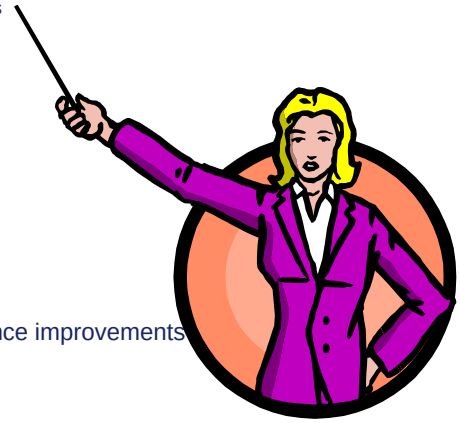
DB2 10 for z/OS
Query Optimizer Enhancements

Carlos Guardia
Executive IT/S
IBM Software Group

Agenda

- Access path management
  - ✓Dynamic Statement Cache Enhancements
  - ✓ Access Path Stability
  - ✓ Instance Based Statement Hints/Options
- Query performance improvements
  - ✓ Predicate application
    - IN-list
    - Complex ORs
    - View/Table expr Merge
  - ✓Safe Query Optimization
  - ✓ Parallelism Enhancements
- RUNSTATS usability and performance improvements
  - ✓ Auto-Stats
  - ✓ RUNSTATS Simplification/Performance



Agenda

- Access path management
  - ✓ Dynamic Statement Cache Enhancements
  - ✓ Access Path Stability
  - ✓ Instance Based Statement Hints/Options
- Query performance improvements
  - ✓ Predicate application
    - IN-list
    - Complex ORs
    - View/Table expr Merge
  - ✓ Safe Query Optimization
  - ✓ Parallelism Enhancements
- ✓ Runstats usability and performance improvements
  - ✓ Auto-Stats
  - ✓ RUNSTATS Simplification/Performance

DSC Enhancements: Literal Replacement

- **Dynamic SQL with literals can now be re-used in the cache (DB2 10 NFM)**
  - 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
    - Can only match with SQL stored with same attribute, not parameter marker
  - If not found, new SQL is prepared and stored in the cache
- 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

Access path stability extensions: Getting Information on Prior Packages

- **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

- **EXPLAIN PACKAGE**
  - 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---+
```

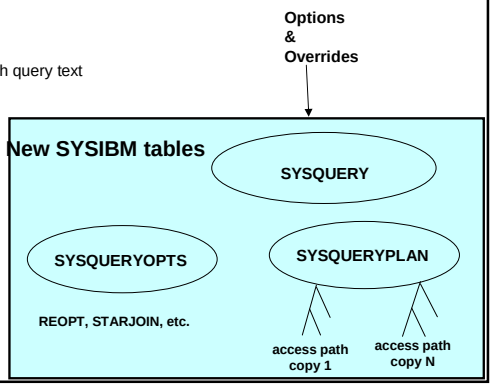
  - COPY-ID can be 'CURRENT', 'PREVIOUS', 'ORIGINAL'

Access path stability extensions: Bind/Rebind package new options

- **BIND PACKAGE and REBIND PACKAGE APREUSE/ APCOMPARE:**
  - **APREUSE YES/NO**: The default is APREUSE(NO).
    - APREUSE(YES) - DB2 attempts to reuse existing access paths.
    - Enforcement is not guaranteed in all cases.
  - **APCOMPARE WARN/ERROR/NONE**:
    - With WARN and ERROR, DB2 compares the old and new access paths.
    - If the access paths are structurally dissimilar, DSNT285I message produced.
      - WARN - DB2 will continue processing.
      - ERROR - DB2 will terminate the processing of the package.
      - NONE - DB2 does not perform access path comp. Default.
  - APREUSE/APCOMPARE – supp. on packages created V9 or higher
- **BIND PACKAGE EXPLAIN(ONLY) & SQLERROR(CHECK)**
  - Existing package copies are not overwritten
    - Performs explain or syntax/semantic error checks on SQL
- **REBIND option APRETAINDUP**
  - Default YES: Retain duplicate for BASIC or EXTENDED
  - Optional NO: Do not retain duplicate access path as PREVIOUS or ORIGINAL (saves space on SPT01)
    - If a duplicate is NOT kept (APRETAINDUP(NO))
      - SWITCH is not possible to non-existent copy
      - EXPLAIN PACKAGE not possible for non-existent copy
  - APRETAINDUP – supp. on packages created V9 or higher

Access Path Stability with Robust hints system

- 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.
- 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.



Agenda

- Access path management
  - ✓Dynamic Statement Cache Enhancements
  - ✓ Access Path Stability
  - ✓ Instance Based Statement Hints/Options
- Query performance improvements
  - ✓ Predicate application
    - IN-list
    - Complex ORs
    - View/Table expr Merge
  - ✓Safe Query Optimization
  - ✓ Parallelism Enhancements
- Runstats usability and performance improvements
  - ✓ Auto-Stats
  - ✓ RUNSTATS Simplification/Performance

Predicate application: improvements

- **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: DB2 10 removes the restriction that only one IN-list predicate can be a matching predicate.
    - 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

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'**

DSNIN indicates that it relates to IN-list
The # after DSNIN (001) represents the predicate number
The # in parenthesis (01) represents the query block number

