Advanced Performance Diagnostics for SQL

David Kalmuk

IBM

Session Code: D03
Monday May 23rd, 3:45 – 4:45pm
Platform: DB2 for Linux, Unix, Windows
Objectives

• Learn how to pinpoint your most expensive SQL statements using the package cache table functions
• Learn how to analyze where a problem query is spending its time using time spent metrics
• Learn how to monitor query sort memory usage and spilling
• Learn how to use the runtime explain capabilities and activity event monitor to capture both the actual cardinalities and the new object metrics for a problem query.
• Take away practical examples you can try out in your own environment.
A quick review of the core DB2 monitoring capabilities
Identifying high impact SQL statements
Analyzing queries using Time Spent
Monitoring query sort memory usage and spilling
Advanced diagnostics using Runtime Explain and Section Actuals
A Quick Review of the Core DB2 Monitoring Capabilities
DB2 Monitoring Basics

• How do I monitor what DB2 is doing?
  • Real-time in-memory metrics using SQL functions
  • Historical data captured using event monitors

• Also of note
  • Snapshot monitoring
  • db2pd

• For this session we’ll focus on the latest generation of monitoring capabilities introduced starting in DB2 9.7
Lightweight Monitoring Functions

- MON_* SQL functions introduced starting in DB2 9.7
- Less impact / more efficient then snapshot functions
  - Direct in-memory access through trusted routines (not fenced wrappers over snapshot apis)
  - Much less latch contention
    - Uses new infrastructure that pushes data up to accumulation points rather than forcing monitor queries to do extensive drilldown
  - Lower CPU consumption
  - Significantly faster response time
  - Less FCM resource usage / internode traffic
- Monitoring data collection carries low overhead – is enabled by default on new databases
Monitoring Perspectives and Dimensions

- DB2 allows monitoring metrics to be accessed through a number of different reporting dimensions
- Allows more effective drilldown, and different perspectives on the data to help isolate problems
- Three main dimensions, each consisting of a number of reporting points with corresponding routines
  - **System / Request**
    - Provide total perspective of application work being done by database system
    - Aggregated through the WLM infrastructure
  - **Data objects**
    - Provide perspective of impact of all activity occurring with the scope of data objects
    - Aggregated through data storage infrastructure
  - **Activity / Query**
    - Provide perspective of work being done by specific SQL statements
    - Aggregated through the package cache infrastructure

In this session we will spend our time here
Access Points: Activity Perspective

- **MON_GET_PKG_CACHE_STMT**
  - Both static and dynamic SQL (historical)
- **MON_GET_PKG_CACHE_STMTDETAILS**
  - XML based output
- **MON_GET_ACTIVITY**
  - Information on current executing activities / queries
- **MON_GET_ACTIVITYDETAILS**
  - XML based output
Some Additional Tips

• Monitoring data is accumulated and maintained in-memory from point of database activation until de-activation
  • Explicitly activate your database to ensure consistent availability of monitoring metrics

• Monitoring metrics are incremented globally at each of the reporting levels and do not reset
  • To compute changes in metrics over a specific period of time take an initial baseline sample and compute deltas from that (eg. compute I/O a particular SQL statement has driven over the past 5 mins)

• Event monitors can be utilized to capture and persist event based data for historical analysis
  • Package cache event monitor for aggregate statement data
  • Activity event monitor for individual statement executions
Identifying High Impact SQL Statements
MON_GET_PKG_CACHE_STMT()

- Ideal entry point for analyzing query problems
- Query a wealth of metrics for any statement that is active in the package cache
  - Rank and order by any of these metric
  - Aggregate metrics accumulated after each statement execution
  - Both static and dynamic SQL
  - Metrics collected by default
  - Low overhead
- Retains significant workload information with a modest PCKCACHESZ
- Package Cache Event Monitor can be configured in cases where cache evictions are causing information to be lost
- Only limitation is that it doesn’t track individual executions
Finding High Impact Queries

```sql
select stmt_exec_time, num_executions, stmt_text
from table(mon_get_pkg_cache_stmt(null,null,null,-2)) as s
order by stmt_exec_time desc fetch first 5 rows only
```

Top 5 queries by statement execution time in server:

