Redbooks

Performance Enhancements

guy_delaporte@fr.ibm.com

List of Topics

Materialized query tables

Indexing enhancements

Stage 1 and indexable predicates

Table UDF cardinality option and block fetch

Trigger enhancements

Distribution statistics on non-indexed columns

Cost-based parallel sort for single and multiple tables

Performance of multi-row operations

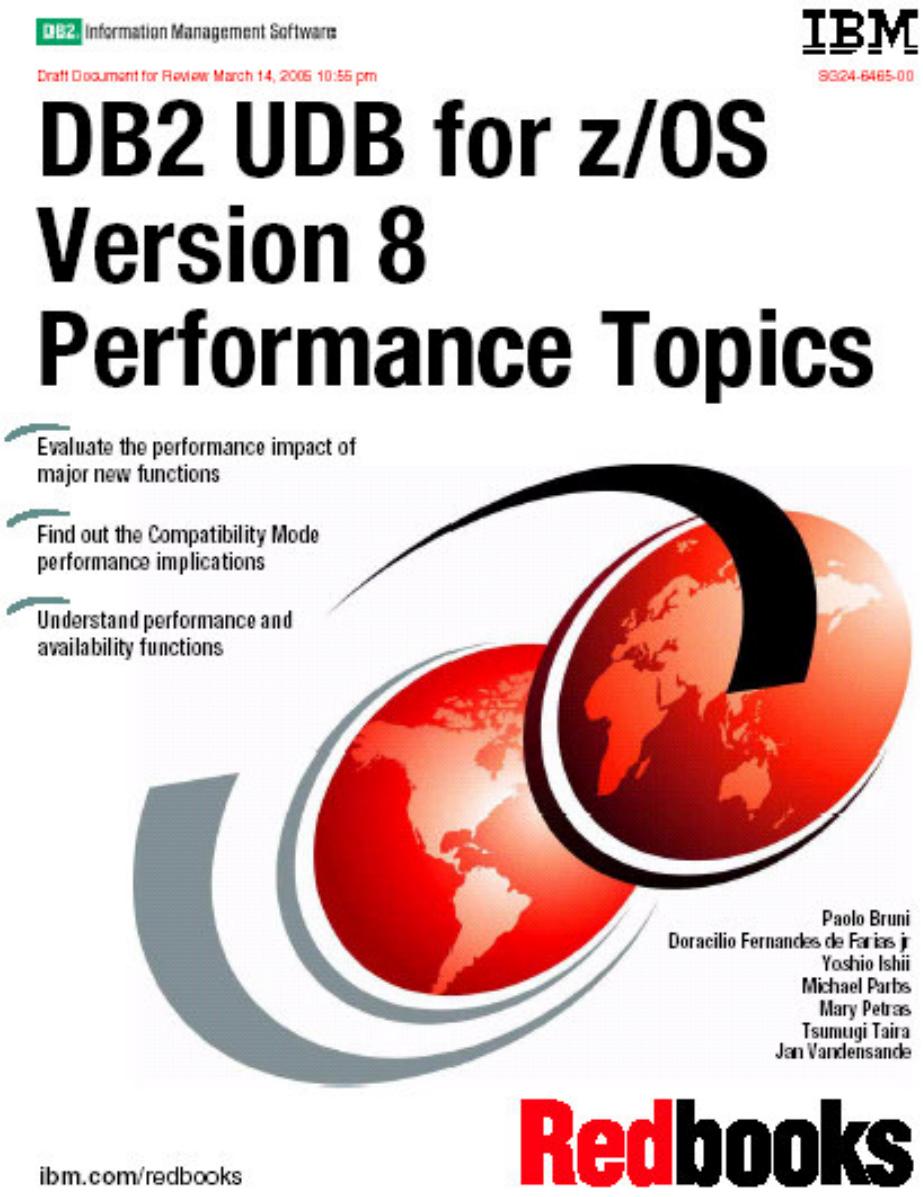
Volatile table support

Data caching and sparse index for star join

Miscellaneous performance enhancements

Visual Explain enhancements

Redbook sur les performances V8 : SG24-6465



Data Warehousing Issues

To improve query performance especially for Data Warehousing

Summary tables are often created manually for users

- To improve the response time
- To avoid redundant work of scanning, aggregation and joins of the detailed base tables (e.g. history)
- To simplify SQL to be coded

User needs to be aware of summary tables and know whether to use them or the base tables depending on the query.

What is a Materialized Query Table (MQT)?

Table containing materialized data derived from one or more source tables specified by a fullselect

- Source tables can be base tables, views, table expressions or user-defined table expression
- MQTs can be accessed directly via SQL or
- Chosen by the optimizer (through **automatic query rewrite**) when a base table or view is referenced

Two types: Maintained by system or by user

Synchronization between base table(s) and MQT

- Using 'REFRESH TABLE' statement
- Batch update, triggers, etc. for user-maintained MQTs

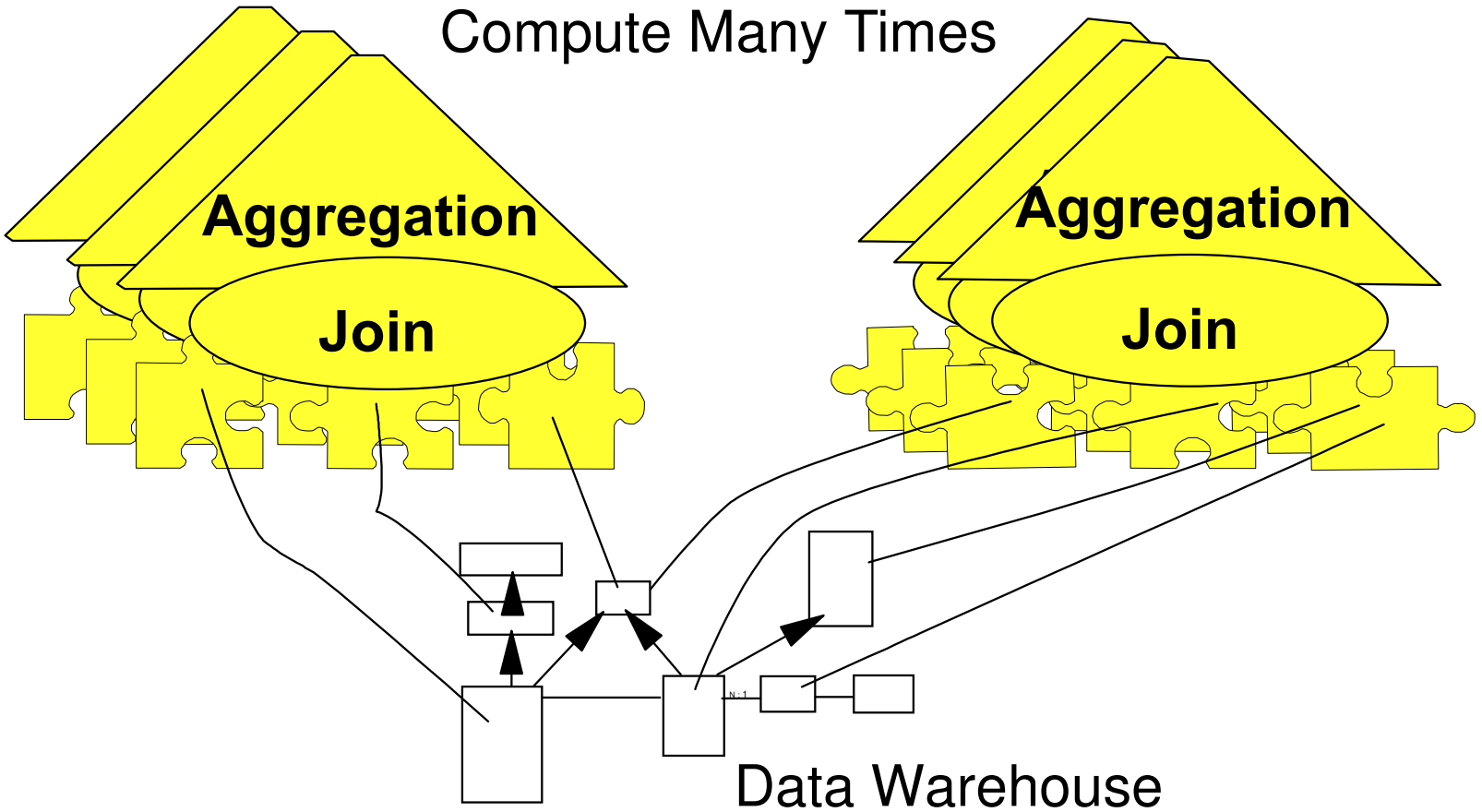
Can provide significant query performance improvement

Without MQT Each Query Re-Computes!

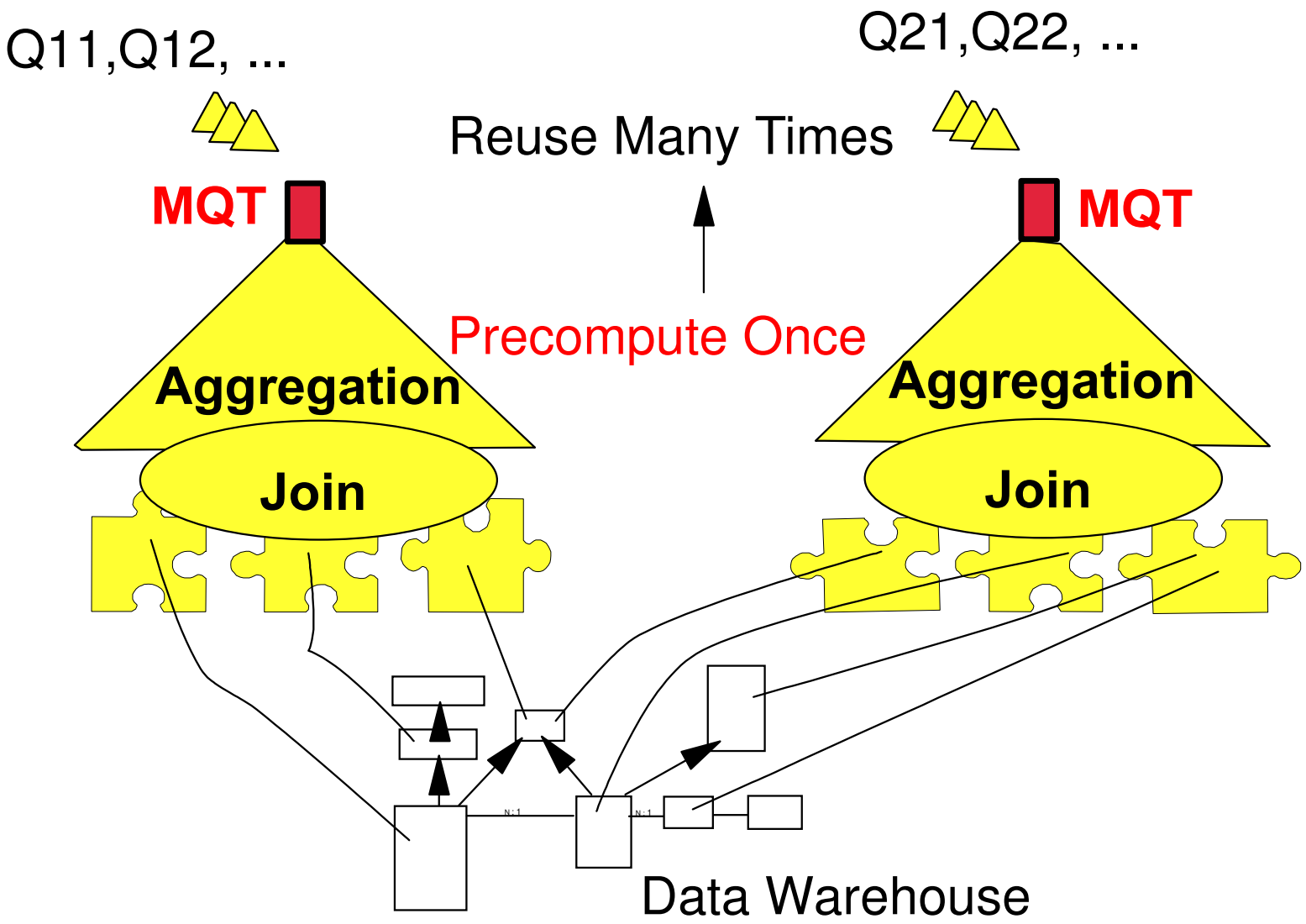
Redbooks

Q11,Q12, ...

Q21,Q22, ...



