IBM Business Process Manager database troubleshooting, Part 3: Improve IBM BPM performance with an Oracle database

Stephan O. Volz

February 10, 2017

IBM Business Process Manager (BPM) typically interacts and communicates with other software products. Therefore, if you understand cross-product troubleshooting tools you can quickly understand and solve problems. This tutorial shows what you can learn from the content of the BPMDB database in IBM BPM to troubleshoot performance problems and database-related issues. This Part 3 of the series shows examples with Oracle, demonstrating and explaining problem scenarios that you might face in your environment.

View more content in this series

IBM® Business Process Manager (BPM) is a platform for processing and orchestrating enterprise business tasks. With proper planning, you can prevent performance issues before the end users of your process applications report them. This article focuses on what you can learn from the `BPMDB` database in IBM BPM to prevent problems and to troubleshoot issues when they occur.

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®. 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.

Now Part 3 focuses on Oracle. Learn tips for database maintenance, statistics, and analysis for troubleshooting IBM BPM and a `BPMDB` database that uses Oracle. For convenience, some information from the previous parts of the series are summarized.

Database maintenance and statistics information

When troubleshooting database-related problems, the first focus should be on the database itself. Oracle provides several automatic tools that can help you create and maintain a well performing system. Oracle includes an automatic tool to collect database statistics. Unfortunately, teams at some organizations disable the Oracle tools without implementing a manual way to collect database statistics.

The following code sample shows two SQL statements that you can use to get a quick overview what is actually stored in the database.

Listing 1. SQL statements to query the database catalog

```
select substr(table_name,1,35) as table_name,
 substr(tablespace_name,1,20) as tablespace_name, num_rows,
 blocks, empty_blocks, avg_space, chain_cnt, avg_row_len,
 last_analyzed from USER_TABLES ;

select substr(ui.index_name,1,35) as index_name,
 substr(ui.table_name,1,35) as table_name,
 substr(ui.tablespace_name,1,30) as tablespace_name,
 substr(uic.column_name,1,35) as column_name,
 uic.column_position, ui.status, ui.clustering_factor,
 ui.last_analyzed from USER_INDEXES ui, USER_IND_COLUMNS uic
 where ui.index_name = uic.index_name order by
 ui.index_name, uic.column_position;
```

These statements are not specific to IBM BPM and can be used in all situations with an Oracle database. The statements are for an IBM BPM database user (for example, as the IBM BPM data source user, with SQL*Plus® connected to the `BPMDB` database, and running the SQL statement). Additional views like `DBA_TABLES` exist for database administrators to query the same information with corresponding filters.

The following two screen captures show the output.

TABLE_NAME	TABLESPACE_NAME	NUM_ROWS	BLOCKS	EMPTY_BLOCKS	AVG_SPACE	CHAIN_CNT	AVG_ROW_LEN	LAST_ANAL
LSW_BRANCH	BPMDB_TB01_8570	11	5	0	0	0	242	22-DEC-16
LSW_PROJECT	BPMDB_TB01_8570	11	5	0	0	0	354	22-DEC-16
LSW_PO_VERSIONS	BPMDB_TB01_8570	73981	3520	0	0	0	336	22-DEC-16
LSW_PO_METADATA	BPMDB_TB01_8570	1682	58	0	0	0	279	22-DEC-16
LSW_PROJECT_DEPENDENCY	BPMDB_TB01_8570	24	5	0	0	0	146	22-DEC-16
LSW_DEP_PATH	BPMDB_TB01_8570	16	5	0	0	0	53	22-DEC-16
LSW_PO_REFERENCE	BPMDB_TB01_8570	897	244	0	0	0	981	22-DEC-16
LSW_PO_DEPENDENCY	BPMDB_TB01_8570	3700	244	0	0	0	240	22-DEC-16
LSW_RT_REFERENCE	BPMDB_TB01_8570	18	5	0	0	0	45	22-DEC-16
LSW_WEB_SERVICE	BPMDB_TB01_8570	0	0	0	0	0	0	09-DEC-16
LSW_WEB_SERVICE_OP	BPMDB_TB01_8570	0	0	0	0	0	0	09-DEC-16

In addition to the `table name`, also check the `last_analyzed` column, which shows the time stamp for when statistics were last collected for the database table. If the entry is a view, or no statistics were collected for the database table, the column is empty.

Current statistics are essential for good performance, because the Oracle optimizer bases the access path calculation on these numbers. Wrong statistics might lead to a bad execution path, which takes a long time to complete. Statistics updates are normally completed by Oracle, unless the Oracle optimizer is turned off. For more information, see Differences between Automatic Statistics Gathering job and GATHER_SCHEMA_STATS .

The SQL statement output also provides useful information about the number of rows and blocks for all tables. Especially large numbers might point you to several performance problems. In the example in the previous screen capture, you see a system that was recently set up. Therefore, a few tables like `LSW_PO_VERSIONS` might have millions of entries for a production system. Examine the problem areas to gain insight whether the issues are caused by large amounts of unneeded data.

The `CHAIN_CNT` column (the number of rows in the table that are chained from one data block to another) indicate the need for a table space reorganization if there is a different number than 0 (a rare case).

In addition to the table statistics, you can view index statistics. With the index statistics query, you can see the statistics, what indexes exist, what columns are included, and in which order.

Running storage space operations like shrink or compact can lead to invalid indexes. The output, as shown in the following screen capture, also lists the status of the indexes (some columns have been skipped).