<table>
<thead>
<tr>
<th>STMT_EXEC_TIME</th>
<th>NUM_EXECUTIONS</th>
<th>STMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>3951764</td>
<td>2218111</td>
<td>SELECT s_quantity, s_dist_01, s_dist_02, ...</td>
</tr>
<tr>
<td>902078</td>
<td>195866</td>
<td>SELECT c_balance, c_delivery_cnt ...</td>
</tr>
<tr>
<td>619547</td>
<td>212999</td>
<td>DECLARE CUST_CURSOR1 CURSOR FOR SELECT ...</td>
</tr>
<tr>
<td>480681</td>
<td>221873</td>
<td>SELECT w_tax, c_discount, c_last, c_credit ...</td>
</tr>
<tr>
<td>441494</td>
<td>20124</td>
<td>SELECT count(distinct S_I_ID) INTO :H ...</td>
</tr>
</tbody>
</table>
More High Impact Queries

Top 5 most CPU intensive queries

```sql
select stmt_exec_time, num_executions,
  (total_cpu_time / 1000) as cpu_time,
  stmt_text
from table(mon_get_pkg_cache_stmt(null,null,null,-2)) as s
order by cpu_time desc fetch first 5 rows only
```

Top 5 most I/O intensive queries

```sql
select stmt_exec_time, num_executions,
  (pool_read_time + pool_write_time +
   direct_read_time + direct_write_time) as io_time
from table(mon_get_pkg_cache_stmt(null,null,null,-2)) as t
order by io_time desc fetch first 5 rows only
```
Queries with the Worst Relative Velocity

```
select total_act_time, total_act_wait_time,
     (case when total_act_time > 0
          then ((total_act_time - total_act_wait_time) * 100
                / total_act_time)
          else 100
     end) as relvelocity,
     stmt_text
from table (mon_get_pkg_cache_stmt(null,null,null,-2)) as t
order by relvelocity fetch first 5 rows only
```

Relative velocity shows the degree to which progress of the query is impacted by waits.

Majority of query time spent in waits!

<table>
<thead>
<tr>
<th>TOTAL_ACT_TIME</th>
<th>TOTAL_ACT_WAIT_TIME</th>
<th>RELVELOCITY</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1481597</td>
<td>1457690</td>
<td></td>
<td>1 DECLARE READ_ORDERLI ...</td>
</tr>
<tr>
<td>228</td>
<td>223</td>
<td>2</td>
<td>2 create view dbtimeme ...</td>
</tr>
<tr>
<td>28</td>
<td>27</td>
<td>3</td>
<td>3 alter table activity ...</td>
</tr>
<tr>
<td>30</td>
<td>29</td>
<td>3</td>
<td>3 create event monitor ...</td>
</tr>
<tr>
<td>35</td>
<td>33</td>
<td>5</td>
<td>5 create event monitor ...</td>
</tr>
</tbody>
</table>
Queries with the Least Efficient Plans

This query shows us how much data we processed to produce a single row of results.

```sql
SELECT rows Returned, rows Read, 
    (case when rows Returned > 0
        then rows Read / rows Returned
        else 0
    end) as ratio,
    stmt_text as stmt
FROM table(mon_get_pkg_cache_stmt(null,null,null,-2)) as p
ORDER BY ratio DESC
FETCH FIRST 10 ROWS ONLY
```

<table>
<thead>
<tr>
<th>ROWS_RETURNED</th>
<th>ROWS_READ</th>
<th>RATIO</th>
<th>STMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11137814</td>
<td>5568907</td>
<td>select count(*) from acti...</td>
</tr>
<tr>
<td>1</td>
<td>5568907</td>
<td>5568907</td>
<td>select min(time_completed)</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>select * from syscat.WORK...</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1</td>
<td>select substr(serviceclas...</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1</td>
<td>select * from dbtimedelta...</td>
</tr>
<tr>
<td>2843729</td>
<td>2843729</td>
<td>1</td>
<td>DECLARE CUST_CURSOR1 CURS...</td>
</tr>
<tr>
<td>2843729</td>
<td>2843729</td>
<td>1</td>
<td>SELECT w_street_1, w_stre...</td>
</tr>
<tr>
<td>29599464</td>
<td>29599528</td>
<td>1</td>
<td>SELECT s_quantity, s_dist...</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>0</td>
<td>alter table control drop...</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>0</td>
<td>create view dbtimemetrics...</td>
</tr>
</tbody>
</table>
Analyzing Queries Using Time Spent
Time Spent Metrics

