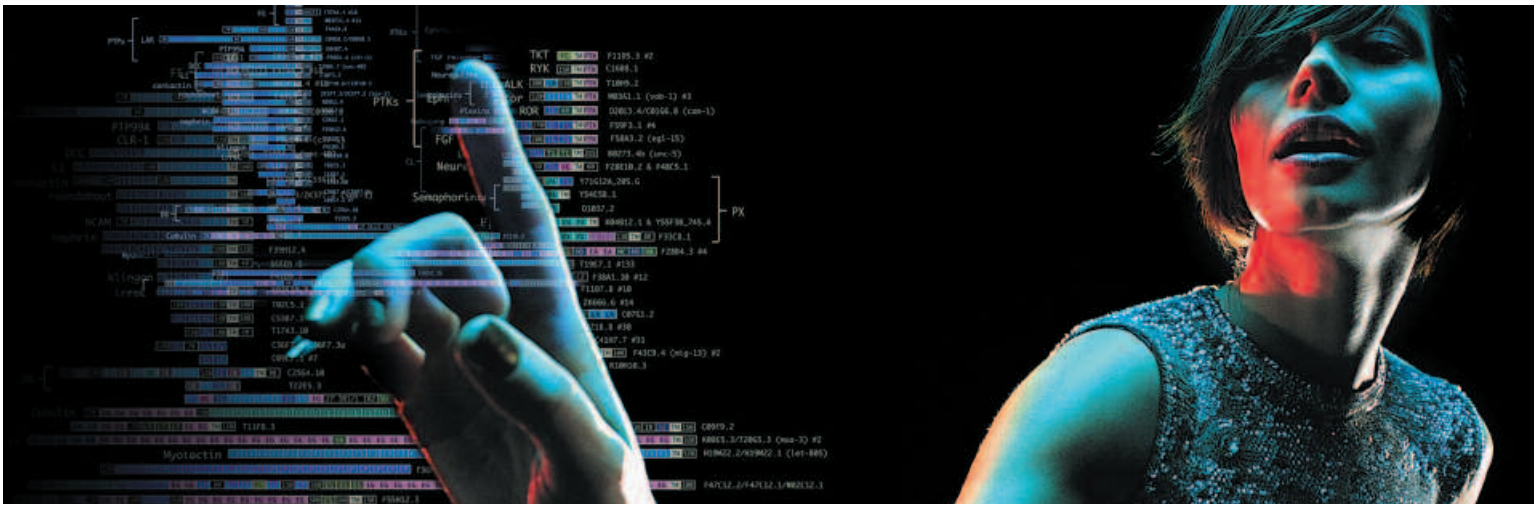


Tune That SQL for Supercharged DB2 Performance!

Craig S. Mullins,
Corporate Technologist, *NEON Enterprise Software, Inc.*

white paper

NEON
Enterprise Software, Inc.

Table of Contents

white paper

Overview

Structured Query Language, better known as SQL, is a powerful tool for manipulating data. It is used in virtually every relational database management system on the market today, not just by DB2, but also by Oracle, Sybase, mySQL, and Microsoft SQL Server.

SQL is a high-level language that provides a greater degree of abstraction than do procedural languages. Most programming languages require that the programmer navigate data structures. The navigation information is encoded in the program and is difficult to change after it has been programmed.

SQL is different. It is designed so that the programmer can specify what data is needed, and not how to retrieve it. A DB2 application programmer will use SQL to define data selection criteria. DB2 analyzes SQL and formulates data-navigational instructions "behind the scenes." These data-navigational instructions are called access paths. By having the DBMS determine the optimal access path to the data, a heavy burden is removed from the programmer. The database has a better understanding of the state of the data it stores, and thereby can produce a more efficient and dynamic access path to the data. The result is that SQL, used properly, can provide for quicker application development.

Quick application development is a double-edged sword. While it can mean reduced application development time and lowered costs, it can also mean that testing and performance tuning are not thoroughly done. The task of tuning the database as well as optimizing the SQL typically falls to the database administrator (DBA).

The DB2 environment and its host system can be tuned to achieve a certain level of performance improvement, but the greatest potential for performance improvement comes from analyzing the SQL code itself and making changes to improve speed and efficiency. The consensus among SQL performance experts is that 80% or more of database performance problems are caused by improperly written and un-tuned SQL.

SQL Query Tuning

SQL is not merely a query language. It can also define data structures, control access to the data, and insert, modify, and delete data. Consolidating these functions into a single language eases communication between different types of users.

