Performance Tuning
BI on SAP NetWeaver
Using DB2 for i5/OS and
i5 Navigator

Susan Bestgen
System i ERP SAP team
System i™ with DB2 for i5/OS™ leads the industry in SAP BI-D and SAP BW benchmark certifications with ten #1 certifications in the past two years. The article Industry Leading BI Performance With System i and DB2 for i5/OS Using BI on SAP NetWeaver describes the implementation of BI on SAP NetWeaver™ on System i and discusses technology features of DB2¹ which contribute to its performance success in the BI (Business Intelligence) technology segment. While hardware and software technology are key, performance improvements can almost always be achieved with intelligent tuning.

This article is a follow-on to the aforementioned article and provides a step by step performance tuning scenario of the SAP BI-D benchmark queries using i5 Navigator. Examples examined include:

- Filtering plan cache statements to identify “expensive” queries
- Using Visual Explain to graphically illustrate query implementation
- Measuring impact of relevant indexes and MQTs for performance of BI queries
- Utilizing DB2’s Index Adviser

While the examples used were specific to the BI-D benchmark queries, the steps performed and suggestions offered could apply to any application which generates similar queries.

iSeries Navigator Database Tools

Optimal performance of BI queries relies on many factors. Size of the underlying tables, complexity of the query, existence of indexes, sophistication of the query optimizer, hardware configurations and other factors all play a role in the performance of any given query on the system. The nature of Business Intelligence is such that many queries are 'ad hoc', generated on the fly as the end user defines how the business data available should be summarized to yield the desired results necessary for making business decisions.

In general, most SAP BI queries perform well on i5/OS due to the design of the SAP database (containing appropriate indexes for most queries), SAP design features which capitalize on the strengths of DB2, and the capabilities of DB2 to optimize complex ad hoc BI type queries. However, any database application can benefit from additional analysis and tuning, so database supplied analysis tools are key. i5/OS provides powerful performance analysis and tuning tools available through iSeries Navigator², a sub-component of the IBM iSeries Access.

¹ All references to DB2 in this article imply function at release level V5R4 for DB2 for i5/OS.

² For information on obtaining and installing iSeries Navigator, vist http://www-03.ibm.com/servers/eserver/iseries/navigator/ for product information,
The goal of the SAP BI-D benchmark is to achieve the maximum throughput (measured by SAP in number of dialog steps per hour) while executing the defined set of benchmark queries. For a database platform, this translates into optimizing the set of queries to minimize response time, which is a factor of both CPU, I/O and response time, while maximizing CPU utilization across the system. The software platform, and specifically the database solution, may use any generally available tools, tunings and techniques to achieve optimal results. Thus, the first step in database tuning is to identify and analyze the longest running queries. iSeries Navigator makes this a simple task. The screen shots that follow were generated running iSeries Navigator after the BI-D benchmark had been run on the System i server.

The first screen of iSeries Navigator allows the user to work with multiple systems to manage system functions, including database performance. DB2 has a built in SQL plan cache, which automatically captures a wealth of information on SQL statements run on the system. There are no monitors to turn on or external tools to run; the plan cache is always active and able to provide detailed analysis information and tuning tips for the queries active in the cache. First, we select the option to show statements in the plan cache (Figure 1):

![Figure 1: Initial iSeries Navigator screen](image-url)
This brings up a screen which allows us to filter the queries in the plan cache to those of most interest. The filter can include criteria such as those with the longest run time, those most frequently executed, queries run after a specified date/time, or any of the other options on this screen. For this example, we will limit the result set to those queries run after a certain time period that specify the tables known to be queried in the BI-D benchmark (the SQL statement text contains 'FBENCH'). This results in a set of twelve queries, with high level information about the performance of the queries. As each query is highlighted, the SQL statement text of the query is displayed at the bottom of the screen (Figure 2):

![Figure 2: Filtered SQL Statements](image)

The first query is a candidate for further analysis, as it has both the most expensive time (in seconds), and also the longest total processing time for the five runs of the query. By selecting the Show Longest Runs button at the bottom of the screen, we see
additional detail about each logged run of this query. In this example, the processing time seems to increase as the number of records returned by the query increases (Figure 3):

![SQL Statement Longest Runs](image)

