IDUG VIRTUAL

2021 Australasia Db2 Tech Conference

IDUG

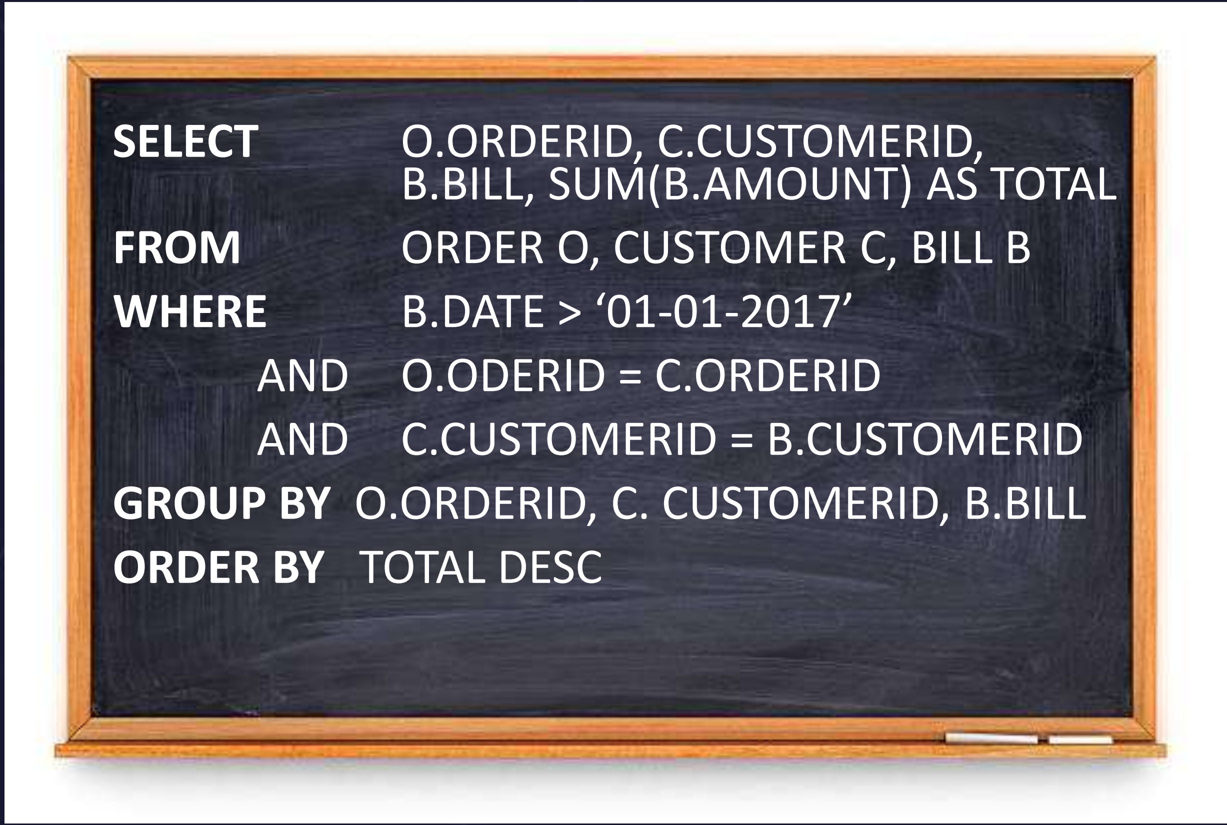
SQL Tuning that Still Works

Sheryl Larsen, BMC

Even with the Intelligence of Db2zAI

z/OS

First SQL Class Db2 V1



1984

- No internet
- No iPhone
- No laptop
- No ear buds
- No email
- No …….

- lwaux (Load Word Algebraic with Update Indexed) instruction
- lwax (Load Word Algebraic Indexed) instruction
- lwbrx or lbrx (Load Word Byte-Reverse Indexed) instruction
- lwz or l (Load Word and Zero) instruction
- lwzu or lu (Load Word with Zero Update) instruction
- lwzux or lux (Load Word and Zero with Update Indexed) instruction
- lwzx or lx (Load Word and Zero

Fast Forward

35 YEARS

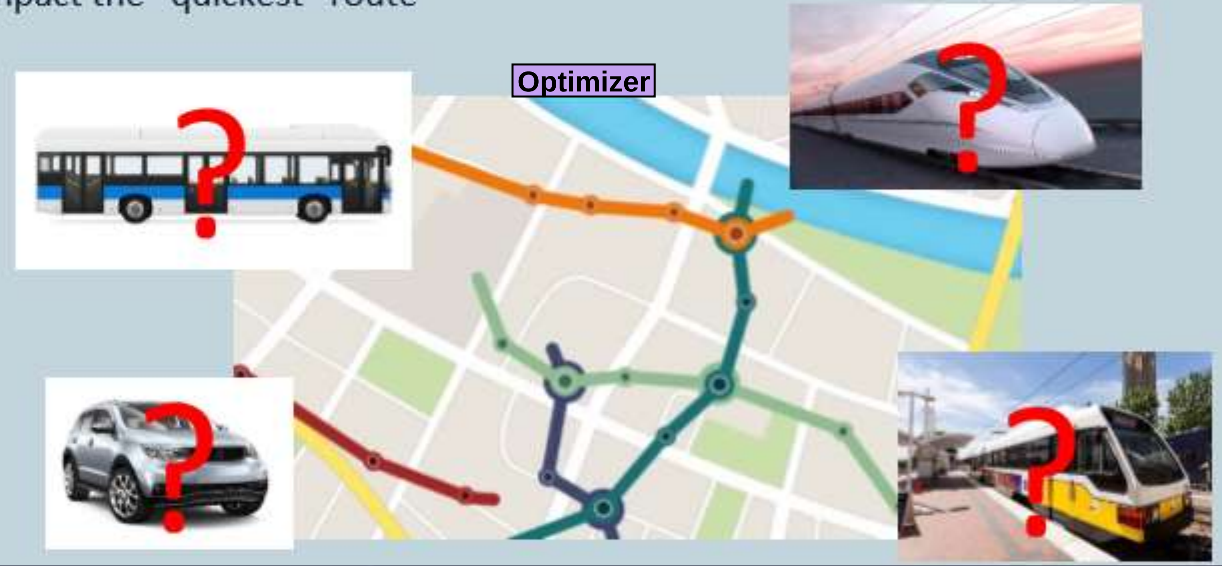
1984
1989
2003
2010
2015
2019

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, …

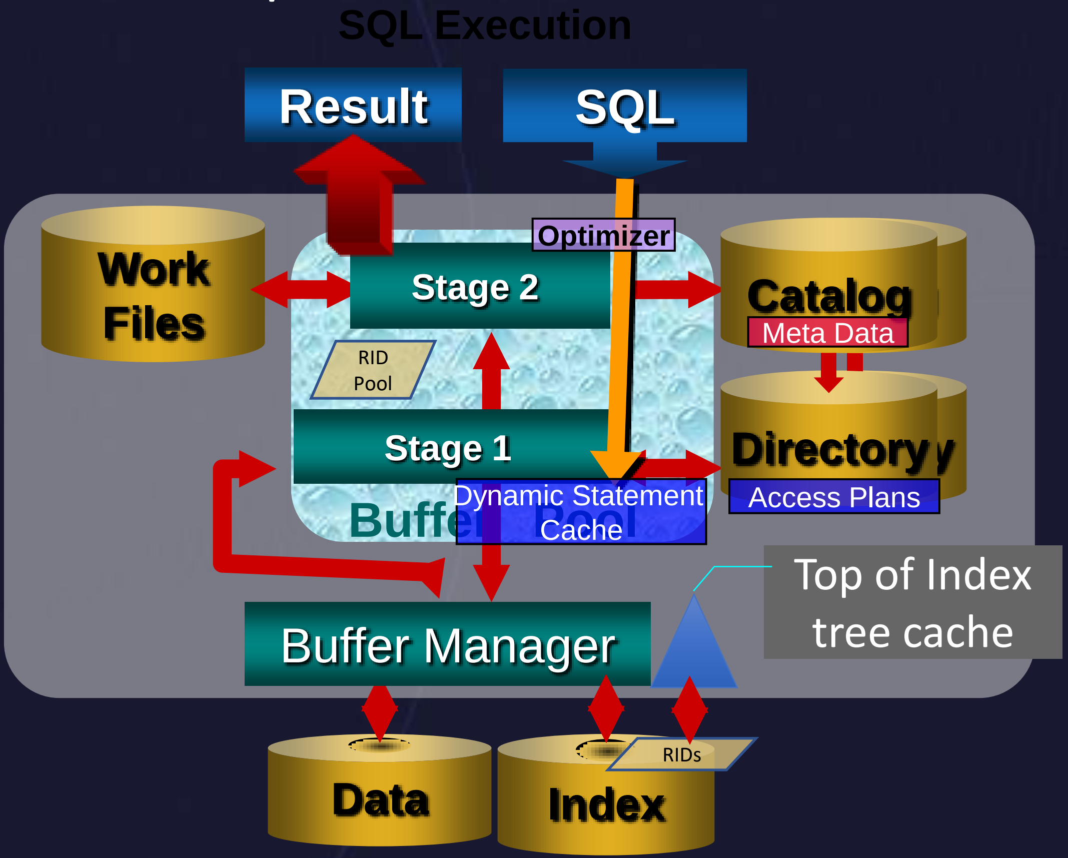
Cost based Optimizer figures out how to get the data

