CA Unified Infrastructure Management

CA UIM Database Best Practices for MySQL

Version 1

This documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be disclosed by you or used for any purpose other than as may be permitted in (i) a separate agreement between you and CA governing your use of the CA software to which the Documentation relates; or (ii) a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2018 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Contents

Chapter 1: Introduction

This guide covers best practices for deploying, tuning, triaging, and maintaining the UIM MySQL database, also known as the Unified Infrastructure Management (UIM), as deployed on MySQL.

The guide includes new and existing information from UIM documentation, support articles, development tools and other internal and external sources. It organizes the information into the following main sections:

- General description of the UIM
- Prerequisites
- Best Practices
- Best Practices for MySQL
- Performance analysis
- Troubleshooting
- Updating Table Indexes
- Advanced NIS indexing
- SQL Tools and Scripts
- Monitoring the Instance Health with Reports

This document does not attempt to fully document the programming interfaces, theory of operation and structure of CA UIM. The documented is intended to be useful for practical issues in deploying, maintaining and tuning database.

Background on the Unified Infrastructure Management

The UIM database is integral and critical to overall CA UIM system operation and performance. The CA UIM solution requires a database to store the QoS, service level, configuration, alarm (optional) and other data that is collected, processed, and displayed by the system.

The UIM database was originally introduced into the CA UIM product to hold historical QoS data derived from raw data to enable Service Level Monitoring (SLM) features. With the introduction of expanded reporting and dashboard features in the Unified Management Portal (UMP), the SLM took on an expanded role and was re-named the Nimsoft Information Store, or *NIS*. During the CA UIM 8.0 release, the NIS became known as the UIM Database.

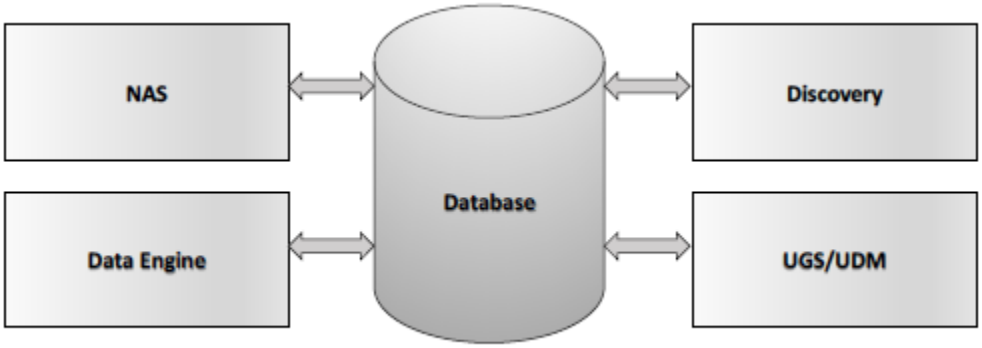
As with any OLTP (Online Transactional Processing) application, the volume of stored data increases over time. When scaled to the needs of large enterprise and managed service providers (MSPs), any database will require periodic maintenance and performance tuning. CA UIM customers, partners, and developers have generated a number of best practice procedures and tuning tips.

Note: Two areas outside the scope of this document are database software upgrades and cross-platform data migration. We recommend you speak with your CA UIM sales engineer regarding these activities.

Chapter2: The UIM Database

These sections provide an overview of the schema and select components of the UIM Database.

UIM Database Within CA UIM (Simplified UIM Database Architecture)

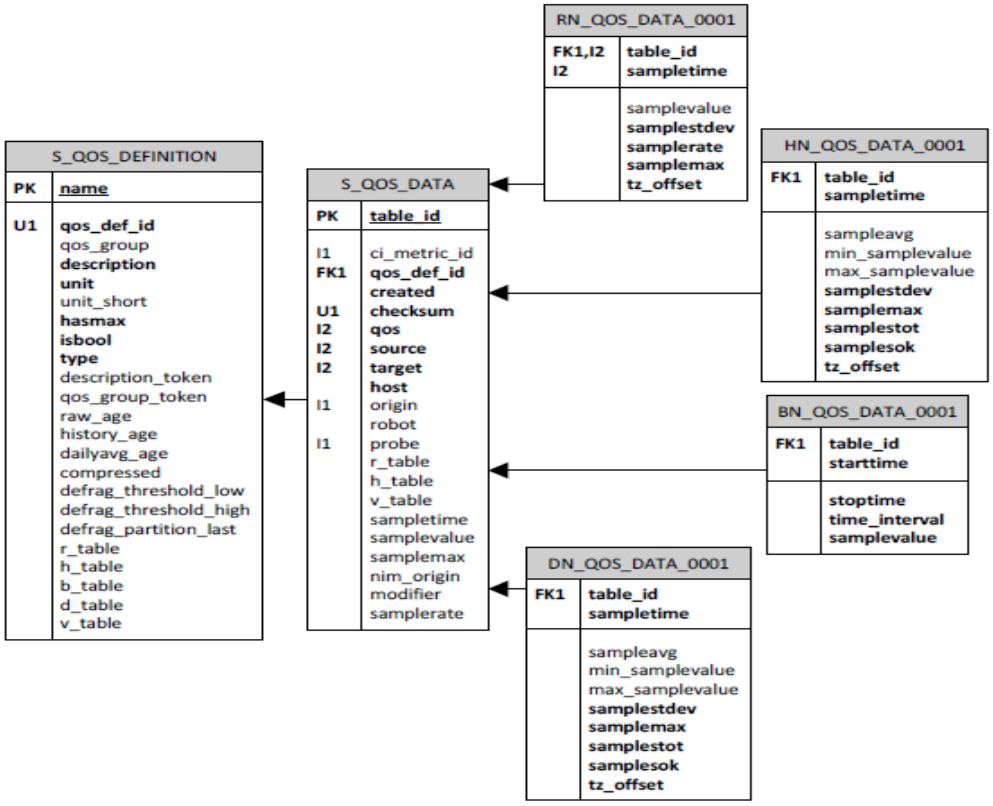


UIM Database Schema Overview

For a picture of how the UIM is structured, below is an abbreviated view, listing the tables in the UIM database. The full view is many pages long, with hundreds of tables listed.

Table	Children	Parents	Columns	Rows
Account_			16	1
Address			18	0
AnnouncementsDelivery			7	30
AnnouncementsEntry			17	0
AnnouncementsFlag			5	0
AssetCategory			14	0
AssetCategoryProperty			9	0
AssetEntries_AssetCategories			2	0
AssetEntries_AssetTags			2	0
AssetEntry			25	127
AssetLink			9	
AssetTag			9	0
AssetTagProperty				14,654
AssetTag[illegible]_0004			8	890,855
[illegible]QOS_DATA_0005			9	890,855
RN_QOS_DATA_0006			9	892,579
RN_QOS_DATA_0007			8	890,856
RN_QOS_DATA_0008			8	890,856
RN_QOS_DATA_0009			9	890,856
RN_QOS_DATA_0010			9	890,856
RN_QOS_DATA_0011			9	890,857
RN_QOS_DATA_0012			9	890,857
RN_QOS_DATA_001[illegible]			9	94,946
RN_QOS_DA[illegible]			[illegible]	94,946
RN_[illegible]ite			7	
WikiNode			11	0
WikiPage			11	2
WikiPageResource			23	2
WorkflowDefinitionLink			4	2
WorkflowInstanceLink			10	0
WSRP_WSRPConsumer			10	0
WSRP_WSRPConsumerPortlet			9	0
WSRP_WSRPProducer			7	0
			7	1
410 Tables			**3,343**	**18,729,661**

Relationships Between Data Engine Tables



Database Tables

These tables are of primary interest:

Table name or pattern	Type of data held
S_QOS_DATA	QoS data used by the data_engine
RN_QOS_DATA_xxxx	Raw QoS data (one RN_table for each qos_def_id)
HN_QOS_DATA_xxxx	Aggregated (1hr interval) QoS data
DN_QOS_DATA_xxxx	Aggregated (1 day interval) QoS data
BN_QOS_DATA_xxxx	Baseline tables
CFG_*	ACE configuration information
CM_GROUP	NIS_Server/USM
CM_DEVICE	Discovery/configuration
CM_NETWORK	Discovery
CM_NIMBUS_*	Discovery/configuration
GRP_*	group_server, deprecated
ump* and QUARTZ_*	UMP and LifeRay
*_	Liferay
*_t_*	Tmp, dashboard_engine
NAS*	Alarms*
tbnLogging	Sp logging

tbnVersion	Version info
S_SLA_* and S_SLO_*	Service Level Agreement info

***Note:** Alarm data is held in a separate NAS database. If the NIS-bridge feature of NAS is enabled, alarms are replicated from NAS to NIS. UMP accesses alarm data from the NIS using the NIS-bridge feature.

