Query Optimization in DB2 for i5/OS

A basic understanding of query optimization helps you get the most out of the system

by Mike Cain and Jim Flanagan

FOR MANY YEARS, I5/OS HAS BEEN A reliable and robust information management system by virtue of the integrated relational database engine known as DB2 for i5/OS. This trend has continued during the rise and acceptance of SQL as the standard means for communicating requests to the DBMS. DB2 for i5/OS supports both the original “record at a time” processing used in HLLs such as RPG and Cobol, as well as SQL “set at a time” processing.

One big difference between these two mechanisms is the programmer’s level of control and responsibility. With HLL record access, the application developer tells DB2 both what to do and how to do it. An example is opening a logical file for random access and reading by key. The developer specifies the index (i.e., keyed logical file) and the access method. With SQL, a higher level of abstraction is used. The application developer tells DB2 what results to produce, not how to produce them. An example of this type of request is find and return all rows where CITY = ‘Rochester’. The SQL request does not specify which, if any, index to use, nor does it specify an access method. In this scenario, the DBMS determines a strategy and method for identifying and accessing the requested rows.

The DB2 component responsible for determining how to carry out the SQL request is the query optimizer. Think of the query optimizer as the “programmer” who designs and writes the program to fulfill the request for data manipulation. In other words, the optimizer determines the strategy and methods for accessing and processing the data. This is no small feat given the robust nature of the query language and the corresponding range of possible requests.

As an implicit or explicit user of SQL and DB2 for i5/OS, you need a basic understanding of query optimization to get the most out of the system. Query optimization is a dynamic and complex proposition. Though DB2 for i5/OS makes the process seem easy, as an i5/OS application developer, you will benefit by keeping up to date with and understanding what this “extra programmer” is doing on your behalf.

Optimizer Overview

Is the optimizer a good programmer or a bad programmer? That naturally depends on the definition of good and bad, but more important, it depends on what the optimizer has to work with. To gain insight into the optimization process, an overview is helpful.

When an SQL query request is made to DB2 for i5/OS, three main components are invoked: the SQL interface, the optimizer, and the database engine. For this discussion, the various database connectivity options (e.g., ODBC, JDBC, DRDA) are irrelevant, because they are all valid means of providing an SQL request.
The SQL interface is responsible for communicating with the requestor, validating syntax, and gathering attributes such as isolation level and cursor behavior. The optimizer is responsible for determining the strategy and methods and building a query plan. The database engine is responsible for executing the query plan that the optimizer provides.

When a new query is introduced or the query environment changes, the SQL data manipulation request is optimized, a query plan is created, an open data path (ODP) is created, and the query plan is executed (Figure 1). Because all these steps take time, DB2 for i5/OS has techniques to minimize, eliminate, or speed up these phases.

To eliminate or speed up query optimization, DB2 saves the query plans from previous requests. Where these plans are stored depends on the type of SQL (i.e., static or dynamic) and the SQL interface (e.g., ODBC, JDBC, DRDA, CLI, embedded). It also depends on which optimizer is used — Classic Query Engine (CQE) or SQL Query Engine (SQE — see “What’s SQE?” below). To facilitate query plan reuse, any literals in the SQL statement are automatically converted to parameter markers. For example, the SQL statement issued as

\[
\text{SELECT * FROM Big_Table WHERE COUNTRY = 'United States' AND STATE = 'Minnesota' AND CITY = 'Rochester'}
\]

is converted to

\[
\text{SELECT * FROM Big_Table WHERE COUNTRY = ? AND STATE = ? AND CITY = ?}
\]

If the application issues the same query but specifies a different value for COUNTRY, STATE, and CITY, the same conversion takes place, and the statement text, query plan, and possibly the ODP match, allowing for more reuse. “The SQL Plan Cache Uncovered,” page ProVIP 39, provides details about where query plans are stored.

Creating and activating an ODP is relatively expensive and time-consuming. To minimize or eliminate this process, DB2 can save, manage, and reuse ODPs by job. This lets subsequent executions of a given SQL request within a given job avoid not only optimization but also full open processing. The net result is that the database engine needs to execute only the plan again.

A final thought for our optimizer overview is that it is always helpful for you to give the optimizer your requirements before running your query. See “Requirements Not Hints,” page ProVIP 47, for information about providing requirements to the optimizer.

