CLASS</td><td>VARCHAR(384)</td></tr> <tr><td>CLASS_SOURCE_ROWID</td><td>ROWID</td></tr> <tr><td>CLASS_SOURCE</td><td>CLOB(10M)</td></tr> <tr><td>IBMREQD</td><td>CHAR(1)</td></tr> </table> <p><b>SYSIBM.SYSJAROBJECTS</b></p> <p>Binary large object representing</p>	JARSCHEMA	VARCHAR(128)	JAR_ID	VARCHAR(128)	CLASS	VARCHAR(384)	CLASS_SOURCE_ROWID	ROWID	CLASS_SOURCE	CLOB(10M)	IBMREQD	CHAR(1)	<p><b>SYSIBM.SYSFOREIGNKEYS</b></p> <p>One row for every column of every foreign key.</p> <table border="1"> <tr><td>CREATOR</td><td>VARCHAR(128)</td></tr> <tr><td>TBNAME</td><td>VARCHAR(128)</td></tr> <tr><td>RELNAME</td><td>VARCHAR(128)</td></tr> <tr><td>COLNAME</td><td>VARCHAR(128)</td></tr> <tr><td>COLNO</td><td>SMALLINT</td></tr> <tr><td>COLSEQ</td><td>SMALLINT</td></tr> <tr><td>IBMREQD</td><td>CHAR(1)</td></tr> </table> <p><b>SYSIBM.SYSINDEXPART</b></p>	CREATOR	VARCHAR(128)	TBNAME	VARCHAR(128)	RELNAME	VARCHAR(128)	COLNAME	VARCHAR(128)	COLNO	SMALLINT	COLSEQ	SMALLINT	IBMREQD	CHAR(1)	<p><b>SYSIBM.SYSPLAN</b></p> <p>One row for each application plan.</p> <table border="1"> <tr><td>NAME</td><td>VARCHAR(24)</td></tr> <tr><td>CREATOR</td><td>VARCHAR(128)</td></tr> <tr><td>VALIDATE</td><td>CHAR(1)</td></tr> <tr><td>ISOLATION</td><td>CHAR(1)</td></tr> <tr><td>VALID</td><td>CHAR(1)</td></tr> <tr><td>OPERATIVE</td><td>CHAR(1)</td></tr> <tr><td>PLSIZE</td><td>INTEGER</td></tr> <tr><td>IBMREQD</td><td>CHAR(1)</td></tr> <tr><td>AVGSIZE</td><td>INTEGER</td></tr> <tr><td>ACQUIRE</td><td>CHAR(1)</td></tr> </table>	NAME	VARCHAR(24)	CREATOR	VARCHAR(128)	VALIDATE	CHAR(1)	ISOLATION	CHAR(1)	VALID	CHAR(1)	OPERATIVE	CHAR(1)	PLSIZE	INTEGER	IBMREQD	CHAR(1)	AVGSIZE	INTEGER	ACQUIRE	CHAR(1)	<p><b>SYSIBM.SYSCOPY</b></p> <p>Contains information needed for recovery.</p> <table border="1"> <tr><td>DBNAME</td><td>CHAR(8)</td></tr> <tr><td>TSNAME</td><td>CHAR(8)</td></tr> <tr><td>DSNUM</td><td>INTEGER</td></tr> <tr><td>ICTYPE</td><td>CHAR(1)</td></tr> <tr><td>START_RBA</td><td>CHAR(10)</td></tr> <tr><td>FILESEQNO</td><td>INTEGER</td></tr> <tr><td>DEVTYPE</td><td>CHAR(8)</td></tr> <tr><td>IBMREQD</td><td>CHAR(1)</td></tr> <tr><td>DSNAME</td><td>CHAR(44)</td></tr> <tr><td>SHRLEVEL</td><td>CHAR(1)</td></tr> </table>	DBNAME	CHAR(8)	TSNAME	CHAR(8)	DSNUM	INTEGER	ICTYPE	CHAR(1)	START_RBA	CHAR(10)	FILESEQNO	INTEGER	DEVTYPE	CHAR(8)	IBMREQD	CHAR(1)	DSNAME	CHAR(44)	SHRLEVEL	CHAR(1)
GRANTOR	VARCHAR(128)																																																																																									
GRANTEE	VARCHAR(128)																																																																																									
SCHEMA	VARCHAR(128)																																																																																									
SPECIFICNAME	VARCHAR(128)																																																																																									
GRANTEDTS	TIMESTAMP																																																																																									
ROUTINETYPE	CHAR(1)																																																																																									
GRANTEETYPE	CHAR(1)																																																																																									
AUTHHOWGOT	CHAR(1)																																																																																									
EXECUTEAUTH	CHAR(1)																																																																																									
COLLID	VARCHAR(128)																																																																																									
JARSCHEMA	VARCHAR(128)																																																																																									
JAR_ID	VARCHAR(128)																																																																																									
CLASS	VARCHAR(384)																																																																																									
CLASS_SOURCE_ROWID	ROWID																																																																																									
CLASS_SOURCE	CLOB(10M)																																																																																									
IBMREQD	CHAR(1)																																																																																									
CREATOR	VARCHAR(128)																																																																																									
TBNAME	VARCHAR(128)																																																																																									
RELNAME	VARCHAR(128)																																																																																									
COLNAME	VARCHAR(128)																																																																																									
COLNO	SMALLINT																																																																																									
COLSEQ	SMALLINT																																																																																									
IBMREQD	CHAR(1)																																																																																									
NAME	VARCHAR(24)																																																																																									
CREATOR	VARCHAR(128)																																																																																									
VALIDATE	CHAR(1)																																																																																									
ISOLATION	CHAR(1)																																																																																									
VALID	CHAR(1)																																																																																									
OPERATIVE	CHAR(1)																																																																																									
PLSIZE	INTEGER																																																																																									
IBMREQD	CHAR(1)																																																																																									
AVGSIZE	INTEGER																																																																																									
ACQUIRE	CHAR(1)																																																																																									
DBNAME	CHAR(8)																																																																																									
TSNAME	CHAR(8)																																																																																									
DSNUM	INTEGER																																																																																									
ICTYPE	CHAR(1)																																																																																									
START_RBA	CHAR(10)																																																																																									
FILESEQNO	INTEGER																																																																																									
DEVTYPE	CHAR(8)																																																																																									
IBMREQD	CHAR(1)																																																																																									
DSNAME	CHAR(44)																																																																																									
SHRLEVEL	CHAR(1)																																																																																									

## Static SQL Access Plans Stored in Directory



## The Big Old Mainframe: DB2 Bind process

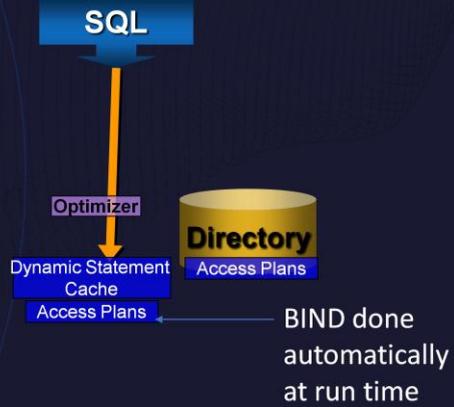


## The Big Old Mainframe

The most comprehensive resource on the web for mainframe developers

- Bind Process reads DBRM which is created in the precompiler stage and creates an 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 a 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.