RN_QOS_DATA_ Table Columns

The RN_QoS_Data_tables hold raw QoS data. This QoS data is written once and never updated. It is aggregated by 1 hour periods and stored in HN_QoS_Data_tables_xxxx. The table has the following columns:

TableID	Sampletime	Samplevalue	Samplestdev	Samplerate	Samplemax	Compressed	Tz_offset	inserttime

Column Name	Description
tableID	unique identifier; key for looking up time series data
Sampletime	time the sample was taken
Samplevalue	QoS value
Samplestdev	standard deviation of the sample
Samplerate	Rate of sampling
Samplemax	Maximum sample value (e.g. 100%)
Tz_offset	time zone offset

RN_tables Indexes

The default indexes on RN_tables are optimized for writing data:

Index	Description
Idx0	SAMPLETIME, TABLE_ID
Idx1	TABLE_ID, SAMPLETIME, TZ_OFFSET, SAMPLERATE

There is no primary key implemented on RN_QoS_DATA_tables as both tableID and sampletime can be duplicated.

About Table Partitioning

As RN_QoS_DATA_tables grow in size the time needed to order and index them increases, slowing performance. Subdividing tables into multiple partitions offers several benefits:

- Partitioning allows data loads, index creation and rebuilding, and backup/recovery to occur at the partition level rather than on the entire table.
- Partitioning improves query performance. In many cases, the results of a query can be achieved by accessing a subset of partitions rather than the entire table.
- Partitioning can significantly reduce the impact of scheduled downtime for maintenance operations.

UIM Database tables can be automatically partitioned if you are using MySQL Database. The partitioning scheme is a sliding window partition on sampletime with one partition per day. If partitioning is enabled, data is aged out of the RN tables by dropping the old partitions rather than deleting rows.

Manual Table Partitioning

Manual partitioning is achieved by running the data_engine partitioning stored procedure manually. You may wish to apply partitioning to your large tables at first. We define large tables as those having over 100 million rows. Partitioning in a selective manner gives you more control over when tables are partitioned as this process can take considerable time to complete.

Use the following approach to apply manual partitioning.
1. Disable the following probes:

- data_engine
- wasp
- dashboard_engine

2. Run the following for each table you wish to partition (one at a time is okay):

```
CALL spn_de_PartitionAdmin__PartitionTable
(
 'RN_QOS_DATA_0001',
 'RN',
'2015-07-14 11:28:49.883',
5,
@returnCode
);
```

3. After this has been done for the larger tables, select the **Partition data tables** check box in the Admin Console data_engine GUI to enable partitioning on the rest of the tables.

4. Re-enabling the following probes:

- data_engine
- wasp
- dashboard_engine

Chapter 3: Prerequisites

This guide assumes that UIM Server and the UIM Database are installed and running. We recommend that you review the requirements and prerequisites for proper UIM Server and database installation. For more information, see the article on Pre-Installation Planning available at wiki.ca.com/uim .

Note: These links are for UIM 8.2. Please refer to the documentation that corresponds to your specific version.

Supported Database Versions

The following database and OS versions are supported:

Database	Supported Operation System (64 bit only)
MySQL 5.6 and MySQL 5.7	Windows Server 2012 R2, Windows Server 2014, Windows Server 2016, Redhat, Ubuntu

Hardware Requirements

We recommend deploying the database on a dedicated physical server.

Database Server Requirements

While every situation is unique, the following deployment size categories give you a starting point for assessing your hardware requirements:

- **Small (One hub, fewer than 100 robots)** - Modest deployment, such as a proof-of-concept for a small business
- **Medium (Up to five hubs, fewer than 250 robots)** - Medium-scale deployment, such as a small government agency
- **Large (Up to twenty hubs, fewer than 500 robots)**
- **Major (Up to fifty hubs, fewer than 1000 robots)**
- **Over 50 hubs or over 1000 robots** - Consult with CA professional services or a CA UIM certified partner.

Deployment size	Processor *64-bit XEON-class, 2.0 GHz or better*	Memory	Storage recommendations
Small	One dual-core	8 GB	Obtain at least 1 TB storage for the database.
Medium	One or two quad-core	12 GB	Use RAID 10 (for speed and reliability).
Large	Two quad-core	12 GB to 18 GB	Spread database files across multiple disks to improve I/O.

			Choose drive subsystems with low latency/seek times, high spindle speeds, high interconnect bandwidth.
Major	Two quad- or eight-core	18 GB to 24 GB	Continually consider data redundancy, synchronization, and database growth.

About Database Performance

Relational database server performance is heavily affected by disk I/O performance and server bus bandwidth. Crowded VM hosts, clusters, or heavily shared storage in VM environments are not recommended for UIM database hosting.

CA recommends starting with at least 1TB of RAID 10 storage for the UIM Database. Also, consider spreading the database files across multiple disks (LUNs) to improve I/O performance. Choose drive subsystems with low latency and seek times, high spindle speeds, and high interconnect bandwidth.

The data redundancy/synchronization model needs to be considered on an on-going basis, taking into account the growth of the database. Selecting the right storage solution is beyond the scope of this document, we recommend that you discuss this with your storage vendor/VAR/consultant.

Chapter 4: General Best Practices

This section covers operational UIM Database best practices.

Best Practice	Comment
Read and observe documented pre-requisites and pre-install information	See the article on Pre-installation Planning available at wiki.ca.com/uim .
Always make a backup of your database before upgrading major CA UIM components (UIM Server and UMP)	Some upgrades contain a non-reversible upgrade script that changes the database structure of some tables.
Run get database information MySQL on a regular basis	Establish a baseline so that system changes can be easily seen and CA UIM support can quickly respond to issues
Use SHOW PROCESSLIST	This View provides information on status.
Set up periodic index maintenance	See the article on data_engine configuration and review the notes and warnings regarding the cost of setting up index maintenance.
Carefully consider the implications of database configuration settings	See the article on data_engine configuration.
Check database size	Use this query to check database size
Check for and correct index fragmentation on a regular basis.	Use get_index_fragmentation to check for index fragmentation. Automatic index maintenance can also be scheduled as described above.
Check for missing nodes in Dynamic Views	Find_missing_nodes_in_dynamic_views

Storage

This section covers storage considerations for UIM database.

Best Practice	Comments
Determine an overall storage strategy	Discuss a storage strategy with your storage vendor.

Backup

This section covers the UIM database backup best practices.

Best Practice	Comments
Plan and schedule regular backups of the database	
Test restore operation before it is needed	
Ensure sufficient disk capacity for backups	
Document backup and restore procedures	

Periodic Maintenance

This section covers the periodic maintenance of the UIM database.

Best Practice	Comment
Set up periodic index maintenance	See the article on data_engine configuration and review the notes and warnings regarding the cost of setting up index maintenance.
Identify skewed and outdated index and column statistics and make sure they are representative and current	Index statistics are used by the MySQL query optimizer to help it determine if and when an index should be used when executing a query.
Database and log file protection and management	
Temp data maintenance	
Data corruption detection	
Performance monitoring	

Chapter 5: Best Practices for MySQL

This section provides a checklist of values and properties to check when setting up and deploying MySQL database.

Checklist

This checklist is hierarchical in method, starting with hardware and OS settings, then MySQL instance, then the UIM database and its maintenance.

- Network Communication
- Cluster Environment
- Windows/Linux Configuration
- Disks configuration
- MySQL Instance Configuration
- Database Level Configuration
- Database Maintenance and Backup Strategy
- Additional Resources

Network Communication

This section covers the network communication best practices for UIM database.

For Windows:

Applicable Versions of Windows	Item	Recommendation
All	NIC full duplex	Network adapters and switch ports should have matching duplex levels or transfer speed settings. Full duplex provides better performance.
All	Network settings	Latest basic input/output system (BIOS) update for the server should be installed. Latest firmware update for the network adapter should be installed. Latest driver update for the network adapter MUST be installed.
All	NetBIOS and Server Message Block enabled	Disable NetBIOS and Server Message Block

		Important! Make sure NetBIOS is not in use.

Cluster Environment

This section covers the cluster environment best practices for UIM database.

