

WHITE PAPER: THE BUSINESS VALUE OF INTELLIGENT DATA ACCESS

The Business Value of Intelligent Data Access

MARCH 2009

Sheryl M. Larsen

PRESIDENT AND FOUNDER OF SHERYL M. LARSEN, INC.

Table of Contents

Executive Summary	1
SECTION 1	2
Optimize database performance with streamlined data access	2
Index and MQT Design for the Workload Pattern	3
SQL Exploitation and Proper Use	3
Optimal Access Path for Data Access	6
Manual Query Rewrite when Necessary	7
SECTION 2	9
Conclusions	9
SECTION 3	9
About the Author	9
SECTION 4	11
CA Quick Reference for Access Paths, DB2 for z/OS	11

CA is proud to have sponsored the following paper by industry expert, Sheryl M. Larsen. The views, cases and best practices presented below are testimony of Ms. Larsen's extensive work supporting DB2 for z/OS within some of the world's largest, most complex enterprises.

This paper is a companion resource to a recorded webcast on this topic delivered by Ms. Larsen available at ca.com and an interactive guide (Adobe Flash) available from CA, "Access Paths for DB2 – A Self-Guided Technical Resource" which was a collaborative project between Ms. Larsen and CA. (<https://www.ca.com/us/Register/form.aspx?CID=173646>)

The interactive technical resource details all 16 access paths the DB2 9 for z/OS optimizer can choose. Access paths are shown in animated detail, including the newly enhanced Nested Loop Join and Direct Row Access, as well as a brand new join called Pair-wise Star Join. This resource can be used as a learning tool covering all available access paths, or leveraged as a quick reference tool for specific access path details and considerations.

Copyright © 2009 CA. All rights reserved. All trademarks, trade names, service marks and logos referenced herein belong to their respective companies. This document is for your informational purposes only. To the extent permitted by applicable law, CA provides this document "As Is" without warranty of any kind, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose, or noninfringement. In no event will CA be liable for any loss or damage, direct or indirect, from the use of this document, including, without limitation, lost profits, business interruption, goodwill or lost data, even if CA is expressly advised of such damages.

Executive Summary

CHALLENGE

HOW DB2 FOR Z/OS APPLICATIONS REQUEST AND OBTAIN DATA FROM LARGE, HIGHLY COMPLEX AND FREQUENTLY CHANGING DB2 DATABASES IS NOT WELL UNDERSTOOD BY MANY INFORMATION TECHNOLOGY PROFESSIONALS.

APPLICATION AND DATABASE TEAMS MAY FAIL TO CONSISTENTLY ADHERE TO BEST PRACTICES FOR DATA ACCESS DESIGN AND MAY LACK TIME AND EXPERTISE TO AUDIT AND TUNE ACCESS PATH METHODS.

PRESSURE TO QUICKLY DEVELOP AND DEPLOY NEW OR MODIFIED APPLICATIONS MEANS THAT MEDIOCRE DATA ACCESS DESIGN MAY BE TOLERATED AS LONG AS THE DATA RESULT SETS ARE ACCURATE. THIS UNNECESSARILY DRIVES UP COSTS AND CAN PUT ATTAINING SERVICE LEVEL AGREEMENTS AT RISK.

OPPORTUNITY

THE SIXTEEN ACCESS PATHS AVAILABLE IN DB2 FOR Z/OS EACH HAVE NOTABLE STRENGTHS AND LIMITATIONS.

A BETTER UNDERSTANDING OF THESE ACCESS PATHS, THEIR USE CASES AND SEVERAL PROVEN "RULES OF THUMB" WILL HELP DATA PROFESSIONALS SELECT AND TUNE INTERACTIONS BETWEEN APPLICATIONS AND DATABASES THAT YIELD BENEFITS TO THE ORGANIZATION EACH TIME DATA IS REQUESTED FROM DB2 FOR Z/OS BY AN APPLICATION.

BENEFITS

WELL DESIGNED AND MAINTAINED DATA ACCESS PATHS CAN ASSIST ORGANIZATIONS IN MEETING SERVICE LEVEL AGREEMENTS, IMPROVE APPLICATION AND DATABASE PERFORMANCE AND, REDUCE WORKLOADS ON HARDWARE, SOFTWARE AND SUPPORTING STAFF.