With MQT Avoid Redundant Computation



Creating an MQT - Example


```
CREATE TABLE MQT1 AS (
 SELECT T.PDATE, T.TRANSID,
        SUM(QTY * PRICE) AS TOTVAL,
        COUNT(QTY * PRICE) AS CNT
 FROM SCNDSTAR.TRANSITEM TI, SCNDSTAR.TRANS T
 WHERE TI.TRANSID = T.TRANSID
 GROUP BY T.PDATE, T.TRANSID)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY SYSTEM
ENABLE QUERY OPTIMIZATION
IN MYDBMQT.MYTSMQT;
```

Populating and Refreshing a MQT

Ensure data currency meets user requirements to avoid out-dated results

Issue REFRESH TABLE statement periodically

- Deletes all rows in the MQT
- Executes the fullselect as defined in the original MQT definition
- Inserts calculated data into the MQT
- Updates the catalog with refresh timestamp and cardinality information
- Executes in a single UOW
- Consider logging and performance impact

For user-maintained MQTs, other population methods can also be used, for example, LOAD, Inserts, Updates, DPROP Apply

The Basics of Automatic Query Rewrite

State of the MQT (that is, must be ready for query optimization and not locked)

Whether the query result can be derived from or directly use the MQT

- There are no predicates in the MQT subselect that are not in the query
- Each column in the result table can be derived from one or more columns in the MQT
- The GROUP BY clauses are compatible

Estimated cost of the queries

- MQTs are not used for short-running queries
- The original and rewritten costs are compared and the lower cost chosen

Only considered for read-only queries

- Dynamically prepared queries only
- Not for static SQL or static SQL with REOPT(VARS)

Note that optimization is considered at the query block level

MQT Exploitation - Simple Example


```
QUERY:
SELECT T.PDATE, AVG (QTY * PRICE) AVGAMT
FROM SCNDSTAR.TRANSITEM TI, SCNDSTAR.TRANS T
WHERE TI.TRANSID = T.TRANSID AND
      T.PDATE >= '2001-01-01'
 GROUP BY T.PDATE;

AFTER QUERY REWRITE:
SELECT PDATE, CASE WHEN SUM(CNT) = 0 THEN NULL
                   ELSE SUM(TOTVAL)/SUM(CNT) END AVGAMT
FROM MQT1
WHERE PDATE >= '2001-01-01'
GROUP BY PDATE;
```

Query rewrite for join

```
CREATE TABLE MQT201 (CRE,INT,COUNT) AS(
SELECT CRE,INT,COUNT(*)
FROM CLA,COI,CIA
WHERE COI.POL=CIA.POL AND
      COI.CVI=CIA.CVI AND
      COI.CLI=CIA.CLI AND
      CIA.CLI=CLA.CLI AND
      CIA.CAI=CLA.CAI
GROUP BY CRE,INT)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY USER
ENABLE QUERY OPTIMIZATION;
```

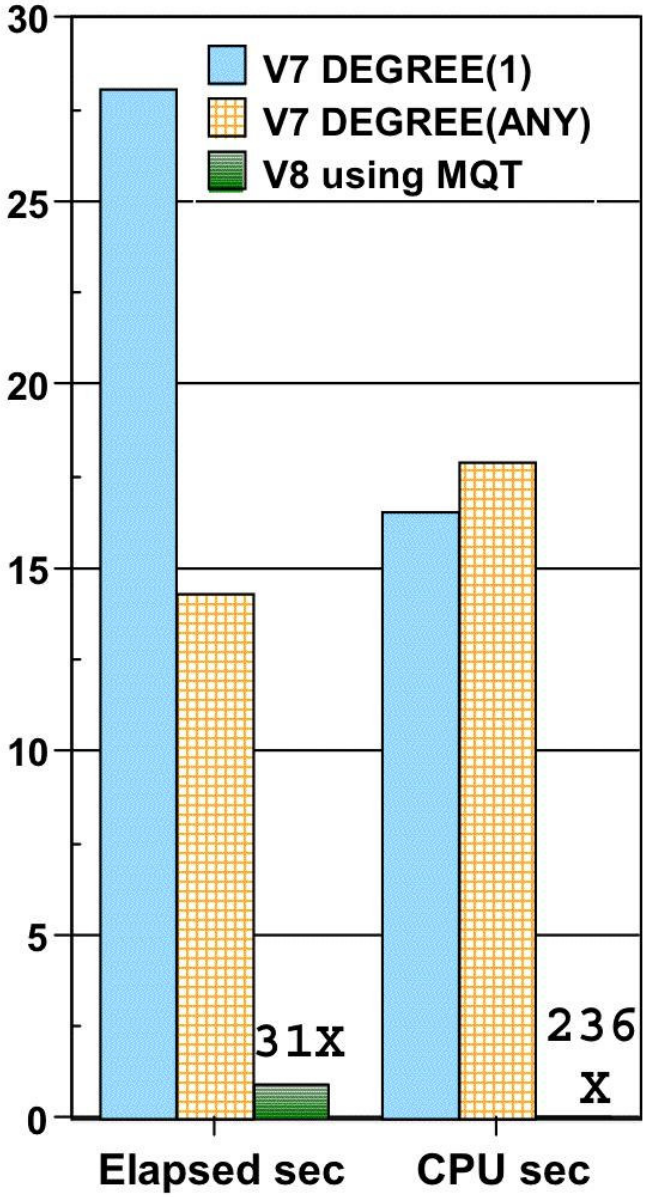
V7 PLAN_TABLE

TABLE SIZE	TNAME	TABLE_TYPE	METHOD
633 K	COI	T	0
1.5 M	CIA	T	1
1.3 M	CLA	T	1

V8 PLAN_TABLE

TNAME	TABLE_TYPE	METHOD
MQT201	M	0

171 rows in MQT



The Role of Constraints

Constraints are very important in determining whether a materialized query table can be used for a query

- Referential constraints (to allow extra lossless joins in MQT, or some dimensions missing in a query)
- Functional dependencies (primary keys and unique indexes, to allow more group-by columns in a query)

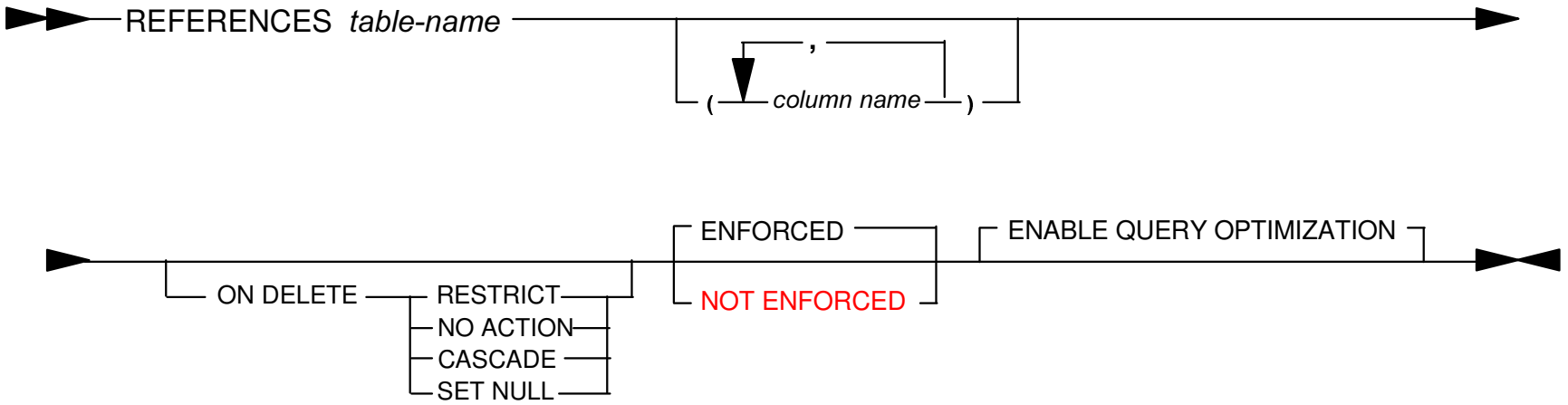
V8 allows informational referential constraints to be specified for base tables, and avoids the enforcement overhead (in a data warehouse environment)

Informational Constraints

Informational RI is not enforced by database manager and is ignored by most utilities

- Except LISTDEF RI, QUIESCE/REPORT TABLESPACESET

Informational RIs are always used by the optimizer in query rewrite



Informational RI Example


```
CREATE TABLE SCNDSTAR.TRANS
   (TRANSID   CHAR(10) NOT NULL PRIMARY KEY,
    ACCTID    CHAR(10) NOT NULL,
    PDATE     DATE NOT NULL,
    STATUS    VARCHAR(15),
    LOCID     CHAR(10) NOT NULL,
    CONSTRAINT ACCTTRAN  FOREIGN KEY (ACCTID)
       REFERENCES SCNDSTAR.ACCT NOT ENFORCED,
    CONSTRAINT LOC_ACCT FOREIGN KEY (LOCID)
       REFERENCES SCNDSTAR.LOC NOT ENFORCED
   )
  IN DBND0101.TLND0101;
```

MQT Administration and Considerations

Information about MQTs is stored in SYSIBM.SYSVIEWS

No primary keys, unique indexes or triggers are allowed on MQTs

All the MQTs and their indexes are dropped if their associated base tables are dropped

Specialized MQTs or generalized MQTs?

- Design issue as to whether to have a few generic MQTs or lots of more specialized MQTs to help with particular queries
- Trade off between performance and maintenance

Summary of Changes

SQL Statements

- CREATE TABLE / ALTER TABLE (extended)
- REFRESH TABLE (new)

Special Registers

- CURRENT REFRESH AGE (new)
- CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION (new)

Catalog Changes

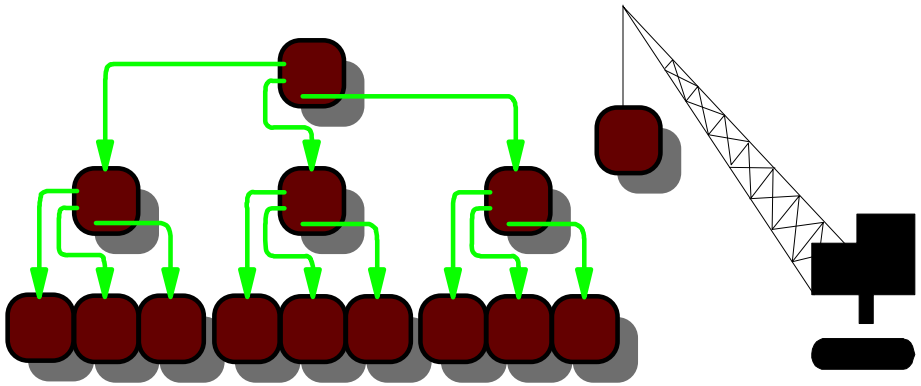
- New table/view type ’M’ for MQT in many catalog tables
- New columns for informational RIs in SYSIBM.SYSRELS
- New column NUM_DEP_MQTS in SYSIBM.SYSTABLES and SYSIBM.SYSROUTINES
- New columns REFRESH_TIME, REFRESH, ENABLE, ISOLATION, MAINTENANCE, and SIGNATURE in SYSIBM.SYSVIEWS

Index Support Improvements

Redbooks

Varying-length index keys (NOT PADDED)

Backward index scan



Varying-Length Index Keys

Prior to V8, data is varying length in the table but padded to the maximum length in indexes

In V8, padding or not padding varying-length columns to the maximum length in index keys is an option that you can choose

Performance Benefits:

- "Index-only access" for VARCHAR data
- Normally reduces the storage requirements for indexes since only actual data is stored

Varying-Length Index Keys - 2

Redbooks

New Keywords in CREATE/ALTER INDEX

- NOT PADDED
  - Require additional bytes to store the length information for the variable length columns within the key
- PADDED
  - Varying-length columns are padded to maximum length
  - May be a better option for performance at the expense of storage because of fast key comparisons since all keys have same length

Indexes are not automatically converted to NOT PADDED in V8

- ALTER INDEX is required

The default is controlled by the PADIX DSNZPARM

```
CREATE UNIQUE INDEX INDEX1
        ON TABLE1(COL01,VARCOL02) NOT PADDED
```

ALTER INDEX Changes

Alter index from PADDED to NOT PADDED

- Index placed in RBDP state
- No access to data through this index
- Reset by REBUILD INDEX

ALTER INDEX INDEX1 NOT PADDED

Alter index from NOT PADDED to PADDED

- Index placed in RBDP state
- No access to data through this index
- Reset by REBUILD INDEX

ALTER INDEX INDEX1 PADDED

Performance Expectation

Factors affecting performance of PADDED / NOT PADDED index:

NOT PADDED indexes can provide a true index-only access path

Size of the index pages

- In general, number of index pages for NOT PADDED index are fewer than for the PADDED index, thus less index getpages and index page I/Os
- Potentially fewer index levels
- Smaller index size favors the index to be chosen for a query (Index-only access)

Extra comparison

- NOT PADDED indexes that contain multiple VARCHAR columns require more than one comparison

Backward Index Scan

In V8, DB2 selects an ascending index and can use a backward scan to avoid the sort for the descending order

In V8, DB2 uses the descending index to avoid the sort and can scan the descending index backwards to provide the ascending order

To be able to use an index for backward scan,

- Index must be defined on the same columns as ORDER BY and
- Ordering must be exactly opposite of what is requested in ORDER BY
  - If index defined as DATE DESC, TIME ASC, can do:
    - Forward scan for ORDER BY DATE DESC, TIME ASC
    - Backward scan for ORDER BY DATE ASC, TIME DESC
- But must sort for
  - ORDER BY DATE ASC, TIME ASC
  - ORDER BY DATE DESC, TIME DESC

Stage 1 and Indexable Predicates

DB2 determines if a predicate is stage 1, based on predicate syntax, type and lengths of constants in the predicate, whether predicate evaluation is done before or after a join operation

Prior to V8, if data types and lengths do not match, the predicate is evaluated at stage 2

Major performance enhancement in DB2 V8 for queries involving predicates with mismatched data types and length comparison

Certain stage 2 predicates become stage 1, and possibly Indexable

- Eliminates table space scan when mismatch between host variable and DB2 column
- Improved situation in unknown join sequence

Mismatched Data Types

Becoming more common as not all programming languages support the full range of SQL data types, for example

- C/C++ has no DECIMAL data type
- Java has no fixed character data type

The following predicate types become stage 1 and also indexable for unlike data types

- Col op expression
- Expression op col
- Col BETWEEN expression 1 and expression 2
- Col IN (list)

Mismatched Operands Numeric Types Comparison


```
EMP( NAME      CHAR (20),
     SALARY    DECIMAL (12,2),
     DEPTID    CHAR (3) );
```

```
SELECT * FROM EMP
WHERE SALARY > :HV_FLOAT ;
```