Applicable Versions of OS and OS Version	Item	Recommendation
All	Cluster nodes hardware	Cluster nodes should have nearly identical hardware on all cluster nodes to simplify configuration and eliminate potential compatibility problems.
All	Memory adjustment	In an Active-Active environment, memory for the MySQL instances should be set in a way that the total memory in the weakest node is split between the nodes. This will ensure that when all instances failover to one node, they will be able to that quickly and with no memory issues.

Windows/Linux Configuration

For more information about Windows Server performance, see Performance Tuning Guidelines for Windows Server 2012 R2.

Applicable Versions of OS	Item	Recommendation
All	Latest service pack	Implement the latest service pack and hotfixes.
All	64-bit hardware and software	Required
All	Paging file	Pagefile should be 1~1.5 times the amount of RAM and should NOT be placed on a drive that contains database files. **Important!** We recommend that you create multiple page files on different disk partitions beside C:\ (or even different

		disk subsystems) for performance reasons.
All Windows	System properties > Advanced setting	**Processor scheduling:** Select Background services. **Memory usage:** Select Programs
All	Unnecessary Services / applications	Number of running apps and services should be minimal. Unnecessary services should be stopped and disabled (Messenger, wireless configuration, etc.).
All	Anti-virus	The best Practice for MySQL is *not* installing anti-virus on a dedicated MySQL environment.

Disks Configuration

This section discusses the general configuration of the disk system.

Applicable Versions of OS	Item	Recommendation
All	Symbolic Links	If the storage type is MyISAM, you symlink the index and data files to another disks to make both the seek and read times better. Symlinks are not supported for InnoDB tables.
All	Log Files	RAID 1+0 or RAID n is the recommended RAID level for Log files in databases.
All	Data Files	RAID1 and RAID 10 are recommended levels for Data files.
All	Configuration Files	Always have a backup for configuration files whether we use RAID or not. RAID 1+0 or RAID 0 are recommended for these files.

MySQL Instance Level Configuration

Note: This section describes the MySQL instance best practices.

Applicable MySQL Versions	Item	Recommendation
All	Latest service pack	Latest service pack and hotfixes.
All	MySQL installed on domain controller	MySQL should never be installed on a domain controller
All	Dedicated machine for MySQL	Best Practices imply that MySQL should be installed on its own dedicated host machine
All	Server allowed protocols (TCP/IP, VIA, NPs, etc.)	Limit the supported protocols.
All	Database files location setting	The database default *data* location should be changed to point to a dedicated disk for data files. The default location for *logs* should be changed to a dedicated location for log files that is optimized for write operations.

MySQL Database Level Configuration

This section describes database level configuration best practices for UIM database.

Applicable MySQL Versions	Item	Recommendation
All	Database files placement	Refer the Disks on SANS **Important!** At a minimum always separate data files from log files on separate physical disks.
All	Database files growth	Use the following general guidelines: • File growth should not be very large because user activity will

		wait for the file operation to complete. • File growth should not be very small to avoid issues with the file becoming full. • Growth should never be in terms of % but in terms of MBs.
All	InnoDB File Per Table	For better performance of UIM disable this feature by setting innodb_file_per_table configuration value to 0.

Database Maintenance and Backup Strategy

This section covers the backup strategy best practices for UIM database.

Applicable MySQL Versions	Item	Recommendation
All	Recovery models	If the binary log mode is not enabled and incremental backup is not enabled on database, then there is no point in time recovery. In case of a failure, such as database corruption, the database will need to be restored from the latest FULL database backup/differential backup. If the binary log mode is enabled and we are following incremental backup, then make sure all files are backed up properly. For better performance, don't backup the databases or binary logs to the same physical disk.
All	Index maintenance tasks	For better performance, it is strongly advised to rebuild indexes based on their size and fragmentation level.

All	Update statistics maintenance task	If automatic statistics gathering option is enabled in the database and there is no massive insertion/update of data, this task is not required (which is true in most cases).
All	Maintenance and backup files cleanup	Don't forget to clean up old backup files according to the database SLA and the amount of free space on the backup drives.
All	Database integrity checks	It is important to run integrity checks on the database. The earlier you find consistency issues, the better.

Additional Resources

https://dev.mysql.com/doc/refman/5.7/en/backup-policy.html

https://dev.mysql.com/doc/refman/5.7/en/disk-issues.html

https://dev.mysql.com/doc/refman/5.7/en/point-in-time-recovery.html

Chapter 6: Performance Analysis

This section provides a checklist for analyzing overall database performance, and can be used for continuous performance analysis and database tuning.

Checklist

Most issues listed in the right column link to a listing of the SQL query that will generate the desired output (valid for MySQL only):

Performance Analysis Area	Issue to identify
Analyze Wait Statistics	Wait times (see example below)
Perform Index Analysis, identifying top issues	Most Costly Unused Indexes Top Costly Missing Indexes Tables without Primary key Tables with no indexes
Identify top SQL queries according to high resource utilization	Top SQL with Highest CPU Top SQL with Highest I/O Top SQL with Highest Duration Top DISK intensive queries

Example Query (Wait Statistics)

This query provides wait statistics that give a good indication of resource bottlenecks from a MySQL perspective:

SELECT * FROM performance_schema.events_waits_summary_global_by_event_name;

Performance schema maintains metadata tables for collecting recent and current wait types. Each event stores min, max, avg and sum values for wait types.

Example Results

EVENT_NAME	COUNT_STAR	SUM_TIMER_WAIT	MIN_TIMER_WAIT	AVG_TIMER_WAIT	MAX_TIMER_WAIT
wait/io/file/csv/update	0	0	0	0	0
wait/io/file/csv/metadata	2	376394823	0	188197202	191813172
wait/io/file/csv/data	4	792715737	0	198178620	425068377
wait/io/file/archive/metadata	0	0	0	0	0
wait/io/file/archive/FRM	0	0	0	0	0
wait/io/file/archive/data	519	27113604387	0	52241758	219338958
idle	125679171	18055865438300774506	0	730771770915	28800506991898509

Chapter 7: Troubleshooting

This section provides best practices to triage and troubleshoot a database.

Best Practice	Comments
Recognize a database where problems are present, learn the signs	One symptom is the data_engine queue backing up
Investigate what other processes are running	Use SHOW PROCESSLIST
Investigate index fragmentation	Use get_index_fragmentation to check index fragmentation
Check disk subsystem(s) and drive failure	Performance impact to RAID Check RAID manager console
Resource constrained?	Task Manger (Windows) Top command in Linux and Solaris

Troubleshooting High CPU Issues

If UIM Server is slow and you identify that the problem is with the CPU (identified using Task Manager in Windows and top or vmstat in Linux), perform the following tasks:

High CPU consumption can occur because of following two reasons

Server-Level CPU Bottleneck

Check the below counters to make sure server CPU is the bottleneck:

Windows

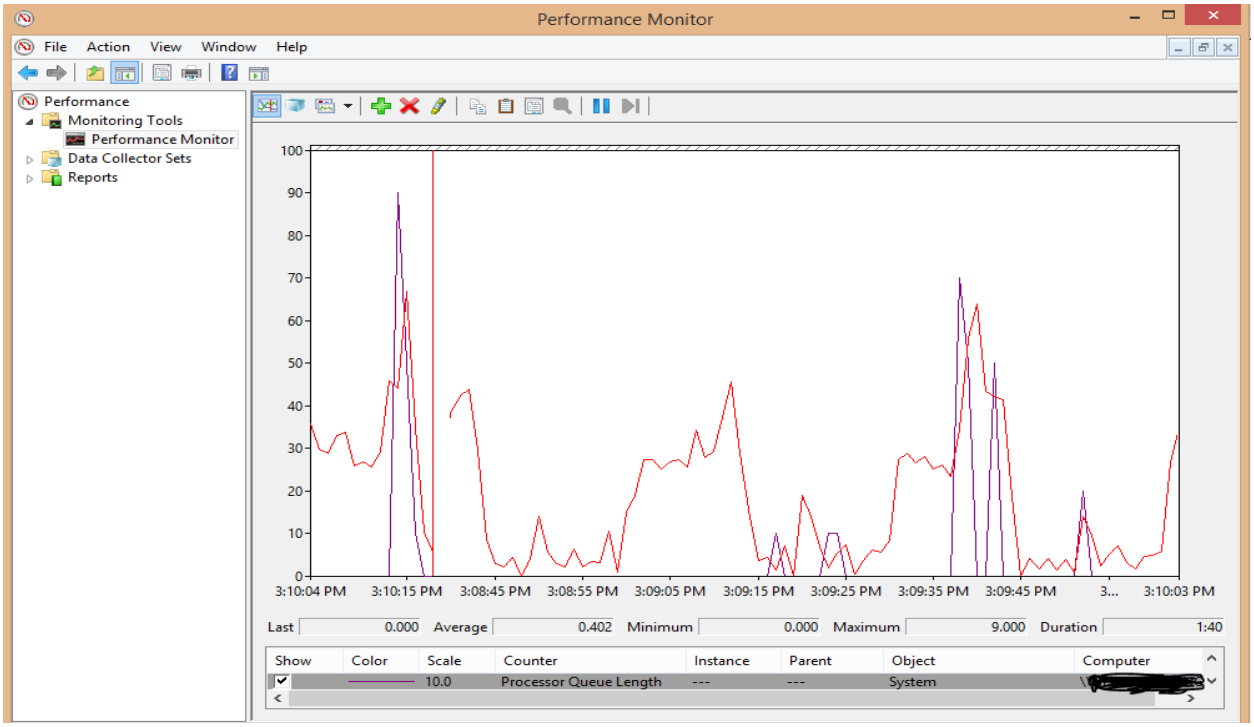
Check the “processor queue length” counter in Windows server. If this number is higher than 0, it implies that there are more requests per core than the system can handle. This can be a cause for significant performance issues in MySQL.