```
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*

Example

```
SELECT P_SIZE, P_TYPE
FROM PART
WHERE P_MFGR IN ('MANUFACTURER#3','MANUFACTURER#2')
AND P_SIZE IN (1,3,5,7,10)
AND P_TYPE IN ('EC','AN','LA','ST','AB');
```

Index UXP@SZPT on table PART columns P_SIZE and P_TYPE

- **In DB2 10, both IN-lists can be matching predicates (CM)**
  - ✓Prior to DB2 10, DB2 could only match on one column of index UXP@SZPT.
  - ✓If both of the IN-lists are matching predicates, then the values of the IN-list predicates are merged and stored in an IN-list in-memory table
- **The values of the IN-list predicates are merged and stored in an IN-list in-memory table**
  - ✓in-memory table is populated with the following pairs of values for the pairing P_SIZE, P_TYPE:
- ✓**Performance:**
  - ✓77% reduction in elapsed time
  - ✓98% reduction in CPU time
  - ✓99.2% reduction in the number of index getpages
  - ✓96% reduction in the number of list prefetch requests

(1,'EC'),(1,'AN'),(1,'LA'),(1,'ST'),(1,'AB'),
(3,'EC'),(3,'AN'),(3,'LA'),(3,'ST'),(3,'AB'),
(5,'EC'),(5,'AN'),(5,'LA'),(5,'ST'),(5,'AB'),
(7,'EC'),(7,'AN'),(7,'LA'),(7,'ST'),(7,'AB'),
(10,'EC'),(10,'AN'),(10,'LA'),(10,'ST'),(10,'AB')

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.

Reducing Matchcols for IN-lists

```
SELECT *
FROM T1                                  <-- Index on C1,C2
WHERE C1 = ?
  AND C2 IN (?, ?, ?, ?, ?) <-- Optimizer may
                                choose not to match
```

- If the equals (=) predicates provide strong filtering
  - **Optimizer may choose not to match on the IN-list**
    - Instead apply as index screening
    - To avoid overhead of additional index probing
  - Example above
    - MATCHCOLS reduced from 2 to 1
    - ACCESSTYPE changed from “N” to “I”
  - Change in the access type from an IN-list index scan to an index access.
  - Example: DB2 10 MATCHCOLS reduced from 4 to 3, ACCESSTYPE changed from “N” to “I”
    - About a 15% reduction in CPU time

Complex ORs: SQL Pagination targets 2 classes of OR queries:

- **Cursor scrolling (pagination) SQL**
  - Retrieve next n rows
    - the same value for the first key column and with values higher than the last returned value for the second key column
    - OR, a value for the first key column that is higher than the last returned value for the first key col.
  - Common in COBOL/CICS and any screen scrolling application
- **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.
- DB2 V9 can’t use OR predicates as matching predicates with single index access.
  - Multi-index access (index ORing) used. It retrieves all RIDs that qualify from each OR condition and then unions the result.
- **DB2 10 can process these types of queries with a single index access**

Simple scrolling – Index matching and ORDER BY

- Scroll forward to obtain the next 50 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
  - Range-list index scan (one matching index scan+list prefetch)
- DB2 9 requires: Multi-index access, list prefetch and sort
  - Scans the index multiple times
- **Performance:**
  - 62% reduction in elapsed tim, 61% reduction in CPU time.
  - 99% reduction in the number of index getpages and no work file required

QBlockno	Planno	Accessname	Access_Type	Matchcols	Mixopseq
1	1	IX1	NR	1	1
1	1	IX1	NR	2	2

New access type (NR = IN-List Range)

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:
  - Converts this OR predicate into a range list with two ranges.
    - There will be 2 index probes, the 1st one is (LASTNAME='JONES' AND FIRSTNAME='WENDY'), the 2nd one is (LASTNAME='SMITH' AND FIRSTNAME='JOHN').
    - Both index probes have two matching columns on the index.

