

SHARE

Technology - Connections - Results

Maximizing the DB2 9 for z/OS Optimizer and SQL Performance Enhancements

James Guo
IBM Silicon Valley Lab

August 26, 2009
Session Number 6126



maxi
ove agility save tim
enges colleagues

Disclaimer



© Copyright IBM Corporation [current year]. All rights reserved.

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

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.

IBM, the IBM logo, ibm.com, DB2, and z/OS are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Agenda



- Global Query Optimization
- Generalized sparse index and in-memory data cache
- Page Range Processing Enhancements
- Histogram Statistics
- REOPT AUTO
- Dynamic Index ANDing for Star Schema
- Plan Stability
- Sort Enhancements
- Indexing Enhancements
- Other Optimization Enhancements

Global Query 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

* Assumes subquery is not transformed to join

- **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)
```

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

* Assumes subquery is not transformed to join

Virtual Tables



- A new way to internally represent subqueries
 - Represented as a Virtual table
 - Allows subquery to be considered in different join sequences
 - May or may not represent a workfile
 - Apply only to subqueries that cannot be transformed to joins
 - In V9, it is less important whether the query is written as a correlated subquery or as a non-correlated subquery



EXPLAIN Output

- Additional row for materialized “Virtual Table”
 - Table type is "W" for "Workfile".
 - Name includes an indicator of the subquery QB number
 - *Example* → “DSNWF(02)”
 - Non-materialized Virtual tables will not be shown in EXPLAIN output.
- Additional column PARENT_PLANNO
 - Used with PARENT_QBLOCKNO to connect child QB to parent
 - V8 only contains PARENT_QBNO
 - Not possible to distinguish which plan step the child tasks belong to.

Generalizing Sparse Index and In-Memory Data Cache



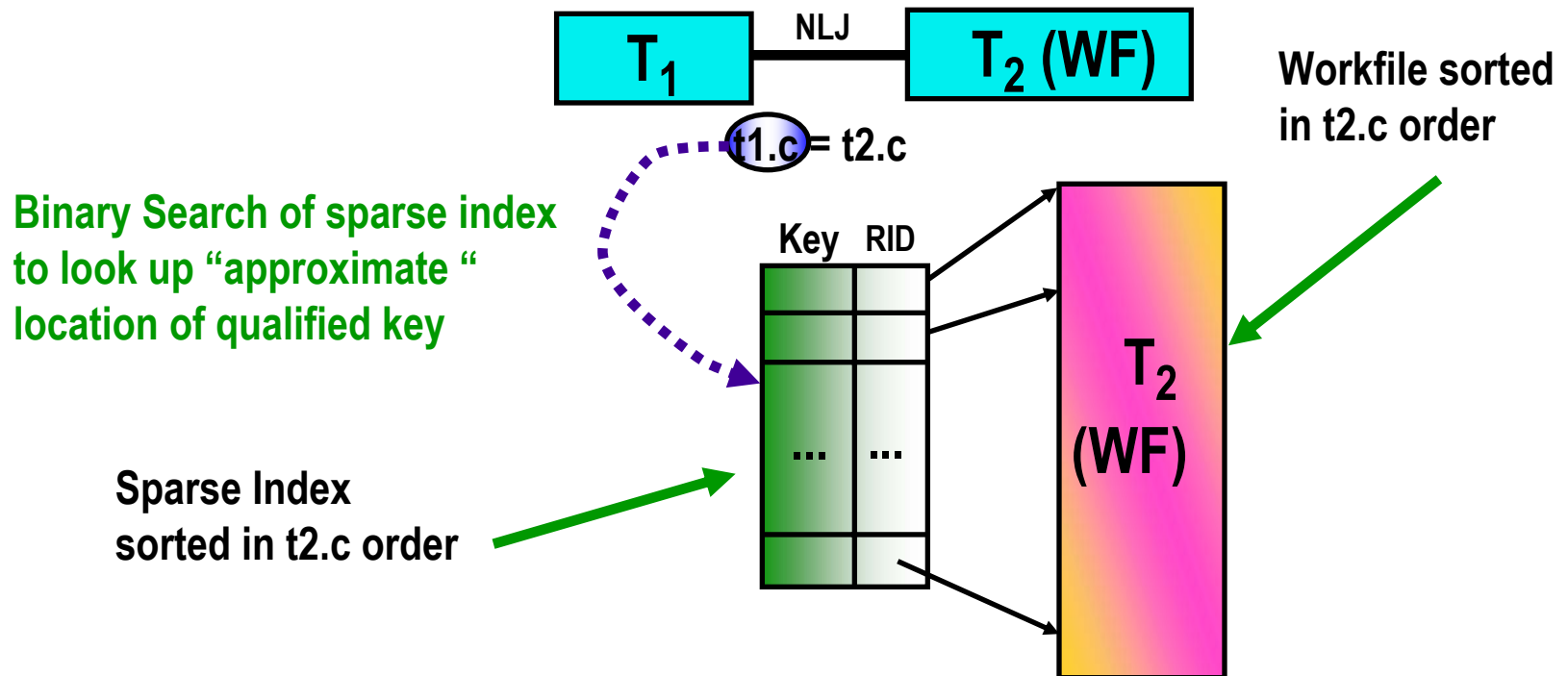


Pre-V9 Sparse Index & in-memory Data Cache

- V4 introduced sparse index
 - for non-correlated subquery workfiles
- V7 extended sparse index
 - for the materialized work files within star join
- V8 replaced sparse index
 - with in-memory data caching for star join
 - Runtime fallback to sparse index when memory is insufficient

How does Sparse Index work?

- Sparse index may be a subset of workfile (WF)
 - Example, WF may have 10,000 entries
 - Sparse index may have enough space (240K) for 1,000 entries
 - Sparse index is “binary searched” to find target location of search key
 - At most 10 WF entries are scanned



Extending Data Caching in DB2 9



- In-memory data caching is extended to non-star join
- V9 will use a local pool above the bar
 - Instead of a global pool used in V8 star join
 - Data caching storage management will be associated with each thread
 - Which can reduce the potential storage contention
- New ZPARM **MXDTCACH**
 - specifies the maximum extent in MB, for data caching per thread.



Benefit of Data Caching

- All tables lacking an index on join column(s):
 - Temporary tables
 - Subqueries converted to joins
 -any table

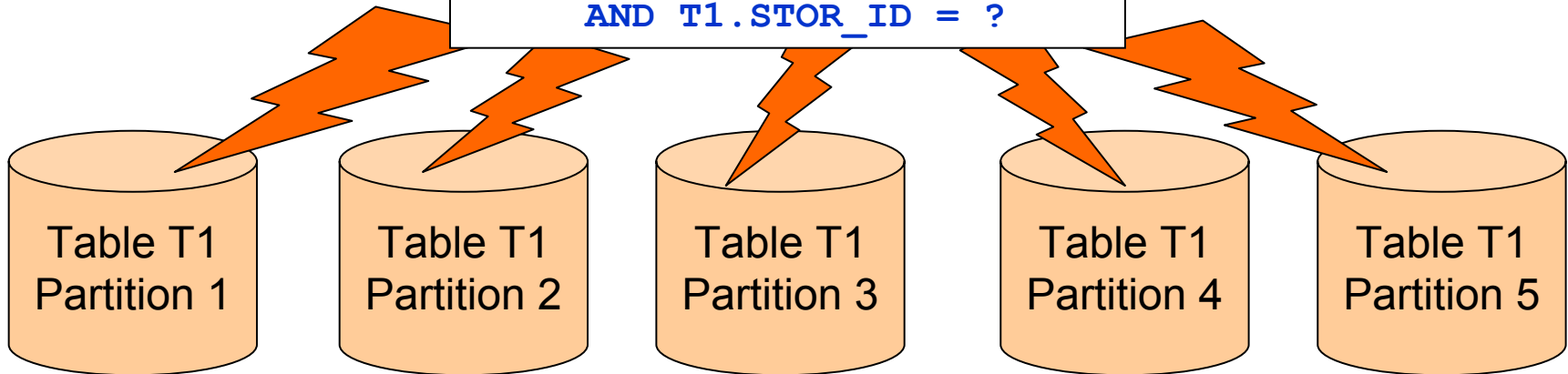
- V9 also supports multi-column sparse index

Page Range Processing Enhancements



Limiting the Partitions Accessed