A set of metrics in DB2 that represent a breakdown of where time is spent within the server
- Represents sum of time spent by each agent thread in the system (foreground processing)
- Provides user with a *relative* breakdown of time spent, showing which areas are the most expensive during request / query processing
- Available in both the system and activity perspectives
  - This presentation will focus on analysis from the activity perspective
  - Can be used for rapid identification and diagnosis of performance problems

Times are divided into:
- Wait times
  - Time agent threads spend blocking on I/O, network communications, etc
- Processing times
  - Time spent processing in different component areas when the agent was not stuck on a wait
- Summary / total times
  - Total time spent in a particular component area including both processing + wait times
“Time Spent” Metrics: Breakdown of Wait + Processing Times in DB2

Total Request Time in DB2

- Direct I/O
- Bufferpool I/O
- Lock Wait Time
- Compile Proc Time
- Section Proc Time
- Commit / Rollback Proc Time
- Other Proc Time
Activity Time Spent Hierarchy

“Time spent” metrics are mutually exclusive and in aggregate form a hierarchy (shown below) that breaks down the time spent executing queries in the database server on behalf of the client. Below we show the hierarchy for the activity perspective.

SQL Statement Execution

WLM_QUEUE_TIME_TOTAL
STMT_EXEC_TIME
TOTAL_ACT_WAIT_TIME
LOCK_WAIT_TIME
LOG_BUFFER_WAIT_TIME
LOG_DISK_WAIT_TIME
FCM_SEND/RECV_WAIT_TIME
DIAGLOG_WRITE_WAIT_TIME
POOL_READ/WRITE_TIME
DIRECT_READ/WRITE_TIME
(…)
TOTAL_SECTION_PROC_TIME
TOTAL_SECTION_SORT_PROC_TIME
TOTAL_COL_PROC_TIME [new]
TOTAL_ROUTINE_NON_SECT_PROC_TIME
TOTAL_ROUTINE_USER_CODE_PROC_TIME
TOTAL_INDEX_BUILD_PROC_TIME [new]
(Any nested query processing)
Analyzing Individual Queries Using Time Spent

Once we have pinpointed our statements of interest, our next step is to drill down into these individual statements to understand where they are spending their time.

By understanding where the time is being spent in the query we can identify where the database server is spending effort, and look for opportunities for tuning.

We can use the EXECUTABLE_ID value from problem statements identified via examples in the previous section to lookup detailed time metrics for statements of interest and perform more in depth analysis.

- Uniquely identifies each query plan in the package cache
"Where is my time being spent?"

```sql
select p.executable_id, r.metric_name, r.parent_metric_name,
    r.total_time_value as time, r.count, p.member
from
    (select stmt_exec_time, executable_id from table(mon_get_pkg_cache_stmt(null,null,null,-2)) as s
        order by stmt_exec_time desc fetch first row only) as stmts,
    table(mon_get_pkg_cache_stmt_details(null, stmts.executable_id, null, -2)) as p,
    table(mon_format_xml_times_by_row(p.details)) as r
order by stmts.executable_id, total_time_value desc
```