- The optimizer is responsible for
  - Choosing the most efficient method of accessing the data for a given SQL statement

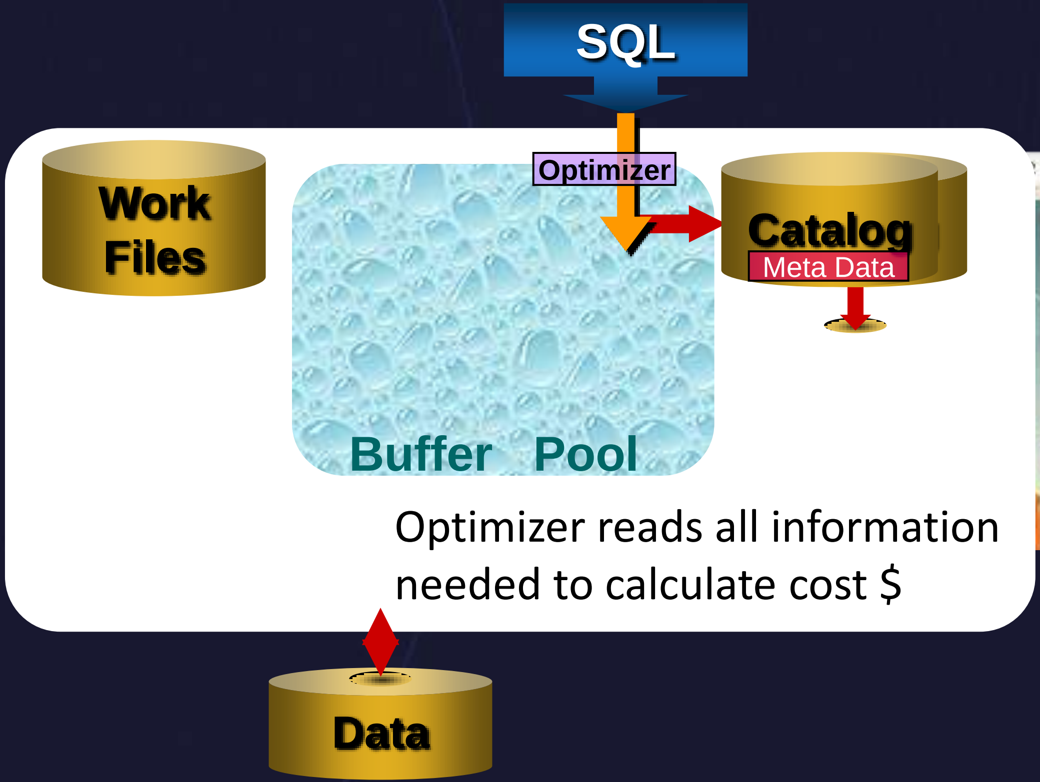
- Think of your transportation choices
  - Start/end location, time of day, construction, traffic, available options/routes
    - All can impact the "quickest" route



DB2 Engine Components

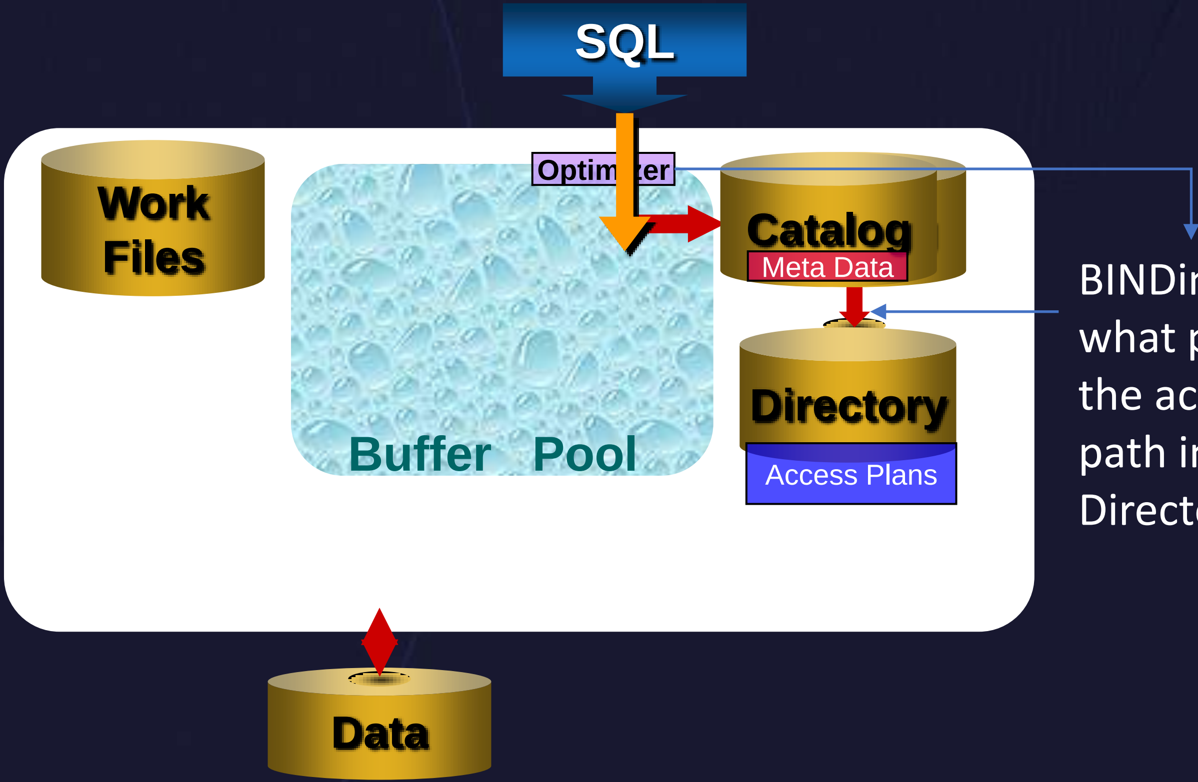


Meta Data – Everything known about each object



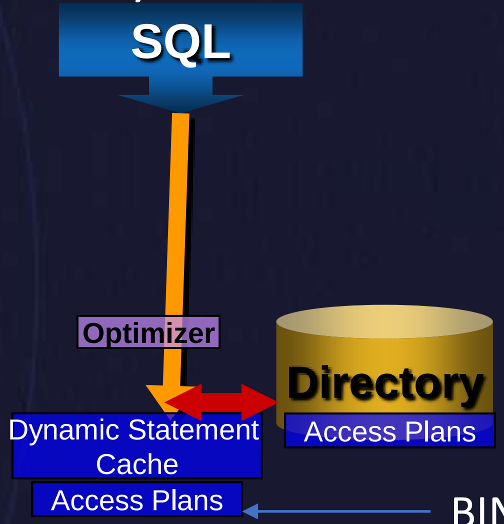
Stored in the Catalog

Static SQL Access Plans Stored in Directory



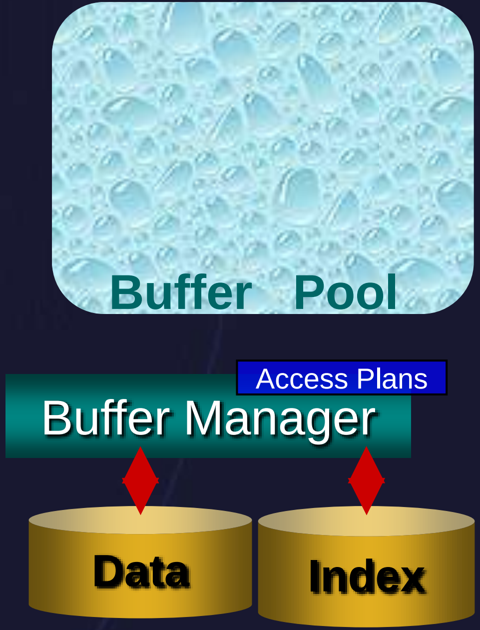
BINDing is what puts the access path in the Directory

