Database Visibility

Overview of Database Visibility
Database Visibility Supported Environments
Database Visibility System Requirements
Administer the Database Agent
  Prepare to Install the Database Agent
  Install the Database Agent
  Verify the Database Agent Installation
  Upgrade the Database Agent
  Uninstall the Database Agent
  Configure the Database Agent
    Database Agent Configuration Properties
    Enable SSL and SSH for Database Agent Communications
  Install the Database Agent as a Windows Service
  Enable Snapshot Correlation for Oracle
  Increasing JVM Memory
  Database Agent Logging
  Configure Controller Settings for Database Agents
  Start and Stop the Database Agent
  Start the Database Agent Automatically on Linux
  Start the Database Agent Automatically on Windows
Add Database Licenses
Add Database Collectors
  Configure IBM DB2 Collectors
  Configure Microsoft SQL Server Collectors
    Microsoft SQL Server on AWS RDS Permissions
  Configure Microsoft Azure Collectors
  Configure MongoDB Collectors
  Configure MySQL Collectors
  Troubleshooting MySQL Permissions
  Configure Oracle Collectors
    Configure Oracle RAC
  Configure PostgreSQL Collectors
  Configure Sybase Collectors
  Configure Sybase IQ Collectors
  Configure Couchbase Collectors
  Configure the Database Agent to Monitor Server Hardware
  Required Monitored Host Permissions
  Configure WMI Permissions and Security
Monitor Databases and Database Servers
  Access Database Visibility from Application Monitoring Views
  Discover Normal Database and Server Activity
  Monitor Database Performance
    View Overall Database and Server Performance
    Database Dashboard
    Database Activity Window
    Database Topology Window
    Database Live View Window
    Database Queries Window
    Database Query Details Window
    Database Query Execution Plan Window
  Database Clients Window
  Database Sessions Window
  Database Blocking Sessions Window
  Database Schemas and Databases Windows
  Database Modules Window
  Database Programs Window
  Database Users Window
  Database Buckets Window
  Database Business Transactions Window
  Database Applications Window
  Database Object Browser Window
  Database Custom Metrics Window
  Monitor Database Server Hardware
  Database Health Rules and Alerts
  Database Monitoring Metrics
  Database Agent Events Reference
Wait State Filtering
Database Visibility

AppDynamics Database Visibility provides end-to-end visibility on the performance of your database. Database Visibility helps you troubleshoot problems such as slow response times and excessive load. Database Visibility provides metrics on database activity, such as:

- SQL statements or stored procedures that are consuming most of the system resources
- Statistics on procedures, SQL statements, and SQL query plans
- Time spent on fetching, sorting, or waiting on a lock
- Activity from the previous day, week, or month

__Installation and Administration__

- Install the Database Agent
- Start and Stop the Database Agent
- Upgrade the Database Agent

__Configuration__

- Configure the Database Agent
- Add Database Collectors

__Using Database Visibility__

- Overview of Database Visibility
- Monitor Database Performance
- Monitor Database Server Hardware

__Reference__

- Database Agent Configuration Properties
- Database Agent Events Reference
- Database Monitoring Metrics
Overview of Database Visibility

Database Visibility provides metrics on the performance of your database and helps troubleshoot performance-related issues.

Components of Database Visibility

AppDynamics Database Visibility consists of four main components:

1. Database Agent
2. Collector
3. Controller
4. Events Service (on-premises only)

Database Agent

The AppDynamics Database Agent is a standalone Java program that collects performance metrics about your database instances and database servers.

You can deploy the Database Agent on any machine running Java 1.8 or higher. The machine must have network access to the AppDynamics Controller and the database instance that you want to be monitored.

A database agent running on a typical machine with 16 GB of memory can monitor about 25 databases. On larger machines, a database agent can monitor up to 200 databases. To monitor more than 100 databases, increase the initially allocated heap size for the Database Agent JVM. See Database Visibility Supported Environments

Collector

The Database Agent Collector is the process that runs within the Database Agent to collect performance metrics about your database instances and database servers. One Collector collects metrics for one database instance. Multiple collectors can run in one Database Agent.

AppDynamics Database Visibility detects when a database back end has matching credentials with a database server being monitored by a Collector. Database Visibility automatically associates the back end with the collector, so you can view its performance with the Application Flow Map, Tier Flow Map, or Node Flow Map.

Controller

The Controller is the central interface where you can see all your database instance and database server performance metrics.

The following types of information are sent to the Controller:

- Database-level metrics, such as the number of queries processed and other database statistics
- Names and attributes of all sessions, clients, queries, and other objects on the monitored system
**Events Service**

Event service stores high volumes of metric data. If you are using the on-premises version of Database Visibility, you must install the Events Service.

The following types of information are sent to the Events Service:

- Time that each query spends at each wait state
- Individual query statistics for databases that support it
- Information about individual execution plans in databases that support it
Database Visibility Supported Environments

This topic discusses application environments and versions that are supported by the Database Visibility Agent.

Database Visibility Support

Once Database Visibility is available, you can create collectors that run on the Database Agent to monitor any of the following systems:

<table>
<thead>
<tr>
<th>Database</th>
<th>Supported through Amazon RDS</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB, MongoDB cluster</td>
<td></td>
<td>2.6 and higher</td>
</tr>
<tr>
<td>MySQL</td>
<td>Yes</td>
<td>All versions including Percona, MariaDB, and Aurora</td>
</tr>
<tr>
<td>Microsoft SQL Server on Linux</td>
<td></td>
<td>SQL Server on Linux is currently available as a public preview and is not recommended for production use.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Database Visibility works well with this preview release, but monitoring results may vary until a stable version of SQL Server on Linux is available.</td>
</tr>
<tr>
<td>Oracle, Oracle RAC</td>
<td>Yes</td>
<td>10g (10.2 and later), 11g, and 12c</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>Yes</td>
<td>All versions</td>
</tr>
<tr>
<td>IBM DB2 LUW</td>
<td></td>
<td>9.x, 10.x, 11.x</td>
</tr>
<tr>
<td>Sybase ASE</td>
<td></td>
<td>15+</td>
</tr>
<tr>
<td>Sybase IQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Couchbase</td>
<td></td>
<td>4.5 and higher</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>32-bit (also works on 64-bit systems)</td>
</tr>
<tr>
<td>Linux</td>
<td>32-bit and 64-bit</td>
</tr>
<tr>
<td>Solaris</td>
<td>All versions</td>
</tr>
<tr>
<td>AIX</td>
<td></td>
</tr>
<tr>
<td>Amazon RDS</td>
<td></td>
</tr>
</tbody>
</table>

To avoid metric value errors, use a 64-bit JRE with the 64-bit operating system.
**Recommended**

If you use a third-party application along with Database Visibility to monitor Sybase, the data displayed in the controller may not display correctly. For accurate metrics, do not use both a third-party application and Database Visibility to monitor Sybase.
Database Visibility System Requirements

On this page:
- Hardware Requirements
- AppDynamics Controller Sizing Requirements
- Software Requirements
- Network Requirements

This page lists hardware and software requirements using Database Visibility.

Hardware Requirements

Hardware requirements vary depending on database activity. If your database activity increases, you may need to adjust your hardware configuration.

The machine running the Database Agent should meet the following hardware requirements:

- 1 GB of heap space and an additional 512 MB of heap space for each monitored database instance. For less busy databases, you may reduce the heap space to 256 MB per monitored database instance.
- 2 GHz or higher CPU.

<table>
<thead>
<tr>
<th>Number of Database Instances Monitored</th>
<th>Heap Space Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>(5 x 512 MB) + 1024 MB = 3,584 MB</td>
</tr>
<tr>
<td>20</td>
<td>(20 x 512 MB) + 1024 MB = 11,264 MB</td>
</tr>
<tr>
<td>100</td>
<td>(100 x 512 MB) + 1024 MB = 52,224 MB</td>
</tr>
</tbody>
</table>

AppDynamics Controller Sizing Requirements

The controller database should meet the following hardware requirements:

- 500 MB of disk space per collector per day
- 500 MB of disk space for the Events Service per day. By default, the Events Service retains data for 10 days.

See Controller System Requirements

Note

The Database Agent requires the Events Service. Start Event Service before you start the Database Agent.

Software Requirements

- The Database Agent runs on a Java Virtual Machine. You must have Java 1.8 or later.
- The operating systems Linux and Windows are supported.
Network Requirements

- The machine on which the database is running or the machine you want to monitor must be accessible from the machine where the Database Agent is installed and running. This machine must have a network connection, internet, or intranet.
- If your databases are behind a firewall, you must configure the firewall to permit the machine running the Database Agent program access to the databases. The database listener port (and optionally the SSH or WMI port) must be open.
- The network bandwidth used between the agent and the controller is approximately 300 KB per minute per collector for a large database with 200 clients using 50 schemas, processing about 10,000 queries a minute. The actual numbers depend on the type of database server, the number of individual schemas on the server, and the number of unique queries executed daily, and therefore varies.
Administer the Database Agent

The Database Agent is a standalone Java program that collects performance metrics about your database instances and database servers. You can view these performance metrics in the Metric Browser of the AppDynamics Controller UI.

To start monitoring your database, you must first install the Database Agent. You may want to install multiple agents for the following reasons:

- You have databases on multiple networks that are not accessible by one machine.
- You have multiple SQL Server instances that each require different credentials, and you want to connect to them through Windows Authentication.
- You want one or more agents to back up a primary agent. Backup agents ensure that your database instances are continually monitored during agent failure or machine downtime.
Prepare to Install the Database Agent

On this page:

- Prerequisites for Installing the Agent
- Controller Communication Information

Related pages:

- Database Visibility
- Database Agent Configuration Properties
- Add Database Collectors
- Start the Database Agent Automatically on Linux
- Start the Database Agent Automatically on Windows

Prerequisites for Installing the Agent

Ensure that your environment meets the following requirements

- Java 1.8 or later
- The machine that you install the agent on has network access to the databases that you want to monitor.

Controller Communication Information

If you are installing using the Database Agent downloaded from the Getting Started Wizard, the Controller communications information is already configured for you in the agent controller-info.xml file.

To complete the Getting Started Wizard - Databases, you need the following information:

- Controller Host Name
- Controller Port Number

If you are installing using the agent installation zip file downloaded from the AppDynamics portal, you need the following information:

- Controller Host Name
- Controller Port Number
- Account Access Key

See Agent-to-Controller Connections.
Install the Database Agent

On this page:
- Install the Agent Software
- License the Agent (on premise only)
- Configure the Agent
- Start the Controller Events Service (on-prem only)
- Launch the Database Agent using System Properties

Related pages:
- Database Visibility
- Database Agent Configuration Properties
- Add Database Collectors
- Enable SSL and SSH for Database Agent Communications
- Start the Database Agent Automatically on Linux
- Start the Database Agent Automatically on Windows

Follow the procedure below to manually install the Database Agent. You can alternatively follow the steps in the Download & Install Wizard, which you can access from the home page in the Controller UI. The wizard simplifies the install process.

To avoid permission issues, you should install the Database Agent as the same user who will run the agent, or as an administrator on the host machine. To run the Database Agent, you must have write privileges to the logging output directory and to the conf directory, which are located in the agent installation directory.

Install the Agent Software

1. Download the version of the installation package that is appropriate for your OS environment from the AppDynamics Download Center (http://download.appdynamics.com).
2. Extract the zip file to the destination directory. Do not use spaces in the agent installation destination directory.
3. Log on as an administrator to the machine where you will be installing the Database Agent.

Windows

Double-click the dbagent-x.x.x.zip file and extract the files to <db_agent_home>. If necessary, you can unblock the zip file before you extract it as follows: Right-click on the zip file, select Properties and choose unblock.

<table>
<thead>
<tr>
<th>There are two versions of the database agent: one that is 32-bit and one that is 64-bit (new in 4.5.2). Choose a version based on your OS requirements.</th>
</tr>
</thead>
</table>

Linux

Enter the following on the command line:

```
unzip dbagent-x.x.x.zip -d <db_agent_home>
```

Running Multiple Database Agents

You can have multiple Database Agents concurrently running the agent jar in the <db_agent_home> directory on the same machine. Some system properties may be required depending on how you'll be using the agents.

License the Agent (on premise only)
Obtain a license.lic file with Database Monitoring licensing from your sales or support representative and put the license file in the directory where you installed the Controller. After placing the license in the directory, the Controller may take a minute or two to detect the new license. Restarting the Controller forces it to detect new licenses.

**Configure the Agent**

You can **configure properties** for the Controller host name, port number, and account access key using either the `<db_agent_home>/conf/controller-info.xml` file or by adding system properties to the Database Agent startup script.

**Controller Host Name**

**Configure using controller-info.xml:** `<controller_host>`

**Configure using System Properties:** `-Dappdynamics.controller.hostName`

*Required:* Yes

*Default:* None

**Controller Port**

**Configure using controller-info.xml:** `<controller_port>`

**Configure using System Properties:** `-Dappdynamics.controller.port`

*Required:* Yes

*Default:* For On-premise Controller installations: Port 8090 for HTTP and 8181 for HTTPS communication.

For SaaS Controller service: Port 80 for HTTP and port 443 for HTTPS communication.

**Account Access Key**

**Configure using controller-info.xml:** `<account_access_key>`

**Configure using System Properties:** `-Dappdynamics.agent.accountAccessKey`

*Required:* Yes

*Default:* None

Optional Configurations:

- Configure the agent to use SSL, see [Enable SSL for Communicating with the Controller](#).
- Configure the agent to use Proxy Settings, see [Proxy Settings for the Controller](#).
- Configure the agent to run automatically when the Machine starts on **Linux** or **Windows**.
- Configure the agent to uniquely identify itself to the Controller, such as when you require multiple agents. See [Multiple Agent Environment Properties](#).
- Configure the agent to act as a backup to another Database Agent. See [Multiple Agent Environment Properties](#).
- Configure the logging level of the Database Agent running on the agent JVM. The attached log file shows an example of the agent log file when the agent logging is set to INFO level. This is the default. The log files are in `<db_agent_home>/logs`.

**Start the Controller Events Service (on-prem only)**

**Linux**

`bin/platform-admin.sh start-events-service`

**Windows**

`bin\platform-admin.exe start-events-service`

**Launch the Database Agent using System Properties**

The following assumes that all the necessary parameters have been specified in the controller-info.xml.

**Method 1: Launch Agent with Start Script**
You can use a script to start the agent. This is the recommended method.

**Linux**

```bash
./start-dbagent -Xms<min_heap_size> -Xmx<max_heap_size> &
```

**Windows**

```bash
start-dbagent.bat -Xms<min_heap_size> -Xmx<max_heap_size>
```

Alternatively, you can launch the agent as a Windows Service (new in 4.5.4). For more information, see [Install the Database Agent as a Windows Service](#).

**Method 2: Launch Agent with Java Command**

Alternatively, you can launch the agent with the following Java commands.

**Linux**

```bash
java -XX:+HeapDumpOnOutOfMemoryError -XX:OnOutOfMemoryError="kill -9 %p" -jar db-agent.jar
```

**Windows**

```bash
C:\java -Djava.library.path="<db_agent_home>\auth\x64" -Ddbagent.name="Scarborough Network Database Agent" -jar <db_agent_home>\db-agent.jar
```

**System Properties**

**Database Agent Name**

- **Configure using System Properties:** `-Ddbagent.name=<db_agent_name>`
  - **Type:** ASCII string, including spaces. If `<db_agent_name>` contains spaces, you must enclose the entire name in double quotes (" ").
  - **Required:** Yes, when you have multiple agents reporting to the same controller.
  - **Default:** Default Database Agent

**Java Library Path**

- **Configure using System Properties:** `-Djava.library.path=<db_agent_home>\auth\x64` (for 64-bit systems) or `<db_agent_home>\auth\x86` (for 32-bit systems)
  - **Type:** ASCII string, including spaces. If `<db_agent_home>` contains spaces, you must enclose the entire name in double quotes (" ").
  - **Required:** Yes, for Windows only
  - **Specify:**
    - For 64-bit systems: `<db_agent_home>\auth\x64`
    - For 32-bit systems: `<db_agent_home>\auth\x86`

**Increase the JVM Memory**

To monitor multiple databases, you may need to increase the JVM memory allocation size. Increased activity on the databases you are monitoring results in increased memory usage.

Use one of the following commands to start the agent, and to initially allocate 1536 MB to the agent instead of the default of 64 MB.
For Windows 64-bit

C:\java -Xmx1536m -Djava.library.path="<db_agent_home>\auth\x64" -jar <db_agent_home>\db-agent.jar

For Linux

% java -Xmx1536m -jar <db_agent_home>/db-agent.jar
Verify the Database Agent Installation

On this page:
- Check the Agent Logs
- Verify that the Agent is Reporting to the Controller
- Verify that the Agent is Running

Related pages:
- Database Agent Configuration Properties
- Agent and Controller Compatibility

Check the Agent Logs

After a successful install, your agent logs, located at <db_agent_home>/logs, should contain the following message:

```
Started AppDynamics Database Agent Successfully
```

If the agent log file is not present, the Database Agent may not be accessing the Database Agent command properties. To troubleshoot, check the application server log file where STDOUT is logged. It will have the fallback log messages, useful for troubleshooting the agent.

Verify that the Agent is Reporting to the Controller

1. Click Settings > AppDynamics Agents.
2. Click the Database Agents tab.

Here, you should see a listing for each Database Agent reporting to the Controller. An agent can have one of the following statuses:

- Active: The agent is running
- Passive: The agent is used as a backup for active agents

If you don't see the Database Agent, check your controller-info.xml properties to ensure they have specified the correct host properties.

Verify that the Agent is Running

Use the following command to verify that the agent process is running:

**Linux:**

```
ps -ef | grep db-agent
```

**Windows:**

1. Open a command line console.
2. Start the Task Manager and click the Processes tab.
3. The agent process should be running. If it is not running, stop and restart the agent.
Upgrade the Database Agent

On this page:

- Important Upgrade Notes
- Stop the Agent
- Back up the Existing Agent Directory
- Install the Agent
- Copy the controller-info.xml File
- Start the New Agent
- Verify the Database Agent Installation

Related pages:

- Database Agent Configuration Properties
- Agent and Controller Compatibility

Important Upgrade Notes

- If you are upgrading the Controller and agents, first upgrade the Controller and then upgrade the Database Agents.
- Stop the Database Agent process before you install the new agent. All Database Agents that are running from the same install location need to be shut down when updating that install location.
- Download the installation package that is appropriate for your OS environment from the AppDynamics Download Center (https://appdynamics.com/download).
- Back up the `<db_agent_home>` directory so you can revert to the previous installation if required. To maintain the same configuration information, you will also need the `<db_agent_home>/conf/controller-info.xml` file.
- If both the old and new agents are 4.2 or newer, you can keep the old agent running while installing the new agent, allowing you to upgrade the agent with no downtime. Once you see the new agent showing up as “ACTIVE” in the agents page in the Controller, stop the old agent.

Stop the Agent

Stop the agent as described for your specific installation in Start and Stop the Database Agent.

Back up the Existing Agent Directory

Make a copy of the existing agent directory, `<db_agent_home>`. Backing up allows you to revert to the previous agent installation if you need to. You can also copy over the controller-info.xml configuration file to the new installation to ensure the agent configuration is maintained.

Install the Agent

Install the Database Agent as described for your specific installation in Administer the Database Agent.

Copy the controller-info.xml File

To ensure the agent configuration is maintained, copy the `<backup_db_agent_home>/conf/controller-info.xml` file to the new installation directory, `<db_agent_home>/conf`.

Start the New Agent

See Start and Stop the Database Agent.

Verify the Database Agent Installation
See Verify the Database Agent Installation.
Uninstall the Database Agent

To uninstall the database agent:

1. Shut down the JVM that the database agent runs on.
2. Delete the installation directory.

If you have the agent installed the agent as a service, you must also stop the Database Agent service.

To prevent a Database Agent from connecting to the Controller, ensure that it doesn't start up. This frees the license associated with the agent in the Controller and makes it available for use by another Database Agent.
Configure the Database Agent

Once you have installed and launched the Database Agent, you can further customize the agent.

- Database Agent Configuration Properties
- Enable SSL and SSH for Database Agent Communications
- Install the Database Agent as a Windows Service
- Enable Snapshot Correlation for Oracle
- Increasing JVM Memory
- Database Agent Logging
Database Agent Configuration Properties

On this page:
- Where to Configure Database Agent Properties
- Example Database Agent controller-info.xml File
- Example Startup Configuration Using System Properties
- Database Agent Properties
  - Required System Properties
  - Optional System Properties
  - Agent-Controller Communication Properties
  - Multiple Agent Environment Properties
  - Multi-Tenant Mode Properties
  - Proxy Properties for the Controller
  - Other Properties

Related pages:
- Administer the Database Agent

Where to Configure Database Agent Properties

You can configure agent properties in the following locations:

- The controller-info.xml file located in the `<db_agent_home>/conf` directory
- The system properties (-D options) section in the JVM start-up command. The system properties override the settings in the controller-info.xml file. System properties are case-sensitive.

```
java -XX:+HeapDumpOnOutOfMemoryError -XX:OnOutOfMemoryError="kill -9 %p" -jar db-agent.jar
```

Alternatively, you can use a script to start the agent.

<table>
<thead>
<tr>
<th>Windows</th>
<th>start-dbagent.bat -Xms&lt;min_heap_size&gt; -Xmx&lt;max_heap_size&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>./start-dbagent -Xms&lt;min_heap_size&gt; -Xmx&lt;max_heap_size&gt; &amp;</td>
</tr>
</tbody>
</table>

Example Database Agent controller-info.xml File
<?xml version="1.0" encoding="UTF-8"?>
<controller-info>
  <controller-host>192.10.10.10</controller-host>
  <controller-port>8090</controller-port>
  <account-access-key>165e65645-95c1-40e3-9576-6a1424de9625</account-access-key>
<!-- The following attribute enables or disables SSL communications between the agent and the Controller.-->
  <controller-ssl-enabled>false</controller-ssl-enabled>
<!-- The following account-related parameters are necessary only for SaaS installations-->
  <!--account-name--> </account-name-->
</controller-info>

Example Startup Configuration Using System Properties

A bash example. Note that the system properties are case-sensitive.

-Dappdynamics.controller.hostName=192.168.1.20
-Dappdynamics.controller.port=8090

Database Agent Properties

This section describes the Database Agent configuration properties, including their controller-info-xml elements and their system property options.