You can launch Performance Monitor using different methods:

Method 1: Start, Control Panel, Administrative Tools, Performance Monitor.

Method 2: Start, search PerfMon.exe

When Performance Monitor is launched, an interface similar to the following is displayed:



Linux

Check run queue (r value) **cpu_count** in Linux. To find whether the MySQL server has the CPU bottleneck, check the run queue value (per vmstat). If this value exceeds the number of processors on the server (cpu_count), the server has the CPU bottleneck. The following example screenshot shows the vmstat output:

```
[root@tuuja01-I7494 ~]# vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0 1596840 168080      0 834300    0    0    12     1    3    1  0  0 100  0
 0
[root@tuuja01-I7494 ~]#
```

If the processor queue length or r value in vmstat command is lower than the CPU cores, then the problem is on the application side. If this value is always more than the CPU cores, increase the CPU cores in the database server.

Application-Level CPU Bottleneck

If the server has enough CPU, consider tuning the database.

Checking database server side bottleneck

- Find the queries that are running when the CPU is at its 100% utilization.
- Update the statistics and check the fragmentation on the tables.
- Check for any big table scans and create appropriate indexes to avoid scans.
- Tune the queries that are taking longer duration.

Troubleshooting Memory Issues

If the UIM Server is slow and you identify that the problem is with the memory (identified using Task Manager in Windows and top or vmstat in Linux), perform the following checks:

Check whether the database server has enough memory to handle the workload:

The buffer pool hit ratio calculates how often a requested data block has been found in the buffer pool without requiring physical disk access.

To check the memory bottleneck on the server, find the buffer pool performance:

Performance = (innodb_buffer_pool_reads / innodb_buffer_pool_read_requests)* 100

innodb_buffer_pool_reads: innodb_buffer_pool_reads parameter indicates the number of requests that cannot be satisfied with InnoDB buffer pool. So server has to read data pages from the disk.

innodb_buffer_pool_read_requests: innodb_buffer_pool_read_requests indicate the number of requests of logical reads from memory. So no need to read from the disk.

The buffer Performance should be lower than 10 percent. If not, then there is an indication that the server has less memory than required. Very minimal percentage of reads indicates that MySQL has enough memory.

Check whether constant swapping is available on the server.

If si and so columns of the vmstat output shows constant swapping, then server is facing memory issue.

Tuning MySQL memory

InnoDB Standard Monitor output provides various metrics pertaining to the operation of the InnoDB buffer pool, under the BUFFER POOL AND MEMORY section. InnoDB Standard Monitor can be accessed using SHOW ENGINE INNODB STATUS. Here is some typical content:

Use the following query to find the InnoDB metrics:

SQL > SHOW ENGINE INNODB STATUS

BUFFER POOL AND MEMORY

Total large memory allocated 2198863872

Dictionary memory allocated 776332

Buffer pool size 131072

Free buffers 124908

Database pages 5720

Old database pages 2071

Modified db pages 910

Pending reads 0

Pending writes: LRU 0, flush list 0, single page 0

Pages made young 4, not young 0

0.10 youngs/s, 0.00 non-youngs/s

Pages read 197, created 5523, written 5060

0.00 reads/s, 190.89 creates/s, 244.94 writes/s

Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000

Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s

LRU len: 5720, unzip_LRU len: 0

I/O sum[0]: cur[0], unzip sum[0]:cur[0]

Recommendations:

- If buffer pool hit rate is lower than 90%, increase innodb_buffer_pool_size.
- If more number of free buffers are present, you may need to decrease innodb_buffer_pool_size.

Swapping

The rule for MySQL memory configuration is you should never want your MySQL to cause the OS to swap. Even less swapping activity causes the MySQL to reduce the performance. Constant swapping of the MySQL can be monitored by using si (Swap in) and so (Swap Out) columns in vmstat command.

