IBM

developerWorks®

IBM Business Process Manager database troubleshooting, Part 2: Solve IBM BPM performance issues with IBM DB2 examples

Learn common problem scenarios with IBM DB2 for Linux, UNIX, and Windows

Stephan O. Volz
Bonnie K. Brummond

September 23, 2015

Learn about some common problem scenarios to use as examples for solving performance and database issues with IBM Business Process Manager (BPM) and the BPMDB database. Examples with IBM DB2 for Linux, UNIX, and Windows demonstrate and explain the root causes of performance issues you might face in your environment.

View more content in this series

This series focuses on what you can learn from the BPMDB database in IBM® Business Process Manager (BPM) to prevent problems and troubleshoot issues. The first two parts of this series focus on IBM DB2. Part 1 describes tips for database maintenance, statistics, and analysis for IBM DB2® for Linux®, UNIX®, and Windows®. Now Part 2 shows seven examples of common troubleshooting scenarios with IBM BPM and a `BPMDB` database that uses IBM DB2 for Linux, UNIX, and Windows. Part 3 focuses on tips for database maintenance, statistics, and analysis for troubleshooting IBM BPM and a `BPMDB` database that uses Oracle.

In Part 1, you learned about database maintenance and statistics for IBM BPM and IBM DB2 for Linux, UNIX, and Windows, the number of process instances in the system, system tasks, the number of stored snapshots, JDBC driver versions, hardware and environment limitations and related performance testing, a detailed troubleshooting analysis, and a summary of data that is required to debug an issue with the BPMDB database. The database that stores the runtime data that is associated with business process definition (BPD) processing is named BPMDB by default. This series addresses the BPMDB database but does not address the CMNDB (messaging and BPEL processing) and PDWDB (Performance Data Warehouse) databases that also are part of IBM BPM.

With proper planning, it is possible to prevent issues that affect performance before they are reported by process participants and other end users. This Part 2 shows seven examples of some common cases so that you can prevent problems and troubleshoot issues with IBM BPM and a BPMDB database that uses IBM DB2 for Linux, UNIX, and Windows. You can review examples of useful procedures for addressing commonly reported problems. In many situations, the steps in these examples are sufficient to gain sufficient insights into where problems are located and to actually solve the problems.

Example 1: Slow performance of database queries

Often when administrators find slow performance for database queries, they suspect that there is a delay on the database side. However, it's not always clear where the performance issues are coming from when you first investigate. Sometimes Process Portal users log in and the page does not build for a long time. Is this performance issue caused by a large number of entries in the database, a browser issue, a high load on the IBM BPM system, or something else?

If you think the problem might be related to the database, the best approach is to cover the IBM BPM and the database side. In the worst case, you might invest your time in collecting some data in vain. However, IBM teams that work with customers see scenarios where the focus was only on one side, and troubleshooting efforts went on for weeks. Based on this experience, it is worth your time to collect a larger amount of data so that you can find a quick resolution to your problem.

The following trace setting can help you find most of the problems between the IBM BPM product and a database system:
`WLE.*=all:com.ibm.bpm.*=all:WAS.clientinfopluslogging=all:org.springframework.jdbc.*=all`

If you already identified a slow-running query, you can run the **db2support** command with the **db2exfmt** option. As shown in the following example, you can run the **db2support** command with the slow-running query as the *`sql_file`*:

```
db2support output_directory -d database_name -sf sql_file -cl 1
```

Specifying `-cl 1` enables collecting **db2exfmt** information. See the Collecting Data for DB2 Compiler Issues support document.

The **db2support** command collects additional background information when the command is run, which can be very useful.

For example, from a **db2exfmt** output, you can see that a table scan was executed instead of an index access. Not every table scan is bad. If you require nearly all the data from a table, a table scan can be much more efficient than an index access. However, your approach depends on the scenario. In most cases with a well designed application, an index access is useful. The **db2exfmt** output example in Figure 1 shows the access path that the optimizer uses for a query to the LSW_BPD_INSTANCE_DATA table.