INDEX_NAME	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	STATUS
SYS_C0074979	UT_IBM_BPM_PROCESSSECURITYC0	OBJECT_ID	1	VALID
TASK_MEASURES_PK	BPM_TASK_ACTIVITY_MEASURES	TASK_MEASURES_ID	1	VALID
TASK_MEASURES_UC	BPM_TASK_ACTIVITY_MEASURES	CREATED_BY_BPD_FLOW_OBJECT_ID	1	VALID
TASK_MEASURES_UC	BPM_TASK_ACTIVITY_MEASURES	USER_ID	2	VALID
THEMEGEN_BRID	BPM_UITHEME_GENSYNC	BRANCH_ID	1	VALID
UQ_EMTA_QTOSTID	LSW_EM_TASK	QUEUE_ID	1	VALID
UQ_EMTA_QTOSTID	LSW_EM_TASK	TASK_OWNER	2	VALID
UQ_EMTA_QTOSTID	LSW_EM_TASK	SCHEDULED_TIME	3	VALID
UQ_EMTA_QTOSTID	LSW_EM_TASK	TASK_ID	4	VALID
UQ_EMTA_TIDTO	LSW_EM_TASK	TASK_ID	1	VALID
UQ_EMTA_TIDTO	LSW_EM_TASK	TASK_OWNER	2	VALID

If an index is in a `UNUSABLE` state, the database administrator needs to rebuild the index. A missing or unusable index can significantly affect performance. In a table with a million rows, if no matching index is found, you might need a full table scan. Therefore, if something is listed as unusable, make sure it is fixed directly.

Also keep in mind, that Oracle provides function-based indexes, which are helpful when SQL statements include functions like `UPPER()` to prevent full table scans (for example, `UPPER(username)`when working with IBM BPM). To improve performance, you can require and create additional custom indexes, but do not modify indexes that are shipped with the product!

To handle multiple environments, implement a convention for change management. You can easily track what custom settings and indexes should be in place on different systems. This approach also can prevent surprises when you promote a new application to production, because indexes are different on the multiple environments.

> The indexes that are included as part of the IBM BPM database creation scripts are created to improve performance. To save yourself and the IBM Support team extra work, don't modify or remove indexes shipped with the product. Because there are a wide variety of IBM BPM workloads, your situation might require creating additional indexes for your specific workload patterns. That approach is absolutely fine. Keep in mind that an index can speed up your read access, but it can cause a bottleneck because each update or insert activity also must be reflected in each index.

Configuration parameters

Oracle databases have several configuration parameters that influence the system behavior, including volume-based settings like assigned memory and parameters that change transaction behavior. For IBM BPM run parameters that change transaction behavior with the default settings, which were tested with the product. Volume-based settings depend on the business workload and therefore can require adjustments.

The following code sample shows a SQL statement for an overview of the parameters and their currently active values. (You must have database administrator permissions to execute the statement.)

Listing 2. Statement to query the Oracle configuration parameters

```
select * from V$SYSTEM_PARAMETER ;
```

The following screen capture shows an example extract of the output from running the SQL statement:

```
      NUM NAME                           VALUE                          DISPLAY_VALUE                  ISDEFAULT
--------- ------------------------------ ------------------------------ ------------------------------ ---------
     1998 ldap_directory_sysauth         no                             no                             TRUE
     2001 max_enabled_roles              150                            150                            TRUE
     2006 07_DICTIONARY_ACCESSIBILITY    FALSE                          FALSE                          TRUE
     2012 audit_sys_operations           FALSE                          FALSE                          TRUE
     2030 db_domain                      boeblingen.de.ibm.com          boeblingen.de.ibm.com          FALSE
     2032 distributed_lock_timeout       60                             60                             TRUE
     2048 max_shared_servers                                                                           TRUE
     2065 use_dedicated_broker           FALSE                          FALSE                          TRUE
     2077 remote_dependencies_mode       TIMESTAMP                      TIMESTAMP                      TRUE
     2084 plsql_code_type                INTERPRETED                    INTERPRETED                    TRUE
     2086 plsql_debug                    FALSE                          FALSE                          TRUE
```

An interesting parameter shown in the previous screen capture is the `distributed_lock_timeout` parameter with a default value of `60` seconds. In some cases you might observe lock contention on some tables. Although often considered, increasing this value is typically not a good choice, because it hides the original problem that a transaction is holding a lock for a long time.

By increasing the `distributed_lock_timeout` value, you only wait longer on the blocked lock, and in the meantime you might acquire more locks that are not released, blocking other transactions. The main problem is actually the transaction, which does not free the lock you wanted to acquire in the first place. This problem is what needs to be solved.

Ask the following question: Why does a transaction hold a lock for over 60 seconds?

The answer might be related to the application (for example, calling a web service synchronously, which acts like it never returns), related to the volume (for example, of 80 million records in a table, nearly 99.9% are already related to completed process instances), or related to limited infrastructure (for example, a slow disk in the server).

Number of process instances in the system

Housekeeping issues for completed process instances is a common problem. IBM BPM is a runtime product and not intended for archiving. Therefore, when process instances complete they should be deleted from the system as quickly as possible. You cannot keep years of business data in an IBM BPM system and maintain good performance without major investments in hardware, databases, and other parts of the system.

Part 1 of this series discussed SQL statements that give you an overview of what is currently stored in the `BPMDB` database. For more details on the interpretation, see Part 1.

For Oracle versions of the SQL statements, see the following examples. You can download sample code from Github at http://github.com/stephan-volz/database-scripts/blob/master/Oracle_ProcessInstances_Tasks.txt.

The following SQL samples count the process instances and tasks in your database and group them by state and name:

Listing 3. SQL statements to query the process instances and tasks sorted by snapshot ID and state

```
-- Provides process instances grouped by process instances state
select snap.name as "name",
       bpd.snapshot_id as "snapshot id",
       code.NAME as "status",
       count(bpd.EXECUTION_STATUS) as "number of instances"
from LSW_BPD_INSTANCE bpd
       left outer join lsw_bpd_status_codes code on
 code.STATUS_ID = bpd.execution_status
        left outer join lsw_snapshot snap on
 bpd.snapshot_id = snap.snapshot_id
group by snap.name, bpd.snapshot_id, code.NAME
order by snap.name, bpd.snapshot_id, code.NAME ;

-- Provides task states grouped by task steps
select snap.name as "snapshot name",
       t.SNAPSHOT_ID,
       t.SUBJECT as "subject",
       code.NAME as "status",
       COUNT(t.STATUS) as "number"
from lsw_task t
       left outer join LSW_TASK_STATUS_CODES code on
 code.STATUS_VALUE = t.STATUS
       left outer join LSW_SNAPSHOT snap on t.snapshot_id =
 snap.snapshot_id
group by  snap.name, t.SNAPSHOT_ID, t.SUBJECT, code.NAME
order by snap.name, t.SNAPSHOT_ID, t.SUBJECT;
```

The connection between process instances and closed tasks is shown in the following SQL statements. The following SQL statements provide completed process instances and the associated tasks.