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2 71 1026120 162564    208  20960    0    0   125   430    3    6  9  1 87  3  0
 0 99 1102616 142996    260  21280 1236 2838  3531  3102 14723 31199  6  1  0 93  0
 3 67 1173916 143160    636  18944 1801 2718  4986  3257 16363 34749 18  2  0 80  0
 2 72 1177024 165064    464  17512   11  106    48   120 9658 19065 40  0  0 59  0
 1 81 1253468 146084    252  18620 1190 2737  3407  3050 6067 15310 23  3  0 74  0
 1 82 1254304 164636    236  18408   11   30    43    32 1789 5617 25  0  0 74  0
 0 60 1335740 148008    352  21308  660 2841  2331  3073 3200 5852 18  1  0 81  0
 0 69 1333020 195528    532  19668  384   15  1294   130 1561 2920  2  2  0 96  0
```

Here in the above screenshot constant swapping is happening. If you have spikes of more than 1MB/sec, or constant swap activity, you might need to revisit your memory configuration.

To know the amount of memory MySQL is using for workload use the below command:

ps aux | grep mysqld

The 5th column in the output will give you the amount of memory MySQL is using. **Don't allow the mysqld process exceed 90% of the system memory.**

You should not allocate more than 90% of your system memory to MySQL, as you must have some reserved memory for OS.

One you know the amount of memory you want for MySQL process; you need to think about what purpose the memory should be used within MySQL. The primary usage of memory in MySQL is workload related because if you have many active connections and at the same time that run heavy selects uses lot of memory for sorting or temporary tables, you might need more memory.

MySQL Database Optimization Parameters

Update the my.cnf file to optimize MySQL server settings, which is the default configuration file in MySQL.

The following configuration items are the main factors that affect MySQL performance:

- **innodb_buffer_pool_size:** InnoDB depends heavily on buffer pool and this parameter should be set properly. The recommended value for this parameter should be 80%-90% of the available physical memory. if you have RAM bigger than your dataset setting it bit larger should be appropriate with that keep in account of your database growth and re-adjust innodb buffer pool size accordingly.
- **innodb_buffer_pool_instances:** If you have enough Buffer Pool then dividing the buffer pool into multiple instances can improve the concurrency, by reducing the contention. This feature is for the systems with more buffer pool available (in multi gigabytes). In MySQL 5.5 the default value for it was 1 which is changed to 8 as new default value in MySQL 5.6. Minimum innodb_buffer_pool_instances should be lie between 1 (minimum) & 64 (maximum). Enabling innodb_buffer_pool_instances is useful in highly concurrent workload as it may reduce contention.
- **innodb_log_file_size:** For good write performance large enough InnoDB transaction logs are required. But also larger log files mean that recovery process will slower in case of crash. Default value has been changed in MySQL 5.6 to 50 MB from 5 MB (old default), but it's still too small size for many workloads. There should be enough redo log space to handle more than an hour of write activity. The larger the value, the less checkpoint flush activity is required in the buffer pool, saving disk I/O.
- **innodb_log_buffer_size**: InnoDB stores changed data records in log buffers inside memory and saves frequent disk I/O for large transactions as it not need to write the log of changes to disk before transaction commit. 4 MB – 8 MB is good start unless you write a lot of huge blobs.
- **max_connections**: Sometimes applications do not close connections properly which is not recommended as will take MySQL resources. A larger value will give the server more time to

recycle idled connections and it will affect the performance of the server. The recommended maximum value is 5000.

- **query_cache_size:** For every identical statement is received, the server retrieves the results from the query cache rather than parsing and executing the statement again and again. The query cache parameter is shared among sessions. The best option is to disable it by setting query_cache_size = 0 (default on MySQL 5.6) and to use other ways to speed up read queries.
- **innodb_file_per_table**: Unlike the MyISAM storage engine, with its separate tbl_name.MYD and tbl_name.MYI files for indexes and data, InnoDB stores the data and the indexes together in a single.ibd file. The tbl_name.frm file is still created as usual. This value is ON by default from MySQL 5.6. This is usually recommended to set 0 for the UIM as UIM will create lot of tables.

Explain Plan for Queries

Explain plan can be used to analyze the individual queries in the database. Depends upon the tables, columns, indexes and filters MySQL optimizer will select the best plan for the query. Explain plan will analyze the query and gives information like how optimizer joins the table, whether index has been used or not, table joining method etc.

Ways to generate the Explain Plan

Using Query

EXPLAIN statement provides the information about the query. Include EXPLAIN keyword before the query to generate the execution plan.

Ex: EXPLAIN query;

Explain output contains below important columns:

I**d:** This is sequential identifier of the SELECT query.

Table: Table name on which the row of output refers.

Partitions: The partition name from which records would be matched.

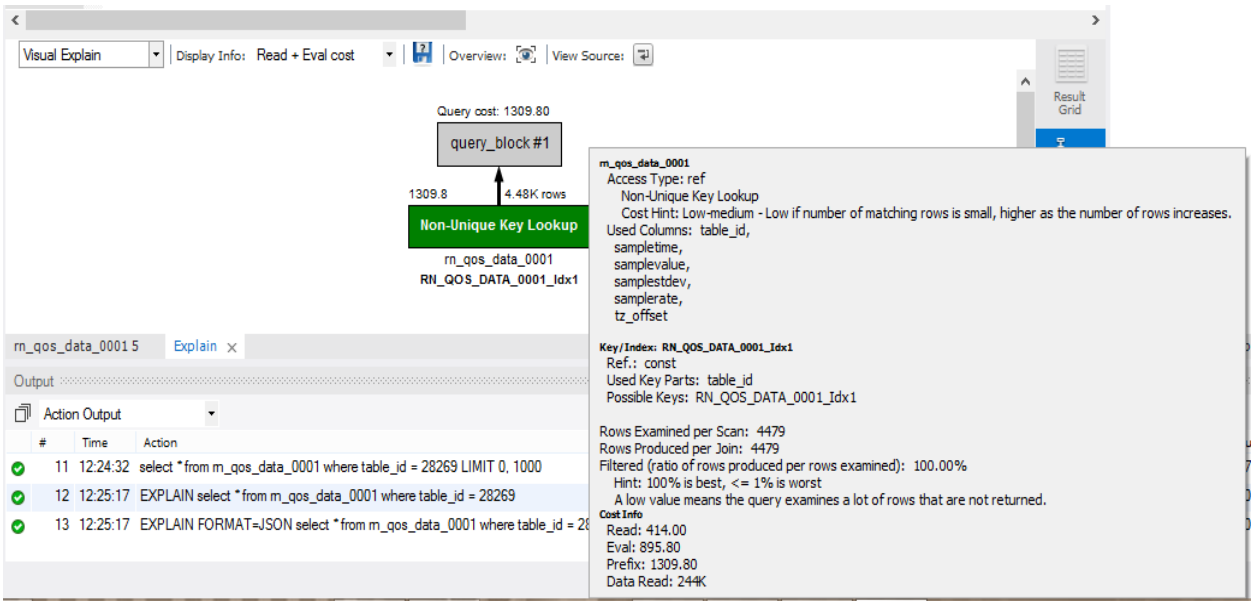
Type: The join type on the tables.

Possible_keys: This column indicates the indexes from which MySQL selects data in a table. If this column is null, then there are no indexes has been used on the table. In this case you may examine the WHERE clause to check any indexes on filtered columns would be suitable. Create the indexes if they are really needed.

Key: Key columns contains the Index that MySQL decided to use.

Using MySQL Workbench

MySQL workbench has the feature to examine the query execution plan in GUI mode. It also allows user to locate and fix the problematic areas in the query. To run the query execution plan, select **Query** menu and select **Explain Current statement**. The explain plan will be generated as shown in below screenshot:



It will give information like Table Access Type, Rows examined per Scan and Possible_Keys etc.

Execution plan will be generated with different nodes in different colors like Red, Orange, Green and Block. Green indicates that there is no bottleneck has been identified whereas red indicates there is a problem with the current operation like full table scan and can be optimized by proper index.

Other MySQL Parameters for UIM

MySQL variables must be set as follows for UIM:

lower_case_table_names=1

local_infile=ON

table_definition_cache=2000

Enable the binary logs only if you use a backup or replication service, which requires the binary log files. To do so, set the following variables:

log_bin

Important! The status of the system variable log_bin specifies whether the binary log is enabled. The --log-bin [=base_name] command-line option enables the binary logging. When you set the --log-bin option, the log_bin system variable is set to ON, not to the base name. The binary log file name is present in the log_bin_basename variable. For more information, see your MySQL documentation.

log_bin_trust_function_creators=ON (if log_bin is enabled)

binlog_format=mixed (if log_bin is enabled)

Use the following procedure to view the setting for each variable.

Follow these steps:

Log in to the MySQL server as the administrator.

For each variable, execute:

show variables like 'variable_name';

If a variable is incorrect or missing, edit the MySQL server configuration file as instructed in your MySQL documentation.

Restart the database if you made any changes.

Max_connections=1000

When using MySQL as the backend database, if you set the data_engine **thread_count_insert** to any value higher than 0, thereby enabling multi-threading, you need to be aware that you could run out of connections. This would manifest itself in the data_engine log that there were "no free connections (er 1040)" because the data_engine could exceed the max number of connections to MySQL.

ERROR 1040 (xxxxx): Too many open connections

This setting should not be changed in a MySQL environment without investigating the MySQL server setting first. There is a hard cap in connections on MySQL that is defined within the MySQL configuration file. The default value for this is 151. If the setting is not high enough, this can cause the data_engine to crash repeatedly once it has uninterrupted access to the DB (e.g., after maintenance is completed.)

To avoid this problem, error and inconsistent connections to the database when inserting data, you need to increase the max connections settings in MySQL.

This error occurs when connection reaches the maximum limit as defined in the configuration file. The variable holding this value is 'max_connections.'

To check the current value of this variable, login as the root user and run the following command:

show global variables like '%max_connections%';

You can login to MySQL using the root user and increase the max_connections variable to a higher value.

SET GLOBAL max_connections = 1000;

This method above does not require a server restart. Please note that after MySQL server restart, the max_connection variable value will again roll back to the previous value. In order to make the max_connections value persistent, modify the value in the configuration file.

Stop the MySQL server:

Service mysql stop

Edit the configuration file my.cnf

vi /etc/my.cnf

Find the variable max_connections under mysqld section.

[mysql]

max connections = 1000

Set the higher value and save the file

Start the server

Service mysqld start

Note: use systemctl manager to stop and start the service if the service command is not working.

Before increasing the max_connections variable value, make sure that the server has adequate memory for new requests and connections. Consult your MySQL DBA to determine if enough resources are available.

MySQL pre-allocates memory for each connection and de-allocates only when the connection is closed. When new connections are querying, the system should have enough resources such as memory, network and computation power to satisfy the user requests.

Also, you should consider increasing the open tables limit in MySQL server to accommodate the additional requests.

MySQL in Large Environments

If you are preparing for a large-scale or major deployment, you can change more database parameters to allow for greater demands of such an environment. We recommend that you begin with the values shown in the following example, and then fine-tune settings depending on your circumstances.

As the MySQL administrator, add these lines to the MySQL server configuration file:

[mysqld]

max_heap_table_size=134217728

query_cache_limit=4194304

query_cache_size=268435456

sort_buffer_size=25165824

join_buffer_size=67108864

max_tmp_tables=64

Additional Resources

https://dev.mysql.com/doc/refman/5.7/en/optimizing-innodb-diskio.html

https://docs.microsoft.com/en-us/azure/virtual-machines/linux/classic/optimize-mysql#mysql-database-optimization

Appendix A: SQL Tools and Scripts

This section lists SQL code to perform the following tasks:

Get Database Size Information

This script provides a listing of data tables, sorted by size.

