Evolution of Star join Optimization

DB2 UDB for z/OS and OS/390

White Paper

July 2002

Terry Purcell
Senior Consultant
Yevich, Lawson & Associates

Gene Fuh
Manager
DB2 OS/390 SQL Optimization
IBM Silicon Valley Lab

Yun Wang
Manager
DB2 OS/390 SQL Technology & Development
IBM Silicon Valley Lab

Yoichi Tsuji
Senior Software Engineer & Team Lead
DB2 OS/390 SQL Optimization
IBM Silicon Valley Lab

Eva Hu
Advisory Software Engineer
DB2 OS/390 SQL Performance
IBM Silicon Valley Lab

Version	Comment	Name	Date
1.0	Initial Version		October 2001
1.1	Updates to include examples of snowflake materialization, and APAR closing dates.	Terry Purcell	April 2002
1.2	Added Sparse Indexes on work files	Terry Purcell	July 2002

1 Introduction

The increased demand for Decision Support Systems (DSS) and Online Analytical Processing (OLAP) for business increases the complexity of database design, query construction and query optimization. The challenges for a Database Management System (DBMS) in supporting data warehouse implementations, and the millions (or billions) of rows of information stored, are being tackled aggressively by Database vendors as they attempt to assert dominance in this vastly growing field.

The concept of a data warehouse or data mart is evolving and perhaps expanding as the business grows. Regardless of how it evolves or expands, there are two major characteristics that require consideration: size and complexity.

Data warehouses often adopt the star schema data model as a method of minimizing the overhead of high volume data redundancy, by normalizing descriptive data into dimensions surrounding the large fact table. This white paper will attempt to address the IBM DB2 for z/OS and OS/390[1] implementation of star schema support and demonstrate IBM's commitment to DSS and OLAP.

2 Background

2.1 Star Schema

There are many kinds of data models in existence to support today's business requirements, but typically for data warehousing, there exists a number of highly normalized tables (known as dimensions or snowflakes) surrounding a centralized table (known as the fact table). This model represents a star schema, named as such due to the dimension tables appearing as points of a star surrounding the central fact table. Figure 1 displays a simple star schema design.

[1] All unqualified references to DB2 refer to DB2 for z/OS and OS/390.

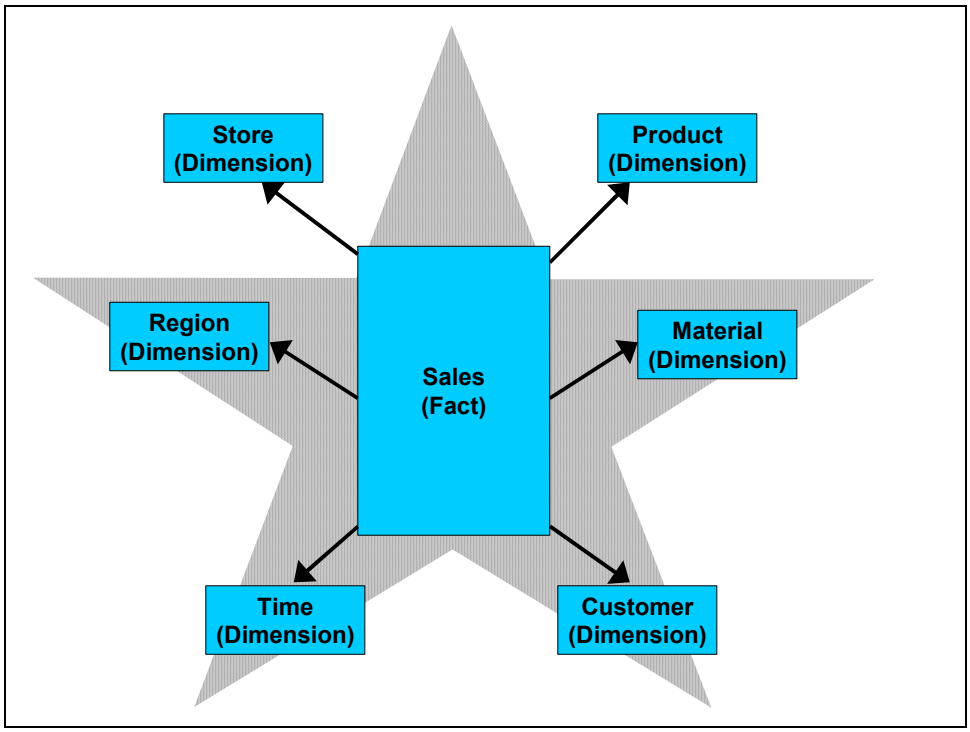

Figure 1. Star Schema

Further normalization of the dimension tables results in each dimension branch representing a snowflake schema design as demonstrated in Figure 2. In this paper, we will use the term "star schema" to also represent the "snowflake schema", unless we are required to explicitly distinguish between them.

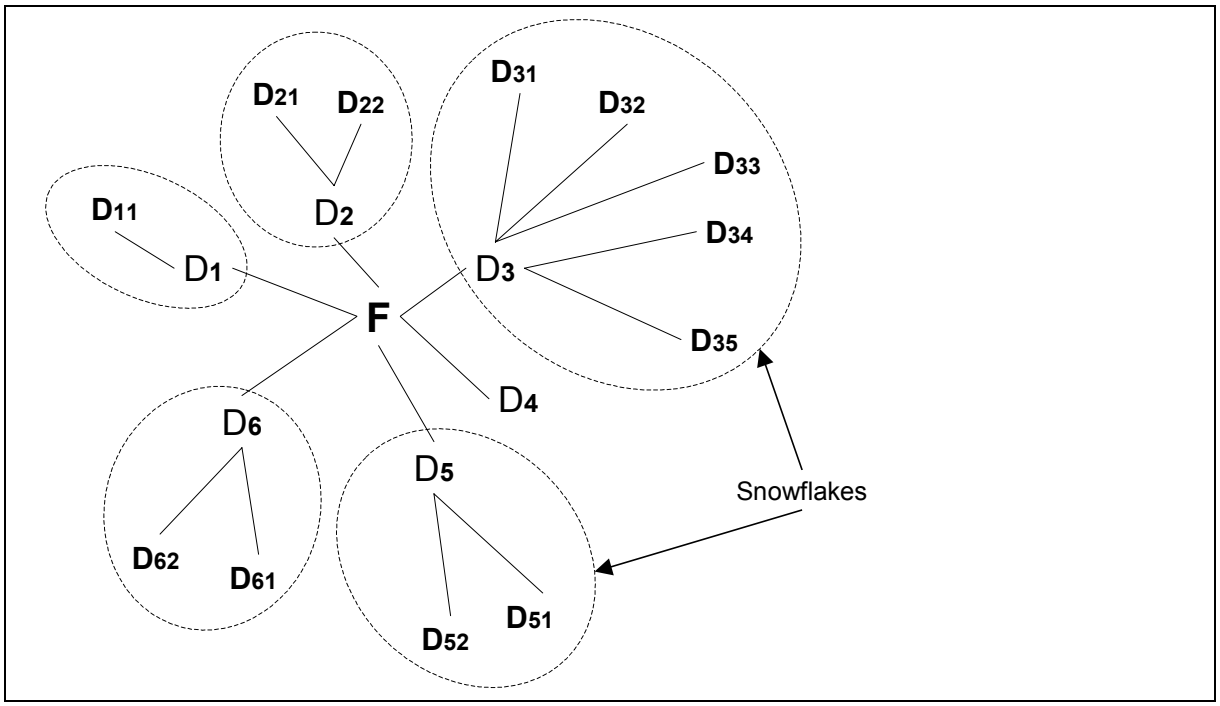

Figure 2. Star Schema incorporating multiple snowflakes

The attributes that generally dictate the star schema or snowflake schema database model are as follows:

- Highly Normalized
  - The dimension tables are highly normalized to avoid maintaining redundant descriptive data in the central fact table.
- Large Fact table
  - Fact table often contains sales type transactions that can be in the order of hundreds of millions, or billions of data rows.
- Relatively Small Dimensions
  - Highly normalized dimension tables contain a finite number of descriptions and detail information for the codes stored in the fact table.
- Sparse "Hyper Cube"
  - There is a high correlation among dimensions leading to a sparse nature of data in the fact table, eg. Product sales are dependent on the climate, therefore the sale of Bermuda shorts is more likely in the state of Florida than Alaska.
- Fact table is dependent on the dimension tables.
  - If the dimension tables exist due to normalization of repetitive fact table data, then there exists a parent-child relationship, with the dimension table as the parent and fact table as the child. There is no requirement however for an explicit foreign key relationship to be defined.

As the result of standard normalization, many data models may resemble a star schema due to the existence of a large table encircled by a number of smaller code tables. These databases can be operational OLTP or batch processing in nature. As will be discussed later, many challenges exist in determining whether the data model is in fact a star schema or simply a resemblance. This can have a major impact on the query optimization utilized.

2.2 Star Schema Queries

Unlike OLTP queries where a large number of short duration queries execute, OLAP queries involve a large number of tables and an immensely large volume of data to perform decision making tasks. Hence, OLAP queries are expected to run much longer than OLTP queries, but are less frequent.

In a purely normalized star schema design, the fact table does not contain attribute information, and merely contains event occurrences. To decode the fact table data, each row must be joined to the relevant dimensions to obtain the code descriptions.

Consequently, we can characterize a typical star schema query as having the following properties:

- Join predicates between the fact and dimension tables are equi-join predicates.
  - Decoding a fact table column requires an equal join to the relevant dimension table, matching on the fact table code value.
- Existence of local predicates on the dimension tables.

    - Assuming the attribute information has been normalized to within the dimension tables, selective predicates are applied to the dimensions rather than the fact table.
- Large number of tables participating in the query.
    - The star schema (and evidently the snowflake schema) dictates that many tables will participate in a join query.

2.3 Characteristics of Viable Access Paths

Considering the attributes of a star schema query, and the complexity of the associated data model, an efficient access path for such a query has three major objectives:

- Encourage a matching index scan of the fact table.
    - The large fact table cannot be scanned in its entirety, unless a large percentage of fact table rows will be retrieved, therefore matching index access must be available on as many selective join predicates as possible.
- Access enough dimension tables before the fact table to minimize the search space.
    - With the filtering provided by the intersection of dimensions, matching index access on as many fact and dimension join columns as possible will reduce the range of data rows that must be retrieved for each fact table access. This implies that there are selective local predicates applied to these dimensions to narrow down the search space.
- Determine the balance between increased fact table filtering and excessive Cartesian result.
    - Although increasing the selective dimensions may limit the qualifying fact table rows, this will increase the size of the Cartesian dimensions that must be joined to the fact table. For a Cartesian result of 10,000 rows, adding a further dimension with as little as 2 table rows will double the Cartesian result; thus doubling the number of rows to be joined to the fact table.

2.4 Technical Issues surrounding Viable Access Paths

The technical issues surrounding the optimization of Decision Support and Data Warehousing queries against a star schema data model can be broken down into either Bind-Time or Run-Time issues. First consideration generally is the sheer size of the fact table, and also the large number of tables that can be represented in the star or snowflake schema.

2.4.1 Bind-time issues

2.4.1.1 Time and Space Complexity

A significant bind issue for star schema processing is the problem of time and space. In today's cost-based optimization, we use the pair-wise join method to determine the join permutation order. This cost-based optimization is known as the dynamic programming technique. However, the number of join permutations grows exponentially as the number of tables being joined increases, and also the space requirement increases for the associated cost information. For example: a fully connected 15-table join constitutes 2**15 – 1 (32767)

possible join permutations. The associated cost information may exceed 100MB of storage for access path analysis, and for each additional table, the storage requirement increases by a factor of two[2].

Complex queries often cannot complete their bind process due to either: there are too many tables involved in the query (SQLCODE –129) or that there isn't enough space to finish the bind (SQLCODE –101).

As a consequence of the increased number of tables, the bind time also increases because of the time required to process those possible permutations.

2.4.1.2 Long Join Sequence

During bind, the cost-based optimizer provides an estimated cost of each join permutation and chooses the most optimal for each query. The accuracy of the cost estimation decreases as the number of join tables increases.

Since most star schema queries involve many tables, the optimizer is constantly challenged by these queries to determine the most optimal table join sequence.

2.4.2 Execution-time issues

When considering possible ways to join the tables in the query, the optimizer routinely avoids constructing Cartesian joins. This is because the absence of a join predicate results in the cross product of all rows in the two tables, generating a large intermediate result set at runtime. However, the alternative to Cartesian join, the pair-wise join, is also challenged to support the potentially large number of tables and rows required to be processed for a star schema query.

2.4.2.1 Pair-Wise Join

In an OLTP world, the query optimizer technology has been designed to handle pair-wise joins; joining tables one at a time based on join predicates and determining the join sequence based on the associated cost. A join pair is formed by the existence of join predicates between tables, either coded explicitly or derived by the optimizer based on other related join predicates (known as predicate transitive closure). With respect to a pair-wise join, tables without join predicates are considered to be unrelated and therefore not valid join pairs.

The pair-wise join method failed miserably though for star schemas. As dictated by the star schema data model, the only table directly related to other tables is the Fact table. As a consequence, the Fact table is most likely chosen in the first pair-wise join, with subsequent dimension tables joined based on cost.

---

[2] See Redbook: SG24-6129 DB2 for z/OS and OS/390 Version 7 Performance Topics, for V7 improvements in the space requirement of the optimization process at bind time.

Even though the intersection of all dimensions with the fact table can produce a small result, the predicates applied to one single dimension table are typically insufficient to reduce the enormous number of fact table rows. For example, a single dimension join to the fact table may find:

- 100+ million sales transactions in the month of December, or
- 10+ million sales in San Jose stores, or
- 10+ million sales of Jeans, but
- Only thousands of rows that match all three criteria.

Hence there is no one single dimension table which could be paired with the fact table as the first join pair to produce a manageable result set.