SQL is, by nature, quite flexible. It uses a free-form structure that gives the user the ability to develop SQL statements in a way best suited to the given user. Each SQL request is parsed by the DBMS before execution to check for proper syntax and to optimize the request. Therefore, SQL statements do not need to start in any given column and can be strung together on one line or broken apart on several lines. Any SQL request could be formulated in a number of different but functionally equivalent ways.

SQL's flexibility makes it intrinsically simple, but flexibility is not always a good thing when it comes to performance. Different but equivalent SQL formulations can result in extremely variable performance. In this section, we'll talk about some of the tools within DB2 to help optimize performance and we'll get into some of the things to watch for in the code itself.

Queries Built for Speed

When you are writing your SQL statements to access DB2 data, keep in mind the three fundamental guidelines listed in this section. These are simple, yet important rules to follow when writing your SQL statements. Of course, SQL performance is a complex topic and to understand every nuance of how SQL performs can take a lifetime. That said, adhering to the following simple rules puts you on the right track to achieving high-performing DB2 applications.

white paper

1. **Always provide only the exact columns that you need to retrieve in the SELECT-list of each SQL SELECT statement.**
   Another way of stating this is "do not use SELECT *". The shorthand SELECT * means retrieve all columns from the table(s) being accessed. This is fine for quick and dirty queries but is bad practice for inclusion in application programs because:
   - DB2 tables may need to be changed in the future to include additional columns. SELECT * will retrieve those new columns, too, and your program may not be capable of handling the additional data without requiring time-consuming changes.
   - DB2 will consume additional resources for every column that is requested to be returned. If the program does not need the data, it should not ask for it. Even if the program needs every column, it is better to explicitly ask for each column by name in the SQL statement for clarity and to avoid the previous pitfall.

2. **Do not ask for what you already know.**
   This may sound simplistic, but most programmers violate this rule at one time or another. For example, consider what is wrong with this simple query:

```
SELECT LASTNAME, FIRST_NAME, JOB_CODE, DEPTNO
FROM           EMP
WHERE          JOB_CODE = 'A'
AND            DEPTNO =  'D01';
```

Look at the SELECT-list. There are four columns specified but only two of them are needed. We know that JOB_CODE will be A and DEPTNO will be D01 because we told DB2 to only return those rows using the WHERE clauses. Every column that DB2 has to access and return to our program adds overhead. Yes, it is a small amount of overhead here, but this statement may be run hundreds, or even thousands, of times a day. And that small overhead adds up to significant overhead.

3. **Use the WHERE clause to filter data in the SQL instead of bringing it all into your program to filter.**
   This too is a common rookie mistake. It is much better for DB2 to filter the data before returning it to your program. This is so because DB2 uses additional I/O and CPU resources to obtain each row of data. The fewer rows passed to your program, the more efficient your SQL will be.

Follow good SQL coding practices (like these three guidelines), and you'll start seeing a performance improvement in your DB2 applications. To further tune the code, you'll need to understand how to leverage the optimizer, update statistics, and manage indexes.

Leveraging the Optimizer

The optimizer is the heart and soul of DB2. It analyzes SQL statements and determines the most efficient access path available for satisfying each statement. It accomplishes this by parsing the SQL statement to determine which tables and columns must be accessed. It then queries system information and statistics stored in the DB2 system catalog to determine the best method of accomplishing the tasks necessary to satisfy the SQL request.

The optimizer is essentially an expert system for accessing DB2 data. An expert system is a set of standard rules that when combined with situational data can return an expert opinion. For example, a medical expert system takes the set of rules determining which medication is useful for which illness, combines it with data describing the symptoms of ailments, and applies that knowledge base to a list of input symptoms. The DB2 optimizer renders expert opinions on data retrieval methods based on the situational data housed in DB2's system catalog and a query input in SQL format.

The notion of optimizing data access in the DBMS is one of the most powerful capabilities of DB2. Remember, access to DB2 data is achieved by telling DB2 what to retrieve, not how to retrieve it. Regardless of how the data is physically stored and manipulated, DB2 and SQL can still access that data. This separation of access criteria from physical storage characteristics is called physical data independence. DB2's optimizer is the component that accomplishes this physical data independence.

If indexes are removed, DB2 can still access the data (albeit less efficiently). If a column is added to the table being accessed, the data can still be manipulated by DB2 without changing the program code. This is all possible because the physical access paths to DB2 data are not coded by programmers in application programs, but are generated by DB2.

Compare this with non-DBMS systems in which the programmer must know the physical structure of the data. If there is an index, the programmer must write appropriate code so that the index is used. If the index is removed, the program will not work unless changes are made. Not so with DB2 and SQL. All this flexibility is attributable to DB2's capability to optimize data manipulation requests automatically.