```
SELECT
    table_schema as 'Database',
    table_name AS 'Table',
    round(((data_length + index_length) / 1024 / 1024), 2) 'Size in MB'
FROM information_schema.TABLES
ORDER BY (data_length + index_length) DESC;
```

Get Database Information (MySQL)

This script provides a summary of the UIM Database deployment size, what is being monitored, probe versions, frequently used probes, database size, data location, and information on specific data tables.

```
-- summary of deployment size
select '1. # qos definitions' as item, COUNT(*) as cnt from S_QOS_DEFINITION
union
```

```
select '2. # qos objects', COUNT(*) from S_QOS_DATA

union

select '3. # robots', COUNT(*) from CM_NIMBUS_ROBOT where is_hub = 0 and alive_time >
DATE_FORMAT(SYSDATE() -(1/24), '%d-%b-yy %H:%i:%s')

union

select '4. # hubs', COUNT(*) from CM_NIMBUS_ROBOT where is_hub = 1 and alive_time >
DATE_FORMAT(SYSDATE() -(1/24), '%d-%b-yy %H:%i:%s')

union

select '5. # computer systems', COUNT(*) from CM_COMPUTER_SYSTEM where alive_time >
DATE_FORMAT(SYSDATE() -(1/24), '%d-%b-yy %H:%i:%s')

-- whats being monitored

select probe, COUNT(distinct qos) as QOS, COUNT(distinct source) as sources, COUNT(distinct target) as
targets

from S_QOS_DATA

group by probe

order by targets desc, QOS desc

-- is everything running the same versions?

select probe_name, pkg_version, COUNT(*) as cnt from CM_NIMBUS_PROBE

where active = 1

and probe_name in ('controller', 'hub')

group by probe_name, pkg_version;

-- most frequently used probes

select probe_name, count(*) as Cnt

from CM_NIMBUS_PROBE

group by probe_name

order by Cnt desc
```

Get Index Fragmentation

This script will be used to provide the index fragmentation

```
SELECT
```

TABLE_SCHEMA,

TABLE_NAME,

CONCAT(ROUND(data_length / (1024 * 1024), 2), 'MB') DATA,

CONCAT(ROUND(data_free / (1024 * 1024), 2), 'MB') FREE ,

(data_free/(data_length+index_length))*100 AS 'Fragmention %'

FROM

information_schema.TABLES

WHERE

TABLE_SCHEMA IN ('ca_uim') AND Data_free > 0;

Recommendation:

Follow this rule to determine whether you need to optimize the table:

1) If the index has “Fragmentation %” more than 30, consider optimizing the table.

Syntax:

Sql > OPTIMIZE TABLE table_name;

Find Missing Nodes in Dynamic Views

-- # of origins not matching

select cs.origin as cmOrigin, d.origin as sqdOrigin , count(*) from CM_COMPUTER_SYSTEM cs inner join CM_NIMBUS_ROBOT r on cs.ip = r.ip and cs.origin = r.origin inner join S_QOS_DATA d on cs.ip = d.host where d.origin <> r.origin group by cs.origin, d.origin order by count(*) desc

-- query to see if origins match among CM_COMPUTER_SYSTEM, CM_NIMBUS_ROBOT, S_QOS_DATA

select cs.origin as cmOrigin, r.origin as robotOrigin, d.origin as sqdOrigin ,

char_length(cs.origin) as cmOriginLen,

char_length(r.origin) as robotOriginLen, char_length(d.origin) as sqdOriginLen,

cs.*,r.*

from CM_COMPUTER_SYSTEM cs

inner join CM_NIMBUS_ROBOT r

on cs.ip = r.ip

and cs.origin = r.origin

```
inner join S_QOS_DATA d
on cs.ip = d.host
where d.origin <> r.origin ;

-- query to see if origins match between CM_COMPUTER_SYSTEM and CM_NIMBUS_ROBOT

select cs.origin as cmOrigin, r.origin as robotOrigin, char_length(r.origin) as robotOriginLen,
char_length(cs.origin) as cmOriginLen ,cs.*,r.* from CM_COMPUTER_SYSTEM cs
inner join CM_NIMBUS_ROBOT r on cs.ip = r.ip where cs.origin <> r.origin;

-- looking for the device from S_QOS_DATA

select * from S_QOS_DATA d
left join CM_CONFIGURATION_ITEM_METRIC m
on d.ci_metric_id = m.ci_metric_id
left join CM_CONFIGURATION_ITEM i
on m.ci_id = i.ci_id
left join CM_DEVICE c
on i.dev_id = c.dev_id
where d.probe = 'cdm' and
d.robot = ''

-- device info

select * from CM_DEVICE d
where d.cs_id = ''
or d.dev_id = ''

-- looking for the device from CM_COMPUTER_SYSTEM

select * From CM_COMPUTER_SYSTEM s
left join CM_GROUP_MEMBER cm
on s.cs_id = cm.cs_id
```

```
left join CM_GROUP cg
on cg.grp_id = cm.grp_id
where s.ip = ''
or s.name = ''
or s.cs_id = ''

-- all left joins to see where things break down

select distinct
c.dev_id,
r.address nimbus_address,
r.ip robotip,
r.domain,
r.hub hubname,
s.name robotname,
cg.name groupname,
s.nimbus_type,
d.source source,
d.origin,
s.os_type os_major,
s.os_name os_minor,
s.os_version,
s.os_description,
d.ci_metric_id,
d.qos,
d.target,
d.r_table,
d.probe,
d.table_id,
d.samplevalue value
```

```
from S_QOS_DATA d
left join CM_CONFIGURATION_ITEM_METRIC m
on m.ci_metric_id=d.ci_metric_id
left join CM_CONFIGURATION_ITEM i
on i.ci_id = m.ci_id
left join CM_DEVICE c
on c.dev_id = i.dev_id
left join CM_COMPUTER_SYSTEM s
on c.cs_id = s.cs_id
left join CM_GROUP_MEMBER cm
on c.cs_id = cm.cs_id
left join CM_GROUP cg
on cg.grp_id = cm.grp_id
left join CM_NIMBUS_ROBOT r
on s.ip = r.ip and
r.origin = d.origin
where
d.probe = 'cdm'
-- and d.robot = ''
-- and d.origin = ''
-- query used by dynamic views to build the tree nodes
select distinct
c.dev_id,
r.address nimbus_address,
r.ip robotip,
r.domain,
r.hub hubname,
s.name robotname,
cg.name groupname,
```

```
s.nimbus_type,
d.source source,
d.origin,
s.os_type os_major,
s.os_name os_minor,
s.os_version,
s.os_description,
d.ci_metric_id,
d.qos,
d.target,
d.r_table,
d.probe,
d.table_id,
d.samplevalue value
from S_QOS_DATA d,
CM_CONFIGURATION_ITEM_METRIC m,
CM_CONFIGURATION_ITEM i,
CM_DEVICE c,
CM_COMPUTER_SYSTEM s,
CM_GROUP_MEMBER cm,
CM_GROUP cg,
CM_NIMBUS_ROBOT r
where
m.ci_metric_id=d.ci_metric_id and
i.ci_id = m.ci_id and
c.dev_id = i.dev_id and
c.cs_id = s.cs_id and
c.cs_id = cm.cs_id and
cg.grp_id = cm.grp_id and
```

```
s.ip = r.ip and
r.origin = d.origin and
d.probe = 'cdm'
UNION
select distinct
c.dev_id,
r.address nimbus_address,
r.ip robotip,
r.domain,
r.hub hubname,
s.name robotname,
cg.name groupname,
s.nimbus_type,
d.source source,
d.origin,
s.os_type os_major,
s.os_name os_minor,
s.os_version,
s.os_description,
d.ci_metric_id,
d.qos,
d.target,
d.r_table,
d.probe,
d.table_id,
d.samplevalue value
from S_QOS_DATA d,
CM_CONFIGURATION_ITEM_METRIC m,
CM_CONFIGURATION_ITEM i,
```

CM_DEVICE c,

CM_COMPUTER_SYSTEM s,

CM_GROUP_MEMBER cm,

CM_GROUP cg,

CM_NIMBUS_ROBOT r

where

m.ci_metric_id=d.ci_metric_id and

i.ci_id = m.ci_id and

c.dev_id = i.dev_id and

c.cs_id = s.cs_id and

c.cs_id = cm.cs_id and

cg.grp_id = cm.grp_id and

r.origin = d.origin and

d.probe = 'RSP'

Find Most Costly Unused Indexes

These are the indexes that are not useful in any query execution and are consuming space on the disk.

Query to find unused indexes:

select * from sys.schema_unused_indexes order by object_name;

Recommendation:

Analyze all the unused indexes. Delete the unused indexes if they are not required.

Find Top Costly Missing Indexes

This script will be used to find missing indexes:

select * from sys.schema_tables_with_full_table_scans

Recommendation: Analyze the above tables to find why there are lot of table scans. If the tables are accessed by using filters from the applications, then it is recommended to create indexes on them.

Find Tables without Primary Key

Use the following query to find tables without primary key.

SELECT

```
   TABLES.table_name
FROM INFORMATION_SCHEMA.TABLES
LEFT JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS c
ON (
      TABLES.TABLE_NAME = c.TABLE_NAME
  AND c.CONSTRAINT_SCHEMA = TABLES.TABLE_SCHEMA
  AND c.constraint_name = 'PRIMARY'
)
WHERE
   TABLES.table_schema = 'ca_uim'
AND c.constraint_name IS NULL;
```

Find Objects with No Indexes

Use the following query to find tables with no indexes.