Given this scenario, then the intermediate result sets being carried from the first join pair are initially huge (and potentially unmanageable) and only diminish as further selective dimensions are joined. This type of join is contrary to run-time performance since there is an unnecessary cost of joining rows which should be filtered out as early as possible.

2.4.2.2 Run-time Alternatives to the Pair-Wise Join Approach.

Given the limitations with the pair-wise join for star schema processing, then the following criteria need to be considered if we are to ensure an efficient access path is available at run-time:

- "Selectively" join from the outside-in.
    - Purely cartesianing all dimension tables before accessing the fact table may not be efficient if the dimension tables do not have filtering predicates applied, or there is no available index on the fact table to support all dimensions. Consequently, the optimizer must consider selecting which dimension tables should be accessed before the fact table to provide the greatest level of filtering of fact table rows. The Cartesian overhead must be balanced by the selectivity. As the number of dimension tables accessed before the fact increases, so does the associated cost.
- Efficient "Cartesian Join" from the outside-in.
    - A physical Cartesian join generates a large number of resultant rows based on the cross product of the unrelated dimensions. A more efficient Cartesian type process is required as the number and size of the dimension tables increase to avoid an exponential growth in storage requirements.
- Efficient "join back"
    - The join back to dimension tables that are accessed after the fact table must also be efficient. Non-indexed or materialized dimensions (including snowflakes) present a challenge for excessive sort merge joins and workfile usage.
- Efficient access of the fact table
    - Due to the generation of arbitrary (or unrelated) key ranges from the Cartesian process, the fact table must minimize unnecessary probes and provide the greatest level of matching index columns based on the pre-joined dimensions.

3 Evolution of the Star Join Solution in DB2 z/OS and OS/390

The initial implementation of star schema support has been available since early releases of DB2. Prior to V6, this was known as "Early Cartesian Processing", and simply involved a physical Cartesian of unrelated dimension tables before joining to a larger table. Since this was a physical Cartesian, then the optimizer was not likely to choose this access path when the number of rows or number of dimension tables increased.

This "Early Cartesian Processing" access path is still available in V6 and beyond, and is based fundamentally on the dynamic programming join permutation strategy. It works well if there are no snowflakes, and the number of tables and rows are small. This access path did not scale however, to support the larger, more complicated star schema designs. Thus, a more sophisticated algorithm to support star schema processing was required.

The following sections detail the evolution of the V6 implementation of Star Schema support. Firstly the initial support is described, and subsequently all problems and solutions are logically grouped together and then presented in implementation sequence.

3.1 Star Schema Enhancement (APAR PQ28813)

V6 APAR PQ28813 introduced the Star Join access path into DB2 in July 1999. This was additional functionality that was not available in the initial GA release of DB2 UDB V6 for OS/390.

The challenge for the new Star Join access path would be in addressing the aforementioned problems with star schema processing for the standard cost based optimizer. These challenges are:

- Detecting the query as a candidate for the Star Join access path.
- Time and space limitations in bind process for large number of tables.
- "Selectively" join from the outside-in.
- Efficient "Cartesian Join" from the outside-in.
- Efficient access of the fact table
- Efficient "join back" to remaining dimensions

3.1.1 Star Schema Detection

The first challenge is that of detecting a star schema. Many data models resemble a star schema, consisting of a larger table surrounded by many smaller code tables. The problem for the optimizer is that the access path that specifically applies to a star schema may be inefficient if the query qualifies but does not assume the attributes of a true star schema.

To ensure the query is in fact a star schema query, then the following conditions must be met for DB2 to use the star join technique (original rules are outlined in APAR PQ28813, with subsequent additions identified with footnotes):

- The query references at least two dimensions.
- All join predicates are between the fact table and the dimension tables, or within tables of the same dimension.
- All join predicates between the fact and dimension tables must be equi-join predicates.
- All join predicates between the fact and dimension tables must be Boolean term predicates.
- No correlated subqueries cross dimensions.
- A single fact table column cannot be joined to columns of different dimension tables in join predicates. For example, fact table column "F1" cannot be joined to column "D1" of dimension table T1 and also joined to column "D2" of dimension table T2.
- After DB2 simplifies join operations, no outer join operations can exist between the fact and dimension tables.
- The data type and length of both sides of a join predicate are the same between the fact and dimension tables.
- The fact table contains at least 25 times the number of rows in the largest dimension table[3].
- Dimensions cannot be a table function.[4]
- A local predicate on a dimension table cannot be “OR”ed with a local predicate of another dimension table.[5]
- The query block must contain at least 10 tables (each table within a snowflake is counted separately)[6].

3.1.2 Snowflake Materialization

The snowflake data model can contain a large number of tables, which generally provides a challenge for cost based optimizers using dynamic programming (or exhaustive search) algorithms to identify the best possible table join sequence.

To simplify access path selection, DB2 always materializes each snowflake into a single dimension before determining the most cost effective join sequence as shown in figure 3.

[3] See subsequent APAR PQ36206 for how to alter the default value of 25
[4] Rule was not initially listed in APAR PQ28813 but was still applicable at initial implementation.
[5] As per footnote 3.
[6] See APAR PQ43846 and ZPARM SJTABLES for how to alter the default value of 10.