Dynamic SQL is stored in the Dynamic Statement Cache

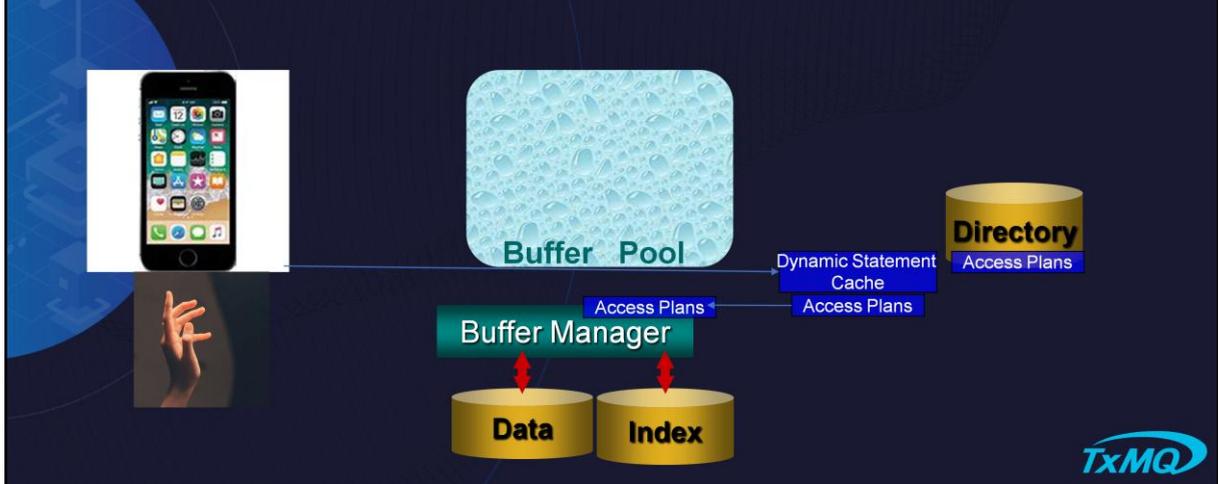


[DSN\\_STATEMENT\\_CACHE\\_TABLE - IBM Documentation](#)



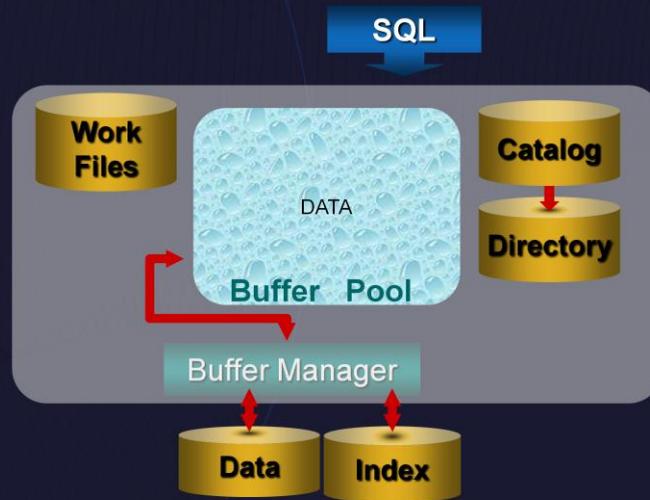
[DSN\\_STATEMENT\\_CACHE\\_TABLE - IBM Documentation](#)

Execution time is when the Access Plan is given to the Buffer Manager



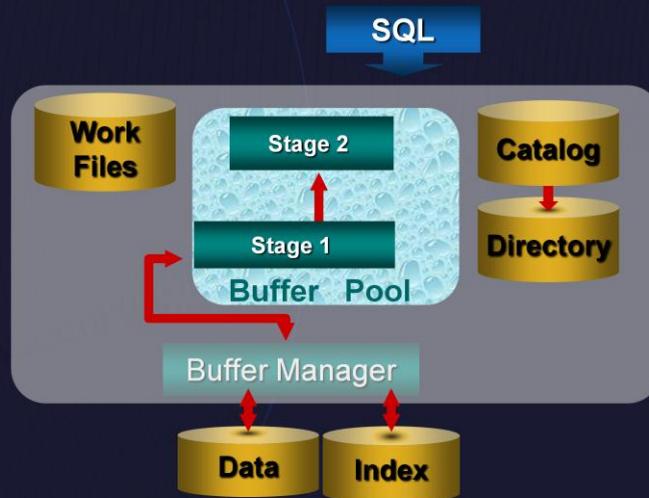
From an app on your phone, to the mainframe storing all the transaction data, your Access Plan is pulled into the Buffer Manager. This component follows the instructions to pull data from index, MQT, data into to Buffer Pool that is not already there to meet your query needs.

## Buffer Pool stores data in memory



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.

## Stage 1 & 2 filter the data



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.

## Indexable Stage 1 Predicates

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

[Summary of predicate processing - IBM Documentation](#)

## Stage 1 Predicates

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

- 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.
- 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**
- Data Screening** = The Stage 1 predicate is a candidate for filtering on the data pages. This is the third point of filtering in DB2.
- 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** <sup>31</sup>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.**

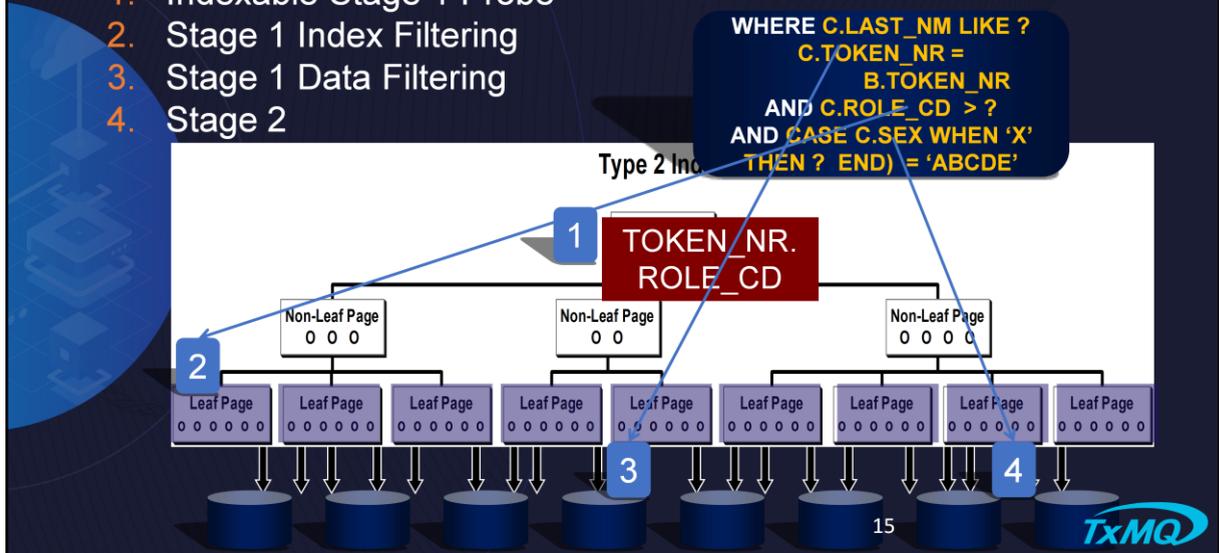
Stage 1 not indexable predicates <sup>31</sup>The following predicates might be evaluated during stage 1 processing, during index screening, or after data page access

**There are 12 of them shown in slide 19 notes.**

Stage 2 partial list is shown in slide 20 notes

## Four Points of Filtering

1. Indexable Stage 1 Probe
2. Stage 1 Index Filtering
3. Stage 1 Data Filtering
4. Stage 2

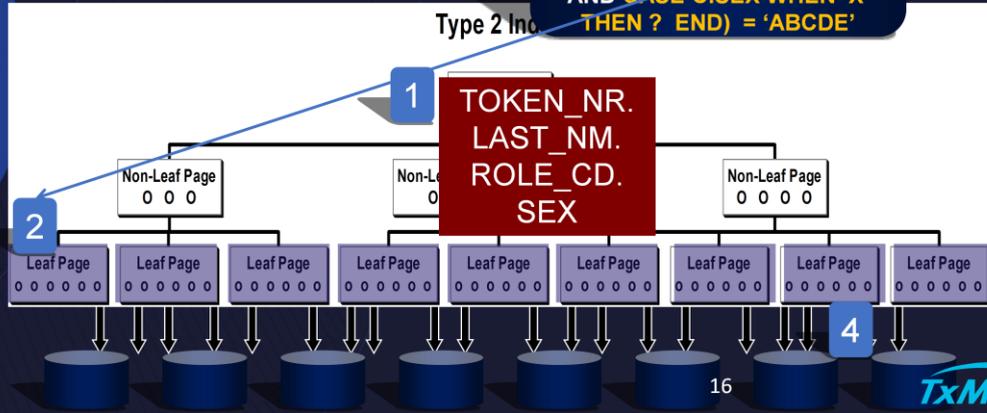


This is an example of a non-optimal index. The query has one join predicate and 3 local filters on the C table. Trouble is there are only two columns in the index leaving the filtering pushed back until steps 3 and 4. A better index would be `TOKEN_NR.LAST_NM.ROLE_CD.SEX`. This moves all the filtering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