Listing 4. SQL statements to query the process instances and associated tasks sorted by snapshot ID and state

```
-- Provides completed process instances and associated tasks
select     snap.name as "snapshot name",
           bpd.snapshot_id as "snapshot id",
           count(distinct bpd.bpd_instance_id) as "number
 completed PI",
           count(ta.status) as "number of closed tasks"
from LSW_BPD_INSTANCE bpd
        left outer join lsw_snapshot snap on
 bpd.snapshot_id = snap.snapshot_id
        left outer join lsw_task ta on bpd.bpd_instance_id
 = ta.bpd_instance_id
where bpd.bpd_instance_id = ta.bpd_instance_id and
 bpd.execution_status = 2
group by snap.name, bpd.snapshot_id;
```

```
– Provides active process instances and associated closed tasks
select  snap.name as "snapshot name",
        bpd.snapshot_id as "snapshot id",
        count(distinct bpd.bpd_instance_id) as "number
 running PI",
        count(ta.status) as "number of closed tasks"
from LSW_BPD_INSTANCE bpd
        left outer join lsw_snapshot snap
 on bpd.snapshot_id = snap.snapshot_id
        left outer join lsw_task ta on bpd.bpd_instance_id
 = ta.bpd_instance_id
where bpd.bpd_instance_id = ta.bpd_instance_id and
 bpd.execution_status = 1 and ta.status = 32
group by snap.name, bpd.snapshot_id;
```

The results give a good indication by the ratio between completed and running process instances, if a cleanup of completed process instances is required. Always check completed process instances to prevent performance problems or additional costs for high end hardware.

> Running custom queries against the IBM BPM product tables is not supported (unless requested by IBM Support for diagnostic reasons). The shared statements provided as part of this article are for diagnostic purposes only, to get a better understanding of the data distribution. Typically, custom queries should not have a big affect on the system performance. However, the number of executions matters. Any query uses resources on the system.

Amount of used storage space

IBM BPM uses binary large objects (BLOBs) to store larger object information. The database system does not always release the space used by process instances that you deleted. This issue can lead to large table sizes without any real information inside (for example, 30 GB for a nearly empty table). If this happens, you need to reorganize the data. If you are using Oracle secure files, reorganizing can get more complicated. For secure files, there is no published compaction of BLOBs (a reduction of the space used by the BLOB data). Data needs to be exported, deleted, and imported. For ways to complete the compaction with internal packages, contact the Oracle support team.

To verify how much space is actually used, see the statements in the following screen capture:

```
SQL> select segment_name from dba_lobs where table_name = 'LSW_STORED_SYMBOL_TABLE' and owner = 'BPMDB_USER' ;

SEGMENT_NAME
--------------------------------------------------------------------------------
SYS_LOB0002401969C00003$$

SQL> select sum(bytes)/1024/1024/1024 from dba_segments where segment_name = 'SYS_LOB0002401969C00003$$';

SUM(BYTES)/1024/1024/1024
-------------------------
               325.296743

SQL> select sum(dbms_lob.getlength(STORED_SYMBOL_TABLE))/1024/1024/1024 from BPMDB_USER.LSW_STORED_SYMBOL_TABLE ;

SUM(DBMS_LOB.GETLENGTH(STORED_SYMBOL_TABLE))/1024/1024/1024
-----------------------------------------------------------
                                                3.74264413
```

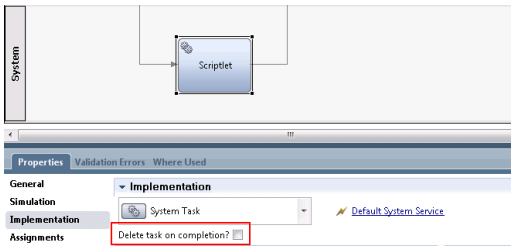
You can see the space used by the segment in red (325 GB) and the required space in green (about 3.7 GB). This is a significant difference, which is caused by the BLOB handling on the database side. In this case, a reduction of the space used by the BLOB data definitely makes sense.

The `LSW_STORED_SYMBOL_TABLE` in the example is an interesting table. It stores variable values during processing. Depending on the way you use local variables in the process design, you can create a contention on the `LSW_STORED_SYMBOL_TABLE` and on the execution context tables. Make sure to reduce the number of used `tw.local` variables in your application code, because you access the `LSW_STORED_SYMBOL_TABLE` and the execution context each time you reflect any changes.

For your JavaScript code, read the `tw.local` variables that you need, and assign them to variables that are used for the operations in the JavaScript code. Then, write the values back to the `tw.local` variables. This approach is especially critical if there are loops involved in the processing.

System tasks

In addition to human tasks, IBM BPM has system tasks that are directly run by the system. The behavior is influenced by the `Delete task on completion` property for system tasks, as shown in the following screen capture.



The setting deletes system tasks immediately after their completion, instead of deleting them at the end of the process lifecycle when the process instance is deleted. In some cases these completed system task entries contribute to more than 40% of the stored data in the `LSW_TASK` table. Starting in IBM BPM V8.5, the setting is enabled by default.

When you are aware of a large number of completed system tasks that are no longer needed, you can change the application and slowly remove the existing system tasks. To identify the affected database entries, use the following statement.

Listing 5. SQL statements to identify potential system tasks and their status

```
select user_id, activity_name, status, count(*)
 from LSW_TASK group by user_id, activity_name, status;
```

For more details, see the section about system tasks in Part 1.

Number of stored snapshots

Another potential area for a bottleneck is the number of snapshots that are stored in a system. (Make sure to check toolkit snapshots, which are frequently neglected.) In this context, you can see why a production system load is best on an IBM BPM Process Server instead of an IBM BPM Process Center. On the Process Center, there are frequent application changes from the connected application developers, possibly causing a large number of snapshots on the system.

Also pay attention to the tuning of snapshot context cache and branch context cache. These caches have a default value of 64. If the number of active snapshots or branches is higher than 64, follow the advice in Tuning branch and snapshot cache sizes in IBM Business Process Manager.