**Figure 3:** Show longest runs of the selected query

The next logical step is to close this longest runs window and chose to Visual Explain the query (Figure 4):
This powerful tool not only gives a quick graphical overview of the query implementation, but it provides a wealth of optimization detail about each step of the query optimization and implementation (Figure 5). While we will not dig deeply into how this query was implemented, the graph alone illustrates it is a complex query with many nodes, all of which represent processing steps necessary to perform the query. A check of the SQL statement text at the bottom of the screen indicates a multiple file join query was run. Note that the query graph font has been reduced in size to illustrate the query graph complexity in a single page. Not all of the detail at each node of the graph can be read, but the detail is not necessary at this stage.
Each node indicates a step of the query processing

Figure 5. Visual Explain output of the selected query

At this point, some decisions can be made regarding the performance of the query. Is it likely to be run again, would performance be improved with tuning, or does it need to run any faster? If the query is one that will be run multiple times and performance improvement is desired, then further analysis is warranted. For benchmark purposes, we know the query is one that will be run over and over - the benchmark is driven by a tool that simulates many users running a defined set of queries repeatedly. This query will be run many times where the only thing that changes is the selection criteria.

While query optimization can be a complex topic, improved query performance is usually achieved by providing the right ‘tools’ for the query optimizer. Optimizer tools can include statistical information which helps the optimizer derive informed plans or additional objects which provide statistics and/or a means to reduce the work done to retrieve the query answer set. Two such objects to consider include indexes and MQTTs (Materialized Query Tables, which are sometimes called automatic summary tables or materialized views).
Indexes can provide additional statistical data and speed up query performance. An index over a table provides a fast method to find and retrieve table entries based on an index key value. Indexes are normally much smaller than the tables they are built over, so performance is improved by reducing the amount of data paged into memory. They provide a fast means of performing any record selection over key fields. Records can be retrieved in the order needed if they index keys match ordering, grouping or joining criteria, which can greatly reduce CPU needed to build an answer set. Indexes are associated with a single database table, so their existence improves the performance of data accessed for that table only.

MQTs are DB2 tables that contains the results of a query, along with the query’s definition. The stored results of an MQT can be substituted into the query’s implementation by the query optimizer for part or all the results requested by the query if the MQT is a ‘match’ to the query or contains a superset of the results of the query. A ‘matching’ MQT implies the MQT definition contains a subset of the selection criteria specified in the query and it therefore holds a superset of the applicable records needed by the query. They have the potential to greatly improve response time for complex queries by precomputed results for complex tasks such as joins, grouping functions, and selection.

In our example, we have a complex join which returns a fairly large number of records (around 44,000 for the larger result sets). While indexes certainly can improve the performance of join queries by speeding access to each individual table, the best option to improve performance is to reduce or eliminate the work done to perform the join. MQTs are perfectly suited for complex BI queries over large amounts of data which join many files.

In this example, an MQT can eliminate the expensive join processing and effectively reduce the final query to a single table query over the MQT. Additional selection and/or grouping which changes with each run of the query will be applied to the MQT result set automatically by the query optimizer. For simplicity in creating the desired MQT for this example, we first copy the SQL statement text from the bottom of the Visual Explain screen into the clipboard. Then, from the main screen of iSeries Navigator, we can start an interface that will allow us run the SQL statement to create the MQT (Run SQL Scripts, Figure 6):
Figure 6. Initiate Run SQL Scripts

After copying the SQL text into the Run SQL Scripts screen, it can be edited to have the criteria needed for the MQT creation. The original statement is as follows (Figure 7):
Figure 7. Original statement text copied to the Run SQL Scripts screen
The steps needed to convert the SQL statement into one that creates an MQT\(^3\) which can be used to implement the query is summarized below and illustrated in Figure 8.

- First, the CREATE TABLE clause is added, with the set of fields that must be present in the MQT specified. All fields in the SELECT clause of the original query are kept, and any fields from the WHERE clause where the selection criteria will change from run to run of the query are included so this selection can be performed at the MQT level. For this example, these additional fields would be:
  - DISTR_CHAN
  - DIVISION
  - INDUSTRY

- The join criteria of the MQT stays the same as the join criteria in the original query.

