IBM System z Technology Summit

# What's new for SQL optimization in DB2 9 and DB2 10 for zOS

# Disclaimer



***THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.***

# Agenda

- **Bind/Prepare**
  - Plan management
  - Hints/Bind options
  - Explain
  - Dynamic Statement Caching
  - REOPT
- **Optimizer costing**
- **Runtime query performance**
- **Indexing**
- **Complex queries**

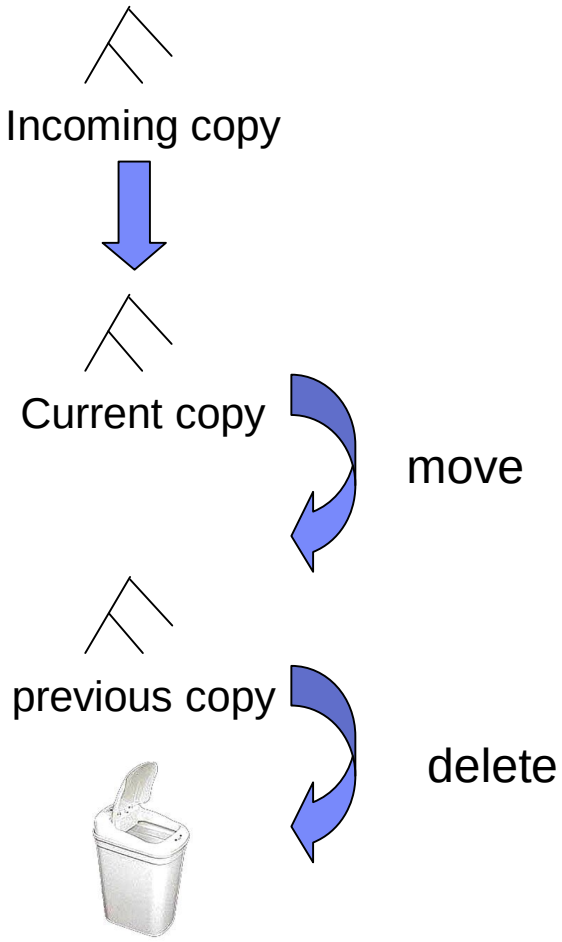
# Plan Management Overview

- **Ability to backup your static SQL packages (DB2 9)**

- **At REBIND**
  - Save old copies of packages in Catalog/Directory
  - Switch back to previous or original version
- **Two flavors**
  - BASIC
    - 2 copies: Current and Previous
  - EXTENDED
    - 3 copies: Current, Previous, Original
  - Default controlled by a ZPARM
  - Also supported as REBIND options

# Plan Management - BASIC support

REBIND … PLANMGMT(BASIC)




REBIND … SWITCH(PREVIOUS)

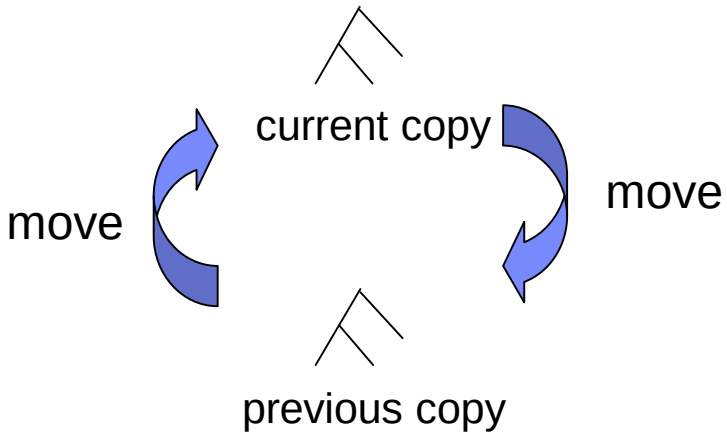
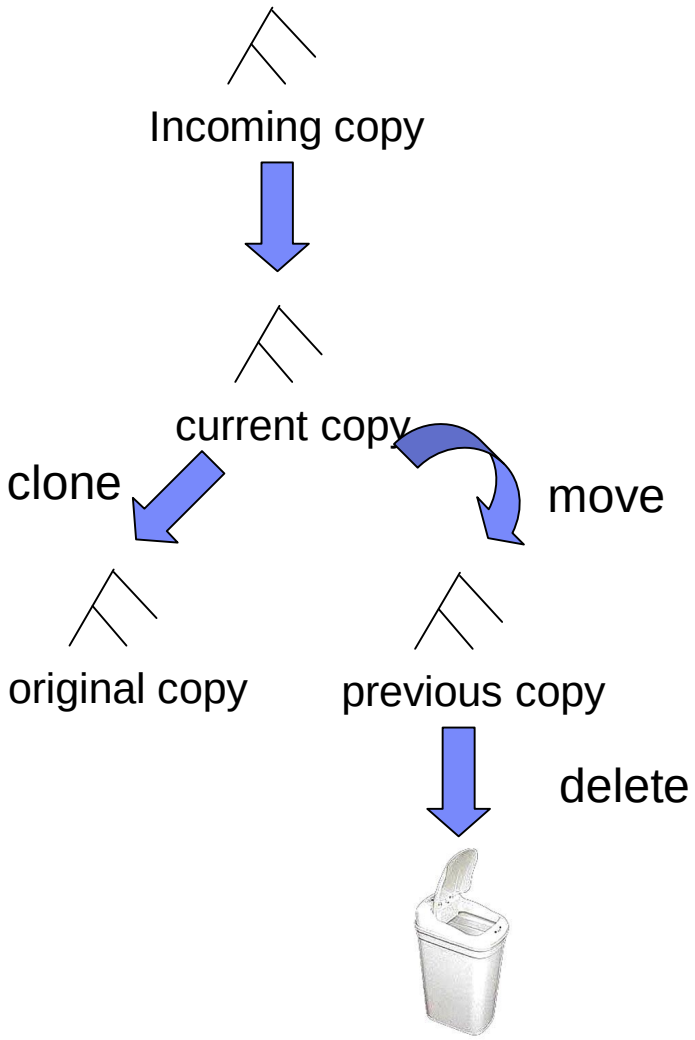


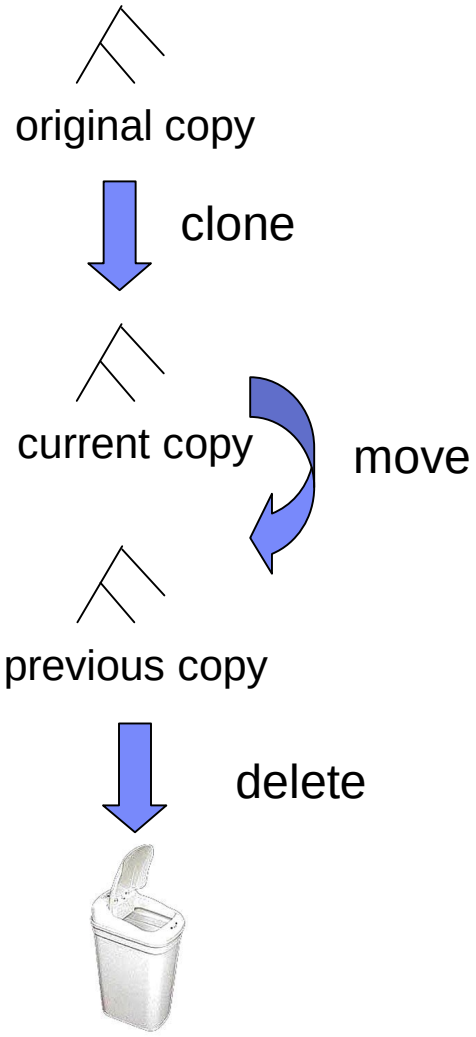

Chart is to be read from bottom to top

# Plan Management - EXTENDED support

REBIND … PLANMGMT(EXTENDED)




REBIND … SWITCH(ORIGINAL)



# Plan Management Notes

- **REBIND PACKAGE …**
  - PLANMGMT (BASIC)

    2 copies: Current and Previous
  - PLANMGMT (EXTENDED)

    3 copies: Current, Previous, Original

- **REBIND PACKAGE …**
  - SWITCH(PREVIOUS)

    Switch between current & previous
  - SWITCH(ORIGINAL)

    Switch between current & original

- **Most bind options can be changed at REBIND**
  - *But a few must be the same …*

- **FREE PACKAGE …**
  - PLANMGMTSCOPE(ALL) – Free package completely
  - PLANMGMTSCOPE(INACTIVE) – Free old copies

- **Catalog support**
  - SYSPACKAGE reflects active copy
  - SYSPACKDEP reflects dependencies of all copies
  - Other catalogs (SYSPKSYSTEM, …) reflect metadata for all copies