View/Table Expression Merge

- Physical materialization is an overhead. In prior versions of DB2, qualifying rows from views and table expressions frequently had to be materialized into a work file for additional processing by a query:
  - Can limit the join sequence considered.
  - Can limit the ability to apply predicates early
  - Cannot create indexes on materialized work files
- Generally merge is preferred over materialization
- ***DB2 10 provides enhancements to avoid materialization to a work file for views and table expressions, especially when they are involved in outer joins***.

Correlated table expression

Table expression is materialized into a work file in DB2 9 while, in DB2 10, no materialization is necessary

```
SELECT *
FROM T1,
TABLE(
SELECT T1.C2 FROM T1 AS T2
WHERE T1.C1 = T2.C1
) AS X;
```

The aggressive merge of the correlated table expression results in a **59% reduction in elapsed time, a 49% reduction in CPU time and a 100% reduction in work file getpages.**

DB2 V9

```
-------------------------------------------------------------
| QUERYNO | QB# | PL# | JT | AT | TNAME | CORNM |
+-----------------------------------------------------------
1_| 100 | 1 | 1 | | R | T1 | ? |
2_| 100 | 1 | 2 | | R | X | ? |
3_| 100 | 2 | 1 | | I | T1 | T2 |
+-------------------------------------------------------------
+-----------------------------------------------------
| ACCESSNAME | TT | T# | METH | MATC | MJC | PF |
-----------------------------------------------------
| | T | 1 | 0 | 0 | | S |
| | W | 3 | 1 | 0 | | S |
| X1 | T | 2 | 0 | 1 | | |
```

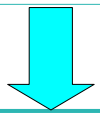
Expression materialization

DB2 10

```
-------------------------------------------------------------
| QUERYNO | QB# | PL# | JT | AT | TNAME | CORNM |
+-----------------------------------------------------------
1_| 100 | 1 | 1 | | I | T1 | ? |
2_| 100 | 1 | 2 | | I | T1 | T2 |
+-------------------------------------------------------------
+-----------------------------------------------------
| ACCESSNAME | TT | T# | METH | MATC | MJC | PF |
-----------------------------------------------------
| X1 | T | 1 | 0 | 0 | | |
| X1 | T | 2 | 1 | 1 | | |
```

Correlated table expression merge query rewritten

```
SELECT *
FROM T1,
        TABLE(
                SELECT * from T3 AS T2
                WHERE T1.C1= T2.C1
              ) AS X;        <-- table expression X is materialized in V9
```



```
SELECT T1.*, T2.C2          Not materialized in V10
FROM T1, T3 AS T2           Query rewritten
WHERE T1.C1 = T2.C2;
```


Safe query optimization: Minimizing Optimizer Challenges

- Potential causes of sub-optimal plans
  - Insufficient statistics
  - Unknown literal values used for host variables or parameter markers
- **Safe Optimization** has the goal to generate an access plan that has the lowest risk associated with it, within the range of access paths that are considered close to being the lowest cost.

  - 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

Minimizing impact of RID failure

- RID access, including List prefetch and Hybrid Join, becomes a performance challenge when the RID-pool overflows
- 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 and it loses all index filtering
  - to prevent RID pool overflow, query optimization has introduced threshold checking logic at bindtime.
- **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