The optimizer performs complex calculations based on a host of information. To simplify the functionality of the optimizer, you can picture it as performing a four-step process:
1. Receive and verify the syntax of the SQL statement.
2. Analyze the environment and optimize the method of satisfying the SQL statement.
3. Create machine-readable instructions to execute the optimized SQL.
4. Execute the instructions or store them for future execution.

The second step of this process is the most intriguing. How does the optimizer decide how to execute the vast array of SQL statements that can be sent its way?

The optimizer has many types of strategies for optimizing SQL. How does it choose which of these strategies to use in the optimized access paths? IBM does not publish the actual, in-depth details of how the optimizer determines the best access path, but the optimizer is a cost-based optimizer. This means that the optimizer will always attempt to formulate an access path for each query that reduces overall cost. To accomplish this, the DB2 optimizer applies query cost formulas that evaluate and weigh four factors for each potential access path: the CPU cost, the I/O cost, statistical information in the DB2 system catalog, and the actual SQL statement.

The Importance of Statistics

Without the statistics stored in DB2's system catalog, the optimizer will have a difficult time optimizing anything. These statistics provide the optimizer with information about the state of the tables that will be accessed by the SQL statement that is being optimized. The types of statistical information stored in the system catalog include:
- Information about tables including the total number of rows, information about compression, and total number of pages.
- Information about columns including number of discrete values for the column and the distribution range of values stored in the column.
- Information about table spaces including the number of active pages.
- Current status of the index including whether an index exists or not, the organization of the index (number of leaf pages and number of levels), the number of discrete values for the index key, and whether the index is clustered.
- Information about the table space and partitions.

Statistics are gathered and stored in DB2's system catalog when the RUNSTATS utility is executed. Be sure to work with your DBA to ensure that statistics are accumulated at the appropriate time, especially in a production environment.

Index for Performance

Perhaps the single most important thing that can be done to assure optimal DB2 application performance is creating correct indexes for your tables based on the queries used by your applications. Of course, this is easier said than done. But we can start with some basics. For example, consider the following SQL statement:

```
SELECT          LASTNAME, SALARY
FROM            EMP
WHERE           EMPNO = '000010'
AND             DEPTNO =  'D01';
```

What index or indexes would make sense for this simple query? The short answer is "it depends." Let's discuss what it depends upon! First, think about all of the possible indexes that could be created. Your first short list probably looks something like this:

- Index1 on EMPNO
- Index2 on DEPTNO
- Index3 on EMPNO and DEPTNO

This is a good start and Index3 is probably the best of the lot. It allows DB2 to use the index to immediately lookup the row or rows that satisfy the two simple predicates in the WHERE clause. Of course, if you already have a lot of indexes on the EMP table you might want to examine the impact of creating yet another index on the table. Factors to consider include:

- Modification impact
- Columns in the existing indexes
- Importance of a particular query

Modification Impact

DB2 will automatically maintain every index that you create. This means that every INSERT and every DELETE to this table will cause data to be inserted and deleted not just from the table, but also from its indexes. And if you UPDATE the value of a column that is in an index, the index will also be updated. So, indexes speed the process of retrieval but slow down modification.

Columns in the Existing Indexes

If an index already exists on EMPNO or DEPTNO it might not be wise to create another index on the combination. However, it might make sense to change the other index to add the missing column. But not always because the order of the columns in the index can make a big difference in access path selection and performance, depending on the query. Furthermore, if indexes already exist for both columns, DB2 potentially can use them both to satisfy this query so creating another index may not be necessary.

Importance of this Particular Query

The more important the query the more you may want to tune by index creation. For example, if you are coding a query that will be run every day by the CIO, you will want to make sure that it performs optimally. Who wants to risk a call from the CIO complaining about performance? So building indexes for that particular query is very important. On the other hand, a query for a low-level clerk may not necessarily be weighted as high, so that query may have to make due with the indexes that already exist. Of course, the decision will depend on the importance of the application to the business – not just on the importance of the user of the application. An additional criterion to factor into your decision is how often the query is run. The more frequently the query needs to be executed during the day, the more beneficial it becomes to create an index to optimize it.