**Query Optimization**

During query optimization, basically four phases are used:

- Query validation
- Query dispatching
- Query methods costing and plan creation
- Query implementation

During query validation, DB2 determines whether this is a new request. If this request has not been optimized previously, no plan currently exists, thus full optimization is required. If this request has previously been optimized, there is a good chance that the plan is saved. This
saved plan must be validated to ensure that it is viable for the current request. If the plan is valid, optimization can be eliminated or minimized. If the plan is not valid, full optimization is required. A plan can be invalidated when the query environment changes or the database objects change. Some common examples include adding an index, dropping an index used in the plan, increasing or decreasing the available computing resources, or changing DB2 capabilities by means of PTF or OS upgrade. DB2 for i5/OS's ability to recognize changes and dynamically reoptimize without user intervention is a hallmark of ease of use!

During query dispatching, DB2 determines which query optimizer can handle the request. The goal is to let the SQE optimize the query, but depending on the DB2 release level, the SQE might be incapable of running a particular query. For example, in V5R3, the SQE cannot handle a query that specifies the LIKE operator. This particular query is dispatched to the CQE. In V5R4, the SQE can handle a query that specifies the LIKE operator. This query is routed to SQE automatically. The query dispatching mechanism is another example of how i5/OS allows for significant upgrading of features and functions with little or no user intervention or application changes.

During query methods costing and plan creation, DB2 determines what methods are available and chooses the best strategy to be employed. The methods and strategy are assembled into a plan that the database engine executes.

During query implementation, the DB2 optimizer creates the ODP for the query. The ODP consists of the cursor, cursor behavior, and data mapping constructs. This phase is also where the optimizer manifests any information about the query and query plan. This information can take many forms and is used for SQL monitoring, analysis, and tuning. Finally, the plan is handed off to the database engine for execution.

**Query Optimization Phase**

The query optimization phase is one of the most important because it is where the user can play a vital role. Proper indexing is one of the largest success factors in providing the optimizer information about the data and providing implementation options to choose from. Indexes can supply statistics about the key column data as well as various access methods.

Without indexes, the optimizer is limited. For example, if the query is requesting all rows where column CITY = 'Rochester', and only one row out of one million rows matches the criteria, the optimizer needs to understand the expected selectivity of the query (i.e., 1/1,000,000) and have some access options available to choose from. If no index with the leading key column of CITY exists, the optimizer is limited to a full table scan to identify and return the row that matches. But if an index with the leading key column of CITY is available, the optimizer can choose to use a full table scan or use the index to help identify and access the row.

The DB2 for i5/OS query optimization process is cost based. This means that the relative cost of each available method and strategy — in terms of estimated time — is used to determine the best plan. The cost is a function of the actual method(s), strategy, data, and static computing resources available for the particular request. When optimizing a query, DB2 does not actually know what other jobs will be running when this particular plan is executed. The optimizer builds the plan based on the job's fair share of resources. When the query plan is ultimately executed, i5/OS work management is used to allocate computing resources across all active jobs.

Using the preceding example, the optimizer “costs,” or estimates, the time necessary to return one row using a full table scan on one million rows as compared with the “cost” to return one row using an index. The technique with the fastest estimated time is used in the plan.

**Clever Programming**

DB2 for i5/OS can optimize and implement SQL queries in many ways. One of the most interesting and

---

**Requirements Not Hints**

Some DBMSs require more help than others to arrive at a good query plan. The query optimizer in these DBMSs provides a way to give a “hint” about the data or environment. The design principle behind the DB2 for i5/OS optimizer suggests that the DBMS should be robust and clever enough to determine and acquire the information with a minimum of user intervention. Thus DB2 for i5/OS does not use the concept of hints.

Because every application environment is different, an application programmer needs a way to communicate database engine requirements to the optimizer and give it implementation options. Several ways to communicate these requirements exist, including system value parameters (SYSVAL), job attributes, connection attributes, i5/OS commands, and SQL requests.

For example, sometimes certain limits might be needed to restrict or govern how much time the optimizer should spend on a query. This requirement can be communicated systemwide with the QORYTMLMT SYSVAL, with the Change Query Attributes (CHGQRYA) command, or by a specific query optimization initialization file (e.g., QAQQINI file).

— M.C. & J.F.
smart techniques that the query optimizer employs are called query rewrite. The optimizer can rewrite the query internally, allowing more streamlined processing and more efficient execution. For example, reading and processing data only to throw it away later makes no sense. The query optimizer tries to eliminate rows as early as possible to reduce unnecessary processing.