Parallelism enhancements: Removal of restrictions

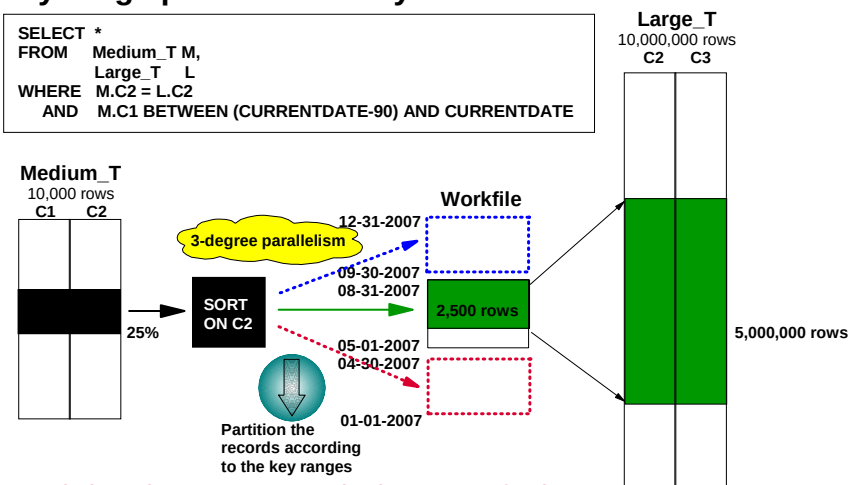
- **Support parallelism for multi-row fetch**
  - **In previous releases**
    - parallelism is disabled for the last parallel group in the top level query block
      - if there is no more table to join after the parallel group
      - and there is no GROUP BY clause or ORDER BY clause
  - Example:- SELECT * FROM CUSTOMER if multi row fetch is used
    - 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 forces customer to choose between multi-row fetch and parallelism.
- DB2 10 removes the restriction if the CURSOR is DECLARED as READ ONLY
  - Ambiguous Cursors will not have the restriction removed
- **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 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?
    - The uneven distribution of the workload could result in elongated elapsed times and/or
      - More of the parallel tasks processing considerably more data than the other tasks.
- **Record range partitioning differs from key range partitioning in that the partitioning is done at execution time instead of bind time and that partitioning is based on an equal number of records instead of based on keys.**
  - These inefficiencies no longer exist with record range partitioning:
    - Limited number of distinct values for leading columns
    - Data skew or data correlation
    - Lack of accurate statistics

Key range partition - Today

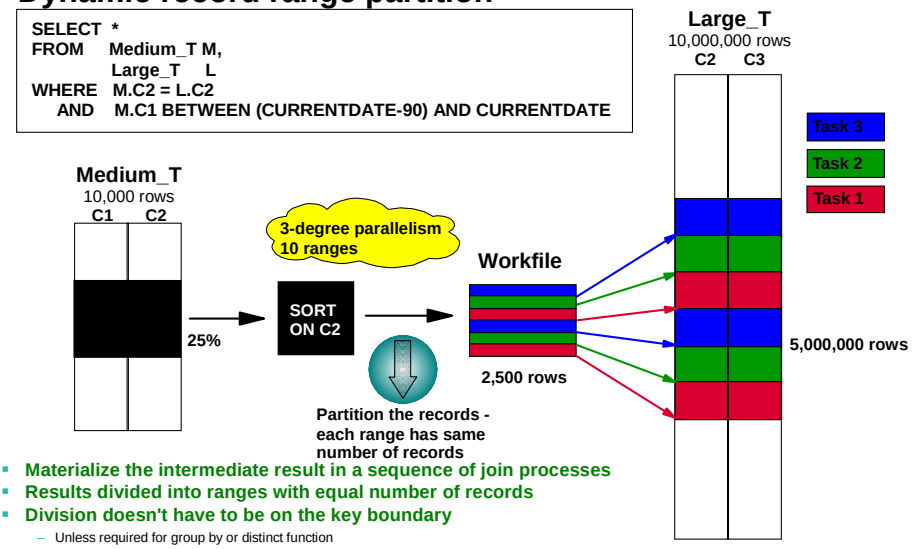
```
SELECT  *
FROM    Medium_T M,
        Large_T  L
WHERE   M.C2 = L.C2
  AND   M.C1 BETWEEN (CURRENTDATE-90) AND CURRENTDATE
```