ONCE IDENTIFIED, DATA ACCESS PATH CHANGES CAN BE QUICKLY IMPLEMENTED TO YIELD IMMEDIATE SAVINGS FOR ORGANIZATIONS WITH LARGE, COMPLEX DB2 FOR Z/OS IMPLEMENTATIONS.

SECTION 1

Optimize database performance with streamlined data access

Database and access architectures are the core of all business information systems. Well-designed and maintained database and access architectures improve overall SLA attainment and delivers greater business value each time data 'flows' through the system.

For a wide range of reasons, designing for and maintaining optimal data access poses a genuine challenge to even the most sophisticated enterprises.

- Data access design crosses (or should cross!) IT functional teams including data modelers, database developers, database administrators and application developers; changes by one group affect the other groups
- Data access design is complex; there are many rules, more exceptions and critical best practices that need to be considered for each scenario
- Data infrastructures are dynamic – evolving and often out-growing initial design parameters and use cases
- Infrequent or insufficient data access audits mean that organizations are often unaware of changes that violate the rules, exceptions and best practices of data access design
- Pressure to develop and quickly deploy application changes can mean quality of data access design may be sacrificed to meet time-to-market objectives

How data is accessed can radically affect performance of applications. Today's developers and architects need to understand the most efficient way to access their data. This paper presents the top four data access best practices and illustrates how greater business value was derived in five specific business cases. Each of these cases began with a detailed data access audit and employed proven data access methods – leveraging SQL enhancements and access path optimization techniques which, in several cases, cut hours of processing to seconds.

As a DB2 consultant, I encounter a range of data access design and tuning situations that validate the importance of following the four best practices outlined below. From an application point of view, these include:

1. Index and MQT design for the workload pattern
2. SQL exploitation and proper use
3. Optimal access path for data access
4. Manual query rewrite when necessary

These rules, which will be covered in detail, are developed with intimate knowledge of the DBMS accumulated from real world data access assignments that begin with detailed data access reviews. Some of my assignments are centered on proactive performance reviews, when I help clients establish SQL review procedures and program design standards. Other assignments are centered on reactive performance reviews, when I help clients find and fix DB2 performance problems in their database, application or data access. Both types of assignments emphasize consistent adherence to DB2 performance rules.

Better data access results in improved CPU utilization which results superior performance and greater scalability. The cost of breaking these rules can usually be calculated as a CPU demand reduction or even dollars per CPU minute savings – metrics any CEO can understand, and which every IT manager worries about. This paper will demonstrate the role data access plays in terms of performance and scalability.

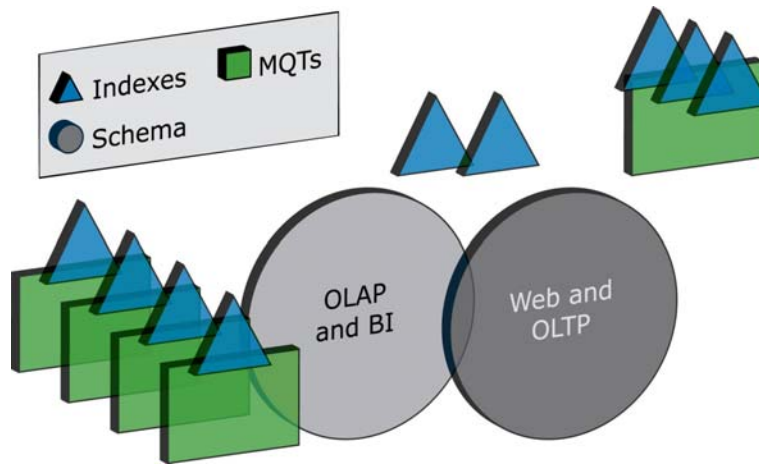
Index and MQT Design for the Workload Pattern

Scalability in DB2 requires smart, well-tuned SQL as well as schema design, performance structures, buffer pool, and storage that are optimized for the workload pattern. Breaking these rules costs CPU; the more data and transaction volume, the more CPU.