- In the WHERE clause, only selection which will not change from run to run of the query is built into the MQT definition. The rest of the WHERE clause record selection is removed. The selection built into the MQT for our example would be:
  - "DP"."SID_0CHNGID" = 0 AND
  - "DP"."SID_0RECORDTP" = 0 AND
  - "DP"."SID_0REQUID" <= 152 AND
  - "X1"."OBJVERS" = UX'0041' AND
  - "X2"."OBJVERS" = UX'0041'

  NOTE: In many cases selection in the MQT is limited to the join criteria only, as 'local' selection containing literal values, such as these, can also change. Limit local selection in an MQT to values you know will not likely change in the queries.

- Grouping fields include the original grouping fields plus any selection (WHERE clause) fields that can change between query runs (add as grouping fields any columns added in the SELECT list).
  - DISTR_CHAN
  - DIVISION
  - INDUSTRY

- We also specify that the MQT be populated with data immediately, refreshes are deferred and controlled by the user, and the MQT should be made available for query optimization.

\(^3\) For complete statement syntax rules and a description of MQT options, refer to the DB2 for i5/OS SQL Reference, which can be found at http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp.
Finally, the statement can be Run (Figure 8), and the MQT will be created on the database server.

For more detailed information about defining MQTs for expensive queries, refer to *Creating and using materialized query tables (MQT) in IBM DB2 for i5/OS™*.

Indexes are optional, but as with any good performance tuning situation, they should be considered over the MQT. It is a good idea to provide indexes over the record selection fields, starting with those fields which are most selective. The statement text to create two of our desired indexes looks like:

```sql
CREATE INDEX R3B40DATA/MQT0716I01 ON R3B40DATA/MQT0716000
  (DISTR_CHAN, DIVISION, INDUSTRY)

CREATE INDEX R3B40DATA/MQT0716I02 ON R3B40DATA/MQT0716000
  (DISTR_CHAN, DIVISION, STAT_CURR, MATL_TYPE, CALMONTH, VERSION, SALESORG)
```
CREATE TABLE R3H107_26000
    (DISTR_CHAN, DIVISION, INDUSTRY, STAT_CURR, MATEL_TYPE, CALMONTH, VERSION,
     SALESORG, INVCD_VAL, RECCOUNT)
AS
     "S2"."B49/S_VERSION", "S3"."B49/S_SALESORG", SUM ("F"."B49/S_INVCD_VAL" ),
     COUNT ( *)
FROM R3H40DATA."/B49/RECHCH07" F, JOIN R3H40DATA."/B49/RECHCH072" D2
ON F."KEY_BENCHCH72" = D2."DIMID" JOIN R3H40DATA."/B49/XMATERIAL" X0
ON D2."KEY_X0"."X0"."SID" JOIN R3H40DATA."/B49/XMATERIAL" X0
ON F."KEY_BENCHCH72"."X0"."SID" JOIN R3H40DATA."/B49/RECHCH07" DT
ON F."KEY_BENCHCH72"."DT"."DIMID" JOIN R3H40DATA."/B49/RECHCH074" D4
ON F."KEY_BENCHCH72"."D4"."DIMID" JOIN R3H40DATA."/B49/VERSION" S9
ON F."KEY_BENCHCH72"."S9"."SID" JOIN R3H40DATA."/B49/RECHCH073" D2
ON F."KEY_BENCHCH72"."D2"."DIMID" JOIN R3H40DATA."/B49/RECHCH07" S3
ON F."KEY_BENCHCH72"."S3"."SID" JOIN R3H40DATA."/B49/RECHCH071" D1
ON F."KEY_BENCHCH72"."D1"."DIMID" JOIN R3H40DATA."/B49/RECHCH72" X0
ON D1."KEY_X0"."X0"."SID" JOIN R3H40DATA."/B49/XMATERIAL" X0
ON X0."KEY_X0"."X0"."SID" JOIN R3H40DATA."/B49/RECHCH077" D0
ON F."KEY_BENCHCH72"."D0"."DIMID" JOIN R3H40DATA."/B49/RECHCH72U" DU
ON F."KEY_BENCHCH72"."DU"."DIMID"
WHERE "D3"."SID_OCHANGED" = 0 AND "D4"."SID_OCHANGED" = 0 AND "DT"."SID_OCHANGED" = 152 AND "X0"."SID_OCHANGED" = UX0041 AND "D4"."SID_OCHANGED" = UX0041
"S2"."B49/S_VERSION", "S3"."B49/S_SALESORG"

Figure 8. Original query altered to create the desired MQT
Now we can see the performance impact of creating an MQT and its indexes for the complex BI query. The benchmark has been executed again, and we again view the plan cache results for a specific start time with statements that contain "FBENCH". We can see the vast improvement in both the most expensive time and total time (Figure 9):

![Figure 9. Plan cache results running again with the new MQT](image)

A quick look at the longest runs further illustrates the point (Figure 10). Processing to retrieve 40,000+ records is now 12-13 seconds, as compared to 100 seconds or more previously.