Required System Properties

Path to the Agent jar File

**Description:** Provides the absolute path to the jar file.
**System Property:** -jar
**Value:** <db_agent_home>db-agent-jar
**Type:** ASCII string, including spaces. If <db_agent_name> contains spaces, you must enclose the entire name in double quotes (" ").
**Required:** Yes
**Example:** -jar="D:\AppDynamics\Database Agent\db-agent.jar"

Path to the Java Library
**Description**: Provides the absolute path to `sqljdbc_auth.dll`.

**System Property**: `-Djava.library.path`

**Value**:
- For 64-bit systems: `<db_agent_home>`\auth\x64
- For 32-bit systems: `<db_agent_home>`\auth\x32

**Type**: ASCII string, including spaces. If `<db_agent_home>` contains spaces, you must enclose the entire name in double quotes (" ").

**Required**: Recommended. Required for SQL Server Windows Authentication on Windows 64-bit systems.

**Example**: `-Djava.library.path="D:\AppDynamics\Database Agent\auth\x64"

### Optional System Properties

**Disable Retry on Auth Failure**

**Description**: Stops retrying to establish JDBC connection when JDBC authentication fails.

**System Property**: `-Dretry.on.auth.failure=false`

**Default**: The system property is set to true.

**Type**: String

**Supported Locales for Parsing Numbers**

**Description**: Specifies which locale to use for parsing numbers. See [Java 7 Supported Locales](#) or [Java 8 Supported Locales](#) for valid values. Use the values for the Java version that you are running in your environment.

**System Property**: `-Ddbagent.language`

**Default**: The system property is set to en-US or en_US.

**Type**: String

### Agent-Controller Communication Properties

**Controller Host Property**

**Description**: This is the host name or the IP address of the AppDynamics Controller, e.g. 192.168.1.22 or myhost or myhost.abc.com. This is the same host that you use to access the AppDynamics Controller UI.

**Element in controller-info.xml**: `<controller-host>`

**System Property**: `-Dappdynamics.controller.hostName`

**Type**: String

**Default**: None

**Required**: Yes

**Controller Port Property**

**Description**: This is the HTTP(S) port of the AppDynamics Controller. This is the same port that you use to access the AppDynamics browser-based user interface. If the Controller SSL Enabled property is set to true, specify the HTTPS port of the Controller; otherwise, specify the HTTP port. See [Controller SSL Enabled Property](#).

**Element in controller-info.xml**: `<controller-port>`

**System Property**: `-Dappdynamics.controller.port`

**Type**: Positive Integer

**Default**: For on premises installations, port 8090 for HTTP and port 8181 for HTTPS are the default ports the Controller listens to. For the SaaS Controller Service, port 80 for HTTP and port 443 for HTTPS are the default ports the Controller listens to.

**Required**: Yes
Account Access Key Property

**Description:** This is the account access key used to authenticate with the Controller.

**Element in controller-info.xml:** `<account-access-key>`

**System Property:** `-Dappdynamics.agent.accountAccessKey`

**Type:** String

**Default:** None

**Required:** Yes. Prior to version 4.1, this property was required only for SaaS and multi-tenant Controllers. The account access key property is now required to authenticate all agent to Controller communications.

**Example:** `-Dappdynamics.agent.accountAccessKey=165e65645-95c1-40e3-9576-6a1424de9625`

Multiple Agent Environment Properties

The following properties are useful when you are configuring your system to include more than one Database Agent.

**Database Agent Name Property**

**Description:** This property uniquely identifies the Database Agent to the Controller.

**System Property:** `-Ddbagent.name=<db_agent_name>`

**Type:** ASCII string, including spaces. If `<agent_name>` contains spaces, you must enclose the entire name in double quotes (" ").

**Default:** Default Database Agent

**Required:** Required when you have more than one Database Agent reporting to the Controller.

You may want to run multiple **Database Agents** under the following conditions:

- When you have databases spread across multiple networks, and one machine cannot access all the databases on all the networks. In this case, you can have a uniquely named Database Agent in each network that monitors the databases only visible on that network.
- If you have multiple SQL Server instances that each require different credentials, and you want to connect to them via Windows Authentication. When using Windows Authentication, the Database Agent will use the credentials of the currently logged in user.
- When you want one or more agents to back up a primary agent. The database agent names for your backup agents must match the database agent name of your primary agent. The last agent you launch will be the primary agent.

**Example:** `-Ddbagent.name="Scarborough Network Database Agent"`

Multi-Tenant Mode Properties

If the AppDynamics Controller is running in multi-tenant mode or if you are using the AppDynamics SaaS Controller, specify the account name and account access key for this agent to authenticate with the Controller.

If the Controller is running in single-tenant mode (the default) there is no need to configure these values.

**Account Name Property**

**Description:** This is the account name used to authenticate with the Controller.

If you are using the AppDynamics SaaS Controller, the Account Name is in the Welcome email AppDynamics sent to you.

**Element in controller-info.xml:** `<account-name>`

**System Property:** `-Dappdynamics.agent.accountName`

**Type:** String

**Default:** None

**Required:**

- Yes for AppDynamics SaaS Controller and other multi-tenant users
- No for single-tenant users.
Proxy Properties for the Controller

These properties route data to the Controller through a proxy.

Proxy Host Property

**Description:** This is the proxy host name or IP address.

**Element in controller-info.xml:** Not applicable

**System Property:** -Dappdynamics.http.proxyHost

**Type:** String

**Default:** None

**Required:** No

Proxy Port Property

**Description:** This is the proxy HTTP(S) port.

**Element in controller-info.xml:** Not applicable

**System Property:** -Dappdynamics.http.proxyPort

**Type:** Positive Integer

**Default:** None

**Required:** No

Proxy User Name

**Description:** The name of the user that is authenticated by the proxy host.

**Element in controller-info.xml:** Not applicable

**System Property:** -Dappdynamics.http.proxyUser

**Type:** String

**Default:** None

**Required:** No

Proxy Password

**Description:** The absolute path to the file containing the password of the user that is authenticated by the proxy host. The password must be the first line of the file.

If Use Encrypted Credentials is false, enter the password in plain text. If Use Encrypted Credentials is true, encrypt the password. See Encrypt Agent Credentials.

**Element in controller-info.xml:** Not applicable

**System Property:** -Dappdynamics.http.proxyPasswordFile

**Type:** String

**Default:** None

**Required:** No

Other Properties

Controller SSL Enabled Property

**Description:** This property specifies whether the agent should use SSL (HTTPS) to connect to the Controller. If SSL Enabled is true, set the Controller Port property to the HTTPS port of the Controller. See Controller Port Property.
Element in controller-info.xml:  <controller-ssl-enabled>

System Property:  -Dappdynamics.controller.ssl.enabled

Type:  Boolean
Default:  False
Required:  No

Disable Sybase sp_sysmon execution

Description:  This property specifies whether the agent should disable Sybase. You may want to disable Sybase monitoring with Database Visibility if you are already using other tools to monitor Sybase. Refer to Sybase Database Permissions for information on configuring Sybase permissions.

If an agent is run with the -Ddbagent.disable.sybase.ase.system.monitoring flag, Database Visibility will stop executing sp_sysmon for Sybase databases. As a result, the following metrics might not show reliable data:

- Calls per Minute (KPI)
- All metrics under Server Statistic in the Metric Browser
- The Load value on the Sybase dashboard

Element in controller-info.xml:  Not applicable

System Property:  -Ddbagent.disable.sybase.ase.system.monitoring

Type:  Boolean
Default:  False
Required:  No

Resolve Oracle RAC topology

Description:  Use local listener to resolve Oracle RAC topology.

Element in controller-info.xml:  No

System Property:  -Ddbagent.oracle.rac.use_local_listener

Type:  Boolean
Default:  False
Required:  Yes
Enable SSL and SSH for Database Agent Communications

On this page:

- Enable SSL for the Database Agent
- Enable SSH for the Database Agent

Enable SSL for the Database Agent

This topic covers how to configure the Database Agent to connect to the Controller using SSL. It assumes that you use a SaaS Controller or have configured the on-premise Controller to use SSL.

The Database Agent supports extending and enforcing the SSL trust chain when in SSL mode.

Requirements

Gather the following information:

- The Controller SSL port.
  - For SaaS Controllers the SSL port is 443.
  - For on-premises Controllers the default SSL port is 8181, but you may configure the Controller to listen for SSL on another port.
- The signature method for the Controller's SSL certificate:
  - A publicly known certificate authority (CA) signed the certificate. This applies for DigiCert, Verisign, Thawte, and other commercial CAs.
  - A CA internal to your organization signed the certificate. Some companies maintain internal certificate authorities to manage trust and encryption within their domain.
  - The Controller uses a self-signed certificate.

Establish Trust for the Controller's SSL Certificate

To establish trust between the Database Agent and the AppDynamics Controller, you must create an agent truststore that contains the root certificate for the authority that signed the Controller's certificate.

If you secured your on-premises Controller with a self-signed certificate, see Keystore Certificate Extractor Utility for instructions to create the agent keystore.

1. Obtain one of the following root certificates:
   - DigiCert Global Root CA for the AppDynamics SaaS Controller
   - the root certificate for the publicly known certificate authority (CA) that signed the certificate for your on-premises Controller
   - the root certificate for the internal CA that signed the Controller certificate for your on-premises Controller

2. Run the Java keytool command to create the Database Agent truststore:

   ```
   keytool -import -alias rootCA -file <root_certificate_file_name>
   -keystore cacerts.jks -storepass <truststore_password>
   ```

   For example:

Make note of the truststore password, you need it to configure the Database Agent.

3. Install the agent truststore to the Database Agent configuration directory:

<db_agent_home>/conf/

Secure the Database Agent Truststore

AppDynamics recommends you take the following security measures to prevent tampering with the Database Agent truststore:

- Secure the truststore file through file system permissions:
  - Make the Database Agent truststore readable by any user.
  - Make the truststore owned by a privileged user.
  - Make the truststore writable only by the specified privileged user.
- Secure the Database Agent configuration files so that they are only readable by the agent runtime user and only writable by a privileged user:

Configure SSL System Properties for the Database Agent

1. Configure the following system properties in the versioned controller-info.xml: <db_agent_home>/<version_number>/conf/controller-info.xml. See "SSL Configuration Properties" on Database Agent Configuration Properties for full details on each property.

   - **Controller Port**: the SSL port for the controller. 443 for AppDynamics SaaS.
     
     <controller-port>443</controller-port>
   
   - **Controller SSL Enabled**: true.
     
     <controller-ssl-enabled>true</controller-ssl-enabled>
   
   - **Controller Keystore Password**: the plain text password for the Database Agent truststore.
     
     <controller-keystore-password>MySecurePassword</controller-keystore-password>
   
   - **Controller Keystore Filename**: path of the Database Agent truststore relative to <db_agent_home>/<version>/conf. Required if you use a truststore other than the default <db_agent_home>/<version_number>/conf/cacerts.jks.
     
     <controller-keystore-filename>../../conf/cacerts.jks</controller-keystore-filename>

   You can specify the Controller port and enable SSL for the Controller in the JVM startup script, but you must specify the truststore password and filename in the controller-info.xml file.

2. Save your change to the controller-info.xml file and restart the Database Agent.

Sample SSL controller-info.xml configuration
<?xml version="1.0" encoding="UTF-8"?>
<controller-info>
  <controller-host>mycompany.saas.appdynamics.com</controller-host>
  <controller-port>443</controller-port>
  <controller-ssl-enabled>true</controller-ssl-enabled>
  <controller-keystore-password>MySecurePassword</controller-keystore-password>
  <controller-keystore-filename>../../conf/cacerts.jks</controller-keystore-filename>
...
Enter output agent truststore file name:
<controller_home>/appserver/glassfish/domains/domain1/config/keystore.jks

The password for the Controller's certificate, which defaults to "changeit". If you don't include a password, the extractor applies the password "changeit" to the output truststore.

- Example command to execute kr.jar

    java -jar kr.jar
    <controller_home>/appserver/glassfish/domains/domain1/config/keystore.jks cacerts.jks <controller_certificate_password>

3. Install the agent trust store to the agent configuration directory:

    <db_agent_home>/conf/

Enable SSH for the Database Agent

For Linux hosts only
Applies only when Database Agent is running on a Linux host.

When the Database Agent is running on Linux and you want to monitor hardware, except to monitor the local host, authentication is required and a password is passed between the Database Agent and the database server.

SSH port option
The SSH port option does not appear unless the Database Agent is running on Linux.

1. On the agent machine, generate the rsa or dsa key as follows:
1. Generate rsa key

% ssh-keygen -b 1024 -f id_rsa -t rsa

or

2. Generate dsa key

% ssh-keygen -b 1024 -f id_dsa -t dsa

This will create a rsa or dsa 1024-bit key and put the keys into /home/<user_name>/.ssh/id_rsa and
/home/<user_name>/.ssh/id_rsa.pub or /home/<user_name>/.ssh/id_dsa and /home/<user_name>/.ssh/id_dsa.pub files. Do
not change the names of these files.

2. Copy the private key, /home/<user_name>/.ssh/id_rsa or /home/<user_name>/.ssh/id_dsa into the <db-agent_home>/keys
directory.

3. On both the monitored machine and the agent machine, verify that you have the correct permissions on the .ssh directory, or
set them as follows:

%.ssh directory permissions

% cd /home/<user_name>
% chmod 755 .ssh

4. On the monitored machine, verify that you have a /home/<user_name>/.ssh/authorized_keys file. If you do not have this file,
create the authorized_keys file on the monitored machine as follows:
4. Create authorized_keys file

```bash
% cd /home/<user_name>/.ssh
% touch .ssh/authorized_keys
```

5. Verify that you have the correct permissions to the `/home/<user_name>/.ssh/authorized_keys` file, or change the permissions as follows:

```bash
% cd /home/<user_name>/.ssh
% chmod 644 authorized_keys
```

6. Append (do not copy) the file `id_rsa.pub` or the file `id_dsa.pub` to the file, `/home/<user_name>/.ssh/authorized_keys`, such as follows:

```bash
% echo /home/<user_name>/.ssh/id_rsa.pub
>> /home/<user_name>/.ssh/authorized_keys
```

or
Append dsa public key to authorized key

% echo /home/<user_name>/.ssh/id_dsa.pub
>> /home/<user_name>/.ssh/authorized_keys

7. The SSH port of the database Collector is set to 22 by default. You can change it by navigating to the Monitoring Hardware section of the Collector configuration dialog. In the dialog, set the SSH port to the port you require.
8. Save your change to the Collector configuration and restart the Database Agent.

Enable SSH via PEM certificate

For Linux, AppDynamics also supports certificate-based authentication via Privacy Enhanced Mail (PEM). To implement certificate-based authentication:

1. Enable the **Use certificate** option in the Monitoring hardware section of the Collector configuration dialog.
2. Copy the PEM file to the `<db_agent_home>/keys` directory. Note, if the `home/<user_home>/ssh` directory exists, the agent will use the certificate found there.
3. Restart the agent.
Monitor SSL-enabled PostgreSQL on Amazon RDS

To monitor a PostgreSQL instance that uses SSL connections, complete the following steps.

1. Download PEM file from Amazon and copy to a local directory.
2. Convert the PEM file to a DER file using the following `openssl` command:

   ```
   openssl x509 -outform der -in rds-combined-ca-bundle.pem -out rds-combined-ca-bundle.der
   ```

3. Add the certificate to the Java keystore using the following command:

   ```
   sudo keytool -import -noprompt -trustcacerts -alias AmazonRDS -file rds-combined-ca-bundle.der -keystore $JAVA_HOME/jre/lib/security/cacerts -storepass changeit
   ```

   You can verify that the certificate was added by running the following command:

   ```
   keytool -list -keystore $JAVA_HOME/jre/lib/security/cacerts -storepass changeit
   ```

4. Re-start the dbagent process to register the certificate you added.

5. In the Controller, create a new collector for PostgreSQL. In the Custom JDBC Connection String field, enter the following JDBC string:

   ```
   jdbc:postgresql://<RDS-Hostname>:<RDS-Port>/postgres?ssl=true
   ```
Monitor SSL-enabled MySQL on Amazon RDS

To monitor a MySQL instance that uses SSL connections, enable MySQL in SSL mode, consulting the online documentation appropriate to your deployment.

Download the files shown below as they are required to complete this procedure:

- ca.pem
- server-cert.pem
- server-key.pem
- client-cert.pem
- client-cert.key

Run the following commands at the command line. Substitute URLs and other information from your deployment for the placeholders in the examples.

1. Test your local SSL connection

```
mysql -h ec2-11-111-111-11.us-west-2.compute.amazonaws.com -u
Testssl --ssl-ca=/etc/certs/ca.pem
--ssl-cert=/etc/certs/server-cert.pem
--ssl-key=/etc/certs/server-key.pem -p
```

2. Verify the remote connection

```
mysql -h ec2-11-111-111-11.us-west-2.compute.amazonaws.com -u
Testssl --ssl-ca=/home/appdynamics/cert/ca.pem
--ssl-cert=/home/appdynamics/cert/client-cert.pem
--ssl-key=/home/appdynamics/cert/client-key.pem -p
```

3. Import the ca.pem file to the default truststore

```
sudo keytool -importcert -alias MySQLCACert -file ca.pem -keystore
$JAVA_HOME/jre/lib/security/cacerts -storepass changeit
```

- Do not forget the password, as you need it in step 7

4. Convert the client key and certificate files to a PKCS #12 archive
5. Import the client key and certificate into a Java keystore

```bash
openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem
-name "mysqlclient" -passout pass:changeit -out client-keystore.p12
```

```
sudo keytool -importkeystore -srckeystore client-keystore.p12
-srcstoretype pkcs12 -srcstorepass changeit -destkeystore
$JAVA_HOME/jre/lib/security/cacerts -deststoretype JKS
-deststorepass changeit
```

6. Verify that the certificate was added

```bash
keytool -list -keystore $JAVA_HOME/jre/lib/security/cacerts
-storepass mypassword
```

7. Start the db-agent after ensuring that the following properties are added

```bash
java -jar
-Djavax.net.ssl.trustStore=$JAVA_HOME/jre/lib/security/cacerts
-Djavax.net.ssl.trustStorePassword=changeit
-Djavax.net.ssl.keyStore=$JAVA_HOME/jre/lib/security/cacerts
-Djavax.net.ssl.keyStorePassword=changeit db-agent.jar
```

8. In the Controller, create a new collector for MySQL
   - In the Configuration screen, add the connection property with Property Name "useSSL" and Property Value "true"
   - Alternatively, you can also use the following custom connection string:

   ```
jdbc:mysql://<RDS-Hostname>:<RDS-Port>/database?useSSL=true
```
Monitor SSL-enabled MongoDB

You can configure your MongoDB instance to use SSL connections.

1. Create a keystore on the database agent host.
2. Import the SSL certificates to this keystore using the following command:

   keytool -importcert -trustcacerts -file <mongodb certificate location> -keystore <keystore e.g. path/mongo.jks> -storepass <keystore password> -alias <alias>

3. Start the database agent with the following flags:

   java -Ddbagent.name=<DBAgent Name>
   -Djavax.net.ssl.trustStore=<keystore e.g. path/mongo.jks>
   -Djavax.net.ssl.trustStorePassword=<keystore password> -jar
   db-agent.jar

4. Create the mongodb collector with custom URL:

   mongodb://<mongodb-host>:27017/?ssl=true
Install the Database Agent as a Windows Service

To create the Database Agent as a Windows Service, run the following command as an administrator:

```
cscript <db_agent_home>\InstallService.vbs <jvm_options>
```

A Windows Services named "Appdynamics Database Agent" will be created.

Alternatively, you can create the database agent as a Windows Service in the silent mode where the script skips user input for installation, run the following command as an administrator:
cscript \<db_agent_home>\InstallService.vbs SilentInstall <jvm_options>

The screenshot below shows the Windows Service Control Manager.

If you choose to use Windows authentication for Microsoft SQL Server, you must change your login credentials and restart the database agent service.

**Uninstall the Database Agent as a Service**

To uninstall the database agent service, run the following command as an administrator:

cscript \<db_agent_home>\UninstallService.vbs
Running the Agent on Windows Using the Task Scheduler

It is recommended to run the Database Agent as a Windows service. However, if you choose to run the agent using Task Scheduler, ensure the agent runs with the highest privileges available. When using Task Scheduler, if the host is restarted, the agent will not start automatically.
Enable Snapshot Correlation for Oracle

In order for the Oracle Collector AppDynamics Database Visibility to collect database queries, sessions, clients, and schemas information that occurred during snapshots captured by the Java Agent, you need to set the jdbc-dbcam-integration-enabled App Agent property for the Java Agent.

For information on the jdbc-dbcam-integration-enabled property, see App Agent Node Properties.

For information on snapshot correlation between the Oracle database Collector and a Java business transaction, see Access Database Visibility from Application Monitoring Views.
Increasing JVM Memory

To monitor multiple databases, you may need to increase the JVM memory allocation size. Increased activity on the databases you are monitoring results in increased memory usage.

Use one of the following commands to start the agent, and to initially allocate 1536 MB to the agent instead of the default of 64 MB.

For Windows 64-bit

C:\java -Xmx1536m -Djava.library.path="<db_agent_home>\auth\x64" -jar <db_agent_home>\db-agent.jar

For Linux

% java -Xmx1536m -jar <db_agent_home>/db-agent.jar
Database Agent Logging

Configure Agent Logging

By default, the Database Agent writes log files to the log directory.

You can configure the agent to generate log files.

1. From the agent home directory, navigate to log directory.
2. Open log4j.xml.
3. Find <logger name="com.singularity"> and <logger name="com.appdynamics">. Inside each of those tags, set level value to debug, as shown below:

```xml
<logger name="com.singularity">
    <level value="debug"/>
</logger>

<logger name="com.appdynamics">
    <level value="debug"/>
</logger>
```

4. Let the agent run for 10 minutes to generate some logs.
5. From the agent home directory, navigate to the log directory to view your logs.
Configure Controller Settings for Database Agents

You can configure parameters for the controller, such as the number of events sent to the Events Service at a time.