In a perfect world, all structured data would be stored in databases and the design would be optimal for the workload pattern. This paper addresses performance design considerations for one of those database types, DB2 for z/OS.

Figure 1 depicts different indexes and Materialized Query Tables (MQTs), for the Web and OLTP patterns versus the OLAP and/BI workload patterns. Differences exist because the data and processing requirements for these application types are different. The underlying schema should reflect those differences. Attributes, such as partitioning, clustering and normalization, are noted to be different in the circles of Figure 1, which show very little overlap in schema design. The triangles also show very little overlap in index design.

FIGURE 1



The demands caused by the data and data access patterns, both actual and anticipated, should dictate optimal design. Schemas and performance structures should reflect true needs for each environment. For example, short answer queries found in WEB and OLTP need indexing for probing and scrolling. The data requests of these workload patterns are usually customer or product centric. Therefore, the lead columns for the indexes are typically CUST_ID and PROD_ID. Each index is scrutinized for its performance savings and update degradation due to the high update rate. In contrast, long answer queries found in OLAP and BI could easily benefit from indexes and MQTs, with less scrutiny due to the low update rate. The data requests of these workload patterns are usually time-based and therefore the lead columns of the indexes are very different than the WEB and OLTP workload patterns.

SQL Exploitation and Proper Use

Data access in this perfect relational DBMS world is from SQL. Today's SQL is a very powerful, flexible and portable language. DB2 developers know that there are many ways to deliver accurate results for the same data request using SQL. The trick is to know which solution performs best for a given environment and architecture. Acknowledging that there are exceptions to the following practices, some SQL Performance Rules of Thumb include:

- Use joins over subqueries when detail row information is required
 - Why? Joins are more efficient than subqueries at combining data from multiple tables especially when the results need to include detail row information across more than one table. For example, give me a list of suppliers that supply red parts. List the supplier number, supplier name, part number, part color, part weight and in stock indicator. A join can efficiently pull this data together

- Use subqueries over joins when detail row information is not required
 - Why? Correlated subqueries are more efficient than joins at detecting the first hook up and ignoring the rest. For example, give me a list of suppliers that supply at least one red part. List the supplier number, supplier name. The correlated subquery can efficiently extract a unique list of suppliers that supply at least one red part. A join would have to join all red parts and then remove the duplicate suppliers with a DISTINCT clause.
- Use INNER JOIN over LEFT JOIN when exceptions are not expected or needed
 - Why? INNER JOINS provide data that qualifies "true" to the join predicate tests. Any data attached to "false" predicate tests are ignored. LEFT JOINS provide data that qualifies "true" and "false" to the predicate tests by supplying NULL for the row data that is "false" or does not join. These are the exceptions and cost more to test and deliver. If exceptions are not needed or expected, the INNER JOIN is faster.
- Use CREATE GLOBAL TEMPORARY TABLE when data is infrequently accessed
 - Why? CREATED TEMPORARY TABLES insert data faster than DECLARED and are efficient at delivering all the data stored in the table regardless of the amount of data.
- Use DECLARE GLOBAL TEMPORARY TABLE with a clustered index when DTT is large and data is frequently accessed
 - Why? DECLARED TEMPORARY TABLES allow indexes to be created for each session. This speeds up relational processing (join, subqueries, recursion) going against the table especially when the DTT is large and frequently accessed. Avoid DDTs when the DTT is small since it costs more to create the index structure than to scan the data.
- Promote Stage 2 predicates and non-indexable Stage 1s if possible (for a list of Stage 1 predicates, refer to Access Path Interactive Flash described in the introduction of this paper)
 - Why? Stage 2 predicates are handled at the last point of filtering in the DB2 engine. Recoding the predicate to adhere to the documented Stage 1 predicates, even if you need multiple Stage 1s, moves the filtering closer to the data and makes the query go faster. For example, value BETWEEN COL1 AND COL2 is Stage 2. Recoding it to be value >= COL1 AND values <= COL2 upgrades the conditions to Stage 1 which can be applied to an available index.
- SELECT only the columns needed
 - Why? SELECTing more columns than needed impacts throughput and resources. For example, retrieving extra columns from DB2 tables consumes unnecessary resources and that is just the beginning. If any queryblock has to be materialized, the extra width of the workfile slows down processing. This can be measured for materialized views, table expressions and any intermediate file needed for ORDER BY, GROUP BY, DISTINCT. The wider the workfile, the slower the process. SELECT only what needs to be seen.
- Do not SELECT columns with known static values
 - Why? Same reason as above. The predicates in the form of ColumnA = value, will provide the same value for each result row if the ColumnA is placed in the SELECT list. This is not a problem unless the query has any materialized views, table expressions and any intermediate file needed for ORDER BY, GROUP BY, DISTINCT.
- SELECT only the rows needed
 - Why? Every row requested is accepted by the calling program and puts extra strain on the network for distributed transactions. The calling program then must use resources to filter out unwanted rows.