- **Invalidation and Auto Bind**
  - Each copy invalidated separately

3 important updates:

1. APAR PK80375 – SPT01 Compression (DB2 V8 & 9)
2. APAR PM09354 – Support DBPROTOCOL change
3. Article – Search for “Escaping the REBIND blues in DB2 9 for z/OS”

# DB2 10 Updates to Plan Management

- **SYSIBM.SYSPACKCOPY**
  - New catalog table
  - Hold SYSPACKAGE-style metadata for any previous or original package copies
  - No longer need to SWITCH to see information on inactive copies
    - Complaint from DB2 9

- **APRETAINDUP option of REBIND**
  - Default YES
    - Retain duplicate for BASIC or EXTENDED
  - Optional NO
    - Do not retain duplicate access path as PREVIOUS or ORIGINAL
      - PREVIOUS/ORIGINAL must be from DB2 9 or later

# What-if? BIND

- **BIND package to see what new**
- **Bind package EXPLAIN(ONLY) and/or SQLERROR(CHECK)**
  - Existing package copies are not overwritten
    - Performs explain or syntax/semantic error checks on SQL
  - Requires BIND, BINDAGENT, or EXPLAIN privilege.
  - Supported for BIND only
    - Not REBIND
    - Targeted to application changes
      - Eg. Development environment is DB2 LUW, production DB2 for z/OS

# Retrieving Access Path with EXPLAIN(NO)

- **EXPLAIN PACKAGE**
  - Extract existing PLAN_TABLE information for packages
    - NOT a new explain
    - The package/copy must be created on DB2 9 or later
  - Useful if you didn't BIND with EXPLAIN(YES)
    - Or PLAN_TABLE entries are lost

```
>>-EXPLAIN----PACKAGE----------->

>>-----COLLECTION--collection-name--PACKAGE--package-name--------->

>----+--------------------------+----+-------------------+-------->
     |                          |    |                   |
     +---VERSION-version-name---+    +---COPY--copy-id---+
```

    - COPY-ID can be 'CURRENT', 'PREVIOUS', 'ORIGINAL'

# Access Path Stability with statement level hints

- **Current limitations in hint matching**
  - QUERYNO is used to link queries to their hints – a bit fragile
  - For dynamic SQL, require a change to apps – can be impractical
- **New mechanisms:**
  - Associate query text with its corresponding hint … more robust
  - Hints enforced for the entire DB2 subsystem
    - irrespective of static vs. dynamic, etc.
  - Hints integrated into the access path repository
- **PLAN_TABLE isn't going away**
- **Only the "hint lookup" mechanism is being improved.**

# Statement level hints (cont.)

- **Steps to use new hints mechanism**
  - Populate a user table DSN_USERQUERY_TABLE with query text
  - Populate PLAN_TABLE with the corresponding hints
  - Run new command BIND QUERY
    - To integrate the hint into the repository.
  - FREE QUERY can be used to remove the hint.

# Statement-level BIND options

- **Statement-level granularity may be required rather than:**
  - Subsystem level ZPARMs (STARJOIN, SJTABLES, MAX_PAR_DEGREE)
  - Package level BIND options (REOPT, DEF_CURR_DEGREE)
- **For example**
  - Only one statement in the package needs REOPT(ALWAYS)

- **New mechanism for statement-level bind options:**
  - Similar to mechanism used for hints
  - DSN_USERQUERY_TABLE can also hold per-statement options

# Literal Replacement

- **Dynamic SQL with literals can now be re-used in the cache**
  - Literals replaced with &
    - Similar to parameter markers but not the same
- **To enable either you:-**
  - Put CONCENTRATE STATEMENTS WITH LITERALS in the PREPARE ATTRIBUTES clause
  - Or set LITERALREPLACEMENT in the ODBC initialization file
  - Or set the keyword enableLiteralReplacement='YES' in the JCC Driver
- **Lookup Sequence**
  - Original SQL with literals is looked up in the cache
  - If not found, literals are replaced and new SQL is looked up in the cache
    - Additional match on literal usability
    - Can only match with SQL stored with same attribute, not parameter marker
  - If not found, new SQL is prepared and stored in the cache

# Literal Replacement …

- **Example:**

```
WHERE ACCOUNT_NUMBER = 123456
```

  – This would be replaced by

```
WHERE ACCOUNT_NUMBER = &
```

- **Performance Expectation**
  - Using parameter marker still provides best performance
  - Biggest performance gain for repeated SQL with different literals
  - NOTE: Access path is not optimized for literals
    - True for parameter markers/host variables today
    - Need to use REOPT for that purpose

# Dynamic SQL REOPT - AUTO

- **For dynamic SQL with parameter markers**
  - DB2 will automatically reoptimize the SQL when
    - Filtering of one or more of the predicates changes dramatically
      - Such that table join sequence or index selection may change
    - Some statistics cached to improve performance of runtime check
  - Newly generated access path will replace the global statement cache copy.

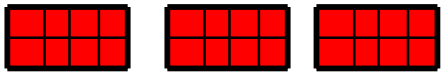
- **First optimization is the same as REOPT(ONCE)**
  - Followed by analysis of the values supplied at each execution of the statement

# Agenda

- **Bind/Prepare**
- **Optimizer costing**
  - RUNSTATS
  - Cost model enhancements
  - Subquery costing
- **Runtime query performance**
- **Indexing**
- **Complex queries**

# Clusterratio Enhancement

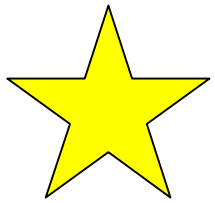
- **New Clusterratio formula in DB2 9**
  - Including new DATAREPEATFACTOR statistic
    - Differentiates density and sequential



Dense (and sequential)

Sequential (not dense)

- **Controlled by zparm STATCLUS**
  - ENHANCED is default
  - STANDARD disables, and is NOT recommended



- **Recommend RUNSTATS before mass REBIND in first version AFTER V8**

# Clusterratio Impacts

- **Clusterratio may be**
  - Higher for indexes
    - With many duplicates (lower colcardf)
      - In recognition of sequential RIDs
    - On smaller tables
      - Less clusterratio degradation from random inserts
    - Indexes that are reverse sequential
  - Lower for random indexes
    - No benefit from dynamic prefetch
- **Clusterratio(CR)/DataRepeatfactor (DRF) patterns**

| | **High DRF** | **Low DRF** |
|---|---|---|
| **High CR** | **Sequential but not dense** | **Density matching clustering or small table** |
| **Low CR** | **Random index** | **Unlikely** |

# Histogram Statistics

- RUNSTATS **will produce equal-depth histogram**
  - Each quantile (range) will have approx same number of rows
    - Not same number of values
  - Another term is range frequency

- **Example**
  - 1, 3, 3, 4, 4, 6, 7, 8, 9, 10, 12, 15 (sequenced)
  - Lets cut that into 3 quantiles.
    - 1, 3, 3, 4 ,4 6,7,8,9 10,12,15

| Seq No | Low Value | High Value | Cardinality | Frequency |
|---|---|---|---|---|
| **1** | **1** | **4** | **3** | **5/12** |
| **2** | **6** | **9** | **4** | **4/12** |
| **3** | **10** | **15** | **3** | **3/12** |

# Histogram Statistics Notes

- **RUNSTATS**
  - Maximum 100 quantiles for a column
  - Same value columns WILL be in the same quantile
  - Quantiles will be similar size but:
    - Will try to avoid big gaps inside quantiles
    - Highvalue and lowvalue may have separate quantiles
    - Null WILL have a separate quantile
- **Supports column groups as well as single columns**
- **Think “frequencies” for high cardinality columns**