To Change the Controller Settings for Database Agents

1. Log in to the Administration Console.
2. Click Controller Settings.
3. Change the settings as needed and Save.

Controller Settings Reference for Database Agents

<table>
<thead>
<tr>
<th>Property Name</th>
<th>About the property</th>
<th>Default</th>
<th>Minimum value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbmon.event.publisher.esclient.batch.size</td>
<td>Batch size for each Event Service client used by Database Monitoring.</td>
<td>1500</td>
<td>200</td>
</tr>
<tr>
<td>dbmon.event.publisher.queue.size</td>
<td>Maximum measurement queue size maintained by Database Monitoring.</td>
<td>100000</td>
<td>100000</td>
</tr>
<tr>
<td>dbmon.config.data.retentionperiod</td>
<td>Number of days to retain database monitoring data (queries, clients, users, sessions, schemas) excluding the current day. Data reported before the retention period might be unrecoverable, even for a momentary update of this property.</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>dbmon.config.max.custommetric property</td>
<td>Number of custom metrics that can be created per collector.</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
Start and Stop the Database Agent

On this page:
- Start the Agent
- Stop the Agent

Start the Agent

If you are using an on-premises version of Database Visibility, you must start the Events Service before you start the Database Agent. Before starting the agent, you must configure the properties in this file. You must specify these properties: Controller Host Property, Controller Port Property, and Account Access Key Property.

Start the Agent in a Command Shell

The following assumes that all the necessary parameters have been specified in the controller-info.xml. The controller-info.xml file describes the properties of agent to Controller communications. Note that <db_agent_home> is the absolute path to the location where the Database Agent is installed. Setting the heap size is recommended as shown below. See Database Visibility System Requirements for more information.

**Windows**

cd <db_agent_home>

start-dbagent.bat -Xms<min_heap_size> -Xmx<max_heap_size>

**Linux**

cd <db_agent_home>

./start-dbagent -Xms<min_heap_size> -Xmx<max_heap_size> &

Stop the Agent

Stop the Agent in a Command Shell

If the Database Agent process is running in the background, you can stop it by simply entering the kill command with the process ID as the argument. If it is running in the foreground in a console, you can press Ctrl+C to shut down the agent.

Stop the Agent Service in Windows

In the Windows Services application, select AppDynamics Database Agent and click Stop.
Start the Database Agent Automatically on Linux

You can enable the database agent to start automatically whenever the machine starts up. To do so, create an initialization script for starting the agent.

1. Create an initialization script that starts the agent, as in the sample initialization script that is an attachment to this page.
   
   In your script, set the JAVA and AGENT_HOME values to the paths for your system. Also configure agent options and heap size, as needed.

2. Enable execution permissions for the script. For example, given an initialization script named db-agent, enter:

   ```
   sudo chmod 775 db-agent
   ```

3. Place the script in the initialization directory on your system, typically /etc/init.d. Alternatively, create a symbolic link to the script from the init.d directory to the script if you want to keep it in another location.

4. Add the script as a service as follows:
   
   - On Red Hat and most Linux Operating Systems, run these commands, replacing "db-agent" with the name of your file or symbolic link:

     ```
     chkconfig --add db-agent
     chkconfig --level 2345 db-agent on
     ```

   - On Ubuntu and Debian Operating Systems, run this command, replacing "appdcontroller" with the name of your file or symbolic link:

     ```
     update-rc.d -f db-agent start 99 2 3 4 5 .
     ```

     In the command:
     - `start` is the argument given to the script (start, stop).
     - `99` is the start order of the script (1 = first one, 99 = last one)
     - `2 3 4 5` are the runlevels to start
     - Be sure to include the dot (.) at the end of the command.

The Database Agent now starts automatically upon machine startup. When setting the agent to automatically start up using Task Scheduler when the machine starts or restarts, ensure the agent runs with the highest privileges available.
Start the Database Agent Automatically on Windows

Related pages:
- Administer the Database Agent
- Database Agent Configuration Properties

You can enable the Database Agent to start automatically whenever the machine starts up. To do so, create a batch file for starting the agent.

1. Create a batch file containing the following line:

```
For 64-bit systems, allocating 2 GB of initial heap size
```

```
java -Xmx2048m -Djava.library.path="<db_gent_home>\auth\x64" -jar
<db_agent_home>\db-agent.jar
```

For a 32-bit system, set `java.library.path="<db_agent_home>\auth\x32"`

If you are monitoring multiple databases, you must increase the size of the memory allocation. For more information, see Database Visibility System Requirements.

2. Click Control panel > Scheduled Tasks.
3. Select the batch file to execute. This is the file created in step 2.
4. Create a new task.
5. Select When my computer starts. When setting the agent to automatically start up using Task Scheduler when the machine starts or restarts, ensure the agent runs with the highest privileges available.
6. Enter the administrator's credentials to run the Database Agent as that user.
7. Click Finish.
8. To start the Database Agent for the first time, right-click the scheduled task and click Run.
Add Database Licenses

The Controller requires a license for the total number of databases to monitor concurrently. One database agent can monitor 100+ databases.

Add New Licenses

1. Obtain a license.lic file with your existing AppDynamics licenses plus the new licenses from AppDynamics support.
2. Copy the license file in the Controller home directory. The Controller may take a few minutes to detect the new license. Restart the Controller to force immediate license detection.

To see license information, navigate to Settings (gear icon) > License. The Application Performance Management and Database Server Monitoring section shows the number of database license units provisioned and the maximum number allowed. In general, one license unit is required for each database instance.

For MongoDB, each primary node requires one license.

If you disable a collector, it still consumes a license.

Transfer a License Between Databases

A database license is allocated to the first agents that register with the Controller up to the licensed limit.

To transfer a license for the Database Agent to another database, you simply remove the Collector for the old database and then add a new Collector for the database you want to monitor instead. See Add Database Collectors.
Add Database Collectors

On this page:
- Required Database Permissions
- AppDynamics Permissions
- Add a Database Collector
- Edit a Database Collector
- Delete a Database Collector
- Export Database Collector Data
- Troubleshooting Collector Problems

To enable the Database Agent to collect data, you must add and configure a Database Collector for each database instance that you want to monitor. The Database Collector is the process that runs within the Database Agent to collect performance metrics about your database instances and database servers.

Required Database Permissions
You must have the required permissions to monitor your databases. See Database Visibility Supported Environments.

AppDynamics Permissions
To add, edit, and create collectors, users need a role with the proper permissions:
- Can Create Collectors
- Can Edit All Collectors
- Can Delete All Collectors
For more information, see Database Permissions.

Add a Database Collector

1. Click the Configuration tab in the left panel menu.
2. Click the Collectors option.
3. In the Collectors window, click the Add ( + ) option.
4. Complete each field of the Create New Collector dialog
5. If you want to add a sub-collector to be monitored, click Advanced >> +Add sub-collector to enter host and port details.

Sub-collectors are applicable only to relational database types.

6. Click OK. Your database administrator can provide you with the necessary details.

After you have added a collector, you can configure the collector for your database. You can also link a database on the application flow maps to a database instance monitored by Database Visibility.

Hardware Monitoring
For information on the fields you must complete in order for the Database Agent to monitor the database server in addition to the database server, see Configure the Database Agent to Monitor Server Hardware.
Verify Collector Setup

Once the Collector is up and running, in just a short time you can start viewing the historical activity data.

The Collector configuration window now has a collector icon you can click to edit the details of the collector if required. The Collector will also appear in the list of Databases shown in the left navigation menu. It might take a few minutes before the Collector and its metrics are reported.

From the left navigation menu, click Databases to see a high-level view of the activity of all the configured Collectors.

Click the name of the database to see more details of the metrics AppDynamics Database Monitoring has captured.

For information on using and interpreting the Collector windows, see Monitor Databases and Database Servers.

Edit a Database Collector

From the Collectors window, you can edit any of the details of the collector except the type of database platform to monitor.

For more information on collector fields, see the collector configuration page for your database.

- Configure IBM DB2 Collectors
- Configure Microsoft SQL Server Collectors
- Configure Microsoft Azure Collectors
- Configure MongoDB Collectors
- Configure MySQL Collectors
- Configure Oracle Collectors
- Configure PostgreSQL Collectors
- Configure Sybase Collectors
- Configure Sybase IQ Collectors

Delete a Database Collector

From the Collectors window, you can delete a database Collector.

Export Database Collector Data

From the Collectors window, you can click on Actions > Export Data to generate a CSV file containing the following data for each collector:

- Collector name
- Host name
- Port number
- Username
- Database type
- Agent name

Troubleshooting Collector Problems

Collectors that have not been configured correctly, or that cannot connect to the database for any reason, will show an error on the Databases overview page and individual database dashboards. Hovering over the error icon displays the potential reason for the error.

If your Collector isn't reporting any metrics after a few minutes, and you know the database is up and running with activity, check the Events window. Agent Diagnostic Events can appear if the password is incorrect or communication errors have occurred. The message summaries on the Events window can help you diagnose and troubleshoot Collector problems.

Check the collector configuration to ensure all the values you entered are correct.
Ensure that your Database Agent has network connection to the databases you want to monitor along with the required permissions. See Database Monitoring Requirements and Supported Environments.
Configure IBM DB2 Collectors

To monitor IBM DB2 with Database Visibility, you must be running DB2 9.x or newer.

Connection Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>The database type that you want to monitor.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>The Database Agent that manages the collector.</td>
</tr>
<tr>
<td>Name</td>
<td>The name you want to identify the collector by.</td>
</tr>
<tr>
<td>Hostname or IP Address</td>
<td>The hostname or IP address of the machine that your database is running on.</td>
</tr>
<tr>
<td>Database</td>
<td>The name of the database instance that you want to monitor.</td>
</tr>
<tr>
<td>Listener Port</td>
<td>The TCP/IP address of the port on which your database communicates with the Database Agent</td>
</tr>
<tr>
<td>Custom JDBC Connection String</td>
<td>The JDBC connection string generated by the database agent, for example, jdbc:db2:///CustomConnection. You can also specify a custom connection string, which is useful for setting custom authentication options.</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for IBM DB2 LUW.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user who is connecting to and monitoring the database through the Database Agent.</td>
</tr>
<tr>
<td>Sub-Collectors</td>
<td>Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type.</td>
</tr>
<tr>
<td>Note</td>
<td>All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub-collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API. You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.</td>
</tr>
<tr>
<td>Connection Properties</td>
<td>Click to add a new JDBC connection property or edit an existing property for relational databases.</td>
</tr>
</tbody>
</table>
Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.

Click to enable verbose mode logging, which logs all communications between the Controller and the Collector. Enable only during troubleshooting because logging consumes a lot of disk space. If you have enabled logging, you can click the logging icon in the Log column of the Collector Administration window to view the log file. The log files are located in the <db_agent_home>/agent directory and have the format <CollectorName>_out.log and <CollectorName>_err.log.

### User Permissions for IBM DB2 LUW

The monitoring user needs SYSMON authority and connect privileges to monitor. In general, this user must be a part of the sysmon_group. To generate an execution plan, you must create the associated EXPLAIN tables in the schema.

### DB2 version 9.7 and greater

For complete Database Visibility functionality, the following monitoring switches of the DB2 server need to be enabled: "TIMESTAMP".

You can enable the "TIMESTAMP" monitoring switch by the following commands:

```bash
update dbm cfg using dft_mon_timestamp on;
update db cfg using mon_act_metrics BASE
```

### Privileges

```bash
grant select on SYSIBMADM.MON_CURRENT_SQL to user DBMon_Agent_User
grant select on SYSIBMADM.MON_LOCKWAITS to user DBMon_Agent_User
grant execute on function SYSPROC.MON_GET_CONNECTION to user DBMon_Agent_User
grant execute on function SYSPROC.MON_GET_PKG_CACHE_STMT to user DBMon_Agent_User
grant execute on function SYSPROC.MON_GET_TRANSACTION_LOG to DBAgent_User
grant execute on function SYSPROC.MON_GET_DATABASE to DBAgent_User
```

Replace DBMon_Agent_User with the user name under which you run the Database Visibility Agent.
DB2 version 9.5

For complete Database Visibility functionality, the following monitoring switches of the DB2 server need to be enabled: "STATEMENT", and "TIMESTAMP".

You can enable these monitoring switches by the following commands:

```sql
update dbm cfg using dft_mon_stmt on;
update dbm cfg using dft_mon_timestamp on;
```

Privileges

```sql
grant select on SYSIBMADM.SNAPSTMT to user DBMon_Agent_User
grant select on SYSIBMADM.SNAPAPPL_INFO to user DBMon_Agent_User
grant select on table SYSIBMADM.ENV_PROD_INFO to user DBMon_Agent_User
```

where DBMon_Agent_User is the user name under which you run the Database Visibility Agent.

User Permissions when restrict_access is set to YES

If your database has the restrict_access parameter set to YES, you must grant the following privileges.
grant select on SYSIBMADM.MON_CURRENT_SQL to user DBMon_Agent_User;
grant select on SYSIBMADM.MON_LOCKWAITSTo user DBMon_Agent_User;
grant execute on function SYSPROC.MON_GET_CONNECTION to user DBMon_Agent_User;
grant select on SYSIBMADM.SNAPAPPL_INFO to user DBMon_Agent_User;
grant EXECUTE on function SYSPROC.MON_GET_PKG_CACHE_STMT to user DBMon_Agent_User;
grant execute on function SYSPROC.MON_GET_TRANSACTION_LOG to user DBMon_Agent_User;
grant EXECUTE on package NULLID.SQLC2K26 to user DBMon_Agent_User;
grant select on SYSBM.SYSDUMMY1 to user DBMon_Agent_User;
grant select on SYSIBMADM.ENV_PROD_INFO to user DBMon_Agent_User;
grant select on SYSIBMADM.ENV_SYS_RESOURCES to user DBMon_Agent_User;
grant execute on function SYSPROC.SNAP_GET_STMT(varchar(),Integer) to user DBMon_Agent_User;
grant select on SYSCAT.STATEMENTS to user DBMon_Agent_User;
grant select on SYSIBMADM.DBCFG to user DBMon_Agent_User;
grant execute on function SYSPROC.SNAP_GET_DB(varchar(),Integer) to user DBMon_Agent_User;
grant EXECUTE on package NULLID.SYSSH200 to user DBMon_Agent_User;
grant select on SYSIBMADM.ENV_SYS_RESOURCETo user DBMon_Agent_User;
grant select on SYSCAT.DBAUTH to user DBMon_Agent_User;
grant execute on function SYSPROC.SNAP_GET_DBM(Integer) to user DBMon_Agent_User;
grant select on syscat.schemata to user DBMon_Agent_User;

Replace DBMon_Agent_User with the user name under which you run the Database Visibility Agent.

**Generate Execution Plans**

To generate an execution plan in DB2 10.1 or later, you must create the associated EXPLAIN tables in the schema.

1. Create a separate file system.
2. Create a separate tablespace.

```sql
db2 "CREATE STOGROUP DBMon_Agent_User ON '/srv/db2/username1/$\{INSTANCENAME\}"
```

3. Create a schema.
3. 

```
db2 "CREATE SCHEMA DBMon_Agent_User"
```

4. Grant the following privileges:

```
db2 "GRANT USE OF TABLESPACE DBMon_Agent_User to user DBMon_Agent_User"
db2 "CREATEIN ON SCHEMA DBMon_Agent_User TO USER DBMon_Agent_User"
```

5. Call the SYSPROC.SYSINSTALLOBJECTS procedure:

```
db2 "CALL SYSPROC.SYSINSTALLOBJECTS('EXPLAIN', 'C', 'tablespace-name', 'schema-name' ) "
```
Configure Microsoft SQL Server Collectors

On this page:
- Connection Details
- User Permissions for Microsoft SQL Server

To monitor Microsoft SQL Server with Database Visibility, you must be running the 2005 version or newer.

To configure an Azure SQL Managed Instance, follow the configuration procedures of a Microsoft SQL Server Collector described in this topic.

Connection Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>The database type that you want to monitor.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>The Database Agent that manages the collector.</td>
</tr>
<tr>
<td>Name</td>
<td>The name you want to identify the collector by.</td>
</tr>
<tr>
<td>Hostname or IP Address</td>
<td>The hostname or IP address of the machine that your database is running on.</td>
</tr>
<tr>
<td>Failover Partner</td>
<td>The hostname or IP address of the failover partner.</td>
</tr>
<tr>
<td>Listener Port</td>
<td>The TCP/IP address of the port on which your database communicates with the Database Agent</td>
</tr>
<tr>
<td>Custom JDBC Connection String</td>
<td>The JDBC connection string generated by the database agent, for example, jdbc:sqlserver://. You can also specify a custom connection string, which is useful for setting custom authentication options.</td>
</tr>
<tr>
<td>Window Authentication</td>
<td>Click to enable Windows authentication when connecting to the database.</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Microsoft SQL Server.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user who is connecting to and monitoring the database through the Database Agent.</td>
</tr>
</tbody>
</table>
Sub-Collectors
Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases.
To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type.

**Note**
- All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub-collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API.
- You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.

Connection Properties
Click to add a new JDBC connection property or edit an existing property for relational databases.

CyberArk
Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.

Exclude Databases
The databases that you want to exclude, separated by commas.

Logging Enabled
Click to enable verbose mode logging, which logs all communications between the Controller and the Collector. Enable only during troubleshooting because logging consumes a lot of disk space. If you have enabled logging, you can click the logging icon in the Log column of the Collector Administration window to view the log file. The log files are located in the <db_agent_home>\agent directory and have the format <CollectorName>_out.log and <CollectorName>_err.log.

**Note**
To handle High Availability (Multi-AZ) for Amazon RDS, set the databaseName JDBC connection string property to a user database.

User Permissions for Microsoft SQL Server
The user account used for monitoring can be a Windows authenticated account (if the Database Agent is running on Windows) or SQL Server authenticated (if the Database Agent is running on Windows or Linux).

Required Permissions to See Execution Plans
The SQL Server user, specified in the Create Collector > Connection Details section must be a SQL Server Authenticated user that is a member of the sysadmin server role or a Windows Authenticated Account with SHOWPLAN access on each database.

For more information, see Showplan Security and SHOWPLAN Permission and Transact-SQL Batches in the SQL Server documentation.

SQL Server Authentication
If you are running AppDynamics Database Visibility on Linux, then you must use SQL Server authentication.

If your SQL Server database uses mixed-mode authentication, then you can use SQL Server authentication or Windows authentication. If you would like to lock the role/permissions for the account down, then the account running AppDynamics Database Visibility requires:
- View any database
- View any definition
- View server state

One additional requirement for I/O monitoring is to give permissions on a System view called sys.sysaltfiles. To do this you need to select the master database > Views > System Views > Properties for sys.sysaltfiles and then give select permissions on the object to the Public role.

Windows Authentication
If you would like to use a Windows authenticated account to connect to the SQL Server database, the following is required:

- When creating the collector from the Create New Collector dialog, do not specify Username and Password in the database Connection Details.
- Also, the agent must be started with the path to its authentication library. For more information, see Windows Authentication for Microsoft SQL Server.
- If you are using the Windows Service to launch the agent, you must change your logon credentials and restart the database agent service.

Server Level Permissions Required for SQL Server Logon
The easiest way to configure permissions for monitoring SQL Server is to grant server-level permissions for the monitoring user. Otherwise, you will need to grant permissions on individual objects, as described in the Object Permissions for Monitoring SQL Server section. Follow the steps below to create a SQL Server user with the minimum permissions required to monitor SQL Server.

1. Using SQL Server Management Studio, create a new login for the AppDynamics SQL Server Database Collector, such as DBMon_Agent_User.
2. From the User Mapping tab, map the new user to the master and msdb databases.

Viewing Object Information
To view object information on the Database > Objects Browser, map the monitoring user to the databases of interest.

3. Once you have created the login, grant the following privileges to the user, substituting DBMon_Agent_User with the name you specified on the Login - New window:

```
use master
GRANT VIEW ANY DATABASE TO DBMon_Agent_User;
GRANT VIEW ANY definition to DBMon_Agent_User;
GRANT VIEW server state to DBMon_Agent_User;
GRANT SELECT ON [sys].[master_files] TO DBMon_Agent_User;
GRANT execute on sp_helplogins to DBMon_Agent_User;
GRANT execute on sp_readErrorLog to DBMon_Agent_User;
```
use msdb
GRANT SELECT on dbo.sysjobsteps TO DBMon_Agent_User;
GRANT SELECT on dbo.sysjobs TO DBMon_Agent_User;
GRANT SELECT on dbo.sysjobhistory TO DBMon_Agent_User;

where DBMon_Agent_User is the name of the SQL Server user account specified in Create New Collector, Connection Details, Username field.

Note: You can execute the statements above as a batch from a query window in Management Studio.

Object Permissions for Monitoring SQL Server

If you chose not to grant View Server State permissions as described in the Server Level Permissions Required for SQL Server Logon section, then you must grant permissions individually for the following objects in order to monitor SQL Server:
GRANT execute on xp_msver to DBMon_Agent_User;
GRANT SELECT on sys.dm_exec_requests to DBMon_Agent_User;
GRANT SELECT on sys.dm_exec_sessions to DBMon_Agent_User;
GRANT SELECT on sys.dm_os_performance_counters to DBMon_Agent_User;
GRANT SELECT on sys.dm_exec_query_stats to DBMon_Agent_User;
GRANT SELECT on sys.fn_virtualfilestats to DBMon_Agent_User;
GRANT SELECT ON [sys].[master_files] TO DBMon_Agent_User;
GRANT SELECT on sys.configurations to DBMon_Agent_User;
GRANT SELECT on sys.dm_exec_sql_text to DBMon_Agent_User;
GRANT SELECT on sys.sysperfinfo to DBMon_Agent_User;
GRANT SELECT on sys.sysprocesses to DBMon_Agent_User;
GRANT SELECT on sys.syscurconfigs to DBMon_Agent_User;
GRANT SELECT on sys.fn_get_sql to DBMon_Agent_User;
GRANT SELECT on sys.partitions to DBMon_Agent_User;
GRANT SELECT on sys.objects to DBMon_Agent_User;
GRANT SELECT on sys.indexes to DBMon_Agent_User;
GRANT SELECT on sys.tables to DBMon_Agent_User;
GRANT SELECT on sys.dm_db_database_page_allocations to DBMon_Agent_User;
GRANT SELECT on master.sys.dm_exec_procedure_stats to DBMon_Agent_User;
GRANT SELECT on sys.dm_os_ring_buffers to DBMon_Agent_User;
GRANT SELECT on sys.dm_os_sys_memory to DBMon_Agent_User;
GRANT SELECT on sys.master_files to DBMon_Agent_User;
GRANT SELECT on sys.dm_io_virtual_file_stats to DBMon_Agent_User;
GRANT SELECT on sys.dm_exec_query_plan to DBMon_Agent_User;
GRANT SELECT on sys.dm_exec_text_query_plan to DBMon_Agent_User;
GRANT SELECT on sys.syscolumns to DBMon_Agent_User;
GRANT execute on sp_spaceused to DBMon_Agent_User;
GRANT SELECT on sys.sysusers to DBMon_Agent_User;
GRANT SELECT on master.dbo.sysconfigures to DBMon_Agent_User;
GRANT SELECT on msdb.dbo.sysjobhistory to DBMon_Agent_User;
GRANT SELECT on sys.sysdatabases to DBMon_Agent_User;
**Microsoft SQL Server on AWS RDS Permissions**

The user account used for monitoring can be a Windows authenticated account (if the Database Agent is running on Windows) or SQL Server authenticated (if AppDynamics Database Visibility is running on Windows or Linux).