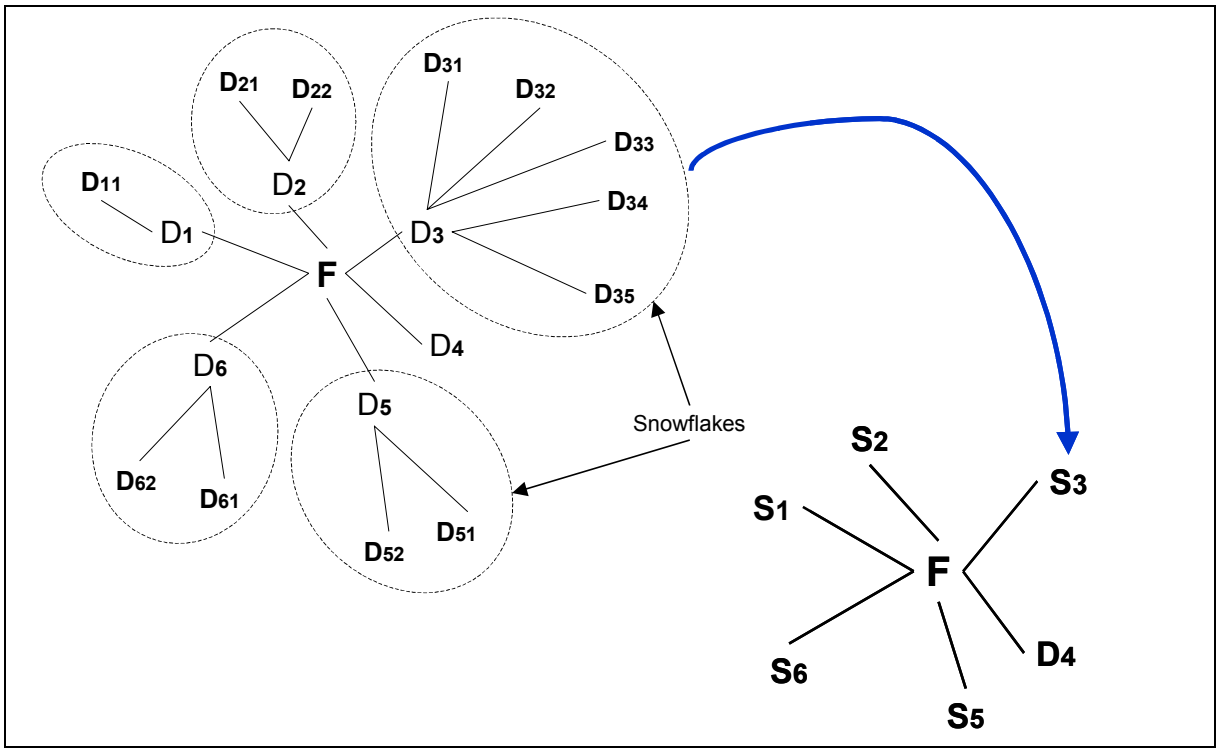

Figure 3. Snowflake materialization

Figure 4 provides the explain output showing the materialization of snowflakes in their separate query blocks (blkno 2 & 3), with one snowflake workfile accessed before, and one after the fact table.

	Qry No	QBlk No	Pln No	TableName	Jn Typ	Mthd	Acc Name	Acc Typ	Mtch Cols
1	10	1	1	PERIOD	S	0	PERIODIXP	I	0
2	10	1	2	REGION	S	1		R	0
3	10	1	3	STORE	S	1		R	0
4	10	1	4	**DSN_DIM_TBLX(02)**	S	1		R	0
5	10	1	5	FSALES	S	1	FSALESIX1	I	4
6	10	1	6	**DSN_DIM_TBLX(03)**		2		R	0
7	10	1	7	PRODUCT		1	PRODIXP	I	1
8	10	1	8			3			0
9	10	2	1	GCUSTOMER		0	GCUSTIXP	I	1
10	10	2	2	CUSTOMER		1	CUSTIX03	I	1
11	10	3	1	GMATERIAL		0	GMATIXP	I	1
12	10	3	2	MATERIAL		1		R	0

Materialized snowflake work files:
DSN_DIM_TBLX(02) : For a snowflake in query block 2
GCUSTOMER - CUSTOMER
DSN_DIM_TBLX(03) : For a snowflake in query block 3
GMATERIAL - MATERIAL

Figure 4. Plan table output for Snowflake materialization.

Similarly, single dimensions accessed before the fact table are sorted into join column order and also materialized into their own workfiles (although this does not appear explicitly in the plan table output). Single dimension tables accessed after the fact table are only materialized if required by the chosen join method.

3.1.3 Join Permutation Strategy

To solve the problem of time and space complexity in the bind process, and hence to overcome the 15 table limit[7] in a FROM clause, queries qualifying based upon all of the above criteria will bypass the dynamic programming join permutation strategy in favor of an outside-in (dimension to fact table) join directed by the available fact table indexes on the fact-dimension join columns.

Note: The following heuristic rules for determining join order have been superseded by APAR PQ43846.

Not all dimensions may be chosen for access before the fact table. Selecting the dimensions to process before the fact table access is based on heuristic rules that are applied to each fact table index in the following priority (a tie between two indexes is resolved based on the subsequent rule(s)):

1. The number of join columns covered by the index
2. The number of leading join columns covered by the index
3. Cluster ratio of the index
4. Arbitrarily chosen after preceding rules have been applied.

[7] See APAR PQ57516 for external ZPARM MXTBJOIN to increase the maximum number of tables allowed in a join.

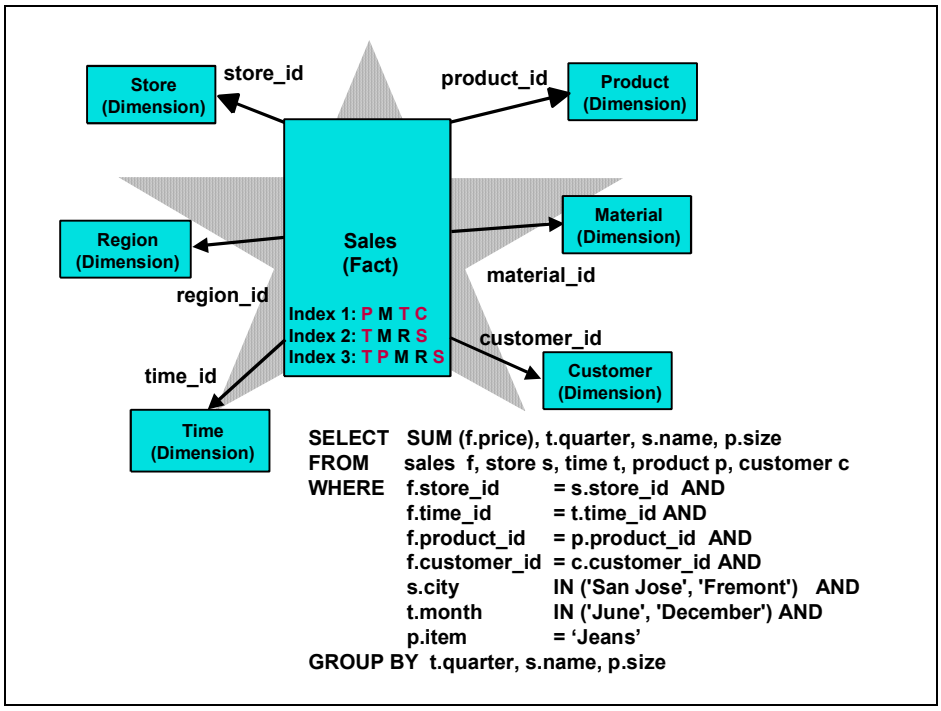

Figure 5. Fact Table Index Selection

The following applies the above heuristic rules to the example in figure 5:

1. The number of join columns covered by the index
   - Index 1 (P, M, T, C) contains 3 join columns (P, T, C)
   - Index 2 (T, M, R, S) contains 2 join columns (T, S)
   - Index 3 (T, P, M, R, S) contains 3 join columns (T, P, S)
   - *Index 1 and 3 qualify from rule 1.*
2. The number of leading join columns covered by the index
   - Index 1 contains 1 leading join column (P)
   - Index 3 contains 2 leading join columns (T, P)
   - *Index 3 is selected based upon the application of rule 2.*

3.1.4 Run-time Efficiency – Logical Cartesian Process via Index Key Feedback

For an efficient Cartesian process, DB2 employs a logical, rather than physical Cartesian of the dimension tables. Each dimension (or snowflake) covered by the chosen fact table index is accessed independently before the fact table. Each qualifying dimension (and snowflake) has all local predicates applied, with the result sorted into join column order, and finally materialized into it's own separate workfile.

If many of the dimensions involve snowflakes, then this preprocessing and materialization significantly reduces the number of overall tables joined, since the snowflake is resolved into a single dimension.

Rather than requiring the physical workfile storage involved in a physical Cartesian, DB2 simulates a Cartesian by repositioning itself within each workfile to potentially join all

possible combinations to the central fact table. The sequence of this simulated Cartesian join respects the column order of the selected fact table index.