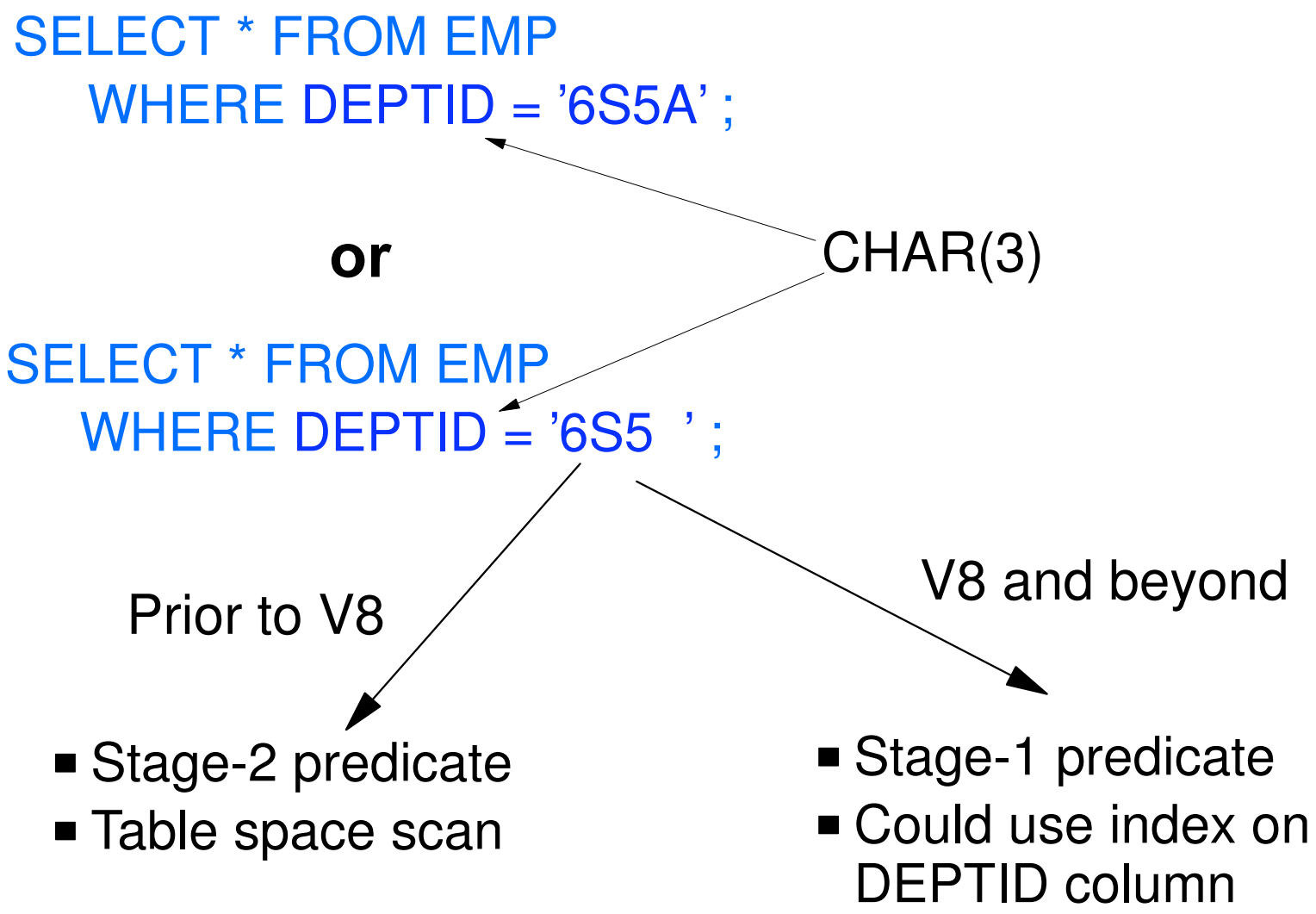
Prior to V8

- Stage-2 predicate
- Table space scan

V8 and beyond

- Stage-1 predicate
- Could use index on SALARY column

Mismatched Operands String Types



Unknown Join Sequences

Where both operands are columns from different tables, whether the predicate is stage 1 or stage 2 is determined by

- The join sequence
- Which column is considered the column and which the expression

A join predicate between a user-defined table function and a base table is stage 1 if the base table is the inner table of the join and the other sargability conditions are met

Unknown Join Sequence Column Expression


```
SELECT E1.*
    FROM EMP E1, EMP E2, DEPT
    WHERE E1.DEPTID = DEPT.ID   AND
                DEPT.MGR = E2.NAME  AND
                E1.SALARY > E2.SALARY * 1.10
;
```

DECIMAL(12,2)

Prior to v8
(w/o PQ54042)

- Stage-2 predicate

V8 and beyond

- If E2 is inner table
  - Stage-2 predicate
- If E1 is inner table
  - Stage-1 predicate
  - Could use index on E1.SALARY

Unknown Join Sequence Column Expression


```
SELECT E1.*
    FROM EMP E1, EMP E2, DEPT
    WHERE E1.DEPTID = DEPT.ID   AND
                DEPT.MGR = E2.NAME  AND
                E1.SALARY > E2.SALARY * 1.10
;
```

DECIMAL(12,2)

Prior to v8
(w/o PQ54042)

- Stage-2 predicate

V8 and beyond

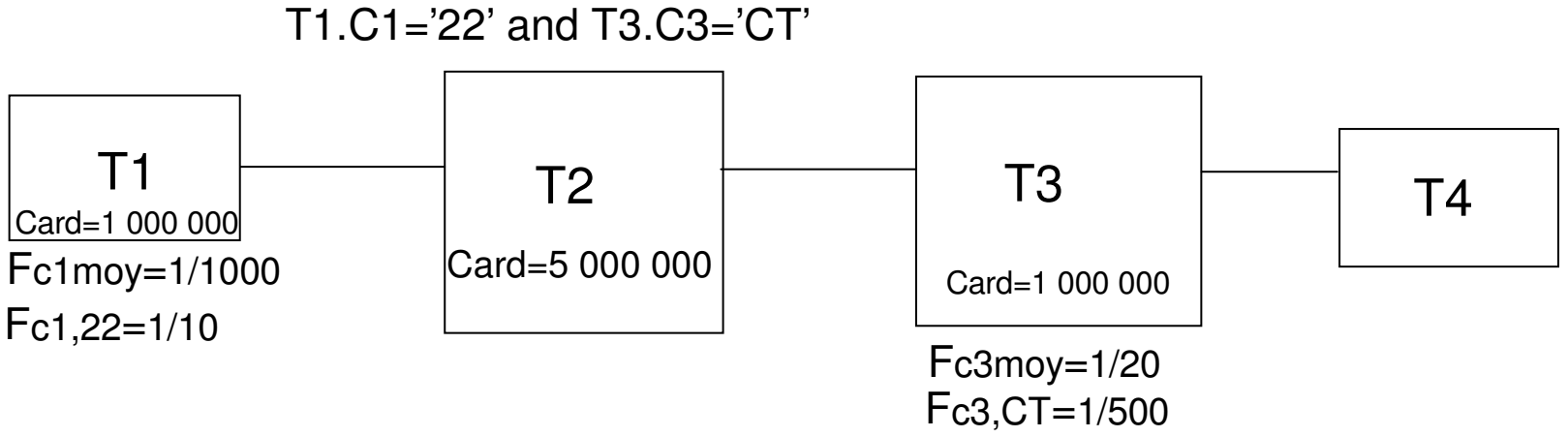
- If E2 is inner table
  - Stage-2 predicate
- If E1 is inner table
  - Stage-1 predicate
  - Could use index on E1.SALARY

The Need for Extra Statistics

Distribution statistics are currently collected for indexed columns only (if ... FREQVAL NUMCOLS(1) COUNT(10))

Non-uniform distribution statistics for non-leading indexed columns are not collected by RUNSTATS, which can result in non-optimal performance

- Less efficient join sequences
- Inappropriate table join method
- Increase in the number of rows that need to be processed



RUNSTATS Enhancement

Can collect distribution statistics on any column, or group(s) of columns, indexed and non-indexed, specified at the table level

Frequency distributions for (non-indexed) columns or groups of columns

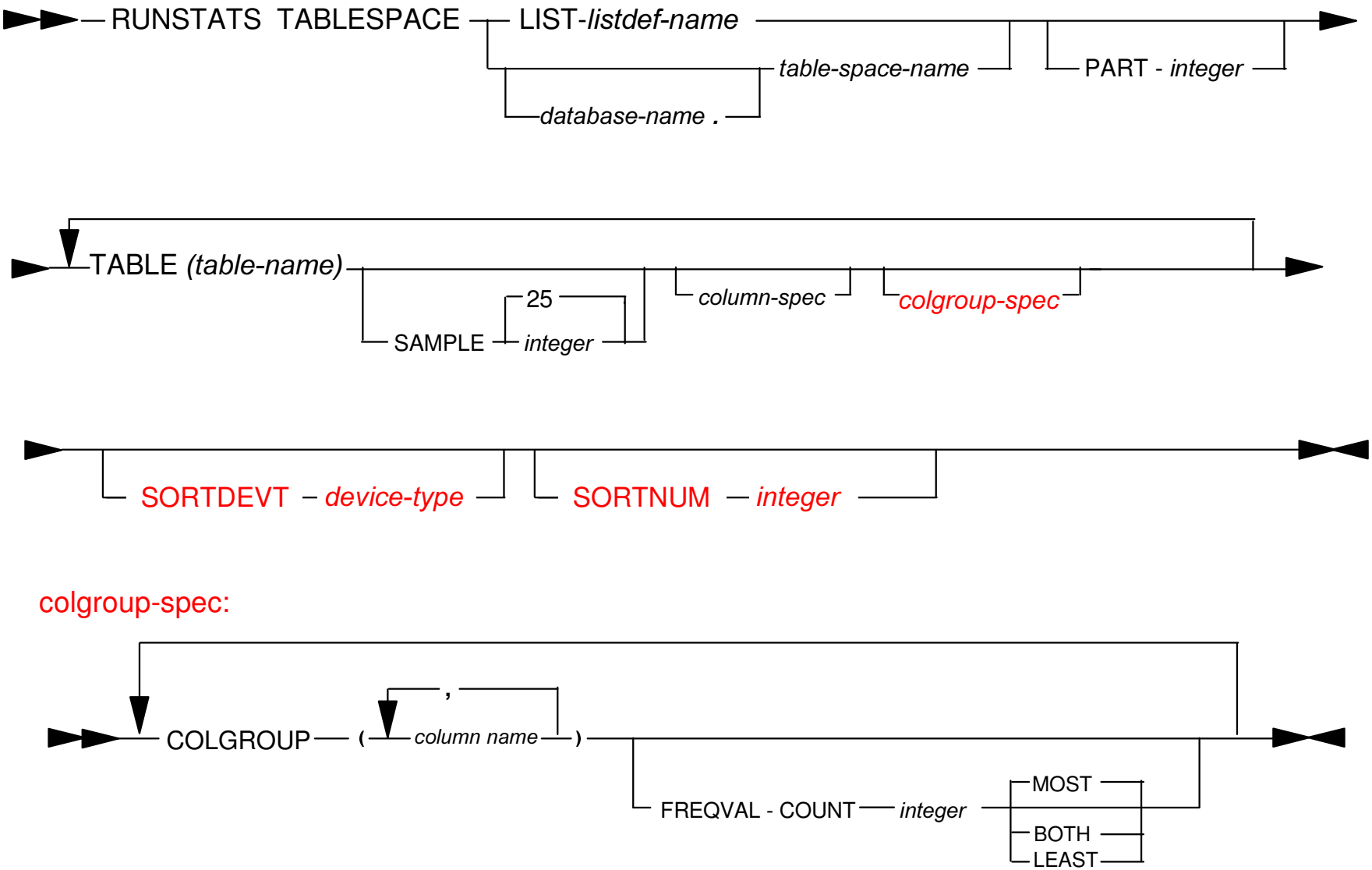
Cardinality values for groups of (non-indexed) columns

LEAST frequently occurring values, along with MOST for both indexed and non-indexed column distributions

New keywords : COLGROUP MOST LEAST BOTH

SORTNUM SORTDEVT

RUNSTATS Utility Syntax Changes



Example 1

- RUNSTATS TABLESPACE DSN8D81A.DSN8S81E TABLE(DSN8810.EMP) COLGROUP(EDLEVEL,JOB,SALARY)

SYSIBM.SYSCOLDIST

NAME	TYPE	NUMCOLUMNS	COLGROUPCOLNO	CARDF
EDLEVEL	C	3	00090008000C	+0.330000E+02

SYSIBM.SYSCOLDISTSTATS

PARTITION	NAME	TYPE	NUMCOLUMNS	COLGROUPCOLNO	CARDF
1	EDLEVEL	C	3	00090008000C	+0.320000E+02
2	EDLEVEL	C	3	00090008000C	+0.000000E+00
3	EDLEVEL	C	3	00090008000C	+0.100000E+02
4	EDLEVEL	C	3	00090008000C	+0.000000E+00
5	EDLEVEL	C	3	00090008000C	+0.000000E+00

EDLEVEL,JOB,SALARY are non-indexed columns
Cardinality value of the column group is stored in the catalog tables

Example 2

- RUNSTATS TABLESPACE DSN8D81A.DSN8S81E TABLE(DSN8810.EMP) COLGROUP(EDLEVEL)
  FREQVAL COUNT 100 MOST

SYSIBM.SYSCOLDIST

NAME	TYPE	NUM COLUMNS	COL GROUPCOLNO	CARDF	COL VALUE	FREQUENCYF
EDLEVEL	C	3	00090008000C	+0.33000E+02	-	-
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø...	+0.952380952380952 3E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.476190476190476 2E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.238095238095238 1E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø...	+0.238095238095238 1E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.238095238095238 1E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.238095238095238 1E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.238095238095238 1E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø...	+0.238095238095238 1E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.238095238095238 1E-01
EDLEVEL	**F**	3	00090008000C	-0.10000E+01	.Ø....	+0.238095238095238 1E-01

....