## Four Points of Filtering

1. Indexable Stage 1 Probe
2. Stage 1 Index Filtering
3. Stage 1 Data Filtering
4. Stage 2

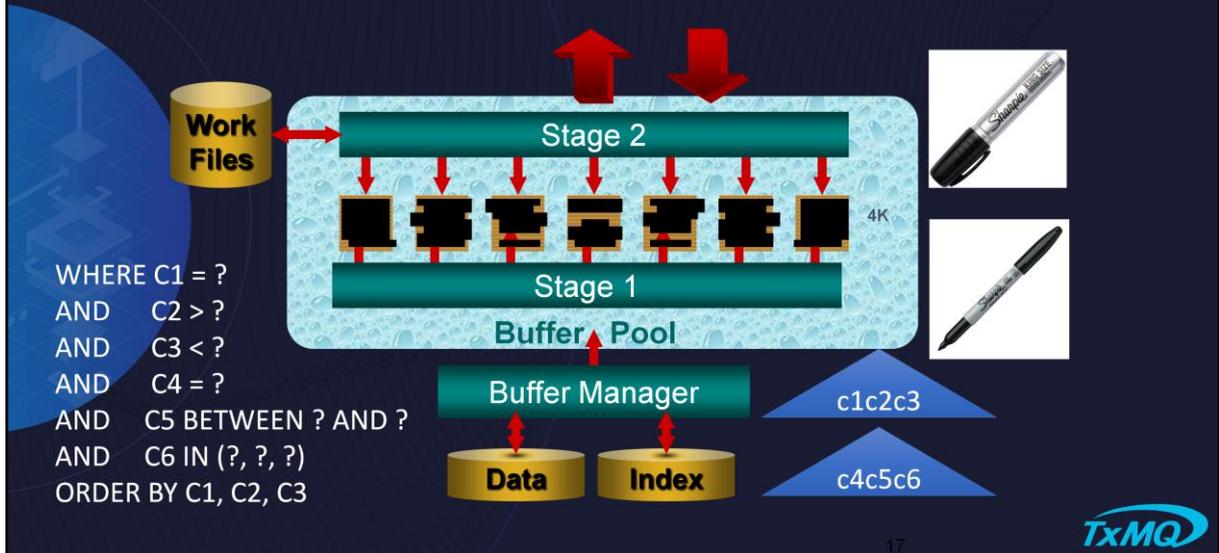
```
WHERE C.LAST_NM LIKE ?
      C.TOKEN_NR =
          B.TOKEN_NR
      AND C.ROLE_CD > ?
      AND CASE C.SEX WHEN 'X'
              THEN ? END) = 'ABCDE'
```



A better index would be `TOKEN_NR.LAST_NM.ROLE_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).

## Filtering – z/OS



Stage 1 Sharpie pen is much thinner than 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 Stage 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,...COLn) IN (cor subq)

•COL NOT IN (cor subq)

•(COL1,...COLn) NOT IN (cor subq)

•T1.COL1 IS DISTINCT FROM T2.COL2 [3](#)

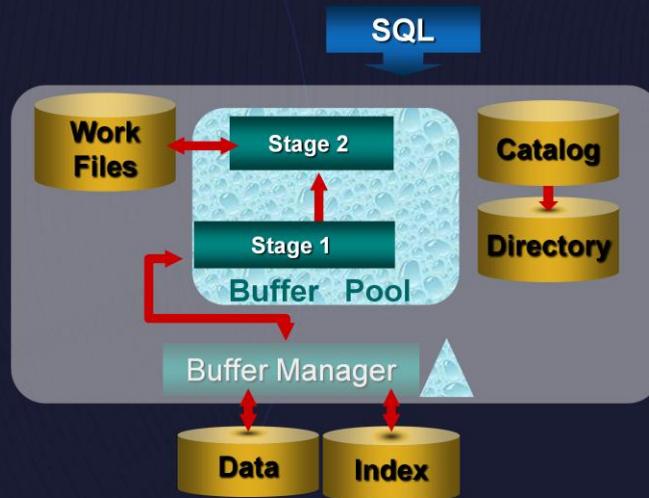
•T1.COL1 IS DISTINCT FROM T2 col expr [8, 11](#)

•COL IS NOT DISTINCT FROM (cor subq)

•EXISTS (subq) [19](#)

## If a Sort is needed for ORDER BY/GROUP BY

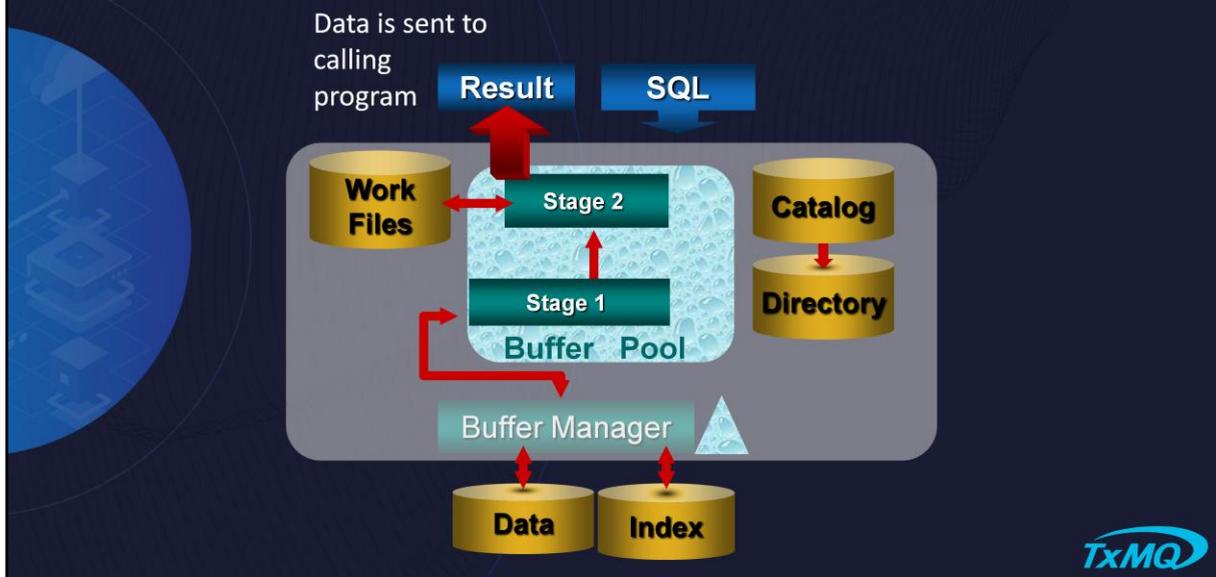
Work Files are filled with the remaining result data and sorted ... sometimes



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

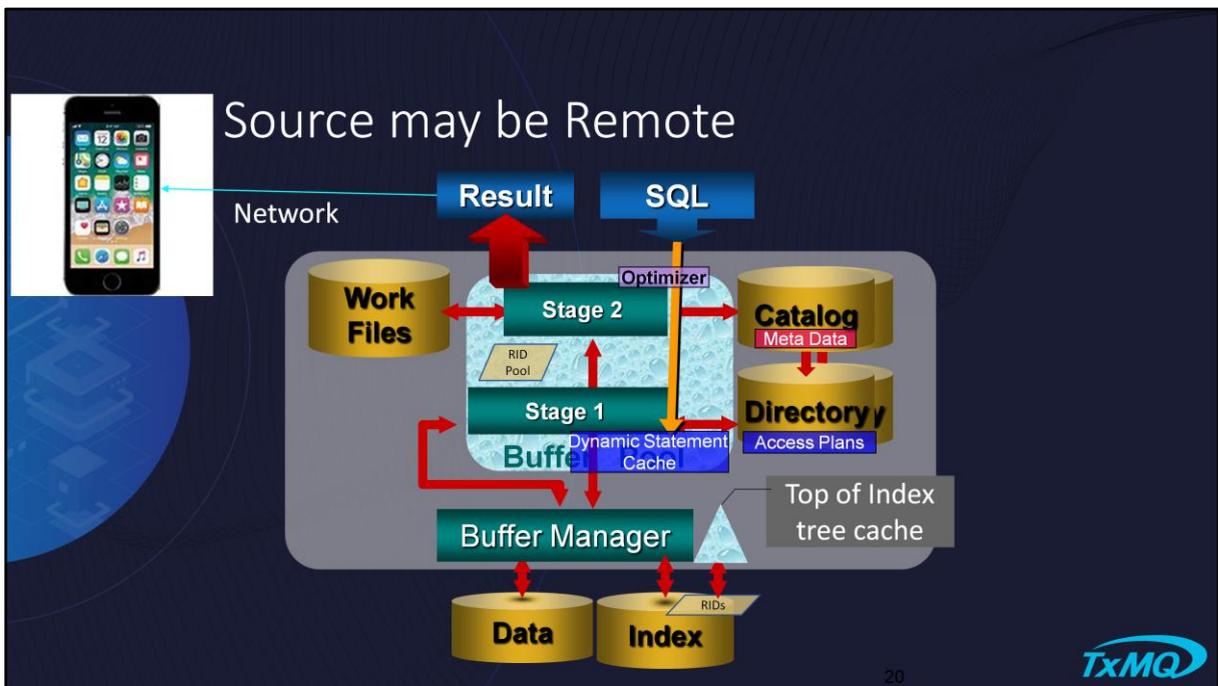
# The Result is brought back in memory



**Indexable and stage 1 predicates-** <sup>31</sup>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,...COLn) 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](#)