Figure 1. Example of db2exfmt output

```
Optimized Statement:
-------------------
SELECT +000000000510. AS "BPD_INSTANCE_ID", Q1.DATA AS "DATA"
FROM DB2INST1.LSW_BPD_INSTANCE_DATA AS Q1
WHERE (Q1.BPD_INSTANCE_ID = +000000000510.)

Access Plan:
-----------
        Total Cost:             7.95499
        Query Degree:           1

               Rows
              RETURN
              (   1)
               Cost
                I/O
                |
                 1
              FETCH
              (   2)
              7.95499
              1.05063
          /-----+------\
         1               158
      IXSCAN       TABLE: DB2INST1
      (   3)    LSW_BPD_INSTANCE_DATA
     0.0100077            Q1
         0
        |
        158
  INDEX: DB2INST1
 LSWC_BPD_INS_D_UQ
        Q1
```

If it is still unclear what is slowing down the system, consider using package cache or snapshot information to narrow down the problem to the IBM BPM product or the database system, which is described in following sections.

Consider additional warnings and considerations for working with indexes. IBM BPM includes several indexes. You might not see a benefit of a specific index included with the product, but you might not see the future scenarios where they are used. From a support perspective, all the indexes that are included with the product are expected to be in place. Deleting any of the default indexes results in performance issues, which cost an organization unnecessary time spent to realize that the indexes were removed, to get them back, and then to tune the system.

To improve the performance in a specific business case, it might be necessary to create more custom indexes. IBM BPM is used in several different scenarios, and an index that is introduced in one scenario might not apply in another. Therefore, it is up to organizations to determine what works best to apply on top of the product-shipped indexes. IBM cannot include all indexes for all situations, because although an index can improve read access, each additional index introduces an extra delay in updating entries.

To determine potentially missing indexes, use the DB2 Design Advisor (the **db2advis** command), as shown in the following example. The **db2advis** command can provide index recommendations. Database (`-d`), file with a SQL statement (`-i`) and a time limit (`-t`) of 5 minutes are given in the example. Be especially careful when you delete indexes. The **db2advis** command focuses on the query that is provided as input. There might be other queries used that show delays because indexes were dropped because of recommendations from the **db2advis** command. Before you delete an index, thoroughly check for those kinds of queries.

The following example shows how to run the **db2advis** command for index recommendations:

```
db2advis -d BPMDB -i MySQLstatement.txt -t 5
```

It's important to understand that IBM BPM contains internal constructs like "select for update" to prevent any modifications on corresponding rows. In some cases, a long-running "select for update" is an indication that something in the process design is not working as expected. For example, you call an external service synchronously and this service has a long response time, which keeps the statement in place until the final commit can be run. In this situation, running the **explain** command on the database server does not find any further information, so check the IBM BPM trace for what was actually triggered.

Example 2: Lock contention on the database

A number of scenarios exist that can lead to a lock contention on the database system. To correlate the activities that are done on the IBM BPM system and the database server, you need to track both sides.

IBM DB2 provides the lock event monitor, which provides a good overview of what statements are involved with the lock contention. To ease the matching between IBM BPM and DB2, use the option with history and values so that the passed parameters are caught. How to set up the DB2 lock event monitor is described in the Lock events for DB2 for Linux, UNIX, and Windows, Part 3 developerWorks tutorial.

An example output is shown in Figure 2. The lock event monitor output example shows the lock holder and requester. Further details about the statements that ran are displayed below the output that is shown in Figure 2. In some situations, the involved statements can help you determine the root cause.

