5 Essential Ways to Use iSeries Navigator — SQL Plan Cache

It’s always on, and it’s always there

by Mike Cain

One of the new and exciting tools now available in V5R4 is a graphical interface to the SQL Query Engine Plan Cache.

Here, I introduce five essential ways to make use of the SQE Plan Cache. Before we get started, a brief overview is in order.

A centralized query plan cache as been a part of the SQL Query Engine (SQE) since its debut in V5R2; but until now, there has not been a public interface to the information contained within this repository. If you’re running your queries on a V5R4 system, iSeries Navigator can now provide access to this valuable information.

One of the most useful things about the SQE Plan Cache is that it’s always on, and it’s always there. Because the SQL Query Engine stores all query plans in the plan cache as a byproduct of its query optimization, this vital information is available without starting any monitoring tools. As long your query is handled by SQE, you have the ability to analyze the plan and runtime feedback.

To obtain the information, use a V5R4 iSeries Navigator client connected to the system running V5R4 i5/OS. To access the SQE Plan Cache, right-click either the local database icon or the SQL Plan Cache Snapshot icon.

There are two sets of information available from the Plan Cache — explainable statements (i.e., queries) and the properties of the cache. Each set of information has its own purpose and usefulness.

One important thing to keep in mind: The information in SQE Plan Cache is dynamic. In other words, this information is being modified as queries come and go in the system. Given that the cache holds only the current query plan, it is not suitable for “tracing” a job’s SQL requests. Starting an SQL performance monitor (i.e., a database monitor) is the best tool for this task.

1. The Big Picture
To get an overall view of the Plan Cache, ask for its properties. Within the Properties window, we find some useful information:

- **Time Of Summary** represents when the properties were captured and displayed.
- **Active Query Summary** is information about the number of queries executed since the last IPL as well as the number of queries currently active. Also included is the number of “full opens” since the last IPL. A full open is a query request that was optimized and a new open data path (cursor) created.
- **Plan Usage Summary** is information about the cache size and the current contents of the cache (number of query plans).

If you display the plan cache properties periodically, you can use the information to note trends such as the fluctuations in the number of active queries and number of query plans stored. Keep in mind that the SQE Plan Cache is considered temporary storage and is cleared at IPL.

In a system with a very high number of unique and active SQL requests, the query plans in the cache can “turn over.” In other words, DB2 for i5/OS automatically manages the plan cache and will replace older plans with new plans, based on the cache size threshold. Thus, it is possible that a given query is no longer represented in the plan cache. The summary values in the properties window can give you an indication of this condition.

Generally speaking, about 10,000 unique statements can be represented in the SQE Plan Cache when the threshold is set to 512 MB. If the current plan cache size is always at or near the default threshold, this might indicate a need to increase the maximum size of the plan cache. Increasing the size will allow more query plans to be stored, and it can potentially eliminate added optimization for queries that have their plans removed.

2. Narrow Your Field of View
To get a list of the explainable queries in the SQE Plan Cache, use the Show Statements option. A new win-
Technology Reports

SystemiNetwork.com

February 2007

System i NEWS

ProVIP 7

For online transaction processing (OLTP) environments, there are normally hundreds or thousands of simultaneous queries, and these requests are expected to run quickly. When a particular SQL request has a poor access plan and is running slowly, finding and tuning it can have a pervasive impact. To find any such query via the SQL Plan Cache, use a combination of filters: the “Minimum runtime for the longest execution” and “Top ‘n’ most frequently run queries.”

To apply any filters and pull out the statements, click Retrieve. To get an updated list, click Refresh. The list of queries is returned initially sorted by the largest total accumulated time, in descending order.

With potentially thousands of SQL queries represented in the plan cache, applying the appropriate filters is essential. This is a good time to assess what it is you are looking for.

 FIGURE 1
SQL Plan Cache Statements window

FIGURE 2
Query filter options

dow appears that includes filtering options on the left and the statements that meet the criteria on the right (Figure 1). To save time, no statements are retrieved from the cache. This allows the analyst to narrow the set of statements, minimizing time and energy.

To apply any filters and pull out the statements, click Retrieve. To get an updated list, click Refresh. The list of queries is returned initially sorted by the largest total accumulated time, in descending order.

With potentially thousands of SQL queries represented in the plan cache, applying the appropriate filters is essential. This is a good time to assess what it is you are looking for.

At a minimum, set the “Top ‘n’” filter to 50 or less. This will allow the initial screen to appear much faster.

For reporting or online analytical (OLAP) environments, there is usually a smaller set of complex queries, and the appearance of these requests is often unpredictable in nature (ad hoc). If a user has a particular query that is long running or seemingly unresponsive, the SQL Plan Cache filtering can help identify the SQL statement and its access plan.

To find such a query, use the filter “Queries ever run by user.” This is the user name that was specified for authentication (sign-on) to i5/OS and DB2, not the name that is part of job’s identification. In addition to filtering by user, you can narrow the focus even further by specifying a specific table or view that is likely to be referenced (Figure 2).

If you don’t see the request that you know is slow running, it may be that this SQL statement is being handle by CQE, not SQE. To identify and capture information for a statement running with CQE, you will have to use an SQL performance monitor. In V5R4, there are many more options for starting a database monitor with filters.

For this situation, you can start a monitor to capture CQE information for a specific user or any queries that exceed a time limit. For any queries you find that are processed by CQE in V5R4, it helps to learn what is preventing the use of SQE. One simple thing to look for is the presence of any table or physical file that has a logical file (LF) with select/omit logic or derived keys. The presence of this LF results in the use of CQE. If the LF is in place only to support high-level languages and record-level I/O,
then you can ignore these logical files by specifying a QAQQINI file with IGNORE_DERIVED_INDEXES = *YES in the query environment.