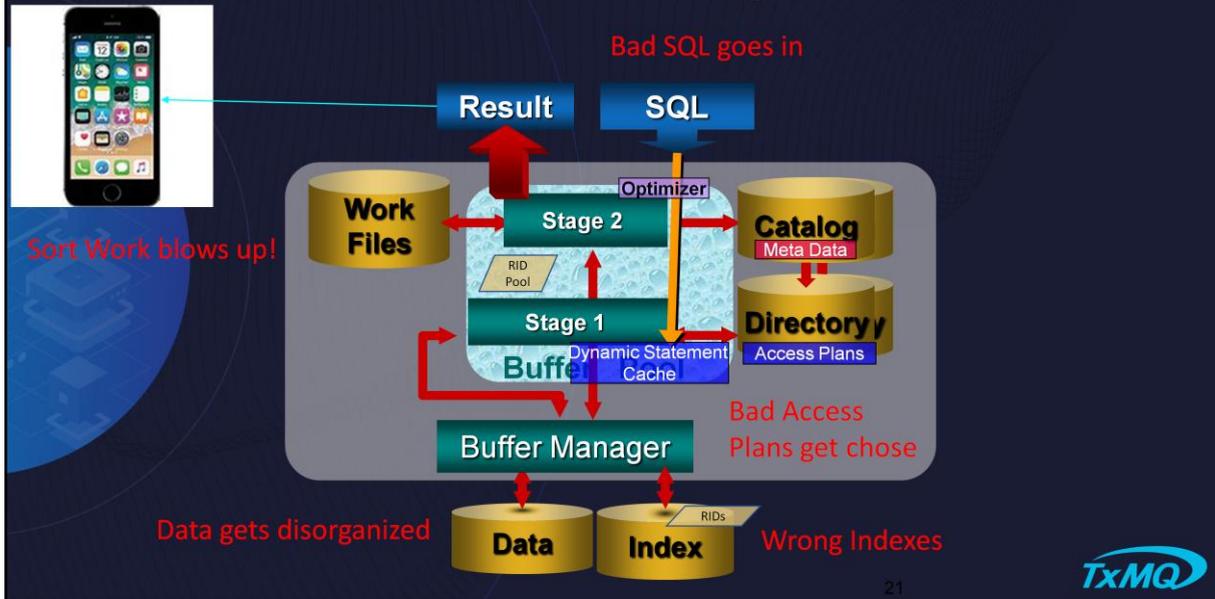


### Stage 1 not indexable predicates [31](#)

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)

# What Could Go Wrong?



21



**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,...COLn) IN (cor subq)
- COL NOT IN (cor subq)
- (COL1,...COLn) 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



# The Db2 Optimizer

## How Does it Decide so Fast?

### Good Input

- 35 years of catalog statistics refinement
- Ability to use some real time information
- Ability to refine scope of data collection- STATSFEEDBACK

### Cost-based Smarts

- 35 years of algorithm refinement
- Creates a cost model for every query
- **Defaults** are used when query values are **unknown**



How close does the optimizer get with '?' or ':hv'?

[DB2 12 for z Optimizer \(ibm.com\)](http://ibm.com)

# DB2 12 for z Optimizer

Terry Purcell





[DB2 12 for z Optimizer \(ibm.com\)](http://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

# Default Statistics

WHERE >? WHERE BETWEEN ? AND ?

COLCARDF	Factor for <, <=, >, >=	Factor for LIKE or BETWEEN
>=100,000,000	1/10,000	3/100,000
>=10,000,000	1/3,000	1/10,000
>=1,000,000	1/1,000	3/10,000
>=100,000	1/300	1/1,000
>=10,000	1/100	3/1,000
>=1,000	1/30	1/100
>=100	1/10	3/100
>=2	1/3	1/10
=1	1/1	1/1
<=0	1/3	1/10

How Close to Reality?

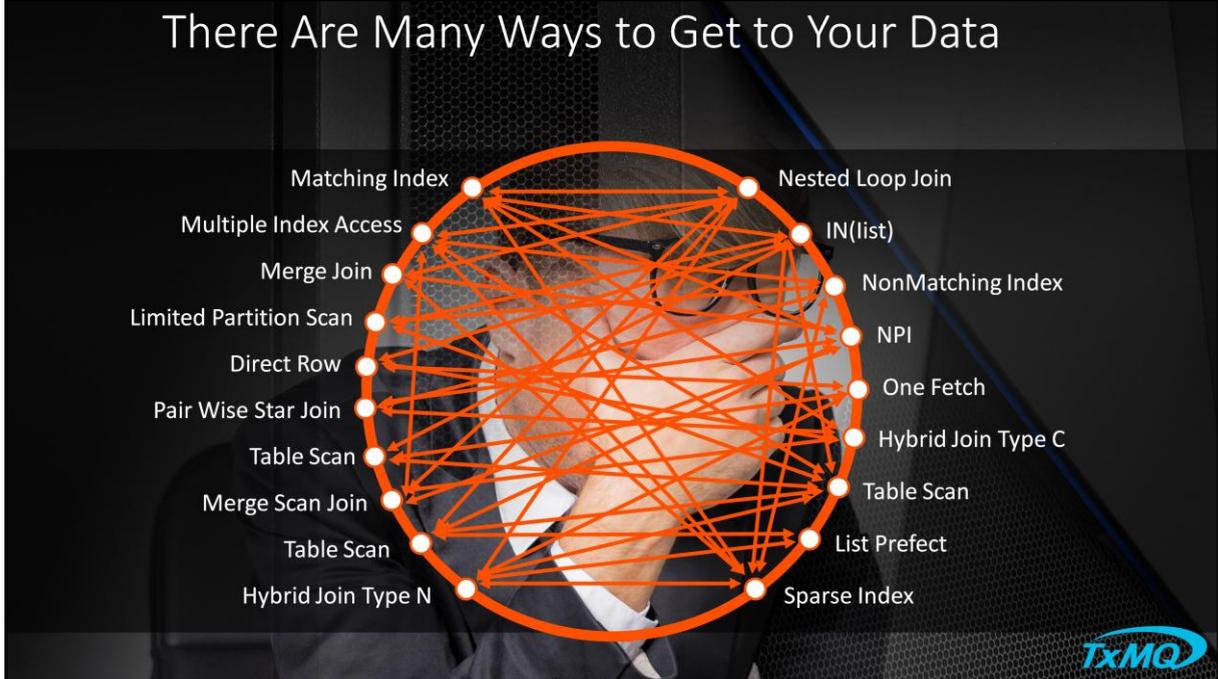
Open ended ranges, '>' etc. are larger 1/1000 vs. closed ended ranges, BETWEEN or LIKE are smaller. 3/10,000. This is the default % of rows the optimizer is estimating will come back from your WHERE clause. Notice that it screams, "Use and Index!"

This may be far from reality.