Figure 2. A lock event monitor output example

```
-------------------------------------------------------
Event ID               : 1
Event Type             : LOCKWAIT
Event Timestamp        : 2015-05-07-14.28.26.336317
Partition of detection : 0
-------------------------------------------------------

Participant No 1 requesting lock
----------------------------------
Lock Name            : 0x0A00080006000E000000000052
Lock wait start time : 2015-05-07-14.28.26.308102
Lock wait end time   : 2015-05-07-14.28.56.316319
Lock Type            : ROW
Lock Specifics       : ROWID=6,DATA_PARTITION_ID=0,PAGEID=14
Lock Attributes      : 00000000
Lock mode requested  : Share (CS/RS)
Lock mode held       : Exclusive
Lock Count           : 0
Lock Hold Count      : 0
Lock rrIID           : 0
Lock Status          : Waiting
Lock release flags   : 00000000
Tablespace TID       : 10
Tablespace Name      : WORKITEM
Table FID            : 8
Table Schema         : BPEDB
Table Name           : E_SWI_T


Attributes            Requester                         Owner
--------------------- ------------------------------    ------------------------------
Participant No        1                                 2
Application Handle    039004                            038893
Application ID        *LOCAL.DB2.150507121337           *LOCAL.DB2.150507115221
Application Name      db2jcc_application                db2jcc_application
Authentication ID     ADMIN02                           ADMIN02
Requesting AgentID    1517                              1519
Coordinating AgentID  1517                              1519
Agent Status          UOW Executing                     UOW Waiting
Application Action    No action                         No action
Lock timeout value    30                                0
Lock wait value       30                                0
Workload ID           1                                 1
Workload Name         SYSDEFAULTUSERWORKLOAD            SYSDEFAULTUSERWORKLOAD
Service subclass ID   13                                13
Service superclass    SYSDEFAULTUSERCLASS               SYSDEFAULTUSERCLASS
Service subclass      SYSDEFAULTSUBCLASS                SYSDEFAULTSUBCLASS
Current Request       Open Cursor                       Execute
TEntry state          1                                 2
TEntry flags1         00000000                          00000000
TEntry flags2         00000200                          00000200
Lock escalation       no                                no
```

From the IBM BPM system side, a trace with the following settings can give insight into what operations are triggered:

```
WLE.*=all:com.ibm.bpm.*=all:WAS.clientinfopluslogging=all:org.springframework.jdbc.*=all
```

With the IBM BPM product trace, you can associate the database operations to activities on the IBM BPM server. Because the focus is on the BPMDB database in IBM BPM, these examples do not go into details for BPEL applications. However, the following trace settings can be helpful when you look for lock contention for BPEL applications:

```
*=info:org.springframework.jdbc.*=all:WAS.clientinfopluslogging=all:com.ibm.bpe.*=all: com.ibm.task.*=all
```

A common lock event monitor error scenario is a custom query that runs against the IBM BPM product database tables in the BPMDB database. Because the query is very complex, it blocks corresponding tables for a long time. This kind of query is not covered by the IBM BPM product tracing, but it shows up in the lock event monitor data collection. It is always good to see both sides of the equation.

Note that it is not supported and not recommended to run custom queries against the IBM BPM product database tables, unless the IBM support team requests it for diagnostic reasons. The example statements that are provided as examples in this tutorial series are for diagnostic purposes only. Use the examples to get a better understanding of the data distribution. The example statements normally do not have a significant impact on the system performance, but to prevent lock contention, run the statements in an `uncommitted read` isolation level (provide the `with UR` parameter at the end of statements). An `uncommitted read` reads entries that are not yet committed, which has the following advantage: the statement does not cause a lock on selected rows. Slight differences in the result set for uncommitted rows that are later revoked are not relevant for investigation.