The cardinality value of the column group and 100 most frequent values are collected for the column group specified (EDLEVEL)

Performance Expectation

Performance gain in certain queries (such as queries that use non-leading indexed columns and/or where non-indexed columns are used as predicates, and that have skewed (non-uniform) data)

CPU and elapsed times for RUNSTATS utility increases if new functions are invoked

- Actual amount of increase depends on the number of columns specified in the COLGROUP keyword
- Number of COLGROUPs specified
- Amount of sorting that needs to be done be RUNSTATS
- Do a specific RUNSTAT to collect non-uniform statistics (1 time / quarter)

Note:

Do not forget to allocate sort workfiles for RUNSTATS when collecting statistics on non-indexed columns

Cost-Based Parallel Sort

Redbooks

DB2 V8 introduces cost-based consideration to determine if parallel sort for single or multiple (composite) tables should be disabled

- Sort data size < 2 MB (500 pages)
- Sort data per parallel degree < 100 KB (25 pages)

Hidden DSNZPARM OPTOPSE control - default is ON

- ON - cost-based parallel sort considerations for both single table and multi-table
- OFF - not cost-based - sort parallelism behavior same as V7, i.e.
  - Single (composite) table : parallel sort
  - Multiple tables : sequential sort only

Considerations:

- Elapsed time improvement
- More usage of workfiles and virtual storage

Query Example - Plan Table

EXPLAIN output to determine if parallel sort is executed or not

```
SELECT * from Ta, Tb, Tc   where Ta.a2 = Tb.b2 and Tb.b3 = Tc.c3;

          SMJ_1          SMJ_2
 Ta    --------> Tb  ---------> Tc  (Merge scan join used to join the tables)
```

Prior to DB2 V8

PLAN	TNAME	ACC PGR ID	SORTC _PGR _ID	SORTN _PGR _ID	JOIN _PGR _ID
1	Ta	1	?	?	?
2	Tb	2	1	2	3
3	Tc	4	?	4	5

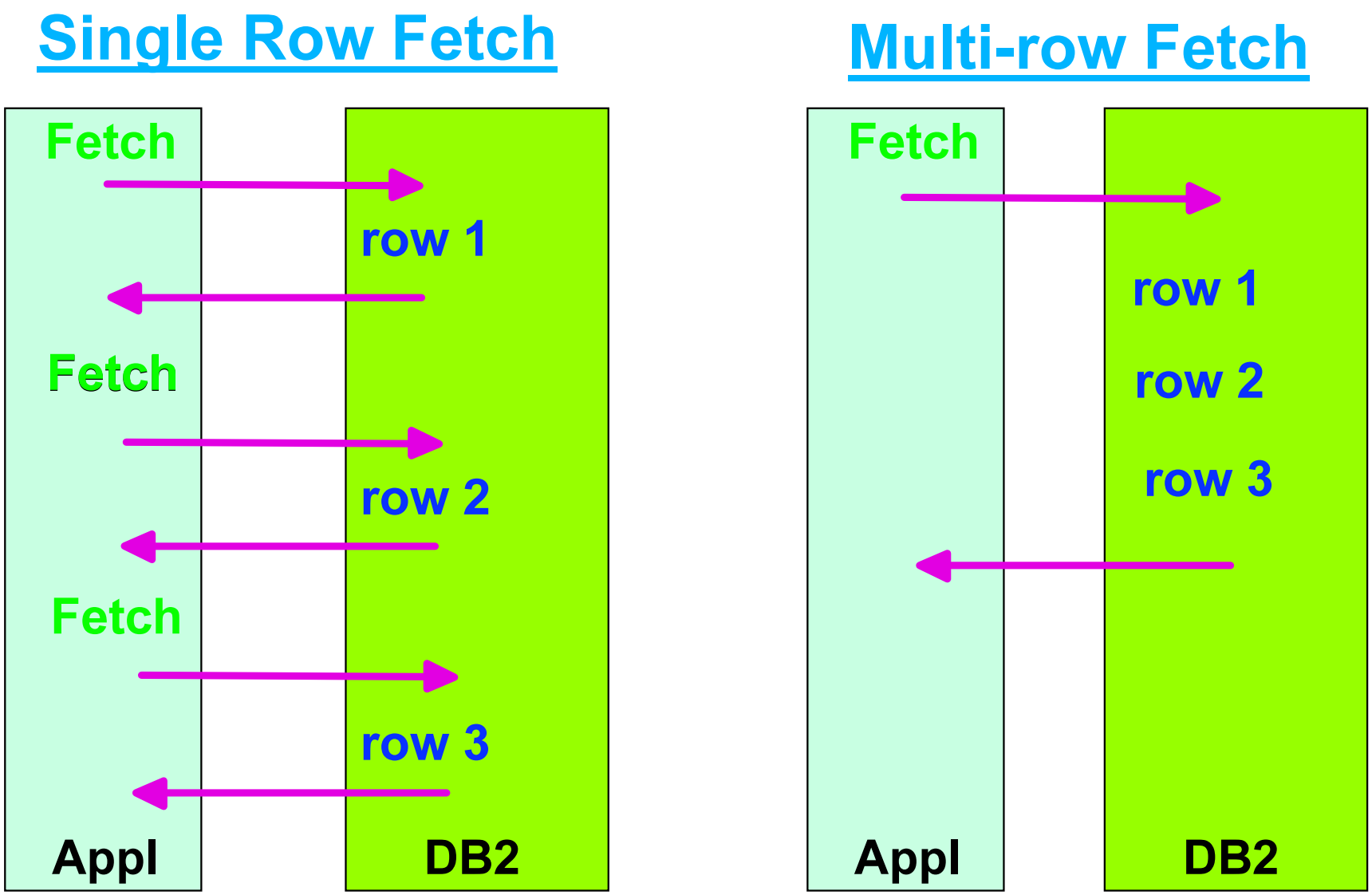
SORTC is executed in sequential mode

DB2 V8 (with Hidden DSNZPARM OPTOPSE set to ON -> Post-Optimizer decides to do parallel sort)

PLAN	TNAME	ACC PGR ID	SORTC _PGR _ID	SORTN _PGR _ID	JOIN _PGR _ID
1	Ta	1	?	?	?
2	Tb	2	1	2	3
3	Tc	4	3	4	5

SORTC is executed in parallel mode

Multi-row Fetch - Local Application

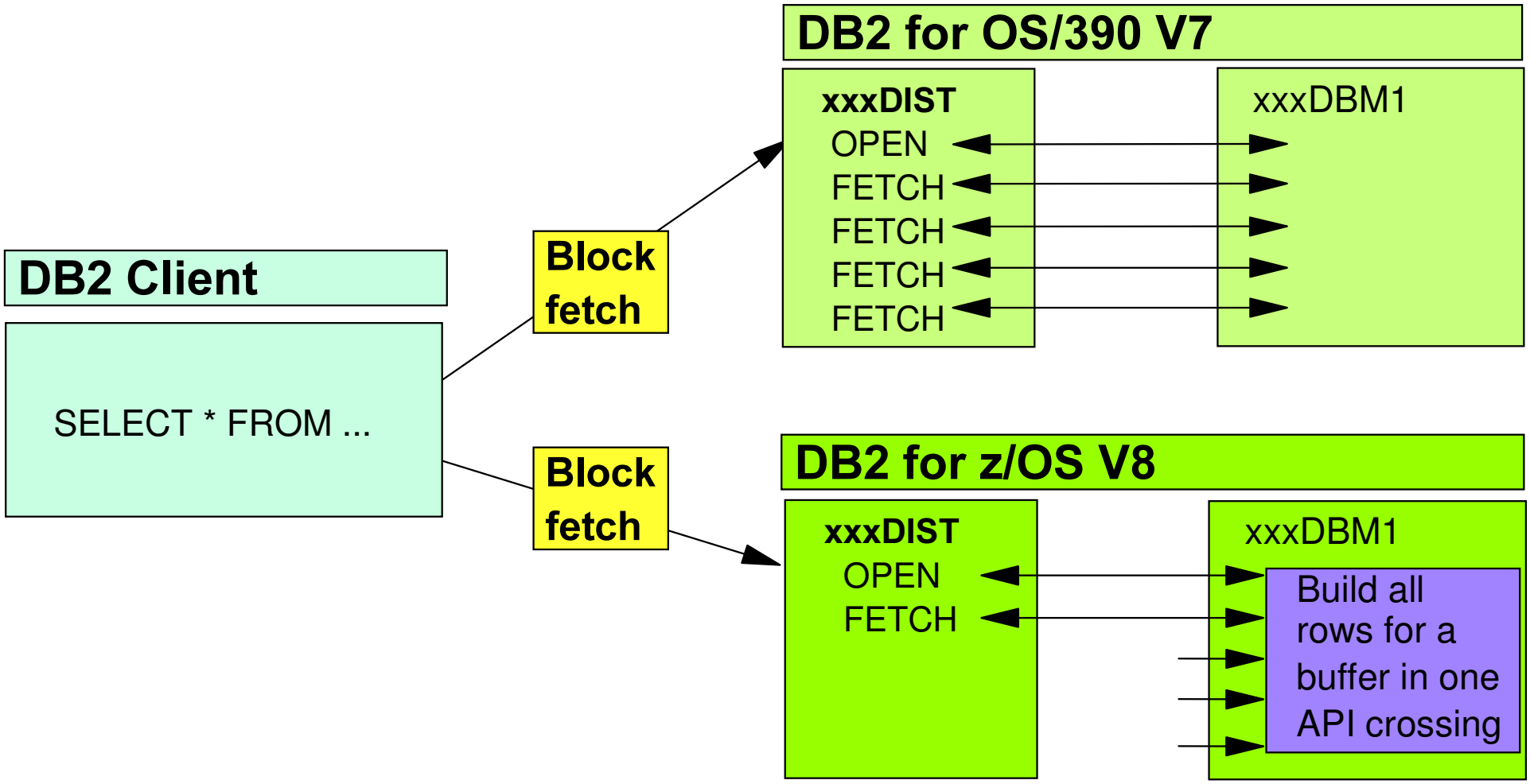


DDF and multi-row FETCH

Redbooks

DDF automatically uses multi-row FETCH for read-only queries

Reduces CPU cost of read-only queries significantly



Volatile Table Enhancement

What is a volatile table

- A table whose contents can vary from empty to very large at run time

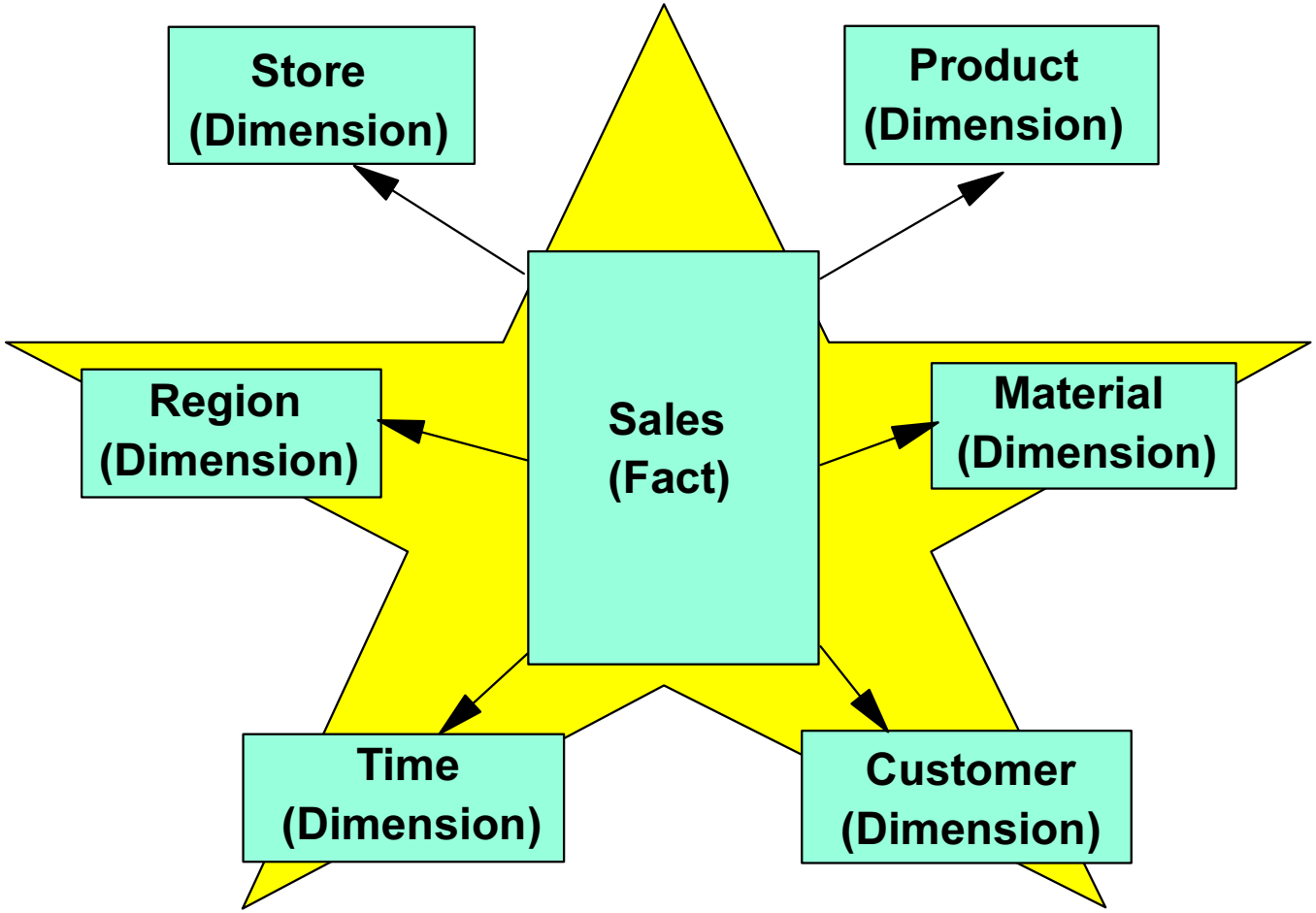
Generating an access plan based on the point-in-time captured statistics can result in incorrect or poorly performing plan