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



## There Are Many Ways to Get to Your Data



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

*From Terry Purcell's RedPaper:*

[DB2 12 for z Optimizer \(ibm.com\)](http://ibm.com)

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

# The Answer: Personalize Your Optimizer

Technology needed:

- **Learns patterns** from workload data collected in your **unique operating environment**
- Uses derived insight in determining optimal access paths for SQL statements

Built on top of the IBM Machine Learning for z/OS (MLz) stack

Leverages MLz services *without requiring data scientist support* –  
Db2 generates model training data, deploys and monitors  
and retrains models via MLz services

- Db2ZAI product ID: 5698-CGN



## 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.

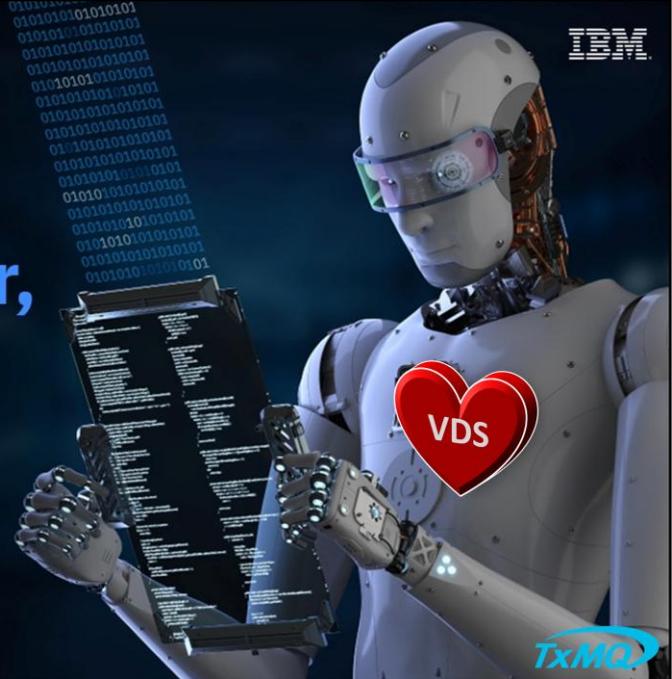
Once trained, the model is ready to be deployed into production, providing insights to the optimizer's access path selection. These insights are in addition to what the optimizer uses today in the selection of the best query path. The information is unique to your environment, and currently unknown to 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

IBM Db2

# AI Makes Db2 Better, Smarter, Faster

IBM Db2 AI for z/OS

#Db2ZAI



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

# Augment the Db2 Z Optimizer with AI/Machine Learning!



1. Correct estimates used for :hv and ?
2. Add OPIMIZE FOR n Rows when # of rows fetched is learned
3. Examine Sort behavior to optimize memory usage
4. Optimize parallelism in packages using history
5. **New Dashboard to set up self tuning and healing**

The Db2 Z Optimization Team Took Action:  
[Db2 AI for z/OS 1.4.0 - IBM Documentation](#)

**IBM Db2® AI for z/OS® aka Db2ZAI**



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




# VDS – Virtual Data Scientist

- Has the data**
  - Catalog statistics
  - Deep execution statistics
  - History
- Knows which algorithm to use**
  - Classification for known patterns
  - Linear Regression for Date/Time sequencing
  - Models for random behavior
- Learns from modeling and scoring**
  - Watches 50 executions
- Provides solutions**
  - A list of ready packages
  - Db2ZAI SQL Performance dashboard
- Cleans up after itself**
  - Keep models current and removes old behavior
  - Db2 12 adds Automatic Statistics Collection

[Automated statistics collection - IBM Documentation](#)



## 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](#)

# Fill in Unknown Values - :hv or ?

Learn from the workload .....

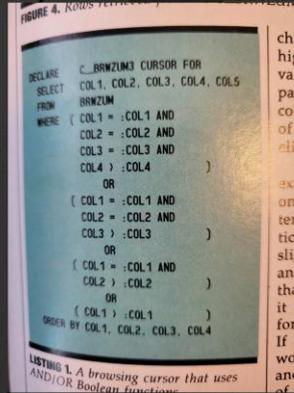
## Customized Filter Factors

For STATIC use REBIND  
For DYNAMIC uses PREPARE

```
WHERE (LASTNAME, FIRSTNAME) > ('SMITH',  
'JOHN')  
WHERE (C1,C2,C3,C4) > (:C1L, :C2L, :C3L, :C4L)
```

## PACKAGE Selection Screen

INCLUDE/EXCLUDE  
Recommended List



**Applies To**  
Any query with :hv or ?



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

Predict # of Rows Qualifying

**Input**  
Track last fetched + SQLCODE  
Repeat 50 times  
Take AVG #

OPTIMIZE FOR **n** ROWS

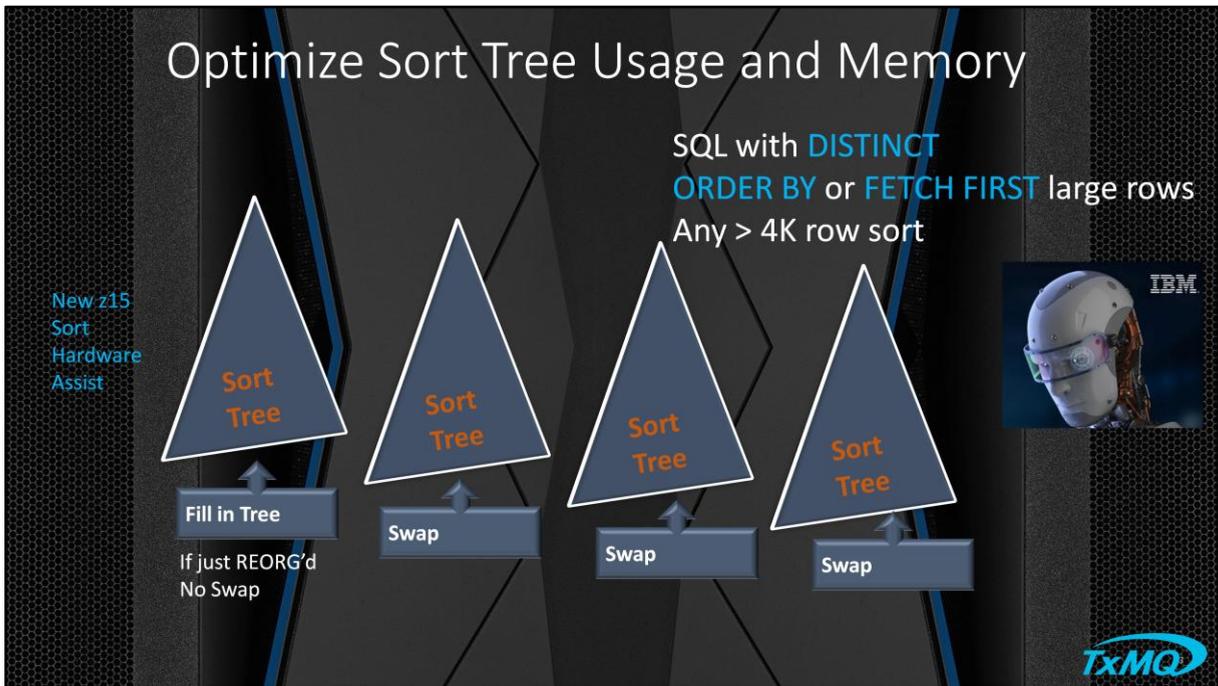
**Applies To**  
Queries qualifying many rows,  
But retrieving only a few

IBM

TxMQ

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.

# Optimize Sort Tree Usage and Memory



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.

Optimize Parallelism in non-OLTP Queries

DEGREE = 'ANY'

DSNZPARM CDDSSRDEF = 'ANY'

**Input**  
Transactions > 120ms  
Never < 10ms

**Output**  
Reduced ELAPSED  
Reduced CPU

IBM

TxMQ

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.