- Use constants and literals instead of host variables if the values will not change
 - Why? Host variables are placeholders for wildly changing data used in predicates. These get filled in at run time for dynamic queries but are defaulted to predefined values, default filter factors, for static queries. These default filter factors may be far from reality compared to the run time value. When the run time value is static, a host variable will mask the reality. Supplying constants and literals keeps the cost calculation real.
- For high volume (transactions or rows) sequence filtering from most restrictive to least restrictive by table, by predicate type
 - Why? There is a documented filtering order within DB2 as follows:
 1. Indexable
 - a. In order of index columns, one column past the last = condition
 2. Stage 1 and on index page (index screening)
 - a. = first
 - b. then range conditions
 - c. LIKE
 - d. noncorrelated subqueries
 3. Stage 1 on data page rows that pass prior tests
 - a. = first
 - b. then range conditions
 - c. LIKE
 - d. noncorrelated subqueries
 4. Stage 2 on rows that pass prior tests
 - a. = first
 - b. then range conditions
 - c. LIKE
 - d. noncorrelated subqueries
 - e. correlated subqueries
- This is the order predicates are processed regardless how they are typed in the query. The reason for this rule is that there may be ties within each category. In this case, the order typed in is relevant so it makes sense to first type in the most restrictive predicate within each predicate type.

SQL performance rule violations will increase CPU & I/O demand and cost time to fix which carves money right out of the bottom line of any business. How much money? I'll share a few anonymous client reactive performance reviews to demonstrate.

Client #1 was approaching a critical threshold of CPU usage and suspected that non-optimal SQL was their main problem. The data access assessment revealed that 95% of their 403 DB2 Stored Procedures needed to be redesigned due to SQL Performance Rule violations. The main rule broken was:

- Use INNER JOIN over LEFT JOIN when exceptions are not expected or needed

LEFT/RIGHT JOINS are usually more expensive than INNER JOINS on the same tables due to the fact that the answer set includes exceptions, rows in one table that do not hook up to the other table. If those exceptions contain critical information, then by all means, use LEFT/RIGHT JOINS. In most of Client #1's reporting applications, the exceptions were either not there or not needed. This cost Client #1 5% CPU per report execution (380 programs *

10 daily executions). After correcting, in one week, CPU consumption was reduced to manageable levels.

Client #2 was also approaching a critical threshold of CPU usage in addition to extremely high I/O wait times. The sequential I/O growth (30.7% increase), was outpacing the business growth, (18.7% increase). The data access assessment revealed sub-optimal SQL in the mission critical workload executing 60,000 times a day. The main SQL Performance Rule violation was:

- Promote Stage 2 predicates if possible

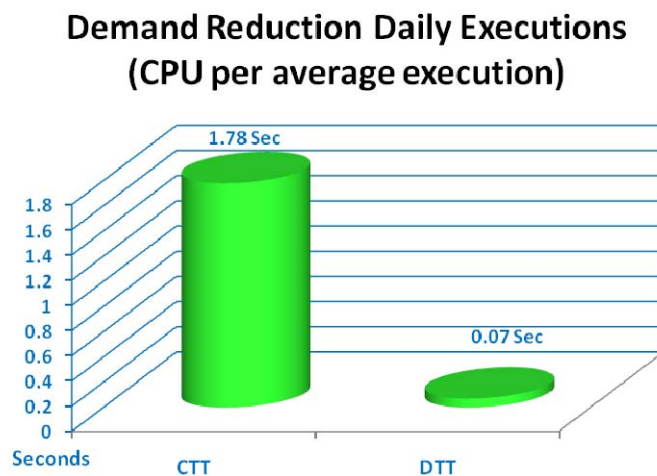
The non-optimal SQL consisted of 270 Stage 2 predicates in the format of WHERE value BETWEEN Col1 AND Col2. These were converted to range predicates which are indexable Stage 1 predicates. After correcting, Client #1 reduced CPU consumption by 9% as measured on a daily basis.