The best strategy is to avoid any unneeded data. Before deleting snapshots, check the Issues with BPMDeleteSnapshot and BPMSnapshotCleanup commands in IBM Business Process Manager (BPM) support document.

JDBC driver versions

The JDBC driver, although rarely used for troubleshooting, can be important for database troubleshooting. Oracle ships two different versions of a JDBC driver, but only drivers that end with `_g.jar` include debugging code for driver information during Java virtual machine (JVM) tracing.

After adding the debug driver on the file system and changing the WebSphere configuration settings to use the new driver file (JDBC Provider settings), there are two trace settings you can use to enable JDBC driver logging: `WAS.Database=all` or `com.ibm.ws.oracle.logwriter=all`.

Also, you must enable a JVM setting, as described in How to enable Oracle JDBC Driver tracing with WebSphere Application Server and Collecting JDBC Driver Tracing in WebSphere Application Server.

In some cases the driver version by itself is an issue. Make sure that you use the latest version and the same version of the driver as the database server. Check the IBM BPM system requirements and the Oracle specification to learn about supported drivers.

Hardware, environment limitations and performance tests

Teams working with IBM customers often find systems with no system scaling or load testing. It is essential for a production system to monitor the general system performance including CPU, memory, disk I/O, and network throughput. Several tools are available for system scaling and load testing. For troubleshooting on Unix systems, consider the `nmon` tool.

For more guidance, see the following IBM Redbooks publication: IBM Business Process Manager V8.5 Performance Tuning and Best Practices.

Monitoring is not a one time event. It must be in place permanently with a team member checking the data on a regular basis. Otherwise, there is no real value or problem prevention. The data from monitoring can make troubleshooting easier. With regular monitoring in place, your team can predict system use and potential bottlenecks (for example, a full hard disk or a load increase). For example, a system might be scaled perfectly today, but next month there might be a new batch job with 5 million new process instances.

When different teams are working with IBM BPM, the database, or the network, it is essential that all teams are aware on the availability for a required service. When the whole system is blocked, verify that the database is started, the Lightweight Directory Access Protocol (LDAP) server can be called from the IBM BPM server, and any required back-end system is reachable. A small

change like a port alteration or a firewall update can block processing. Don't rely on a problem ticket system. Make sure that everything is working on the systems.

IBM teams that work with IBM BPM customers often see development teams that could have already solved situations on their own - if they had only put these monitoring steps in place.

When the connection to the database is not working well, or a large number of connections are established, and the processing is slow, you can end up with out of memory situations. These problems also occur when the connection pool is completely used up. The database connections show up in the heap memory dumps and Java core files, but you still must determine the origin of the problem.

A large memory usage on the IBM BPM side might be caused by imprecise application queries that request millions of rows as a result set. IBM BPM product lifecycle improvements limit the data that is transferred between the IBM BPM system and the database by the product. Make sure you are using the latest version of IBM BPM to benefit from the latest improvements.

Also consider connection pool sizes. If you run load tests on an application or system and run out of connections, you might need to change the connection pool size to a higher value. For valuable tips for starting values, see IBM Business Process Manager V8.5 Performance Tuning and Best Practices.

For a well performing database system, the I/O speed for the transaction logs storage for the database is important. Consider using a solid-state drive (SSD).

If you follow all the previous information, you solve most performance problems with IBM BPM and an Oracle database. In some cases you might need to collect further data. The following sections describe the generally required data on both the Oracle database and IBM BPM side and give examples of common troubleshooting use cases.

Detailed analysis: Solving performance problems with the Oracle database

On an Oracle database system, the most prominent tool for troubleshooting performance issues is an Automatic Workload Repository (AWR) report. This report is based on automatically collected database snapshots that are taken by default every hour and kept for 8 days. (Do not confuse the snapshots taken on an Oracle database with the snapshots from a IBM BPM system.) The automatically collected database snapshots must be completed before you create an AWR report.

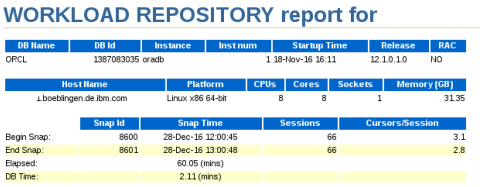
Because an AWR report requires an additional license, there is an alternative report named StatsPack that is included in the Oracle Database Standard Edition. The following sections describe AWR reports, because they are the most common reports for IBM BPM systems that use Oracle. However, you can create similar reports with StatsPack by sharing an Oracle Statspack report instead of an AWR report.

Oracle example 1: AWR reports

A few sections in an AWR report are of special interest with your IBM BPM environment. An Oracle database has a wide facet of tuning areas, but this tutorial focuses only on the most frequently seen problems in the context of an IBM BPM environment. Consider other areas on a case by case basis.

When reviewing an AWR report, make sure that the covered time period also matches the time when a problem occurred in your IBM BPM system. The AWR report is generated based on at least two snapshots. In most situations, a one hour time period best fits the requirements to analyze a specific performance problem. Always check the top of an AWR report for the time period for the generated report. Make sure it matches the time the problem occurred, and correlate the observations with the time period. For example if you see 5000 SQL statements, this number might be large for a 5 minute period, but normal for 10 hours of a work day.

The following screen capture shows an example AWR report with a one-hour time period.



WORKLOAD REPOSITORY report for

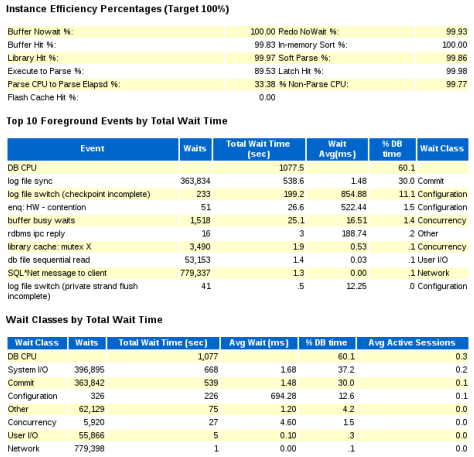
DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
ORCL	1387083035	oradb	1	18-Nov-16 16:11	12.1.0.1.0	NO

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
⊥boeblingen.de.ibm.com	Linux x86 64-bit	8	8	1	31.35

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	8600	28-Dec-16 12:00:45	66	3.1
End Snap:	8601	28-Dec-16 13:00:48	66	2.8
Elapsed:		60.05 (mins)		
DB Time:		2.11 (mins)		

Instance efficiency and top wait event information

The new Oracle Wait Interface gives you a more detailed and better understanding than the earlier information of the buffer pool hit ratio. However, you also can review the **Buffer Hit** ratio that is listed in the AWR report, shown in the following screen capture.



Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	100.00	Redo NoWait %:	99.93
Buffer Hit %:	99.83	In-memory Sort %:	100.00
Library Hit %:	99.97	Soft Parse %:	99.86
Execute to Parse %:	89.53	Latch Hit %:	99.98
Parse CPU to Parse Elapsd %:	33.38	% Non-Parse CPU:	99.77
Flash Cache Hit %:	0.00		

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
DB CPU		1077.5		60.1	
log file sync	363,834	538.6	1.48	30.0	Commit
log file switch (checkpoint incomplete)	233	199.2	854.88	11.1	Configuration
enq: HW - contention	51	26.6	522.44	1.5	Configuration
buffer busy waits	1,518	25.1	16.51	1.4	Concurrency
rdbms ipc reply	16	3	188.74	.2	Other
library cache: mutex X	3,490	1.9	0.53	.1	Concurrency
db file sequential read	53,153	1.4	0.03	.1	User I/O
SQL*Net message to client	779,337	1.3	0.00	.1	Network
log file switch (private strand flush incomplete)	41	.5	12.25	.0	Configuration

Wait Classes by Total Wait Time

Wait Class	Waits	Total Wait Time (sec)	Avg Wait (ms)	% DB time	Avg Active Sessions
DB CPU		1,077		60.1	0.3
System I/O	396,895	668	1.68	37.2	0.2
Commit	363,842	539	1.48	30.0	0.1
Configuration	326	226	694.28	12.6	0.1
Other	62,129	75	1.20	4.2	0.0
Concurrency	5,920	27	4.60	1.5	0.0
User I/O	55,866	5	0.10	.3	0.0
Network	779,398	1	0.00	.1	0.0

A good ratio is above 90%, and a better ration is above 95%. After a restart, the buffer needs to be filled, so the ratio is not good right after a restart. Even with a good ratio, you can still have problems on the system, so analyze the Oracle Wait Interface.

With wait statistics in Oracle, it is easy to see how a database system spends most of its time. First check the **Top 10 Foreground Events by Total Wait Time** section and the **Wait Classes by Total**

Wait Time section. IBM teams working with customers have seen situations with 5 digit numbers for an AWR report that covers one hour, quickly showing where the problem is located.

The previous screen capture also shows that most of the time is spent for waiting on CPU. Corresponding data demonstrates that it is important to monitor system resources to know about problems like a CPU contention before your end users notice issues.

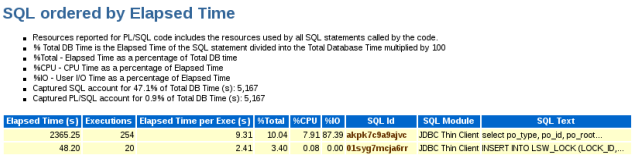
Nearly all wait classes in the Oracle Wait Interface sections like `Wait Classes by Total Wait Time` are experienced. Therefore, an AWR report is the basic report to check first, especially when there is a performance problem somewhere on IBM BPM or the database side. In several cases, the queries in the AWR report might also give some hints on the IBM BPM issues.

SQL statement information

The next important section in the AWR report is **SQL Statistics**, where you can see the execution times for SQL statements and the data that is requested. Execution times have different units. Sometimes the time is calculated for all the executions of a statement, and in other cases, it is for a single statement execution. Check what is listed in the specific section before determining a conclusion.

Everything in this section for a single SQL statement execution that takes longer than a second warrants a further look. There are valid cases that can take a longer time. For example, you might collect statistics for large tables that can also be listed here. Or, you can ignore a monthly report generated only once a month. However, if you see statements that take significant time or request millions of records, and you see these statements issued multiple times, investigate further and address problems.

The following screen capture shows an example **SQL Statistics** section.



SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100
- %Total - Elapsed Time as a percentage of Total DB time
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Captured SQL account for 47.1% of Total DB Time (s): 5,167
- Captured PL/SQL account for 0.9% of Total DB Time (s): 5,167

Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO	SQL Id	SQL Module	SQL Text
2365.25	254	9.31	10.04	7.91	87.39	akpk7c9a9ajvc	JDBC Thin Client	select po_type, po_id, po_root...
48.20	20	2.41	3.40	0.08	0.00	01syg7mcja6rr	JDBC Thin Client	INSERT INTO LSW_LOCK (LOCK_ID,...

The two statements take 9.31 and 2.41 seconds for each execution. Check this area on the affected system. Missing indexes and slow I/O all can play a role in this problem. As long as the database response for the query is slow, IBM BPM is not likely any faster.

Segment statistics information

The Segment statistics section of the AWR report is the place to look for large numbers of physical reads and writes, table scans, and row lock waits. Large amounts of physical reads and writes should also be reflected in a lower buffer pool hit ratio. Physical in this context means the data needs to be fetched from disk, which is much more expensive than a fetch from memory.

Table scans read a full table. In some cases this approach makes sense. If the table is not big, or you need most of the table content, an index access does not make sense. There are no wrong reasons to run full table scans, but it depends on your situation.

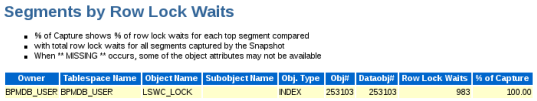
In several cases, the execution time for a full table scan is long, so it makes sense to check which tables are affected by the table scans. A frequently seen table index scan happens on the `IDX3_GGMX` index, which is needed for the group management.

If you see a specific wait class that is using a large amount of time, check for valuable hints with a search engine. In most cases, you can find a similar issue with a corresponding solution.

There are further improvements for user and groups. See the Teams in Business Process Manager V8.5 tutorial series.