Example 3: Read timeouts from network problems or slow database response

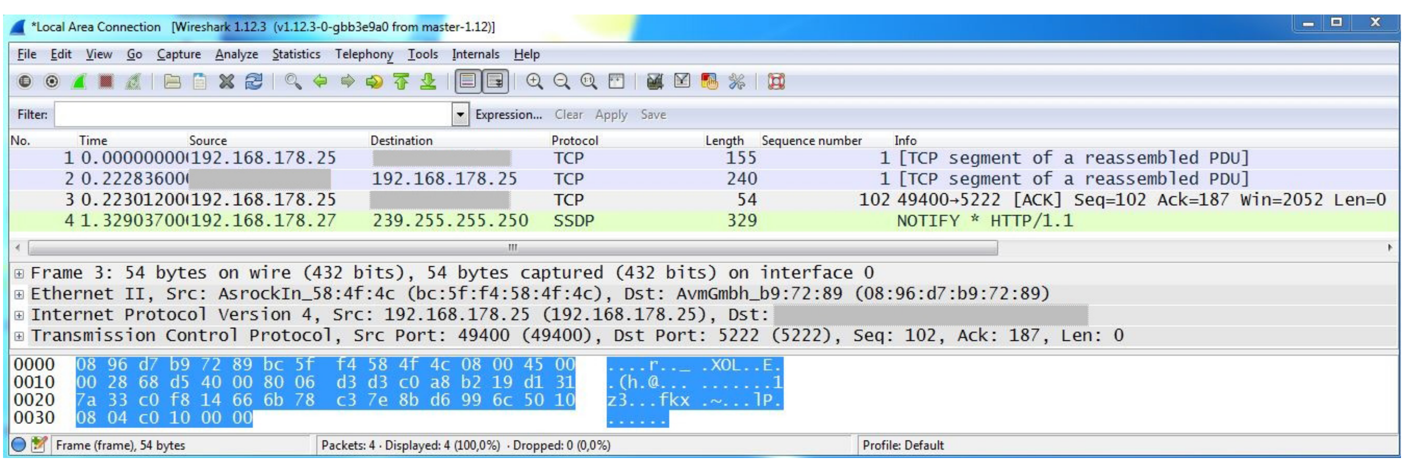
When you connect an IBM BPM system to a database, normally a network also involved. Because the communication between IBM BPM and IBM DB2 support teams is typically close, considering both systems makes it easy to identify problems in the network layer. When structural barriers exist between departments, or a different database vendor is used, consider the following approach for identifying problems in the network layer.

Since anything might be a potential problem, tracking all sides is recommended, especially because problems can change over time and might not always have the same root cause. A network trace can help to identify if a request was sent out or received. Several products are useful tools for capturing and analyzing a network trace, including Wireshark.

Consider an example in Figure 3 where a network packet is sent out and the destination address was obfuscated for security purposes. A Wireshark trace can show if a packet was sent out or not. For a production system, the data volume can become significant, and corresponding filtering can help limit the qualifying data. You can filter on known issues that can be barriers to successfully completing database operations, for example, packet resubmissions.

It's important to be aware of the system topology because everything between the IBM BPM system and the database system is a potential root cause of performance issues. Three components that might be easily overlooked are firewalls, network switches, and antivirus software. Also, consider hardware problems like a defective network adapter or a broken cable.

Figure 3. Example of a Wireshark trace that shows a Transmission Control Protocol (TCP) request sent



Example 4: Package cache information and DB2 snapshots

You can use the DB2 database package cache information to analyze performance problems and identify problematic statements without a major impact on the system. The developerworks article Mining your package cache for problem SQL in DB2 for Linux, UNIX, and Windows by Ember Crooks provides some sample queries that can be used to identify bottlenecks.

For this tutorial, adjust the `master query #1` of the example in the Ember Crooks' article slightly. With the `substr(SQL_TEXT,1,20)` notation, only the first 20 characters of the SQL statement are printed, which might not be enough for an IBM BPM system, so replace the string with the `SQL_TEXT` column name only.

This approach is critical in identifying the SQL statements that most often cause performance problems.

There is no major delay in the example that is shown in Figure 4 because the average execution time is less than 2 seconds. Of course, there is still some room for improvement. In most performance critical situations in the real world, the numbers are much larger.

Figure 4. Example output of package cache statement in the DB2 Control Center

STATEMENT	TOTAL_SECTION_SORT_TIME	PCT_TOT_SRT	NUM_EXECUTIONS	PCT_TOT_EXEC	AVG_EXEC_TIME
select t0.PROJECT_ID,t0.CREATED_ON,t0.LAST_MO...	2784	43.58	17308	0.89	1.58
select t0.BRANCH_ID,t0.CREATED_ON,t0.LAST_MOD...	0	0.00	969203	50.16	0.03
select BRANCH_ID,CREATED_ON,LAST_MODIFIED,AR...	0	0.00	452934	23.44	0.04
select SNAPSHOT_ID,CREATED_ON,NAME,ACRONYM...	0	0.00	104602	5.41	0.79
select propvalue from lsw_system where propkey=?	0	0.00	121211	6.27	0.04

An alternative to using the package cache is using DB2 snapshot information.