# Db2ZAI New SQL Optimization Dashboard Automation



Auto Top 10  
Collection  
Package



IBM

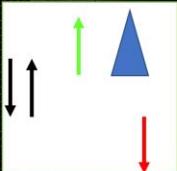


Models



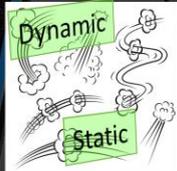
workload

Patterns



Show Results

Training 50X



Take Actions

Learning Complete

Let CACHE refresh + FREE STABILIZED DYNAMIC QUERY

Rebind or Auto phase in\*

Re-run using models

0 - 10%

Improvement in workload

\*Trace 318 + Version 505 is required

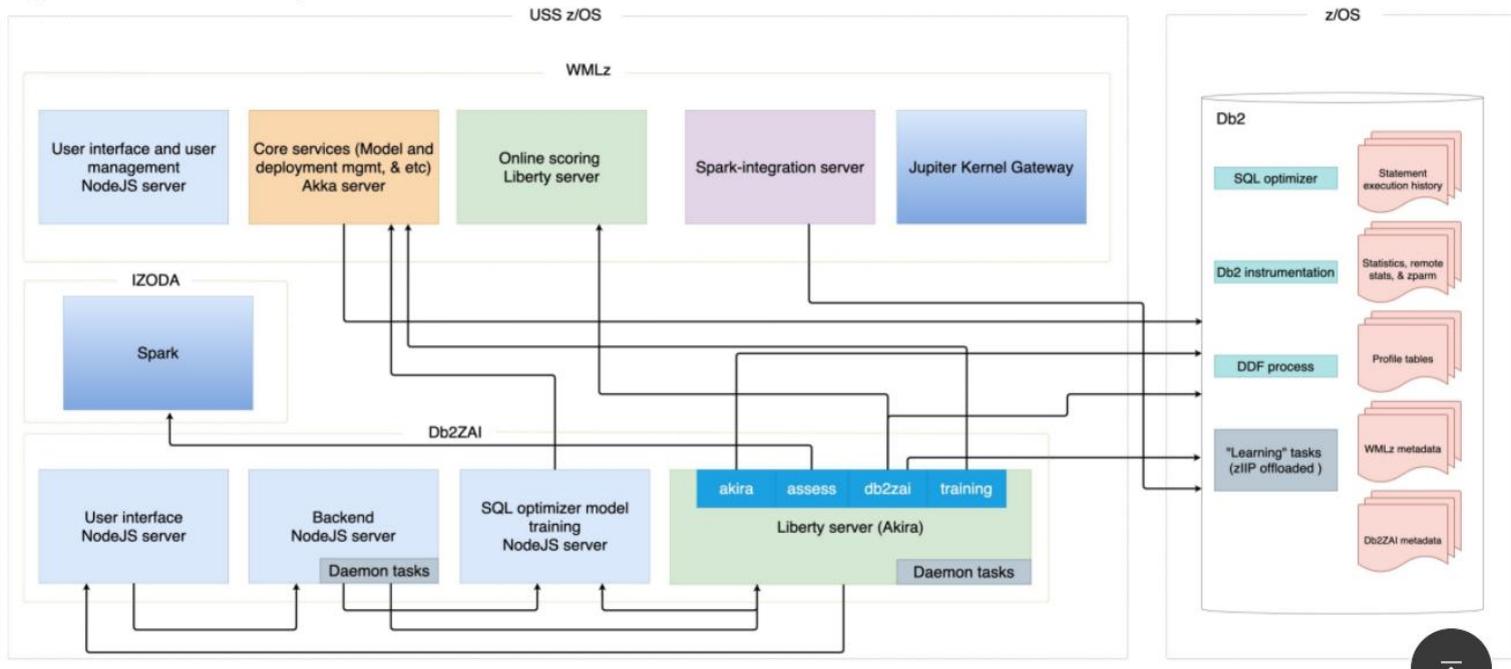
[Overview of IBM Db2 AI for z/OS - IBM Documentation](#)


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



Self Healing and Tuning Workloads

93% improvement over the baseline access path

No human can do this, but a human must set this up

Intervals + Frequency for Top 10 Auto

IBM

TxMQ

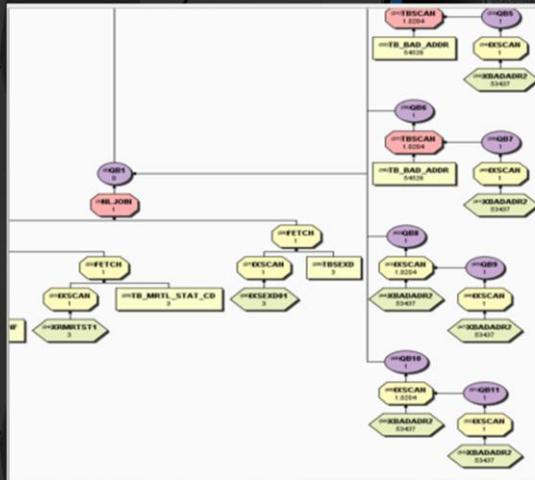
[What's new in IBM Db2 AI for z/OS 1.4.0 - IBM Documentation](#)

### 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.

IBM  
Recommended  
Use Cases

1. Define work periods
2. Training first time
3. Scheduled assessments
4. Daily & on-demand



The larger the graph and the more rows involved,  
the more costly it is.



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

# The 5<sup>th</sup> 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**



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

## Dynamic Queries

Had no Sheriff

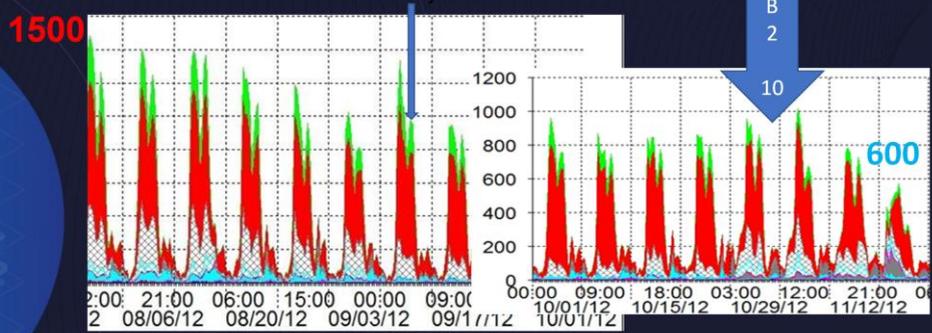


39



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.

# DB2 CPU Reduction



## 900 MIP Reduction



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!

# Db2ZAI: Augment the Db2 Z Optimizer with AI/Machine Learning!



1. Fill in “unknown” values in queries – Use Classification, Linear Regression and Model random behavior to correct estimates
2. Predict number of rows processed and add OPIMIZE FOR n = Optimal Rows
3. Examine Sort behavior to optimize memory usage
4. Optimize Parallelism in non-OLTP packages
5. **Set up workloads in the new dashboard to keep and eye on your dynamic and static**



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

# Best Practices for Query Design From IBM

code mathematics on columns in predicates.

Sort only on the columns that are needed.

No need to ORDERBY BY EMPNO, LASTNAME when you can ORDERBY EMPNO.

Watch out for the LIKE predicate. Begins With logic is indexable. Contains is not indexable. Ends With is not indexable.

Do not code Not Between. Rewrite it as >:HV OR :HV<

Use Fetch First XX Rows whenever possible. Make sure cardinality statistics exist for all columns in all tables.

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.

When joining two tables the execution is faster if the larger table is on the left side of the join.

Code WHERE clauses with columns that have unique or good indexes.

Prioritize WHERE clauses to maximize their effectiveness.

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 only.

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 columns 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.

This is what  
Db2 AI for z/OS  
Cannot Do!

[IBM Data Virtualization Manager for z/OS | IBM Redbooks](https://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg248514.html?Open)

TxM

## Best practices for query design from IBM Efficient SQL list: 8.8.1 SQL best practices

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 only.
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 columns 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.

<https://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg248514.html?Open>

# 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!

# Tuning Techniques to Apply When Necessary

## Learn Traditional Tuning Techniques

OPTIMIZE FOR n ROWS

No Ops

Index & MQT Design



## Experiment with Extreme Tuning Techniques

DISTINCT Table Expressions

Odd/old Techniques

Manual Query Rewrite

44



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

## OPTIMIZE FOR n ROWS FETCH FIRST n ROWS

- Both clauses influence the Optimizer
  - To encourage index access and nested loop join
  - To discourage list prefetch, sequential prefetch, and access paths with Rid processing
  - Use FETCH n = total rows required for set
  - Use OPTIMIZE n = number of rows to send across network for distributed applications
  - Works at the statement level

45



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

# Fetch First Example

```

SELECT S.QTY_SOLD
, S.ITEM_NO
, S.ITEM_NAME
FROM SALE S
WHERE S.ITEM_NO > :hv
ORDER BY ITEM_NO

```

- Optimizer choose List Prefetch Index Access + sort for ORDER BY for 50,000 rows
- All qualifying rows processed (materialized) before first row returned = .81 sec (less than 1 sec)
- <.1sec response time required
- Adaptive index** is a Db2 12 enhancement to multi-index and single index list prefetch-based

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

TxMQ

[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.