Dynamic SQL is stored in the Dynamic Statement Cache

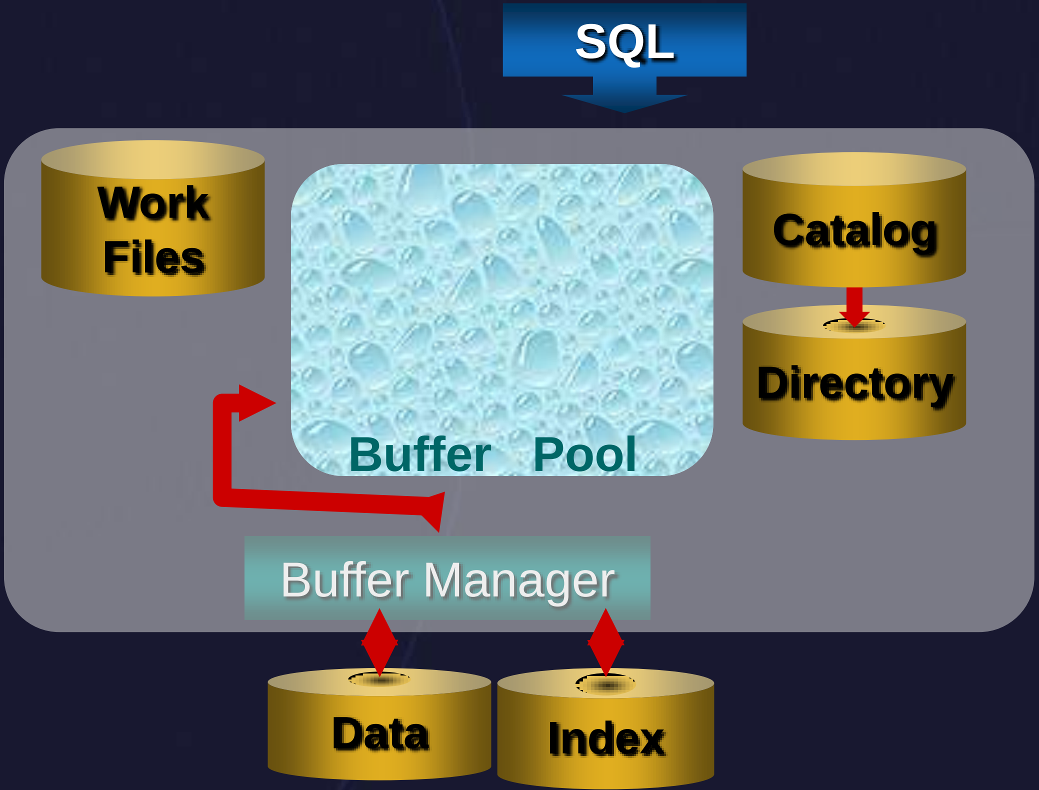


BIND done automatically at run time

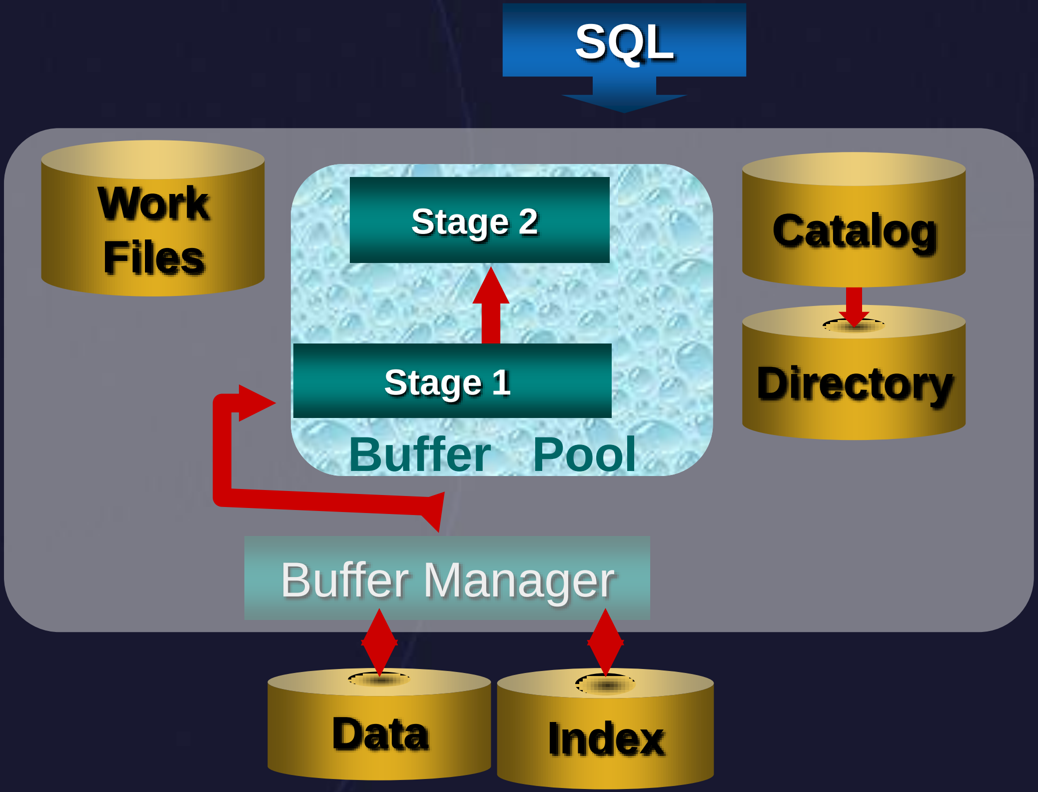
Execution time is when the Access Plan is given to the Buffer Manager



Buffer Pool stores data in memory



Stage 1 & 2 filter the data



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.

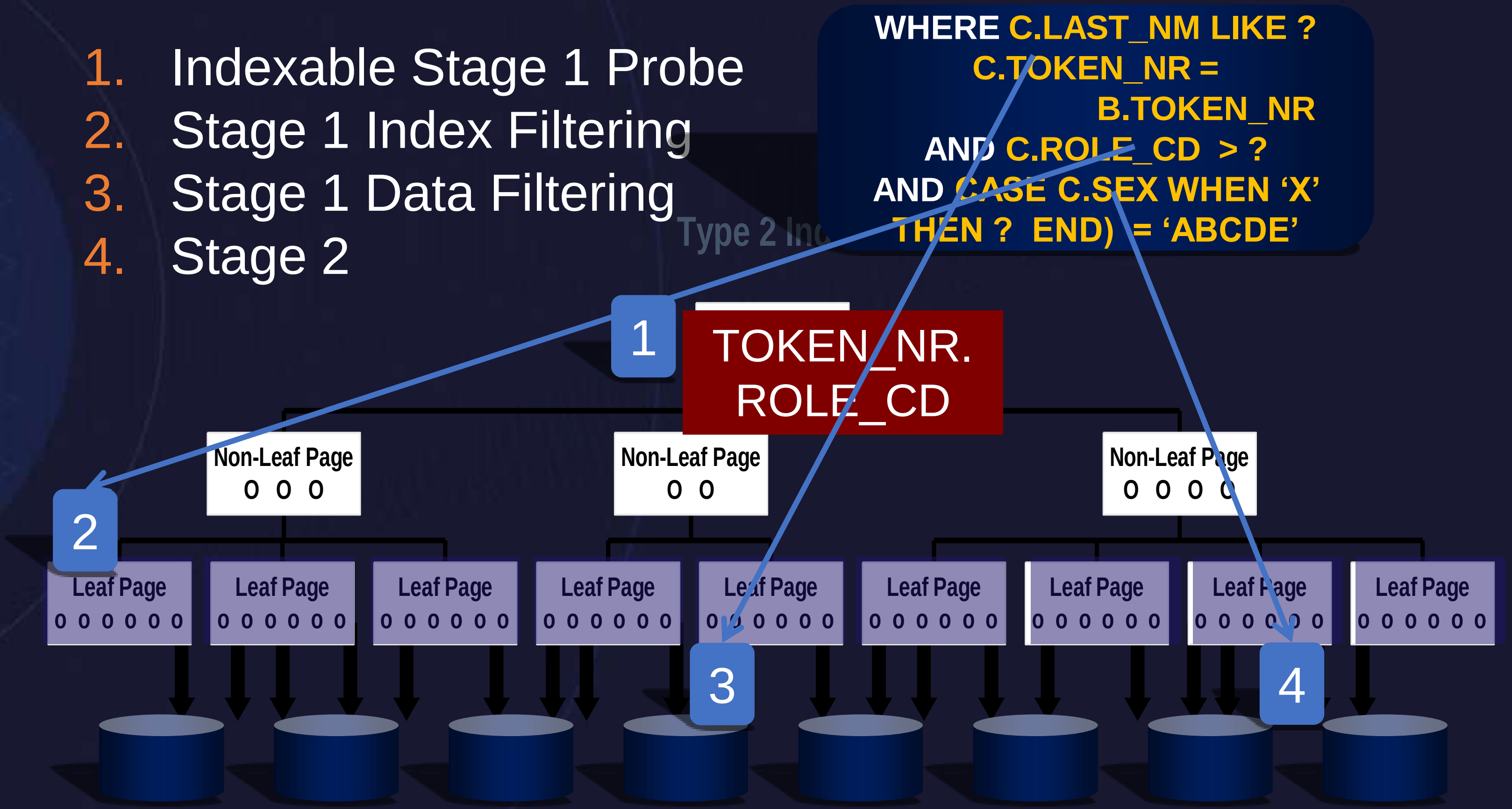
2. **Index Screening** = The Stage 1 predicate is a candidate for filtering on the index leaf pages. This is the second point of filtering in DB2. If partitioned filters limiting partitions are also applied