Client #3 outsourced application development of Project X to reduce time-to-market. Unfortunately, the developers assigned to Project X did not have a good understanding of the range of transactions that needed to be considered. As a result, they created a data access architecture that assumed all transactions were equal in priority. To make matters worse, there were no performance reviews for high volume data extracts. When implemented, this application consumed 25% of a new z/9 on a daily basis. Early in the analysis, I discovered misuse of SQL. The SQL Performance Rule violated was:

- Use DECLARE GLOBAL TEMPORARY TABLE with clustered index when DTT is large and data is frequently accessed

One particular transaction was the top CPU consumer by far. This extract transaction used Created Temp Tables (CTT) instead of Declared Temp Tables (DTT) for medium to large amounts of data. The main difference between the two types of temporary tables is indexes; CTTs do not allow them but DTTs do. These CTTs were nested within ten recursive SQL statements. These temporary tables contained, on average, at least 10,000 rows per execution. Once converted, a clustering index was added for each DTT. After correction, the average CPU time dropped significantly as depicted in Figure 2 below.

FIGURE 2



With this small coding change, the 6,000 daily executions resulted in a total daily savings of 2.8 CPU hours Monday through Friday!

Optimal Access Path for Data Access

As stated in the beginning of this white paper, how the data is accessed can radically affect performance of applications. Intimate knowledge of DB2's access paths is necessary to do

proper data access reviews. CA provides a free online DB2 Access Path Tutorial (<http://www.ca.com/us/demos/collateral.aspx?cid=185950>) to help those who want to be proficient data access reviewers. Lack of skill in this area can cost CPU and I/O. I will share a few more client performance reviews to demonstrate.

Client #4 was rapidly approaching a CPU upgrade as data volumes increased for a financial reporting application. After learning from the team that the application was developed without data access reviews for any of the 3,000 static programs, I ran queries against the PLAN_TABLE. This revealed 5,000 table scans and 6,000 sorts of some very large tables. Reports executing the most frequently were tuned. One report accessed 900,000 pages versus 8,000 pages with tuned access path. The total CPU savings from the tuning effort is shown in Figure 3 below.

FIGURE 3



Client #5 had a mission critical application that was experiencing greater workloads and, as a result, hurting performance. During analysis I discovered many expensive access paths within the application's workload. Specifically there were:

- 989 List Prefetches
- 145 Hybrid Joins
- 172 Multiple Index Accesses
- 126 Index Scans
- 325 Table Scans

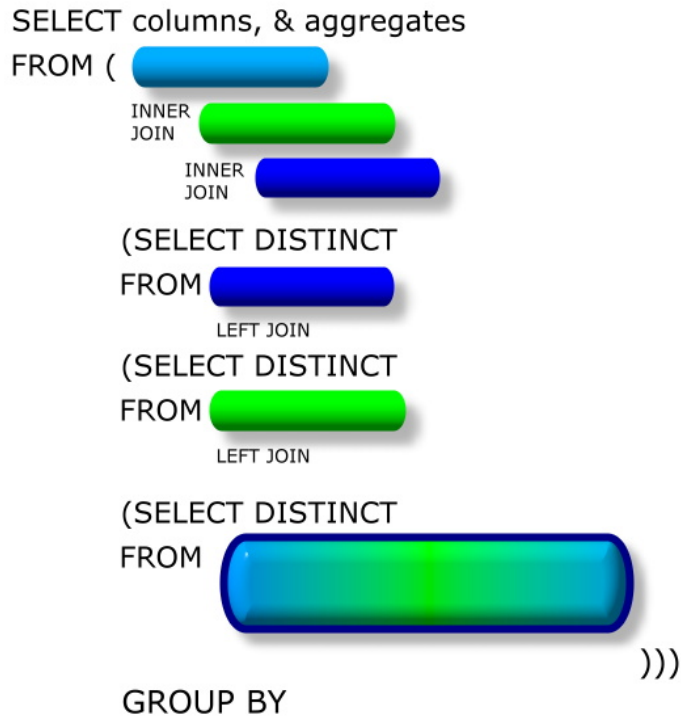
These particular access paths would be easily justified if the entire result set were read through to the end or if the tables being accessed were very small. Unfortunately, the majority of access paths did not fall into either category. Many indexes were altered, some were added. And, many limited fetch and OPTIMIZE clauses were added to queries. The result was a drastic reduction of CPU with one program in particular saving the client \$153K annually. This figure is calculated using the client's cost per CPU second.