```
SELECT SUM(GROSS_SALES)
FROM T1
WHERE T1.MONTH = ?
AND T1.STOR_ID = ?
```

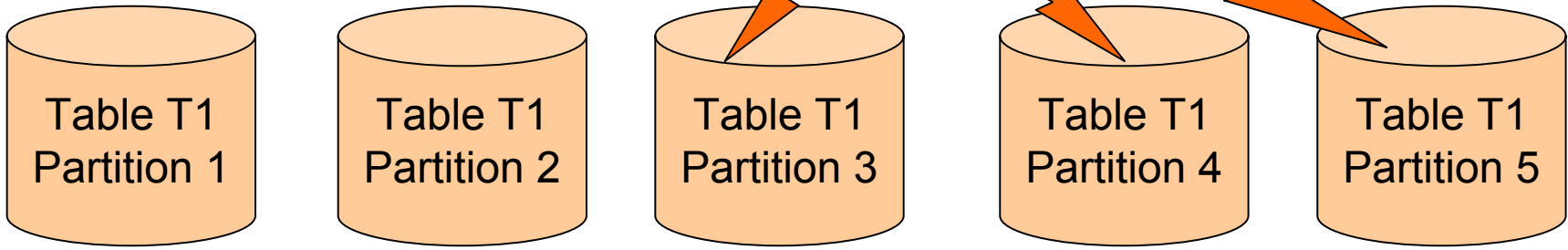


- With DPSIs or tablespace scan of partitioned tablespace
 - beneficial to avoid accessing partitions with no qualifying rows
- Done using page range screening,
 - V8 support for local predicates on the leading partitioning key(s)
- **Reduces qualified rows read without indexing**



Page Range Screening Enhancements

```
SELECT SUM(GROSS_SALES)
FROM T1
WHERE T1.MONTH = ?
AND T1.STOR_ID = ?
```



- DB2 9 introduces two page range screening enhancements:
 - Join predicates
 - Non-matching predicates

Page Range Screening Using Join Predicates

- Using join predicates for page range screening:

```
SELECT * FROM T1, T2
WHERE T1.TRANS_MONTH = T2.TRANS_MONTH
AND T1.CUSTNUM = T2.CUSTNUM
```

Non-Indexed partition key

DPSI key

Make the following assumptions about this query:

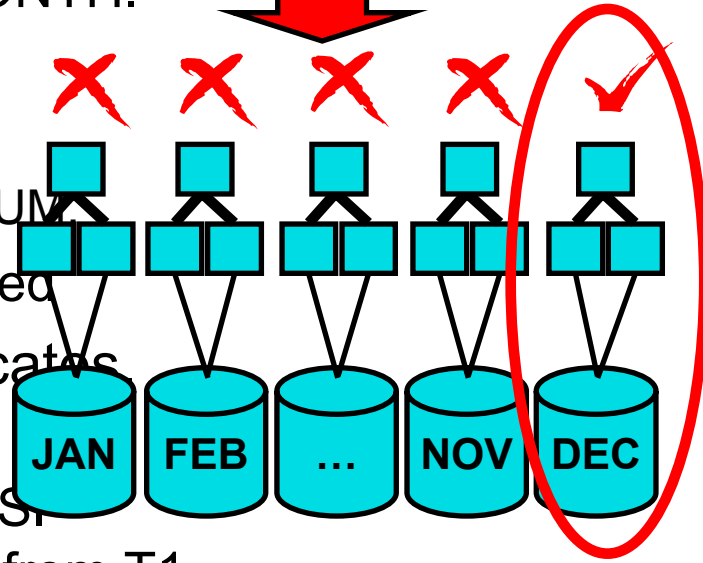
- T1 and T2 are both partitioned on TRANS_MONTH.
- There is a **DPSI** index on T2.CUSTNUM.
- Access path for the query NLJ of T1 to T2, T2 accessed via the DPSI index on T2.CUSTNUM.



Before this change, all parts of the DPSI accessed

Page range screening only for **local** predicates

Now, with the join predicate being used for page range screening, only 1 partition of the DPSI index on T2 needs to be accessed for each row from T1.



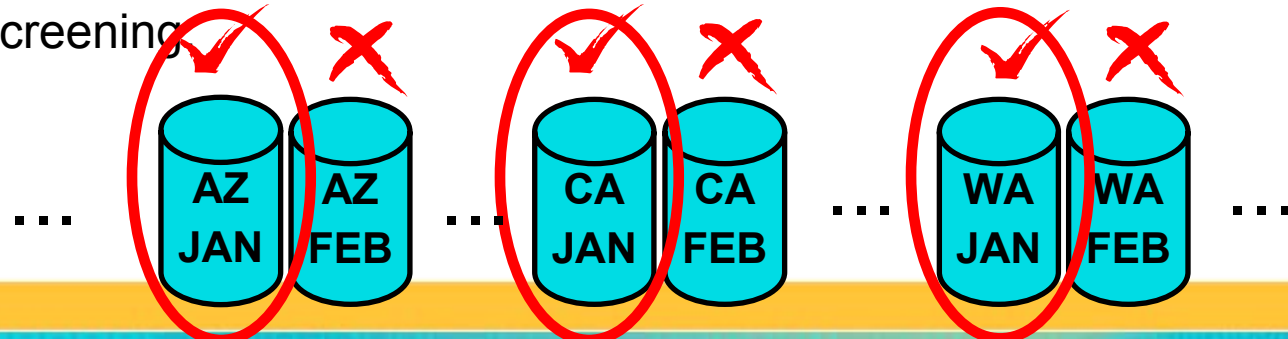
Page Range Screening Using Non-matching Predicates

- Using non-matching predicates for page range screening:

```
SELECT SUM(GROSS_SALES) FROM T1
WHERE T1.MONTH = JAN
      AND T1.STOR_ID = ?
```

← 2nd partition key

- Make the following assumptions about this query:
 - T1 is partitioned on **STATE**, **MONTH**.
 - 50 states x 12 months = 600 partitions.
 - The access path for the query is a matching index scan using DPSI index on T1.STOR_ID
- Before this change, all 600 parts of the DPSI index are accessed.
- Now, as only one month is of interest, the number of partitions needing to be accessed is reduced to 50 instead of 600 when the predicate on T1.MONTH is used by page range screening



Histogram Statistics





RUNSTATS Histogram Statistics

- 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
- “frequencies” for high cardinality columns



Example 1

Sparse and **dense** ranges

SALES_DATE BETWEEN '2006-12-10' AND
'2006-12-24'

returns significantly more rows than a 2 week
range in March

Query: SELECT * FROM T WHERE T.C1 between 'a *sparse range*'
SELECT * FROM T WHERE T.C1 between 'a *dense range*'

Example 2

Example#2: When gaps exist in ranges

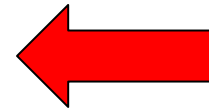
SAP uses **INTEGER** (or **VARCHAR**) to store **YEAR-MONTH** data.

There are 12 values in 200501~200512, but zero value in 200513~200600.

Query: SELECT * FROM T WHERE T.C1 between 'a *skipped range*'

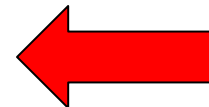
SELECT * FROM T WHERE T.C1 between 'a *non-skipped range*'

Query: T.C1 between 200512 and 200601;



**90 valid numerics,
but only 2 valid dates**

T.C1 between 200501 and 200512;



**12 valid numerics,
and 12 valid dates**

REOPT Auto Based On Parameter Marker Change



REOPT enhancement for dynamic SQL



- V8 REOPT options
 - Dynamic SQL
 - REOPT(NONE, ONCE, ALWAYS)
 - Static SQL
 - REOPT(NONE, ALWAYS)
- V9 Addition for **Dynamic** SQL
 - Bind option REOPT(AUTO)

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



Two new columns in DSN_STATEMENT_CACHE_TABLE

Description of new columns in DSN_STATEMENT_CACHE_TABLE	
Column Name	Description
BIND_RA_TOT	Total number of rebinds that have been issued for the dynamic statement due to REOPT(AUTO).