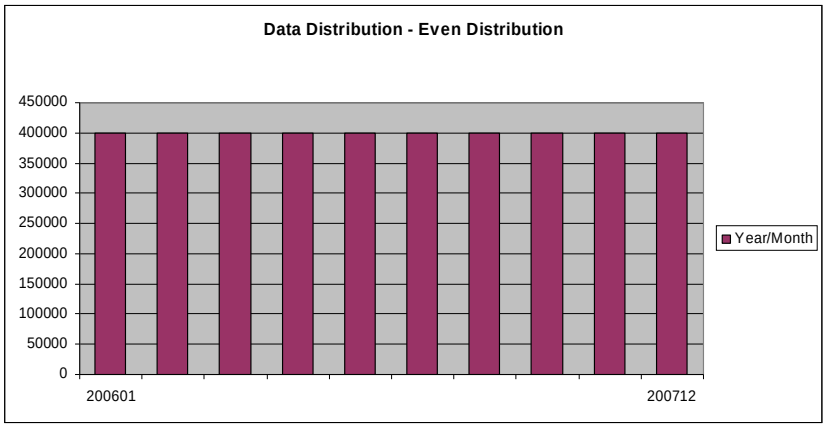
# Histogram Statistics Example

- **SAP uses INTEGER (or VARCHAR) for YEAR-MONTH**

WHERE YEARMONTH BETWEEN 200601 AND 200612

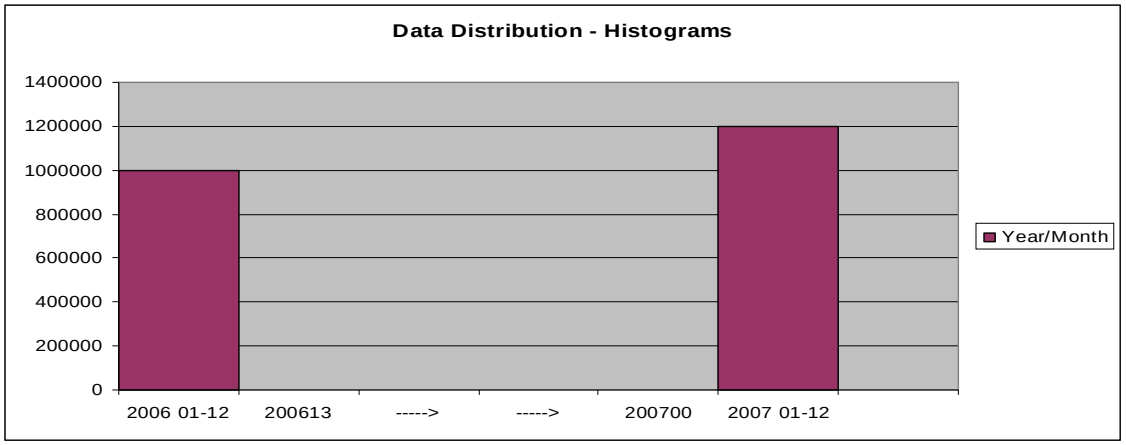
- Assuming data for 2006 & 2007
  - FF = (high-value – low-value) / (high2key – low2key)
  - FF = (200612 – 200601) / (200711 – 200602)
  - **10% of rows estimated to return**

Data assumed as evenly distributed between low and high range



# Histogram Statistics Example

- Example (cont.)
  - Data only exists in ranges 200601-12 & 200701-12
    - Collect via histograms
      - 45% of rows estimated to return




No data between
200613 & 200700

WHERE YEARMONTH BETWEEN 200601 AND 200612

# Autonomic Statistics Solution Overview

- **Autonomic Statistics is implemented though a set of Stored Procedures**
  - *Stored procedures are provided to enable administration tools and packaged applications to automate statistics collection.*
    - ADMIN_UTL_MONITOR
    - ADMIN_UTL_EXECUTE
    - ADMIN_UTL_MODIFY
  - Working together, these SP's
    - Determine what stats to collect
    - Determine when stats need to be collected
    - Schedule and Perform the stats collection
    - Records activity for later review
  - *See Chapter 11 "Designing DB2 statistics for performance" in the DB2 10 for z/OS Performance Monitoring and Tuning Guide for details on how to configure autonomic monitoring directly within DB2.*

# RUNSTATS Simplification/Performance Overview

- **RUNSTATS options to SET/UPDATE/USE a stats profile**
  - Integrate specialized statistics into generic RUNSTATS job
    - RUNSTATS … TABLE tbl COLUMN(C1)… **SET PROFILE**
      - Alternatively use **SET PROFILE FROM EXISTING STATS**
    - RUNSTATS … TABLE tbl COLUMN(C5)… **UPDATE PROFILE**
    - RUNSTATS … TABLE tbl **USE PROFILE**

- **New option for page-level sampling**
  - But what percentage of sampling to use?
    - RUNSTATS … TABLE tbl **TABLESAMPLE SYSTEM AUTO**

# Optimizer Validation with Realtime Stats

- **Index Probing & RTS lookup**
  - Estimate # of rids within a given start/stop index key range at bind/prepare
- **Exploited when these two conditions are met.**
  - Query has matching index-access local predicate
  - Predicate contain literals, or REOPT(ALWAYS|ONCE|AUTO)
- **And 1 of the following is also true**
  - Predicate is estimated to qualify no rows
  - Stats indicate the table contains no rows
  - Table is defined as VOLATILE or qualifies for NPGTHRSH
- **New EXPLAIN table to externalize runtime estimates**
  - User managed DSN_COLDIST_TABLE

# DB2 10 - Minimizing Optimizer Challenges

- **Potential causes of sub-optimal plans**
  - Insufficient statistics
  - Unknown literal values used for host variables or parameter markers

- **DB2 10 Optimizer will evaluate the risk for each predicate**
  - For example: WHERE BIRTHDATE < ?
    - Could qualify 0-100% of data depending on literal value used
  - As part of access path selection
    - Compare access paths with close cost and choose lowest risk plan

# Extending VOLATILE TABLE usage

- **VOLATILE TABLE support added in DB2 V8**
  - Targeted to SAP Cluster Tables
    - Use Index access whenever possible
    - **Avoids list prefetch**
      - Can be a problem for OR predicates or UPDATEs at risk of loop

- **DB2 10 provides VOLATILE to general cases**
  - Tables matching SAP cluster tables will maintain original limitations
    - Table with 1 unique index
  - Tables with > 1 index will follow NPGTHRSH rules
    - Use Index access whenever possible
    - **No limitation on list prefetch**
    - Less chance of getting r-scan when list-prefetch plan is only alternative

# Global Optimization - Problem Scenario 1

- **V8, Large Non-correlated subquery is materialized***

```
SELECT * FROM SMALL_TABLE A
WHERE A.C1 IN
        (SELECT B.C1 FROM BIG_TABLE B)
```

  - "BIG_TABLE" is scanned and put into workfile
  - "SMALL_TABLE" is joined with the workfile

- **V9 may rewrite non-correlated subquery to correlated**
  - Much more efficient if scan / materialisation of BIG_TABLE was avoided
  - Allows matching index access on BIG_TABLE

```
SELECT * FROM SMALL_TABLE A
WHERE EXISTS
  (SELECT 1 FROM BIG_TABLE B WHERE B.C1 = A.C1)
```

# Global Optimization - Problem Scenario 2

- **V8, Large outer table scanned rather than using matching index access***

```
SELECT * FROM BIG_TABLE A
WHERE EXISTS
  (SELECT 1 FROM SMALL_TABLE B WHERE A.C1 = B.C1)
```

  - "BIG_TABLE" is scanned to obtain A.C1 value
  - "SMALL_TABLE" gets matching index access

- **V9 may rewrite correlated subquery to non-correlated**

```
SELECT * FROM BIG_TABLE A
WHERE A.C1 IN
  (SELECT B.C1 FROM SMALL_TABLE B)
```

  - "SMALL_TABLE" scanned and put in workfile
  - Allows more efficient matching index access on BIG_TABLE

# Global Optimization

- **Global opt internally represent subqueries as virtual tables**
  - Allows subquery to be considered in different join sequences
  - May or may not represent a physical workfile
    - Additional row added to PLAN_TABLE for non-correlated subq
      - PM30425 adds this new row for correlated
  - Apply only to subqueries that cannot be transformed to joins
    - SELECT only (not INSERT/SELECT, UPDATE, DELETE)

Correlated or non-correlated?.....I shouldn't have to care!