**Minimum Permissions Required for SQL Server Logon**

You can use the procedure below to create a SQL Server user with the minimum permissions required.

Use the following to create a SQL Server logon user that provides the minimal level of permissions required in order to gain full AppDynamics Database Visibility/SQL Server functionality.

1. Using SQL Server Management Studio, create a new login for the AppDynamics SQL Server Database Collector, such as DBMon.Agent_User.
2. From the User Mapping tab, map the new user to the master and msdb databases.

### Viewing Object Information

To view object information on the Database > Objects Browser, map the monitoring user to the databases of interest.

3. Once you have created the login, give the following privileges to the user, substituting DBMon.Agent_User with the name you specified on the Login - New window:

   **Note:** You can execute the following as a batch from a query window in Management Studio. The example shows grants to DBMon.Agent_User; remember to change this if you have set up a different login.

   ```
   use master
   ALTER SERVER ROLE processadmin ADD MEMBER DBMon.Agent_User;
   GRANT VIEW ANY DATABASE TO DBMon.Agent_User;
   GRANT VIEW ANY definition to DBMon.Agent_User;
   GRANT VIEW server state to DBMon.Agent_User;
   ```

   where DBMon.Agent_User is the name of the SQL Server user account specified in Create New Collector, Connection Details, Username field.

   To generate an execution plan on AWS RDS, you need additional permissions. The SHOWPLAN permission must be provided explicitly for each database:

   ```
   USE <DB NAME>
   go
   GRANT SHOWPLAN to DBMon.Agent_User;
   go
   ```
Configure Microsoft Azure Collectors

To monitor Microsoft Azure with Database Visibility, you must be running version 2008 or newer.

To configure an Azure SQL Managed Instance, follow the configuration procedures of a Microsoft SQL Server Collector. See Configure Microsoft SQL Server Collectors.

Procedures in this topic are specific to an Azure SQL Database collector and not applicable for an Azure SQL Managed Instance.

Connection Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>The database type that you want to monitor.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>The Database Agent that manages the collector.</td>
</tr>
<tr>
<td>Name</td>
<td>The name you want to identify the collector by.</td>
</tr>
<tr>
<td>Hostname or IP Address</td>
<td>The hostname or IP address of the machine that your database is running on.</td>
</tr>
<tr>
<td>Database</td>
<td>The name of the database instance that you want to monitor.</td>
</tr>
<tr>
<td>Failover Partner</td>
<td>The hostname or IP address of the failover partner.</td>
</tr>
<tr>
<td>Listener Port</td>
<td>The TCP/IP address of the port on which your database communicates with the Database Agent</td>
</tr>
<tr>
<td>Custom JDBC Connection String</td>
<td>The JDBC connection string generated by the database agent. You can also specify a custom connection string, which is useful for setting custom authentication options.</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Microsoft Azure.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user who is connecting to and monitoring the database through the Database Agent.</td>
</tr>
</tbody>
</table>
**Sub-Collectors**

Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type.

**Note**
- All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub-collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API.
- You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.

**Connection Properties**

Click to add a new JDBC connection property or edit an existing property for relational databases.

**CyberArk**

Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.

**Logging Enabled**

Click to enable verbose mode logging, which logs all communications between the Controller and the Collector. Enable only during troubleshooting because logging consumes a lot of disk space. If you have enabled logging, you can click the logging icon in the Log column of the Collector Administration window to view the log file. The log files are located in the `<db_agent_home>/agent` directory and have the format `<CollectorName>_out.log` and `<CollectorName>_err.log`.

---

**User Permissions for Microsoft Azure**

The user account used for monitoring can be a Windows authenticated account (if the Database Agent is running on Windows) or an SQL Server authenticated account (if AppDynamics Database Visibility is running on Windows or Linux).

**Minimum Permissions Required for SQL Server Logon**

You can create a SQL Server authenticated login or user with minimal level of permissions. To create a SQL Server login, you must be connected to the master database from an admin account via SQL Server Management Studio (SSMS).

1. Run the command given below to create a login. Specify a secure password in the command.

   ```sql
   CREATE LOGIN DBMon_Agent_User WITH PASSWORD = 'Password123'
   ```

2. Run the following command in your Azure SQL database to create a user account for the newly created login:

   ```sql
   CREATE USER DBMon_Agent_User FOR LOGIN DBMon_Agent_User WITH DEFAULT_SCHEMA = dbo
   ```
3. While connected to your Azure SQL database, run the command given below to grant the pre-requisite roles and privileges:

```
grant VIEW DATABASE STATE to DBMon_Agent_User
```
Configure MongoDB Collectors

On this page:
- Connection Details
- User Permissions for MongoDB

To monitor MongoDB with Database Visibility, you must be running version 2.2 or newer.

If you are configuring a collector for a MongoDB cluster, you only need to configure one collector for the entire cluster. You can choose any node in the cluster to connect to, and the entire cluster will automatically be detected.

Connection Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>The database type that you want to monitor.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>The Database Agent that manages the collector.</td>
</tr>
<tr>
<td>Name</td>
<td>The name you want to identify the collector by.</td>
</tr>
<tr>
<td>Hostname or IP Address</td>
<td>The hostname or IP address of the machine that your database is running on.</td>
</tr>
<tr>
<td>Listener Port</td>
<td>The TCP/IP address of the port on which your database communicates with the Database Agent</td>
</tr>
<tr>
<td>Custom Connection String</td>
<td>The connection string generated by the database agent. You can also specify a custom connection string, which is useful for setting custom authentication options.</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for MongoDB.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user who is connecting to and monitoring the database through the Database Agent.</td>
</tr>
<tr>
<td>CyberArk</td>
<td>Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.</td>
</tr>
<tr>
<td>Exclude Databases</td>
<td>The databases that you want to exclude, separated by commas.</td>
</tr>
<tr>
<td>Logging Enabled</td>
<td>Click to enable verbose mode logging, which logs all communications between the Controller and the Collector. Enable only during troubleshooting because logging consumes a lot of disk space. If you have enabled logging, you can click the logging icon in the Log column of the Collector Administration window to view the log file. The log files are located in the &lt;db_agent_home&gt;\agent directory and have the format &lt;CollectorName&gt;_out.log and &lt;CollectorName&gt;_err.log.</td>
</tr>
</tbody>
</table>

User Permissions for MongoDB

For MongoDB versions prior to 2.6.X, the readAnyDatabase and ClusterMonitor built-in roles are required in order to monitor using AppDynamics Database Visibility. For MongoDB sharded clusters, the monitoring user must have access to all shards.

For MongoDB 2.6 and later, the new clusterMonitor built-in role in addition to readAnyDatabase is required.
If you choose to create a new user to monitor MongoDB, the user must be created in the admin database.

You can configure user roles as shown in the sample query below:

```javascript
use admin
db.createUser({ user: "tanujaAdmin",
pwd: "tanuja123",
  roles: [
  { role: "clusterMonitor", db: "admin" },
  { role: "readAnyDatabase", db: "admin" },
  ]
})
```
Configure MySQL Collectors

On this page:
- Connection Details
- User Permissions for MySQL

To monitor MySQL with Database Visibility, you must be running version 2.2 or newer.

Connection Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>The database type that you want to monitor.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>The Database Agent that manages the collector.</td>
</tr>
<tr>
<td>Name</td>
<td>The name you want to identify the collector by.</td>
</tr>
<tr>
<td>Hostname or IP Address</td>
<td>The hostname or IP address of the machine that your database is running on.</td>
</tr>
<tr>
<td>Listener Port</td>
<td>The TCP/IP address of the port on which your database communicates with the Database Agent</td>
</tr>
<tr>
<td>Custom JDBC Connection String</td>
<td>The JDBC connection string generated by the database agent, for example, jdbc:mysql://. You can also specify a custom connection string, which is useful for setting custom authentication options.</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for MySQL.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user who is connecting to and monitoring the database through the Database Agent.</td>
</tr>
<tr>
<td>Sub-Collectors</td>
<td>Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type.</td>
</tr>
</tbody>
</table>

Note
- All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub-collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API.
- You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.

Connection Properties
- Click to add a new JDBC connection property or edit an existing property for relational databases.

CyberArk
- Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.
<table>
<thead>
<tr>
<th>Exclude Databases</th>
<th>The databases that you want to exclude, separated by commas.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging Enabled</td>
<td>Click to enable verbose mode logging, which logs all communications between the Controller and the Collector. Enable only during troubleshooting because logging consumes a lot of disk space. If you have enabled logging, you can click the logging icon in the Log column of the Collector Administration window to view the log file. The log files are located in the &lt;db_agent_home&gt;/agent directory and have the format &lt;CollectorName&gt;_out.log and &lt;CollectorName&gt;_err.log.</td>
</tr>
</tbody>
</table>

**User Permissions for MySQL**

The MySQL user the Database Agent uses to monitor the MySQL database must have "SELECT", "PROCESS", and "SHOW DATABASES" privileges on all databases.

The user must also have the "REPLICATION CLIENT" privilege to collect the following metrics:

- Slave_io_running
- Slave_sql_running
- Seconds_behind_master
- SQL_Delay

If you do not have a suitable existing user, you can use the command below to create a new user.

```
GRANT SELECT,PROCESS,SHOW DATABASES on *.* to 'DBMon_Agent_User'@'host' identified by 'password';
GRANT REPLICATION CLIENT ON *.* to 'DBMon_Agent_User'@'host';
FLUSH privileges;
```

Substitute `DBMon_Agent_User` with the username under which you run the Database Visibility Agent. Substitute `host` with the hostname or IP address of the machine running the AppDynamics Database Agent, and substitute `password` with a secure password.

Set the `max_allowed_packet` parameter to 1073741824 on the server.
Troubleshooting MySQL Permissions

Viewing MySQL Error Logs

To see the error log file, the following conditions must be met:

- `secure_file_priv` is either empty or the name of the folder containing the error log file. You must restart MySQL for changes in the `secure_file_priv` variable to take effect.
- The error log file is readable to all users
- `max_allowed_packet` is larger than the error log file size
- The user has FILE privileges
Configure Oracle Collectors

On this page:
- Connection Details
- User Permissions for Oracle

To monitor Oracle with Database Visibility, you must be running Oracle 10g or newer.

If you are configuring a collector for Oracle RAC, you only need to configure one collector for the entire cluster. You can choose any node in the cluster to connect to, and the entire cluster will automatically be detected.

Connection Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>The database type that you want to monitor.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>The Database Agent that manages the collector.</td>
</tr>
<tr>
<td>Name</td>
<td>The name you want to identify the collector by.</td>
</tr>
<tr>
<td>Hostname or IP Address</td>
<td>The hostname or IP address of the machine that your database is running on.</td>
</tr>
<tr>
<td>Listener Port</td>
<td>The TCP/IP address of the port on which your database communicates with the Database Agent.</td>
</tr>
<tr>
<td>Custom JDBC Connection String</td>
<td>The JDBC connection string generated by the database agent, for example, jdbc:oracle:thin:@(DESCRIPTION=... You can also specify a custom connection string, which is useful for setting custom authentication options.</td>
</tr>
<tr>
<td>Use Service Name</td>
<td>Select if the field that follows is the service name.</td>
</tr>
<tr>
<td>SID or SERVICE_NAME</td>
<td>The SID or service name of the database you want to monitor.</td>
</tr>
<tr>
<td>Connect as a sysdba</td>
<td>Click if you want to connect to the database as a sysdba user and enable password files. Running the collector with a sysdba account allows the collector to monitor an Oracle instance that is on standby.</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Oracle.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user who is connecting to and monitoring the database through the Database Agent.</td>
</tr>
<tr>
<td>Sub-Collectors</td>
<td>Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type.</td>
</tr>
</tbody>
</table>

**Note**
- All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub-collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API.
- You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.
## Connection Properties

Click to add a new JDBC connection property or edit an existing property for relational databases.

## CyberArk

Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.

## Exclude Schemas

The schemas that you want to exclude, separated by commas.

## Oracle Wallet

Click to enable Oracle Wallet.

- Truststore Location: location of the e-wallet file on the DB Agent host
- Truststore Type: type of Oracle Wallet. There are two truststore types:
  - PKCS12 (default)
  - SSO: enables auto-login. If you use SSO, you only need to provide the truststore location and truststore type.
- Truststore Password: password for Oracle Wallet

If the SSL_CLIENT_AUTHENTICATION parameter is set to true in listener.ora and sqlnet.ora, then click the Enable SSL Client Authentication box.

- Keystore Location: location of the e-wallet file on the DB Agent host
- Keystore Type: type of Oracle Wallet
- Keystore Password: password for Oracle Wallet

## Logging Enabled

Click to enable verbose mode logging, which logs all communications between the Controller and the Collector. Enable only during troubleshooting because logging consumes a lot of disk space. If you have enabled logging, you can click the logging icon in the Log column of the Collector Administration window to view the log file. The log files are located in the `<db_agent_home>/agent` directory and have the format `<CollectorName>_out.log` and `<CollectorName>_err.log`.

## User Permissions for Oracle

### For versions of Oracle 10g and later

The following permissions are required for the Oracle user:

- CREATE SESSION
- SELECT_CATALOG_ROLE

To create a user with these permissions, you can run the following SQL. In this SQL, change "password" to a safe and secure password, and change the tablespace names, "users" and "temp" to those available in your Oracle instance.

```
CREATE USER DBMon_Agent_User IDENTIFIED BY password
default tablespace users
temporary tablespace temp;

GRANT CREATE SESSION, SELECT_CATALOG_ROLE TO DBMon_Agent_User;
```

where DBMon_Agent_User is the user name under which you run the Database Visibility Agent.

### For versions of Oracle 12c and later with Multitenant Container Database Option enabled

---

Copyright © AppDynamics 2012-2019

Page 74
The following permissions are required for the Oracle user:

- CREATE SESSION
- SELECT_CATALOG_ROLE

To create a user with these permissions, you can run the following SQL. In this SQL, change "password" to a safe and secure password, and change the tablespace names, "users" and "temp" to those available in your Oracle instance.

```
CREATE USER C##DBMon_Agent_User IDENTIFIED BY password default tablespace users temporary tablespace temp CONTAINER=ALL;
GRANT CREATE SESSION, SELECT_CATALOG_ROLE TO C##DBMon_Agent_User CONTAINER=ALL;
ALTER USER C##DBMon_Agent_User QUOTA 100M ON USERS;
alter user C##DBMON_AGENT_USER set container_data=all container=current;
```

Permissions Required for Oracle Explain Plans

To generate an execution plan for a specified query, paste the query text into the Explain another query box. You must provide a plan table in the AppDynamics monitoring user's schema. You can find a sample output table named PLAN_TABLE in the UTLXPLAN.SQL script. To update or view the plan table, you need INSERT and SELECT privileges. To explain a query, you need access privileges for the tables and views included in that query.
AppDynamics Database Visibility can generate explain plans within its SQL drill-down window. To enable this functionality, you must have a plan table accessible to the AppDynamics Database Visibility schema user. You can create this plan table with the following command from sqlplus when logged on as the AppDynamics Database Visibility user:

Windows:

```
@?\rdbms\admin\utlxplan.sql
```

Linux:

```
@?/rdbms/admin/utlxplan.sql
```

Permissions for individual views and tables

You can grant permission to individual views and tables even if the SELECT_CATALOG_ROLE permission is not available to your organization.

1. Execute permissions on dbms_application_info.set_module.
2. Select permissions on the following privileges:
   - dba_data_files
   - dba_ind_columns
   - dba_indexes
   - dba_objects
   - dba_segments
   - dba_sequences
   - dba_synonyms
   - dba_tab_columns
   - dba_tables
   - dba_users
   - dba_views
   - gv$instance
   - v$archive_dest
   - v$archive_dest_status
   - v$database
   - v$datafile
   - v$event_name
   - v$instance
   - v$license
   - v$log
   - v$parameter
   - v$process
   - v$session
   - v$sesstat
   - v$session_wait
   - v$sga
   - v$sql
   - v$sql_plan
   - v$sqlstats
   - v$sqltext
   - v$statname
v$sysmetric
v$sysstat
v$system_event

When you are granting access to v$ views, you must grant SELECT on the underlying objects, which are named using the v_ format, e.g. GRANT SELECT on v_$archive_dest.
Configure Oracle RAC

Related pages:
- Database Topology Window

If you are configuring a collector for Oracle RAC, you only need to configure one collector for the entire cluster. You can choose any node in the cluster to connect to, and the entire cluster is automatically detected.

If you have sub-collectors, it will disable automatic cluster discovery.

When the Database Visibility collector connects to the hostname or IP address defined in the collector configuration, it auto-discovers the RAC instance. Once a successful connection is made to a node within the RAC, then a query on GV$INSTANCE will be used to return the details of the other member nodes and to populate the Topology Window.

When monitoring Oracle RAC, the Controller displays the aggregate data for the entire cluster.

User Permissions for Oracle RAC

When you upgrade to 4.3, you may encounter the following error: Network access denied by access control list (ACL). You can resolve the error by enabling ACLs for the monitoring user.

```sql
begin
  dbms_network_acl_admin.create_acl (acl => 'Resolve_Access.xml',
                                    description => 'Resolve Network Access using UTL_INADDR',
                                    principal => '<DBMON USER>',
                                    is_grant => TRUE,
                                    privilege => 'resolve',
                                    start_date => null,
                                    end_date => null);
  commit;
end;
/

begin
  dbms_network_acl_admin.assign_acl (acl => 'Resolve_Access.xml',
                                     host => '*',
                                     lower_port => null,
                                     upper_port => null);
  commit;
end;
/
```
Configure PostgreSQL Collectors

On this page:

- Connection Details
- User Permissions for PostgreSQL

You can monitor any version of PostgreSQL with Database Visibility.

Connection Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>The database type that you want to monitor.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>The Database Agent that manages the collector.</td>
</tr>
<tr>
<td>Name</td>
<td>The name you want to identify the collector by.</td>
</tr>
<tr>
<td>Hostname or IP Address</td>
<td>The hostname or IP address of the machine that your database is running on.</td>
</tr>
<tr>
<td>EnterpriseDB</td>
<td>Click if your PostgreSQL database installation is an EnterpriseDB distribution.</td>
</tr>
<tr>
<td>Listener Port</td>
<td>The TCP/IP address of the port on which your database communicates with the Database Agent</td>
</tr>
<tr>
<td>Custom JDBC Connection String</td>
<td>The JDBC connection string generated by the database agent, for example, jdbc:postgresql://. You can also specify a custom connection string, which is useful for setting custom authentication options.</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for PostgreSQL.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user who is connecting to and monitoring the database through the Database Agent.</td>
</tr>
<tr>
<td>Sub-Collectors</td>
<td>Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type.</td>
</tr>
</tbody>
</table>
| Note                      | - All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub–collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API.  
- You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data. |
| Connection Properties     | Click to add a new JDBC connection property or edit an existing property for relational databases. |
Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk website and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.

The databases that you want to exclude, separated by commas.

Click to enable verbose mode logging, which logs all communications between the Controller and the Collector. Enable only during troubleshooting because logging consumes a lot of disk space. If you have enabled logging, you can click the logging icon in the Log column of the Collector Administration window to view the log file. The log files are located in the <db_agent_home>/agent directory and have the format <CollectorName>_out.log and <CollectorName>_err.log.

**User Permissions for PostgreSQL**

The monitoring user must either have a role of superuser or be granted access to the pg_stat_activity table using the technique described below.

The following script allows you to create a SECURITY DEFINER function that is owned by the superuser, and runs the query you want, thus allowing non-superusers to see the contents of pg_stat_activity by calling the get_sa() function.

```
CREATE FUNCTION get_sa() RETURNS SETOF pg_stat_activity AS
$$ SELECT * FROM pg_catalog.pg_stat_activity; $$
LANGUAGE sql
VOLATILE
SECURITY DEFINER;

CREATE VIEW pg_stat_activity_allusers AS SELECT * FROM get_sa();

GRANT SELECT ON pg_stat_activity_allusers TO public;
```

The monitoring user must also be able to connect remotely to the PostgreSQL instance from the AppDynamics for Databases machine.
Configure Sybase Collectors

On this page:
- Connection Details
- User Permissions for Sybase

To monitor Sybase with Database Visibility, you must be running version 15 or newer.

Connection Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>The database type that you want to monitor.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>The Database Agent that manages the collector.</td>
</tr>
<tr>
<td>Name</td>
<td>The name you want to identify the collector by.</td>
</tr>
<tr>
<td>Hostname or IP Address</td>
<td>The hostname or IP address of the machine that your database is running on.</td>
</tr>
<tr>
<td>Listener Port</td>
<td>The TCP/IP address of the port on which your database communicates with the Database Agent</td>
</tr>
<tr>
<td>Custom JDBC Connection String</td>
<td>The JDBC connection string generated by the database agent, for example, jdbc:sybase:Tds:. You can also specify a custom connection string, which is useful for setting custom authentication options.</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Sybase.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user who is connecting to and monitoring the database through the Database Agent.</td>
</tr>
</tbody>
</table>

Sub-Collectors

Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type.

Note
- All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub–collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API.
- You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.

Connection Properties

Click to add a new JDBC connection property or edit an existing property for relational databases.

CyberArk

Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.
Exclude Databases | The databases that you want to exclude, separated by commas.
---|---
Logging Enabled | Click to enable verbose mode logging, which logs all communications between the Controller and the Collector. Enable only during troubleshooting because logging consumes a lot of disk space. If you have enabled logging, you can click the logging icon in the Log column of the Collector Administration window to view the log file. The log files are located in the `<db_agent_home>/agent` directory and have the format `<CollectorName>_out.log` and `<CollectorName>_err.log`.

User Permissions for Sybase

For complete AppDynamics Database Visibility functionality, the monitoring user requires the permissions listed in the table below.