The sparseness of data within the fact table implies a significant number of values generated by the Cartesian process are not to be found by a join to the fact table. To minimize the CPU overhead of joining unnecessarily derived rows to the fact table, DB2 introduces an index key feedback loop to return the next highest key value whenever a not-found condition is encountered.

A hit on the fact table index will return the matching fact table row. A miss will return the next valid fact table index key so that data manager can reposition itself within the dimension workfiles, thus skipping composite rows with no possibility of obtaining a fact table match. Figure 6 demonstrates the index key feedback technique.

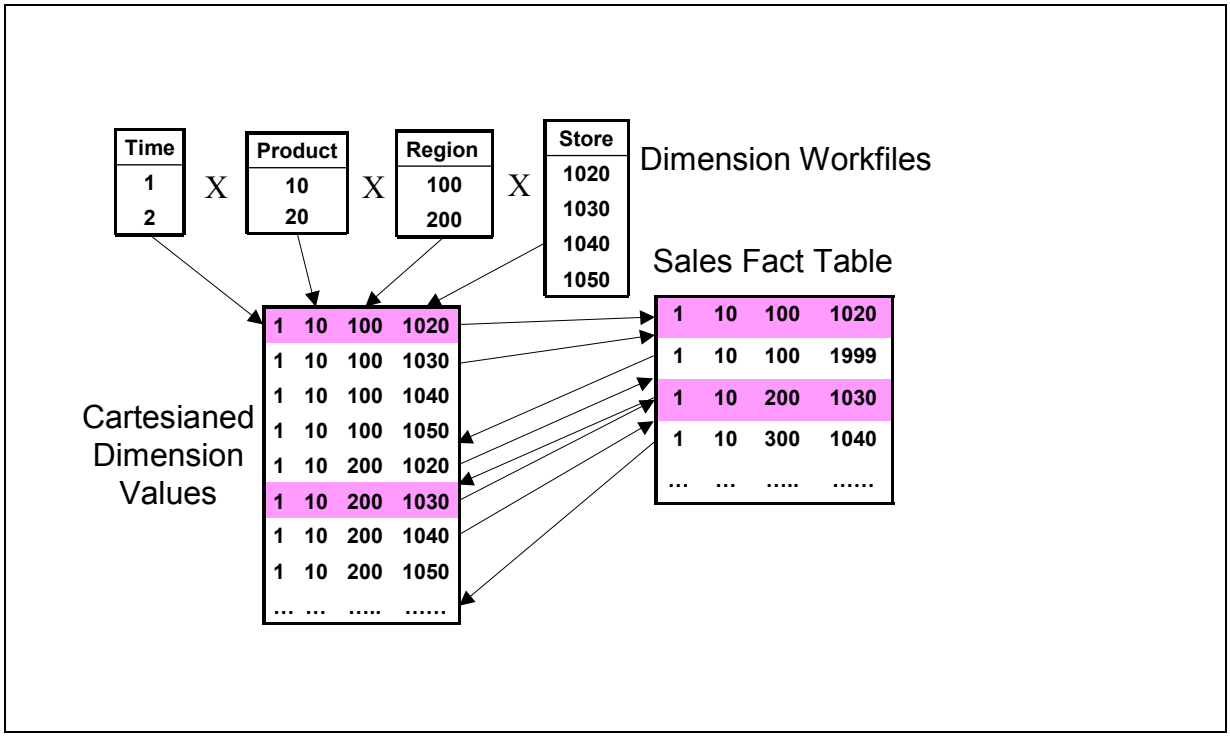

Figure 6. Index Key Feedback Loop

The following table summarizes the fact table index key probes and feedback opportunities from figure 6:

Cartesianed Dimension Key	Fact Table Key Status	Next Fact Key Feedback	Dimension Keys Skipped
1 10 100 1020	Matched		
1 10 100 1030	Unmatched	1 10 100 1999	
1 10 200 1020	Unmatched	1 10 200 1030	1 10 100 1040 –> 1 10 100 1050
1 10 200 1030	Matched		
1 10 200 1040	Unmatched	1 10 300 1040	

To further improve the performance of the join to the fact table, the entire join has been pushed down to data manager; but only for star join access from the composite to the fact table. This ensures a reduced path length since rows no longer need to be returned to RDS for the join to occur. The join method used by this process is a nested loop join.

3.1.5 The Join-Back Process

The join "back" from the fact table to the remaining dimensions can utilize the standard nested loop join method. This is the optimal join method due to the linear nature, and ability to avoid materialization of the intermediate result set. A nested loop join however requires the existence of dimension table indexes on the join columns to avoid repeated scans since these dimensions are the inner tables in a pair-wise join.

Note: The join sequence of dimension tables joined after the fact table has been superseded by APAR PQ43846. The requirement for merge-scan (sort merge) join due to the lack of indexes on workfiles has been addressed by APAR PQ61458.

If no index exists for a join to subsequent dimensions (or materialized snowflake), then the pre-joined dimension and fact table result is materialized to become the inner composite table in a merge-scan join with the remaining dimensions or snowflakes.

Figure 7 illustrates the concept of the pre-joined dimensions being materialized to become the inner composite table for a merge-scan join.

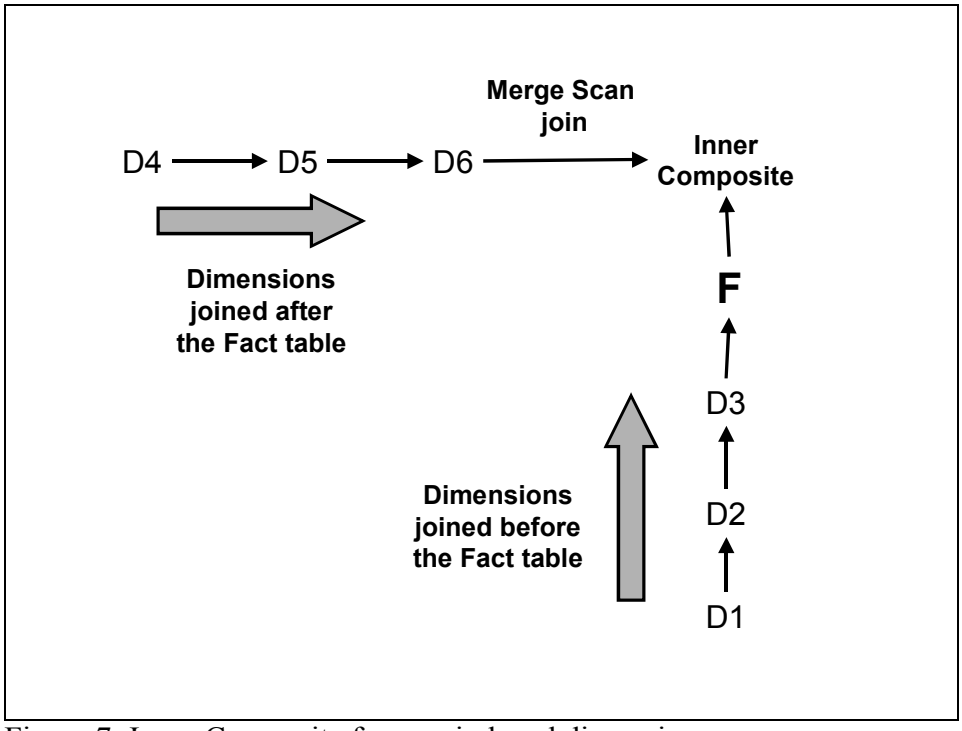

Figure 7: Inner Composite for non-indexed dimensions

3.2 Fact Table Detection

3.2.1 Cardinality rule failed to detect Star Join (APAR PQ33666)

Some queries that used the Star Schema enhancement encountered a SQLCODE –101 (The statement is too long or too complex) with a star join of more than 15 tables in a SQL statement. The query did not qualify for Star Schema support due to the rule that stipulates the fact table must have a cardinality at least 25 times that of the largest dimension.