- **Executable ID for our statement(s) of interest**
- **Find statement with most time in server**
- **Show me the full hierarchy of waits + processing times for a particular statement**
- **Format XML details to produce row based format for time spent metrics**
<table>
<thead>
<tr>
<th>EXEC_ID</th>
<th>METRIC_NAME</th>
<th>PARENT_METRIC_NAME</th>
<th>TIME</th>
<th>COUNT</th>
<th>MEMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>x'00000001...'</td>
<td>STMT_EXEC_TIME</td>
<td></td>
<td>6676617</td>
<td>110191</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>TOTAL_ROUTINE_NON_SECT_PROC_TIME</td>
<td>STMT_EXEC_TIME</td>
<td>6008956</td>
<td>110191</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>TOTAL_ROUTINE_USER_CODE_PROC_TIME</td>
<td>TOTAL_ROUTINE_NON_SECT_PROC_TIME</td>
<td>6008956</td>
<td>110191</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>POOL_READ_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>372754</td>
<td>52135</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>STMT_EXEC_TIME</td>
<td>372754</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>TOTAL_SECTION_PROC_TIME</td>
<td>STMT_EXEC_TIME</td>
<td>294907</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>WLM_QUEUE_TIME_TOTAL</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>FCM_TQ_RECV_WAIT_TIME</td>
<td>FCM_RECV_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>FCM_MESSAGE_RECV_WAIT_TIME</td>
<td>FCM_RECV_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>FCM_TQ_SEND_WAIT_TIME</td>
<td>FCM_SEND_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>FCM_MESSAGE_SEND_WAIT</td>
<td>FCM_SEND_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>LOCK_WAIT_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>DIRECT_READ_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>DIRECT_WRITE_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>LOG_BUFFER_WAIT_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>LOG_DISK_WAIT_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>POOL_WRITE_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>AUDIT_FILE_WRITE_WAIT_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>AUDIT_SUBSYSTEM_WAIT_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>DIAGLOG_WRITE_WAIT_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>FCM_SEND_WAIT_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>FCM_RECV_WAIT_TIME</td>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x'00000001...'</td>
<td>TOTAL_SECTION_SORT_PROC</td>
<td>TOTAL_SECTION_PROC_PROC</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Common Statement Bottlenecks

• I/O bottlenecks
  • Large bufferpool read / write times may indicate excessive table scans occurring, spilling to temps, or a poorly tuned I/O subsystem
  • Unexpected direct read / write times may indicatelobs that aren’t inlined properly, or unexpected usage of temps in query plan

• Locking bottlenecks
  • Large lock wait times indicate contention problems in your workload are affecting your query performance

• Routine bottlenecks
  • Large routine times may indicate inefficiencies or problems with procedures or user defined functions

• Reclaim wait bottlenecks [PureScale]
  • Large reclaim wait times indicate cross member page contention is impacting your query execution

• Diagnostic or audit bottlenecks
  • Diag log or audit wait times may indicate cases where diagnostic or audit related logging is unexpectedly impacting query performance
New Time Spent Metrics in DB2 10.5

- TOTAL_BACKUP_TIME / TOTAL_BACKUP_PROC_TIME / TOTAL_BACKUPS [DB2 10.5 Cancun]
  - New time spent category for online backups

- TOTAL_INDEX_BUILD_TIME / TOTAL_INDEX_BUILD_PROC_TIME / TOTAL_INDEXES_BUILT [DB2 10.5 Cancun]
  - New time spent category for index creation / recreations

- TOTAL_COL_TIME / TOTAL_COL_PROC_TIME
  - Time spent in the columnar runtime
Example: Assessing Efficiency of Columnar Query

```sql
SELECT TOTAL_SECTION_TIME, TOTAL_COL_TIME,
    DEC((FLOAT(TOTAL_COL_TIME)/
        FLOAT(NULLIF(TOTAL_SECTION_TIME,0)))*100,5,2)
    AS PCT_COL_TIME
FROM TABLE(MON_GET_PKG_CACHE_STMT(NULL,NULL,NULL,-1)) AS T
WHERE STMT_TEXT = 'SELECT * FROM TEST.COLTAB A, TEST.ROWTAB B WHERE A.ONE = B.ONE'
```

<table>
<thead>
<tr>
<th>TOTAL_SECTION_TIME</th>
<th>TOTAL_COL_TIME</th>
<th>PCT_COL_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>80.00</td>
</tr>
</tbody>
</table>

Compute the ratio of columnar processing time to overall section processing time to see how much we're leveraging the columnar runtime.