![Figure 10. Response times for the selected query](image)
To verify the new MQT is being used, we can perform a Visual Explain (Figure 11). This much simplified query graph illustrates a single table - the MQT just created - is being queried (the Table Probe node), and it uses index and’ing/or’ing to retrieve the desired rows from the MQT. Articles *Indexing and Statistics Strategies for DB2 UDB for iSeries* and *The Power and Magic of LPG* provide additional information on indexing strategies.

Figure 11. New Visual Explain output of the selected query

Another powerful feature of Visual Explain is the Index Advisor. We'll look at another query where performance can be improved by adding an index. Back at the statement overview screen on a subsequent run, the second query is selected for analysis (Figure 12):
Figure 12. Selecting a second query for analysis

The top run times for this query, while fairly fast at less than 2 seconds, are as follows (Figure 13):

Figure 13. Individual run times for the selected query

Even though the response times are not excessive, we can Visual Explain the query to see if improvements can be made (Figure 14). At this point, the query graph tells us an MQT has been used to satisfy this join query (the Table Probe node using MQT070800), and an index over the MQT was also used (the Index Probe node). We
can see if any indexes are advised by the query optimizer that might improve the response time of this query by using the Index Advisor button, which is the last button in the task bar.

![Index advisor button](image)

Figure 14. Visual explain for the selected query and Index Advisor selection

The next display shows that one index is advised, and the optimizer indicates which file the index should be created over, what key fields are advised, and what type of index is recommended (either Binary Radix or EVI). In this case, it advises a binary radix index over MQT named MQT0708000 with 6 key fields (Figure 15):
From here, we can click on “Create”. On the displayed screen, we simply fill in the desired name of the index (MQT0708I02) and press OK (Figure 16). iSeries Navigator connects to the database server and creates the desired index.
The last step of this exercise is to measure the results. Again, we refresh the main screen of queries with the updated time of the last set of runs and find the same query we have just performance tuned (Figure 17):

![Figure 17. Result of rerunning with the new index and selected the desired query](image)

A quick look at the individual run times shows the response time improvement, from 1-2 seconds down to a maximum of .02 seconds (Figure 18):

![Figure 18. Runtimes for the selected query](image)
The Visual Explain can verify the newly created index was, in fact, used to implement the query and improve the response time. First we highlight the "Index Probe" node, then right click to select the option to display index information (Figure 19):

![Visual Explain of the query](image)

Figure 19. Visual explain of the query

Here we can verify the newly created index (MQT0708I02) was, in fact, the index used to implement this query (Figure 20):
Additional Index Advisor Capabilities

Index Advisor can helpful in more generic tuning situations as well. Our previous example assumed some knowledge of the file names of interest in the SQL statements. In most tuning situations, one would look at a performance across the entire system or within a specific library, or database schema, to further analyze performance. Index Advisor can aid in both of these situations.

Back at the main iSeries Navigator screen, we look at the Database options available. A system generated relational database entry name provides a system level view of iSeries Navigator options. If you have multiple IASPs (independent auxiliary storage pools), there will be one system relational database entry name per IASP. Selecting the system level name gives the index advisor options available (Figure 21):

- **Index Advisor**: View all indexes advised across the entire system
- **Clear All Advised Indexes**: A reset function to clear the list of advised indexes
- **Condense Advised Indexes**: Combines index advised at the table level by comparing the index advised keys. Index advised records whose keys are a subset...
of other indexed advised records are all rolled up to the ‘superset’ of keys over the table which would satisfy all indexed advised queries over the table.

- **Prune Advised Indexes:** Removes all advised index entries for objects which no longer exist.