<table>
<thead>
<tr>
<th>Permission type</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role permission</td>
<td>• sa_role</td>
</tr>
<tr>
<td></td>
<td>• mon_role</td>
</tr>
<tr>
<td>Select permission</td>
<td>• master.dbo.monWaitEventInfo</td>
</tr>
<tr>
<td></td>
<td>• master.dbo.sysconfigures</td>
</tr>
<tr>
<td></td>
<td>• master.dbo.sysmonitors</td>
</tr>
<tr>
<td></td>
<td>• master.dbo.monProcess</td>
</tr>
<tr>
<td></td>
<td>• master.dbo.monProcessLookup</td>
</tr>
<tr>
<td></td>
<td>• master.dbo.monProcessSQLText</td>
</tr>
<tr>
<td></td>
<td>• master.dbo.monProcessProcedures</td>
</tr>
<tr>
<td>Execute permission</td>
<td>• sp_sysmon</td>
</tr>
</tbody>
</table>

To create a new dedicated user for AppDynamics Database Visibility, you can use following sample user creation script. Before running the script, change "password" to a more secure value.

```sql
exec  sp_addlogin  'DBMon_Agent_User', 'password', @defdb='master',
@deflanguage='us_english', @fullname='DBMon_Agent_User monitoring account', @auth_mech = 'ANY'
go
exec  sp_locklogin  'DBMon_Agent_User', 'unlock'
go
exec  sp_role 'grant', 'mon_role', 'DBMon_Agent_User'
go
```

where `DBMon_Agent_User` is the user name under which you run the Database Visibility Agent.

Also, the following configuration parameters must be set to 1 (true) in order to monitor the Sybase ASE database with AppDynamics Database Visibility: "enable monitoring", "wait event timing", "SQL batch capture", and "object lockwait timing". You should also set "max SQL text monitored" to at least 8192 (8kB).

Here is an example of the commands required to configure these settings:
If the value for “max SQL text monitored” was previously less than 4096, then increasing this setting will require that you restart the Sybase ASE instance.

To monitor Sybase version 15.7 and later without the sa_role permission, run the following commands:

```sql
use sybsystemprocs
grant execute on sp_sysmon to mon_role
```

If you choose to monitor Sybase using sp_sysmon, you may encounter the following errors:

- Thread utilization is incorrectly reported. View the official report [here](#).
- Timeslice error in mmap64 or mda_flush_iostats. View the official report [here](#).
Configure Sybase IQ Collectors

You can monitor any version of Sybase IQ with Database Visibility.

Connection Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>The database type that you want to monitor.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>The Database Agent that manages the collector.</td>
</tr>
<tr>
<td>Name</td>
<td>The name you want to identify the collector by.</td>
</tr>
<tr>
<td>Hostname or IP Address</td>
<td>The hostname or IP address of the machine that your database is running on.</td>
</tr>
<tr>
<td>Listener Port</td>
<td>The TCP/IP address of the port on which your database communicates with the Database Agent</td>
</tr>
<tr>
<td>Custom JDBC Connection String</td>
<td>The JDBC connection string generated by the database agent. You can also specify a custom connection string, which is useful for setting custom authentication options.</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Sybase IQ.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user who is connecting to and monitoring the database through the Database Agent.</td>
</tr>
<tr>
<td>Sub-Collectors</td>
<td>Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type.</td>
</tr>
</tbody>
</table>

Note

- All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub-collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API.
- You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.

Connection Properties

Click to add a new JDBC connection property or edit an existing property for relational databases.

CyberArk

Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.
Logging Enabled

Click to enable verbose mode logging, which logs all communications between the Controller and the Collector. Enable only during troubleshooting because logging consumes a lot of disk space. If you have enabled logging, you can click the logging icon in the Log column of the Collector Administration window to view the log file. The log files are located in the `<db_agent_home>/agent` directory and have the format `<CollectorName>_out.log` and `<CollectorName>_err.log`.

User Permissions for Sybase IQ

To monitor with a non-DBA account, you need the following:

- User account with SERVER OPERATOR system privilege.
- Execution privileges on several procedures which can be granted as follows:

```sql
grant execute on sp_iqconnection to DBMon_Agent_User
grant execute on sp_iqtransaction to DBMon_Agent_User
grant execute on sp_iqcontext to DBMon_Agent_User
grant execute on sp_iqstatus to DBMon_Agent_User
grant execute on sp_iqcolumn to DBMon_Agent_User
grant execute on sp_iqindex to DBMon_Agent_User
```

where `DBMon_Agent_User` is the user name under which you run the Database Visibility Agent.
- Grant DBA authority to `DBMon_Agent_User` as follow:

```sql
grant DBA to DBMon_Agent_User
```
Configure Couchbase Collectors

To monitor Couchbase with Database Visibility, you must be running Couchbase 4.5 and higher. And the version of the Database Agent should be higher than 4.5 to monitor Couchbase databases.

Connection Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>The database type that you want to monitor.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>The Database Agent that manages the collector.</td>
</tr>
<tr>
<td>Name</td>
<td>The name you want to identify the collector by.</td>
</tr>
<tr>
<td>Hostname or IP Address</td>
<td>The hostname or IP address of the machine that your database is running on.</td>
</tr>
<tr>
<td>Database</td>
<td>The name of the database instance that you want to monitor.</td>
</tr>
<tr>
<td>Listener Port</td>
<td>The TCP/IP address of the port on which your database communicates with the Database Agent</td>
</tr>
<tr>
<td>Custom JDBC Connection String</td>
<td>The JDBC connection string generated by the database agent, for example, jdbc:db2://. You can also specify a custom connection string, which is useful for setting custom authentication options.</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Couchbase.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user who is connecting to and monitoring the database through the Database Agent.</td>
</tr>
<tr>
<td>Logging Enabled</td>
<td>Click to enable verbose mode logging, which logs all communications between the Controller and the Collector. Enable only during troubleshooting because logging consumes a lot of disk space. If you have enabled logging, you can click the logging icon in the Log column of the Collector Administration window to view the log file. The log files are located in the &lt;db_agent_home&gt;\agent directory and have the format &lt;CollectorName&gt;_out.log and &lt;CollectorName&gt;_err.log.</td>
</tr>
</tbody>
</table>

User Permissions for Couchbase

The monitoring user must have Full Administrator or Read-only Administrator privileges. Alternatively, if you do not want the monitoring user to have Administrator privileges, you can assign a combination of Data Monitoring and Query System Catalog privileges (available for Couchbase versions 5 and later).
**Monitor Sybase Databases Using Kerberos Authentication**

In order for the Database Agent to connect to and monitor Sybase databases using Kerberos authentication, the following are required:

- In the Connection Details section of the Collector configuration dialog, specify the following JDBC connection string:

```
```

- Start the Database Agent using the `-Djavax.security.auth.useSubjectCredsOnly=false` option such as the following:

```
nohup java -Djavax.security.auth.useSubjectCredsOnly=false /<db_agent_home>/db-agent.jar &
```
Configure the Database Agent to Monitor Server Hardware

On this page:
- Complete the Hardware Monitoring Section
- Resolve Metric Reporter Type Mismatch Problem

Related pages:
- Add Database Collectors
- Monitor Database Server Hardware

In addition to monitoring the database server, the Database Agent can also monitor the hardware that hosts the database server. To configure the Database Agent to monitor the database server hardware, complete the Hardware Monitoring section of the Collector configuration dialog.

Click Databases > Configuration > Collectors and then choose an existing collector to monitor the hardware for that database server and click Edit, or click Add to setup a new collector to monitor a different database server and the hardware hosting that server.

Complete the Hardware Monitoring Section

The following describes the fields in the Collector Configuration dialog that you must complete if you want to monitor the database server host hardware, in addition to the database server.

Monitor Operating System: Select if you want CPU consumption metrics collected from the monitored host.

Operating System: Specify the operating system of the monitored host: Windows, Linux, Solaris, AIX, or RDS.

Use Local WMI: Check this box if you want to monitor the machine the Database agent is running on, i.e. localhost. When selected, the authorization fields are not used because no authorization is required.

Domain: For Windows only, specify the name of the domain in which the hardware resides.

SSH Port: For Linux, AIX, and Solaris only, specify the Secure Shell (SSH) port number the Controller should use for encrypted communications with the monitored host. The default port number of 22 will be used if you do not specify a different port number here.

Use certificate: For Linux, AIX, and Solaris only, AppDynamics also supports certificate-based authentication via Privacy Enhanced Mail (PEM). To implement certificate-based authentication, enable the Use certificate option and copy the PEM file to the <db_agent_home>/keys directory. Note, if the $HOME/.ssh directory exists, the agent will use the certificate found there. This option appears only if the agent is running on a machine running Linux, AIX or Solaris.

Username: Specify the name of the user the Database Agent uses to log on to the monitored host. To collect OS metrics from a Windows host, the configured user (or Collector Service user if using Windows Authentication) must be able to establish a WMI connection to the target host and collect Windows Performance Counters.

Password: Specify the password of the user the Database Agent uses to log on to the monitored host. The number of echo characters shown in the password text field should not be interpreted to imply the number of characters stored for the (encrypted) user password.

Enabling Hardware Monitoring is unnecessary for SQL Server. By default, Database Visibility collects most of the hardware metrics that you will need.

Resolve Metric Reporter Type Mismatch Problem

Switch from the Java Hardware Monitor to the Hardware Monitor if you see messages similar to the following in the logs:
com.singularity.ee.agent.commonservices.metricgeneration.metrics.MetricReporterTypeMismatchException: Aggregator of OBSERVATION already exists for metric Metric Identifier[Hardware Resources|Network|Incoming packets/sec] ID[0]
Required Monitored Host Permissions

On this page:

- Required SSH Permissions

Related pages:

- Configure a Controller SSH Key

Database Agent User Permissions

Database Visibility collects the stats from common commands like vmstat/iostat or gathering metrics from the file system such as /proc and as such the user that runs the Database Agent requires no special permissions, just the ability to run those common commands and write to files in the Database Agent directory.

Database Agent Collector Authentication

When monitoring the database host, Database Visibility must log in to the host system to gather system information. There are three ways to authenticate the Database Collector to access the monitored host:

- Specify a username and password in the Collector configuration dialog
- Place a PEM file or an id_rsa file in the <agent home>/keys directory
- If the Database Agent is running on LINUX, Solaris or AIX, place SSH keys in the <home>.ssh directory of the user running the agent. You can create an SSH key using the same procedure as described for the Controller, Configure a Controller SSH Key.

Required SSH Permissions

When monitoring the database host, Database Visibility uses SSH on Linux and Windows systems to gather system information. SSH access can be through either an authenticated username/password or a private key.

You can create an SSH key using the same procedure as described for the Controller, Configure a Controller SSH Key.
Configure WMI Permissions and Security

On this page:

- Requirements to Monitor Windows 7 and Higher Systems (agent running on Unix-like platform)
- Additional Requirements to Monitor Windows 2012 and Higher Systems (agent running on Unix-like platform)
- General Considerations for all Platforms

Related pages:

- WMI and the Database Agent on the AppDynamics Community

To monitor Windows-based machine hardware with AppDynamics Database Visibility, AppDynamics uses Windows Management Instrumentation (WMI) to remotely gather the metrics. WMI is often complicated to troubleshoot when the Database Agent is running on a Linux or Unix-like machine. This topic identifies requirements for the target machine configuration that can help you avoid some problems and pitfalls. It also provides some additional considerations regarding using WMI to monitor a SQL Server database agent and preventing unauthorized remote access to WMI.

The following are required when the Database Agent is hosted on AIX, Linux or Solaris platforms to monitor Windows 7 and higher systems.

- Ensure that the named Windows account is a member of the local Administrators group.
- Ensure User Account Meets Minimum Security Requirements When Using WMI
- Enable Remote Registry Access
- Grant access to WBEM scripting locator

The following are required when the Database Agent is hosted on AIX, Linux or Solaris platforms to monitor Windows 2012 and higher systems.

- Grant full control permissions to select registry keys

The following is required when the Database Agent is hosted on Windows.

- Ensure that the named Windows account is a member of the local Administrators group.

Requirements to Monitor Windows 7 and Higher Systems (agent running on Unix-like platform)

The following are required when the Database Agent is hosted on AIX, Linux or Solaris platforms to monitor Windows 7 and higher systems.

Ensure User Account Meets Minimum Security Requirements When Using WMI

Enable Security Options for Windows Systems that are part of a Domain

Ensure the named Windows account has the correct permissions for WMI Control.

1. Run the wmic msc program.
2. Right-click the WMI Control icon on the left and click Properties.
3. Click the Security tab.
4. Click the root node of the tree, and click Security.
5. Ensure that the named user account running the Database Agent has the relevant permissions.
   - The minimum permissions that your remote Windows account needs for the Database Agent are:
     - Execute Methods
     - Enable Account
     - Remote Enable

If the named Windows account does not have all of these permissions, you might see an access denied error or the following
error:

Error=800706BA The RPC server is unavailable. SWbemLocator

or

Error=80070005 Access is denied SWbemLocator

Enable Classic Security Options for Local (non-domain) Windows Systems

Applies to Windows computers that are not part of a domain.

1. Open the Control panel, and go to Administrative Tools > Local Security Policy. The Local Security Settings window appears.
2. Go to Local Policies > Security Options.
3. Change the value of “Network access: Sharing and security model for local accounts.” to Classic.

Enable Remote Registry Access

The Remote Registry service must be running on the target machine. If the Remote register service is off, you will see the following error:

Message not found for errorCode: 0xC0000034

or

Access is denied

By default Windows 7 and above systems will still deny remote access to the registry, even if the Remote Registry service is started.

To test this, try to connect to the slave registry via regedit on another machine. If you get a error similar to Access is denied, run powershell as an administrator on the slave, and execute Enable-PSRemoting. Restart the machine and try launching the slave again.

Grant Access to WBEM Scripting Locator

The Database Agent requires full access to the WBEM Scripting Locator. On the target system allow full access to the WBEM Scripting Locator as follows:

As an Administrator on the target machine, launch regedit.
2. Locate the registry key: 76A64158-CB41-11D1-8B02-00600806D9B6 in HKEY_CLASSES_ROOT\CLSID
3. Right-click the key and click Permissions.
4. Click Advanced, and then on the Owner tab change the owner to the Administrators group. Click Apply.
5. On the Permissions tab change the permissions for the Administrators group to Full Control. Click Apply.

Configure the Firewall

WMI uses RPC which listens on port 135 but then allocates a dynamic port for subsequent communication. Configure your Firewall to always allow the TCP port 135 exception and follow the dynamic RPC ports. If there is a problem with the firewall, port 135 then you will probably see this error:

ERROR: Message not found for errorCode: 0xC0000001

For more information, see How to configure RPC dynamic port allocation to work with firewalls.

Additional Requirements to Monitor Windows 2012 and Higher Systems (agent running on Unix-like platform)

In addition to the requirements described in Requirements to Monitor Windows 7 and Higher Systems, the following are also required when the Database Agent is hosted on AIX, Linux or Solaris platforms to monitor Windows 2012 and higher systems.

Grant Full Control Permissions to Select Registry Keys

For the Database Agent running on AIX, Linux or Solaris to monitor Windows 2012 (64-bit) and above systems, complete the following changes on the target system.
1. As an Administrator on the target machine, launch regedit.
2. Change the permissions for both of the following registry keys to **Full Control**:
   72C24DD5-D70A-438B-8A42-98424B88AFB8 in HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Wow6432Node\CLSID
   76A64158-CB41-11D1-8B02-00600806D9B6 in HKEY_CLASSES_ROOT\CLSID
3. Find the following registry key:
   72C24DD5-D70A-438B-8A42-98424B88AFB8 in HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Wow6432Node\CLSID
4. Right click and click **Permissions**.
5. Change the owner to the Administrators group.
6. Change the permissions for the Administrators group to **Full Control**.
7. Change owner back to TrustedInstaller. User is "NT Service\Trusted Installer" on the local machine.
8. Repeat steps 4 to 6 above for the following registry key:
   76A64158-CB41-11D1-8B02-00600806D9B6 in HKEY_CLASSES_ROOT\CLSID.
10. Restart the Remote Registry service, using **Administrative Tools > Services**.

**General Considerations for all Platforms**

These topics apply to the Database Agent running on Windows systems.

**Use Windows Authentication for Microsoft SQL Server**

To use Windows Authentication for the Database Agent to connect to a Microsoft SQL Server database instance, you must use a command similar to following to start the Database Agent; specifying the path to the Database Agent authentication library.

**Windows 64-bit**

```java
java -Djava.library.path="C:\dbagent404\auth\x64" -jar db-agent.jar
```

**Windows 32-bit**

```java
java -Djava.library.path="C:\dbagent404\auth\x86" -jar db-agent.jar
```

Also, the Windows account used to start the Database Agent must be a Windows user who can authenticate with the database server.
Monitor Databases and Database Servers

The topics in this section show you how to use the features of AppDynamics to monitor and troubleshoot your database environments.

- Access Database Visibility from Application Monitoring Views
- Discover Normal Database and Server Activity
- Monitor Database Performance
- Monitor Database Server Hardware
- Database Health Rules and Alerts
- Database Monitoring Metrics
- Database Agent Events Reference
- Wait State Filtering
- Configuring Custom Metrics
- Configure Query Literals Security

For roles and permissions required to view Database Monitoring windows, see Roles and Permissions.
Access Database Visibility from Application Monitoring Views

On this page:
- View Database Activity in Database Visibility
- View Business Transaction Snapshot Correlated Database Details

Related pages:
- Monitor Database Performance
- App Agent Node Properties

View Database Activity in Database Visibility

Once you have configured a database Collector in AppDynamics Monitoring, you can access Database Visibility either from the AppDynamics Home page or you can link to the Database Visibility database instance Dashboard by right-clicking the database and selecting View in Database Visibility on the Application Flow Map, Tier Flow Map or Node Flow Map.

Database backends with the same hostname, port number, and database type as a database server already configured in a database Collector are automatically matched with the Collector, and drill-downs from the Application Flow Map, Tier Flow Map or Node Flow Map to Database Visibility are enabled.
View Business Transaction Snapshot Correlated Database Details

Available for Java, .NET, and PHP applications and their relational database backends, snapshot correlation shows the details of queries executed by the business transaction. It also shows the clients, sessions, and schemas in which those queries were executed.
View Database Details

1. On the Business Transactions page, double click on any business transaction to view the queries that are executed in that transaction.
2. Click the DB Queries tab. On this tab, you can view the queries that were issued by this business transaction.

View Correlated Database Details

1. From a list of business transactions with snapshots, choose a longer running business transaction that accesses the database, one that takes a few seconds or more. These are more likely to have captured database details.
2. Double-click Drill Down above the database icon. The correlated database details window appears.
3. Click the Queries, Clients, Sessions, and Schema tabs to view details of database activity that occurred around the time when the snapshot was captured. For relational databases, these details reflect the database activity from the transaction snapshot. For non-relational databases, these details reflect the database activity that occurred around the time of the transaction snapshot.

For Oracle backends only: If you want to list only the queries executed by the Transaction Snapshot in the Transaction Snapshot window, you can enable the jdbc-dbcam-integration-enabled node property for the Java Agent.
Discover Normal Database and Server Activity

On this page:
- View Performance Metrics on the Metric Browser

Related pages:
- Dynamic Baselines
- Health Rules

AppDynamics Database Visibility automatically learns to detect performance anomalies using baselines that are specific to your database and database server environments.

AppDynamics creates baselines by collecting metrics from your monitored databases and servers over defined periods of time. This establishes what is normal for your environment. You can also create your own baselines.

View Performance Metrics on the Metric Browser

On the Metric Browser for Database Visibility, you can visualize performance metrics and see how they deviate from expected behaviors established by the baseline.
Monitor Database Performance

Related pages:

- Access Database Visibility from Application Monitoring Views
- Monitor Databases and Database Servers

The subtabs for each database provide detailed information about your database. This section describes the various windows for the database instance and how to interpret the displayed information.

From most of these subtabs you can drill down to the Queries view.

- View Overall Database and Server Performance
- Database Dashboard
- Database Activity Window
- Database Topology Window
- Database Live View Window
- Database Queries Window
- Database Clients Window
- Database Sessions Window
- Database Blocking Sessions Window
- Database Schemas and Databases Windows
- Database Modules Window
- Database Programs Window
- Database Users Window
- Database Buckets Window
- Database Business Transactions Window
- Database Applications Window
- Database Object Browser Window
- Database Custom Metrics Window

Important information
Garbage collection affects all monitored entities, including database queries, sessions, clients, and schemas. Entities that are garbage collected will be displayed as unavailable.
View Overall Database and Server Performance

On this page:
- Access the Databases Window
- Features of the Databases Window

Related pages:
- Access Database Visibility from Application Monitoring Views

Database Visibility provides overall views of your databases that show key performance indicators.

Access the Databases Window

From AppDynamics Home, click Databases.

Features of the Databases Window

On the views of the Databases window you can:

- Show only databases meeting certain criteria by entering search criteria in the search box on the top right of the page.
- Click Filters to show only databases meeting certain search criteria, such as criteria that describes Health, Load, Time in database or Type.
- Click the + symbol to add a new database Collector.
- Click Actions to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.
- Click View to switch between card and list view.
- See the overall performance of your databases and if there are any critical issues with them. The server health or status indicates whether criteria has been met for normal, warning or critical health rules. The green checkmark indicates that the database is operating within normal conditions, the red exclamation mark indicates that at least one critical health rule has been violated.
- Click the name of the database to switch to the Database Dashboard for that specific database.

Databases Card View

From the Databases card view, you can see:

- The Total Executions graph that provides insight into the total number of calls (transactions for PostgreSQL databases) monitored for the database over the selected time period.
- The Time in Database graph that indicates the total time spent executing those calls during the selected time period. This metric is expressed in hours, minutes, and seconds.
- CPU Usage metrics that when enabled shows the percentage of CPU capacity consumed by the database.
Databases List View

From the Databases list view, you can:

- Click **View Options** to turn the spark charts on and off.
- Get to the databases you want to see quickly if you are monitoring many databases.
- Click on any point in the spark charts to see a metric for that point in time. For example, clicking on a point in the calls Trend shows how many calls were monitored at that time.
- Click on a column name to sort the list on that key.
Database Dashboard

On this page:

- Permissions
- Access the Database Dashboard
- Features of the Database Dashboard

Related pages:

- Access Database Visibility from Application Monitoring Views
- View Overall Database and Server Performance
- Discover Normal Database and Server Activity
- Database Health Rules and Alerts
- Monitor Database Performance

The dashboard for each individual database provides detailed information about your database.

Permissions

To view database collectors, users need a role with the Can View All Collectors permission.

Access the Database Dashboard

You can access the Database Dashboard in one of the following ways:

- In AppDynamics Home, on the Databases card, click the name of the database for which you want to see the Database Dashboard.
- In the Databases overview window, click the name of the database for which you want to see the Database Dashboard.