See the GET SNAPSHOT command topic in the IBM DB2 documentation on IBM Knowledge Center for details about DB2 snapshots.

You can use the following example steps for creating a database snapshot, which can be very useful for troubleshooting. Complete the following steps to collect a database snapshot to identify long running queries. It is essential that timing is enabled, which is the default.

1. Enable the monitor switches and reset the monitor data with the following DB2 commands:
```
db2 update monitor switches using timestamp on
db2 update monitor switches using lock on
db2 update monitor switches using bufferpool on
db2 update monitor switches using sort on
db2 update monitor switches using statement on
db2 reset monitor all for DB BPMDB
```
2. Let the problem that you are investigating occur. Normally this process completes over a few hours. The impact is not significant because there is not a large amount of data recorded.
3. At the end of the problem recreation get the snapshot to a file by running the following commands:

`db2 get snapshot for all on BPMDB  >  snapshot-for-all.txt`

`db2 "SELECT * FROM SYSIBMADM.TOP_DYNAMIC_SQL ORDER BYAVERAGE_EXECUTION_TIME_S DESC FETCH FIRST 20 ROWS ONLY" > top20sqls.txt`

4. Turn off the monitor switches again with the following commands:

```
db2 update monitor switches using timestamp off
db2 update monitor switches using lock off
db2 update monitor switches using bufferpool off
db2 update monitor switches using sort off
db2 update monitor switches using statement off
```

The database snapshot information typically provides sufficient data to identify long running queries. Discuss further steps with the database administrator if there are any indications of delay for the database. In some cases, indexes can help. In other cases, it is also necessary to check data distribution or the query in question. For example, process participants in your IBM BPM application aren't able to watch 20 million data sets that are requested from a database, so it doesn't make sense to query that many data sets. If you modify the query, you can reduce the impact on the database system and on IBM BPM, which needs to process the data that is returned from the query.

The SYSIBMADM.TOP_DYNAMIC_SQL database administrator view was implicitly included in the snapshot collection example. There are many more views to help analyze problems. See Supported administrative SQL routines and views in the IBM DB2 documentation.

Example 5: DB2 in-memory monitoring

DB2 for Linux, UNIX, and Windows V9.7 introduced a new in-memory monitoring concept to replace the traditional snapshot generation. Several configuration parameters affect the behavior. This tutorial provides a short overview of these features, but you can see more in the IBM DB2 documentation.

You can use the following two SQL statements for querying in-memory monitoring:

- db2 "call monreport.dbsummary(300)", which is a database summary for 300 seconds monitoring interval
- db2 "call monreport.pkgcache(30)", which is a query for all dynamic and static SQL statements that are updated in the last 30 minutes

Figure 5 shows a portion of the `dbsummary` report for an idle system. The example output is for the in-memory monitoring call for a `dbsummary` of the last 300 seconds, extracted from the IO, the buffer pool, and locking section only.

Figure 5. Example dbsummary report

```
Buffer pool
--------------------------------------------------------------------------------
Buffer pool hit ratios

Type              Ratio             Reads (Logical/Physical)
---------------   ---------------   ---------------------------------------------
Data              100               669/0
Index             100               866/0
XDA               0                 0/0
Temp data         100               50/0
Temp index        0                 0/0
Temp XDA          0                 0/0

I/O
--------------------------------------------------------------------------------
Buffer pool writes
  POOL_DATA_WRITES      = 0
  POOL_XDA_WRITES       = 0
  POOL_INDEX_WRITES     = 0
Direct I/O
  DIRECT_READS          = 2060
  DIRECT_READ_REQS      = 47
  DIRECT_WRITES         = 0
  DIRECT_WRITE_REQS     = 0
Log I/O
  LOG_DISK_WAITS_TOTAL  = 7

Locking
--------------------------------------------------------------------------------
                        Per activity                     Total
                        ------------------------------   ----------------------
LOCK_WAIT_TIME          0                                0
LOCK_WAITS              0                                0
LOCK_TIMEOUTS           0                                0
DEADLOCKS               0                                0
LOCK_ESCALS             0                                0
```

An advantage is that you need to run only one command to get a complete overview on the system use of input and output, buffer pools, locking, and other parameters that are not shown in the example. Running the command is rather simple and is especially useful for several system problems, including limitations with database hardware or locking behavior.