Majority of processing occurred in the highly optimized columnar runtime.
Monitoring Query Sort Memory Usage and Spilling
Monitoring Sort Memory Usage

- Sort memory can be monitoring through the following metrics:
  - SORT_SHRHEAP_ALLOCATED (current)
  - SORT_SHRHEAP_TOP (high watermark)
  - SORT_CONSUMER_SHRHEAP_TOP (per consumer hwm) (DB2 10.5 Cancun+)

- Accessible at multiple levels of reporting:
  - MON_GET_DATABASE (Database level)
  - MON_GET_PKG_CACHE_STMT (Query level) (DB2 10.5 Cancun+)
  - MON_GET_SERVICE_SUBCLASS_STATS (Subclass level) (DB2 10.5 Cancun+)
  - Others (DB2 10.5 Cancun+)

- Example:

```
SELECT SORT_SHRHEAP_ALLOCATED,
     SORT_SHRHEAP_TOP
FROM TABLE(MON_GET_DATABASE(-1))
```

Obtain current and maximum sort usage for the database
Monitoring Sort Consumers

- Total individual sort consumer counts including:
  - TOTAL_SORT_CONSUMERS (overall total) (DB2 10.5 Cancun+)
  - TOTAL_HASH_GRPBY
  - TOTAL_HASH_JOIN
  - TOTAL_OPAF_FUNC
  - TOTAL_SORTS
  - TOTAL_COL_VECTOR_CONSUMERS (DB2 10.5 Cancun+)

- Memory throttling and overflow / spill counts:
  - POST_THRESHOLD_HASH_GRPBY / HASH_GRPBY_OVERFLOW
  - POST_THRESHOLD_HASH_JOIN / HASH_JOIN_OVERFLOW
  - POST_THRESHOLD_OPAF_FUNC / OPAF_FUNC_OVERFLOW
  - POST_THRESHOLD_SORTS / SORT_OVERFLOW
  - POST_THRESHOLD_COL_VECTOR_CONSUMERS
Monitoring Sort Consumers

• Active sort consumer counts and high watermarks
  • ACTIVE_SORT_CONSUMERS / ACTIVE_SORT_CONSUMERS_TOP (DB2 10.5 Cancun+)
  • ACTIVE_HASH_GRPBY / ACTIVE_HASH_GRPBY_TOP
  • ACTIVE_HASH_JOINS / ACTIVE_HASH_JOINS_TOP
  • ACTIVE_OLAP_FUNCS / ACTIVE_OLAP_FUNCS_TOP
  • ACTIVE_SORTS / ACTIVE_SORTS_TOP
  • ACTIVE_COL_VECTORS_CONSUMERS / ACTIVE_COL_VECTOR_CONSUMERS_TOP (DB2 10.5 Cancun+)

• Also accessible at multiple levels of reporting
  • MON_GET_DATABASE (Database level)
  • MON_GET_PKG_CACHE_STMT (Query level)
    MON_GET_SERVICE_SUBCLASS_STATS (Subclass level)
  • Others
Monitoring for Spilling

with ops as
( select
  (total_sorts + total_hash_joins + total_hash_grpbys)
  as sort_ops,
  (sort_overflows + hash_join_overflows + hash_grpby_overflows)
  as overflows,
  sort_shrheap_top as sort_heap_top
from table(mon_get_database(-2)))
select sort_ops,
  overflows,
  (overflows * 100) / nullif(sort_ops,0) as pctoverflow,
  sort_heap_top
from ops;

<table>
<thead>
<tr>
<th>SORT_OPS</th>
<th>OVERFLOWS</th>
<th>PCTOVERFLOW</th>
<th>SORT_HEAP_TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>300</td>
<td>25</td>
<td>12777216</td>
</tr>
</tbody>
</table>

About 25% of our sort operations overflowed and spilled indicating some tuning may be worthwhile.

If SORT_HEAP_TOP is near the configured SHEAPTHRES_SHR it indicates that our SORTHEAP is overconfigured relative to our concurrency limits.

Extract percentage of sort operations that have spilled and high watermark sort usage.
### Monitoring Query Sort Usage and Consumers

```sql
SELECT SORT_SHRHEAP_TOP,
      SORT_CONSUMER_SHRHEAP_TOP,
      ACTIVE_SORT_CONSUMERS_TOP,
      NUM_EXECUTIONS,
      (TOTAL_SORTS +
       TOTAL_HASH_JOINS +
       TOTAL_HASH_GRPBY +
       TOTAL_COL_VECTOR_CONSUMERS) AS SORT_OPS,
      (SORT_OVERFLOWS +
       HASH_JOIN_OVERFLOWS +
       HASH_GRPBY_OVERFLOWS) AS SORT_OVERFLOWS,
      (POST_THRESHOLD_SORTS +
       POST_THRESHOLD_HASH_JOINS +
       POST_THRESHOLD_HASH_GRPBY +
       POST_THRESHOLD_COL_VECTOR_CONSUMERS) AS THROTTLED_SORT_OPS,
      SUBSTR(STMT_TEXT, 1, 255) AS STMT_TEXT
FROM TABLE(MON_GET_PKG_CACHE_STMT(NULL, NULL, NULL, -2))
```

