Monitoring PostgreSQL database with Verax NMS
# Table of contents

Abstract ......................................................................................................................................... 3

1. Adding PostgreSQL database to device inventory ............................................................ 4

2. Adding sensors for PostgreSQL database ....................................................................... 7

3. Adding performance counters for PostgreSQL database ................................................. 9

4. Creating custom event processing rules for PostgreSQL database ............................... 11

5. PostgreSQL RDBMS plugin overview ............................................................................... 12

Summary ..................................................................................................................................... 19
Abstract
This publication provides an overview on how to monitor and manage PostgreSQL database using Verax NMS (including the free Express edition available at http://www.veraxsystems.com/en/downloads - please read terms & conditions for limitations of the Express version).

Tools used:
- PostgreSQL Database: http://www.postgresql.org

Agenda:
1. Adding PostgreSQL database to the list of monitored applications within Verax NMS.
2. Configuring availability sensors and performance counters for the database.
3. PostgreSQL database plugin features overview.
4. Setting up alarms and notification policies.
1. **Adding PostgreSQL database to device inventory**

In order to include PostgreSQL database instance to be monitored by Verax NMS, add an application instance to the device actually running this instance.

**Note:** Verax NMS allows for creating multiple instances for applications of the same type on a single device.

In order to add PostgreSQL database server to the device running its instance, perform the following steps:

1. Log in into the Verax NMS and select *Home* from the main menu.
2. Select a device running PostgreSQL database instance from the left-side *aspects* view.

![Aspect hierarchy tree](image)

3. In *Summary tab* select *Manage applications* from the *actions* section.

![Adding a new managed application](image)

4. A pop-up dialog is displayed.

![Applications list dialog](image)
5. Select **Add application** option from the context menu and click **Go**. A dialog window is displayed.

![Add managed application](image)

**Figure 4: Adding PostgreSQL database – parameters dialog**

Verax NMS will ask to enter the following application-specific parameters:

- **Instance name** - Name of the application instance. You can enter any name describing the monitored application instance.
- **Host** - Address of the host running the application instance. In most cases, the host address is an IP address of the device the application instance is assigned to.
- **Port** - Port of application server.
- **User** - Username used to connect to the database.
- **Password** - Password used to connect to the database.
- **Database** – name of the database.

**Note**: application-specific parameters depend on the selected application type.

6. Provide the necessary information and click **Save changes**.

7. The system will ask if you want to add a default set of sensors and counters for PostgreSQL database. Since, in this example sensors and performance will be added manually – click **No**.

The initial set of PostgreSQL database monitors includes:

- Configuration: software version, host platform, status and system variables
- Database instance inventory
- Predefined sensor and counter templates to monitor most important SQL performance characteristics.
8. The newly added database is now visible in the aspect tree within the host’s node in *Managed Applications* category.

![Figure 5: Aspects tree](image1)

![Figure 6: PostgreSQL database properties window](image2)
2. Adding sensors for PostgreSQL database

PostgreSQL database instance can be monitored by:

- Obtaining database information and statistics on demand using the *Show advanced view* action available on **Summary tab**, which shows several views with major information about the database server as well as performance characteristics.
- Checking application or service availability by configuring SQL-based Postgre sensors (or JDBC sensors alternatively).
- Collecting performance data by configuring performance counters using either SQL-based Postgre counters (or JDBC counters alternatively) or SQL-based Postgre counter templates.

**Sensors** are active monitors periodically querying the device services for which they are configured and waiting for their responses. If a query is returned with an expected response, the queried service is considered "available." If a response is not received (timed out), or if the response is not as expected, the queried service is considered "unavailable."

The system includes a number of pre-configured sensors. The following types of sensors for PostgreSQL database application type are available by default:

- **PostgreSQL Sensor** - checks if given SQL query run on specified database returns expected result.
- **JDBC** - checks if a given SQL query run on a specified database returns the expected result.

In order to add a sensor, perform the following steps:

1. Select device from the aspect tree in **Home view** (PostgreSQL database in this case).
2. Select **Monitors tab** and switch to sensor list by clicking **Sensor list** link in the upper-right corner of the tab field. The sensor list is displayed.

3. Select **Add** from the global action menu and click **Go**. The wizard dialog is displayed.

![Figure 7: Adding a sensor](image)
4. Select the sensor you want to add and click **Next**.

![Add new sensor dialog](image)

**Figure 8: Specifying sensor parameters**

5. A dialog shows up with all sensor parameters to be provided. Specify the sensor parameters and click **Finish**.

6. Once the sensors have been added, they are visible on the sensor list (**Monitors tab**).
3. Adding performance counters for PostgreSQL database

Performance counters measure system activity and performance (metrics). The application retrieves their current values in predefined intervals. The aim of probing and collecting data is to analyze and convert the data into a performance graph/chart. The user can define a counter manually or load it from a template.

Performance counter templates are templates with defined probing parameters for specified devices in order to improve and speed up counter creation.

Each Verax NMS counter template is characterized by the following information:

- **Name and description** - unique identifier and optional description,
- **Device type** - type of a device,
- **Protocol type** - protocol used,
- **Probing interval** - pauses between probing.

Counter templates are needed when the counter creation method is set to "from template". Counter templates provide defined probing parameters for specified devices in order to improve and quicken counter creation.