BIND_RO_TYPE	'N' - REOPT(NONE) or its equivalent '1' - REOPT(ONCE) or its equivalent 'A' - REOPT(AUTO) 'O' - Current plan is deemed as optimal and no need for further REOPT(AUTO)

REOPT AUTO - Considerations

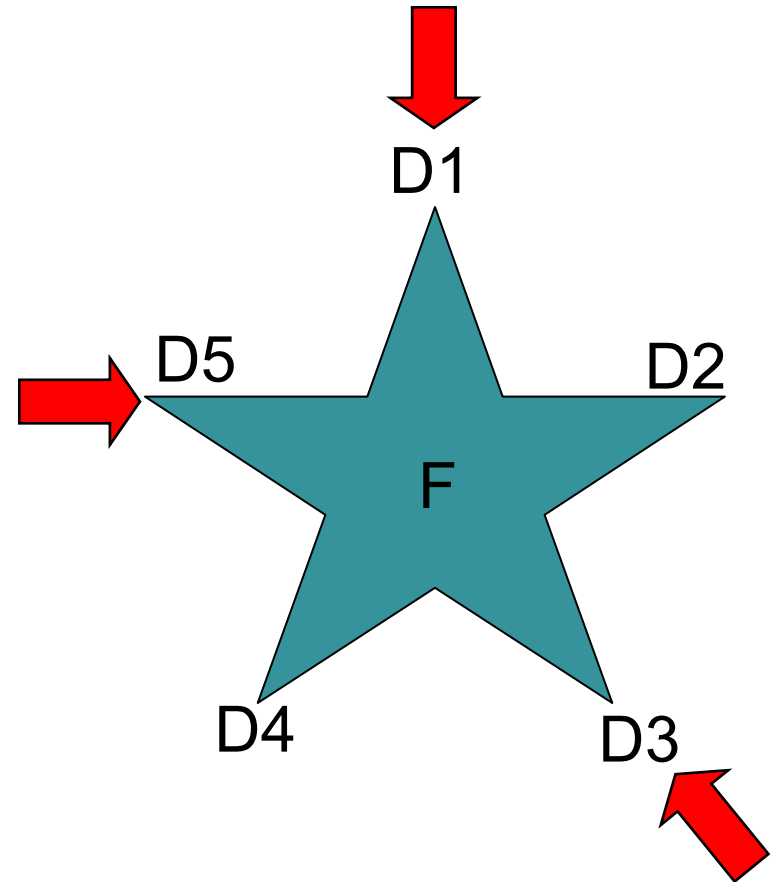
- Consider REOPT(AUTO) to achieve better balance between costs of **re-optimization** and costs of **processing** a statement
- You might use REOPT(AUTO) instead of ALWAYS or NONE for cached dynamic SQL when:
 - Execution time varies depending on parameter marker values
 - especially for non-uniform data that is joined to other tables
- For such SQL statements, performance gain from new access path might or might not be greater than performance cost of reoptimization at run time

Dynamic Index ANDing for Star Schema



Dynamic Index ANDing Challenge

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

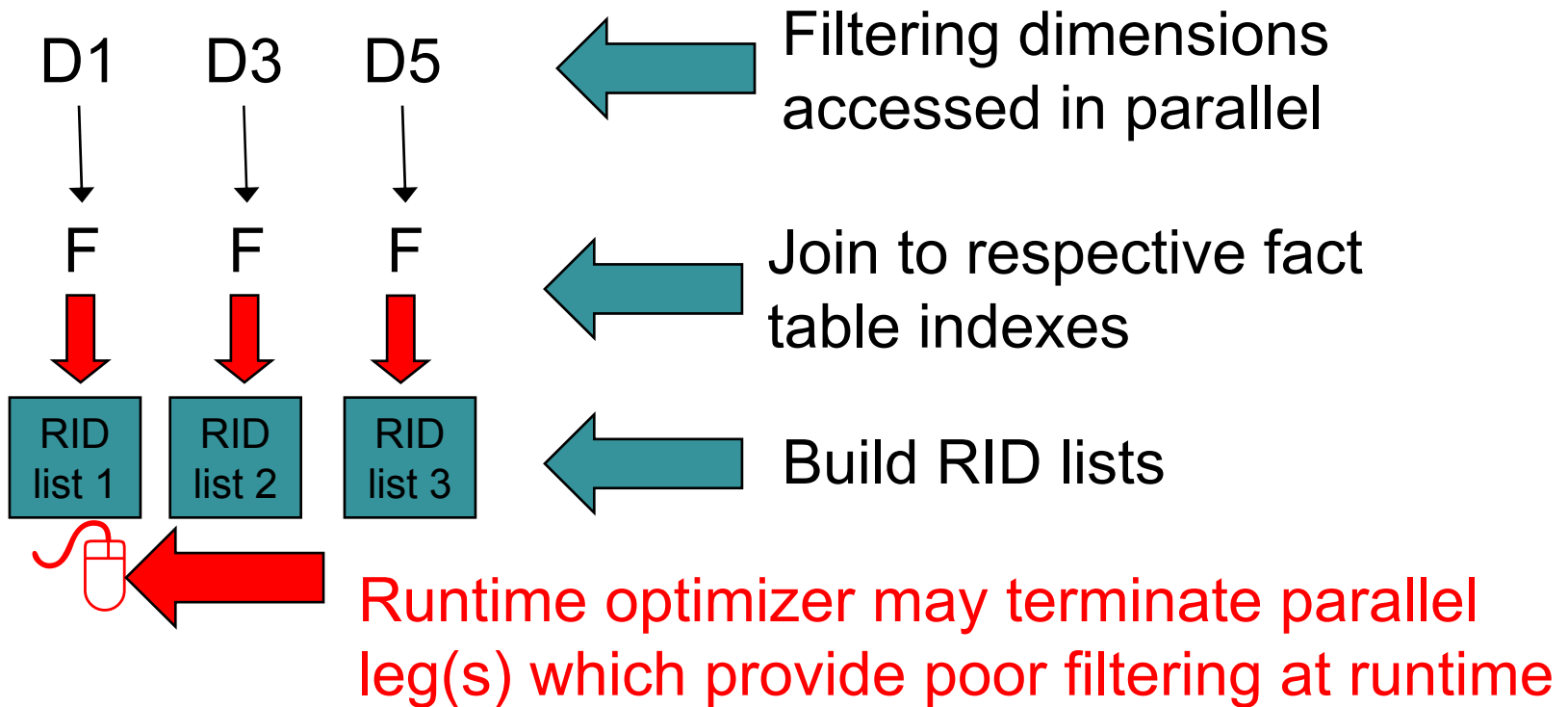


Pair-wise Join (PWJ) Solution

- Prior to V9, a well-tuned multi-column index design is key to good performance for star join workload
 - The lack of tuning skill and the cost of index make it difficult
- Access path decision at bind time may not be optimal and it can cause bad query performance
 - Need to have optimization at runtime to prevent disaster query
- Solution – Use dynamic index ANDing (aka PWJ) to join the fact table and the dimension table(s) via single column indexes.

Index ANDing – Pre-Fact

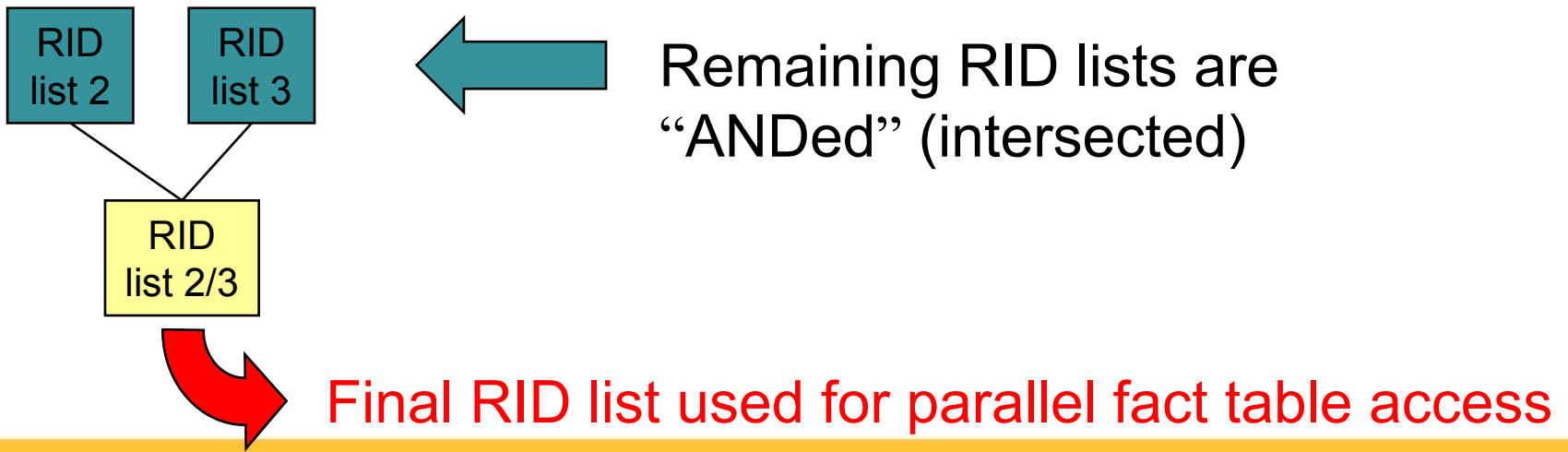
- Pre-fact table access
 - Filtering may not be (truly) known until runtime



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





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

More zIIP Offload for BW workload in V9



- For 100 star join query BW workloads
 - With **well-tuned index design (not typical)**
 - **37%** average offload in V8 and **62%** in V9
 - With **inadequate multi-column index support**
 - **16%** average offload in V8 and **78%** in V9
 - These figures are very sensitive to queries themselves as well as hardware environment, eg zIIP utilization

Key Messages



- V9 performs better than V8 for BW workloads with well-tuned index design