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.

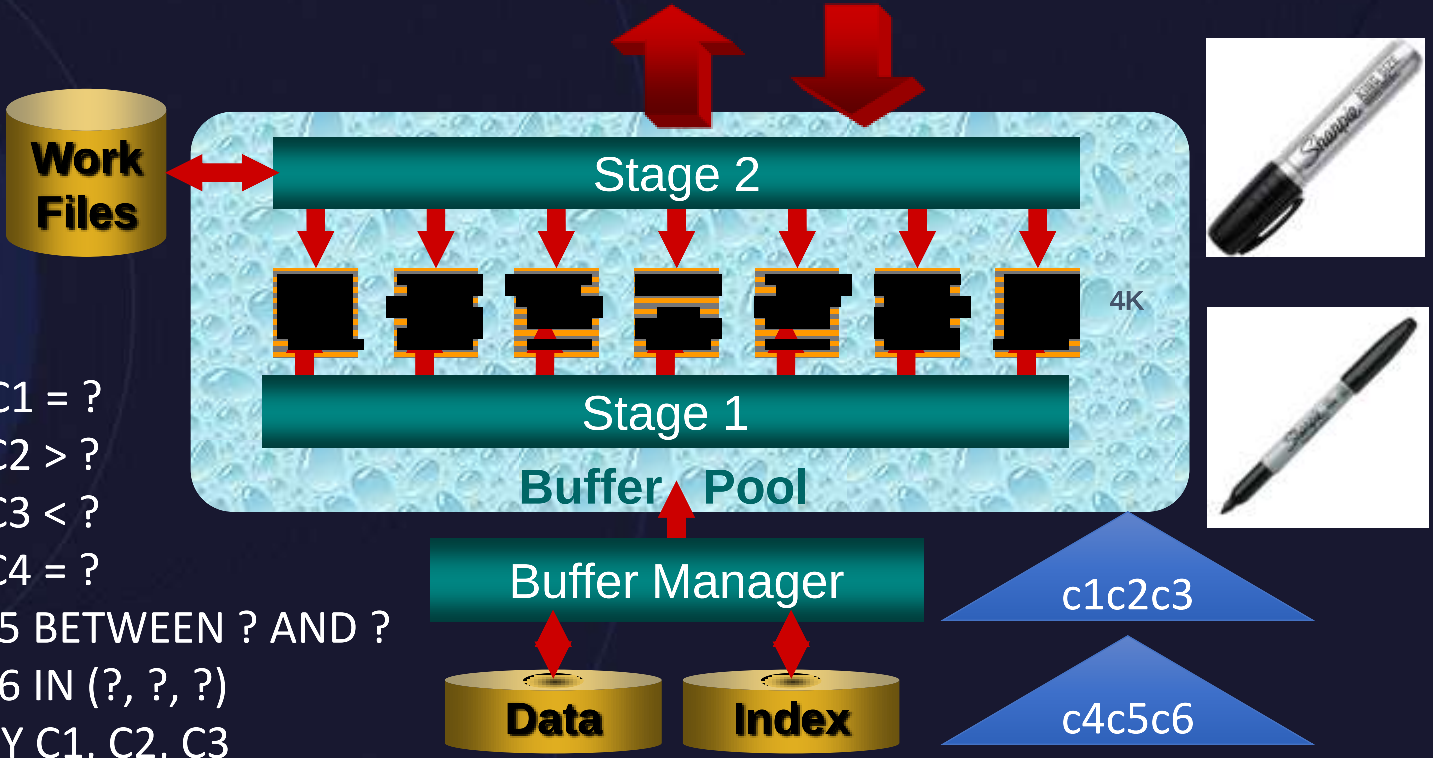
Four Points of Filtering

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



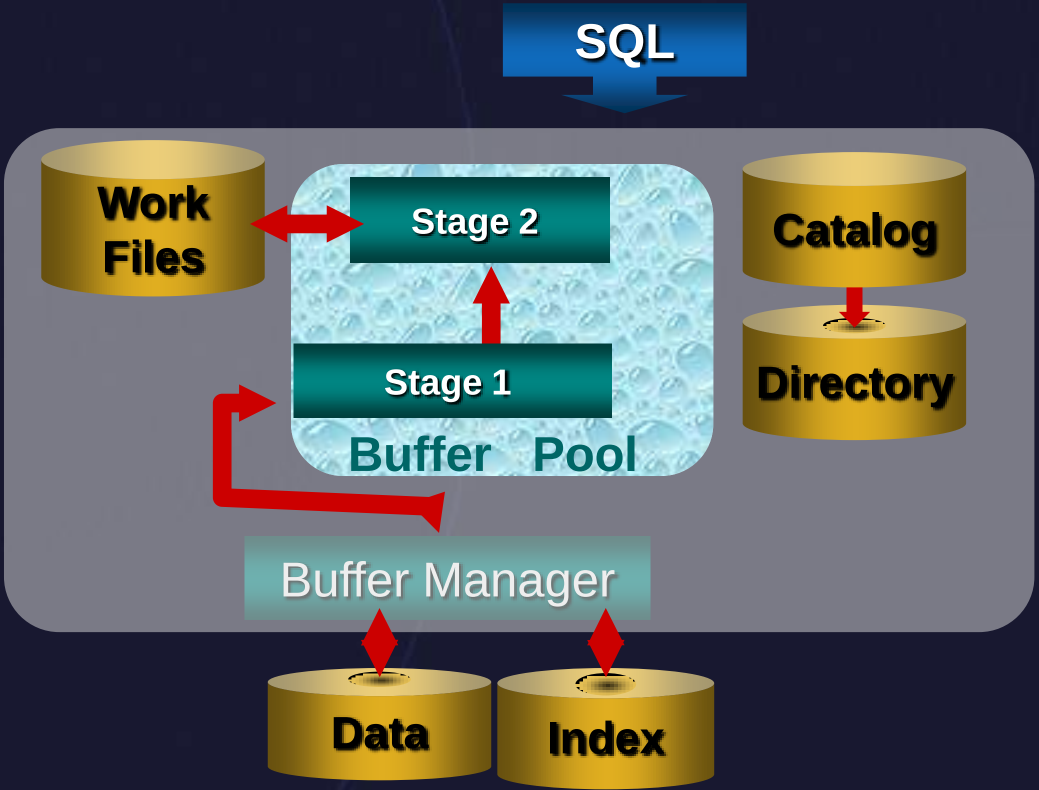
Filtering – z/OS

WHERE C1 = ?
AND C2 > ?
AND C3 < ?
AND C4 = ?
AND C5 BETWEEN ? AND ?
AND C6 IN (?, ?, ?)
ORDER BY C1, C2, C3