# Agenda

- **Bind/Prepare**
- **Optimizer costing**
- **Runtime query performance**
  - Sort/sort avoidance
  - Sparse index
  - Predicate application
- **Indexing**
- **Complex queries**

# GROUP BY Sort Avoidance

- **Improved sort avoidance for GROUP BY**
  - Reorder GROUP BY columns to match available index

```
SELECT … FROM T1
GROUP BY C2, C1      ←GROUP BY in C2, C1 sequence
Index 1 (C1, C2)     ←Index in C1, C2 sequence
```

  - Remove 'constants' from GROUP BY ordering requirement

```
SELECT … FROM T1
WHERE C2 = 5         ←C2 Constant
GROUP BY C2, C1
```

    - ordering requirement reduced to just C1

# GROUP BY Sort Avoidance Implications

- **Implications of improved sort avoidance for GROUP BY**
  - May improve query performance!!!
  - Data may be returned in a different order
    - Always been true in any DB2 release
      - Also true in other DBMSs
    - Relational theory states that order is NOT guaranteed without ORDER BY

# Sort Performance Enhancements

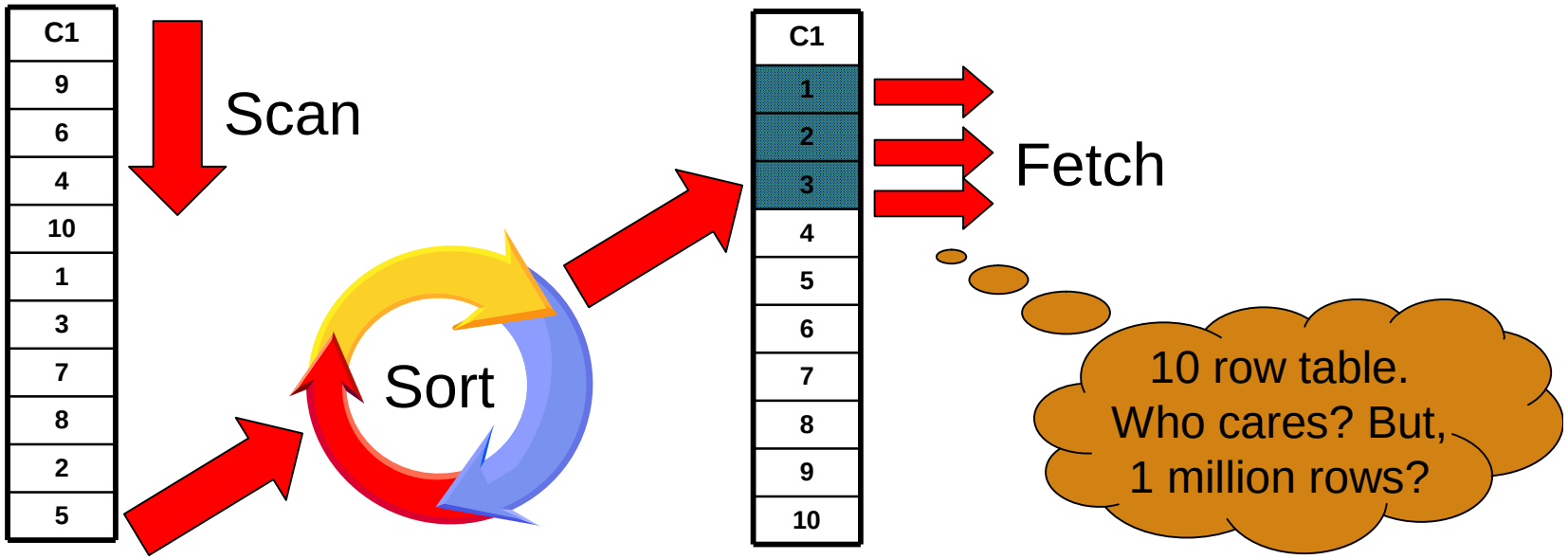
- **FETCH FIRST n ROWS ONLY (FFnR) and Sort**
  - DB2 9 added in-memory replacement for FFnR to avoid sort
    - Provided (n * (sort key + data)) < 32K
  - DB2 10 extends this to 128K

- **Avoid workfile usage for small sorts**
  - DB2 9 avoided allocating WF for final sort only
    - If <= 255 rows and result < 32K (sort key + data)
  - DB2 10 extends this to intermediate sorts also
    - Except for parallelism or SET function

# Improving sort with FETCH FIRST

- **DB2 V8 example**
  - Sort is not avoided via index
    - Must sort all qualified rows

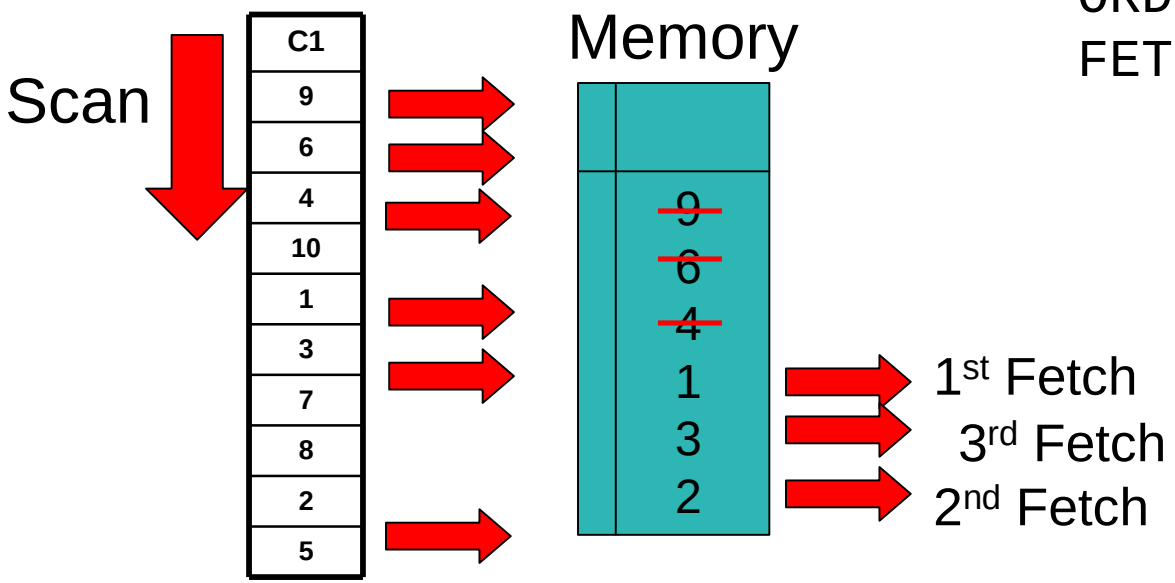
```
SELECT C1
FROM   T
ORDER BY C1
FETCH FIRST 3 ROWS ONLY
```



# Improving sort with FETCH FIRST

- **DB2 9 example**
  - New algorithm for in-memory swap avoids (traditional) sort
    - Pointers maintain order

```
SELECT C1
FROM  T
ORDER BY C1
FETCH FIRST 3 ROWS ONLY
```




Suggestion: Add FETCH FIRST when subset is required.

# Improvements to predicate application

- **Major enhancements to OR and IN predicates**
  - Improved performance for AND/OR combinations and long IN-lists
    - General performance improvement to stage 1 predicate processing
  - IN-list matching
    - Matching on multiple IN-lists
    - Transitive closure support for IN-list predicates
    - List prefetch support
    - Trim IN-lists from matching when preceding equals are highly filtering
  - SQL pagination
    - Single index matching for complex OR conditions

- **Many stage 2 expressions to be executed at stage 1**
  - Stage 2 expressions eligible for index screening
    - Not applicable for list prefetch
  - Externalized in DSN_FILTER_TABLE column PUSHDOWN

# IN-list Table - Table Type 'I' and Access Type 'IN'

- The IN-list predicate will be represented as an in-memory table if:
  - **List prefetch is chosen, OR**
  - **More than one IN-list is chosen as matching.**
  - **The EXPLAIN output associated with the in-memory table will have:**
    - **New Table Type: TBTYPE – 'I'**
    - **New Access Type: ACTYPE – 'IN'**

```
SELECT *
   FROM T1
   WHERE T1.C1 IN (?, ?, ?);
```

| QBNO | PLANNO | METHOD | TNAME | ACTYPE | MC | ACNAME | QBTYPE | TBTYPE | PREFETCH |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | DSNIN001(01) | ***IN*** | 0 | | SELECT | ***I*** | |
| 1 | 2 | 1 | T1 | I | 1 | T1_IX_C1 | SELECT | T | ***L*** |

# IN-list Predicate Transitive Closure (PTC)

```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
  AND T1.C1 IN (?, ?, ?)

  AND T2.C1 IN (?, ?, ?) ←   Optimizer can generate
                             this predicate via PTC
```

- **Without IN-list PTC (DB2 9)**
  - Optimizer will be unlikely to consider T2 is the first table accessed
- **With IN-list PTC (DB2 10)**
  - Optimizer can choose to access T2 or T1 first.

# SQL Pagination

- **Targets 2 types of queries**
  - Cursor scrolling (pagination) SQL
    - Retrieve next n rows
      - Common in COBOL/CICS and any screen scrolling application
    - Not to be confused with “scrollable cursors”
  - Complex OR predicates against the same columns
    - Common in SAP

- **In both cases:**
  - The OR (disjunct) predicate refers to a single table only.
  - Each OR predicate can be mapped to the same index.
  - Each disjunct has at least one matching predicate.