```
SELECT * FROM INFORMATION_SCHEMA.tables
WHERE table_schema = 'ca_uim'
AND table_name NOT IN
(
SELECT  table_name -- , count(*)
FROM (
SELECT  table_name, index_name
FROM information_schema.statistics
WHERE table_schema = 'ca_uim'
GROUP BY  table_name, index_name) tab_ind_cols
GROUP BY table_name
)
```

Find Top SQL with Highest CPU

Use the following query to find top 20 CPU- and Disk-intensive queries.

```
SELECT SUBSTR(digest_text, 1, 50) AS digest_text_start
```

```
    , count_star
    , TRUNCATE(avg_timer_wait/1000000000000,6), es.*
 FROM performance_schema.events_statements_summary_by_digest es
 ORDER BY avg_timer_wait DESC
LIMIT 20;
```

You can also use the following query to find high cost SQL statements:

```
SELECT * FROM sys.x$statement_analysis
```

Recommendation: Analyze the queries and create all appropriate indexes on the tables involved in the queries listed above. If all the indexes are in place and still query is taking more CPU, then try to increase the number of cores.

Find Top File (table) with Highest I/O

Use the following query to find the top 10 queries information with more number of disk reads.

```
select * from sys.x$io_global_by_file_by_bytes
```

Recommendations:

Analyze the queries and try to decrease the number of disk reads by creating proper indexes and updating statistics on the tables listed by above query.

Find Statistics Update Time

To check for table statistics, use the following query:

```
select * from mysql.innodb_table_stats where database_name='ca_uim';
```

To check for index statistics, use the following query:

```
SELECT * FROM mysql.innodb_index_stats WHERE database_name='ca_uim';
```

Update Statistics Queries

Query to update the statistics on the table is as follows:

```
ANALYZE TABLE table_name;
```

By default, Automatic Statistics Collection is enabled in MySQL. If it is disabled, you can enable it by changing the below configuration parameter:

```
innodb_stats_auto_recalc=ON
```

Recommendation:

Table and index statistics should be up-to-date. If statistics are not up-to-date, then optimizer will select wrong query execution plan.

Find Active sessions

Query to find active sessions information is as follows:

```
SHOW FULL PROCESSLIST
```

Find top DISK- intensive queries

Below query will give the tables with full scans:

```
select * from sys.schema_tables_with_full_table_scans
```

Appendix B: Monitoring the Instance Health with Reports

MySQL Server should be monitored to find the information like CPU, I/O and concurrent user connections etc. MySQL Server can be monitored in following three ways:

- Operating System utilities.
- MySQL commands.
- MySQL Workbench.

Operating System utilities

There are some Operating system utilities to know about MySQL related information like %CPU, %Memory and swapping etc.

Top: Top command will give you the information like %CPU usage, %Memory usage. If MySQL is using High CPU and if it is not coming down while no operation is running, then CPU is the bottleneck. Check the below things if MySQL has high CPU:

- Check all the connections with SHOW FULL PROCESSLIST
- Identify long running queries and tune them if they required.
- Increase the buffer pool size if there is more free memory (check using **free –h** command).

Vmstat: This command will give information regarding swapping. If so and si columns are showing high swapping, then try to increase the MySQL memory.

Steps to change the buffer pool size:

- Open the my.cnf file.
- Edit the innodb_buffer_pool_size parameter.
- Save and exit the file
- Restart the MySQL server.

Monitoring Sessions through queries

Connection manager threads always handle client connection requests on the network interfaces that the server listens to. On all platforms, one manager thread handles TCP/IP connection requests within MySQL. Each client connection will be associated to a thread dedicated to it by connection manager that handles authentication and request processing for that connection.

Query to find Maximum Connections to the Server:

```
SELECT @@max_connections;
```

Query to find the total number of clients that have currently open connections to the server.