Note: The following hidden ZPARM has been superseded by APAR PQ36206.

APAR PQ33666, available March 2000, addressed the above problem by providing a new hidden zparm (SPRMSJR) to allow users to manually set the star join ratio to increase or decrease the fact/dimension cardinality ratio rule according to their application needs. In addition, the new zparm provided the capability to disable star join if needed for performance reasons.

The possible values for SPRMSJR are:

Values	Descriptions
0	Enable Star Join (initial default)
-1	Disable Star join
1	Star Join, The Fact table will be the largest table in the Star join query. No Fact/Dimension ratio checking. More Star Join will be exploited if ratio is 1.
>1	This is the Star Join Fact table and the largest dimension root table ratio. The higher the ratio, the less chance the star join will be invoked

3.2.2 Simplify alteration of Star Join ratio (APAR PQ36206)

APAR PQ36206, available May 2000, introduces an externalized zparm to make it easier to change the star join parameter.

The new zparm, STARJOIN, is altered by adding the keyword STARJOIN=n, where n is:

Values	Descriptions
ENABLE	Enable Star Join
DISABLE	Disable Star join (default)
1	The Fact table will be the largest table in the Star join query.
2 - 32768	This is the Star Join Fact table and the largest dimension root table ratio. The fact table must be n times larger than the largest dimension table.

This zparm supersedes the previous implementation for the hidden zparm SPRMSJR. The default was also altered to be DISABLE, that is, not to use the newer star join techniques introduced by PQ28813 and the related APARs. When star join is disabled by this zparm, DB2 can still use the “early Cartesian join” technique.

3.2.3 Incorrect Fact Table Detection (APAR PQ43116)

In situations where the fact table did not have the highest cardinality in the star schema, then the incorrect table was chosen as the fact table. The application of the additional star join criteria subsequently failed in detecting the query as a candidate for star join processing. If greater than 15 tables existed in the FROM clause, then a SQLCODE –129 (The statement contains too many tables) was returned.

APAR PQ43116, available December 2000, introduced topology checking to detect the fact table if the cardinality rule fails. Before issuing a SQLCODE –129 due to too many tables, an attempt will be made to choose star join based on the table with the highest number of join predicates as being the central fact table.

3.2.4 Increase the number of tables required in a query block (APAR PQ43846)

Although the major impetus of APAR PQ43846, implemented May 2001, was the introduction of new heuristic rules for access path selection for star joins, there was an additional change to the rules for star schema detection.

The new detection rules requires that the query block must contain at least 10 tables for star join to be enabled for the query.

Subsequently, APAR PQ51765, implemented September 2001, introduced a new zparm SJTABLES to control this threshold value.

The possible values for SJTABLES are:

Values	Descriptions
0	Use the default value 10.
1-3	Star join is considered for all qualified queries by the other star join detection rules.
> 3	Star join is considered for qualified queries only when the number of tables in the star join query block (including the fact, dimension, and snowflake) is greater than or equal to the specified value.

In the context of the impact of APAR PQ43846 (discussed in more detail in section "Regression of existing queries due to requirement for inside-out processing"), this new rule may be considered minor, however, it is important to consider that failure of any of the star schema detection rules will disqualify the query in choosing the star join access path. Implementation of APAR PQ51765 allows greater control of this threshold.

3.2.5 Further Fact Table Detection Problems (APAR PQ49925)

In a minority of cases, both the cardinality rule and topology check (APAR PQ43116), have failed to correctly identify the fact table for a query against a star schema containing snowflake dimensions.

The topology check determines that the fact table has the highest number of join predicates in the query. If the fact table is incorrectly identified, then the fact table will be considered to be a dimension table (within a snowflake) to be materialized and potentially accessed early in the table join sequence. This will make the task of determining the optimal join sequence difficult, given that the heuristic rules are focused on fact table.

Figure 8 depicts a snowflake schema whereby the fact table (F) does not have the greatest number of join predicates in the query. In fact, the normalized dimension, DB, has a total of 13 joined tables, whereas the central fact table (F), contains only 7.