![Index Advisor at the system level](image)

Figure 21. Index Advisor at the system level

If the system view is too global and you’d like to focus on a specific database schema, the same options are available by first selecting the schema of interest, then right clicking to select index advisor, and finally picking the index advisor option of interest (Figure 22).
At an even more selective level, you can first select the schema of interest and expand to view its list of objects. From there, by selecting the tables in the schema, the entire list of tables is retrieved. A specific table can be selected, and again there are options to see the indexes advised, to clear the advised list, or to condense the list (Figure 23).
For this example, we’ll go back to the database schema level and clear the index advised list. Next, we’ll run our test and then view what indexes are advised within the scope of the benchmark. So, the option selected is R3B40DATA (schema name) -> Index Advisor -> Clear All Advised Indexes (Figure 24).
After running our test, and selecting Index Advisor at the schema level, the resulting screen provides information on indexes advised by the query optimizer for just those SQL statements which were executed during the benchmark run. At the bottom of the screen, we can see the optimizer advises indexes over sixty-nine unique objects. Detail information for each object in the list includes the name of the object (both long and short), along with the keys advised, the type of index, how many times an index was advised, rows in the table when advised, query estimate times, and other information (Figure 25).

![Index Advisor Screen](image)

Figure 25. Schema level indexes advised after a clear and run of the benchmark

At this point, one can select a specific table and choose options to create the recommended index, remove this table from the list, or show the SQL statement(s) which generated the index advised (Figure 26). If choosing to create the recommended index, the prefilled index creation screen would appear, much like Figure 16.
When choosing to show the SQL statements, iSeries Navigator will prefill the plan cache filter screen to select only those queries over the desired object which have an index advised (Figure 27). Retrieving those statements would return a list of statements like those shown in Figure 2. One can then examine statements in detail and view Visual Explain on those statement to determine if creating the advised index is desired.
Conclusion

Performance tuning strategies like index and MQT creation can greatly improve individual query performance. As demonstrated in the examples above, iSeries Navigator simplifies the task of finding the queries, determining how they might be improved, and then validating and measuring the results of any changes. These examples highlight only a subset of the power of iSeries Navigator to identify and performance tune queries. Much more detailed information is available in Visual Explain. The reader can reference the list at the end of this article for additional information on how to improve performance using information available via iSeries Navigator.
Special Notices

Performance is based on measurements using standard SAP benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
For More Information

5 Essential Ways to Use iSeries Navigator - SQL Plan Cache
http://www.systeminetwork.com/artarchive/20803/5_Essential_Ways_to_Use_iSeries_Navigator___SQL_Plan_Cache.html

5 Essential Ways to Use iSeries Navigator — Visual Explain
http://www.systeminetwork.com/artarchive/20791/5_Essential_Ways_to_Use_iSeries_Navigator___Visual_Explain.html

Creating and using materialized query tables (MQT) in IBM DB2 for i5/OS_

Indexing and Statistics Strategies for DB2 UDB for iSeries_

Industry Leading BI Performance with System i and DB2 for i5/OS using BI on SAP NetWeaver
http://w3.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101087

The Power and Magic of LPG
http://www.mcpressonline.com/mc?1@127.QqQ0cfsnjgb.18995@.6b21733d!sectionID=.5bfbae44

Reviewing DB2 for i5/OS query optimizer and database engine feedback mechanisms

SAP NetWeaver™ Business Intelligence_
http://www.sap.com/platform/netweaver/components/bi/index.epx

SQL Performance Diagnosis on IBM DB2 Universal Database for iSeries

Star Schema Join Support within DB2 for i5/OS - Version 3
http://ibm.com/servers/enable/site/education/abstracts/16fa_abs.html
About the Author

Susan Bestgen has worked for IBM Rochester for 19 years. She began her career in database development, focusing on query optimization. For the past 11 years she has worked on the IBM ERP team. She helped with the initial product port of SAP to System i, designing and implementing functional and performance enhancements to the database interface. She currently works on benchmarking and performance of System i with SAP NetWeaver.

Acknowledgments

The author would like to thank the following people for their technical and editorial comments:

IBM Rochester:

- Rob Bestgen, DB2 for i5/OS SQE Chief Architect
- Michael Cain, DB2 for i5/OS Center of Competency Chair
- Kolby Hoelzle, i5/OS Development, SAP on System i
- Ron Schmerbauch, i5/OS Development, SAP on System i
- Steve Tlusty, i5/OS Development, SAP on System i

IBM Germany

- Eric Kass, i5/OS Development, SAP on System i
- Anja Winkler, SAP BI Consultant, SAP NetWeaver BI on System i

SAP Germany

- Dorothea Rink, Senior Developer, Sap NetWeaver on System i
- Axel Stein, Development Architect, Sap NetWeaver BI on System i