```
SHOW GLOBAL STATUS LIKE '%Threads_connected%';
```

Note: Above command provides real-time information on how many clients are currently connected to the server. This can help in traffic analysis or in deciding the best time for a server restart.

Query to find the number of connection attempts (successful or not) to the MySQL server.

SHOW GLOBAL STATUS LIKE 'Connections';

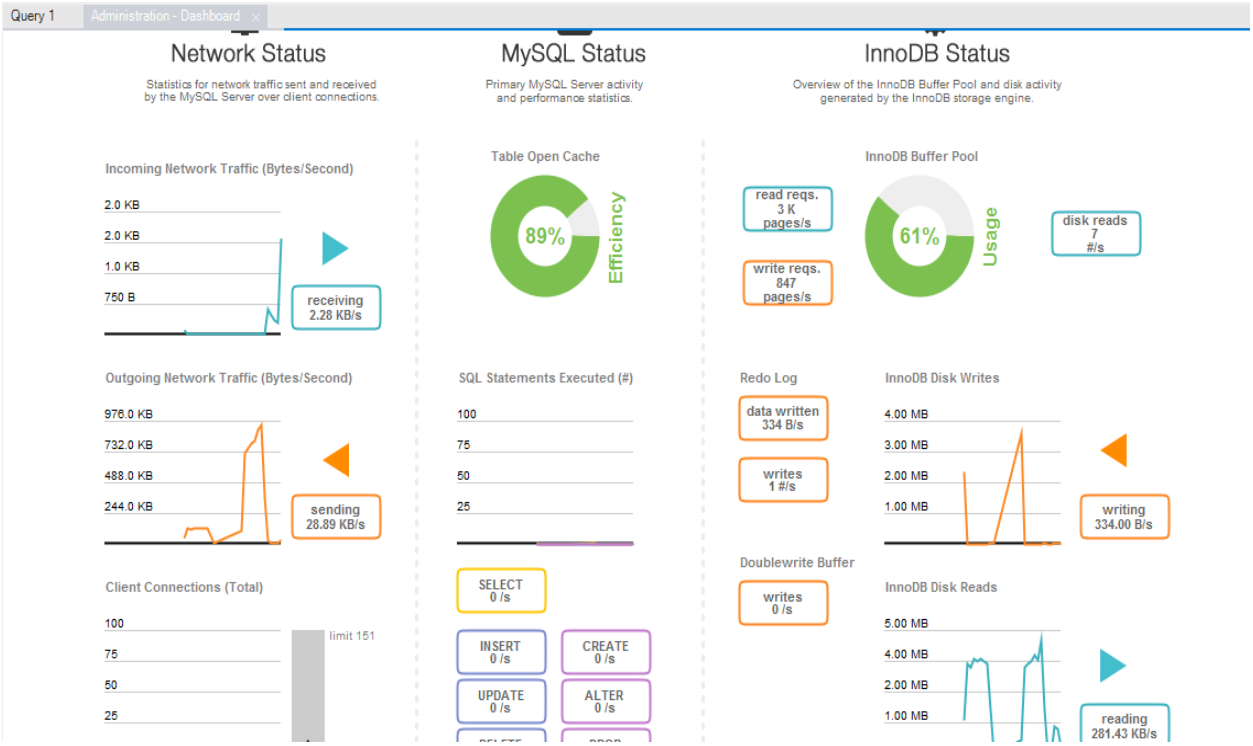
Note: The above command can give you a good idea of how many people and applications are accessing the database. Over time, these numbers reveal busiest times and average usage numbers.

Query to check for unusual numbers of queries running concurrently and struggling to complete in time.

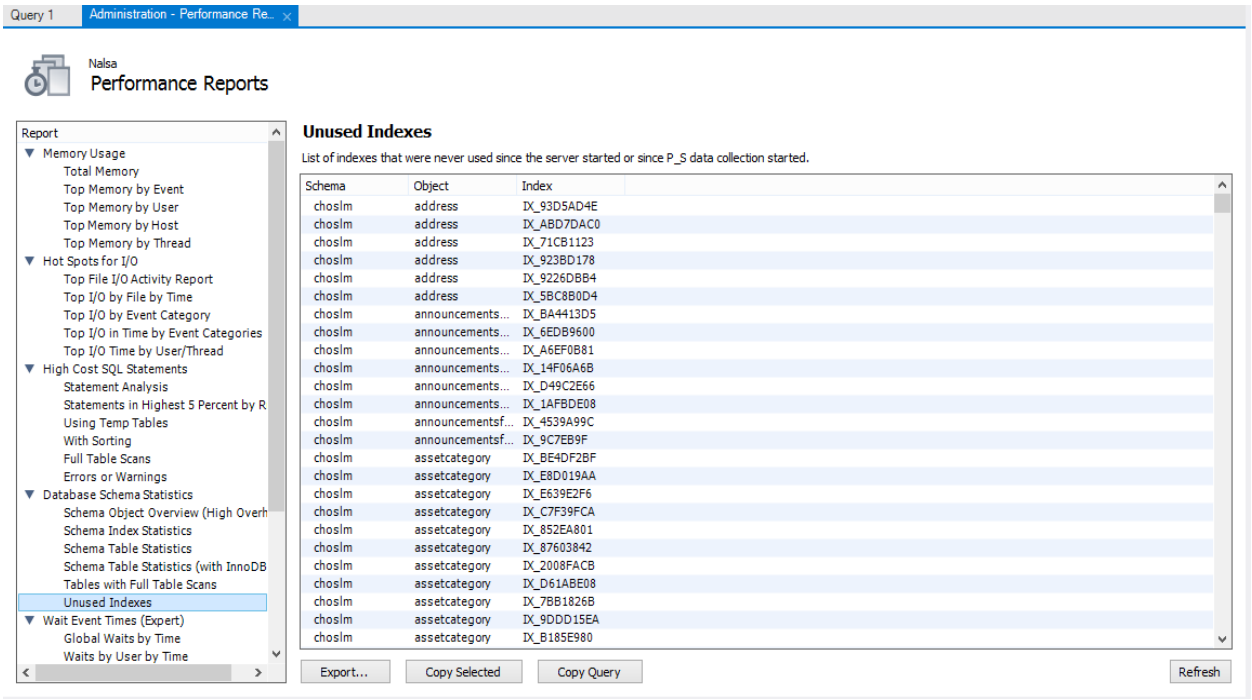
SHOW GLOBAL STATUS LIKE "Threads_running";

Using MySQL Workbench

MySQL Workbench has a dashboard feature, which can be useful to monitor information related to Network, Memory, Connections and Disks etc. Open the workbench and click on **dashboard** under **Performance** section. Below is the sample screenshot of the dashboard:

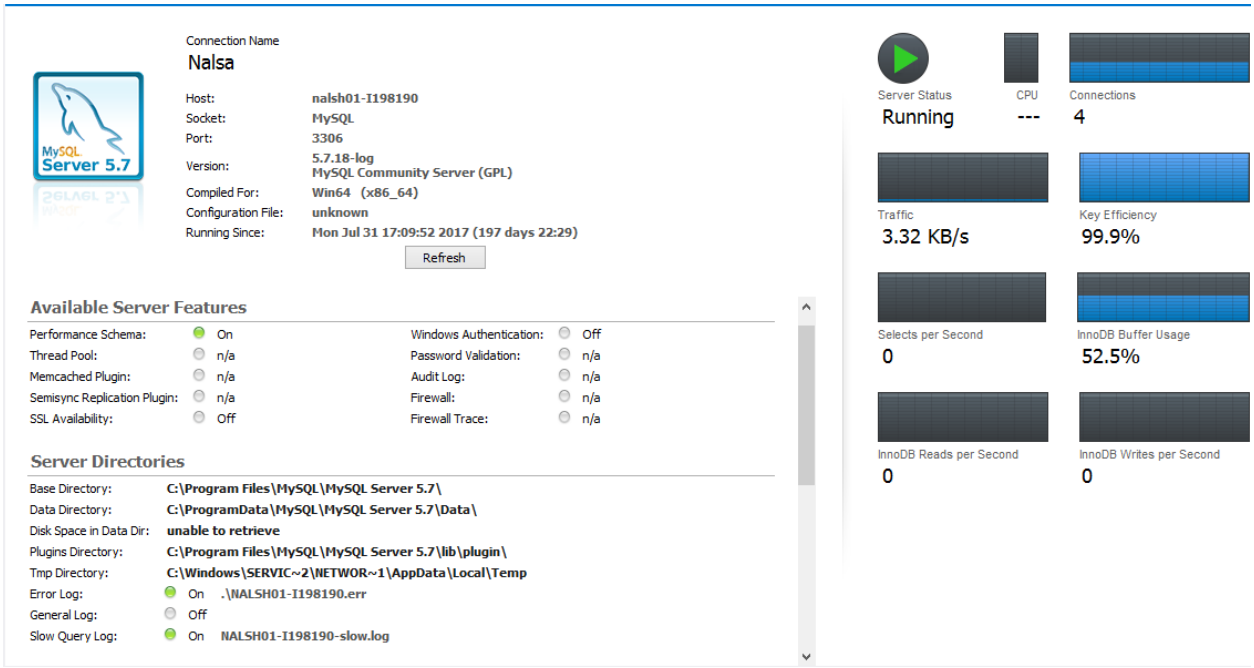


It also allows user to create many Performance Reports like Memory Usage, I/O Reports, High Cost SQL Statements, Schema Statistics, Wait Events, InnoDB statistics and User Resource Statistics. Click on **Performance Reports** under **Performance** section and it will open window as shown in below screenshot:

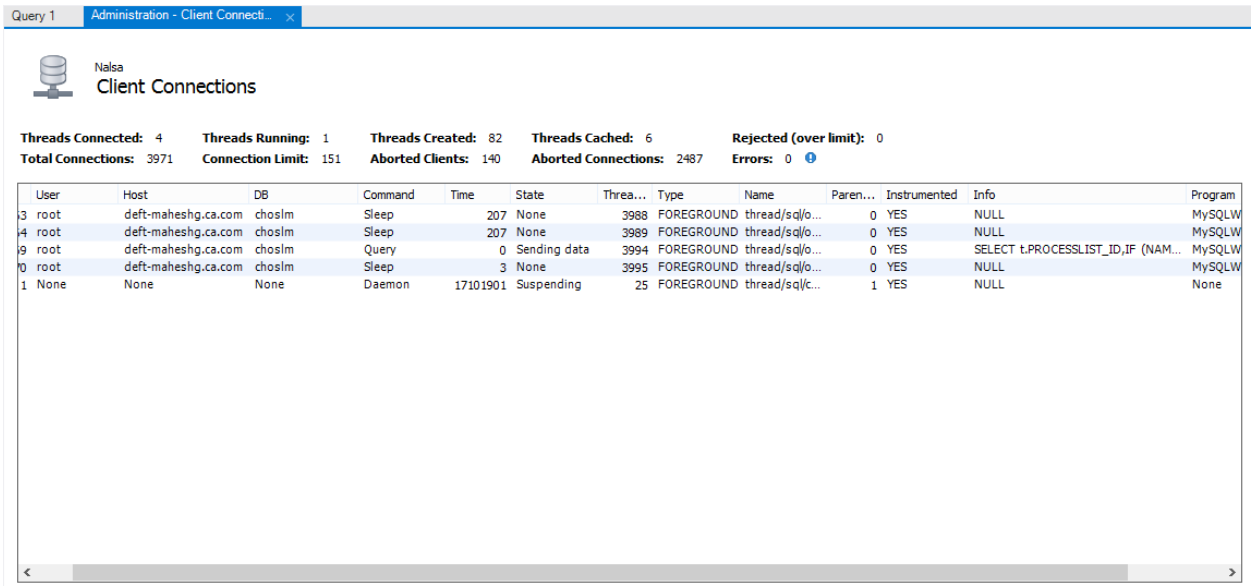


Just click on the report type, it will automatically generate and display report on right pane.

Click on Server status under MANAGEMENT to know the Server current status as shown in below screen:



Click on Client Connections under management to know all the connections and query execution status as shown in below screenshot:



The above output is equivalent to SHOW FULL PROCESSLIST command. Time column in the above output shows the time taken by the query. If any query is taking more time, then analyze the query to improve the performance.