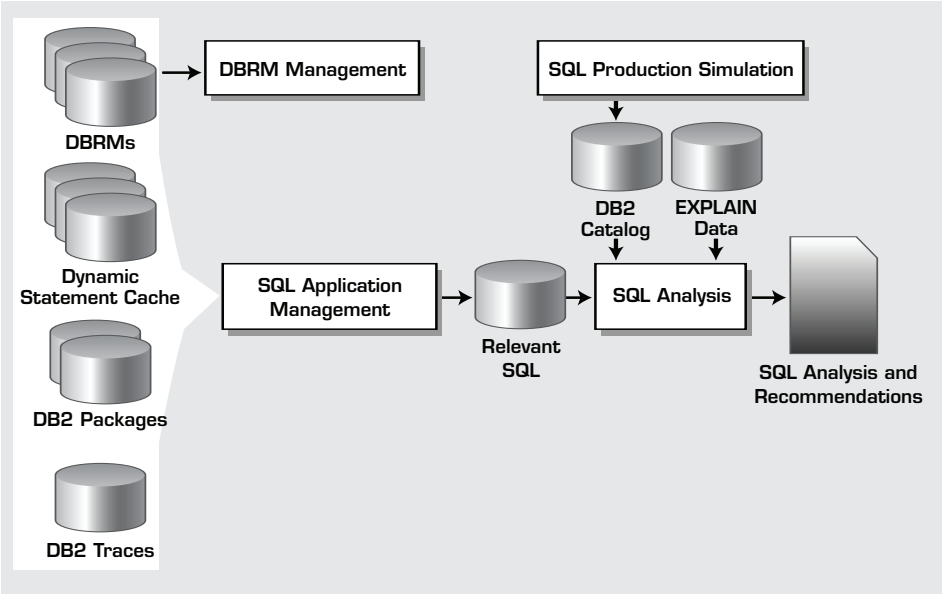
There is much more to index design than we have covered so far. For example, you might consider index overloading to achieve index only access. If all of the data that a SQL query asks for is contained in the index, DB2 may be able to satisfy the request using only the index. Consider our previous SQL statement. We asked for LASTNAME and SALARY given information about EMPNO and DEPTNO. And we also started by creating an index on the EMPNO and DEPTNO columns. If we include LASTNAME and SALARY in the index as well then we never need to access the EMP table because all of the data we need exists in the index. This technique can significantly improve performance because it cuts down on the number of I/O requests.

Keep in mind, though, that it is not prudent (or even possible) to make every query an index only access. This technique should be saved for particularly troublesome or important SQL statements. And you should always examine the impact on other queries and programs when deciding whether to add columns to any index.

Let NEON's DB2 Experts Help

SQL PerformanceExpert is a complete toolbox that helps produce optimized SQL through a two-tiered approach.

- Pre-production—Perform tuning during development stage, identifying and intercepting potential problems before they are moved into production
- Post-production—Analyze production applications to address high-consumption processes



SQL PerformanceExpert exploits DB2 V8 and V9 features, allowing you to fully optimize your SQL workload. By taking advantage of features such as multi-row fetch and EXPLAIN tables, SQL PerformanceExpert provides a new milestone of speed and functionality.

Through intelligent analysis and ongoing monitoring, SQL statements that negatively affect application performance are easily identified. SQL performance is easily improved by implementing recommendations provided about poorly performing SQL statements.

By quickly identifying only new and changed SQL, the QA workload is reduced. Analysis includes the capturing and preparation of dynamic SQL statements that are needed for comprehensive analysis. SQL PerformanceExpert provides standards for SQL application development through an expert rule system that can be easily customized. Recursive viewing performs analysis of predicates within a view.

SQL PerformanceExpert's production simulation enables development of high-performance applications, by generating reliable catalog statistics for use during testing and SQL analysis. It provides a method to create DB2 catalog statistics in the test environment to allow application access paths to be simulated during testing.

The product's DBRM checking feature reduces I/O activity and locking problems with fast and efficient examination of DBRMs, identifying potential performance problems. Use DBRM checking to determine compliance with current quality assurance rules by verifying new programs during the compile procedure and analyzing a single DBRM or whole DBRM libraries—without accessing the DB2 catalog. You can easily generate predefined rules and recommendations that help improve quality and performance; these rules can be customized according to severity level.

white paper

Package management examines the consistency of load modules, DBRMs, and the DB2 catalog to locate packages that are unnecessary or that are not bound, automatically cleaning up packages that are no longer needed.

Summary

Properly tuned SQL and a well-tuned DB2 environment can yield noticeable performance improvements. These can mean faster response time for DB2 applications, a better user experience, and faster throughput. The key is a combination of programming practice, system optimization, and effective use of software tools to automate simulation and code analysis.

About NEON Enterprise Software

NEON Enterprise Software is a technology leader in enterprise data management software and services. As the rules of business change, our solutions let you efficiently control, protect, retain and manage data to comply with today's business and legal requirements. Founded in 1995, NEON Enterprise Software serves customers worldwide with its team of dedicated industry experts.

For more information about NEON Enterprise Software, visit www.neonesoft.com or call 281.491.6366 or 888.338.6366.

white paper