Here's an example from the real world. IBM teams worked with an organization that observed a problematic SQL statement that was easily identified with the in-memory reporting and used up 60% of the complete DB2 server CPU time during multiple executions: `select t0.USER_ID,t0.USER_NAME,t0.FULL_NAME,t0.PROVIDER from LSW_USR_XREF t0 where UPPER(USER_NAME) = UPPER(?)`

The main problem for the query is the `UPPER()` function, which requires a table scan. In this specific case, it is possible to create a generated column in the table that can be accessed by an index. (Note that the database table cannot be accessed during this procedure.) To solve that example problem, first back up the database and then run the following commands:

1. `db2 "reorg table LSW_USR_XREF use tempspace1"`
2. `db2 "set integrity for LSW_USR_XREF OFF NO ACCESS CASCADE IMMEDIATE"`
3. `db2 "alter table LSW_USR_XREF add column USER_NAME_UP VARCHAR(256) generated always as (UPPER(USER_NAME)) NOT NULL"`
4. `db2 "set integrity for LSW_USR_XREF immediate checked FORCE GENERATED"`
5. `db2 "create index username_up_idx ON LSW_USR_XREF (USER_NAME_UP)"`

After you run the commands, queries use an index scan instead of the table scan that was used in the previous example.

It is important to verify the changes. In some situations, you might need to run integrity checks on other tables so that the tables are not blocked, which would prevent SQL statements on the affected tables from running successfully.

See IBM BPM fixes for JR51631, which can reduce the number of executions for the query.

Example 6: IBM Performance Analysis Suite

The IBM Performance Analysis Suite tool can help you better understand database-related information about configuration and runtime data. Tools do not eliminate the need to understand the basis of performance troubleshooting, but they can help to more quickly see potential problem areas. There are several options to analyze database product settings. This tutorial focuses on two simple examples.

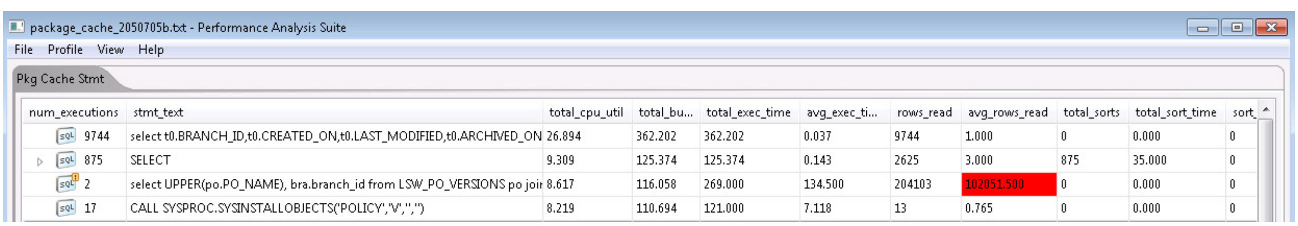
The following example uses a SQL statement to read package cache information, querying package cache information for analysis with the IBM Performance Analysis Suite:

```
select * from table(mon_get_pkg_cache_stmt('d',null,null,-2)) as t where
t.num_exec_with_metrics <> 0
```

This output can be imported into the IBM Performance Analysis Suite and the output is shown in Figure 6. It shows the IBM Performance Analysis Suite window that is loaded with the output of the package cache statement. Critical values for the Average Index Logical Read are marked. These values are good indicators that an index is missing or poorly designed for how it is used.

The field of the rows read is marked in red for one of the captured statements. This marking makes it easy to see large result sets and then dig deeper for the reason. In some cases, large result sets are not required, but they occur because they are not explicitly considered. You can see that the execution time of this statement is also rather large compared to the typical queries in the IBM BPM context.