- For example, if statistics are gathered when the volatile table is empty, the optimizer tends to favor accessing the volatile table using a table scan rather than an index

Declaring a table as "volatile" influences the optimizer to favor matching index access rather than depend on the existing statistics for that table

CREATE TABLE ... **VOLATILE**

Star Join Processing Enhancements



Star Schema Data Model

Star schema data model

- Often used in data warehouses, data marts and OLAP
  - To minimize data redundancy
  - Has highly normalized tables (dimensions or snowflakes)
  - Has a centralized table (fact table)

Star schema query characteristics

- Join predicates between the fact and dimension tables are equi-join predicates
- Existence of local predicates on the dimension tables
- Large number of tables (dimensions/snowflakes and fact tables) participating in the query

Typical Star Schema Query

id	month	qtr	year
1	Jan	1	2002
2	Feb	1	2002
3	Mar	1	2002
4	Apr	2	2002
5	May	2	2002
6	Jun	2	2002
39	Mar	1	2003

time (dimension)
39 rows

id	city	region	country
1	New York	East	USA
2	Boston	East	USA
3	Chicago	East	USA
4	San Jose	West	USA
5	Seattle	West	USA
6	Los Angles	West	USA

region (dimension)
1,000 rows

id	item	class	department
1	stereo	audio	audio-visual
2	cd player	audio	audio-visual
3	television	video	audio-visual

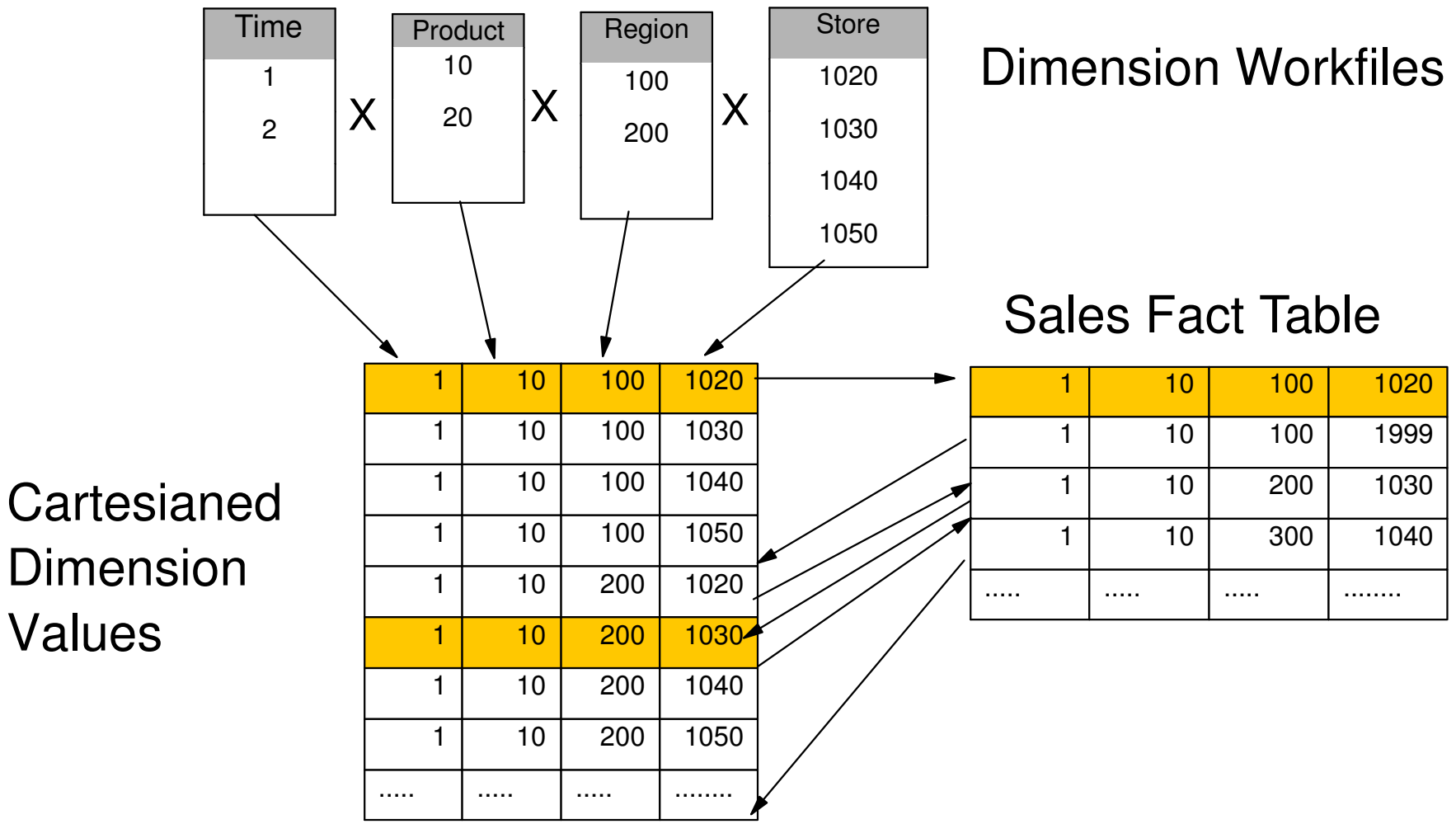
product (dimension)
60,000 rows

time	locn	prod	customer	seller	....
1	1	1	123	22	
2	5	2	345	33	
2	2	2	567	66	
2	2	1	789	12	
3	3	3	112	23	
3	6	2	348	78	
2	6	1	777	60	

sales (fact)
150 million rows

```
SELECT *
FROM  SALES S, TIME T,
          REGION L, PRODUCT P
WHERE  S.TIME = T.ID
     AND  S.REGION = L.ID
     AND  S.PRODUCT = P.ID
     AND  T.YEAR = 2002
     AND  T.QTR = 1
     AND  L.CITY IN  ('Boston','Seattle')
     AND  P.ITEM = 'stereo';
```

Index Key Feedback Technique

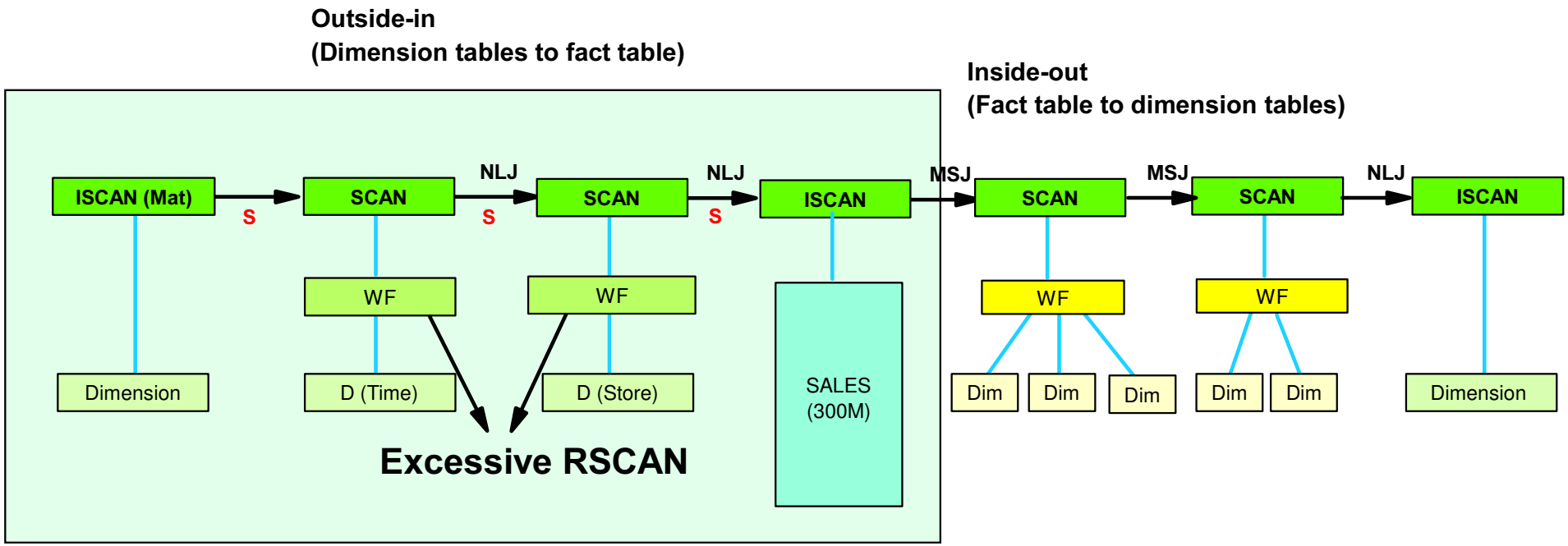


Workfiles in Outside-In Processing

Redbooks

During 'logical' cartesian product, DB2 frequently repositions in the dimension workfiles

Processing of the workfiles can only be done by scanning them over and over again (RSCAN)



Inside-Out Stage Workfile Challenges

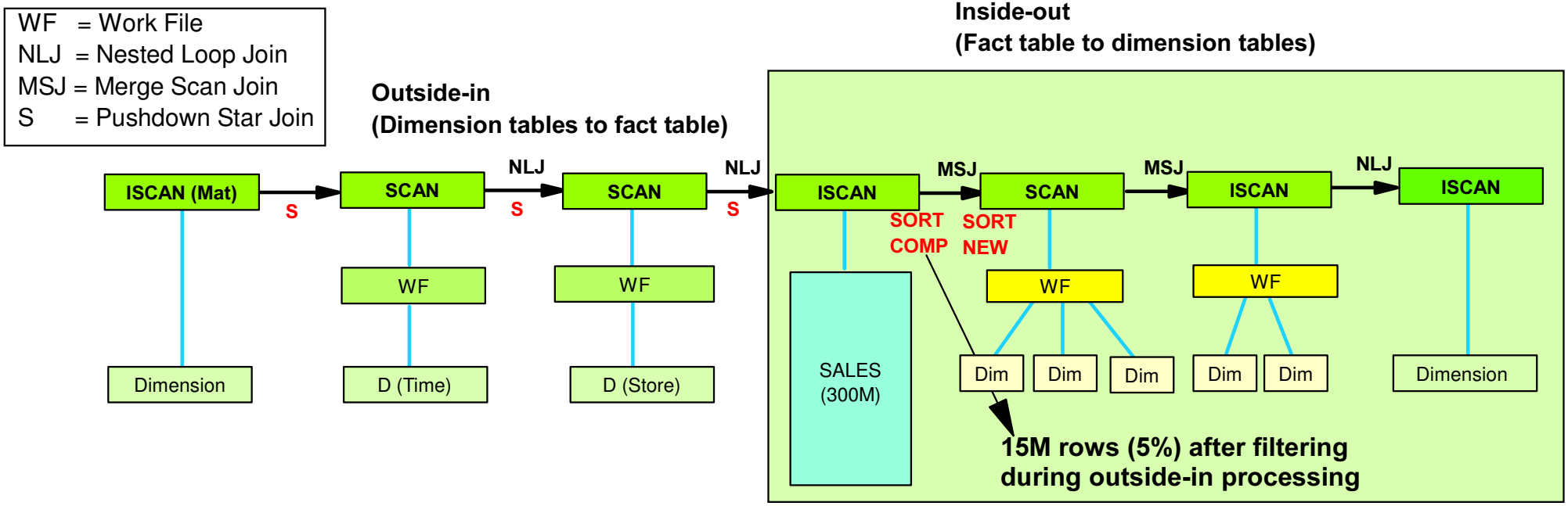
Redbooks

Typical data warehouse queries touch significant portion of fact table

In the inside-out stage, sort is performed on the composite table (the result of outside-in processing) at the start of MSJ processing (materialized work file has no index)

Sorting the large result set is a performance concern (time and space)

Sort of "new" table (materialized snowflake - workfile) before MSJ



Star Join Workfile Challenges

Where do they come from?