# Simple scrolling – Index matching and ORDER BY

- Scroll forward to obtain the next 20 rows
  - Assumes index is available on (LASTNAME, FIRSTNAME)
  - WHERE clause may appear as:

```
WHERE (LASTNAME='JONES' AND FIRSTNAME>'WENDY')
   OR (LASTNAME>'JONES')
ORDER BY LASTNAME, FIRSTNAME;
```

  - DB2 10 supports
    - Single matching index access with sort avoided 
  - DB2 9 requires
    - Multi-index access, list prefetch and sort
    - OR, extra predicate (AND LASTNAME >= ‘JONES’) for matching single index access and sort avoidance

# Complex OR predicates against same index

- Given WHERE clause
  - And index on one or both columns

```
WHERE (LASTNAME='JONES' AND FIRSTNAME='WENDY')
   OR (LASTNAME='SMITH' AND FIRSTNAME='JOHN');
```

- DB2 9 requires
  - Multi-index access with list prefetch
- DB2 10 supports
  - Matching single index access – no list prefetch
  - Or, Multi-index access with list prefetch

# Minimizing impact of RID failure

- **RID overflow can occur for**
  - Concurrent queries each consuming shared RID pool
  - Single query requesting > 25% of table or hitting RID pool limit
- **DB2 9 will fallback to tablespace scan***
- **DB2 10 will continue by writing new RIDs to workfile**

  - Work-file usage may increase
    - Mitigate by increasing RID pool size (default increased in DB2 10).
    - MAXTEMPS_RID zparm for maximum WF usage for each RID list

* Hybrid join can incrementally process. Dynamic Index ANDing will use WF for failover.

# Agenda

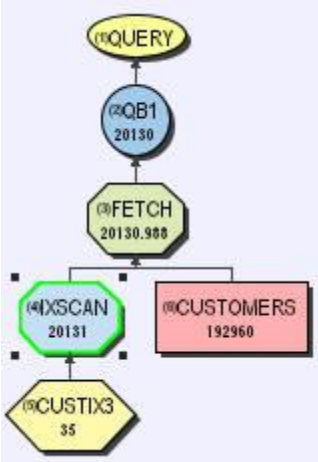
- **Bind/Prepare**
- **Optimizer costing**
- **Runtime query performance**
- **Indexing**
  - Index on expression
  - Tracking index use
  - Sparse index
  - Include columns
- **Complex queries**

# Index on Expression

- **DB2 9 supports “index on expression”**
  - Can turn a stage 2 predicate into indexable

```
SELECT *
FROM CUSTOMERS
WHERE YEAR(BIRTHDATE) = 1971
```

```
CREATE INDEX ADMF001.CUSTIX3
    ON ADMF001.CUSTOMERS
(YEAR(BIRTHDATE) ASC)
```




Previous FF = 1/25
Now, RUNSTATS collects frequencies. Improved FF accuracy

| Name | Value |
|---|---|
| Input RIDs | 192960 |
| Index Leaf Pages | 241 |
| Matching Predicates | Filter Factor |
| ADMF001.CUSTOMERS.= CAST(1971 AS INTEGER) | 0.1043 |
| Scanned Leaf Pages | 26 |
| Output RIDs | 20131 |
| Total Filter Factor | 0.1043 |
| Matching Columns | 1 |

# Index Enhancement - Tracking Usage

- **Additional indexes require overhead for**
  - Utilities
    - REORG, RUNSTATS, LOAD etc
  - Data maintenance
    - INSERT, UPDATE, DELETE
  - Disk storage
  - Optimization time
    - Increases optimizer's choices
- **But identifying unused indexes is a difficult task**
  - Especially in a dynamic SQL environment

# Tracking Index Usage

- **RTS records the index last used date.**
  - SYSINDEXSPACESTATS.LASTUSED
    - Updated once in a 24 hour period
      - RTS service task updates at 1st externalization interval (set by STATSINT) after 12PM.
    - if the index is used by DB2, update occurs.
    - If the index was not used, no update.

- **"Used", as defined by DB2 as:**
  - As an access path for query or fetch.
  - For searched UPDATE / DELETE SQL statement.
  - As a primary index for referential integrity.
  - To support foreign key access

# Tracking Index Usage Implications

- **What can you do with this information?**
  - LAST_USED only shows when the index was last used
    - Cannot predict future use
  - Assume you decide to DROP an index due to lack of usage
    - Is the index UNIQUE?
      - Is there another index that can guarantee that UNIQUEness?
    - Related statistics will be dropped
      - Same issue as “What If?” Optimization
      - For index on C1, C2, C3
        > RUNSTATS options to collect statistics

```
COLGROUP (C1) FREQVAL COUNT 10
COLGROUP (C1, C2, C3)
```

# Data Caching and Sparse Index

- **Data Caching**
  - Built at runtime
    - Is a runtime enhancement to sparse index
  - Extended to non-star join in DB2 9

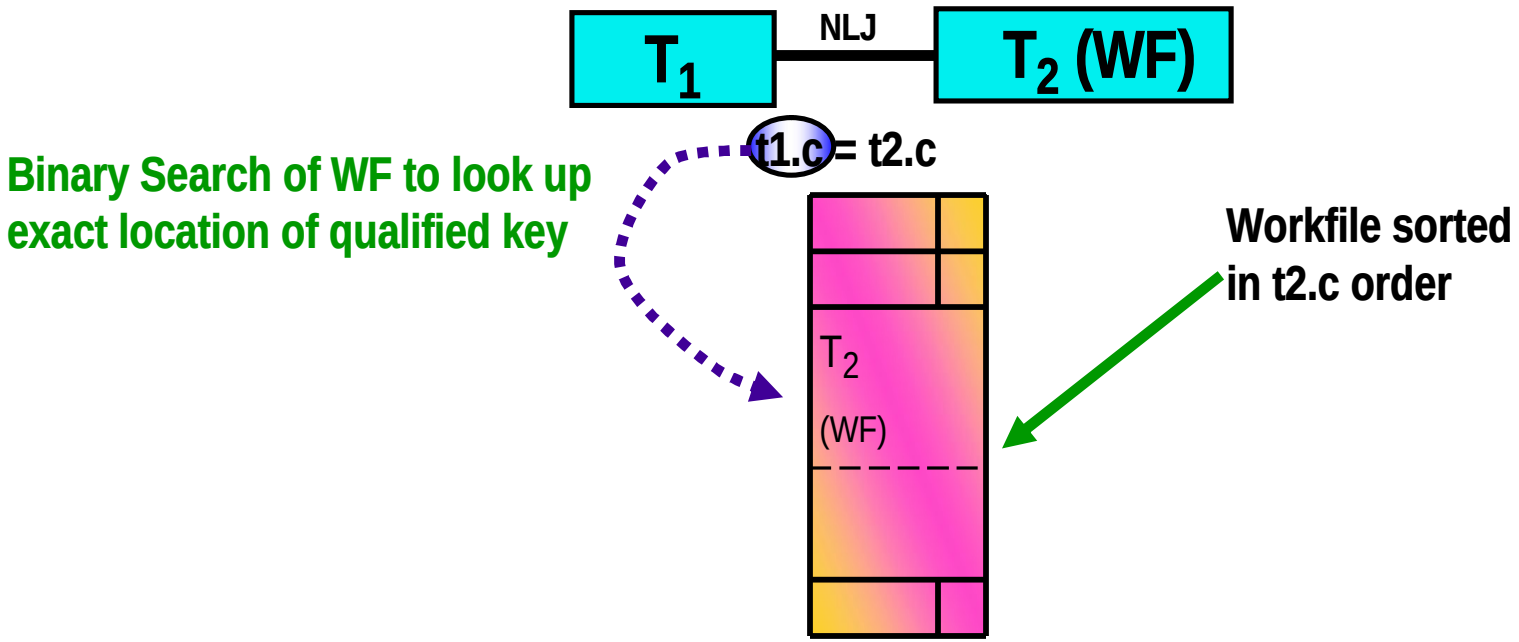
- **New ZPARM MXDTCACH**
  - Maximum extent in MB, for data caching per thread
  - If memory is insufficient
    - Fall-back to sparse index at runtime

- **Considered when lacking an index on join column(s):**
  - Temporary tables
  - Subqueries converted to joins
  - …..any table

# How does Data Caching WF work?

- Data Cache contains the full result of materialized result
  - Sparse index will be a subset of WF entries

- Example, WF may have 10,000 entries
  - Cache is “binary searched” to find target location of search key



# Index Include Columns

- **Index INCLUDE columns**
  - Create an Index as UNIQUE, and add additional columns
  - Ability to consolidate redundant indexes

```
INDEX1 UNIQUE (C1)
INDEX2 (C1,C2)
```