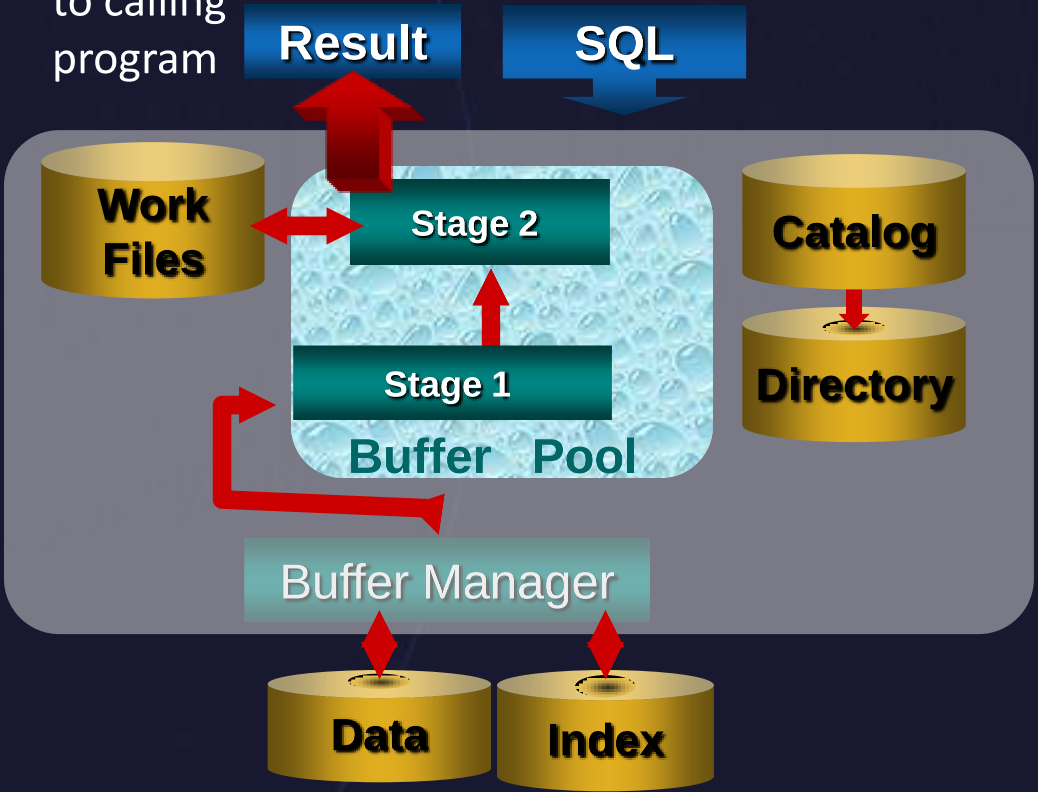
If a Sort is needed for ORDER BY/GROUP BY

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

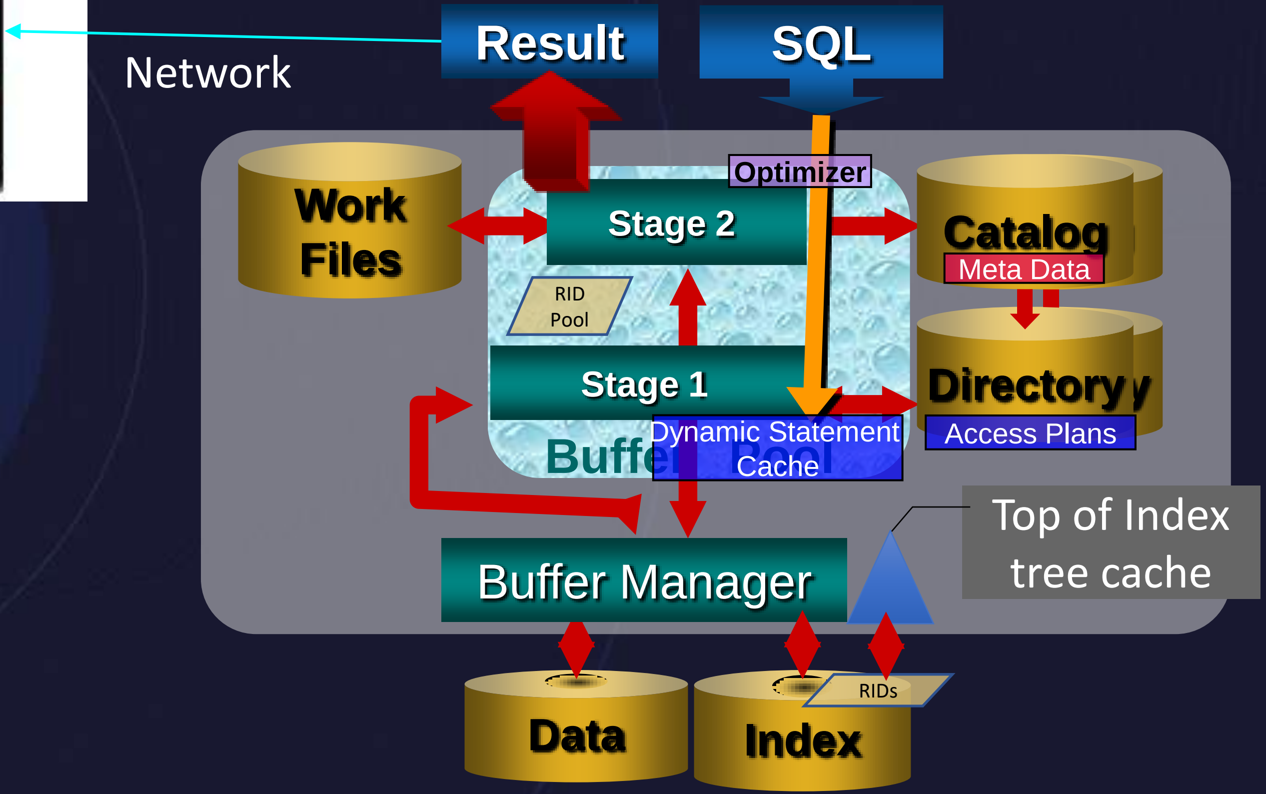


The Result is brought back in memory

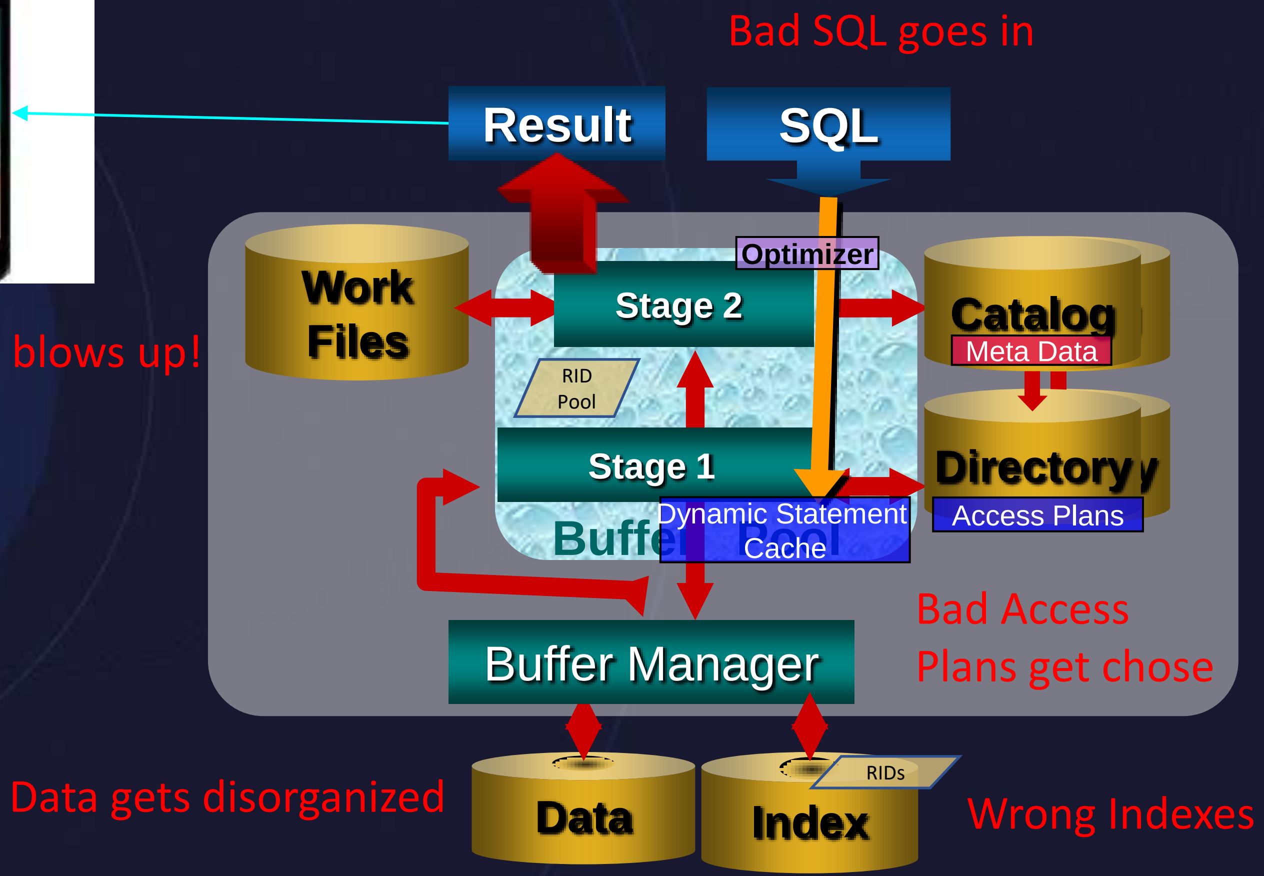
Data is sent to calling program



Source may be Remote



What Could Go Wrong?