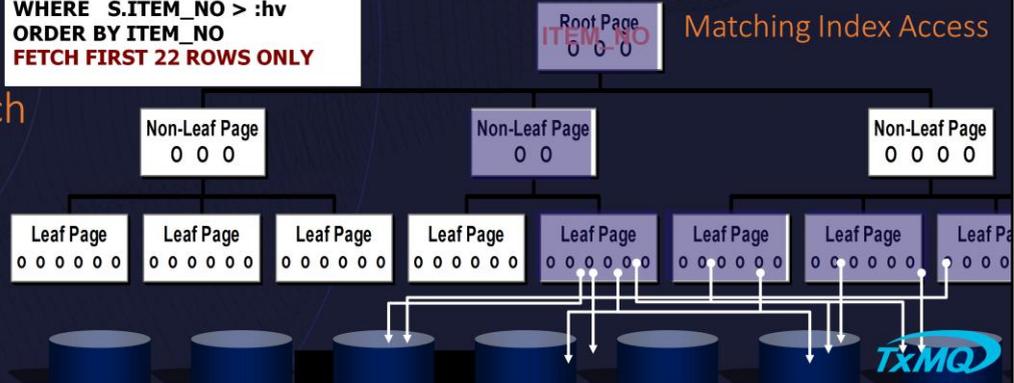
## Limited Fetch in Action

### Query #1 Tuned

```
SELECT S.QTY_SOLD, S.ITEM_NO  
      , S.ITEM_NAME  
FROM   SALES  
WHERE  S.ITEM_NO > :hv  
ORDER BY ITEM_NO  
FETCH FIRST 22 ROWS ONLY
```

Turns off  
List Prefetch

- Optimizer now chooses Matching Index Access (first probe .004 sec)
- No materialization
- Cursor closed after 22 items displayed (22 \* .0008 repetitive access)
- $.004 + .017 = .021$  sec



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

## No Operation (No Op)



- +0, CONCAT '' also -0, \*1, /1
  - Place no op next to predicate
  - Use as many as needed
  - Discourages index access, however, preserves Stage 1
  - Can Alter table join sequence
  - Can fine tune a given access path
  - Can request a table scan
  - Works at the predicate level

Does not Benefit  
Db2 on Linux,  
UNIX or  
Windows

48



Rarely used but very powerful so be careful.

## No Op Example CONCAT ''

SALES\_ID.MNGR.REGION Index

MNGR Index

REGION Index

```
SELECT S.QTY_SOLD
       , S.ITEM_NO
       , S.ITEM_NAME
FROM   SALE S
WHERE  S.SALES_ID > 44
       AND S.MNGR = :hv-mngr
       AND S.REGION BETWEEN
           :hvlo AND :hvhi
ORDER BY S.REGION
```

```
.....
FROM   SALE S
WHERE  S.SALES_ID > 44
       AND S.MNGR = :hv-mngr
       AND S.REGION BETWEEN
           :hvlo AND :hvhi CONCAT ''
ORDER BY R.REGION
```

- Optimizer chooses Multiple Index Access
- The table contains 100,000 rows and there are only 6 regions
- Region range qualifies 2/3 of table
- <.1sec response time required
- No Op allows Multiple Index Access to continue on first 2 indexes
- Two Matching index accesses, two small Rid sorts, & Rid intersection

Blocks Indexable

Stage 1 access but still allows

Stage 1 Leaf Page Screening



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 than the other query blocks), block Matching Access (only from the Optimizer) but still allow all the filtering on other indexes Leafpages.

## No Op Example - Scan

SALES\_ID.MNGR.REGION Index

MNGR Index

REGION Index

```
SELECT S.QTY_SOLD
       , S.ITEM_NO
       , S.ITEM_NAME
FROM   SALE S
WHERE  S.SALES_ID > 44 +0
       AND S.MNGR = :hv-mngr CONCAT ``
       AND S.REGION BETWEEN
           :hvlo AND :hvhi CONCAT ``
ORDER BY S.REGION
FOR FETCH ONLY
WITH UR
```

- If you know the predicates do very little filtering, force a table scan
- Use a No Op on every predicate
- This forces a table scan
- FOR FETCH ONLY encourages parallelism
- WITH UR for read only tables to reduce CPU

Should this be Documented?

[Query performance in the Db2 12 initial release - IBM Documentation](#)



### [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](#)

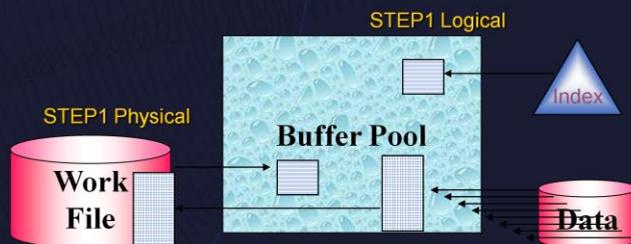
The merge capabilities of user-defined table functions are enhanced in Db2 12 to be similar to the capabilities of views.

#### •[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.

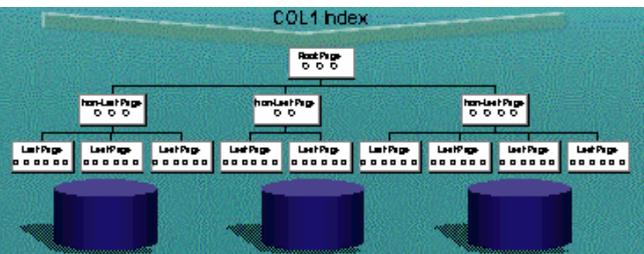
# DISTINCT Table Expressions

- Table expressions with DISTINCT
  - FROM (SELECT DISTINCT COL1 FROM T1 .....) AS STEP1 JOIN T2 ON ... JOIN T3 ON ....
  - Used for forcing creation of logical set of data
    - No physical materialization if an index satisfies DISTINCT
  - Can encourage sequential detection
  - Can encourage a Merge Scan join

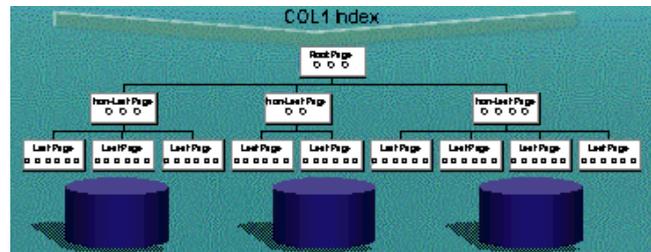


Discovered this technique when a customer called and asked how they could get their batch Merge Scan Join back into production. A migration changed it to a Nested Look Join.

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



2



## Sequenced Workfile



## Sequenced Workfile



4



Result

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

## DISTINCT Table Expressions Example

52

- SELECT Columns  
FROM **TABX**, **TABY**,  
(SELECT DISTINCT COL1, COL2 .....  
FROM **BIG\_TABLE Z**  
WHERE local conditions) AS BIGZ  
WHERE join conditions
- Optimizer is forced to analyze the table expression prior to joining TABX & TABY



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!

## Typical Join Problem

```
SELECT COL1, COL2 .....  
FROM ADDR, NAME, TAB3, TAB4, TAB5, TAB6, TAB7 WHERE  
join conditions  
AND TAB6.CODE = :hv
```

Cardinality 1

- Result is only 1,000 rows
- ADDR and NAME first two tables in join
- Index scan on TAB6 table
  - Not good because zero filter



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

## Tuning Technique

```
SELECT COL1, COL2 .....  
FROM ADDR, NAME,  
  (SELECT DISTINCT columns  
   FROM TAB3, TAB4, TAB5, TAB6, TAB7  
   WHERE join conditions  
    AND (TAB6.CODE = :hv OR 0 = 1))  
AS TEMP  
WHERE join conditions
```

Keeps large tables  
joined last



This Photo by Unknown Author  
is licensed under [CC BY-SA-NC](#)

Gets rid of Index Scan



BIG tables held to be last involved in the Join

## Using Common Table Expressions

```
WITH TEMP AS (SELECT DISTINCT columns
FROM TAB3, TAB4, TAB5, TAB6, TAB7
WHERE 4 join conditions
AND (TAB6.CODE = :hv OR 0 = 1)) AS TEMP
SELECT COL1, COL2 .....
FROM ADDR, NAME, TEMP
WHERE join 2 conditions
```



For the newbies.

