IT'S TIME FOR DB2 SQL PERFORMANCE TESTING TO SHIFT LEFT

Overview: Learn how Db2 SQL performance testing can "shift left" so developers remediate defects earlier to improve software quality and reduce costs.

Writing applications that access Db2 requires skilled developers who can not only write application code using Java (or COBOL or C, etc.) but also write effective SQL. But SQL that works is not the same thing as SQL that performs well. Both effectiveness and efficiency are required.

As such, tools and techniques that enable developers to improve the quality and performance of their code are essential for modern application development success. One technique for improving code quality that is gaining support is for development practices to "shift left."

What Is Shifting Left?

The shift-left ideology arose out of the application testing discipline, where it is important to find and prevent software defects as early as possible in the software development lifecycle (SDLC).

Traditionally, software development progresses from requirements to design to development to testing and deployment through support. This is usually depicted in a graph showing these tasks from left to right.

Requirements gathering and design are shown to the left, delivery and testing to the right. With a shift-left development mentality, processes on the right are moved to the left, so they are performed earlier in the development lifecycle.

There are many types of shifting left that can be done, such as shifting portions of design into the requirements gathering phase, or by moving deployment earlier with a continuous delivery approach, such as is common with DevOps. But perhaps the predominant focus for modern development has been to shift testing processes earlier into the development cycle.

Why Shift Left?

Why are development teams shifting left? Industry analysis shows that the earlier in the process defects and problems are discovered and corrected, the less costly it is to deliver quality software. Software quality guru Capers Jones has written, "The cost of finding and fixing bugs or defects is the largest single expense element in the history of software."

And Stephen H. Kan, in his book *Metrics and Models in Software Quality Engineering*, writes that "defect removal at earlier development phases is generally less expensive. The closer the defects are found relative to where and when they are injected, the less the removal and rework effort."

By shifting left, software development teams can increase the velocity, quality and efficiency of software delivery.

Ways to Shift Left

To shift left requires training and automation. It is not reasonable to simply start performing tasks earlier; you need the appropriate culture, training and tools to succeed.

A culture that adopts Agile development techniques and DevOps lends itself to the shift-left approach. Such a culture embraces communication and teamwork across development and operational disciplines that can make it easier to shift left.

Additionally, it is important that training and education are made available as developers begin to perform tasks they heretofore did not perform. Without training, of course, it will take longer to become proficient with new tools and techniques.

And finally, automation is needed. By automating as much of the development and testing lifecycle as possible, more work can be accomplished earlier and with higher quality. Automation has been the driving force behind the adoption of DevOps practices, and it helps to enable shifting left. Over time, as AI and machine learning techniques improve, AI-enabled automation will be incorporated into the SDLC to further improve development with intelligent automation.

SQL Performance Testing

Although there are many types of software testing (unit, integration, system, regression, etc.) and all of them can be shifted left to varying degrees, developing applications that use Db2 require SQL performance testing.

Db2 SQL is very flexible, and there are multiple ways to write SQL queries to achieve the same results. For example, you can combine multiple tables using a join or a subselect and achieve the same results. But each SQL formulation is likely to perform differently, one better than the other. And this is only one example of the many different ways to formulate SQL statements to achieve the same purpose.

The development mindset is usually to write code that matches the requirements and delivers the expected results, not necessarily to assure the best performance. It is imperative, therefore, to conduct SQL performance testing on all programs before they are migrated to a production environment, or performance may suffer… perhaps to the point of being unusable. An application that performs so poorly that the user doesn't wait for the results is just as bad as no application at all.

For these reasons, it is vitally important to measure, analyze and improve all SQL as it progresses from development to production. The more SQL performance testing that can be accomplished by developers—that is, can be shifted left—the earlier we can find performance problems. And that means the cost of delivering high-quality Db2 applications will decline.

Db2 Resources for Assuring SQL Performance

Db2 provides many resources for evaluating the performance of SQL statements, but you need to know they are there and how to use them properly in order to successfully take advantage of them to test your code's performance.

You can use Explain to gather SQL access path details that describe the techniques Db2 will use to satisfy your SQL. This includes things like whether and how an index is used, techniques to combine data from multiple tables, where sorting is invoked and many other details. The data is encoded in tables and must be queried and interpreted to make sense of it.

To adequately test SQL performance, you also will need access to the Db2 Catalog, which contains information about Db2 objects and details about their size and structure (statistics). For example, you can use the Db2 Catalog to find the columns in a table, how many rows are in the table, what indexes exist, clustering and partitioning details and so on. Again, the information is encoded in the Db2 Catalog tables and must be queried and interpreted to make sense of it.

Often times, the performance of SQL is ignored in the test environment. Rarely are test data volumes the same as production; therefore, SQL that works efficiently in test may fail in production. But there are viable techniques to make Db2 test environments mimic production without requiring vast amounts of data.