A simple example of this idea is a grouping query that specifies the HAVING clause, which applies selection to the groups. The groups must be formed before the selection. Grouping and aggregating is often a costly process. Handling and then eliminating a large set of rows is unnecessary if the query can be rewritten to avoid processing the superfluous data. With this query

\[
\text{SELECT} \ \text{CITY, SUM(REVENUE)} \ \text{FROM} \ \text{SALES} \ \text{GROUP BY CITY} \ \text{HAVING} \ \text{CITY} = 'Rochester';
\]

the HAVING clause predicate (CITY = 'Rochester') is “pulled up,” and the optimizer rewrites the query to specify a WHERE clause predicate. This lets the database engine eliminate any irrelevant rows before aggregation. Because only one group (i.e., CITY) is returned, the grouping logic can also be effectively eliminated. The resulting query is much more efficient and should be faster:

\[
\text{SELECT} \ \text{CITY, SUM(REVENUE)} \ \text{FROM} \ \text{SALES} \ \text{WHERE} \ \text{CITY} = 'Rochester';
\]

**Window into the Optimizer’s World**

Now that we’ve covered the query optimizer and the different optimization phases, let’s look at what is going on inside the optimizer. The best way to accomplish this is to use iSeries Navigator, which offers many tools for viewing and interacting with the optimizer. This set of tools is collectively called the On Demand Performance Center.

The optimizer has several objects and methods that it can select from during the costing process. As we discussed in the query optimization phase, one of the more important objects is an index. Because an ideal set of indexes is not always available, how does the optimizer show what indexes it did use and what indexes it would like to have available?

**Autonomic Index Advisor and Show Indexes**

iSeries Navigator provides an interface to the DB2 for i5/OS autonomic Index Advisor in V5R4. Figure 2 shows how to launch the Index Advisor for the SALES table in an application’s KEYPRODX schema (i.e., right-click the table name and select Index Advisor/Index Advisor). Figure 3 shows that several indexes are advised by the query optimizer, and for each of these indexes, there is information about the key columns, number of times the indexes are advised, reason the indexes are advised, and more. In addition, any temporary indexes that the database engine created are represented in the list. These temporary indexes are significant because they show that the optimizer not only is advising that a permanent index be created but also is “creating” these indexes.

Recently, a Condense Advised Indexes menu option has been added (as Figure 2 shows) by means of PTF (i.e., in the latest database group PTF and the latest iSeries Access for Windows PTF for V5R4). This feature takes the detailed index advice, such as that in Figure 3, and condenses it by removing any duplicate advice — thereby automating the consolidation process and allowing for more efficient analysis. Figure 4 shows this condensed advice.
After the condensed list of advised indexes is displayed, another valuable function is available to complete the indexing picture. This function, called the Index Analyzer, can be launched from the Index Advisor screen by right-clicking an advised index and selecting Show Indexes to display a window that shows the existing indexes for the table (Figure 5). Important information for each index in the list includes the last time this index was used and how many times it was used. It also shows that a temporary index exists for this table. The display furthermore shows a last-used date for HLL programs that use native record access via keyed logical files.

By comparing the condensed indexes and the list of existing indexes, the report provides valuable information that can help with creating the “perfect index!” Creating the perfect index is an art — and the ultimate goal — and these tools can help you get closer. More information about indexing technology and strategy is available at [ibm.com/servers/enable/site/education/abstracts/indxng_abs.html](http://ibm.com/servers/enable/site/education/abstracts/indxng_abs.html). An additional resource is the DB2 for i5/OS SQL Performance Monitoring and Tuning Workshop, which you can enroll in for a fee ([ibm.com/servers/eserver/iseries/service/igs/db2performance.html](http://ibm.com/servers/eserver/iseries/service/igs/db2performance.html)).

### Automatic Statistics
Statistics are another key part of a cost-based optimizer. In DB2 for i5/OS, the optimizer collects and refreshes column statistics automatically. The statistical data includes information such as value cardinality, frequently used values, and value distribution. iSeries Navigator also provides an interface to look at the statistics information collected and used. To view the statistics for a specific table, right-click the table name and select the Statistic Data menu option, which displays the Statistics Data Details dialog box that Figure 6 shows.

Another important concept about the optimizer is that it can use existing indexes not just for faster data access but also for statistical purposes. The Index Analyzer in Figure 7, where several of the indexes advised have been created, shows this use of indexes. For the listed indexes, it shows the last time the index was used just for its statistics and how many times it was used for statistics (i.e., the report provides both a date/time of last use and a count of how many uses). The display also shows that the optimizer was able to directly use the indexes when running the queries.

Now that we’ve discussed indexes and statistics, let’s look at tools that help us view the optimizer’s world.