- Materialization of snowflakes
- Sort of dimension tables

Typically small (hundreds or thousands)

Today, RSCAN is the only option

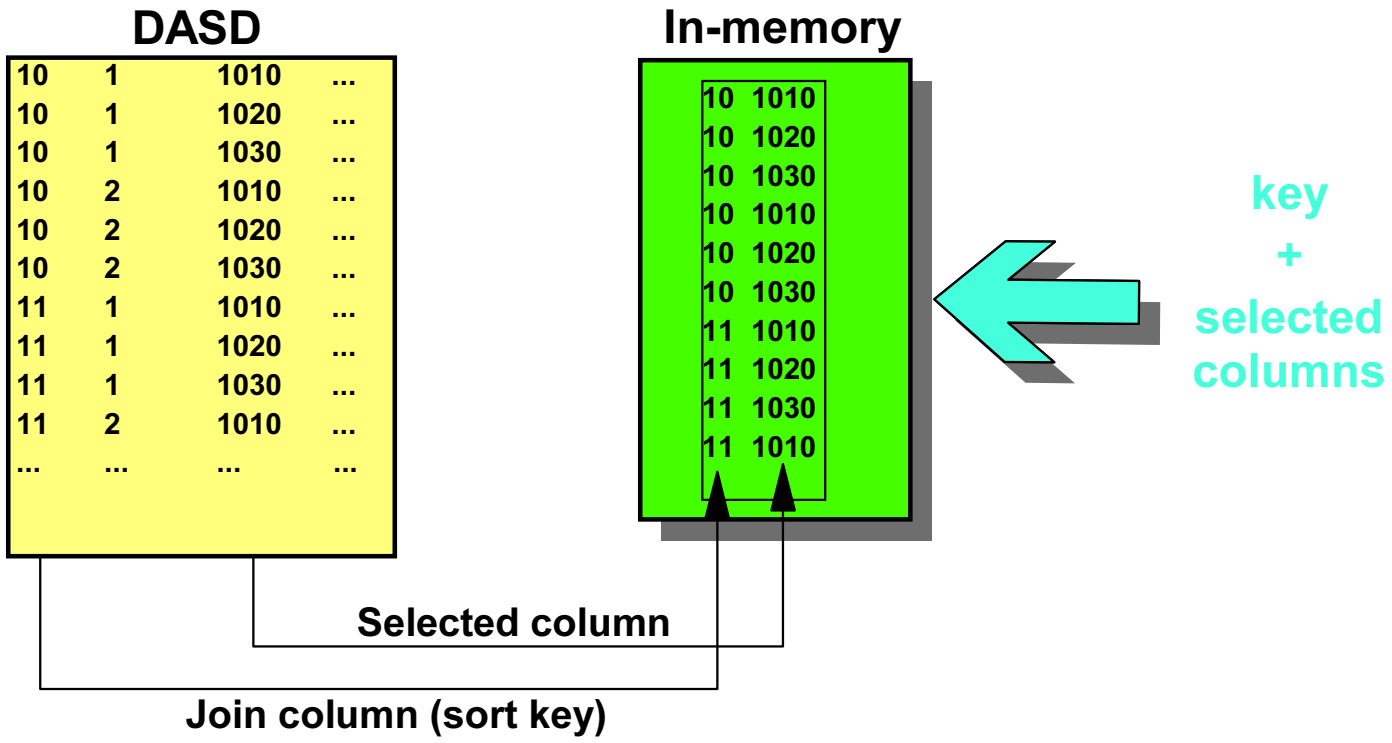
- Outside-in processing
- Inside-out processing (sparse index can also be used in V7)

Version 8 enhancements

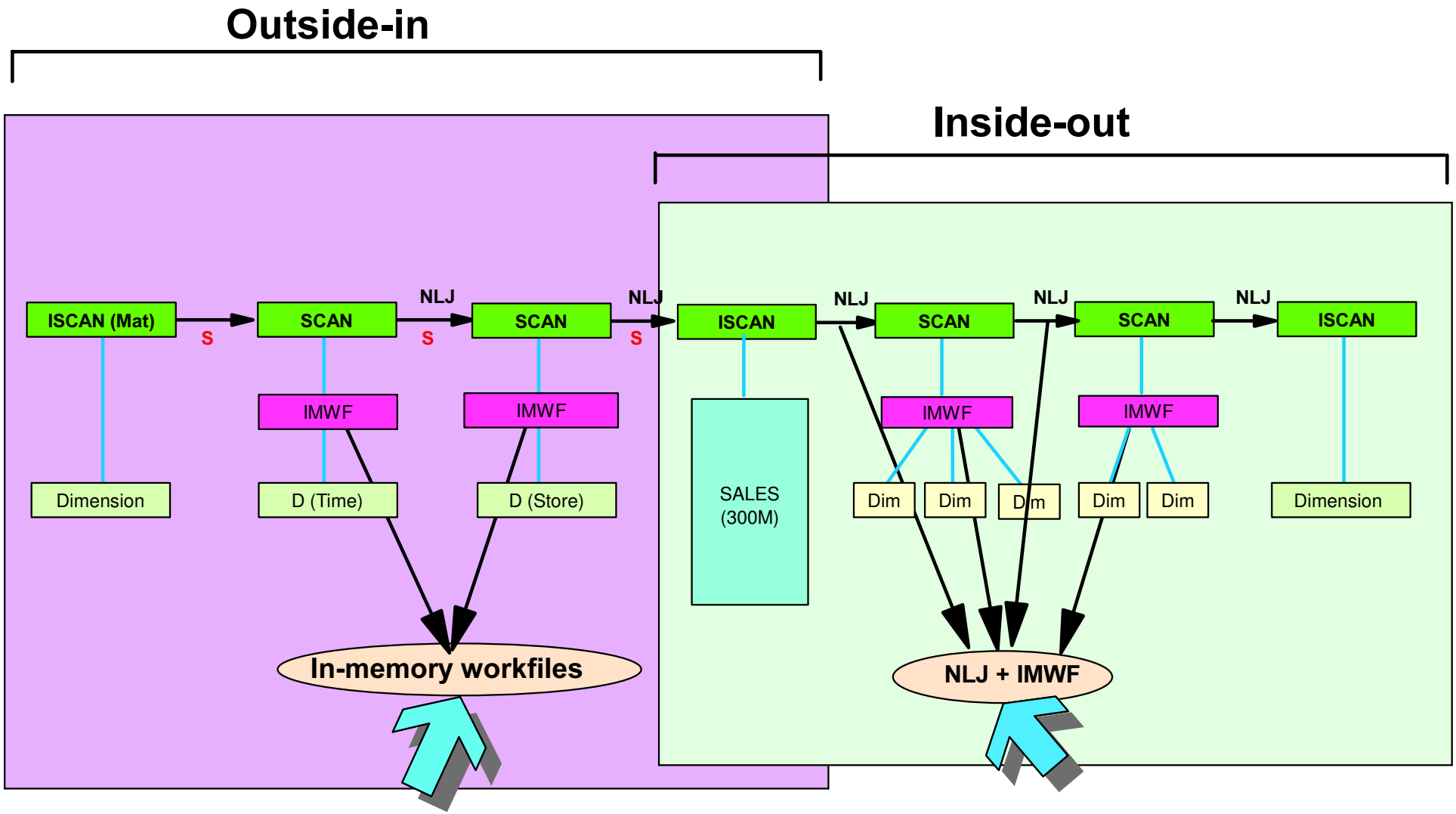
- In-memory workfiles
- Sparse index on workfiles (also during outside-in processing)

In-memory Workfiles

- In-memory data structure - above the bar (DSNZPARM SJMXPOOL)
- Sorted in the join column order
- Containing only the join column and the selected columns
- Binary search for the target row
- More beneficial for large join composite
- Ideal for scanning dimension workfiles



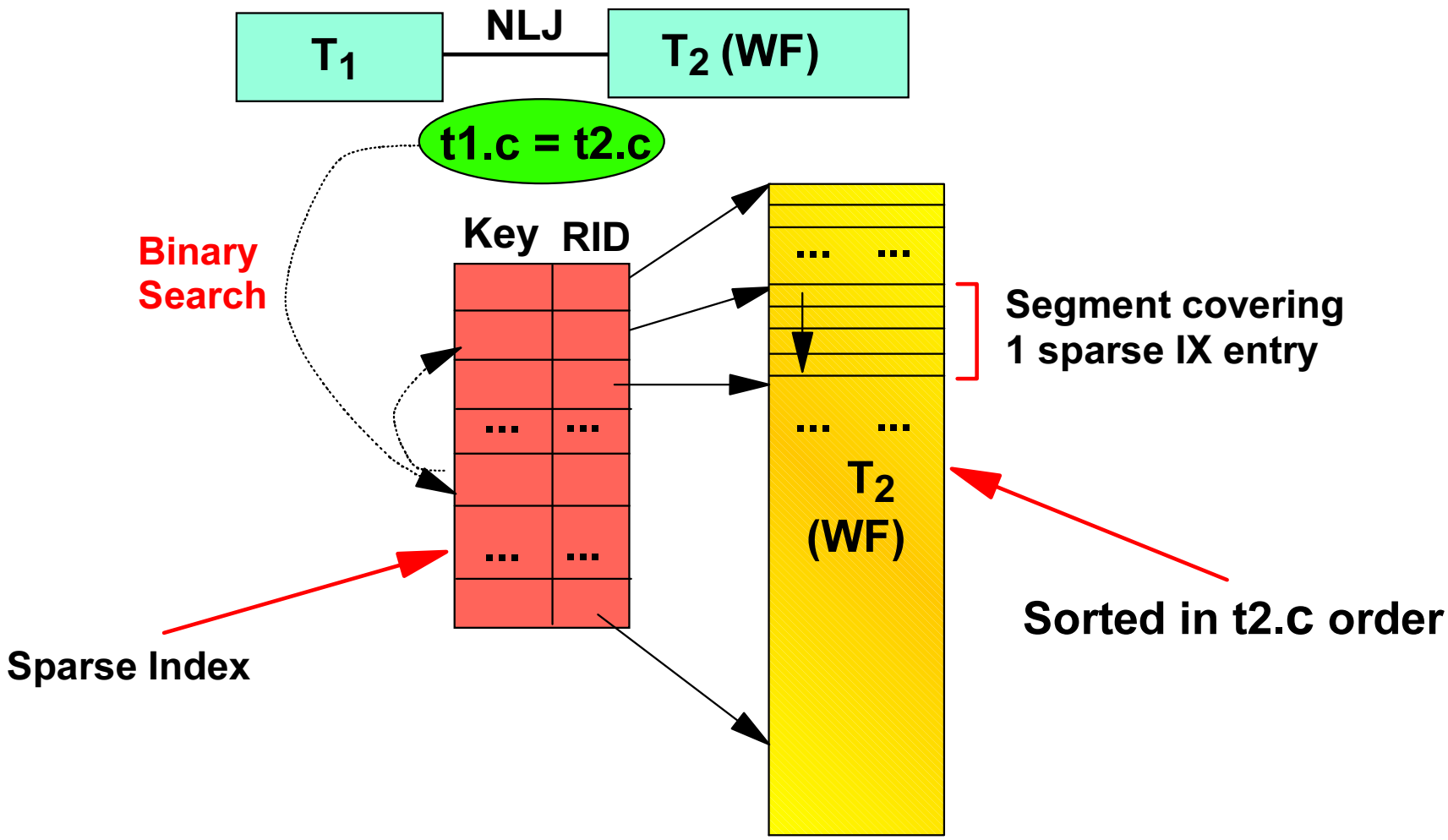
In- memory Workfile Benefits



Sparse Index on Workfiles for Star Join

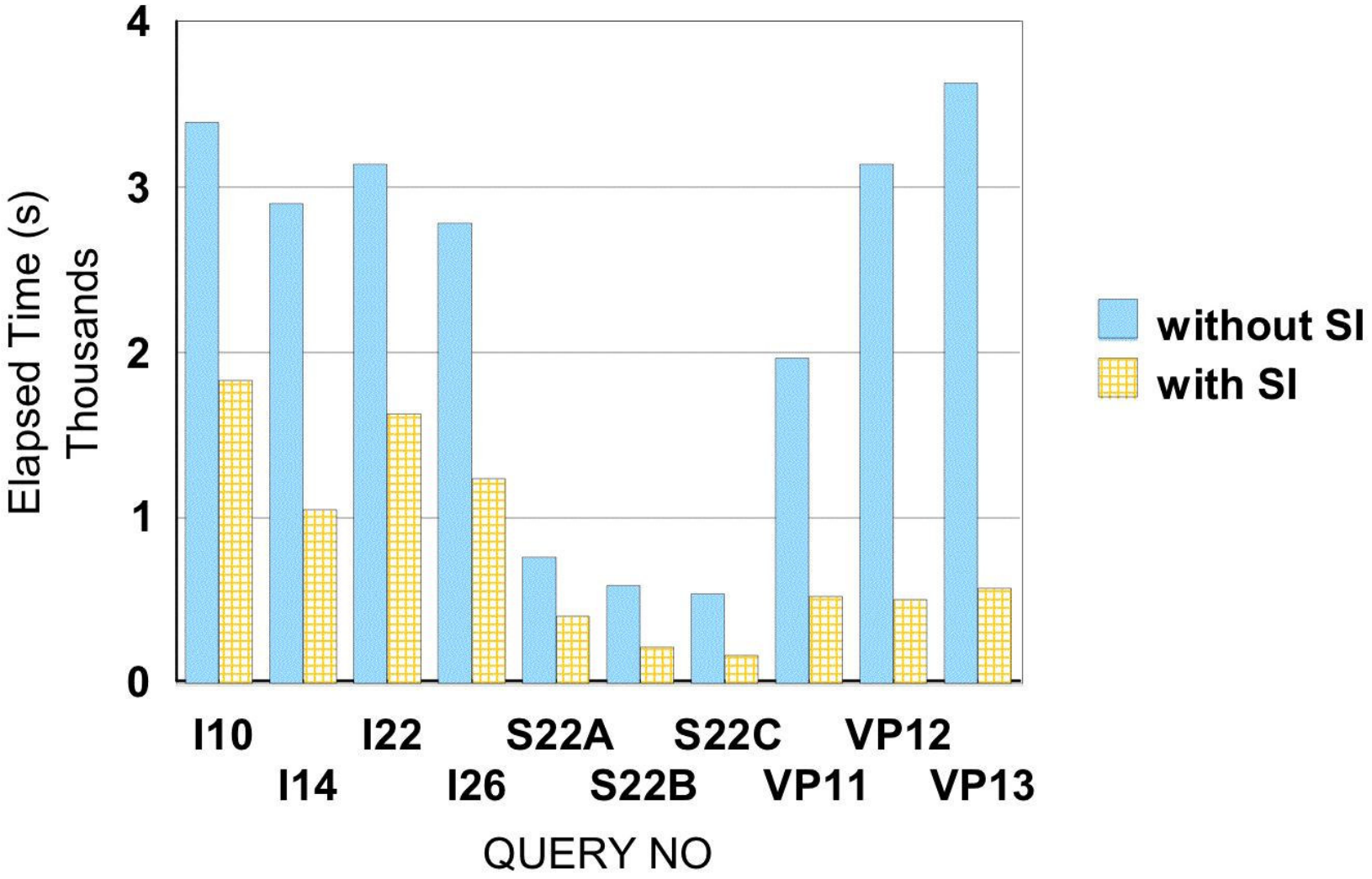
- Introduce sparse index on the join key for the materialized work files
- Thus nested loop join is possible. It replaces the merge scan join and eliminates the sort of the large outer composite table (fact table)
- Sparse index
  - In-memory virtual index
  - Probed through an equal-join predicate
  - Binary search for the target portion of the table
  - Sequential search within the target portion if it is sparse
  - The denser the faster - in favor of small work files
  - More beneficial for large join composite
  - backup solution for dimension work file access in star schema scenario if in-memory workfiles fail