In order to add PostgreSQL database performance counters, perform the following steps:

1. Select device from aspect tree in *Home view*.
2. Select *Monitors tab* and switch to the counter list by clicking *Counter list* link in the upper-right corner of the tab field. A counter list is displayed.

3. Select *Add* from the global action menu and click *Go*. Select the counter you want to create and click *Next*.
4. Once the data has been loaded, the edit window shows up with all counter attributes to be provided.

![Add new counter dialog box](image)

Figure 10: Selecting measure metrics

5. Specify the rest of the counter parameters (alarms, thresholds etc.) and click **Finish**.

6. The new counter has been created and is now visible on the counter list.
4. Creating custom event processing rules for PostgreSQL database

Events are processed by Event Processing Rules running under control of the Event Processing Engine. The Event Processing Engine within the system is able to process events fast without materializing them in database. Verax NMS comes with a set of embedded, flexibly customized processing rules such as: De-duplication, Pairwise matching, Event forwarding, Intermittent failure, Scheduled Maintenance, etc. It also provides users with the ability to implement their own processing rules using JRuby scripting language.

Verax NMS provides complex fault management, such as alarm collection, filtering, blocking, thresholds and correlation (scripted, user-defined rules defining business logic for alarm correlation, cleaning, root-cause, etc.) as well as alarm management actions, e.g. assignment, change of status, clearing, annotation and others. It also enables users to create alarms based on network data etc.

In this example it is shown how to assign basic event processing rules such as:

- Alarm generating
- Event dropping
- Event forwarding
- Severity assigning

To assign an event processing rule, perform the following steps:

1. After selecting the desired host go to Events tab.
2. Select events, choose Assign processing rules and click Go.
3. A dialog window is displayed (see figure below).
4. Select rule category and click Add new rule. A dialog window is displayed (see figure below).

![Figure 11: Creating custom processing rule](image)

The newly created event processing rule is now visible and active (there’s no need to logout).
5. PostgreSQL RDBMS plugin overview

Verax NMS PostgreSQL plugin supports multiple instances of PostgreSQL 8.1 and higher running in 32 and 64 bit environments. The communication between Verax NMS and PostgreSQL server(s) takes place via SQL queries (no management agent is required on PostgreSQL server).

General information view

The view presents general server and database information such as:

- Server information: version, compiler and operating system
- General configuration:
  - Configuration file path
  - TCP listen port for incoming connections
  - Archive mode status
  - Autovacuum mode status
- Memory and disk usage statistics:
  - Total database size on disk
  - Block size
  - Working memory
  - Cache size
  - Shared and temporary buffers
  - Maintenance operations memory

![Figure 12: PostgreSQL RDBMS plugin – general view](image)
Database settings view
The view displays a list of all current server and database configuration parameters.

![Database settings view](image)

Fig. 13: PostgreSQL RDBMS plugin – settings view

Tablespace view
The view displays a list of all database tables including name, size, average row length and expected vs. real pages used. It also provides a graphical presentation of the most fragmented tables which require optimization.

![Tablespace view](image)

Fig. 14: PostgreSQL RDBMS plugin – tables view
Process view

The view displays a list of processes currently accessing the database providing the following information:

- Process id (PID)
- Connecting user name, port, IP address and application name
- Last transaction start time
- Last SQL query start time
- Current SQL query text

![Process view](image)

Vacuum view

PostgreSQL Vacuum is a background daemon for reclaiming unused storage (occupied by deleted records). The Vacuum view provides the following information, essential for optimizing database performance:

- Auto-vacuum enable status and threshold
- Delay time
- Number of processes
- Cost limit (on I/O operations used for vacuum in a single run), cost delay (number of seconds to suspend vacuum for, when cost limit is reached) and memory used by the vacuum daemon

![Vacuum view](image)
Memory view

The view provides comprehensive statistics on database memory including:

- Disk block size
- Effective cache size
- Shared buffer size
- Temporary buffers size
- Working memory size
- Maintenance operations memory size
- Number of checkpoint segments
- WAL (Write-Ahead Logging) update method
- WAL buffers size
- Stack depth per process
- A list of preloaded shared libraries
- Maximum number of connections allowed
- Maximum number of files that can be opened per connecting process

Figure 16: PostgreSQL RDBMS plugin – memory view
Performance view

Provides statistics for identifying database performance bottlenecks including:

- Top processes by run time
- Top objects by size
- Top objects by rows read
- Top objects by physical reads
- Top objects by number of buffer gets

Figure 17: PostgreSQL RDBMS plugin – performance view
Transaction summary

Provides a comprehensive summary of top transactions for each database on the server including:

- Buffer hit rate
- Number of committed transactions
- Number of transactions rolled-backs
- Total number of rows inserted since database creation
- Total number of rows updated since database creation
- Total number of rows deleted since database creation

Figure 18: PostgreSQL RDBMS plugin – transactions view
Diagnostics

The diagnostics report allows troubleshooting database performance. The report presents the following information:

- Tables that are missing primary keys
- Unanalyzed tables
- Vacuumed tables (i.e. ones that are being cleaned up at the moment)
- Tables with large number of sequence scans without indexes, along with problem severity classification (from minor to extreme)

![Figure 19: PostgreSQL RDBMS plugin – diagnostics view](image-url)
Summary
If you performed all actions described in chapters 1-5 you are now able to monitor PostgreSQL database.