Consolidate to

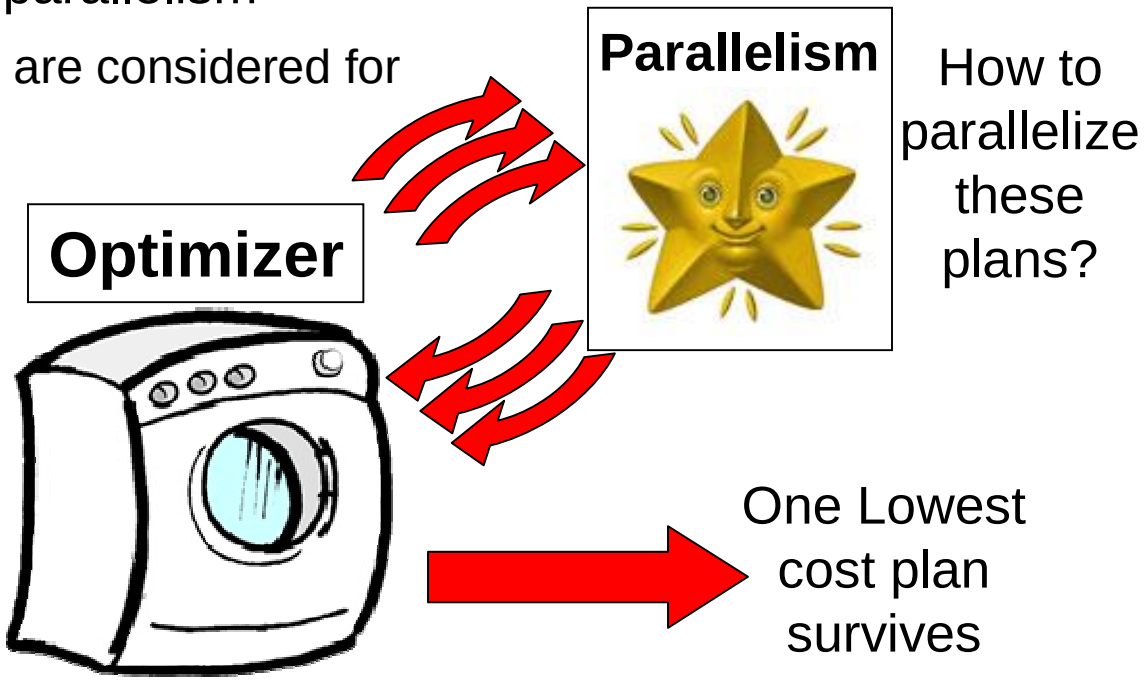
```
INDEX1 UNIQUE (C1) INCLUDE (C2)
```

# Agenda

- **Bind/Prepare**
- **Optimizer costing**
- **Runtime query performance**
- **Indexing**
- **Complex queries**
  - Parallelism
  - BI/DW

# Parallelism Enhancements

- **In V8**
  - Lowest cost is BEFORE parallelism
- **In DB2 9**
  - Lowest cost is AFTER parallelism
    - Only a subset of plans are considered for parallelism



# Additional DB2 9 Parallelism Enhancements

- **Degree can cut on non-leading table**
  - –Benefit for leading workfile, 1-row table etc.
- **Histogram statistics exploited for more even distribution**
- **New zparm PARA_EFF**
  - –Controls optimizer cost reduction applied for parallelism benefit
    - Default 50 (%)
    - Lower PARAMDEG can tolerate higher PARA_EFF
    - Higher PARAMDEG may mean lower PARA_EFF

# Removal Of Parallelism Restrictions #1

- Support parallelism for multi-row fetch
  - **In previous releases**
    - parallelism is disabled for the last parallel group in the top level query block
      - if there is no more table to join after the parallel group
      - and there is no GROUP BY clause or ORDER BY clause
  - Example:- SELECT * FROM CUSTOMER
    - There is no parallel group in the query and there are no table joins
    - There is no GROUP BY clause
    - There is no ORDER BY clause
    - So NO PARALLELISM will be used

- **This restriction is only removed if the CURSOR is DECLARED as READ ONLY**
  - Ambiguous Cursors will not have the restriction removed

# Removal Of Parallelism Restrictions #2

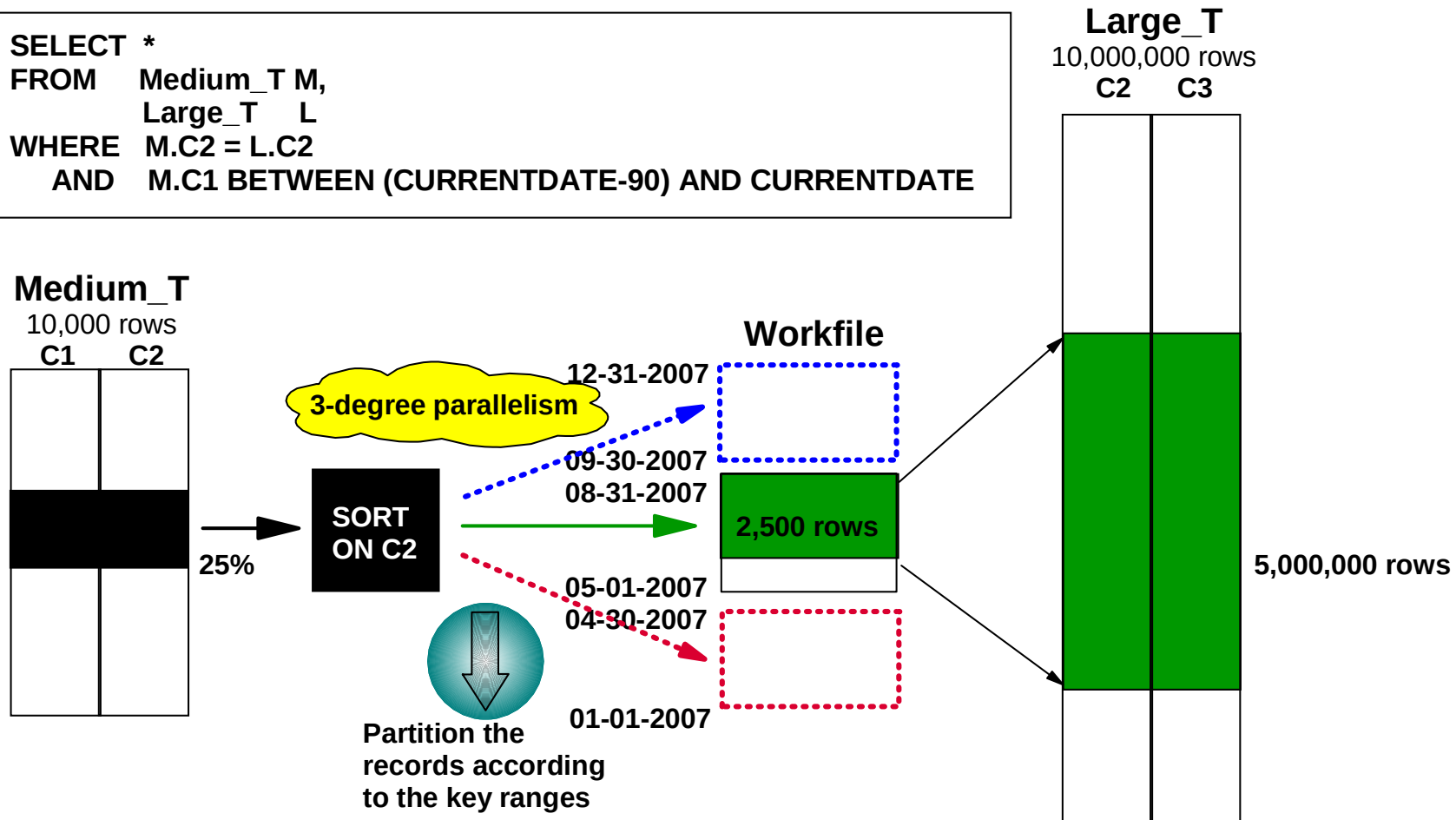
- **Allow parallelism if a parallel group contains a work file**
  - DB2 generates temporary a work file when view or table expression is materialized
  - This type of work file can not be shared among child task in previous releases of DB2, hence parallelism is disabled
  - DB2 10 will make the work file shareable
    - only applies to CP mode parallelism and no full outer join case

# Parallelism Enhancements - Effectiveness

- **Previous Releases of DB2 may use Key Range Partitioning**
  - Key Ranges Decided at Bind Time
  - Based on Statistics (low2key, high2key, column cardinality)
    - Assumes uniform data distribution
    - Histograms can help
      - But rarely collected
  - If Statistics are outdated or data is not uniformly distributed what happens to performance?