<table>
<thead>
<tr>
<th>Sort usage + biggest individual consumer + active consumers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate sort operators</td>
</tr>
<tr>
<td>Aggregate sort overflows / spills</td>
</tr>
<tr>
<td>Aggregate throttled sort operators</td>
</tr>
</tbody>
</table>

Total memory usage

Biggest single operator

4 distinct sort consumers in this query
Advanced Diagnostics using Runtime Explain and Section Actuals
Introducing Runtime Explain

- There may be cases when a more detailed analysis of query execution is required than can be provided with basic monitoring metrics such as time spent.
- In these cases the tool we typically turn to is the EXPLAIN feature of DB2 – which we will refer to herein as the “SQL Compiler EXPLAIN”
  - This capability compiles an input SQL statement and allows you to format and view the query plan.
  - Expected to be a generally accurate approximation of the query you actually ran.
  - May differ due to differences in compilation environment and/or table statistics from when your query was compiled.
- In DB2 9.7 we also introduced the ability to perform a “Runtime EXPLAIN” (otherwise known as an explain from section) which produces output directly from a compiled query plan in the engine.
  - Allows you to generate plan output from the actual section you were executing.
  - Enables additional diagnostic features like section actuals and object metrics.
Explain from Section Procedures

- A set of stored procedures provided that allow you to format a runtime section into the explain tables
  - EXPLAIN_FROM_CATALOG
  - EXPLAIN_FROM_SECTION
  - EXPLAIN_FROM_ACTIVITY
  - EXPLAIN_FROM_DATA
- Explain table content can then be processed using the standard explain tools (eg. db2exfmt)
- Explain output can be generated from any of the following sources:
  - Static or dynamic statement entries in the package cache
  - Any cache entry captured by the new package cache event monitor
  - Static statement from the catalog tables
  - Statement execution captured with section by the activity event monitor
Section Actuals + Object Metrics

- One of the key benefits of the explain from section capability is the ability to capture and format “section actuals” and “object metrics”
  - All EXPLAIN output will contain cardinality estimates for individual operators in the plan
  - Explains generated from captured activity data (EXPLAIN_FROM_ACTIVITY) will also contain actual cardinalities and metrics per-data object within the query
- Examining this output gives you a detailed indication of what actually happened during the query execution
  - How closely actual cardinalities matched estimates
  - What activity occurred on individual data objects (in DB2 10.1+)
- In order to examine these metrics we will need to capture an execution of our SQL statement of interest using the activity event monitor
Capturing Activities to Obtain Detailed Explain Metrics

- The **activity event monitor** in DB2 allows the capture of execution details for individual SQL statements as well as several other recognized activities (e.g. Load).
- It can be configured to capture a variety of different metrics as well as the section data which includes actual cardinalities and object metrics.
- Since the capture of individual activities is quite granular we offer a fair degree of flexibility allowing the following data capture options:
  - Capture data for all activities running in a particular WLM workload.
  - Capture data for all activities running in a particular WLM service class.
  - Capture data for activities that violate a particular WLM threshold.
- We can also enable the capture of activities run by a specific application using the WLM_SET_CONN_ENV procedure.
- Our final example will demonstrate how to capture a statement of interest using the activity event monitor and then obtain the detailed explain metrics.
Step I: Prereq Setup Steps

call sysproc.sysinstallobjects('EXPLAIN','C',null,null)

create event monitor actEvmon for activities write to table activity ( table activity, in monitorTBS ),
activityvals ( table activityvals, in monitorTBS ),
activitystmt ( table activitystmt, in monitorTBS ),
activitymetrics ( table activitymetrics, in monitorTBS ),
control ( table control, in monitorTBS )
manualstart
Step II: Capturing the Activity Data

```
set event monitor actEvmon state 1
call wlm_set_conn_env(null,
   '<collectactdata>WITH DETAILS, SECTION</collectactdata>
   <collectactpartition>ALL</collectactpartition>
   <collectsectionactuals>BASE</collectsectionactuals>')</call wlm_set_conn_env(null,
   '<collectactdata>NONE</collectactdata>
   <collectsectionactuals>BASE</collectsectionactuals>')</set event monitor actEvmon state=0

select t1.ident, sum(t1.data) as data,
   sum(t2.moredata) as moredata
from t1,t2
where t1.ident=t2.ident
group by t1.ident
```