- V9 **outperforms** V8 for BW workloads with inadequate multi-column index support
- **Increased query parallelism resulting in greater zIIP off-load** in V9 for BW workload
- V9 improves the out-of-box solution which **reduces the burden of index design and query tuning** from users
- Overall, **a significant TCO reduction in V9** for BW customers

Plan Stability





Plan Stability Overview

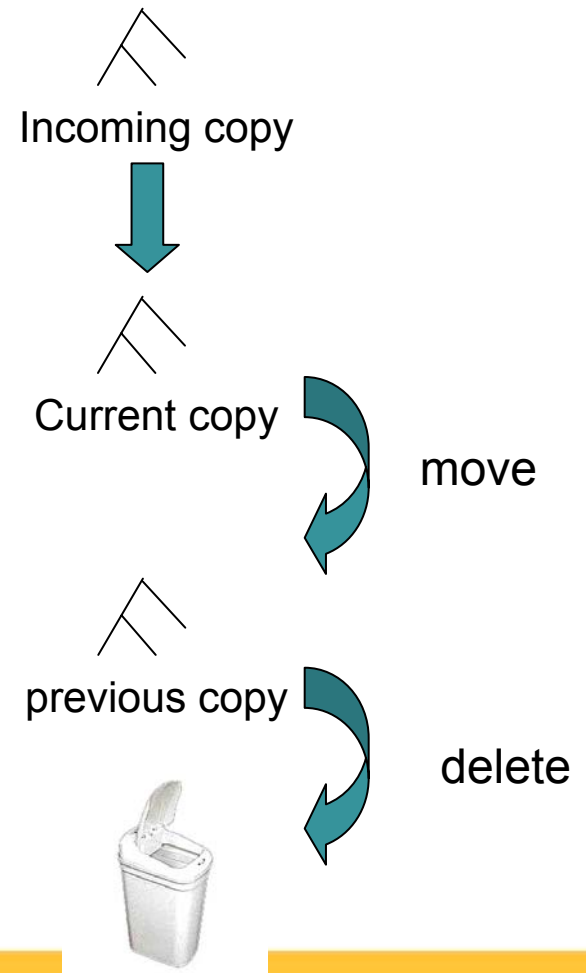
- Ability to backup your static SQL packages
- At REBIND
 - Save old copies of packages in Catalog/Directory
 - Switch back to previous or original version
- Two flavors
 - BASIC
 - 2 copies: Current and Previous
 - EXTENDED
 - 3 copies: Current, Previous, Original
 - Default controlled by a ZPARM
 -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



SHARE
Technology - Connections - Results

Plan Stability - BASIC support

REBIND ... PLANMGMT(BASIC)



REBIND ... SWITCH(PREVIOUS)

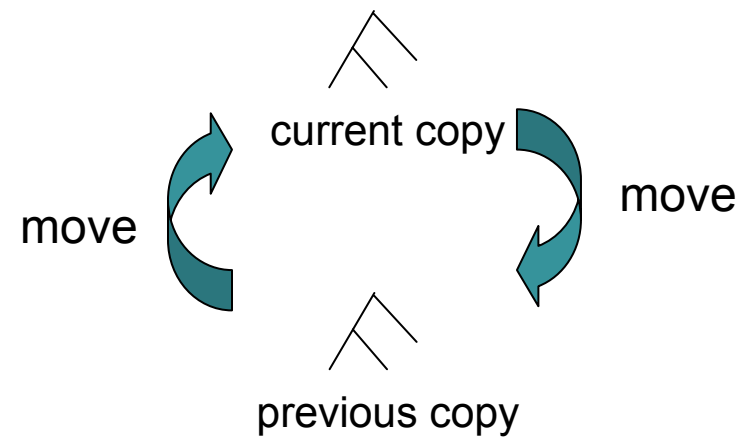
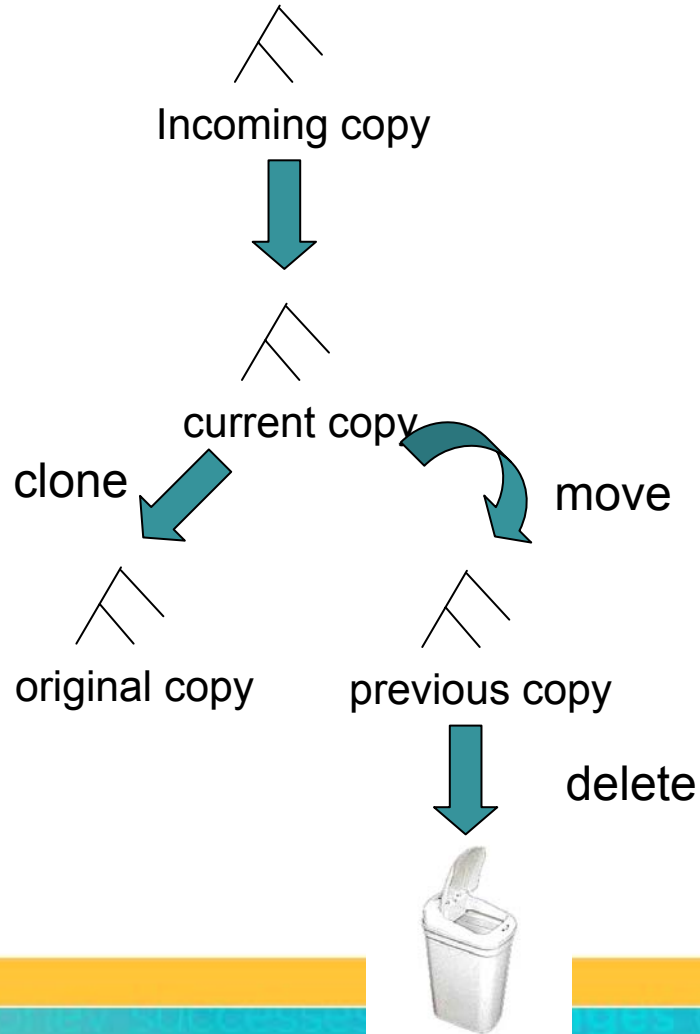


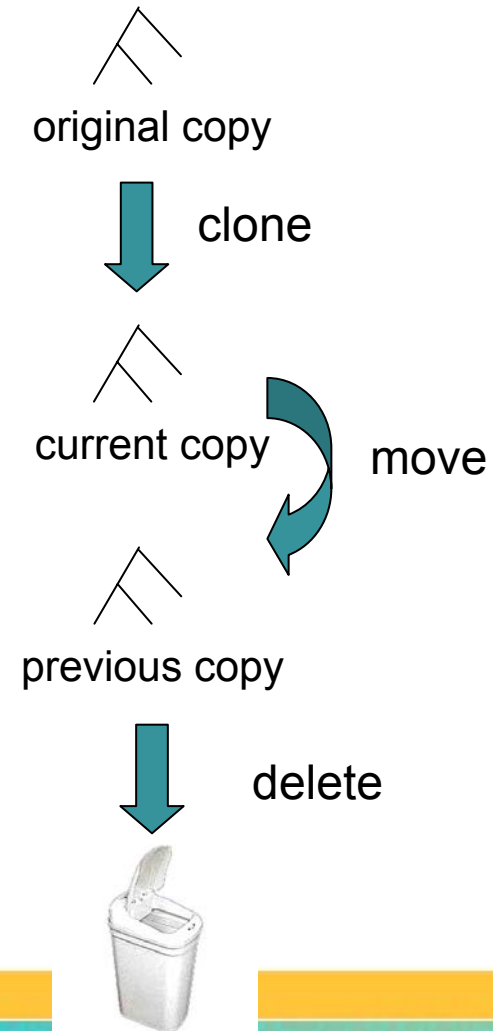
Chart is to be read from bottom to top

Plan Stability - EXTENDED support

REBIND ... PLANMGMT(EXTENDED)



REBIND ... SWITCH(ORIGINAL)



Sort Enhancements



Sort Performance Improvement



- Group By sort and Distinct sort (CM)
 - Group By sort improvement when no column function and no available index
 - Eg SELECT a1 FROM A GROUP BY a1
 - Distinct sort improvement when no index or only non unique index available
 - V9 - Can use non-unique index and avoid sort for distinct
 - Eg SELECT DISTINCT a1 FROM A
- Up to 2 times improvement observed in queries in-house

Sort Performance - continued

- Fetch First N Rows with Order By (CM)
 - Example: Get top 10 Americans in income tax paid
 - Avoid tournament tree sort for small N
 - Up to -50% cpu in one measurement test
 - Supported in Subselect also in V9 (NFM)
- In-memory work file for small sorts (CM)
 - 10 to 30% cpu reduction for short-running SQL calls with small sorts
 - Beneficial for online transaction with relatively short-running SQL calls in which the number of rows sorted can be small.

Sort Performance - continued