# Key range partition - Today

```
SELECT  *
FROM    Medium_T M,
        Large_T  L
WHERE   M.C2 = L.C2
  AND   M.C1 BETWEEN (CURRENTDATE-90) AND CURRENTDATE
```



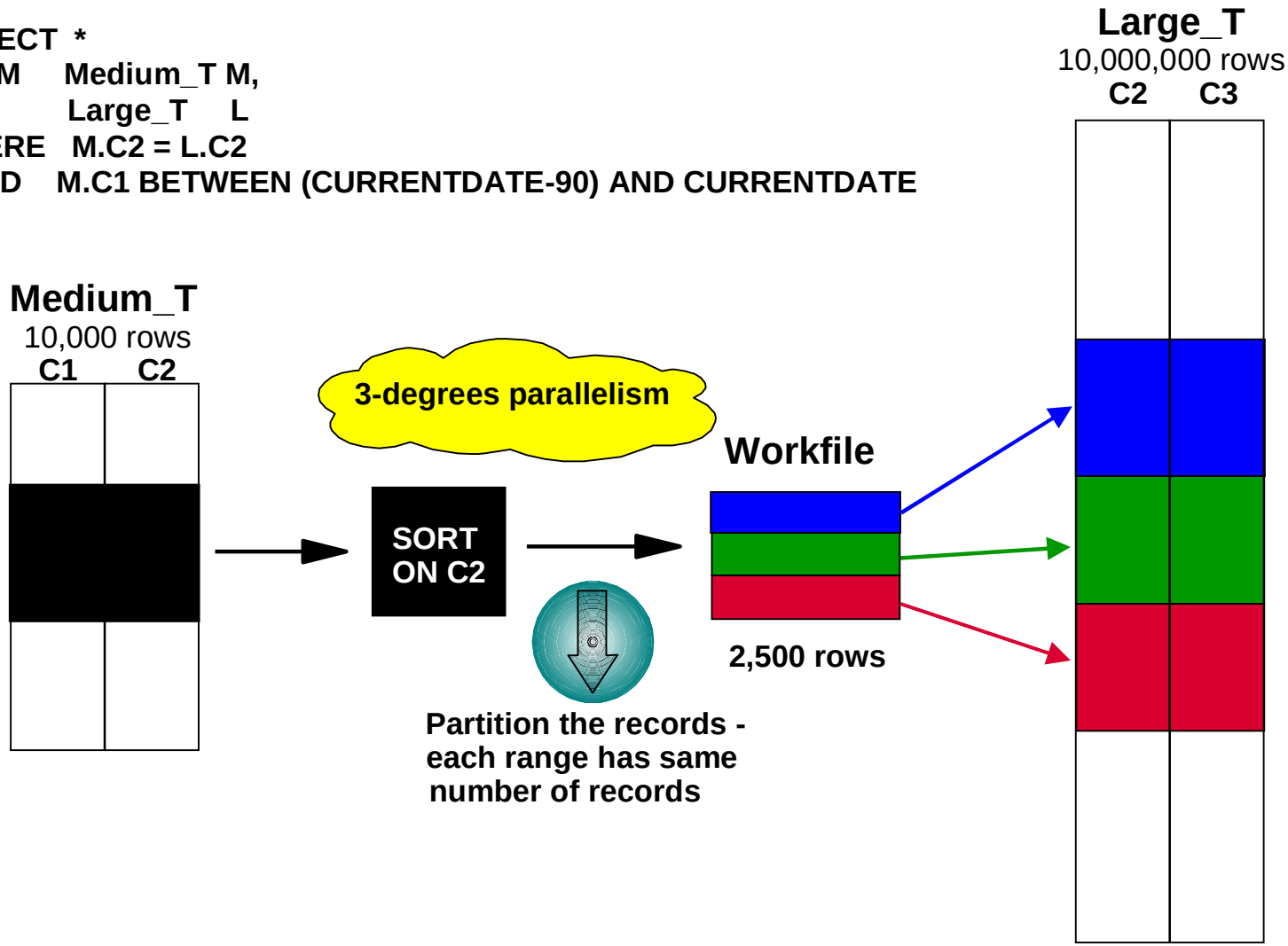

**M.C1 is date column, assume currentdate is 8-31-2007, after the between predicate is applied, only rows with date between 06-03-2007 and 8-31-2007 survived, but optimizer chops up the key ranges within the whole year after the records are sorted :-(**

# Parallelism Effectiveness – Record range

- **DB2 10 can use** Dynamic record range partitioning
  - Materialize the intermediate result in a sequence of join processes
  - Results divided into ranges with equal number of records
  - Division doesn't have to be on the key boundary
    - Unless required for group by or distinct function
  - Record range partitioning is dynamic
    - no longer based on the key ranges decided at bind time
  - Now based on number of composite records and parallel degree
    - Data skew, out of date statistics etc. will not have any effect on performance

# Dynamic record range partition

```
SELECT  *
FROM    Medium_T M,
        Large_T  L
WHERE   M.C2 = L.C2
  AND   M.C1 BETWEEN (CURRENTDATE-90) AND CURRENTDATE
```



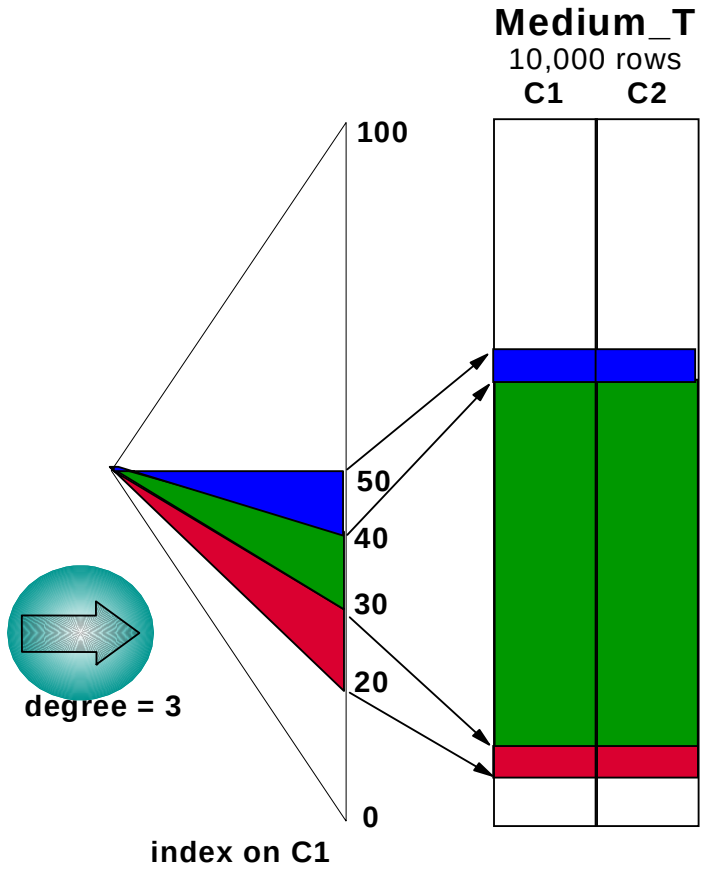
# Parallelism Effectiveness - Straw Model

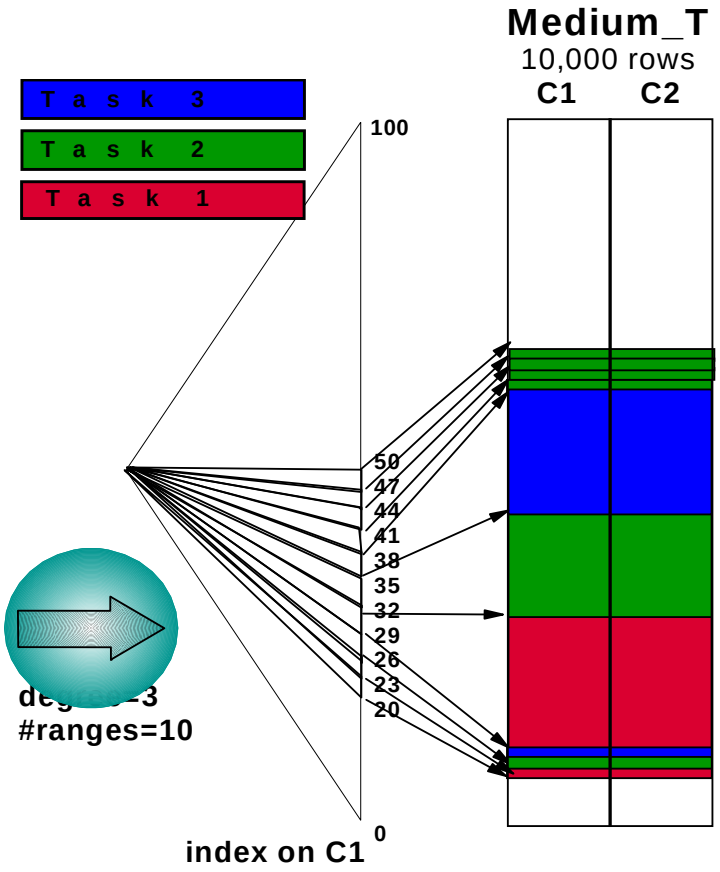
- **Previous releases of DB2 divide the number of keys or pages by the number representing the parallel degree**
  - One task is allocated per degree of parallelism
  - The range is processed and the task ends
  - Tasks may take different times to process