Enable the event monitor and setup to capture a statement on my connection

Execute the statement I'm interested in

Disable collection and the event monitoring once I am done.
Step II: Another approach

set event monitor actEvmon state 1
update db cfg using section_actuals base
alter service class sysdefaultsubclass under sysdefaultuserclass
  collect activity data on all database partitions with details, section

(Queries of interest run and are captured...)

alter service class sysdefaultsubclass under sysdefaultuserclass
  collect activity data none
update db cfg using section_actuals none
set event monitor actEvmon state 0

Enable the event monitor on the default subclass, and collect details and section data.

Disable the event monitor once I am done.
Step III: Locating the activity of interest

```sql
select a.appl_id, a.uow_id, a.activity_id, a.appl_name,
       s.executable_id, s.stmt_text
from activity as a,
     activitystmt as s
where a.appl_id = s.appl_id
  and a.uow_id = s.uow_id
  and a.activity_id = s.activity_id
  and s.stmt_text like 'select * from t1%'
```

Identifiers for the activity

<table>
<thead>
<tr>
<th>APPL_ID</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>EXECUTABLE_ID</th>
<th>STMT_TEXT</th>
<th>APPL_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>*LOCAL.davek.100917004844 62</td>
<td>1</td>
<td>x'010000...1E00'</td>
<td>select * from t1,t2 where...</td>
<td>db2bp</td>
<td></td>
</tr>
</tbody>
</table>
Step III: An alternate approach

```
select a.appl_id, a.uow_id, a.activity_id, a.appl_name,
    m.total_cpu_time, s.executable_id, s.stmt_text
from
    activity as a,
    activitystmt as s,
    activitymetrics as m
where a.appl_id = s.appl_id and
    a.uow_id = s.uow_id and a.activity_id = s.activity_id
and
    a.appl_id = m.appl_id and
    a.uow_id = m.uow_id and a.activity_id = m.activity_id
order by total_cpu_time desc fetch first 5 rows only
```

Find the captured activities with the largest CPU time

<table>
<thead>
<tr>
<th>APPL_ID</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>APPL_NAME</th>
<th>TOTAL_CPU_TIME</th>
<th>EXECUTABLE_ID</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>*LOCAL.davek.100917004844 62</td>
<td>1</td>
<td>db2bp</td>
<td>30500</td>
<td>x'0100...01E00'</td>
<td>select t1.ident, sum(t1.d</td>
<td></td>
</tr>
<tr>
<td>*LOCAL.davek.100917004844 54</td>
<td>1</td>
<td>db2bp</td>
<td>5360</td>
<td>x'0100...00900'</td>
<td>CALL wlm_set_conn_env(?,?</td>
<td></td>
</tr>
<tr>
<td>*LOCAL.davek.100919015109 20</td>
<td>1</td>
<td>db2bp</td>
<td>444</td>
<td>x'0100...05000'</td>
<td>SELECT TABNAME, TABSCHEMA</td>
<td></td>
</tr>
<tr>
<td>*LOCAL.davek.100919015109 25</td>
<td>1</td>
<td>db2bp</td>
<td>406</td>
<td>x'0100...05000'</td>
<td>SELECT TABNAME, TABSCHEMA</td>
<td></td>
</tr>
</tbody>
</table>
Step III: Notes on DPF and PureScale

- In DPF, statement execution is distributed across multiple partitions
  - Activity data must be collected on all partitions to capture all the work done by the query
  - Each partition involved in the query will generate an activity record and separate section actuals corresponding to that partition’s contribution
  - The explain process will amalgamate information across partitions automatically

- In PureScale statement execution is local to a particular member
  - Only the coordinator member will execute the query plan and generate section actuals
  - Note that the statement execution may still involve contention on global resources that are being contended for by other members
Step IV: Performing and Formatting the Explain from Section

Identifiers for the activity

call explain_from_activity('*LOCAL.davek.100715194643', 85,1, 'ACTEVMON', null, ?,?,?,?,?)