Lock timeouts or row lock waits are not as common with Oracle as they are with DB2. However, it is still necessary to know which table contains the lock contention. Two typical tables with this issue are `LSW_LOCK` and `LSW_BPD_INSTANCE`.

Lock contention on the `LSW_LOCK` table is frequently seen, as shown in the following screen capture.



Segments by Row Lock Waits

- % of Capture shows % of row lock waits for each top segment compared
- with total row lock waits for all segments captured by the Snapshot
- When ** MISSING ** occurs, some of the object attributes may not be available

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Obj#	Dataobj#	Row Lock Waits	% of Capture
BPMDB_USER	BPMDB_USER	LSWC_LOCK		INDEX	253103	253103	983	100.00

For more information, see the entry for Why do I see lock contention on BPM for the LSW_LOCK table? on dW Answers.

The lock contention on the `LSW_BPD_INSTANCE` table is most likely created by parallel paths in the application. Calling a synchronous service can lock the process instance for a significant amount of time, while another application activity fails to change the process instance information.

This kind of problem requires an application review or a trace to identify the activities that lead to the lock contention (described further in the Troubleshooting details for IBM BPM section).

Other useful information

There is much more information in an Oracle AWR report. This tutorial addresses three important sections for solving several performance related problems.

An AWR report also includes recommendations for memory sizing and information on wait classes and I/O bottlenecks. When you already know that delays are caused by the database, check with your database administrator to see if there are further tuning steps. In some cases it helps to consult the database administrator for IBM BPM issues, for example, if the hardware size does not allow more verbose tracing methods. You can gain further insights from the database side, if appropriate for your situation.

Oracle example 2: The Oracle alert.log file

If the problem is more related to function instead of performance, the AWR report most likely is not sufficient to debug the issue.

Oracle has another log file named `alert.log`, where larger issues are logged. Check and share this log with IBM Support if there is a function-related problem that you cannot solve by yourself. The log might contain information that affects performance but is not covered by the AWR report.

Oracle example 3: Access plan output

If you are already aware that a specific SQL statement is slow, an access plan output is required to see where the time is spent, for example, using the explain command, as shown in the following screen capture.

```
Execution Plan
----------------------------------------------------------
Plan hash value: 155744470

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                 |    12 |  1536 |     9  (34)| 00:00:01 |
|   1 |  SORT ORDER BY                |                 |    12 |  1536 |     9  (34)| 00:00:01 |
|   2 |   HASH GROUP BY               |                 |    12 |  1536 |     9  (34)| 00:00:01 |
|   3 |    MERGE JOIN                 |                 |    12 |  1536 |     7  (15)| 00:00:01 |
|*  4 |     TABLE ACCESS BY INDEX ROWID| LSW_PROJECT     |     6 |   258 |     2   (0)| 00:00:01 |
|   5 |      INDEX FULL SCAN          | LSWC_PROJECT_PK |    12 |       |     1   (0)| 00:00:01 |
|*  6 |     SORT JOIN                 |                 |    24 |  2040 |     5  (20)| 00:00:01 |
|   7 |      TABLE ACCESS FULL        | LSW_SNAPSHOT    |    24 |  2040 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

The access plan output can be influenced by the statistics, covered earlier. In most cases, a slow performance of a specific statement is connected to missing or invalid indexes, large data volume, wrong statistics, and the resulting incorrect access plans.

Oracle example 4: An Oracle 10046 or 10053 trace

A more powerful measure to dig deeper in performance problems of an Oracle database is a 10046 or 10053 trace.

To identify problematic statements in the first place, an AWR report is sufficient. If there are doubts about why a statement is behaving in a specific way and explain plans (how the database optimizer runs SQL statements) cannot help, these kind of traces can give further insights.

A common problem that arises in the IBM BPM context is a missing session because IBM BPM uses a shared database connection pool and a specific session cannot be identified. You need to enable a trace on the complete database, which can have significant performance impacts. Carefully consider the usage needs (consider off-shift hours) and the used trace level.

Alternatively, if you know the `sql_id` from the AWR report, you can limit the trace to this specific statement to reduce the performance impact of the trace for the database significantly and make it easier to find corresponding trace files.

The following table and the following code sample show the possible level settings for a 10046 trace and how to enable and disable it across the system. The main goal for the 10046 trace is to identify bottlenecks. The 10053 trace focuses more on the optimizer calculations. Both can be valuable, depending on your requirements.

Table 1. Trace levels that can be used with a 10046 trace

Trace level	Description

0	No trace. Like switching sql_trace off
2	The equivalent of a regular sql_trace
4	Same as level 2, but with bind variable values
8	Same as level 2, but with wait events
12	Same as level 2, but with bind variable values and wait events

The following code sample shows commands to enable and disable the 10046 trace with bind variables and wait times. Take special care to consider the performance impact.

Listing 6. Commands to enable and disable the 10046 trace with bind variables and wait times (level 12) on the system level

```
SQL> ALTER SYSTEM SET EVENTS '10046 trace name context
 forever, level 12';

            <recreate the issue>

SQL> ALTER SYSTEM SET EVENTS '10046 trace name context off';
```

It is also possible to limit a trace to a specific `sql_id` as shown in the following code sample. Replace the `sql_id` with the ID of the SQL statement. Use only one of the commands at a time, and disable tracing as shown.

Listing 7. Commands to enable a SQL statement specific trace for the 10046 or 10053 trace

```
SQL> -- SQL Trace (10046)
SQL> ALTER SYSTEM SET EVENTS 'sql_trace [sql:sql_id]
 bind=true, wait=true';

SQL> -- SQL Trace (10053)
SQL> ALTER SYSTEM SET EVENTS 'trace[rdbms.SQL_Optimizer.*]
[sql:sql_id]';
```

Consider this approach if a specific statement is causing problems. With the 10053 trace, you can gain insights on the used statistics as well as the cost calculations by the optimizer. (Consult your database administrator for these advanced steps.)

The following screen capture shows an extract of a 10053 trace where the cost is calculated. Based on these numbers the optimizer decides if a full table access or an index access costs less.