Features of the Database Dashboard

On the Database Dashboard you can:

- Click the dropdown arrow next to the database Collector name at the top of the page to choose a database that you want to see the dashboard for.
- Click the down-arrow next to the clustered database Collector name to navigate between different nodes of the cluster.
- Click the status icon below SERVERS in the top section of the dashboard to go directly to Events where you can see any recent events.

On the Database Dashboard you can see the following information:

- **Server Health**: Server Health at the top of this window indicates the extent to which health rules are being violated:
  - Green - Healthy server
  - Yellow/orange - server with warning-level violations
  - Red - server with critical-level violations
- **Type**: The database type.
- **Time Spent in Database and Executions**:
  - Load - At a glance, you can see the total number of calls (transactions for PostgreSQL databases) during the specified time period and the number of calls for any point in time.
  - Time spent in Database - The total time spent executing SQL statements during the specified time period.
  - Changes - Changes made to the database configuration parameters.
  - Max CPU - It displays the CPU cores present in the database server and is disabled by default. The CPU core count is collected using SQL (for Oracle, DB2, and MSSQL) and Hardware monitoring (if enabled for MySQL, PostgreSQL and Sybase).
- **Top 10 SQL Wait States (not available for Couchbase)**: Activities that contribute to the time it takes the database to service the request. The wait states consuming the most time may point to performance bottlenecks. For example, a db file sequential read wait state may be caused by segment header contention on indexes or by disk contention. See your database platform documentation for descriptions of the SQL wait states. For example, the following was developed from information in a Microsoft Customer Service blog for SQL Server:
  - Wait State - CHECKPOINT_QUEUE
- **Description** - Used by background worker that waits on events on queue to process checkpoint requests.
- **Recommended Action** - You should be able to safely ignore this one as it just indicates the checkpoint background worker is waiting for work to do. I suppose if you thought you had issues with checkpoints not working or log truncation you might see if this worker ever “wakes up”. Expect higher wait times as this will only wake up when there is work to do.

- **Top 10 Phases (Couchbase only)**: The phases that have processed the most documents.
- **Average number of active connections**: The average number of sessions that are actively running a query during the selected time period.

### Accessing the Comparison Report
Click on any data point to view the time comparison report, which shows query run times and wait states 15 minutes before and 15 minutes after the selected time.

---

The CPU, Memory, Disk I/O, and Network I/O graphs display when the Database Agent has been configured to also monitor the database host hardware. See, "Configure the Database Agent to Monitor Server Hardware" on Add Database Collectors.

- **CPU**: The CPU graph shows the relative percentages of CPU processing time used for handling system and users processes.
- **Memory**: The Memory graph shows the percentage of total memory in use at any point in time.
- **Disk I/O**: The Disk I/O graph shows disk usage, the volume of data read and written.
- **Network I/O**: The Network I/O graph shows network activity, the volume of data sent and received.
Accessing the Metric Browser

To see more information about a specific metric, double click any point on the graph and the Metric Browser opens displaying that metric. You can then hover over a point on the graph in the Metric Browser for more information about the metric. This feature is available for all the graphs except the SQL wait states graph.
Database Activity Window

On this page:
- Access the Database Activity Window
- Features of the Database Activity Windows
  - Wait State
  - Phase
  - Top Activity
  - Time Comparison
  - I/O
  - Top Query
  - Query Wait State
  - Parameter Changes
  - BT Activity

Related pages:
- Access Database Visibility from Application Monitoring Views
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The following describes the reports available in Database Visibility on the Database Activity Window. The reports available depend on the database platform being monitored. The reports are listed in alphabetical order and indicate which database platform supports them.

Access the Database Activity Window

To access the Database Activity window

1. To view a database's activity, click the name of the database.
2. Click the Activity tab.

Features of the Database Activity Windows

To view the database reports of different databases, click the down arrow next to the database name at the top of the page. From the list, select the database you want to view or search for the database in the search bar. Click the refresh icon to show only databases that meet that search criterion.

The activity reports available are described below. The reports available are database-dependent. The reports are listed in alphabetical order and indicate which database platform supports them. The reports will only show 10 days of data, which is the Events Service limit.

**Wait State**

**Description:** This report displays time-series data on Wait Events (states) within the database. Each distinct wait is color-coded, and the Y-axis displays time in seconds. This report also displays data in a table and highlights the time spent in each wait state for each SQL statement.

**Relevance:** The wait states consuming the most time may point to performance bottlenecks. For example, db file sequential reads may be caused by segment header contention on indexes or by disk contention.

**Platform:** IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, and Sybase IQ

**Max CPU** - It displays the CPU cores present in the database server and is disabled by default. The CPU core count is collected using SGL (for Oracle, DB2, and MSSQL) and Hardware monitoring (if enabled for MySQL, PostgreSQL and Sybase).

**Phase**

**Description:** This report displays the number of documents processed during a phase of query execution.

**Relevance:** The phase processing the most documents may point to performance bottlenecks.

**Platform:** Couchbase
Top Activity

**Description:** Display the top time in database SQL statements in a time-series view. This report also displays data in a table and highlights the time spent in the database for each of 10 top SQL statements. Only queries reported by newer agents are grouped; the queries reported by older agents may not appear upon grouping.

**Relevance:** Use this report to see which SQL statements are using the most database time. This helps to determine the impact of specific SQL statements on overall system performance allowing you to focus your tuning efforts on the statements that have the most impact on database performance.

**Platform:** IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, Sybase IQ, and Couchbase

Time Comparison

**Description:** This report allows you to compare the performance of two databases during the same time period based on a specific statistic type. You can click Group Similar to group together queries with the same number of parameters. Only queries reported by newer agents are grouped; the queries reported by older agents may not appear upon grouping.

**Relevance:** You may want to compare the performance of your development and production databases before and after you tune the SQL queries or add an index or join. This report can help you determine the effectiveness of any performance tuning procedures you have implemented.

**Platform:** IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, and Sybase IQ

I/O

**Description:** This report gives information on physical I/O performed by the database instance.

The SQL Server and PostgreSQL I/O reports include the following metrics:

- Time spent in the database
- Query throughput
- CPU usage (if hardware monitoring is enabled)
- Read and Write I/O. This metric is only available for SQL Server.
- Per file statistics (if the agent is run with the `dbagent.mssql.datafile.statistics` property). This metric is only available for SQL Server.
- Per database statistics. This metric is only available for PostgreSQL.

**Relevance:** Your physical disk I/O may be affecting database performance. Poor response times may mean one of the following:

- You're doing too much physical I/O and need to adjust your I/O capacity
- You're scanning tables or indexes when you should be doing seeks
- Your database tables are missing indexes
- Your SQL needs to be tuned

**Platform:** Microsoft SQL Server, Oracle, PostgreSQL

Top Query

**Description:** This report displays the top 10 SQL statements for the specified statistic in a time-series view. You can filter queries by command type using the Command Type dropdown menu.

**Relevance:** Use this report to see which SQL statements are using the most database resources. This helps to determine the impact of specific SQL statements on overall system performance allowing you to focus your tuning efforts on the statements that have the most impact on database performance.

**Platform:** IBM DB2, Microsoft SQL Server, Oracle

Query Wait State

**Description:** This report displays the wait times for all queries.

**Relevance:** Use this report to see how much time queries are spending in different wait states.

**Platform:** IBM DB2, Microsoft SQL Server, MongoDB, MySQL, Oracle, PostgreSQL, Sybase, and Sybase IQ
**Parameter Changes**

**Description:** This report displays changes to the database configuration parameters. You can select a parameter change and click one of the following options:

- **View Event Details:** Displays the parameter change event details, such as comments associated with the parameter change.
- **View Comparison Report:** Compares the query run times and wait states 15 minutes before and 15 minutes after the database parameter was changed. The comparison report is not available for clusters.

**Relevance:** Use this report to keep track of the changes you make to the database configuration parameters.

**Platform:** IBM DB2, Microsoft SQL Server, Microsoft Azure, MySQL, Oracle, PostgreSQL, Sybase ASE, and Sybase IQ

**BT Activity**

**Description:** This report displays the top (Java or PHP) business transactions that make calls to the database.

**Relevance:** Use this report to see which business transactions are affected by the database.

**Platform:** IBM DB2, Microsoft SQL Server, Microsoft Azure, MySQL, Oracle, PostgreSQL, Sybase ASE, and Sybase IQ
Database Topology Window

On this page:
- Access the Database Topology Window
- Features of the Database Topology Window

The Topology view displays current activity for the routing service and shards of a MongoDB database, Oracle RAC server or a custom cluster. The Topology view shows key performance metrics for the routing service and shards or nodes of the database.

Access the Database Topology Window

To access the Database Topology Window

1. To view a database's topology, click the name of the database.
2. Click the Topology tab.

Features of the Database Topology Window

On the Database Topology window you can:

- Select the Auto Refresh box. From the Every list, you can choose how often the system updates the live view.
- Click the dropdown arrow next to the database Collector name at the top of the page to choose a node or cluster that you want to see the metric views for. You can either select the database Collector name from the list or search for the database Collector by entering text in the search bar and then clicking the refresh icon to show only database Collectors that meet those search criteria.
- Hover over the sections of the trends to see their details.
- Click a column title on the Topology name list to sort the topologies using that column as the sort key.

For the Topology window you can see the following information:

- **Health**: Indicates whether any health rules for the routing service or shards of the replica set or nodes have been violated.
- **Role**: The role performed by the shard. The routing service routes the queries to the shards to balance the load. The members or shards of the replica sets can have either primary or secondary shard role.
- **Num Queries**: The number of queries handled.
- **Queries Trend**: The trend of queries handled over time.
- **Time in Database**: The time required for the database to respond to the queries.
- **Time Spent in Database Trend**: The trend of time spent in the database over time.

If you change the port number for a monitored database node, the Topology window shows a new node with the new port number, as well as the original node with the old port number. The original node remains in the Topology window for the duration of the retention period that you specify in Controller Settings.
<table>
<thead>
<tr>
<th>Name</th>
<th>Health</th>
<th>Role</th>
<th>Num...</th>
<th>Queries Trend</th>
<th>Time in...</th>
<th>Time Spent in Database Trend</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbmon5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MongoClient-4.5.1.779</td>
<td></td>
<td>standalone node</td>
<td>0</td>
<td></td>
<td>00:00:00</td>
<td></td>
</tr>
<tr>
<td>routing service</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>routing_service: ip-10-0-0-81:27917</td>
<td></td>
<td>routing service</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>routing service: ip-10-0-0-82:27917</td>
<td></td>
<td>routing service</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>se0 repl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>se0_repl: e2-c3-214-126-160.us-west2</td>
<td></td>
<td>secondary shard</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>se0_repl: e2-c3-214-184-86.us-west2</td>
<td></td>
<td>primary shard</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set1 repl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set1_repl: e2-c3-214-126-160.us-west2</td>
<td></td>
<td>secondary shard</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set1_repl: e2-c3-214-184-86.us-west2</td>
<td></td>
<td>primary shard</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Database Live View Window

On this page:
- Access the Database Live Window
- Features of the Database Live Window

Related pages:
- Access Database Visibility from Application Monitoring Views
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The Live View displays current activity for the database server. The live view shows key performance metrics for the last time period, which can be refreshed in a specified time interval between 10 seconds and 5 minutes.

Access the Database Live Window

To access the Database Live Window

1. To view a database's current performance, click the name of the database.
2. Click the **Live View** tab.
3. For clustered databases, select the node to view its details.

Features of the Database Live Window

On the Database Live window you can:

- Select the Auto Refresh box and then from the Every list, choose how often you want the system to update the live view.
- Click the down arrow next to the database collector name at the top of the page to choose to view the live view of a different database Collector by either selecting the database Collector name from the list or by searching for the database collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Hover over the sections of the charts to see their details.
- Click the View list to view All Active Sessions or All Sessions.
- Click a column title on the Session List to sort the Session List using the clicked column as the sort key.

On the Database Live window you can see the following information:

- **CPU Usage**: If you have enabled Hardware Monitoring for the server the chart shows the percentage of available CPU resources consumed by users and the system and a break down of how the CPU is being used.
  - IO - Disk input and output.
  - System - Database operating system activities.
  - User - User interaction with the database.
- **Memory Usage**: Displays how much memory is in use and how much is available.
- **Query wait states**: Activities that contribute to the time it takes the database to service the request. The wait states consuming the most time may point to performance bottlenecks. For example, database file sequential reads may be caused by segment header contention on indexes or by disk contention.
- **Session List**: A table describing database usage instances and their properties. The database session is the application interaction with the database, a new session is started for each request. The columns displayed are database dependent.
- **Blocking Tree**: A tree showing all the sessions causing blocks. Nested under each blocking session are the sessions being blocked and the corresponding queries and blocked objects.
For SQL Server, the CPU Usage and Memory Usage statistics are displayed in Live View even when hardware monitoring is disabled. In the Live View for database clusters, you must select a node from the dropdown menu to view that node's activity.
The Queries window displays the SQL statements and stored procedures that consume the most time in the database. You can compare the query weights to other metrics such as SQL wait times to determine SQL that requires tuning.

Access the Database Queries Window

To access the Database Queries Window

1. To view a database's queries, click the name of the database.
2. Click the **Queries** tab.

Features of the Database Queries Window

On the Database Queries window you can:

- View the top N queries. These are the queries that consumed the greatest amount of database time to complete.
- Choose to display the top 5, 10, 100 or 200 queries.
- Click the name of a column to sort the Query list using that key.
- Click **Filter by Wait States** to choose wait states to filter the Query list by. The filtered list only displays queries that caused the selected wait states.
- Check the **Group Similar** box to group together queries with the same syntax. Queries that share the same syntax but different 'IN' clause parameters are still grouped together. Query grouping is case-insensitive. Double-click a query group to dive into the Group Details for more information about that query group. Only queries reported by newer agents are grouped; the queries reported by older agents may not appear upon grouping.
- Search for a specific query, by entering text in the search bar that may appear in the Query. This is useful if you found a slow query in the Slow Database calls window of AppDynamics Application monitoring.
- Double-click a query or select a query and click **View Query Details** to dive into the Query Details for more detailed information about that specific query. If you see a message saying that query details are unavailable, you may need to update your permissions or extend your query retention period.
- Click the down arrow next to the database Collector name at the top of the page to choose to view the database queries of a different database Collector by either selecting the database Collector from the list or by searching for the database Collector by entering text in the search bar and then clicking the refresh icon to show only database Collectors that meet that search criteria.
- **Click Actions** to take one of the following actions:
  - Export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.
  - Rename the selected query so that it is easier to identify. Note that the query name is associated with the query itself. Thus, when a query’s retention period expires, both the query and the query name disappear.

On the Database Queries window you can see the following information:

- **Query Id**: This is the unique ID assigned to the query internally by the database.
- **Query**: The text of the query, custom query name, or, in the case of a stored procedure invocation, the name of the stored procedure. If the user does not have the ANY permission to view the stored procedure name, the definition of the stored procedure is shown instead of the stored procedure name.

For SQL Server, Azure, Oracle, and DB2 the metrics Number of Executions, Average Response Time and Elapsed Time are inferred from the query index. When you are viewing the queries with Group Similar checked, the statistics reflect only the queries that have captured query statistics. Queries that do not have query statistics are excluded from the statistic values.
For databases such as MySQL and MongoDB which do not have a query index, no data will appear in the Number of Executions and Average Response Time columns. However, Elapsed Time can be inferred for MySQL and MongoDB databases using the wait state index. Elapsed Time can also be inferred using the wait state index when query statistics for a query run on an SQL Server, Azure, Oracle or DB2 database are otherwise not available. When query statistics are not available for a query run on an SQL Server, Azure, Oracle or DB2 database, the Number of Executions and Average Response Time statistics cannot be reported and are marked with a hyphen (-). This means that the database did not provide statistics for this query in the selected time range.

- **Elapsed Time**: The total time consumed by all executions of this query. Query statistics such as Number of Executions, Average Response Time and Elapsed time are available for databases supporting the query index, such as SQL Server, Azure, Oracle, and DB2. For databases where the query index is not available, the elapsed time is inferred from the wait state index. The query sampling frequency is once every second. If a query takes less than 1 second and happens to occur in the time between sampling instances, its elapsed time is reported as 0 seconds.

- **Number of Executions**: (supported for SQL Server, Azure, Oracle, DB2) The number of times the query ran during the specified time period. If you are monitoring Greenplum, the number of executions appears as 0.

- **Average Response Time (hh:mm:ss)**: (supported for SQL Server, Azure, Oracle, DB2) The average time required during the specified time period to respond to the query.

- **Weight (%)**: The percentage of the total time consumed by the query.

Once you have identified the statements that are consuming the most resources, you can dig down deeper for details that can help you tune the SQL statement. For in-depth query details, click a SQL statement and then click View Query Details. See Database Query Details Window.
Database Query Details Window

On this page:
- Access the Database Query Details Window
- Features of the Database Query Details Window

Related pages:
- Access Database Visibility from Application Monitoring Views
- Add Database Collectors
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

Once you have identified the statements on the Database Queries window that are spending the most amount of time in the database, you can dig down deeper for details that can help you tune those SQL statements. The database instance Query Details window displays details about the query selected on the Database Queries window.

Access the Database Query Details Window

To access the Database Query Details Window
1. To view a database's query details, click the name of the database.
2. Click the Queries tab.
3. Select a query and click View Query Details.

Features of the Database Query Details Window

On the Database Query Details window you can:
- Click Execution Plan to view an explanation plan for the query you selected on the Queries window if it was a SELECT query.
- Click on the graphs to see metrics for that specific point in time.
- Click the down arrow next to the database Collector name at the top of the page to choose to view the query details of a different database Collector.

On the Database Query Details window you can see the following information:
- Resource consumption over time: For SQL Server, Azure, Oracle, and DB2, shows the amount of time the query spent in the database using resources, the number of executions, and the amount of CPU time consumed. Query statistics are only available for the aforementioned database platforms.
  The resource consumption card is also available for query groups. If all queries in the group have query metrics (e.g. elapsed time and number of executions), then the aggregate of the group's metrics are shown. If none of the queries in the group have query metrics, then the aggregate elapsed time of the queries in the group, taken from the wait state index, is shown. If some queries in the group have query metrics and others do not, then the aggregate value of the available query metrics is shown.
- Business Transactions Executing Similar Queries: Displays the Java or PHP business transactions that execute queries similar to this query.
- Disk and buffer usage: Displays data when hardware monitoring is enabled for this database collector. This graph shows at a glance how effectively the database buffer is used. SQL that runs frequently is best kept in the buffer as physical disk reads are much more time-consuming. You may need to optimize your buffer manager if you see that the ratio of physical reads to buffer gets is too high.
- Clients: Shows the machines that executed the selected SQL statement and the percentage of the total time required to execute the statement performed by each machine. The table also shows the applications, nodes, and tiers of the machines.
- <database_type> wait states: Activities that contribute to the time it takes the database to service the selected SQL statement. The wait states consuming the most time may point to performance bottlenecks. For example, a db file sequential read wait state may be caused by segment header contention on indexes or by disk contention. See your database platform documentation for descriptions of the SQL wait states and recommended actions.
- Query Active in Database/Schema: Shows the schemas that have been accessed by this SQL.
- Query ID (Oracle) SQL Handle (SQL Server): A unique ID that allows the database server to more quickly locate this SQL statement in the cache.
- Query: The entire syntax of the selected SQL statement. You can click the pencil icon in the top right corner of the Query card to edit the query name so that it is easy to identify.
- Queries in Group (only in Group Details view): The top 100 queries in the query group
- Users: The users that executed this query.
- Programs/Applications: The program/application from which this query was executed.
To see how the SQL was executed internally within the database, click the Execution Plan tab. The statement execution plan is the sequence of operations the database performs to run the statement.

**Oracle Explain Plan Limitations**
For Oracle, AppDynamics Database Visibility displays execution plans for only SELECT, UPDATE, INSERT, and DELETE statements because Database Visibility relies on the Oracle EXPLAIN PLAN statement to obtain the execution plans and the Oracle EXPLAIN PLAN statement only provides the Oracle optimizer's execution plans for SELECT, UPDATE, INSERT, and DELETE statements. See also, User Permissions for Oracle.

If you have a poorly performing piece of SQL that you are trying to tune, the obvious place to start is to look at the most costly step of the execution plan.

Tuning SQL is a vast topic, but a couple of things to look out for include:

- Index or table scans: May indicate a need for better or additional indexes.
- Bookmark Lookups: Consider changing the current clustered index, using a covering index and limiting the number of columns in the SELECT statement.
- Filter: Remove any functions in the WHERE clause, don't include views in your Transact-SQL code, may need additional indexes.
- Sort: Does the data really need to be sorted? Can an index be used to avoid sorting? Can sorting be done at the client more efficiently?
Database Query Execution Plan Window

On this page:
- Access the Database Query Execution Plan Window
- Features of the Database Query Execution Plan Window

Related pages:
- Access Database Visibility from Application Monitoring Views
- Add Database Collectors
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The Database Query Execution Plan window can help you to determine the most efficient execution plan for your queries. Once you've discovered a potentially problematic query, you can run the EXPLAIN PLAN statement to check the execution plan that the database created. A query's execution plan reveals whether the query is optimizing its use of indexes and executing efficiently. This information is useful for troubleshooting queries that are executing slowly.

**Access the Database Query Execution Plan Window**

**To access the Database Query Execution Plan Window**

1. To view a query execution plan, click the name of the database.
2. Click the **Queries** tab.
3. Click a SELECT statement you want to examine and click **View Query Details**.
4. Click the **Execution Plan** tab.

**Features of the Database Query Execution Plan Window**

On the Database Query Execution Plan window, you can take the following actions:

- Click **Explain** to view the execution plan for a query.
- Click **Schema** to choose a different schema and then explain the execution plan based on that schema. If you want to view the execution plan for a query that belongs to a schema outside the scope of your database account permissions, you can enter a username and password for a different database account that has access to the schema.
- If you want to improve a suboptimal query, you can modify the query and paste it into the **Explain another query** box. Click the **Explain** button to generate the execution plan for your modified query and determine whether the modified query is more efficient.
- Click the down arrow next to the query name at the top of the page to choose to view queries and query execution plans of a different database Collector. You can either select the database Collector name from the list or search for the database Collector in the search bar. Click the refresh icon to show only database Collectors that meet your search criteria.

If you are using Oracle, SQL Server or Azure you can view the execution plan in one of the following ways:

- Select a cached execution plan and click **View Cached Plan** to view the plan details.
- Double-click a cached execution plan.