Figure 6. Example of output package cache statements in IBM Performance Analysis Suite



num_executions	stmt_text	total_cpu_util	total_bu...	total_exec_time	avg_exec_ti...	rows_read	avg_rows_read	total_sorts	total_sort_time	sort_
9744	select t0.BRANCH_ID,t0.CREATED_ON,t0.LAST_MODIFIED,t0.ARCHIVED_ON	26.894	362.202	362.202	0.037	9744	1.000	0	0.000	0
875	SELECT	9.309	125.374	125.374	0.143	2625	3.000	875	35.000	0
2	select UPPER(po.PO_NAME), bra.branch_id from LSW_PO_VERSIONS po joi	8.617	116.058	269.000	134.500	204103	102051.500	0	0.000	0
17	CALL SYSPROC.SYSINSTALLOBJECTS('POLICY','V','','')	8.219	110.694	121.000	7.118	13	0.765	0	0.000	0

The next example shows how the IBM Performance Analysis Suite can connect to a database system and check corresponding settings (see Figure 7) and object information (see Figure 8). In the object information example, the statistics for LSW_SYSTEM_SCHEMA are not collected. Missing statistics are marked in red.

Figure 7. Connecting to a database system and checking the configuration in IBM Performance Analysis Suite

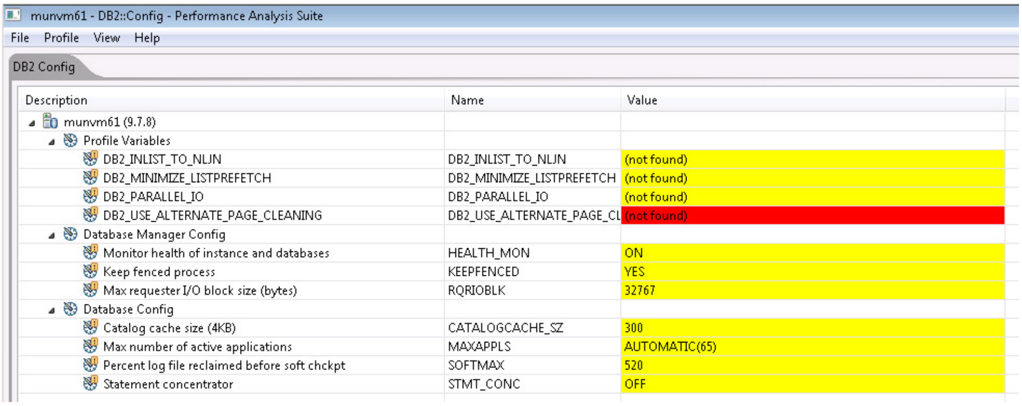
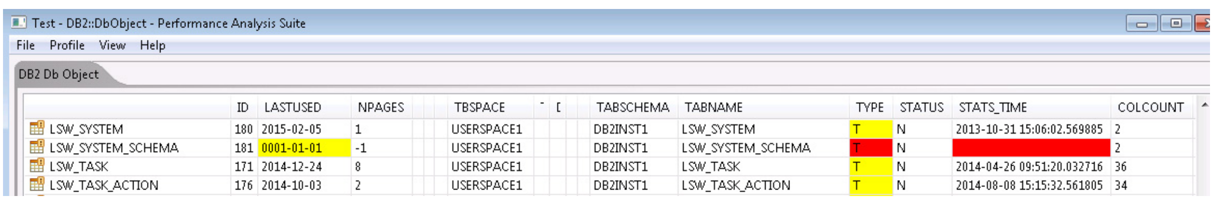


Figure 8. Connecting to a database system and reading table information in IBM Performance Analysis Suite



Example 7: Instrumentation logs

Instrumentation logs are another tool to help you troubleshoot performance issues. IBM Support document #1613989 describes how to use instrumentation logs.

The example in Figure 9 shows long running database queries that are visualized with the Instrumentation Log Report Tool for IBM BPM (IRLT) written by Andy Fedotov, which eases the interpretation of the instrumentation log data. Identifying specific statements might still require a trace. In the example output from the IRLT tool, you can see that several database requests use the most total time. It needs to be determined (for example, by a trace) why there are so many requests, or if the performance of specific statements can be improved.