Agenda

- Review of what 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
- What other query attributes are red flags to optimal performance?

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'?**

The Trillions of Optimizer Cost-based Results

Good for Everybody

Great for a Few

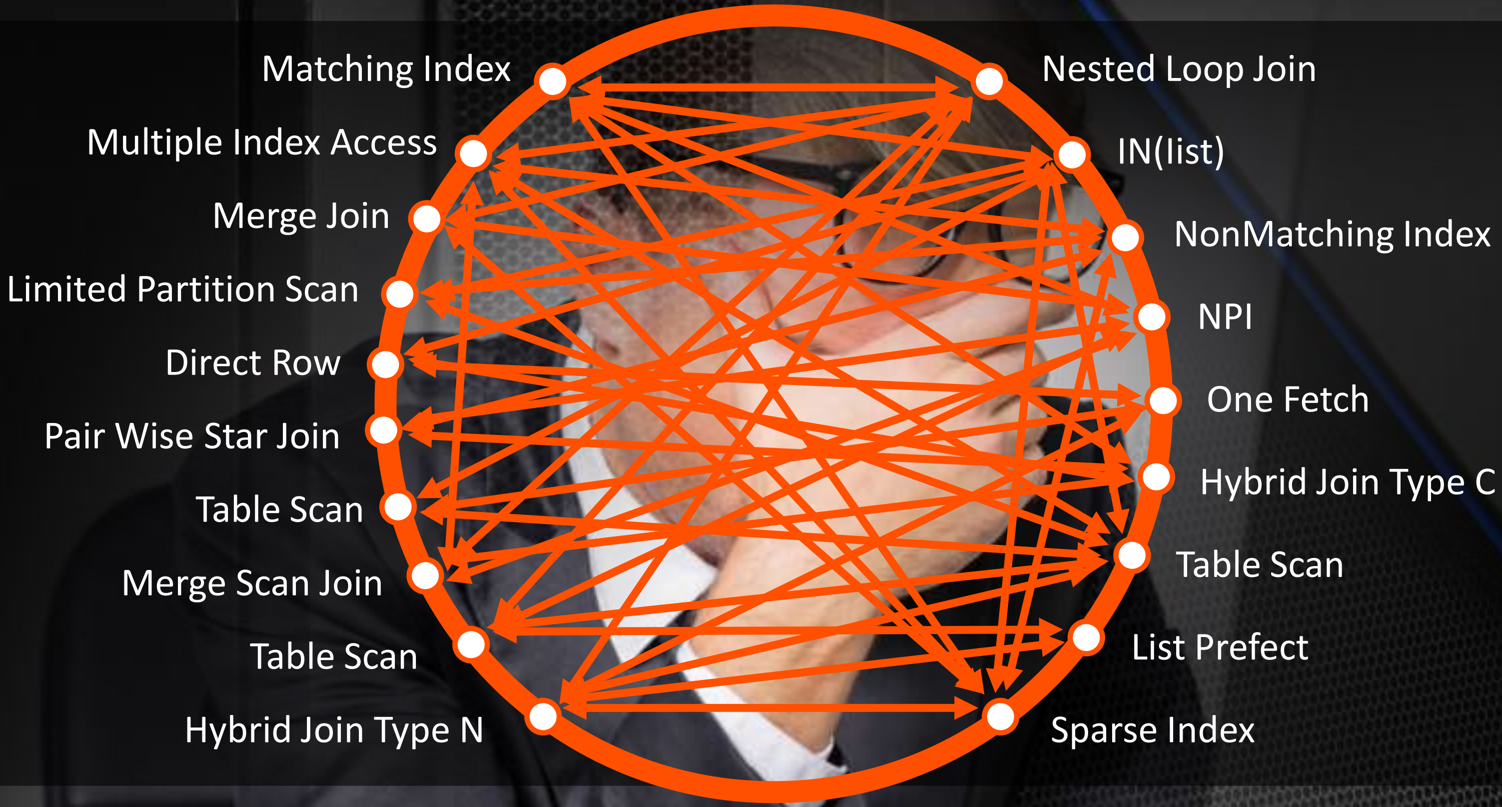
Default Statistics

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?

There Are Many Ways to Get to Your Data



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

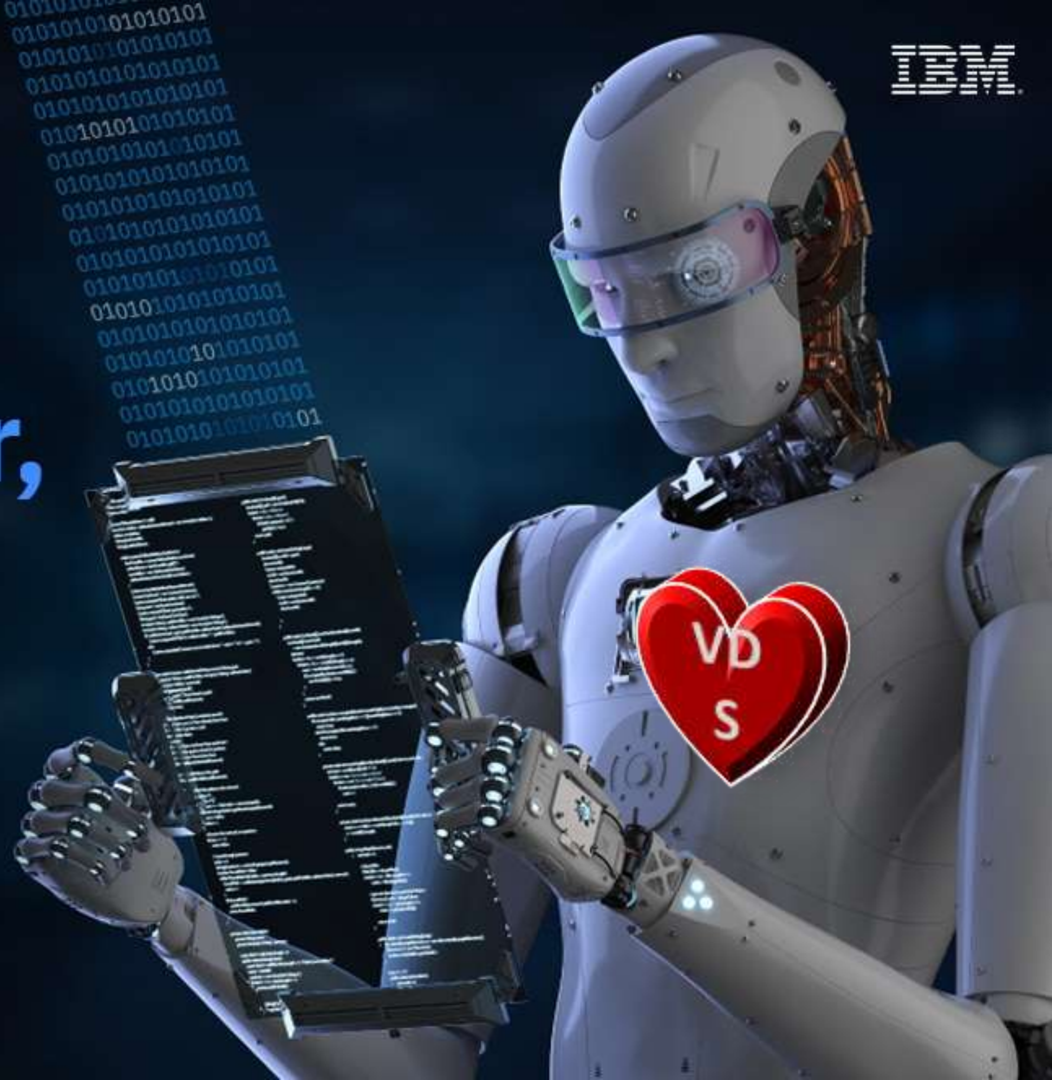
Built on top of the IBM Machine Learning for z/OS (MLz) stack

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

IBM **Db2**

AI Makes Db2
Better, Smarter, Faster



IBM Db2 AI for z/OS

#Db2ZAI

IDEA: 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

The Db2 Z Optimization Team Took Action:
https://www.ibm.com/support/knowledgecenter/en/SSGKMA_1.2.0.2/src/ai/ai_home.html

IBM Db2® AI for z/OS® aka Db2zAI

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 100 executions**

Provides solutions

- **A list of ready packages**
- **Db2ZAI SQL Performance dashboard**

Cleans up after itself

- **Keep models current and removes old behavior**

[redacted]

Fill in Unknown Values - :hv or ?