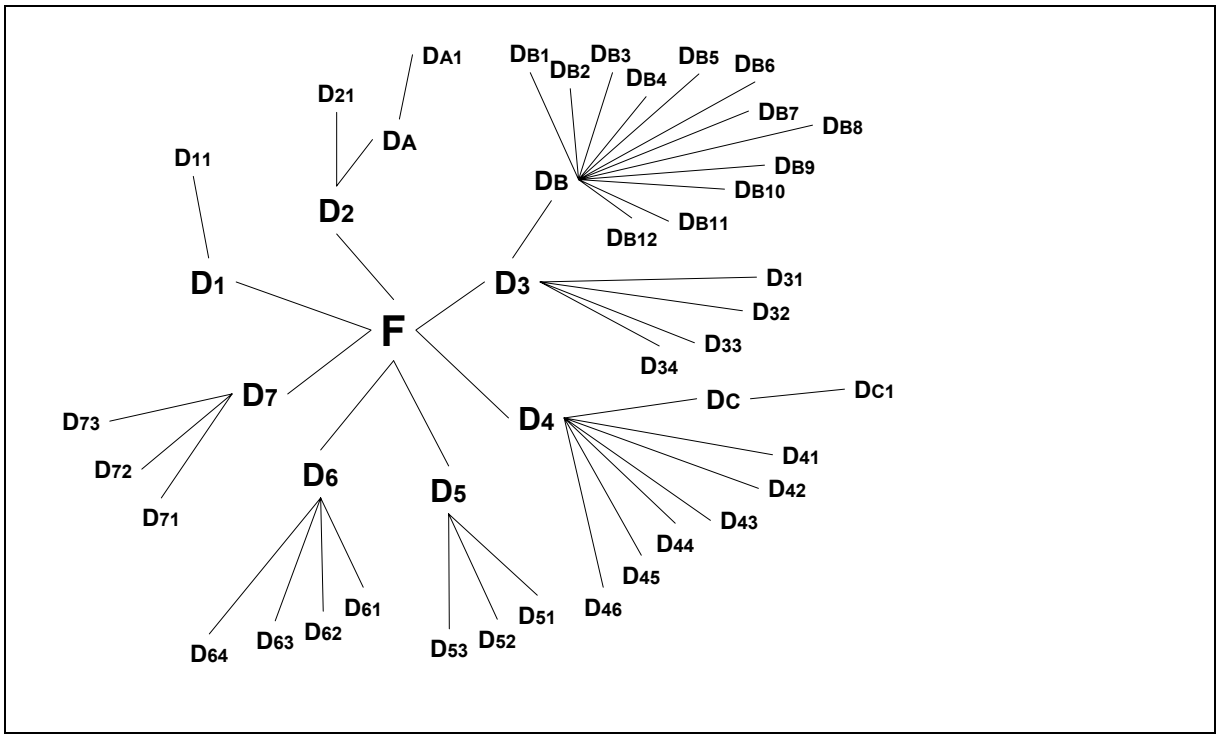

Figure 8. Complex Snowflake Schema

APAR PQ49925, available December 2001, provides a fix to resolve this rare situation. Beginning from the outside-in, the optimizer will evaluate the join predicates between each pair-wise join. For each set of join predicates between two tables, the table with a unique index on the join predicates is considered to be the parent in a parent-child relationship. As the optimizer continues from the outside-in, the table without any further children (and therefore only has parents) is considered to be the fact table.

This parent-child unique index detection will improve the fact table detection where previously the cardinality or topology checking had failed to interpret the correct star schema data model.

Figure 9 demonstrates the three levels of fact table detection that are now adopted to identify the query as a candidate for star join processing.

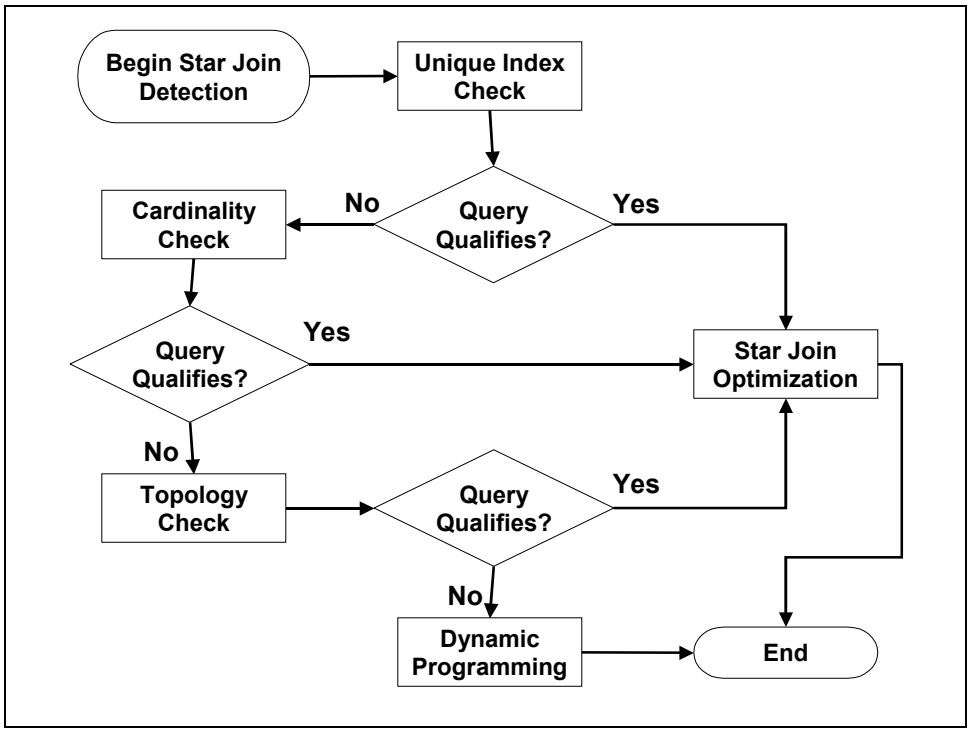

Figure 9. Fact Table Detection Flowchart

The first method used to detect the fact table is the Unique Index detection. If this method detects a fact table, then the remaining star schema detection rules are applied. These rules were identified in the section entitled "Star schema detection".

If the first check fails to identify a fact table, or this fact table fails against the subsequent rules, then the second fact table detection rule is applied. This second method is to evaluate the cardinality of the fact table and largest dimension against the STARJOIN zparm value. If a fact table is identified, then the remaining rules are applied.

The third and final method is the topology check. This is applied if the previous two methods have failed to qualify the query for star join processing. Once again, if a fact table is detected, then the additional rules are applied.

A successful match on any of the three methods will immediately qualify the query for star join optimization. A failure of all three will result in the query being optimized using the standard dynamic programming technique.

3.3 Minimize Regression with ability for inside-out processing (APAR PQ43846)

The initial star join implementation used heuristic rules, rather than dynamic programming, to generate the join sequences. When star join is enabled and a star schema is identified in a query, the optimizer evaluates a small subset of the possible join permutations, using the heuristic rules in order to save bind time. This could result in a join sequence having a higher cost compared with the optimal join sequence.

Queries which did little or no filtering on the dimension tables would not benefit from an early Cartesian of the dimensions. In this case an access path that accesses the fact table first (inside-out) would prove the most beneficial.

APAR PQ43846, available May 2001, completely restructured the possible join permutations offered by the star join access method. The V7 forward fit fix is APAR PQ47833. A pre-requisite APAR for this change to both V6 and V7 is APAR PQ46473 (May 2001).

The upper bound number of possible join permutations now becomes 4 * n + 4 (where n is the total number of index columns in all fact table indexes). This limited number of join permutations ensures that there is no exponential explosion in CPU and space consumption (time/space complexity) as the number of tables increases in the query.

3.3.1 First heuristic – Selective outside-in join

The first heuristic applies the following process to determine the potential join permutations:

- For each fact table index I(C1, C2, …., Cm):
    - For each possible prefix P(C1, C2, …, Cn), complete the join sequence of their corresponding dimensions D1 ⇨ D2 ⇨ … ⇨ Dn ⇨ Fact table, and all remaining tables joined based on the following four heuristics:
        - Remaining tables joined based upon the cardinality (lowest cardinality to highest).
        - Remaining tables joined based upon the cardinality (lowest cardinality to highest) with dimensions having an index on the join column selected first.
        - Remaining tables joined based upon the reduction ratio or selectivity.
        - Remaining tables joined based upon the reduction ratio or selectivity with dimensions having an index on the join column selected first.

3.3.2 Second heuristic – Inside-out join

The potential for regression for queries which did not meet the requirements for outside-in processing, but passed the star join detection rules would be catered for by this second heuristic that focuses on the fact table as the first table in the join sequence.

Beginning with the fact table as the first table in the join sequence, the second set of heuristic rules are then applied to determine the join permutations for all dimension tables:

- Remaining tables joined based upon the cardinality (lowest cardinality to highest).
- Remaining tables joined based upon the cardinality (lowest cardinality to highest) with dimensions having an index on the join column selected first.
- Remaining tables joined based upon the reduction ratio or selectivity.
- Remaining tables joined based upon the reduction ratio or selectivity with dimensions having an index on the join column selected first.