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 :-(

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
```



- 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 number of workload elements

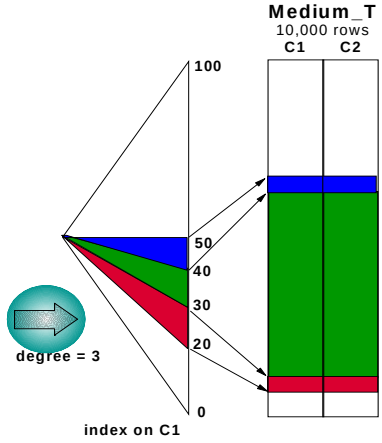
Parallelism Effectiveness - Straw Model

- 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

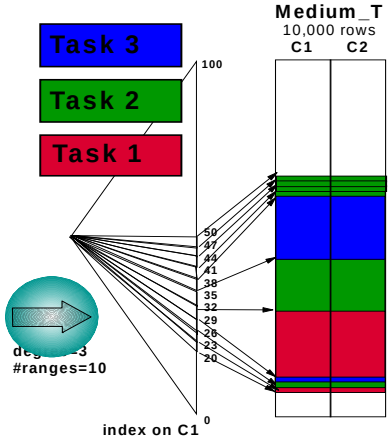
STRAW Model

- **Break up the data into a greater number of key ranges**
  - ✓ Executes a number of tasks equal to the degree of parallelism.
  - ✓ Each task is a smaller key range, so the elapsed time for each task is smaller.
  - ✓ Each task will continue on the next available range after it finishes the current one
- **The straw model provides perfm. benefits over record range part. for queries having a skewed dist. of keys**

```
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

Misc. query 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
  - DB2 10 **extends this to intermediate sorts also**
    - Except for parallelism or SET function
- **Hash support for large sorts**
  - Potential for reduction in number of merge passes
- **Index include columns**
- **Hash access path**

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 (Table with 1 unique index) will maintain original limitations
  - **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

NPGTHRSH indicates the tables for which DB2 favors matching index access. Values are:
-1: Favors matching index access for all tables.
0: Selects the access path based on cost, and no tables qualify for special handling. This is the default.
n>=1: Favor matching index access for tables for which NPAGES<=is less than n.

Agenda

- Access path management
  - ✓Dynamic Statement Cache Enhancements
  - ✓ Access Path Stability
  - ✓ Instance Based Statement Hints/Options
- Query performance improvements
  - ✓ Predicate application
    - IN-list
    - Complex ORs
    - View/Table expr Merge
  - ✓Safe Query Optimization
  - ✓ Parallelism Enhancements
- ✓Runstats usability and performance improvements
  - ✓ Auto-Stats
  - ✓ RUNSTATS Simplification/Performance

Autonomic RUNSTATS

- Collecting stats is a difficult and time consuming manual process
  - Need to look at the queries to figure out what stats are needed
  - Need to repeatedly look at the RTS tables to figure out when to recollect
- Inadequate stats collection leads to poor or inconsistent query performance
- Solution is to automate the process
  - More efficient, accurate and stable
- **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

RUNSTATS Simplification/Performance Overview

- Autonomics: RUNSTATS options to SET/UPDATE/USE a stats profile
  - Integrate specialized statistics into generic RUNSTATS job
  - Stored in catalog table SYSIBM.SYSTABLES_PROFILES
    - RUNSTATS … TABLE tbl COLUMN(C1)… **SET PROFILE**
    - RUNSTATS … TABLE tbl COLUMN(C5)… **UPDATE PROFILE**
    - RUNSTATS … TABLE tbl **USE PROFILE**

- New option for page-level sampling: TABLESAMPLE
  - SAMPLE worked at the row level, TABLESAMPLE works at page level.
  - The **TABLESAMPLE SYSTEM** sampling applied to a table means that ***each page of the table is included in the sample with probability P/100***.
    - For each page that is included, all rows on that page qualify for the sample.
  - **AUTO option: RUNSTATS will determine a sampling rate based on the size of the table**, the larger the table the smaller is the sampling rate.
  - **REPEATABLE option**: ensures that repeated executions of RUNSTATS return the same sample.
    - RUNSTATS … TABLE tbl **TABLESAMPLE SYSTEM AUTO**

धन्यवाद
Hindi

多謝
Traditional Chinese

ขอบคุณ
Thai

多谢
Simplified Chinese

Спасибо
Russian

Thank You
English

Gracias!
Spanish

شكراً
Arabic

Obrigado
Brazilian Portuguese

Merci
French

Bedankt
Nederlands

Danke
German

ありがとうございました
Japanese

நன்றி
Tamil

감사합니다
Korean

Carlos Guardia
cguardia@es.ibm.com