### Visual Explain
Visual Explain is the ultimate pictorial view of the query plan. A Visual Explain picture shows the whole query environment, including factors that affect the optimizer’s decision for selecting the access methods and objects for doing a specific optimization plan. Figure 8 shows the Visual Explain representation of the Group By query.
shown previously. The Visual Explain picture provides not only an idea of what the plan is but also gives a perspective on how it is executed. In this case, it is showing an Aggregation (Group By) over a Table Scan. It also includes a Temporary Index step, a flag for possible query performance tuning.

**SQL Plan Cache and SQL Plan Cache Snapshots**

Another powerful addition to the V5R4 tools is the SQL Plan Cache Viewer (Figure 9), which lets you view the system’s recent and current query activity. This ability gives you a convenient way to be proactive and find opportunities for query tuning. For instance, you could select the filter for the Top ‘n’ queries with the largest total accumulated runtime to retrieve the 10 longest-running queries on the system. In this example, the filter for Queries that use or reference these objects is selected and the application schema is filled in. To perform query tune right away, select a specific query and click the Run Visual Explain button. Or to save these for later viewing, select one or more rows and click the Create Snapshot button to make a permanent Plan Cache Snapshot of these.

One of the key benefits that this viewer offers is the ability to view the performance characteristics of statements run in the past without having captured them with a detailed SQL Performance Monitor. For example, you can use the Viewer to help you solve performance problems by using the date filter to see the statements run during a particular time frame, without having to rerun the statements. An SQL Plan Cache Snapshot is also useful for capturing a well-running query environment so that you have something to compare to after, for example, significant changes in the application or system environment. For information about SQL Plan Cache, see “The SQL Plan Cache Uncovered,” “5 Essential Ways to Use iSeries Navigator — SQL Plan Cache” (February 2007, article ID 20803 at SystemiNetwork.com), and “Preparing for the V5R2 SQL Query engine” (May 2003, article ID 16376).

**Detailed SQL Performance Monitor**

The detailed SQL Performance Monitor (i.e., the STRDBMON command) is a feedback tool whose main strengths are in capturing the historical data and transactional history of an application. iSeries Navigator provides a way to start this monitor, including several new filters added in V5R4. These filters can cast a smaller net to capture performance information to later view through the detailed SQL Performance Monitor analysis tool.

After the SQL performance monitor data is captured, you can go to the SQL Performance Monitors folder, find the monitor that you want to work with in the list, right-click it, and select Analyze to display an overview dialog box that lets you view a detailed database monitor (Figure 10). The analysis window is new in V5R4 and shows many high-level characteristics of the captured performance data. You can select any category and drill down into the details of it. For instance, you
could select the Full Indexes Created category and click the Statements button to show a report of the statements that generated this and the recommended index to create. You can also invoke the Visual Explain tool from the drill-down reports and do a deep analysis on a specific statement.

**Compare Analysis Tool**

SQL Plan Cache Snapshots and the detailed SQL Performance Monitors offer an option to compare with another SQL Plan Cache Snapshot or detailed SQL Performance Monitor. This Compare Analysis tool is also new in V5R4. After we select two objects to compare from the SQL Performance Monitors or SQL Plan Cache Snapshots lists and provide filters for percentage difference and statement runtime, the dialog box that Figure 11 shows is displayed. This dialog box shows the statements that had the biggest difference in performance, and they can be further analyzed.

In this example, a detailed SQL Performance Monitor was captured before the query was tuned. Then, by using the feedback tools, we can see that the optimizer is asking for an index to help it out. So the recommended index is created, and another detailed SQL Performance Monitor is captured. The new Monitor is then compared with the original, and we see that the statement’s runtime has improved. Note that we can use Visual Explain from this dialog box, and it would result in two Visual Explain pictures being displayed in a horizontally tiled format, letting us see the picture view of the methods that the optimizer chose in each case.

**A Strong Ally**

The DB2 query optimizer integrated in i5/OS provides a very important component for application developers. The optimizer determines the best available methods and strategy for an SQL request so that developers can focus their valuable time on the business logic. Understanding how and when query optimization occurs will make a positive impact on your use of DB2 for i5/OS. Because most new, modern applications require access to data with SQL, using the advanced tooling and information that iSeries Navigator provides lets you obtain all the benefits of the advanced database management system embedded in i5/OS.

Mike Cain is a senior technical staff member within the IBM Systems and Technology Group and team leader of the DB2 for i5/OS Center of Competency. Before his current position, he worked as an IBM AS/400 systems engineer and technical consultant. Mike lives in Rochester, Minnesota, and can be reached at mcain@us.ibm.com.

Jim Flanagan is an advisory software engineer at IBM in database development for DB2 for i5/OS. He is the team leader of iSeries Navigator — Database. Jim lives in Rochester, Minnesota, and can be reached at jflanagan@us.ibm.com.