Customized Filter Factors

For STATIC use REBIND

For DYNAMIC uses PREPARE

PACKAGE Selection Screen

INCLUDE/EXCLUDE

Recommended List

Learn from the workload .....

```
DECLARE   C_BRWZUM3 CURSOR FOR
SELECT   COL1, COL2, COL3, COL4,
FROM   BRWZUM
WHERE   ((COL1 = :COL1-LAST
    AND   COL2 = :COL2-LAST
    AND   COL3 = :COL3-LAST
    AND   COL4 > :COL4-LAST )
 OR
        (COL1 = :COL1-LAST
    AND   COL2 = :COL2-LAST
    AND   COL3 > :COL3-LAST )
    OR
        (COL1 = :COL1-LAST
    AND   COL2 > :COL2-LAST )
    OR
        (COL1 > :COL1-LAST))
 ORDER BY COL1,COL2,COL3,COL4
```

Applies To

Any query with :hv or ?

Predict # of Rows Qualifying

OPTIMIZE FOR n ROWS

Input

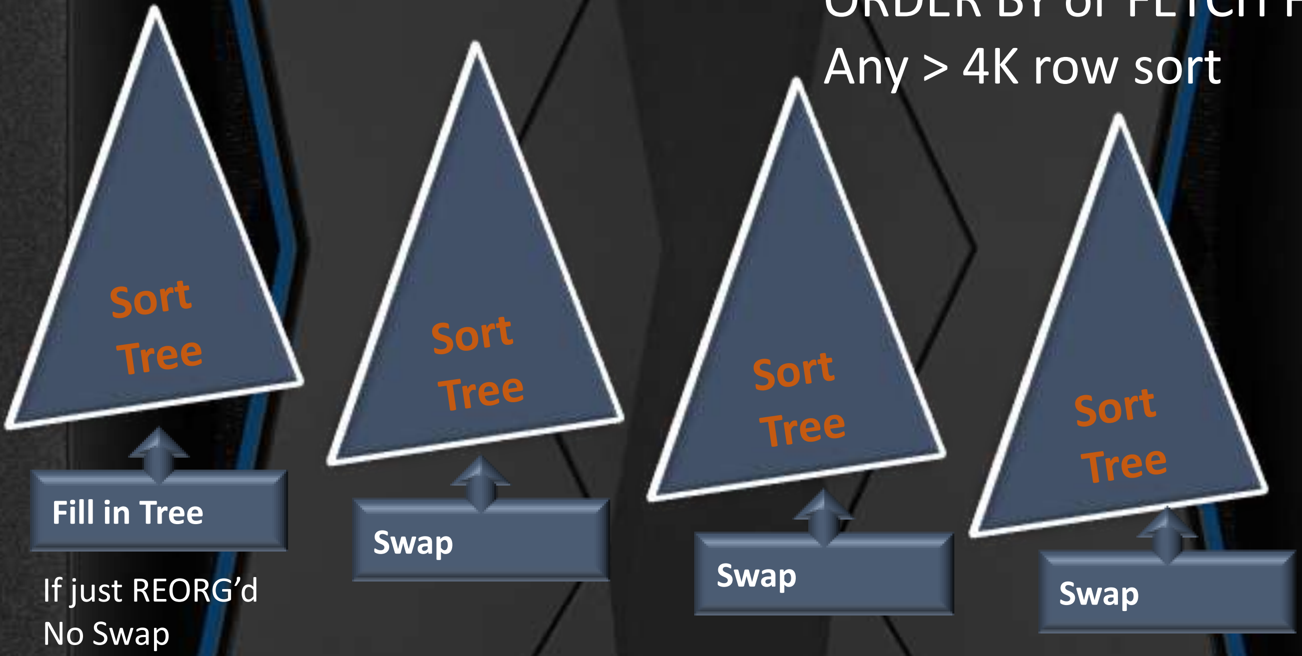
Track last fetched + SQLCODE
Repeat 100 times
Take AVG #

Applies To

Queries qualifying many rows,
But retrieving only a few

Optimize Sort Tree Usage and Memory

SQL with DISTINCT
ORDER BY or FETCH FIRST large rows
Any > 4K row sort



Optimize Parallelism in non-OLTP Queries

DEGREE = ‘ANY’

DSNZPARM CDDSSRDEF = ‘ANY’

Input

Transactions > 120ms

Never < 10ms

Output

Reduced ELAPSED

Reduced CPU

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

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

SQL Tuning Examples

```
WHERE S.SALES_ID > 44
    AND  S.MNGR = :hv-mngr
    AND  S.REGION BETWEEN
           :hvlo AND :hvhi CONCAT ` `
```

No Operation

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

Limited Fetch

All the Possible Access Paths

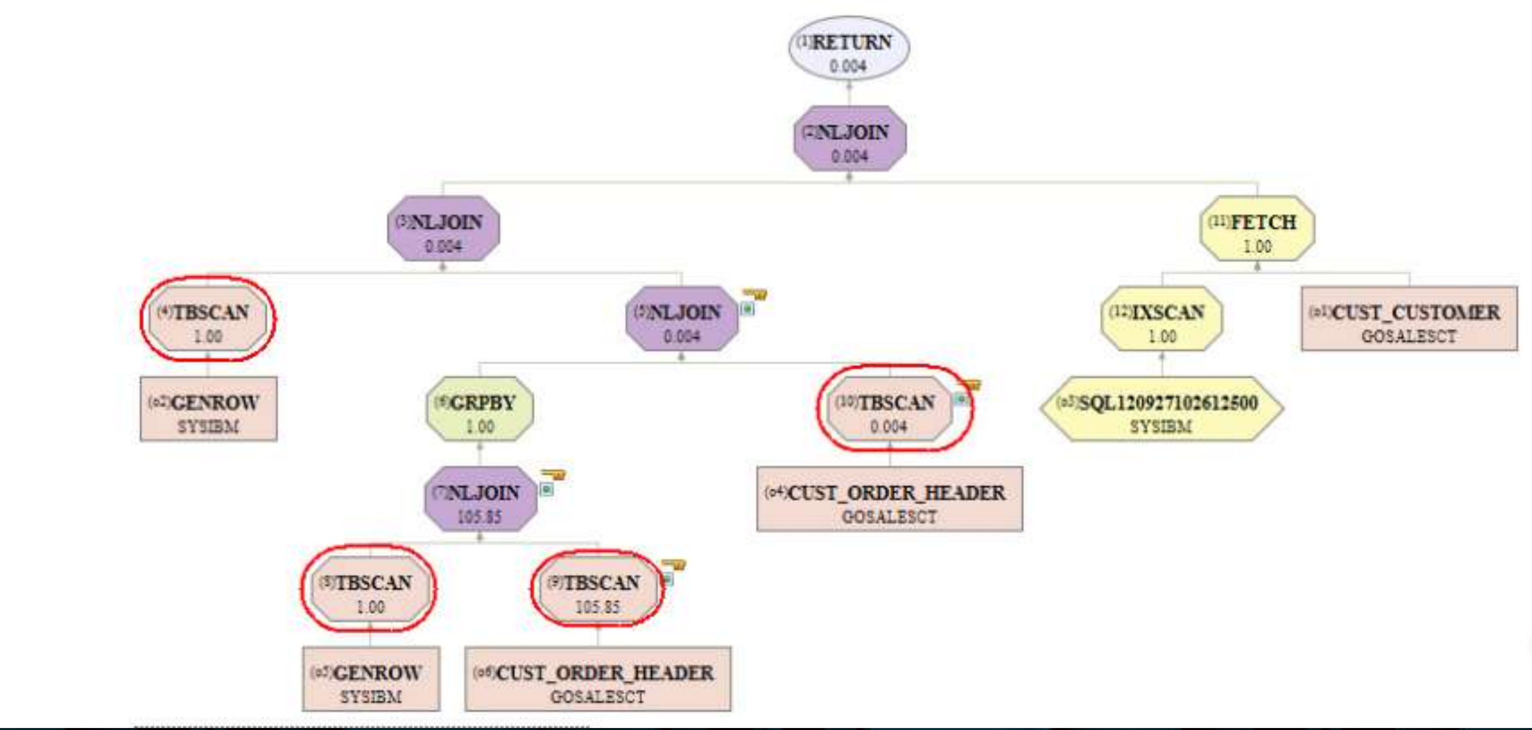
Index	Table	Join
One Fetch **IN(list) Index Access**	**Limited Partition Scan Using Non-partitioning index (NPI)**	**Nested Loop**
Matching Index Access **Sparse Index Access**	**Limited Partition Scan Using Partitioning Index**	**Hybrid Join: Type C or Type N**
NonMatching Index Access	**Limited Partition Scan Using Data Partitioned Secondary Index (DPSI**	**Star Join: Cartesian or Pair-wise**
List Prefetch	Table Scan	**Merge Scan**
Multiple Index Access	Partitioned Table Scan	Direct Row