DBAs can deliver scripts to developers that can be used when they create or refresh their test data. These scripts copy statistics from the production Db2 Catalog and use them to update test statistics; or, for new tables and columns, these scripts populate estimated production statistics.
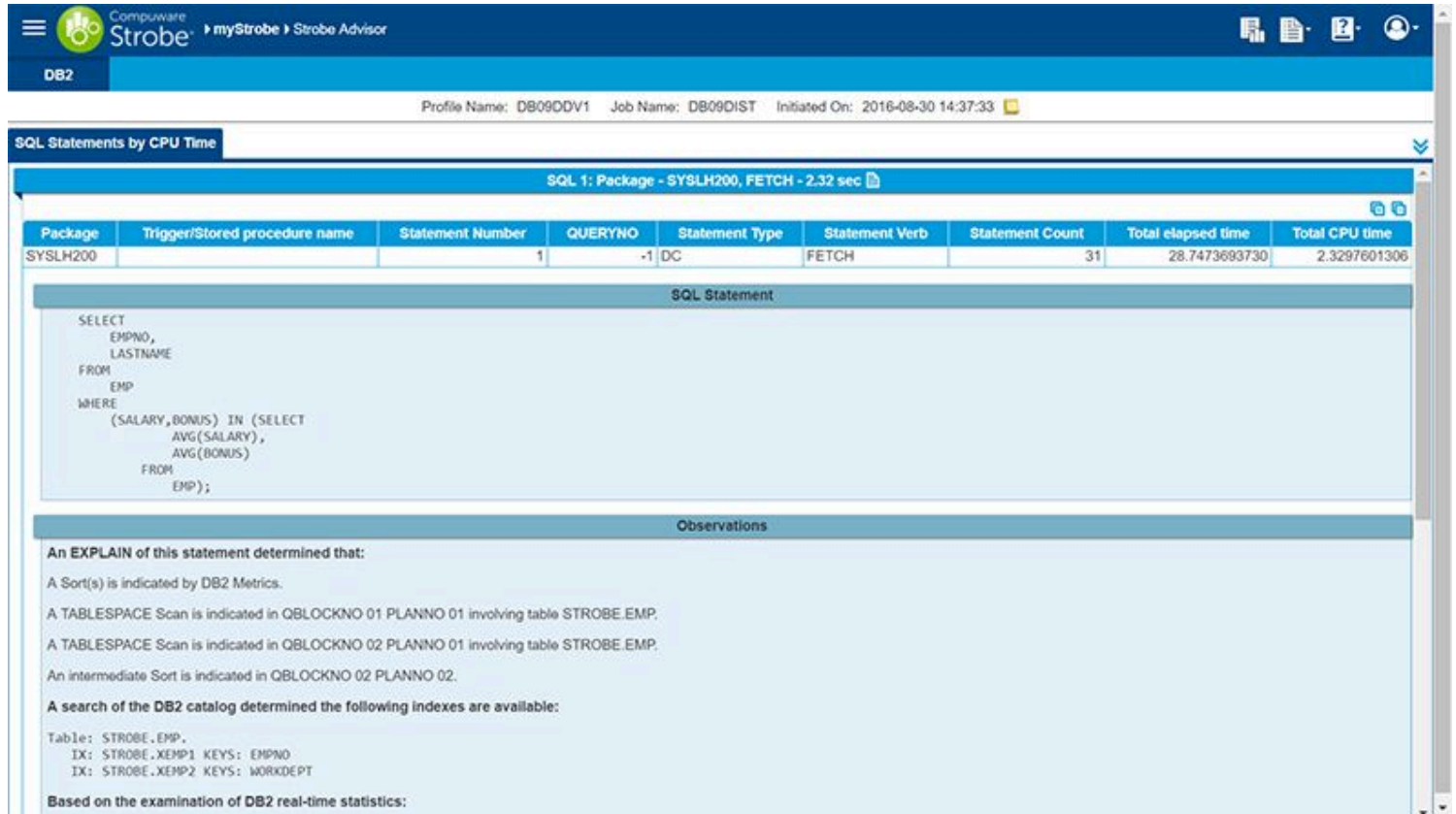
Furthermore, the test Db2 subsystem usually is configured with fewer resources (CPU, memory, etc.) than the production subsystem, and this can impact SQL optimization and access paths. DBAs can model the test subsystem to look like the production subsystem using profiles that simulate production CPU count, CPU speed, buffer pool size, and other system settings.

If the test subsystem is made to resemble the production subsystem, then developers can shift left to perform SQL performance testing earlier, instead of leaving it for DBAs and performance analysts to conduct later in the process.

Automating to Shift Left

To shift left requires automation and intelligent tools that can make difficult, time-consuming tasks easier. With proper planning, education and tools, developers can take on SQL performance testing and improvement earlier in the development lifecycle.

For example, a tool like Compuware Strobe can be used to simplify SQL performance testing by gathering together all of the pertinent performance information for the queries in your programs.

By deploying automation to measure the performance of programs and their queries, developers can assess the performance details and recommendations made by the tool and remediate defects earlier, thereby improving software quality and lowering overall costs.

A beneficial byproduct of shifting SQL performance testing to the left is the experience developers will gain as they tune the queries they write using tools that offer expert guidance and automation. This means that developers will become more skilled at writing efficient queries in the first place, and thereby will code fewer problematic SQL statements.

Summary

Shifting SQL performance testing and optimization to the left can improve code quality and reduce software development costs. Furthermore, the practice aligns well with other modern Agile and DevOps techniques deployed by most new software development projects. Instead of waiting to find performance problems in production, developers can use these methods to simulate production volume and use automated tools to optimize SQL in test, before it ever gets to production.