If you are using MySQL, you can also take the following actions:

- Choose to explain the query on another schema. In **Plan Details**, from the list on the right, choose the schema name and then click **EXPLAIN**.
- Copy the text in the Parsed SQL output box and paste it into the **Explain another query** box where you can edit it and then explain the edited version of the query.

On the Database Query Execution Plan window you can see the following information:

- **Cached Execution Plan(s)**: identifying details of the cached execution plan, which vary from one database to another. For SQL Server, the execution plan is rendered as a diagram.
- **Plan Details**: execution plan details, showing the step by step procedure the database followed to process the selected query.
- **Referenced Objects**: the database objects accessed by the execution plan.
### Cached Execution Plan(s)

Select a cached execution plan to explain

<table>
<thead>
<tr>
<th>Plan_Hash_Value</th>
<th>Last Used</th>
<th>Executions</th>
<th>Average Elapsed Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2995648711</td>
<td>Mon Sep 17 2018 13:25:00 GMT-0700 (PDT)</td>
<td>207</td>
<td>3.576</td>
</tr>
</tbody>
</table>

### Plan Details

Execution Plan for: New Plan

<table>
<thead>
<tr>
<th>Operation</th>
<th>Name</th>
<th>Optimizer Mode</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (MCPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0] SELECT STATEMENT</td>
<td>ALL_ROWS</td>
<td>1</td>
<td>N/A</td>
<td>56.3k</td>
<td></td>
</tr>
<tr>
<td>[1] SORT (AGGREGATE)</td>
<td>N/A</td>
<td>1</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>[2] MERGE JOIN (CARTESIAN)</td>
<td>106.3m</td>
<td>N/A</td>
<td>56.3k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3] INDEX (FAST FULL SCAN)</td>
<td>SYS_C007014</td>
<td>10.3k</td>
<td>N/A</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>[4] BUFFER SORT</td>
<td>10.3k</td>
<td>N/A</td>
<td>56.3k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5] INDEX (FAST FULL SCAN)</td>
<td>SYS_C007014</td>
<td>10.3k</td>
<td>N/A</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Number of steps: 5
Explained for schema: APPDYN

### Referenced Objects

<table>
<thead>
<tr>
<th>Tree Node</th>
<th>Operation</th>
<th>Object Type</th>
<th>Object Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>INDEX FAST FULL SCAN</td>
<td>INDEX (UNIQUE)</td>
<td>SYS_C007014</td>
</tr>
<tr>
<td>5</td>
<td>INDEX FAST FULL SCAN</td>
<td>INDEX (UNIQUE)</td>
<td>SYS_C007014</td>
</tr>
</tbody>
</table>
Database Clients Window

On this page:

- Access the Database Clients Window
- Features of the Database Clients Window

Related pages:

- Access Database Visibility from Application Monitoring Views
- Database Queries Window
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The Database Clients window shows you the hostname or IP addresses of the Top N clients using the database. A database client is any host that accesses the database instance. To see the Top N Queries run by the client, click a client name and then click View Top Queries. From the client top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Clients Window

To access the Database Clients window

1. To view a database's clients, click the name of the database.
2. Click the Clients tab.

Features of the Database Clients Window

On the Database Clients window you can:

- Choose to view information for the top 10, 50, 100 or 200 clients sorted by total time spent in the database.
- Choose to view information for one client in particular.
- Click the down arrow next to the database name at the top of the page to choose to view the database clients of a different database by either selecting the database from the list or by searching for the database by entering text in the search bar and then clicking the refresh icon to show only databases that meet that search criteria.
- Click Actions to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

On the Database Clients window you can see the following information:

- **Client**: The names of the machines that have connected to the database.
- **Elapsed Time**: The total time spent by the client executing queries.
- **Weight (%)**: The percentage of time the client was connected to the database in comparison with connection times of other clients.
Database Sessions Window

On this page:
- Access the Database Sessions Window
- Features of the Database Sessions Window

Related pages:
- Access Database Visibility from Application Monitoring Views
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The Sessions window shows you the Session ID of the Top N sessions using the database sorted by time spent. To see the Top N SQL run by a session, click the Session ID and then click View Top Queries. From the Client top queries page, to see more details about a query, double-click a query to open the Query Details window and from there see the Execution Plan of the query.

Access the Database Sessions Window

To access the Database Sessions Window

1. To view a database's sessions, click the name of the database.
2. Click the Sessions tab.

Features of the Database Sessions Window

On the Database Sessions window you can:
- Choose to view information for the top 10, 50, 100 or 200 sessions consuming database resources.
- Double click a session ID to view the queries run in that particular session.
- Click the down arrow next to the database name at the top of the page to choose to view the database sessions of a different database by either selecting the database from the list or by searching for the database by entering text in the search bar and then clicking the refresh icon to show only databases that meet that search criteria.
- Click Actions to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

On the Database Sessions window you can see the following information:

- **Session ID**: the Session ID of each database instance usage.
- **Weight (%)**: the percentage of time that instance was using the database in comparison with the database usage of other sessions.

---

**Oracle-4.5.1.779**

<table>
<thead>
<tr>
<th>Session ID</th>
<th>Client</th>
<th>Elap. Time</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ip-10-0-0-33.us-west-2.compute.internal</td>
<td>00:53:00</td>
<td>99.9</td>
</tr>
<tr>
<td>787</td>
<td>qiang.mac.corp.appdynamics.com</td>
<td>00:00:03</td>
<td>0.1</td>
</tr>
<tr>
<td>399</td>
<td>ip-10-0-9-91</td>
<td>00:00:01</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Database Blocking Sessions Window

On this page:
- Access the Database Blocking Sessions Window
- Features of the Database Blocking Tree Window

The Blocking Tree displays a tree view of sessions that are blocking other sessions for Oracle, SQL Server, Sybase ASE, and DB2. You can use the blocking tree to determine the cause of deadlocks.

Access the Database Blocking Sessions Window

To access the Database Topology Window

1. To view a database's blocking sessions, click the name of the database.
2. Click the **Blocking Sessions** tab.

Features of the Database Blocking Tree Window

On the Database Blocking Tree Window you can:

- Expand the tree of a blocking session to see the sessions it is blocking

On the Database Sessions window you can see the following information:

- **Session ID**: the Session ID of the sessions causing the block, and the sessions being blocked
- **Client**: the names of the machines where the session is blocked
- **User**: the user of the blocked session
- **Query**: the query that is blocked or causing the block
- **Blocked Object**: the object that is blocked
- **First occurrence of block**: the time that the block first occurred in the specified time range
- **Block Duration**: the amount of time that the session was blocked or causing the block
Database Schemas and Databases Windows

On this page:
- Access the Database Schemas or Databases Window
- Features of the Database Schemas or Databases Window

The Database Schemas window is available only for Oracle database servers.

The Database Databases window is available for other database servers.

The database Schemas or Databases window shows you the names of the Top N busiest schemas or databases on the database server. To see the Top N Queries run on a specific Database or Schema, click a name and then click View Top Queries. From the Schema or Database top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Schemas or Databases Window

To access the Database Schemas or Databases window

1. In AppDynamics Home, on the Databases card, click the name of the database that you want to view the schemas or databases for.
2. Depending on the platform, click either the Schemas or Databases tab.

Features of the Database Schemas or Databases Window

On the Database Schemas and Databases window you can:

- Choose to view information for the top 10, 50, 100 or 200 schemas or databases containing the most time spent executing queries by the database server.
- Double click the name of a schema or database to view the top queries for one schema database in particular, or select the schema or database and click View Top Queries.
- Click the down arrow next to the Database Collector name at the top of the page to choose to view the database schemas or databases of a different database Collector by either selecting the database Collector from the list or by searching for the database Collector by entering text in the search bar and then clicking the refresh icon to show only database Collectors that meet that search criteria.
- Click Actions to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

On the Database Schemas or Databases window you can see the following information:

- Schema or database: The names of the schemas or database on the database server.
- Weight (%): The percentage of time the schema or database was used by the database server in comparison with the usage percentage of other schemas or databases.
Database Modules Window

On this page:
- Access the Database Modules Window
- Features of the Database Modules Window

The Database Modules window is available only for Oracle database servers. The Module field is often identical to the Program field unless it has been programmatically set with a call to DBMS_APPLICATION_INFO. Some packaged applications, such as Oracle E-Business Suite, automatically set the Module field with relevant identifiers.

The database Modules window shows you the names of the Top N busiest modules on the Oracle database server. To see the Top N Queries run on the database by a module, click a module name and then click View Top Queries. From the Modules top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Modules Window

To access the Database Modules window

1. To view a database’s modules, click the name of the database.
2. Click the Modules tab.

Features of the Database Modules Window

On the Database Modules window you can:

- Choose to view information for the top 10, 50, 100 or 200 modules most frequently accessing the database server.
- Double click the name of a module to view the top queries for one module in particular, or select the module and click View Top Queries.
- Click the down arrow next to the database collector name at the top of the page to choose to view the modules of a different Oracle database collector by either selecting the database collector from the list or by searching for the database collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Click Actions to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

On the Database Modules window you can see the following information:

- **Module**: The name of the module run by the active sessions in the database.
- **Weight (%)**: The percentage of time the module was consuming the database resources in comparison with the usage percentage of other modules.
Database Programs Window

On this page:

- Access the Database Programs Window
- Features of the Database Programs Window

The Database Programs window is available for Oracle and SQL Server.

The Programs window shows you the names of the Top N programs on the database instance based on time. To see the Top N Queries run on the database by a program, click a program name and then click View Top Queries. From the Programs top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Programs Window

To access the Database Programs window

1. To view a database's programs, click the name of the database.
2. Click the Programs tab.

Features of the Database Programs Window

On the Database Programs window you can:

- Choose to view information for the top 10, 50, 100 or 200 programs most frequently accessing the database server.
- Double click the name of a program to view the top queries for one program in particular, or select the program and click View Top Queries.
- Click the down arrow next to the database collector name at the top of the page to choose to view the programs of a different Oracle database collector by either selecting the database collector from the list or by searching for the database collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Click Actions to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

On the Database Programs window you can see the following information:

- **Program**: The name of the program run by an active session in the database.
- **Weight (%)**: The percentage of time the program was consuming the database resources in comparison with the usage percentage of other programs.
Database Users Window

On this page:
- Access the Database Users Window
- Features of the Database Users Window

The database Users window shows you the users using the database. To see the Top N Queries run by a user, double-click the User name, or single-click the User name and then click View Top Queries. From the User top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Users Window

To access the Database Users window
1. To view a database's users, click the name of the database.
2. Click the Users tab. You may need to click the expand button (>>) to see the Users tab.

Features of the Database Users Window

On the Database Users window you can:
- Use the Top users dropdown menu to view information for the top 10, 50, 100 or 200 users sorted by total time spent in the database.
- Click a user to view information for about that user.
- Click the down arrow next to the database name at the top of the page to choose to view the database users of a different database. You can either select the database from the list or search for the database by entering text in the search bar and clicking the refresh icon to show only databases that meet that search criterion.
- Search for the database by entering text in the search bar and then clicking the refresh icon to show only databases that meet that search criterion.
- Click Actions to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

On the Database Users window you can see the following information:
- User: The names of the user accounts that have connected to the database.
- Elapsed Time: Duration of time that the user spent executing queries.
- Weight (%): Time the user spent executing queries as a percentage of the total time spent by all users.
Database Buckets Window

On this page:

The Buckets window is available only for Couchbase collector.

The Buckets window shows you the names of the top 200 buckets sorted in descending order based on operations. To see the details of a specific bucket, click a bucket name and then click View Bucket Details.

Access the Buckets Window

1. In AppDynamics Home, on the Databases card, click the name of the Couchbase collector that you want to view the buckets for.
2. Click the Buckets tab.

Features of the Buckets Window

On the Buckets window you can:

- Double click the name of the bucket to view the bucket statistics for the selected time range.
- Click the down arrow next to the Couchbase collector name at the top of the page to choose to view the buckets of a different Couchbase collector by either selecting the Couchbase collector from the list or by searching for the Couchbase collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Click Actions to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

On the Buckets window you can see the following statistics for a bucket:

Operations (ops), Bytes Read (bytes_read), Bytes Written (bytes_written), Total Disk Size, in MB (couch_total_disk_size), Hit Ratio (hit_ratio)
Database Business Transactions Window

On this page:
- Supported Platforms
- Access the Database Business Transactions Window
- Features of the Database Business Transactions Window

The Database Business Transactions Window shows you the top Java or PHP business transactions, ranked by their cumulative execution time on the database. To see the Top N SQL run by a business transaction, click the business transaction and then click View Top Queries. From the Business Transaction top queries page, you can double-click a query to open the Query Details window.

Supported Platforms

AppDynamics supports Java as an application framework. Relational databases are supported as backends that need to be mapped on the Java APM to a monitored instance in database visibility. For more information, see Linking Database with application backend.

Access the Database Business Transactions Window

To access the Top Business Transactions window

1. To view a database's business transactions, click the name of the database.
2. Click the Business Transactions tab.

Features of the Database Business Transactions Window

On the Database Business Transactions window you can:
- Choose to view information for the top 10, 50, 100 or 200 business transactions executing queries on the database.
- Double-click a business transaction to view the queries run in that particular transaction.
- Click the down arrow next to the database name at the top of the page to choose to view the business transactions that access a different database by either selecting the database from the list or by searching for the database by entering text in the search bar and then clicking the refresh icon to show only databases that meet that search criteria.
- Click Actions to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

On the Database Business Transactions window you can see the following information:

- Business Transaction: the business transactions that run the most number of queries on the database.
- Application: the application that each business transaction is executing in.
- Weight (%): the percentage of time that business transaction took to execute compared to other business transactions.

![Database Business Transactions Window Image](image-url)
Database Applications Window

On this page:
- Access the Database Applications Window
- Features of the Database Applications Window

The Database Applications window is available for Postgres, Sybase ASE, Sybase IQ, and DB2.

The Applications window shows you the names of the Top N Applications on the database instance based on time. To see the Top N Queries run on the database by an application, click the name of an application and then click View Top Queries. From the Applications top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Applications Window

To access the Database Applications window:
1. To view a database's applications, click the name of the database.
2. Click the Applications tab.

Features of the Database Applications Window

On the Database Applications window you can:
- Choose to view information for the top 10, 50, 100 or 200 applications most frequently accessing the database server.
- Double click the name of an application to view the top queries for one application in particular, or select the application and click View Top Queries.
- Click the down arrow next to the database collector name at the top of the page to choose to view the applications of a different MongoDB database collector by either selecting the database collector from the list or by searching for the database collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Click Actions to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

On the Database Applications window you can see the following information:
- Applications: The name of the application run by an active session in the database.
- Weight (%): The percentage of time the application was consuming the database resources in comparison with the usage percentage of other applications.
Database Object Browser Window

On this page:
- Access the Database Objects Window
- Features of the Database Objects Window

Related pages:
- Access Database Visibility from Application Monitoring Views
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

When you detect a problem with database performance, you may be able to correct the problem by adding an index, refusing access to certain users, or creating a new index on a table. You can browse through the database objects to find ways to tune the database, including the indexes that are currently available on the various tables. The contents of the Database Object Browser are updated when you access them or click Refresh.

Note: The objects available for browsing are database dependent. For clustered databases, select the node to browse its objects.

Access the Database Objects Window

To access the Database Objects window

1. To view a database's objects, click the name of the database.
2. Click the Objects tab.

Features of the Database Objects Window

On the Database Objects window you can:

- Click an object type in the tree to see more information about it.
- For Oracle, search for a specific database item.
- Click the down arrow next to the database name at the top of the page to choose to view the database objects of a different database by either selecting the database from the list or by searching for the database by entering text in the search bar and then clicking the refresh icon to show only databases that meet that search criteria.
The following are some examples of browsable content.

- Database uptime and version for all platforms
- Users: who has access to the database for all platforms (supported for Oracle, MySQL)
- Job Status: status of SQL agent job (supported for SQL Server)
- Parameters: database initialization parameters (supported for Oracle)
- Variables: system variables indicating how the database was configured.
- Current Statistics: the current value of statistics collected. The various statistics available are database dependent and are a superset of those shown in the Database Monitoring - Metric Browser page.
- System Waits: SQL waits experienced by the database
- Database Objects: search for specific objects, and browse through database schemas, tables, columns, and indices.
  - Search
Database Custom Metrics Window

On this page:
- Access the Database Custom Metrics Window
- Features of the Database Custom Metrics Window

Related pages:
- Configuring Custom Metrics

The Custom Metrics window displays metrics for the custom queries that you specified for Database Visibility to monitor.

Access the Database Custom Metrics Window

To access the Database Custom Metrics window

1. To view a database's objects, click the name of the database.
2. Click the Custom Metrics tab.

Features of the Database Custom Metrics Window

On the Database Custom Metrics window you can:

- View the values for each row of the custom query output. For example, if the query returns four rows, then the graph displays four lines.
- Double click the graph to view the custom metric in the Metric Browser.
Monitor Database Server Hardware

On this page:
- Access the Database Server Hardware Metrics Window
- Features of the Hardware Metrics graphs
- Comparing Server and Database Metrics

Access the Database Server Hardware Metrics Window

To access the Hardware Metrics

Hardware metrics are reported on:
- The Database Dashboard window.
- The Database Live window.
- The Metric Browser

Features of the Hardware Metrics graphs

On the hardware metrics graphs you can:
- Click anywhere on the graphs to get details about activity at that point in time.
- Double-click KPI points in the graph to display the Metric Browser showing the selected metric. Metrics are collected by the agent and reported once a minute.

Comparing Server and Database Metrics

You can view and compare hardware and database metrics in the Metric Browser. This can help you visualize the effect of server performance on database performance.
Database Health Rules and Alerts

You can configure AppDynamics Database Visibility to alert you when certain conditions are met or exceeded for monitored databases, operating systems, and server hardware. Use the Getting Started Wizard to help you through the process if you're new to using AppDynamics Pro.

You configure Health Rules, Actions, and Email Digests for monitoring databases almost exactly the same as you would configure these for monitored applications. To view the health rules for your databases, click Alert & Respond > Health Rules and select Databases from the dropdown menu at the top. The Affects tab of the Create Health Rule dialog is specific to Database Health Rules. In the Affects tab, you can apply a database health rule to all databases, all databases of a specific database type, or to select instances of a specific database type being monitored. You select the metrics on which to base the database health rule. The health rule is violated when specified critical or warning conditions are met.

Each Health Rule is only evaluated for the subset of databases or database that you currently have permissions for. If your permissions have changed since you created the Health Rule, then your current permissions are applied to the Health Rule evaluation.

Database Policy Actions are limited to email or SMS message notifications and custom actions that have been uploaded to the Controller.

Once you have defined Health Rules and Actions, you can create a Policy based on a Health Rule to send you an email or an SMS message when a Health Rule is violated.
Database Monitoring Metrics

On this page:

- Key Performance Indicators (KPI)
- Couchbase Metrics
- DB2 Server Metrics
- MongoDB Server Metrics
- Microsoft SQL Server Metrics
- Microsoft SQL Azure Metrics
- MySQL Server Metrics
- Oracle Server Metrics
- PostgreSQL Server Metrics
- Sybase ASE and IQ Server Metrics

Related page:

Metric Browser

This document describes metrics collected by the Database Agent. It shows the path to the metric from the main AppDynamics menu down to the relevant branch of the Metric Browser tree. Some of these metrics are displayed in the Database Visibility user interface and so are also documented with the name of the window, column, section, and metric name.

The Metric Browser window name uses the following format:

- Metric Browser - application_name: When you access the Metric Browser from Applications.
- Metric Browser - Database Monitoring: When you access the Metric Browser from Infrastructure.

For most types of metrics in the browser, you can click any of the points in the graph to view more information about the metric observed at that point in time. The information shown includes the metric identifier, date and time of the observation, along with any of the following values relevant to the metric:

- **Obs** (observed value): the average of all data points seen for that interval. For a cluster or a time rollup, this represents the weighted average across nodes or over time.
- **Min**: the minimum data point value seen for that interval
- **Max**: the maximum data point value seen for that interval
- **Sum**: the sum of all data point values seen for that interval.
- **Count**: number of observations aggregated in that one point. For example, a count of 5 indicates that there were 5 1-minute data points aggregated into one point.

**Key Performance Indicators (KPI)**

These metrics are available for all database platforms supported.

**Calls per Minute**: The number of SQL calls to the database per minute.

**DB Availability**: The times when the database is available (has an active connection). If the database is not available, nothing is reported.

**Number of Connections**: The number of connections established with the database at any point during the selected time period. A connection is a session established between a database client and a server. Since the value displayed is the average number of connections over a time range, it is common to see a very low number or 0 for this metric.

**Time Spent in Execution (s)**: The current amount of time the database spent executing SQL statements.

**Total Database Size** (PostgreSQL only): The amount of disk space (in MB) that the database is using.

**Couchbase Metrics**

- **Gc_num**: Number of objects garbage collected
- **Gc_pause_time**: Garbage collection pause time
- **Gc_pause_percent**: Garbage collection pause percentage
**Memory usage**: Memory (in GB) currently used by Couchbase

**Memory total**: Memory (in GB) used by Couchbase over the total period of time

**Memory system**: Memory (in GB) used by the system.

**Cpu_user_percent**: Percentage of CPU used by the user

**Cpu_sys_percent**: Percentage of CPU used by the system

**Request_completed_count**: Number of requests completed

**Request_active_count**: Number of active requests

**Request_per_sec_1min**: Requests per second in the last minute

**Request_per_sec_5min**: Requests per second in the last 5 minutes

**Request_per_sec_15min**: Requests per second in the last 15 minutes

**Request_time_mean**: Average request time

**Request_time_median**: Median request time

**Request_time_95**: 95th percentile request time

**Request_time_99**: 99th percentile request time

**Request_prepared**: Number of requests prepared

**Request_time_80**: 80th percentile request time

**Request_time_median**: Median request time

**Request_time_95**: 95th percentile request time

**Request_time_99**: 99th percentile request time

**Request_prepared**: Number of requests prepared

**Cpu_utilization_rate**: Rate of CPU utilization

**Swap total**: Total amount of swap available

**Swap used**: Amount of swap used

**Mem total**: Total available memory

**Mem free**: Total amount of free memory

**Cmd get**: Compare and Swap gets

**Couch docs actual disk size**: The physical memory used in the node

**Couch docs data size**: Data size of couch documents associated with a node

**Couch spatial data size**: Size of object data for spatial views

**Couch spatial disk size**: Amount of disk space occupied by spatial views

**Couch views actual disk size**: Amount of disk space occupied by Couch views

**Couch views data size**: Size of object data for Couch views

**Curr items**: Number of active items in memory

**Curr items tot**: Total number of items

**Ep bg fetched**: Disk reads per second

**Get hits**: Number of get hits

**Mem used**: Engine's total memory usage

**Ops**: Spatial operations

**Vb replica curr items**: Number of in memory items

**DB2 Server Metrics**

**ACTIVE_SORTS**: The number of sorts in the database that currently have a sort heap allocated.

**AGENTS_TOP**: At the database level, it is the maximum number of agents for all applications.

**APPLS_CURCONS**: The number of applications that are currently connected to the database.

**APPLS_IN_DB2**: The number of applications that are currently connected to the database, and for which the database manager is currently processing a request.