Db2 11

Makes Dynamic

(Bold names use an Index)

Access Path Analysis



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

Tuning SQL

- FIND ALL Expensive Queries

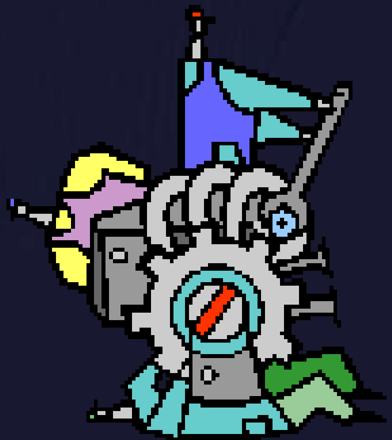
```
---------+---------+---
PROGNAME       PROCSU
---------+---------+---
EXPNPROG    121,059,664
EXPNPROG     21,059,664
ONESECPG         79,664
SUBSECPG          9,664
CHEEPPRG             64
FREEPROG              4
```

PROCSU is Too Expensive to Calculate!

2,147,483,647

2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647
2,147,483,647

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
- Anti-Joins
- Manual Query Rewrite

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

Fetch First Example

Query #1

```
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
- <.1sec response time required

Query #1 Tuned

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

- 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

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

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

No Op Example - Scan

SALES_ID.MNGR.REGION Index | MNGR Index | REGION Index

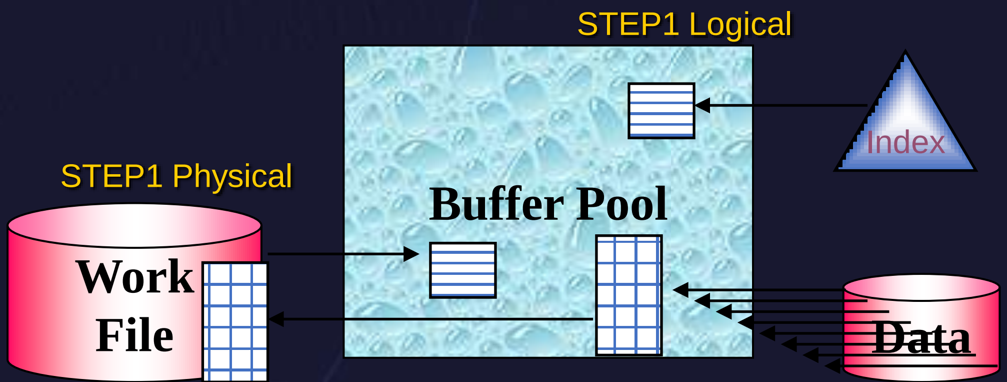
```
SELECT  S.QTY_SOLD
              , S.ITEM_NO
              , S.ITEM_NAME
FROM          SALE S
WHERE   S.SALES_ID > 44 +0
    AND       S.MNGR = :hv-mngr CONCAT ' '
    AND       S.REGION BETWEEN
               :hvlo AND :hvhi CONCAT ' '
ORDER BY S.REGION
FOR FETCH ONLY
WITH UR
```

- 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?

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



DISTINCT Table Expressions Example

```
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

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

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

Gets rid of Index Scan

Put a Query on a Diet

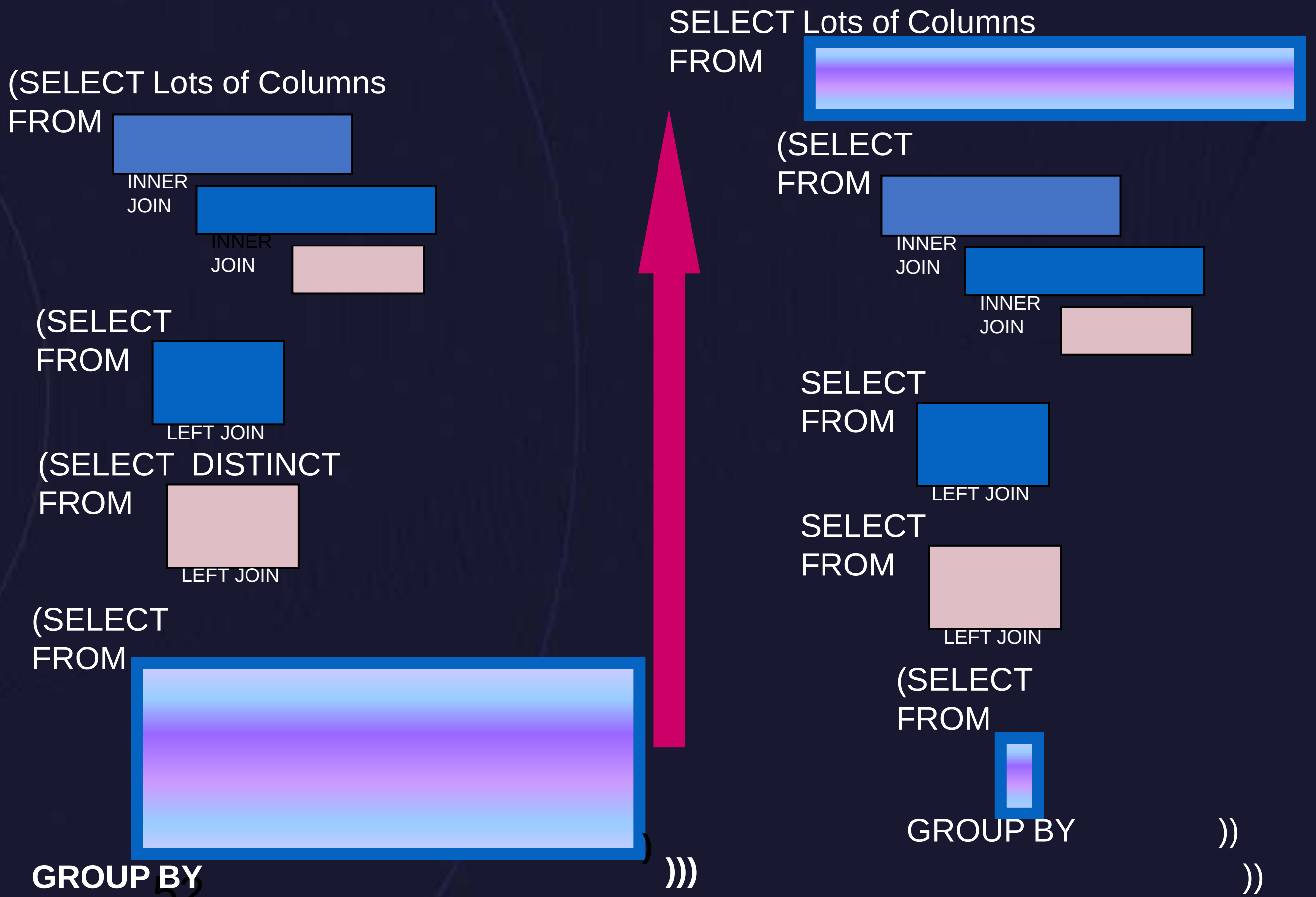
For Extreme Cases

(used on all platforms)

A Typical Data Warehouse Query

- Initial cost of 16 million timerons
  - WOULD NOT FINISH!
- 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

Before and After



Tuning Technique

- Extreme Cross Query Block Optimization
- 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!

What other query attributes are red flags to optimal performance?

What ever Tony said!

SQL Tuning Confidence Level

0% -- 100%

Speaker: Sheryl Larsen
Company: BMC
Email Address:
sheryl_Larsen@bmc.com

Don’t forget to fill out a session evaluation!