To determine which query engine was used via the database monitor data, interrogate column QQC16 where QQRID = 3014. The value of ‘N’ indicates CQE is used, and column QVC43 will contain the reason SQE was not used. The query in Figure 3 shows an example.

3. Focus on the Important Stuff

Now that you have used the filters to find a set of queries, and you have the queries shown in the List of statements window, let’s focus on a few primary columns of interest:

- **Last Time Run** — when this particular query was last executed
- **Most Expensive Time** — the longest runtime of this particular query
- **Total Processing Time** — the total runtime of all executions of this particular query
- **Total Times Run** — the number of times this particular query was executed

By clicking a particular column heading, you can sort the list by ascending or descending order. By sorting the Total Processing Time in descending order, you can analyze a particular long-running query instance. By considering the Total Times Run value and the Most expensive time value, you can get an indication of how often this query is used and whether or not further analysis is warranted. It’s best to focus on queries that have relatively long average runtimes. A query that is run thousands of times and never takes longer than a few milliseconds is probably working well.

Once you identify a problem query and want to investigate its access plan, invoke Visual Explain by right-clicking the query in the list. It pulls from the SQL Plan Cache all the data you need to draw a Visual Explain diagram.

If a query is executed many times, and you want to determine whether the individual runtimes are disparate, select the query and click Show Longest Runs. This will pull out the top 10 longest runs for the plan cache.

If one execution out of the 10 is appreciably higher than all the others, scroll the report window over to view the Total Cached Results Used (Figure 4). SQE can save and reuse intermediate and final results when the same query, with the same data, is executed again. The first query in the sequence does all the work, and subsequent queries make use of information saved from the earlier execution. The Total Cached Results Used column can provide a clue that this benefit is occurring.

Another reason one instance of the query can be higher than all the others is actually due to the benefit of i5/OS single-level storage. The first query performed the physical I/O on the database objects, and the subsequent queries were able to take advantage of the pages already resident in memory. In this case, main memory is acting like cache.

4. Take a Picture; It’ll Last Longer

Now that you have identified a particular query of interest, you might want to save the data that describes the access plan and runtime characteristics. The iSeries Navigator interface provides the ability to take a “snapshot” of the plan cache data (Figure 5).

The snapshot results in a set of rows placed in a table. The output (rows and columns) looks similar to a detailed SQL performance monitor table. The information about the queries of interest is now persistent. Once a snapshot is created, iSeries Navigator will keep track of the snapshot, allowing for future analysis. This future analysis can include the comparison of two snapshots.

Imagine identifying a particular set of queries at two different intervals and capturing the information in two different snapshots. iSeries Navigator can compare two snapshots, showing the differences between the same query represented in each snapshot table. By right-clicking the first snapshot, you can invoke the compare

---

**Figure 3**

Query to determine which query engine was used

```
SELECT      x.qqjnum as "Job Number” ,
x.qqucnt as "Query ID” ,
x.qqi5 as “Query Instance”
y.qvc43 as “Reason for CQE” ,
x.qq1000 as “SQL Text”
FROM        –DB monitor table name– x,
–DB monitor table name – y
WHERE x.qqjfld = y.qqjfld
AND         x.qqrid = 1000
AND         x.qqucnt <> 0
AND         y.qqrid = 3014
AND         y.qqc16 = ‘N’
ORDER BY    qqucnt
qqi5;
```

**Figure 4**

Total Cached Results Used column

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Total Cached Results Used</th>
<th>Optin</th>
</tr>
</thead>
<tbody>
<tr>
<td>522448</td>
<td>42785</td>
<td>0.17</td>
</tr>
<tr>
<td>522448</td>
<td>42784</td>
<td>0.031</td>
</tr>
<tr>
<td>522284</td>
<td>5</td>
<td>0.062</td>
</tr>
<tr>
<td>522429</td>
<td>4</td>
<td>0.015</td>
</tr>
<tr>
<td>522284</td>
<td>4</td>
<td>0.104</td>
</tr>
</tbody>
</table>
function. A dialog appears to let you specify the second snapshot and thresholds (Figure 6).

5. Keep Track of the Information

Given that the SQE Plan Cache data is cleared at IPL, it might be advantageous to capture the information before powering down your system.

To capture the queries via iSeries Navigator, right-click the SQL Plan Cache Snapshot icon and select New, and then Snapshot (Figure 7). Beside the name of the snapshot, you can specify filtering options to tailor the output. Once the snapshot is created, you can analyze the information via the iSeries Navigator interface.

To programmatically initiate the creation of a snapshot, use the i5/OS stored procedure DUMP_PLAN_CACHE in schema QSYS2. The procedure has two input parameters and can be called via any SQL interface.

The first parameter is the schema or library name where the snapshot table will be placed, and the second parameter is the name of snapshot table that will be created:

```
CALL QSYS2/DUMP_PLAN_CACHE 'schema name', 'snapshot name'
```

To help automate the process of creating a snapshot before any normal IPL, use the power down system (PWRDWN SYS) exit point QIBM_QWC_PWRDWN SYS. If you register a program at this exit point via the WRKREGINF command, i5/OS will call the program prior to the power down operation. Your exit point program can use the RUNSQLSTM command to call the stored procedure.

Another useful collection strategy is to systematically collect SQE Plan Cache information by creating a snapshot on a regular basis at a consistent interval — for example, after the weekend batch process or every Monday morning. Once the data is collected at regular intervals, you can determine trends and spikes and evaluate them. And if a query does go bad for some reason, having a benchmark of a good run will be valuable in understanding what, if anything, changed in the query environment.

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. Prior to his current position, Mike worked as an IBM AS/400 systems engineer and IT consultant. Mike is in Rochester, Minnesota, and can be reached at mcain@us.ibm.com.