How Does Sparse Index Work



Sparse index (SI) performance

- Large elapsed time reduction



Example of a Star Join Access Path

Query#	PlN#	Corr Name	Table Name	Join Mtd	Join Type	Acc Type	Access Name	Sort New
11001	1	DP	/BI0/D0SD_C01P	0	S	I	/BI0/D0SD_C01P~0	N
11001	2	DT	/BI0/D0SD_C01T	1	S	T		Y
11001	3	DU	/BI0/D0SD_C01U	1	S	T		Y
11001	4	F	/BI0/F0SD_C01	1	S	I	/BI0/F0SD_C01~0	N
11001	5	D5	/BI0/D0SD_C015	1		I	/BI0/D0SD_C015~0	N
11001	6	D3	/BI0/D0SD_C013	1		I	/BI0/D0SD_C013~0	N
11001	7		DSN_DIM_TBLX(02)	1		T		Y
11001	8	D2	/BI0/D0SD_C012	1		I	/BI0/D0SD_C012~0	N

Access_type T indicates either "sparse index" or "data caching" is used

The final decision is taken at runtime and cannot be shown by EXPLAIN

REOPT(ONCE)

Redbooks

Bind option that controls when the Optimizer builds the access path information for dynamic SQL applications

- By default, access path is built at PREPARE
- REOPT(ONCE)
  - Defers access path selection until OPEN
  - Values of host variables on OPEN are used to build access path
  - Resulting access path is cached in the global prepared cache

New terminology

- REOPT(NONE) equivalent to NOREOPT(VARS)
- REOPT(ONCE)
- REOPT(ALWAYS) equivalent to REOPT(VARS)

DTT ON COMMIT DROP TABLE Option

Redbooks

V7

- You must explicitly drop a DTT before COMMIT. Otherwise the remote connection cannot become inactive

V8

- New ON COMMIT DROP table option on the CREATE DECLARED GLOBAL TEMPORARY TABLE statement
- DTT implicitly dropped at commit if no open held cursor against the DTT
- Significant enhancement for DDF connections that can now become inactive without an explicit drop of the DTT

Non-correlated EXISTS subquery

Prior to V8, ALL qualifying rows of a non-correlated EXISTS subquery are retrieved and stored in a workfile

- Potentially a lot of processing to find all qualifying rows for the subquery
- Potentially a lot of workfile space to store all these rows

DB2 V8 stops evaluating the non-correlated EXISTS subquery as soon as ONE qualifying row is found

- One qualifying row is sufficient to evaluate the predicate as true
  - Example:

```
SELECT EMPNO,LASTNAME
  FROM DSN8810.EMP
  WHERE EXISTS (
              SELECT *
                FROM DSN8810.PROJ
                WHERE PRSTDATE > '2005-01-01');
```

IN List Processing Enhancements

Redbooks

V7 Enhancements to IN list processing

- Predicate pushdown for IN list predicates (V7 APAR PQ73454)
- Correlated subquery transformation enhancement (V7 APAR PQ73749)
- Activated by INLISTP DSNZPARM (default value 0 in V7 = disabled)

By default enabled in V8 (INLISTP=50)

- IN list predicate pushdown into nested table expression (NTE) or materialized view (MV)
  - Better filtering inside NTE and MV ; fewer resulting rows
  - Potential index usage on columns when resolving NTE or MV
- Correlated subquery transformation enhancement
  - IN list predicates generated by predicate transitive closure "pulled up" to parent query block
    - Filtering can be done at parent level, resulting in fewer invocations of subquery executions
    - IN list predicate on parent query block can take advantage of existing indexes and provide a better access path for parent query block

Visual Explain Enhancements

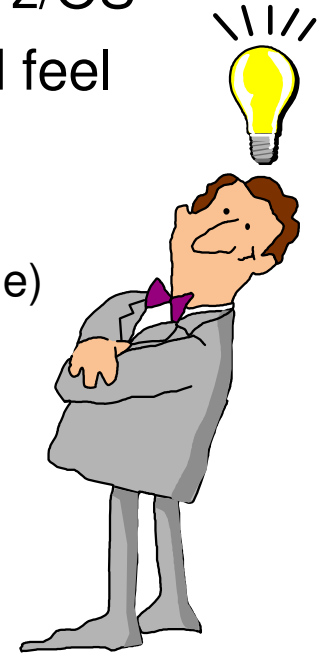
Redbooks

Component of no-charge Management Client Package feature

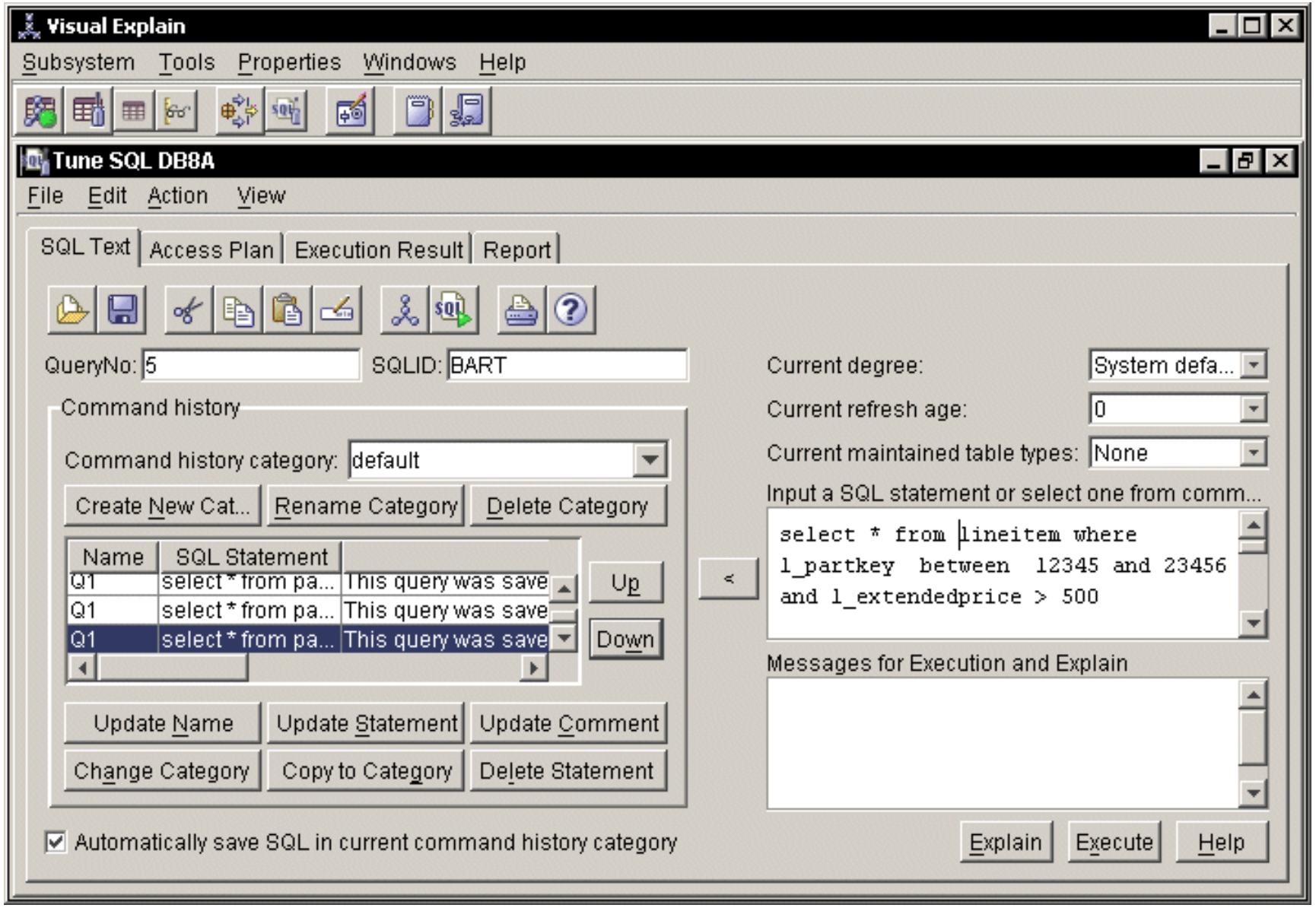
- As in V7

Complete rewrite of V7 product

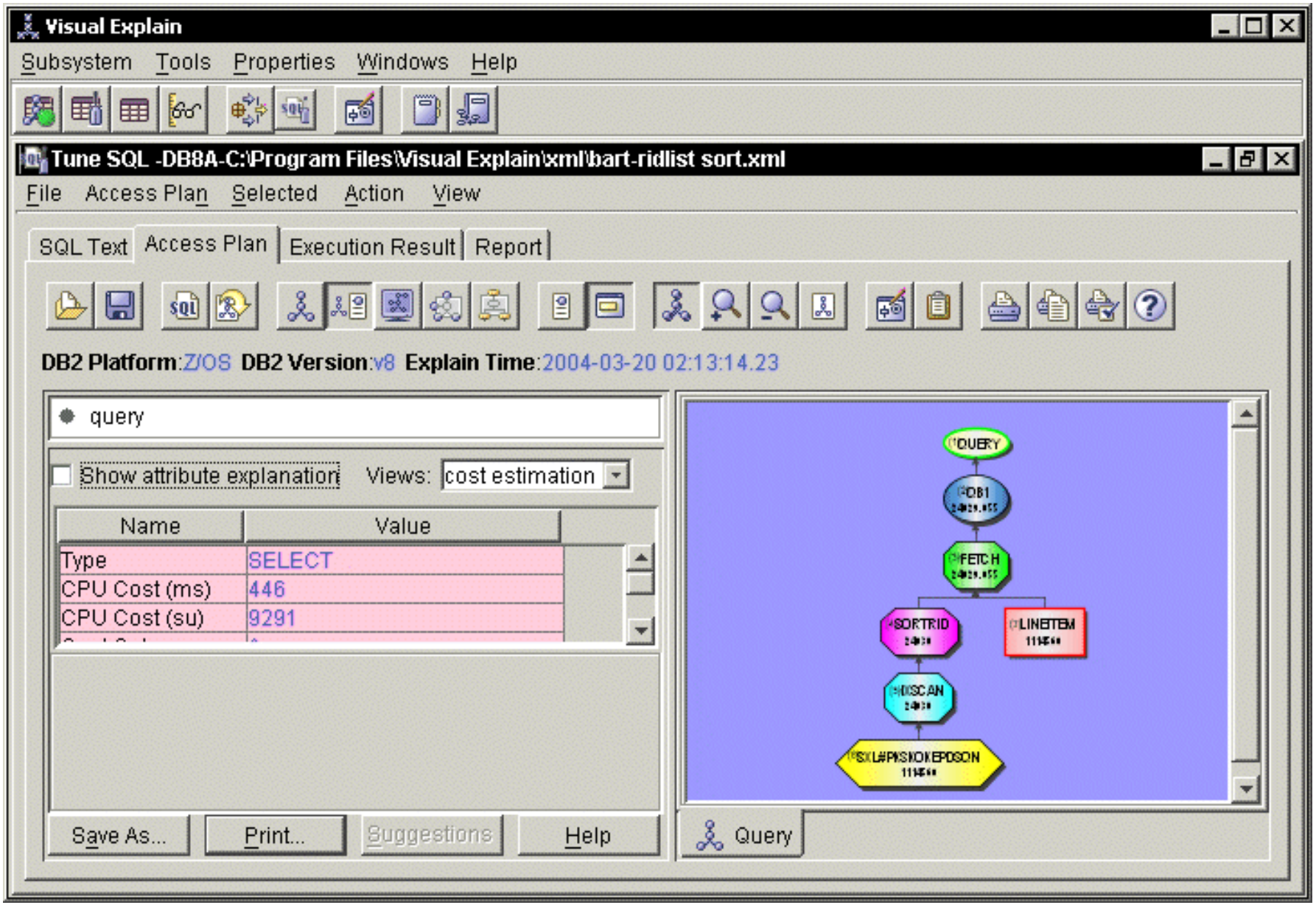
- Rewritten in Java, connecting via DB2 Connect to DB2 for z/OS
- Basic capabilities of V7 Visual Explain with better look and feel
- Enhanced explain capabilities
  - Qualified row estimates
    - Including for individual tables in a join (as if table was the outer table)
  - Predicate information (Matching, screening, Stage 1, Stage 2)
  - Limited partition scan details
  - Parallelism details
  - Sort estimation
- Enhanced reporting
  - Generate HTML reports
  - XML output file
  - Generate information for IBM Service Team