Figure 9. Example output from the IRLT tool with sample data from this tutorial

```
Report generated with Instrumentation Log Report Tool for IBM BPM
(c)2014 andy.fedotov@gmail.com

Instrumentation log started at : 12:57:43.083
Instrumentation log ended at   : 13:01:01.194
Recording duration (secs)      : 198.111
Report Generated on            : 2015-07-05 13:06:10 +0200

==> System transactions summary: Key performance points
```

Transaction name	Count	Average (ms)	Median (ms)	Max (ms)	Total (ms)	TPS
Task: Resume Workflow Engine	14	1145	120	6932	16035	0.071
Task: Load Execution Context	3	10	12	14	30	0.015
Task: Save Execution Context	21	24	12	180	501	0.106
BPD: Load Execution Context	12	13	12	31	153	0.061
BPD: Save Execution Context	10	16	11	44	161	0.050
PersistenceServices (DB Access)	2552	13	5	719	33121	12.882
- findByPrimaryKey	946	23	6	719	21631	4.775
- bulkFindByPrimaryKey	8	10	10	20	82	0.040
- findByFilter	1447	6	3	141	8740	7.304
- save	45	22	16	127	1008	0.227
Do Job (Service Step Workers)	30	461	16	6495	13842	0.151
- ScriptWorker	12	125	45	916	1503	0.061
- SwitchWorker	2	15	15	27	29	0.010
- CoachWorker	-	-	-	-	-	-
- CoachNGWorker	4	1450	232	5321	5799	0.020
- SubProcessWorker	-	-	-	-	-	-
- ExitPointWorker	11	1	1	7	16	0.056
- JavaConnectorWorker	-	-	-	-	-	-
- WSConnectorWorker	-	-	-	-	-	-
- SCAConnectorWorker	1	6495	6495	6495	6495	0.005
- ILOGDecisionWorker	-	-	-	-	-	-
Eval Script	12	123	43	914	1479	0.061

Conclusion

You reviewed seven examples of troubleshooting the BPMDB database in IBM BPM. Nearly all of these examples can be used in other situations.

The statements in this series apply to all IBM BPM releases through IBM BPM V8.5.6, the current release at time of publishing. Future releases might require adjustments to the SQL statements, although no major changes are expected.

To learn about troubleshooting in environments with an Oracle database, see Part 3.

Acknowledgments

The authors want to thank Richard Metzger and Torsten Wilms for their review and suggestions for this tutorial. Any included error is the fault of the authors only.

The authors also acknowledge the work of Ember Crooks, who wrote Mining your package cache for problem SQL in DB2 for Linux, UNIX, and Windows, and Andy Fedotov who wrote the Instrumentation Log Report Tool for IBM BPM.

Related topics

- Lock events for DB2 for Linux, UNIX, and Windows, Part 3: Use the lock event monitor in DB2 9.7 to solve concurrency issues
- Mining your package cache for problem SQL in DB2 for Linux, UNIX, and Windows
- DB2 Tuning Tips for OLTP Applications
- Influence query optimization with optimization profiles and statistical views in DB2 9
- Purging data in IBM Business Process Manager
- IBM Performance Analysis Suite (developerWorks community)
- 5 Things to Know About IBM BPM Performance Tuning (developerWorks blog)
- Collecting Data for DB2 Compiler Issues (IBM Support document)
- Reading and decoding instrumentation files for WebSphere Lombardi Edition (WLE), and IBM Business Process Manager (BPM) products (IBM Support document)
- Tuning connection pools (IBM WebSphere Application Server documentation on IBM Knowledge Center)
- ILRT - Instrumentation Log Report Tool for IBM BPM on GitHub (contributed by Andy Fedotov)
- BPM system performance evaluation basics (BP3 website)
- IBM Business Process Manager V8.5 Performance Tuning and Best Practices: An IBM Redbooks publication
- IBM Business Process Manager V8.0 Performance Tuning and Best Practices: An IBM Redbooks publication
- IBM Business Process Manager V7.5 Performance Tuning and Best Practices: An IBM Redbooks publication