Manual Query Rewrite when Necessary

Tuning SQL is usually easy using methods such as OPTIMIZE FOR 1 ROW, no operations, REOPT(AUTO), as well as additional methods, such as fake filtering and DISTINCT table expressions. However, sometimes drastic situations demand extreme measures. One extreme measure would be looking across query blocks to make decisions about the sequence of data access and GROUP BY work. The good news is this technology is starting to show up in the DB2 optimizer during its own query rewrite phase. As of DB2 9, it's called global optimization. In general, it is the process of looking across the whole query, across subselects and subqueries, in order to optimize the full select. An example comes from Client #5, who had a stored procedure that would not finish even after 49 hours of elapsed time. This stored procedure was started on Friday and I was presented this stored

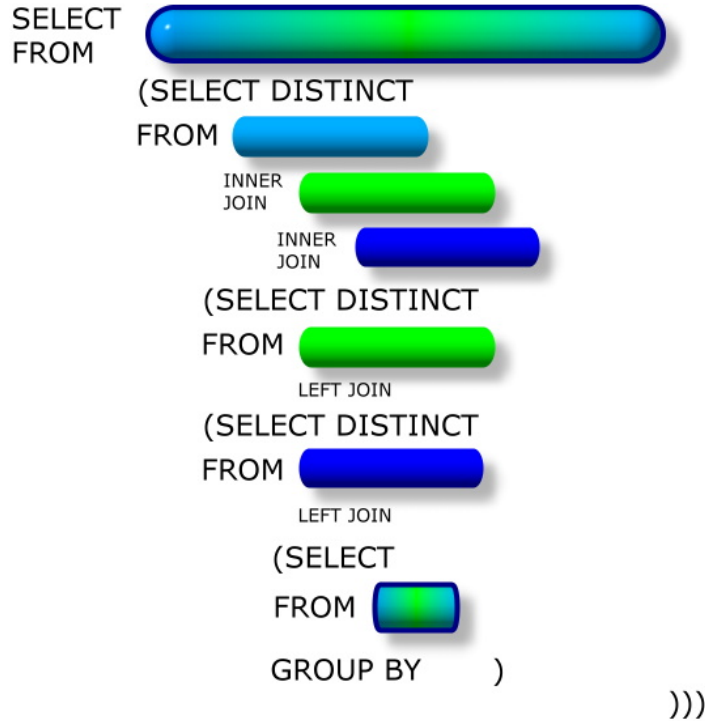
procedure on the following Monday morning. The main query had four query blocks that can be seen in Figure 4.

FIGURE 4



The inner-most query was retrieving data from a table containing 350 columns which was then joined to five more tables before GROUPing and calculating a MAX, AVG and STDEV. The DB2 optimizer could not rewrite this statement due to the complexity and all the DISTINCT references within table expressions. A person, on the other hand, could read across all the query blocks and determine that the GROUP BY work was needed on only one table, the biggest one. This approach enables a manual query rewrite, by a human, to push the GROUP BY work into the inner-most query block. IBM calls this GROUP BY push down which is coming to DB2 on z/OS but not too soon. Grouping is a form of compression reducing the detail row to summary rows. I took it one step further by compressing the SELECT list down to only the columns needed for grouping, aggregates, and future join conditions. This dramatically changed the first query block and the number of rows joined back to the five tables but did not produce the desired result. An additional query block was added on top to join back to the 338 columns as shown in Figure 5.