# Put a Query on a Diet

For Extreme Cases  
(used on all platforms)

56

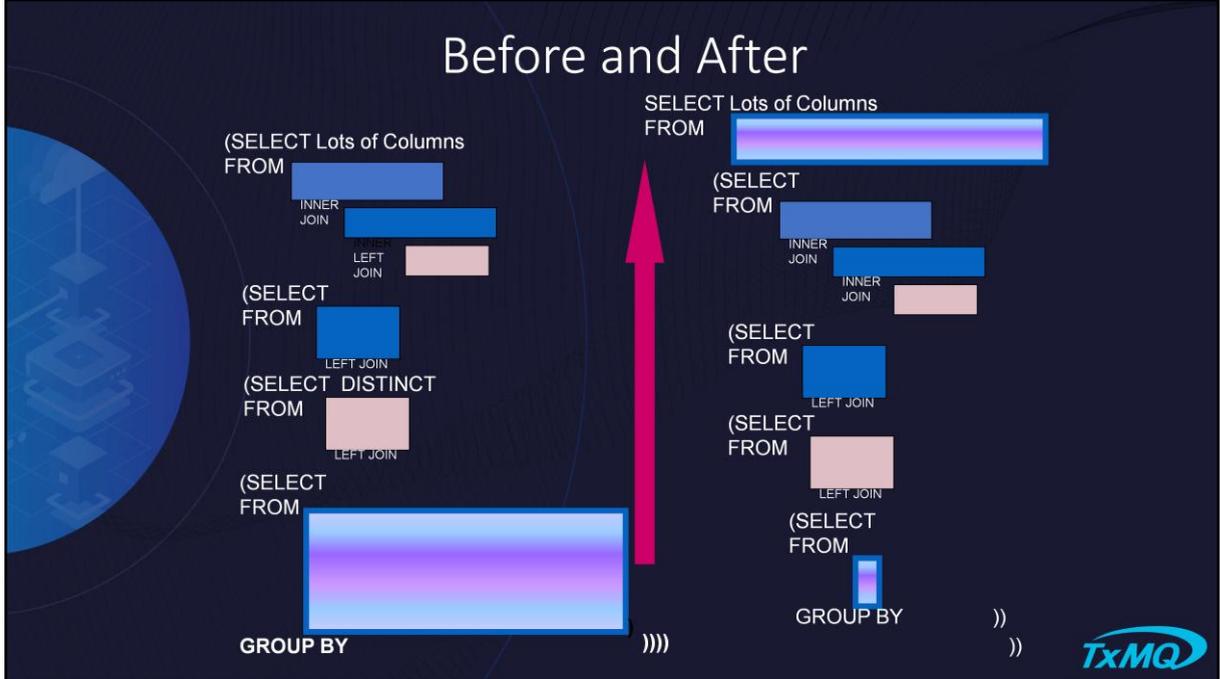


## A Typical Data Warehouse Query

- Initial cost of **16 million timerons**
  - **WOULD NOT FINISH!**
- Included a DISTINCT table expression and GROUP BY
- Initial join involved all columns and all rows
- The very wide and very deep set was dragged through many more query steps



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.

## Tuning Technique

- Identify and pre-qualify the core set of data and only select the keys early on
- Once all the steps are complete, go back and get the remaining columns
- Referred to as “**Group By Push Down**” and “**put your query on a diet**”
  - Keeping it thin through the DB2 engine
- Brought cost down to **270,000 timerons**
  - Query now finishes in **4 minutes!**

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.

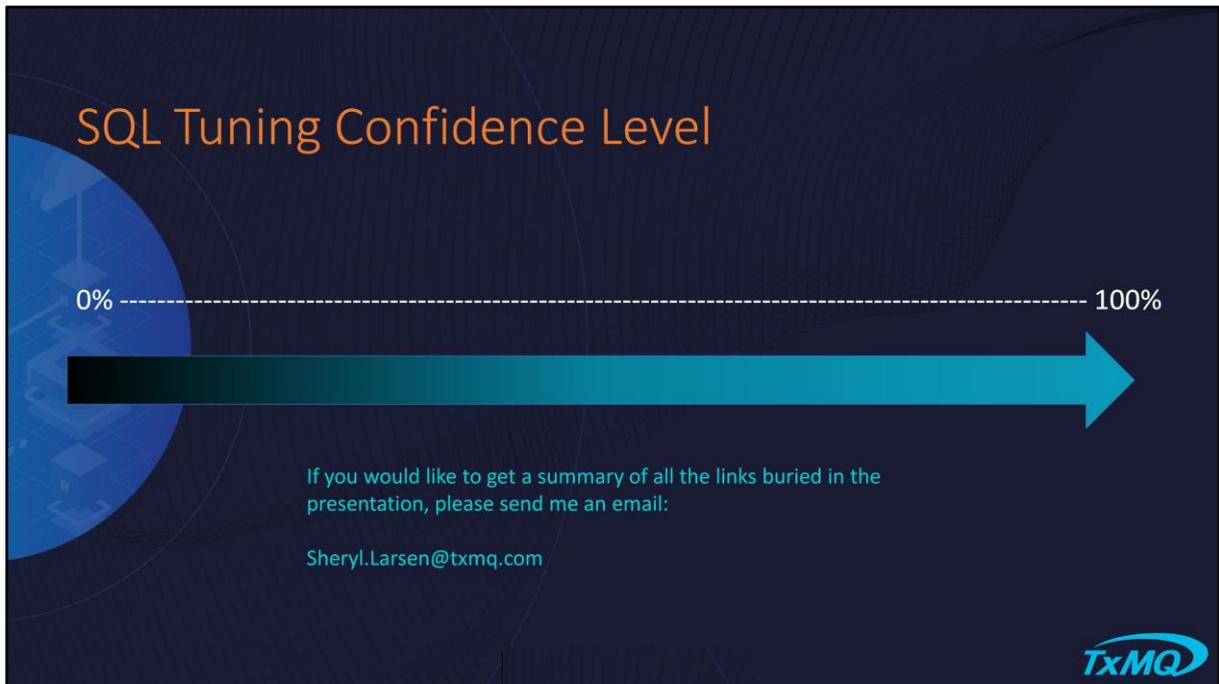
## Three Main Stories

- ✓ What happens inside the Db2 engine when I make a selection on my cell phone? **The SQL optimizes and executes very efficiently but could use some help in 5 use cases.**
- ✓ What can the Db2 AI for z/OS 1.4.0 do to make my workload go faster? **Humans setting up the new SQL Optimization dashboard makes 0-10% of queries go faster**
- ✓ What doesn't the Db2ZAI do that humans will still have to do? **There are 20 SQL performance rules plus creative tuning techniques that STILL WORK!**



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](mailto:sheryl.larsen@txmq.com)  
[LinkedIn](#) • [Twitter](#) • [Facebook](#)

**TxMQ, Inc.**  
*Imagine, Transform, Engage*  
[TxMQ.com](http://TxMQ.com) • [TxMQStaffing.com](http://TxMQStaffing.com) TxMQ is a Premier IBM Business Partner

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

Speaker: Sheryl Larsen

Email Address: Sheryl.Larsen@txmq.com

[TxMQ.com](http://TxMQ.com) • [TxMQStaffing.com](http://TxMQStaffing.com)  
*Imagine, Transform, Engage*

TxMQ is a Premier IBM Business Partner  
[LinkedIn](#) • [Twitter](#) • [Facebook](#)  
• 716-636-0070  
• [info@txmq.com](mailto:info@txmq.com)



[Db2 12 for z/OS Catalog Tables - BMC Blogs - BMC Software](#)

A new interactive Catalog application from BMC:

[DB2 12 for z Optimizer \(ibm.com\)](#)

[The Big Old Mainframe: DB2 Bind process](#)

[Db2 AI for z/OS 1.4.0 - IBM Documentation](#)

[Overview of IBM Db2 AI for z/OS - IBM Documentation](#)

[DSN STATEMENT CACHE TABLE - IBM Documentation](#)

[Db2 12 for z/OS: SQL Reference \(ibm.com\)](#)

[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](#)