Perform an explain on the activity of interest...

db2exfmt -d sample -w -1 -n % -# 0 -s % -o explain.txt

Now format the most recent data in the explain tables to a readable text file.
Step V: Examining the Explain Output (Cardinalities)

Access Plan:

- **Total Cost**: 30.8779
- **Query Degree**: 1
- **Rows**
  - **Rows Actual**
  - **RETURN**
    - (1)
    - **Cost**
    - **I/O**
    - **6**
    - **30**
    - **GRPBY**
    - (2)
    - 30.8423
    - **NA**

- **HSJOIN**
  - (3)
  - **30.7964**
  - **NA**

  /-----+------\
  60 2.66667
  50 NA

- **TBSCAN**
  - **FILTER**
  - (4)
  - 9.40141
  - **NA**
  - **IXSCAN**

- **SORT**
  - (5)
  - 9.40141
  - **NA**
  - **IXSCAN**

- **TBSCAN**
  - **INDEX**: DAVEK
    - (6)
    - 9.27088
    - **NA**
    - **TABLE**: DAVEK

- **SORT**
  - (7)
  - **9.40141**
  - **NA**
  - **IXSCAN**

- **TBSCAN**
  - **INDEX**: DAVEK
    - (8)
    - 9.27088
    - **NA**
    - **TABLE**: DAVEK

- **SORT**
  - (8)
  - **9.40141**
  - **NA**
  - **IXSCAN**

- **TBSCAN**
  - **INDEX**: DAVEK
    - (9)
    - 9.27088
    - **NA**
    - **TABLE**: DAVEK

---

**Estimated Cardinality** (vs) **Actual Cardinality**

- **60**
- **50**
- **16**
- **16**
- **-1**
- **NA**
- **NA**
- **NA**
- **NA**
- **NA**
- **NA**
- **NA**
Step V: Examining the Explain Output (Object Metrics)

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Type</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVEK</td>
<td>IDX1</td>
<td>Index</td>
<td>object_index_l_reads:5&lt;br&gt;object_index_lbp_pages_found:5&lt;br&gt;object_index_gbp_indep_pages_found_in_lbp:5</td>
</tr>
<tr>
<td>DAVEK</td>
<td>T2</td>
<td>Table</td>
<td>rows_reads:60&lt;br&gt;object_data_l_reads:10&lt;br&gt;object_data_lbp_pages_found:10&lt;br&gt;object_data_gbp_indep_pages_found_in_lbp:10</td>
</tr>
</tbody>
</table>
Per-object metrics available through EXPLAIN (10.1+)

- **Tables**
  - Rows inserted / updated / deleted / read
  - Overflow creates / accesses
  - Lock wait time + lock escalations
  - Direct reads / writes
  - Bufferpool metrics for data, xda, columnar storage

- **Indexes**
  - Bufferpool metrics for index storage
Questions?
DB2 Monitoring Resources

- Tuning and Monitoring Database System Performance

- DB2 Monitoring Enhancements for BLU Acceleration
How Can You Get Started with Data Server Manager?

**DB2 ADVANCED EDITIONS**
- Included as part of DB2 Advanced Editions
- Target: Existing and New Customers

**BUSINESS VALUE OFFERING - PERFORMANCE MANAGEMENT OFFERING**
- Purchase in support of non-advanced DB2 editions
- Target Customers: DB2 customers expanding their footprint incrementally

**LICENSE EXCHANGE OFFER FROM EXISTING TOOLS TO PERFORMANCE MANAGEMENT OFFERING**
- Move existing customers (OPM, OQWT, OCM) to DB2 Advanced Editions (via trade-up part number) or Performance Management Offering (via license exchange)
- Target Customers: Existing DB2 customers who are already using the tools

**DB2 NON-ADVANCED EDITION CUSTOMERS**
- Included in DB2 (basic database administration and performance mgmt.)
- Target Customers: DB2 customers who just want base functionality
Please fill out your session evaluation before leaving!