- Heavier use of 32K work file BP to help large work file record performance (CM)
 - V8 uses 4K BP if less than 4KB row
 - V9 uses 32K BP for more records to gain improved performance, especially I/O time
 - Less work file space and faster I/O, for example 15 2050byte records on one 32K page vs 8 records on 8 4K pages
 - Some measurement example
 - <10% difference for 50 and 85byte records as 4K BP continues to be used
 - 10-50% improvement for 95byte and bigger records because of 32K BP

Sort Performance - Recommendation



- For smaller records, 4K BP is better because of 255 rows per page limit
 - eg over 90% wasted space on 32K page for 10byte records
- Recommendation
 - Assign bigger 32K work file BP
 - Allocate bigger/more 32K work file datasets
 - If 4K work file BP activity is significantly less, corresponding BP size and work file datasets can be reduced.
- New statistics on how often more optimal 32K workfiles ran out and 4K workfiles used instead, or vice versa

Indexing Enhancements



Index on Expression



- DB2 9 supports “index on expression” (**NFM**)
 - Can turn a stage 2 predicate into indexable
- Example: Create Index ABC on Employee(salary+bonus, bonus*100/salary)
 - Orders of magnitude improvement if a predicate using such an index
 - Not eligible for zIIP offload as index expression evaluation done at load or unload rather than build index phase

Tracking Index Usage



- Tracking via real time stats
SYSINDEXSPACESTATS.LASTUSED(NFM)
- Indicates when index used in SELECT/FETCH, searched UPDATE/DELETE, and Referential Integrity check
- Useful in getting rid of unnecessary indexes
- PK44579 8/07 to support the use of index in Referential Integrity, Rid list processing, set functions, and XML values index

Data Partitioned Secondary Index (DPSI) Improvement



- Enhanced page range screening
 - To avoid having to visit all partitions when DPSI is used wherever possible
- More parallelism with DPSI
 - Parallel support for DPSI when DPSI is used as index access to data and ordering is expected
 - Example: `SELECT * FROM T1 ORDER BY C1` with DPSI on C1
- More index look-aside
- Index-only access with DPSI and ORDER BY
