by Dan Cruikshank

At ACME Enterprises, operations or applications personnel have always performed most of the activities associated with the database. As systems have become larger and new technologies have emerged (e.g., the Internet and the Web), the advanced capabilities of the database have gone virtually unnoticed by the current staff. The operations and applications personnel have always focused on new techniques and methods (e.g., LPAR, networking, SOA, Java, WebSphere) that are directly related to their own primary function. As a result, the database functions became less important — until a program broke because a limit was exceeded (e.g., the number of members in a file) or an ad hoc query consumed all available resources.

Recognizing a growing need for better database management, ACME recently promoted its most senior programmer analyst to the position of database architect (DBA). The new DBA was tasked with the following objectives:

- Improve the performance of the legacy applications.
- Improve the performance of newer programs containing embedded SQL.

Having established a plan and methodology for capturing Performance Explorer (PEX) data, the DBA was ready to tackle the growing SQL workload. As the DBA began brushing up on database optimization for DB2 UDB for System i, he learned that there are two database optimizers (Figure 1), and that IBM will provide all future enhancements to the SQL Query Engine (SQE) and little or no enhancements to the Classic Query Engine (CQE). He was also startled to learn that specifying a DDS-defined logical file object on the FROM clause of an SQL statement forces CQE optimization. In addition, if a derived index (e.g., select/omit logical files) exists over any physical file object accessed during SQE optimization, the optimization will be rerouted to the CQE optimizer

Autor

Dan Cruikshank has been an IT professional since 1972 and an IBM employee since 1998. He has served in several capacities in both operations and programming, many application migrations from various platforms to the IBM System i family. Since 1993, he has been focused primarily on resolving i5/iSeries (AS/400) application and system performance issues at several IBM customer accounts. In 1999, he also took on the role of instructor for the IBM DB2 UDB for iSeries SQL Optimization Worksho

Künstler Burgy Zapp

This is a major concern, as ACME (like most long-term System i customers) has a substantial portfolio in older technology (i.e., Query/400 and Open Query File (OPNQRYF) command use). This investment in proprietary non-SQL interfaces and DDS-defined databases prevented ACME from reaping the benefits of new, enhanced data access methods introduced with SQE.

To demonstrate this to upper management, the DBA identified an SQL INSERT with a subselect consisting of a multitable join that was running for several minutes to several hours. Due to existing DDS restrictions, the SQL statement was optimized by CQE. By providing an override via the i5 (System i) INI function (explained later), the DBA was able to force the SQL statement to be optimized by SQE, which resulted in a plan that took 20 seconds to execute.

Database Optimization Basics

To take advantage of the new SQE optimizer, you should have a basic understanding of how a query is optimized. The optimization phase of SQL or query processing follows this simplified process:

1. Establish the optimization goal.
2. Dispatch the query to one of the optimization engines.
3. Cost-compare the plans.
4. Generate the best plan based on cost.

The optimization goal.

From optimizing HLL programs, the DBA at ACME learned that certain data access methods performed better than others based on the environment in which they're used (i.e., interactive versus batch). It was up to the development staff to determine the data access method before implementing a program. For example, an RPG developer at ACME now chooses the READ operation over the READE operation when developing a program destined for a batch environment.

The DB2 UDB for i5 optimizer also determines an optimization goal before cost-comparing the plan for a query. These optimization goals are referred to as First I/O or All I/O. First I/O biases the optimizer to create a plan that will return the first set of rows as quickly as possible. All I/O biases the optimizer to create a plan that will return the entire result as quickly as possible.

The fastest way to return the first set of rows is via indexes. The fastest way to return the entire result set may require the use of parallelism and/or temporary intermediate results.

The environment determines the initial optimization goal setting for the optimizer. Generally, all dynamic SQL interfaces (ODBC, JDBC, or interactive SQL) default to First I/O. The default can be changed implicitly to All I/O by using one of the following techniques:

- package support (extended dynamic)
- RUNSQLSTM
- INSERT + subSELECT
- CLI
- static embedded SQL in HLL programs

You can override the optimization goal explicitly through the use of the HLL precompiler options, the QAQQINI file, or the OPTIMIZE FOR n ROWS clause on an SQL statement.

The query dispatcher. As of V5R2, DB2 UDB for i5 comes with two optimizers (Figure 1). The query task dispatcher is responsible for determining the appropriate optimizer based on the type of query being processed. In essence, SQL queries are dispatched to SQE, and non-SQL queries (OPNQRYF,

Query/400) are dispatched to CQE. Portions of the new SQE optimizer are no longer a part of OS/400 or i5/OS. The code is shipped as part of the System Licensed Internal Code (SLIC).
As of V5R4, the following types of queries are dispatched to CQE:

- SQL statements that contain
- DDS logical file references
- User-Defined Table Functions (UDTFs)
- LOWER or UPPER scalar function
- Sort sequences and CCSID translation between columns
- Non-SQL queries (QQQQry API, Query/400, OPNQRYF)

All other SQL statements are optimized by the SQE optimizer. In addition, only SQE optimizes new functions such as INTERSECT and EXCEPT. Should the SQE optimizer encounter a derived index (e.g., a DDS logical file with the SELECT/OMIT keyword specified), the dispatcher will reroute the SQL statement to the CQE optimizer.

Cost-comparing the query.

Like all cost-based RDBMS optimizers, the DB2 UDB for i5 optimizer relies on statistics to estimate the cost of the possible plans that can be used to implement a query. Unlike some other RDBMS optimizers, the i5 optimizer gathers statistics dynamically from the environment and existing database objects. This includes the tables and their associated indexes (or keyed DDS logical files).

The optimizer's first step is to establish a default cost based on the assumed selection determined by the predicates used within the query. Each predicate has a default filter factor. For example, the optimizer assumes 10 percent of the rows in the table will be accessed when an equal predicate is used.