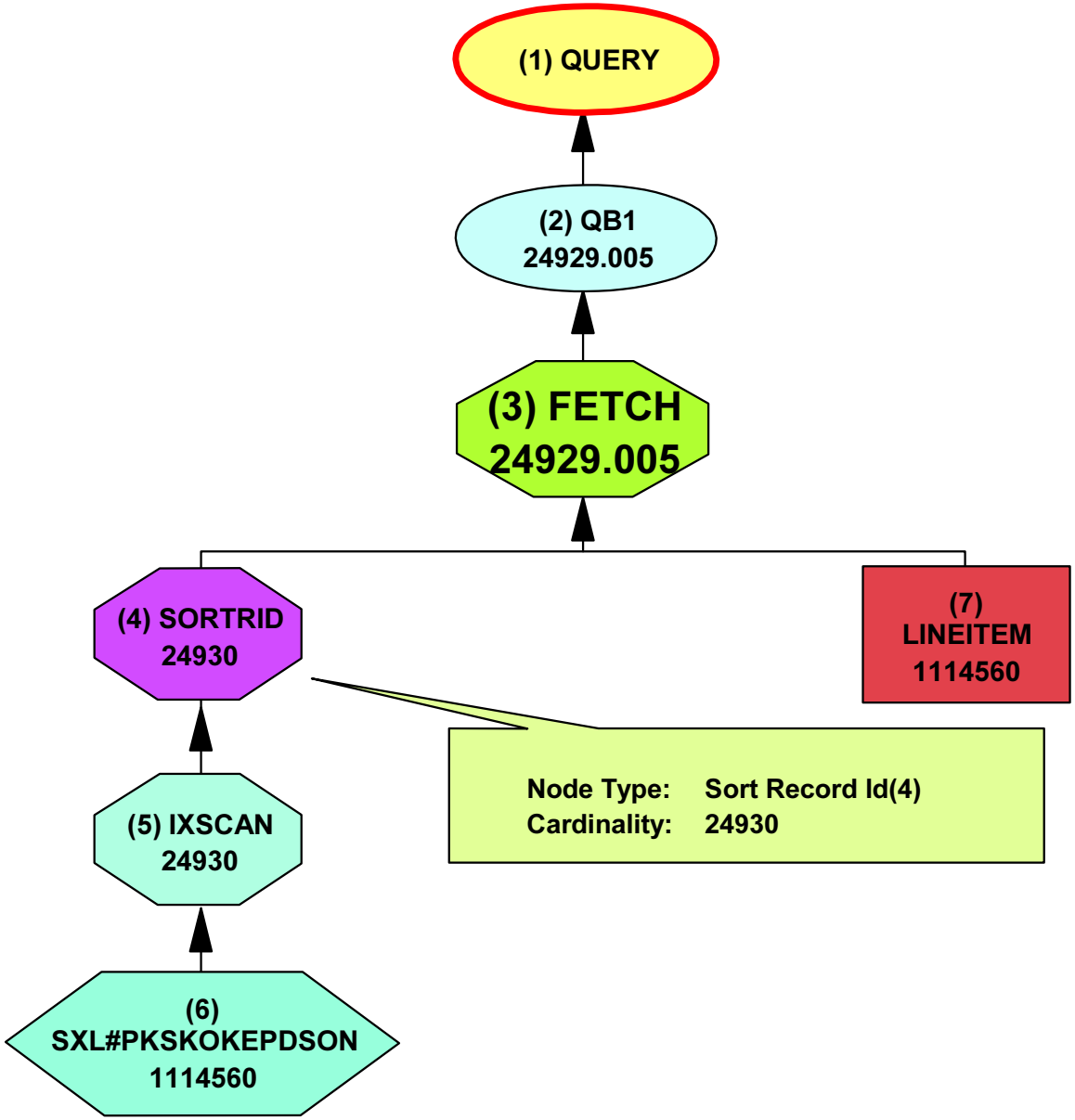
Tune SQL



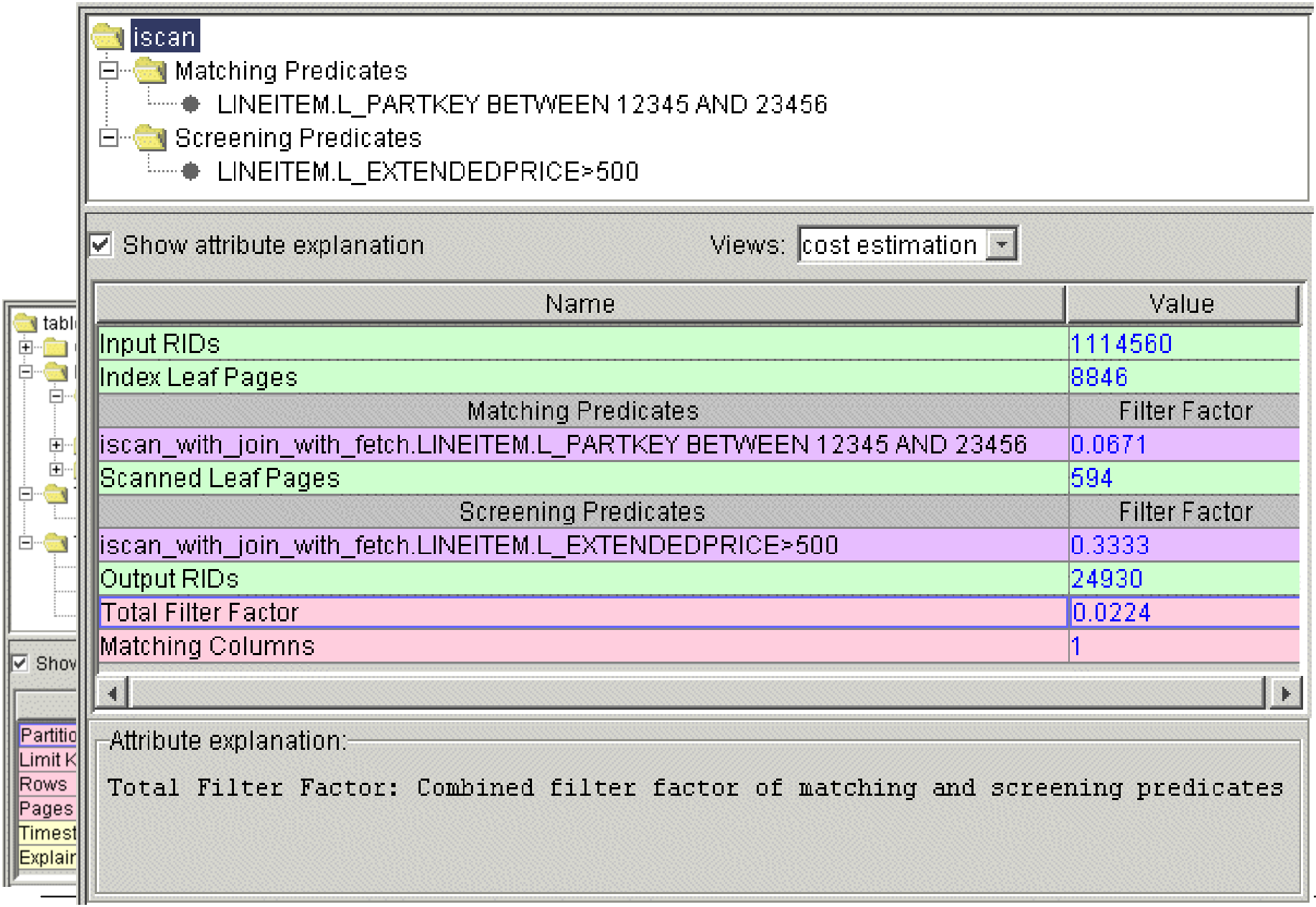
Access Plan



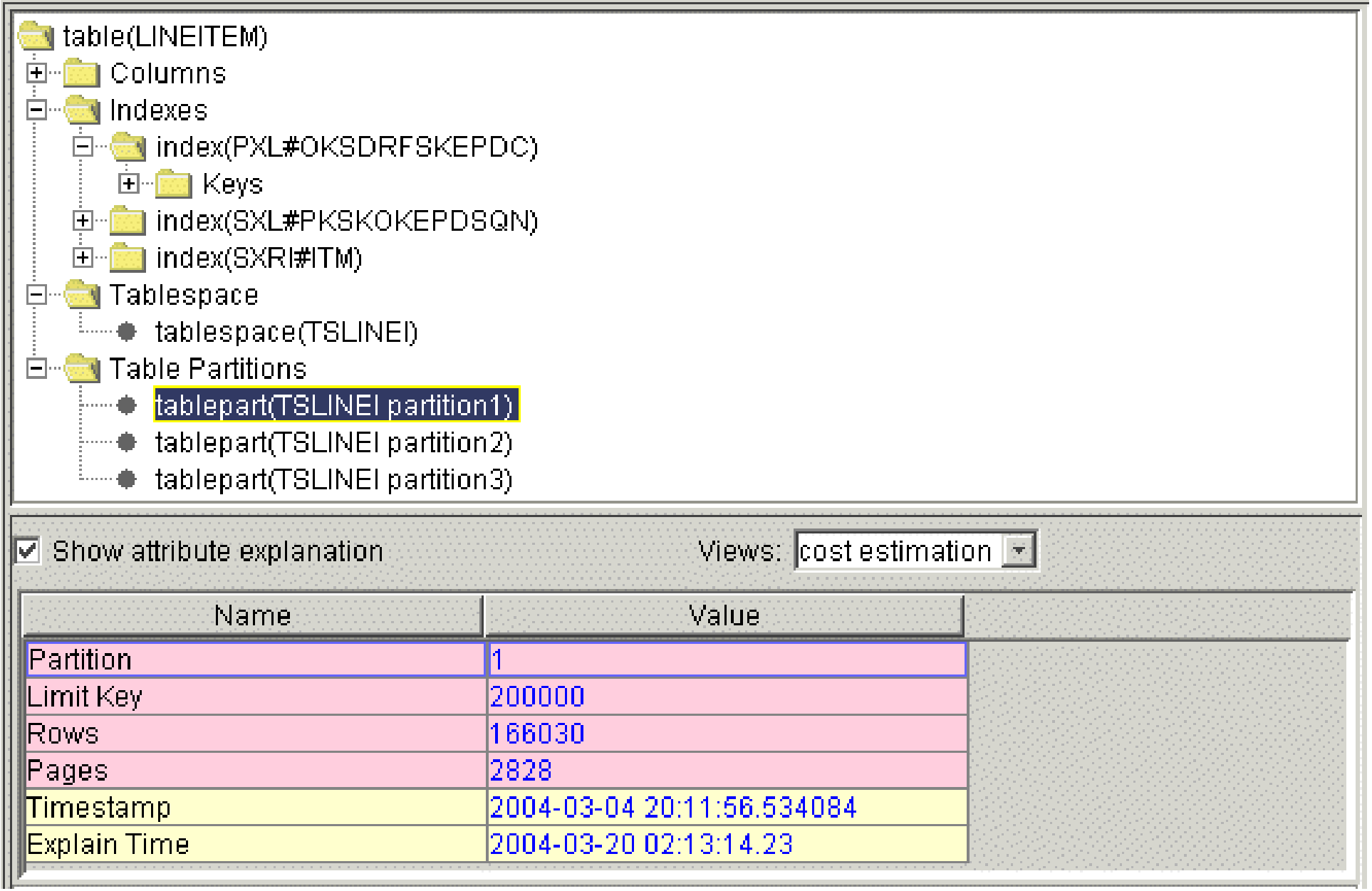
Access Plan



Predicate Processing and Filtering



Predicate Processing and Filtering



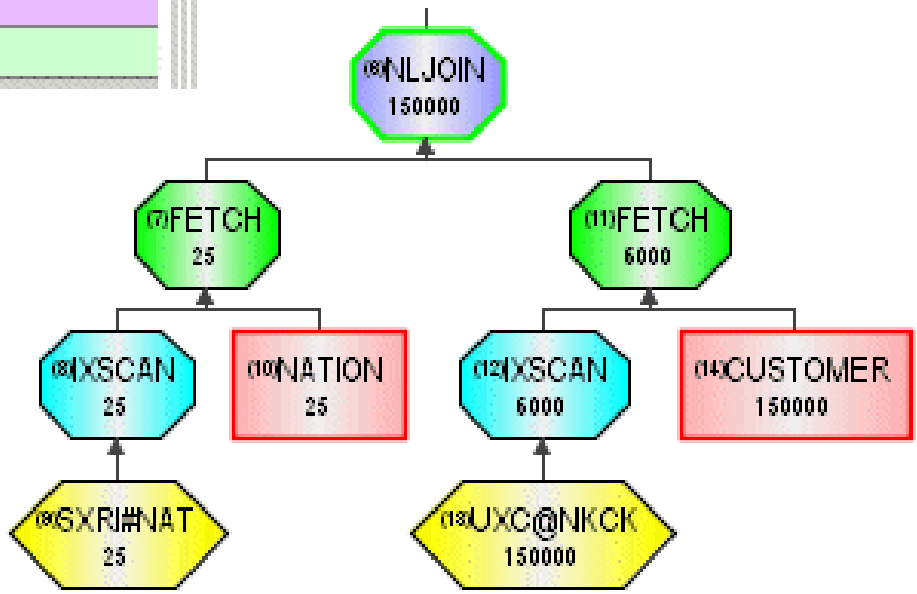
Nested Loop Join Details

Redbooks

join
Join Predicates
C.C_NATIONKEY=N.N_REGIONKEY

Show attribute explanation Views: cost estimation

Name	Value
Outer Input Cardinality	25.0
Inner Input Cardinality	6000.0
Join Predicates	Filter Factor
join.C.C_NATIONKEY=N.N_REGIONKEY	0.04
Output Cardinality	150000



Generating Reports