- Unique DPSI support when DPSI columns are superset of partitioning columns
 - E.g. partitioning key on c1.c2 and DPSI key on c1.c2.c3

Other Optimization Enhancements





Clusterratio Enhancement

- New Clusterratio formula in DB2 9
 - Better awareness of prefetch range
 - More accurate CR for lower cardinality indexes
- DB2 9 adds new statistic collected by RUNSTATS
 - DATAREPEATFACTOR differentiates density and sequential

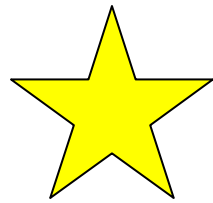


Dense (and sequential)



Sequential (not dense)

- Recommend RUNSTATS before mass REBIND in DB2 9



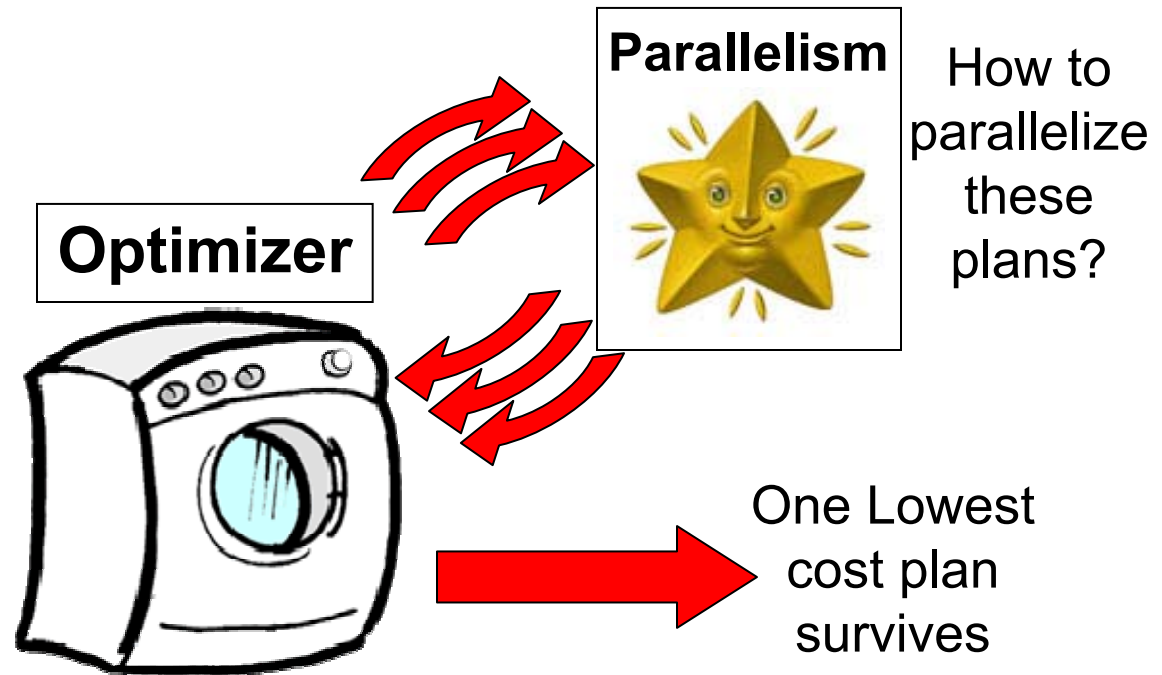
Sequential Access



- Sequential prefetch only used for tablespace scan in V9
 - Dynamic prefetch used instead for other access paths
 - Dynamic prefetch tracks sequential access at runtime
 - Sequential prefetch is based upon bind/prepare prediction
 - *At runtime, data may not be page sequential*
- PK70398 Sequential detection enhancements
 - Improved tracking for the clustered range

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 Parallelism Enhancements

- In V8
 - Degree cut on leading table (exception star join)
- In DB2 9
 - Degree can cut on **non-leading** table
 - Benefit for leading workfile, 1-row table etc.
 - Histogram statistics exploited for more even distribution
 - For index access with NPI
 - CPU bound query degree \leq # of CPUs * 4
 - \leq # of CPUs in V8



ORDER BY & FETCH FIRST in subqueries

- ORDER BY can be wrapped inside additional SQL
- ORDER BY and FETCH FIRST n ROWS ONLY in *subselect / fullselect*
 - ability to select the top *n* rows

```
(SELECT * FROM T1 ORDER BY C1)  
UNION ALL  
(SELECT * FROM T2 ORDER BY C2)
```

```
SELECT EMP_ACT.EMPNO, PROJNO  
FROM EMP_ACT  
WHERE EMP_ACT.EMPNO IN  
    (SELECT EMPLOYEE.EMPNO  
     FROM EMPLOYEE  
     ORDER BY SALARY DESC  
     FETCH FIRST 3 ROWS ONLY )
```

Merge



- MERGE
 - A new SQL DML statement in zOS V9
 - Combine Update and Insert operations into one statement
- SELECT FROM MERGE
 - a SELECT statement
 - Show updated/inserted rows
 - Including DB2 generated values
- SELECT FROM UPDATE/DELETE
 - V8 has SELECT FROM INSERT