- **DB2 10 can use the Straw Model workload distribution method**
  - More key or page ranges will be allocated than the number of parallel degrees
  - The same number of tasks as before are allocated (same as degree)
  - Once a task finishes it's smaller range it will process another range
  - Even if data is skewed this new process should make processing faster

# STRAW Model

```
SELECT  *
FROM    Medium_T M
WHERE   M.C1 BETWEEN 20 AND 50
```



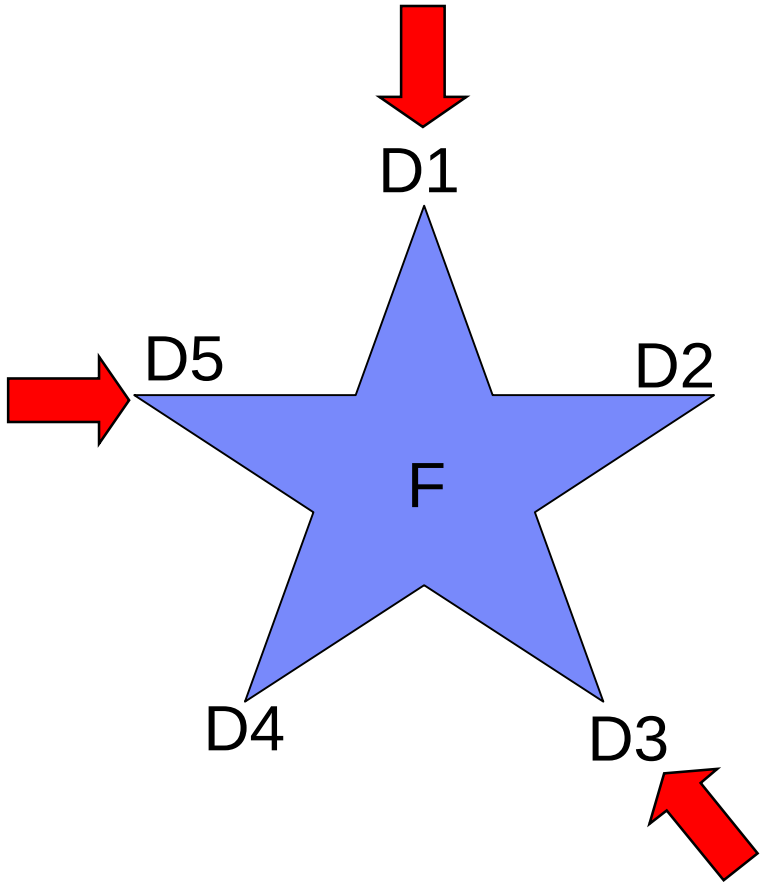




**Divided in key ranges before DB2 10**

**Divided in key ranges with Straw Model**

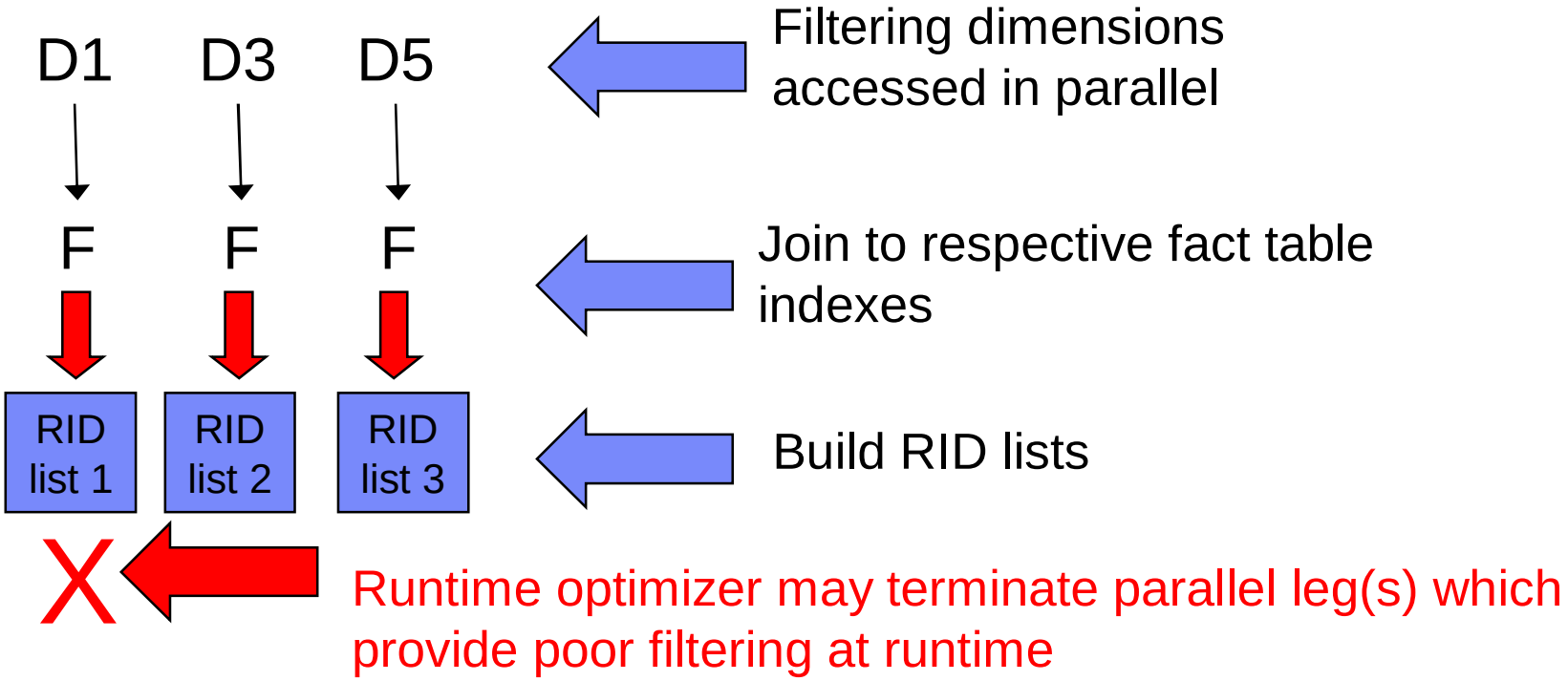
# Dynamic Index ANDing Challenge

- **Filtering may come from multiple dimensions**
  - Creating multi-column indexes to support the best combinations is difficult



# Index ANDing – Pre-Fact

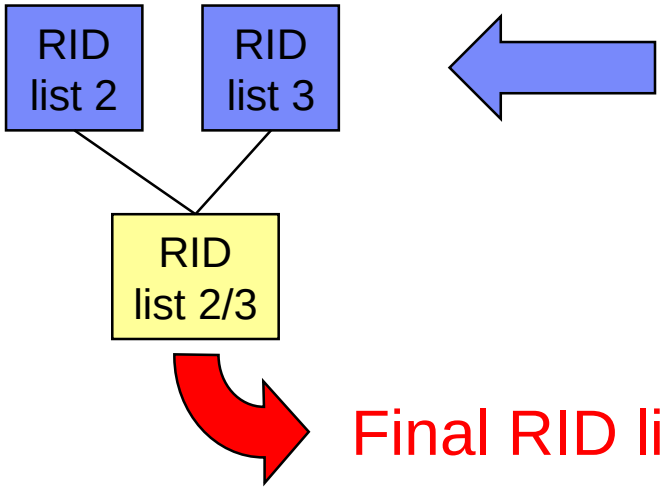
- **Pre-fact table access**
  - –Filtering may not be (truly) known until runtime



# Index ANDing – Fact and Post-Fact

- **Fact table access**
  - –Intersect filtering RID lists
  - –Access fact table
    - •From RID list
- **Post fact table**
  - –Join back to dimension tables

Using parallelism




Remaining RID lists are "ANDed" (intersected)

Final RID list used for parallel fact table access

# Dynamic Index Anding Highlights

- **Pre-fact table filtering**
  - Filtering dimensions accessed concurrently
- **Runtime optimization**
  - Terminate poorly filtering legs at runtime
- **More aggressive parallelism**
- **Fallback to workfile for RID pool failure**
  - Instead of r-scan

APAR PK76100 – zparm to enable EN_PJSJ