```
Access path analysis for LSW_SNAPSHOT
***************************************
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for LSW_SNAPSHOT[S]
  SPD: Return code in qosdDSDirSetup: NOCTX, estType = TABLE
  Table: LSW_SNAPSHOT  Alias: S
    Card: Original: 24.000000  Rounded: 24  Computed: 24.000000  Non Adjusted: 24.000000
  Scan IO  Cost (Disk) =   4.000000
  Scan CPU Cost (Disk) =   67383.040000
  Total Scan IO  Cost  =   4.000000 (scan (Disk))
                       =   4.000000
  Total Scan CPU  Cost =   67383.040000 (scan (Disk))
                       =   67383.040000
  Access Path: TableScan
    Cost:  4.001697  Resp: 4.001697  Degree: 0
      Cost_io: 4.000000  Cost_cpu: 67383
      Resp_io: 4.000000  Resp_cpu: 67383
```

Oracle example 5: ADDM and ASH reports

In addition to the AWR report, Oracle includes two other reports: the Automatic Database Diagnostic Monitor (ADDM) and the Active Session History (ASH).

This tutorial provides basic background on the interaction and troubleshooting and does not cover these reports. You use these reports in a similar way as the AWR report. The ADDM provides recommendations for performance improvements. The ASH report provides session history information.

In Oracle Database 12c, ADDM recommendations are included as part of the AWR report. Check the Oracle documentation for details.

These tools can give you further pointers on required changes or bottlenecks. The IBM Support team might request these reports to diagnose and solve specific issues.

Detailed analysis: Tracing the IBM BPM system

To correlate what is sent by the IBM BPM system and what is observed on the database side, you need a trace to know what is happening. There are several different trace settings available.

The following basic trace covers some basic SQL statement information:
`WAS.clientinfopluslogging=all:org.springframework.jdbc.*=all`

The `WAS.clientinfopluslogging` trace is required to log the SQL statements and transaction boundaries. Sometimes there are different versions of `org.springframework.jdbc` (for example limitations to `org.springframework.jdbc.core`).

If you are using the Java Spring Framework, the core part traces the values that are passed to the SQL statement. The database trace subsection tracks when a connection from the data source is acquired and returned. This is valuable information especially for tracking execution. Always use the `org.springframework.jdbc.*=all` trace setting.

To include the background from where the statement was originally triggered, you also must add the IBM BPM product tracing:
`WLE.*=all:com.ibm.bpm.*=all:WAS.clientinfopluslogging=all:org.springframework.jdbc.*=all`

(Adding the IBM BPM product tracing might affect performance, but is the only way to analyze problems efficiently.)

It is important to check the number of trace files and the size of each trace file. A good trace should cover more than a few minutes and include the problem in question. If an activity lasts 15 minutes, make sure that the trace covers at least 20 minutes, so the complete navigation history is covered.

In addition, IBM BPM and the database must be tracked at the same time frame. (AWR reports can be only generated after the problem scenario, but the corresponding snapshots are by default kept for 8 days and written every hour.)

In some situations you might need to trace the following additional information:

- Track database connection pool usage with the following additional settings:
  `WAS.j2c=all:RRA=all:Transaction=all`
- Security related problems combined with the database are best addressed with the following additional settings of VMM and security traces:
  `com.ibm.websphere.wim.*=all: com.ibm.ws.wim.*=all:`
  `com.ibm.wsspi.wim.*=all:`
  `com.ibm.ws.security.*=all:`
  You can use different publications that use reduced security trace settings. Evaluate the performance impact compared to the value gained out of the traces and the problem investigated. As a general recommendation, check what is really needed and can be handled by your system. If you are in doubt it's better to collect more data than too little data.
- The following additional trace setting from WebSphere Application Server logs the JDBC Resource adapter code from WebSphere Application Server, which also includes the adapter code and the SQL statement information:
  `com.ibm.ws.rsadapter.jdbc.*=all`
  This trace setting collects data that is not frequently required from a IBM BPM perspective. The reduced set for tracking only prepared SQL statement is a good alternative:
  `com.ibm.ws.rsadapter.jdbc.WSJdbcPreparedStatement.*=all`
  Not all statements must be executed as a prepared statement.
- Enable a deeper analysis with JDBC driver tracing from the IBM BPM server side. For Oracle, enable the JDBC driver trace on the IBM BPM server as previously described.
  Nearly all cases of database issues can be solved without such a trace. However, a few very special situations required this kind of tracing, so be aware that it exists.

Data required to debug a BPMDB related issue

This section summarizes the minimum data collections (previously described) to debug a problem.

Consider the following summary of what data is required to debug an issue related to the `BPMDB` database:

1. Output of the SQL statements, which are available in the SQL statement file on Github: http://github.com/stephan-volz/database-scripts/blob/master/Oracle_ProcessInstances_Tasks.txt.
2. An AWR or Statspack report from the time you first noticed the problem, in a short timeframe, such as one hour. Your database administrator should be able to advise on the database findings. Share the information with the IBM Support team.
3. An explain output or even a database trace (if there is already a performance problem for a specific statement).
4. A trace from the IBM BPM side while the problem is recreated.

IBM BPM example 1: Slow performance of database queries

In several cases, when slow performance is observed for database queries, at least there is a suspicion that there is a delay from the database side, but not all cases are clear from the beginning.

Sometimes a process participant logs into the IBM BPM Process Portal and the page does not load for a long time. Is this issue caused by a large number of entries in the database, a browser issue, a high load on the IBM BPM system, or another cause?

If you think the problem might be related to the database, cover the IBM BPM and the database side. In the worst case you might collect some data in vain, but you save time instead of focusing on only one side. From this perspective, it is worth it to collect a larger amount of data to come to a quick conclusion.

The following trace setting should cover most of the problems between the IBM BPM product and a database system, including Oracle:

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

If you already identified a slow running query, you can create an explain output, The Oracle Enterprise Manager Web-based interface of your Oracle database system or the developer tool SQL Developer can help you visualize corresponding execution plans.

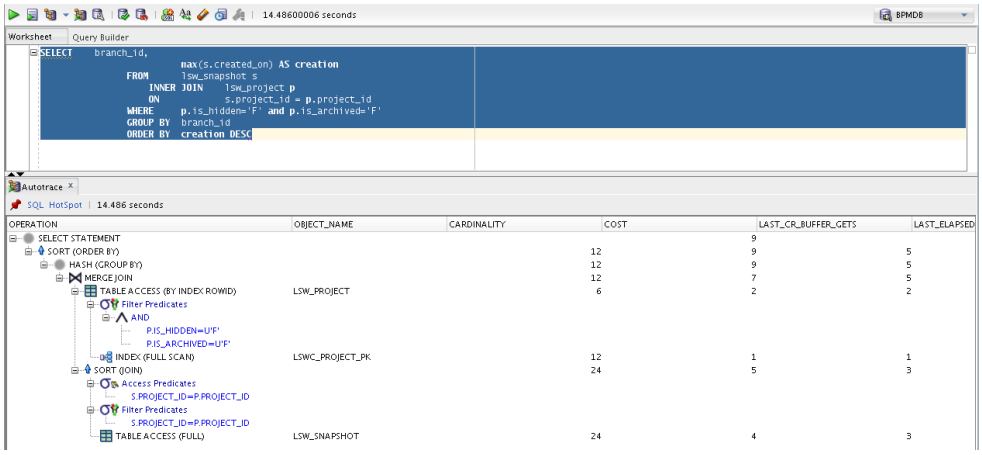
Check for `Table access full`, larger numbers for costs, and use of temp space.

Your database administrator can help on this kind of question.

Not every table scan is bad. If you require nearly all data from a table, a table scan can be much more efficient than an index access.

In some situations, the problem is triggered by the application, for example requesting millions of records that are not needed in the end.

The example in the following screen capture shows a `Table access full` on 24 entries of the `LSW_SNAPSHOT` table, which should not be a big concern.



IBM BPM includes several indexes. You might not see a benefit by a specific index included with the product, but you might not be aware of scenarios where you need it.

The IBM Support team expects indexes shipped with IBM products to be in place. Deleting any of these default indexes results in performance issues that cost both your organization and the support engineers an unnecessary waste of time to realize that these indexes are removed, to get them back, and then start to tune the system.

To improve the performance in the specific business case, it can be still necessary to create additional custom indexes.

Oracle provides several tuning tools, depending on your license level. One well known tool to give recommendations is the ADDM report, but there are others like the SQL Tuning Advisor. If you see a slow SQL statement, use the tools to get an idea of what you can improve.

Be aware that SQL statements related to IBM BPM use constructs like `select for update` to prevent any modifications on corresponding rows during a running transaction. At the end of the transaction, the changes (for example, a single process instance) are committed, and the lock is released. Long running `select for update` statements indicate that the process design should be improved.

For example, if you are working on a process instance and call an external service synchronously, the service has a long response time. While the service runs, the database keeps the lock for the `select for update` statement on the `LSW_BPD_INSTANCE` table. When the processing is completed, the final commit is executed and the lock is released. For this kind of problem, an explain output from the database system does not help, because the processing is blocked by the activities executed by IBM BPM.

IBM BPM example 2: Slow performing saved searches

If you are using searchable variables in your application, you might see performance issues with corresponding search queries.

Take a different approach for a client and a server side view of the problem.

If you are using the responsive Process Portal with IBM BPM V8.5.7 and more recent releases, see the Performance recommendations for the new IBM Business Process Manager V8.5.7.0 Process Portal support document.

If you use saved searches frequently, troubleshooting is simple for the Process Server side. Check that you have implemented search accelerator tables (also called pivot tables) for search optimization with Oracle. Do not take this approach for a Process Center environment. For each change of searchable variables in the deployed applications, the two pivot tables must be dropped and recreated. (It is not sufficient to only call the `DataLoad` script.)

Use the `schemaGenerator` and `DataLoad` scripts, as described in the JR42110 support document and the Saved search acceleration tools documentation.

Because a development environment has frequent changes, maintaining the tables becomes an issue. This approach is more for production, QA, and test environments. You see significant

impact, depending on the circumstances of application design, usage, and hardware. In some situations the IBM team saw improvements of a factor 100.

To track the client side, especially the browser, a tool like Fiddler gives you corresponding long calls from a browser perspective. You can map this info to the full IBM BPM trace, which was previously explained. If there is a delay in the server response, you see it quickly. Delays in the browser for issues like JavaScript processing are more difficult to track, but you can see them from the Fiddler time line.

IBM BPM example 3: Instrumentation logs and a large amount of queries

As a last point, which we already covered in the DB2 articles in more detail, instrumentation logging is an option for troubleshooting. See http://www.ibm.com/support/docview.wss?uid=swg21613989.

The instrumentation logs give insights on the cache usage for IBM BPM. Many of the same SQL statements sent to the database might be caused by a cache on the IBM BPM side that is too small. Instrumentation logs are a good tool to capture the cache misses and hits when there are a large number of the same statements running. Based on the results, you can adjust the size of your caches for your workload.

Make sure you have the latest version of IBM BPM to avoid issues with a large number of queries for the `LSW_MANAGED_ASSET` table. Check the currently set value and how many assets are expected. The latest IBM BPM releases also print the number of assets as part of the recommended general IBM BPM trace.

Conclusion

You learned about several different tools and approaches to troubleshoot your `BPMDB` database when you use Oracle with IBM BPM. You can use most of these tools for other database issues, even for scenarios without IBM BPM.

This tutorial addresses IBM BPM releases until IBM BPM 8.5.7. Future releases might require adjustments to the SQL statements, although no major changes are expected.

Now you are equipped with skills to identify potential issues quickly, and you are prepared to troubleshoot issues with your own IBM BPM environment. You can practice by testing the shared examples.

To learn about troubleshooting in environments with IBM DB2, see Part 1 and Part 2 of this series.

Acknowledgments

The author thanks Bonnie Brummond and Hendrik Engler for their review and comments. Any included error is the fault of the author only.

Related topics

- IBM Business Process Manager database troubleshooting, Part 1
- IBM Business Process Manager database troubleshooting, Part 2
- Purging data in IBM Business Process Manager
- Teams in Business Process Manager V8.5 series
- Reading and decoding instrumentation files (support document)
- 5 Things to Know About IBM BPM Performance Tuning (blog)
- Tuning connection pools (product documentation)
- IBM Business Process Manager V8.5 Performance Tuning and Best Practices (an IBM Redbooks publication)