3.3.3 Cost Estimation for Reduced Join Permutations

After applying the first heuristic to all fact table indexes, DB2 considers the costing for each join permutation with index skipping and without. Although index skipping (also known as index feedback loop) reduces the CPU overhead of attempted probes into the fact table for invalid dimension combinations, it is unable to utilize list prefetch for data page access.

A list prefetch RID sort occurs in stage 2 (RDS). Considering the star join (before the fact table) is processed entirely within stage 1 (data manager), it is thus unable to invoke list prefetch. Poorly clustered indexes are therefore at a disadvantage since they require synchronous I/O to access the data pages. Consequently, indexes with a clusterratio of less than 50 percent are ignored for selection for the Star Join access path.

The index skipping technique also has the overhead of repeated scans of the dimension workfiles to accommodate the repositioning based upon next key feedback. Combining this with the requirement for all pre-joined dimensions to be materialized, then there is a definite tradeoff between increasing the number of dimensions joined before the fact table to increase fact table filtering, and the associated overhead that each additional table brings.

The first heuristic (outside-in) will produce a maximum of four access path alternatives for every leading key column combination (C1, C1/C2, C1/C2/C3 etc) of all fact table indexes. Each access path will be costed with and without the index skipping technique.

The second heuristic will only produce a maximum of four access path alternatives.

All possible table join permutations derived by the two main heuristic rules (regardless of duplicate access paths) will be then costed using formulas from the standard dynamic programming cost based optimization. This costing will also take into consideration the aforementioned overhead associated with many dimensions accessed before the fact table. The lowest cost access path will be chosen.

Only those access paths utilizing the outside-in processing and index skipping (and hence data manager push-down) will result in a star join (JOIN_TYPE = ‘S’) access path being identified in the plan table.

It is anticipated that the minimized join permutations derived from the aforementioned heuristic rules will determine the most efficient access path in majority of situations. The goal of reducing the possibility of query regression, whilst still reducing the bind time/space complexity is therefore achieved with this initiative.

3.4 Sparse Index on Snowflake Work Files (APAR PQ61458)

Although materialization of snowflakes reduces the total number of tables thus simplifying access path selection, this increase in work files can prove to be a performance bottleneck. Considering the inside-out join (from fact table outwards) is driven by the fact table, then a nested loop join involving a snowflake work file would require the entire work file to be

scanned as many times as there are rows in the outer composite table (fact table). Considering the fact table is typically still large after the application of restrictive dimensions, sort merge (merge scan) join is generally considered the most effective join method given the absence of indexing on these materialized work files.

A sort merge join usually requires a sort of both the new (snowflake work file) and composite (fact table) tables for the join. The fact table intermediate result is typically very large, making the sort a significant source of overhead. This composite result may need to be sorted many times to support multiple sort merge joins. In addition, each sort step forces the parallel group to merge thus minimizing the benefit of parallelism.

APAR PQ61458, available July 2002, addresses this bottleneck by introducing a sparse index on snowflake work files accessed after the fact table. This sparse index is constructed in order of the work file join key, thus allowing a nested loop join to replace the sort merge join and eliminate the sort of the large outer composite table. Both sort merge and nested loop join methods will be costed, with nested loop becoming a viable alternative due to the availability of the sparse index access on the inner table (snowflake work file). A nested loop join allows the parallel pipeline to continue rather than merged as required for a sort merge join.

Star Join access paths not involving snowflake access after the fact table are unaffected by this enhancement.

3.4.1 Details of Snowflake Work File Sparse Indexes

The sparse index is implemented as an in-memory structure within DB2's internal Dynamic Virtual Storage (DVS). The size of the structure can range from 8K to 240K bytes depending on the number of entries in the work file. Each entry of the index contains the sort key, 5-byte RID and approximately 1-2 bytes of overhead. If the total number of work file entries is small enough, then all entries can be represented in the index thus providing a one-to-one relationship between the index and work file. The index itself only becomes sparse if the number of index entries cannot be contained within the 240K byte maximum of the index.

The index structure is flat rather than a b-tree structure of standard indexes on data tables. The index is probed through an equal-join predicate and a binary search of the index is utilized to find the target segment. A sequential search of the work file is subsequently initiated within the target segment to find the corresponding join row. Currently, there is no possibility of index-only access from the work file index.

Explain output in the plan table will contain a 'T' in the ACCESS_TYPE column for a work file table on which the sparse index access method is chosen. The corresponding ACCESS TYPE of IFCID 22 will also contain 'T'.

With a nested loop join, it is expected that the work file buffer pool getpage count may show a substantial increase. This is due to the fact that the sort merge join requires generally one

sequential pass through the data, whereas a nested loop join invokes multiple probes into the work file for each outer table row. The best performance can be obtained by ensuring the work file bufferpool has enough space to contain the snowflake work file.

4 Summary

4.1 APAR Summary

The following tables list the relevant V6 and V7 APARs that have been referred to in this document. Additional prerequisite APARs are also listed:

APAR	V6 PTF	Date Available	Description
PQ28813	UQ33085	Jul. 1999	Original star join support
PQ33666	UQ40281	Mar. 2000	Prereq for PQ36206 (ZPARM update)
PQ36206	UQ42008	May. 2000	ZPARM update (Star join ratio)
PQ43116	UQ48813	Dec. 2000	Allow Star Join in certain cases to avoid SQLCODE -129 (star join detection based on join graph was added)
PQ44018	UQ49934	Feb. 2001	Additional fix on PQ43116
PQ46473	UQ52544	May. 2001	A prerequisite PTF for PQ43826
PQ43846	UQ53875	May. 2001	The new star join enhancement PTF for V6
PQ49925	UQ60214	Dec 2001	Improve Star Join Detection
PQ51765	UQ57573 UQ57574	Nov 2001	New Zparm SJTABLES

APAR	V7 PTF	Date Available	Description
PQ46473	UQ54337	May. 2001	A prerequisite PTF for PQ47833
PQ47833	UQ57153	Dec 2001	PQ43846 forward-fit to V7
PQ51765	UQ57575	Nov 2001	New Zparm SJTABLES
PQ61458	UQ67433	Jul 2002	Sparse Index on Snowflake Workfiles

4.2 Conclusion

IBM's commitment to Star Schema processing is evidenced by not only introducing the star join access method and it's corresponding optimization enhancements, but also by the rapid response to issues related to query regression or incorrect star schema detection.

The performance improvements delivered with the star join access path herald vast enhancements to the current DB2 optimization technology. There is still potential for further improvements to address some of the current exposures, and IBM is regularly evaluating industry trends and techniques employed by the non-OS/390 optimizer.

Any optimization techniques introduced to deal with the complexities of the Star and Snowflake schemas will, over time, be propagated to more access methods, providing enhanced performance to a greater array of queries from OLTP to Decision Support.

4.3 Bibliography

SG24-6108 DB2 UDB Server for OS/390 Version 6 Technical Update, IBM Corporation
SG24-6129 DB2 for z/OS and OS/390 Version 7 Performance Topics, IBM Corporation