FIGURE 5



This manual query rewrite completed in two minutes. The IBM DB2 optimizer and query rewrite teams, both z/OS and LUW, are headed this way so that some day, every GROUP BY query can have this sophisticated rewrite technique, no matter how complex.

Today's developers/architects need to understand the most efficient way to access their data and breaking DB2 performance rules costs companies money. If you are serious about improving performance and designing for scalability, you will make sure your staff has intimate knowledge of the DBMS of deployment, regardless of development architecture or rapid application development methodology. This is especially important in high volume, high transaction rate applications where performance and scalability can be extremely costly.

SECTION 2

Conclusions

Intelligent data access that generates business value is achieved by leveraging SQL enhancements and access path optimization techniques which can cut hours of processing, sometimes days, to minutes or seconds. Well considered changes can provide companies with agility and scalability many data professionals (and business executives!) only dream of. In the future, the DB2 query re-write engine and optimizer may well take care of many of the performance rules mentioned in this document. In the meantime, code carefully.

SECTION 3

About the Author



Sheryl M. Larsen is president and founder of Sheryl M. Larsen, Inc., a consulting practice specializing in DB2 Systems Analysis and Advanced Education. She is an internationally recognized researcher, consultant and lecturer, specializing in large scale DB2 systems and is known for her extensive expertise in optimizing SQL. Sheryl has over 23 years experience

in DB2, has published many articles, and co-authored a book, DB2 Answers, Osborne-McGraw-Hill, 1999. Currently, she is President of the Midwest Database Users Group www.mwdug.org, a member of IBM's DB2 Gold Consultants program and just announced an inaugural member of IBM's Data Champion program www.ibm.com/software/data/champion/. She was voted into the IDUG "Speaker Hall of Fame" in 2001 and was the Executive Editor of the IDUG Solutions Journal magazine 1997-2000.

SECTION 4

CA Quick Reference for Access Paths, DB2 for z/OS

Excerpts below are from the white paper: "The Business Value of Intelligent Data Access; Data access is the key to optimize database performance" and from the CA Interactive Access Path Technical Resource available at www.ca.com/db.

The interactive technical resource details all 16 access paths the DB2 9 for z/OS optimizer can choose. Access paths are shown in animated detail, including the newly enhanced Nested Loop Join and Direct Row Access, as well as a brand new join called Pair-wise Star Join. This resource can be used as a learning tool covering all available access paths, or leveraged as a quick reference tool for specific access path details and considerations.

Index	Join
One Fetch	Nested Loop Join
IN(list) Index Access	Merge Scan Join
Matching Index Access	Hybrid Join Type N
Sparse Index Access	Hybrid Join Type C
Non-matching Index Access	Star Join Cartesian
List Prefetch	Star Join Pair-wise
Multiple Index Access	
Table	Partitioned Table
Table Scan	Limited Partitioned Scan Using Clustered Partitioned Index
Partitioned Table Scan	Limited Partitioned Scan Using Non-Clustered Partitioned Index (NPI)
In-memory Cache Scan	Limited Partitioned Scan Using Data Partitioned Secondary Index (DPSI)
Limited Partition Scan	

SQL Exploitation and Proper Use – Rules of Thumb

1	Use joins over subqueries when detail row information is required
2	Use subqueries over joins when detail row information is not required
3	Use INNER JOIN over LEFT JOIN when exceptions are not expected or needed
4	Use CREATE GLOBAL TEMPORARY TABLE when data is infrequently accessed
5	Use DECLARE GLOBAL TEMPORARY TABLE with a clustered index when DTT is large and data is frequently accessed
6	Promote Stage 2 predicates and non-indexable Stage 1's if possible
7	SELECT only the columns needed
8	Do not SELECT columns with known static values
9	SELECT only the rows needed
10	Use constants and literals instead of host variables if the values will not change
11	For high volume (transactions or rows) sequence filtering from most restrictive to least restrictive by table, by predicate type