**BINDS_PRECOMPILES**: The number of binds and pre-compiles attempted. You can use this element to gain insight into the current level of activity within the database manager. This value does not include the count of int_auto_rebinds, but it does include binds that occur as a result of the REBIND PACKAGE command.

**CONNECTIONS_TOP**: The highest number of simultaneous connections to the database since the database was activated.

**COORD_AGENTS_TOP**: The highest number of coordinating agents. If the peak number of coordinating agents represents too high a workload for this node, you can reduce this upper boundary by changing the `max_coordagents` configuration parameter.

**DDL_SQL_STMTS**: The number of SQL Data Definition Language (DDL) statements that were executed.

**DEADLOCKS**: The number of deadlocks that have occurred.

**DIRECT_READ_REQS**: Use the following formula to calculate the average number of sectors that are read by a direct read: `direct_reads / direct_read_reqs`.

**DIRECT_READ_TIME**: Time spent doing direct read operations.
DIRECT_READS: Direct reads are performed in units, the smallest being a 512-byte sector. They are used when: Reading LONG VARCHAR columns, Reading LOB (large object) columns, Performing a backup.

DIRECT_WRITE_REQS: Use the following formula to calculate the average number of sectors that are written by a direct write: direct_writes / direct_write_reqs.

DIRECT_WRITE_TIME: Time spent doing direct write operations.

DIRECT_WRITES: Direct writes are performed in units, the smallest being a 512-byte sector. They are used when: Writing LONG VARCHAR columns, Writing LOB (large object) columns, Performing a restore, Performing a load, Allocating new extents for SMS table space if MPFA is enabled (which is the default).

DYNAMIC_SQL_STMTS: The number of dynamic SQL statements that were attempted.

ELAPSED_EXEC_TIME_MS: The total time (in milliseconds) required to execute all the statements for a particular application during the specified time period.

ELAPSED_EXEC_TIME_S: The total time (in seconds) required to execute all the statements for a particular application during the specified time period.

FAILED_SQL_STMTS: The number of SQL statements that were attempted, but failed.

INT_AUTO_REBINDS: The number of commits initiated internally by the database manager.

INT_COMMITS: The number of commits initiated internally by the database manager.

INT_DEADLOCK_ROLLBACKS: The number of forced rollbacks initiated by the database manager due to a deadlock. A rollback is performed on the current unit of work in an application selected by the database manager to resolve the deadlock.

INT_ROLLBACKS: The number of rollbacks initiated internally by the database manager.

INT_ROWS_DELETED: The number of rows deleted from the database as a result of internal activity.

INT_ROWS_INSERTED: The number of rows inserted into the database as a result of internal activity caused by triggers.

INT_ROWS_UPDATED: The number of rows updated from the database as a result of internal activity.

LOCK_ESCALS: The number of times that locks have been escalated from several row locks to a table lock.

LOCK_LIST_IN_USE: The total number of bytes of lock list memory in use.

LOCK_TIMEOUTS: The number of times that a request to lock an object timed-out instead of being granted.

LOCK_WAIT_TIME: The total elapsed time (in milliseconds) spent waiting for locks.

LOCK_WAITS: The number of times that applications or connections waited for locks.

LOCKS_HELD: The number of locks currently held.

LOCKS_WAITING: The number of agents waiting on a lock.

LOG_HELD_BY_DIRTY_PAGES: The amount of log (in bytes) corresponding to the difference between the oldest dirty page in the database and the top of the active log.

LOG_READ_TIME_NS: The total elapsed time spent by the logger reading log data from the disk. For event monitors that write to tables, the value of this element is given in microseconds by using the BIGINT data type.

LOG_READ_TIME_S: At the database level, this is the number of subagents for all applications.

LOG_READS: The number of log pages read from disk by the logger.

LOG_TO_REDO_FOR_RECOVERY: The size of the log (in bytes) that will have to be redone for crash recovery.

LOG_WRITE_TIME_NS: The total elapsed time spent by the logger writing log data to the disk. For event monitors that write to tables, the value of this element is given in microseconds by using the BIGINT data type.

LOG_WRITE_TIME_S: At the database level, this is the number of subagents for all applications.

LOG_WRITES: The number of log pages written to disk by the logger.

NUM_ASSOC_AGENTS: At the database level, this is the number of subagents for all applications.
**NUM_INDOUBT_TRANS**: The number of outstanding indoubt transactions in the database. Indoubt transactions hold log space for uncommitted transactions, which can cause the logs to become full. When the logs are full, further transactions cannot be completed. The resolution of this problem involves a manual process of heuristically resolving the indoubt transactions. This monitor element provides a count of the number of currently outstanding indoubt transactions that must be heuristically resolved.

**NUM_LOG_BUFFER_FULL**: The number of times agents had to wait for log data to write to disk while copying log records into the log buffer.

**NUM_LOG_DATA_FOUND_IN_BUFFER**: The number of times log data was read from buffer instead of from disk, which is slower.

**NUM_LOG_PART_PAGE_IO**: Number of I/O requests issued by the logger for writing partial log data to disk. To determine if the current disk is adequate for logging, use this metric in conjunction with log_writes, log_write_time, and num_log_write_io.

**NUM_LOG_READ_IO**: Number of I/O requests issued by the logger to read log data from disk. To determine if the current disk is adequate for logging, use this metric in conjunction with log_reads and log_read_time.

**NUM_LOG_WRITE_IO**: Number of I/O requests issued by the logger to write log data to disk. To determine if the current disk is adequate for logging, use this metric in conjunction with log_writes and log_write_time.

**POOL_ASYNC_DATA_READ_REQS**: Number of asynchronous read requests by the prefetcher to the operating system. These requests are usually large block I/Os of multiple pages.

**POOL_ASYNC_DATA_READS**: Number of data pages read in from the table space physical containers by asynchronous engine dispatchable units for all types of table spaces. To determine the number of physical read that were performed synchronously, use this metric along with the pool_data_p_reads metrics.

**POOL_ASYNC_DATA_WRITES**: Number of times a buffer pool data page was physically written to disk by either an asynchronous page cleaner or prefetcher. To determine how well the buffer page cleaners are performing, use this metric in conjunction with pool_data_writes and pool_async_data_writes metrics.

**POOL_ASYNC_INDEX_READS**: Number of index pages read in from the physical table space containers by asynchronous engine dispatchable units for all types of table spaces. To determine how well the prefetchers are working, compare the ratio of asynchronous reads to total physical reads. Use this information to tune the num_ioservers configuration parameter.

**POOL_ASYNC_INDEX_WRITES**: Number of times a buffer pool index page was physically written to disk by either an asynchronous page cleaner or prefetcher.

**POOL_ASYNC_READ_TIME**: Total number of milliseconds spent reading in data and index pages from physical table space containers by asynchronous engine dispatchable units for all types of table spaces. Use this metric to analyze the I/O work being performed.

**POOL_ASYNC_WRITE_TIME**: Total number of milliseconds spent writing data or index pages from the buffer pool to disk by database manager page cleaners. Use this metric to analyze the I/O work being performed.

**POOL_DATA_FROM_ESTORE**: Number of buffer pool data pages read from the extended storage monitor.

**POOL_DATA_L_READS**: Number of data pages which have been requested from the buffer pool (logical) for regular and large table spaces. The functionality to record buffer pool information at the statement level is supported for API and CLP snapshot requests.

**POOL_DATA_P_READS**: Number of index pages read in from the physical table space containers (physical) for regular and large table spaces. The functionality to record buffer pool information at the statement 7 level is supported for API and CLP snapshot requests.

**POOL_INDEX_L_READS**: Number of index pages which have been requested from the buffer pool (logical) for 9 regular and large table spaces. The functionality to record buffer pool information at the statement level is supported for API and CLP snapshot requests.

**POOL_INDEX_P_READS**: Number of index pages read in from the table space containers (physical) for regular and large table spaces. The functionality to record buffer pool information at the statement level is supported for API and CLP snapshot requests.

**POOL_INDEX_WRITES**: Number of times a buffer pool index page was physically written to disk.

**POOL_DRTY_PG_STEAL_CLNS**: Number of times a page cleaner was invoked because a synchronous write was needed during the victim buffer replacement for the database.

**POOL_DRTY_PG_THRSH_CLNS**: Number of times a page cleaner was invoked because a buffer pool had reached the dirty page threshold criterion for the database.

**POOL_INDEX_L_READS**: Number of index pages which have been requested from the buffer pool (logical) for regular and large table spaces. The functionality to record buffer pool information at the statement level is supported for API and CLP snapshot requests.

**POOL_INDEX_P_READS**: Number of index pages read in from the table space containers (physical) for regular and large table spaces. The functionality to record buffer pool information at the statement level is supported for API and CLP snapshot requests.

**POOL_INDEX_WRITES**: Number of times a buffer pool index page was physically written to disk.

**POOL_LSN_GAP_CLNS**: Number of times a page cleaner was invoked because the logging space used had reached a predefined criterion for the database.

**POOL_NO_VICTIM_BUFFER**: Number of times an agent did not have a preselected victim buffer available.
POOL_READ_TIME: Number of milliseconds spent reading in data and index pages from the physical table space for all types of table spaces.

POOL_TEMP_DATA_L_READS: Number of data pages which have been requested from the logical buffer pool for temporary table spaces.

POOL_TEMP_DATA_P_READS: Number of data pages read in from the physical table space containers for temporary table spaces.

POOL_TEMP_INDEX_L_READS: Number of index pages which were requested from the logical buffer pool for temporary table spaces.

POOL_TEMP_INDEX_P_READS: Number of index pages read in from the physical table space containers for temporary table spaces.

POOL_WRITE_TIME: Number of milliseconds spent physically writing data or index pages from the buffer pool to disk.

PREFETCH_WAIT_TIME: Number of milliseconds spent waiting for an I/O prefetcher server to finish loading pages into the buffer pool.

ROWS_DELETED: Number of row deletions attempted.

ROWS_INSERTED: Number of row insertions attempted.

ROWS_READ: Number of rows read from tables.

ROWS_SELECTED: Number of rows that have been selected and returned to the application.

ROWS_UPDATED: Number of row updates attempted.

SEC_LOG_USED_TOP: The maximum number of bytes of secondary log space used.

SEC_LOGS_ALLOCATED: Number of secondary log files currently being used for the database.

SELECT_SQL_STMTS: The number of SQL SELECT statements that were executed.

SORT_HEAP_ALLOCATED: The number of allocated pages of sort heap space for all sorts at the level chosen and at the current time.

SORT_OVERFLOWS: The number of sorts that ran out of sort heap and may have required disk space for temporary storage.

SORT_SHRHEAP_ALLOCATED: The total amount of shared sort memory allocated in the database.

SORT_SHRHEAP_TOP: Database-wide shared sort memory high-water mark in 4k pages.

STATIC_SQL_STMTS: The number of static SQL statements that were attempted.

TOT_LOG_USED_TOP: The maximum amount of total log space used (in bytes).

TOTAL_APP_COMMITS: Total number of commit statements issued by the client application.

TOTAL_APP_ROLLBACKS: Total number of rollback statements issued by the client application.

TOTAL_CONS: Number of newly opened connections to the database.

TOTAL_LOG_AVAILABLE: Number of bytes of active log space in the database that is not being used by uncommitted transactions.

TOTAL_LOG_USED: Number of bytes of active log space currently used in the database.

TOTAL_SEC_CONS: The number of connections made by a subagent to the database at the node.

TOTAL_SORT_TIME: The total elapsed time (in milliseconds) for all sorts that have been executed.

TOTAL_SORTS: The number of sorts that have been executed.

TOT_LOG_USED_TOP: The maximum amount of total log space used (in bytes).

UID_SQL_STMTS: Number of SQL UPDATE, INSERT, and DELETE statements that were executed.

UNREAD_PREFETCH_PAGES: Number of pages that the prefetcher read in that were never used.

MongoDB Server Metrics

asserts_msg: The number of message asserts. These are internal server errors that have a well defined text string. Stack traces are logged for these.
asserts_regular: The number of regular asserts raised since this process started.

asserts_user: The number of user asserts. These are errors that can be generated by a user such as out of disk space or duplicate key.

asserts_warning: The number of warnings raised since this process started.

BackgroundFlushingFlushes: The number of times the database has flushed all writes to disk.

BackgroundFlushingTotalMs: The number of milliseconds (ms) that the mongod processes have spent writing (i.e. flushing) data to disk.

ConnectionsAvailable: The number of unused available connections that the database can provide. Consider this value in combination with the value of current to understand the connection load on the database, and the UNIX ulimit Settings document for more information about system thresholds on available connections.

ConnectionsCurrent: The number of connections to the database server from clients. This number includes the current shell session. Consider the value of available to add more context to this datum. This figure will include the current shell connection as well as any inter-node connections to support a replica set or sharded cluster.

CursorTimedOut: The number of cursors that have timed out since the server process started. If this number is large or growing at a regular rate, this may indicate an application error.

CursorTotalOpen: The number of cursors that MongoDB is maintaining for clients. Because MongoDB exhausts unused cursors, typically this value small or zero. However, if there is a queue, stale tailable cursors, or a large number of operations this value may rise.

globalLockActiveClients: The number of connected clients.

globalLockCurrentQueue: The number of operations queued waiting for the lock. A consistently small queue, particularly of shorter operations should cause no concern. Also, consider this value in light of the size of queue waiting for the read lock (e.g. readers) and write lock (e.g. writers) individually.

MemMapped: The number of megabytes of memory mapped by the database. Because MongoDB uses memory-mapped files, this value will be roughly equivalent to the total size of your databases.

MemResident: The amount of RAM, in megabytes (MB), currently used by the database process. In normal use this value tends to grow. In dedicated database servers this number tends to approach the total amount of system memory.

MemVirtual: The quantity, in megabytes (MB), of virtual memory used by the mongod process. With journaling enabled, the value of virtual is at least twice the value of mapped. If virtual value is significantly larger than mapped (e.g. 3 or more times), this may indicate a memory leak.

NetworkBytesIn: The amount of network traffic, in bytes, received by this database. Use this value to ensure that network traffic sent to the mongod process is consistent with expectations and overall inter-application traffic.

NetworkBytesOut: The amount of network traffic, in bytes, sent from this database. Use this value to ensure that network traffic sent by the mongod process is consistent with expectations and overall inter-application traffic.

NetworkNumRequests: The number of distinct requests that the server has received. Use this value to provide context for the bytesIn and bytesOut values to ensure that MongoDB network utilization is consistent with expectations and application use.

OpCountersCommand: The number of commands sent to MongoDB.

OpCountersDelete: The number of Delete operations.

OpCountersGetmore: The number of GetMore operations.

OpCountersInsert: The number of Insert operations.

OpCountersQuery: The number of Query operations.

OpCountersUpdatedAt: The number of Update operations.

OplogMaxSize (new in 4.5.4): The maximum size of the operation log

OplogSize (new in 4.5.4): The size of the operation log

OplogTimeDiffinSec (new in 4.5.4): The difference (in seconds) between the first entry in the long and the last entry in the log

ReplCommand: The number of Replicated Commands issued to the database.

ReplDelete: The number of Replicated Delete operations.
**Repl_getmore**: The number of GetMore operations. This counter can be high even if the query count is low. Secondary nodes send getMore operations as part of the replication process.

**Repl_insert**: The number of replicated insert operations.

**Repl_query**: The number of Replicated Queries.

**Repl_update**: The number of Replicated Update Operations.

**Replication_MyState** (new in 4.5.4): An integer between 0 and 10 that represents the replica state of the current member.

**Index_missRatio**: Ratio of index hits to misses. If there are a lot of index misses then you should look at your queries to see if they are optimally using your indexes. You may need to add new indexes to make the queries run faster. You can explain the query to see see which index queries are hitting and the total execution time so you can compare query performance before and after adding the new indexes.

---

**Microsoft SQL Server Metrics**

**Active cursors**: The number of active cursors.

**Active Temp Tables**: The number of temporary tables/table variables in use.

**Active Transactions**: The number of active transactions for the database. AU cleanup batches/sec The number of batches per second that were completed successfully by the background task that cleans up deferred dropped allocation units.

**AU cleanup batches/sec**: The number of batches per second that were completed successfully by the background task that cleans up deferred dropped allocation units.

**AU cleanups/sec**: The number of allocation units per second that were successfully dropped the background task that cleans up deferred dropped allocation units. Each allocation unit drop requires multiple batches.

**Auto-Param Attempts/sec**: The number of auto-parameterization attempts per second. Total should be the sum of the failed, safe, and unsafe auto-parameterizations. Auto-parameterization occurs when an instance of SQL Server tries to parameterize a Transact-SQL request by replacing some literals with parameters so that reuse of the resulting cached execution plan across multiple similar-looking requests is possible. Note that auto-parameterizations are also known as simple parameterizations in newer versions of SQL Server. This counter does not include forced parameterizations.

**Average Latch Wait Time (ms)**: Average latch wait time (in milliseconds) for latch requests that had to wait.

**Average Wait Time (ms)**: Average amount of wait time (in milliseconds) for each lock request that resulted in a wait.

**Backup/Restore Throughput/sec**: Read/write throughput for backup and restore operations of a database per second. For example, you can measure how the performance of the database backup operation changes when more backup devices are used in parallel or when faster devices are used. Throughput of a database backup or restore operation allows you to determine the progress and performance of your backup and restore operations.

**Batch Requests/sec**: The number of Transact-SQL command batches received per second. This statistic is affected by all constraints (such as I/O, number of users, cache size, complexity of requests, and so on). High batch requests mean good throughput.

**Buffer cache hit ratio**: Percentage of pages found in the buffer cache without having to read from disk. The ratio is the total number of cache hits divided by the total number of cache lookups over the last few thousand page accesses. The ratio is displayed as a percentage. After a long period of time, the ratio moves very little. Because reading from the cache is much less expensive than reading from disk, you want this ratio to be high. Generally, you can increase the buffer cache hit ratio by increasing the amount of memory available to SQL Server.

**Bulk Copy Rows/sec**: The number of rows bulk copied per second.

**Bulk Copy Throughput/sec**: Amount of data bulk copied (in kilobytes) per second.

**By-reference Lob Create Count**: Count of large object (lob) values that were passed by reference. By-reference lob values are used in certain bulk operations to avoid the cost of passing them by value.

**By-reference Lob Use Count**: Count of by-reference lob values that were used. By-reference lob values are used in certain bulk operations to avoid the cost of passing them by-value.

**Cache Hit Ratio**: Ratio between cache hits and lookups. The ratio is displayed as a percentage.

**Cache Object Counts**: The number of cache objects in the cache.

**Cache Objects in use**: The number of cache objects in use.
Cache Pages: The number of 8-kilobyte (KB) pages used by cache objects.

Cached Cursor Counts: The number of cursors of a given type in the cache.

Checkpoint pages/sec: The number of pages flushed to disk per second by a checkpoint or other operation that require all dirty pages to be flushed.

CLR Execution: Total execution time in CLR (microseconds)

Connection Memory (KB): Total amount of dynamic memory the server is using for maintaining connections.

Count Lob Readahead: Count of lob pages on which readahead was issued.

Count Pull In Row: Count of column values that were pulled in-row from off-row.

Count Push Off Row: Count of column values that were pushed from in-row to off-row.

Cursor memory usage: Amount of memory consumed by cursors in kilobytes (KB).

Count/sec: Times each type of cached cursor has been used. Cursor memory usage Amount of memory consumed by cursors in kilobytes (KB).

Cursor Requests/sec: The number of SQL cursor requests received by the server. Cursor worktable usage Number of worktables used by cursors. Data File(s) Size (KB) Cumulative size (in kilobytes) of all the data files in the database including any automatic growth. Monitoring this counter is useful, for example, for determining the correct size of Database pages Number of pages in the buffer pool with database content.

Cursor worktable usage: Number of worktables in use by cursors.

Data File(s) Size(KB): Cumulative size (Kb) of all data files in the database including any automatic growth. Monitoring this counter to determine the correct size of tempdb.

Database pages: Number of database pages in use.

DBCC Logical Scan Bytes/sec: The number of logical read scan bytes per second for database console commands (DBCC).

Deferred Dropped rowsets: The number of rowsets created as a result of aborted online index build operations that are waiting to be dropped by the background task that cleans up deferred dropped rowsets.

Dropped rowset cleanups/sec: The number of rowsets per second created as a result of aborted online index build operations that were successfully dropped by the background task that cleans up deferred dropped rowsets.

Dropped rowsets skipped/sec: The number of rowsets per second created as a result of aborted online index build operations that were skipped by the background task that cleans up deferred dropped rowsets created.

Errors/sec: The number of errors/sec.

Event Notifications Delayed Drop: The number of event notifications waiting to be dropped by a system thread.

Extent Deallocations/sec: The number of extents deallocated per second in all databases in this instance of SQL Server.

Extents Allocated/sec: The number of extents allocated per second in all databases in this instance of SQL Server.

Failed AU cleanup batches/sec: The number of batches per second that failed and required retry, by the background task that cleans up deferred allocation units. Failure could be due to lack of memory or disk space, hardware failure, and other reasons.

Failed Auto-Params/sec: The number of failed auto-parameterization attempts per second. This should be small. Note that auto-parameterizations are also known as simple parameterizations in later versions of SQL Server.

Failed leaf page cookie: The number of times that a leaf page cookie could not be used during an index search since changes happened on the leaf page. The cookie is used to speed up the index search.

Failed tree page cookie: The number of times that a tree page cookie could not be used during an index search since changes happened on the parent pages of those tree pages. The cookie is used to speed up the index search.

Forced Parameterizations/sec: The number of successful forced parameterizations per second.

Forwarded Records/sec: The number of records per second fetched through forwarded record pointers.

Free list stalls/sec: The number of requests per second that had to wait for a free page.

Free Space in tempdb (KB): The amount of space (in kilobytes) available in FreeSpace Page.

FreeSpace Page Fetches/sec: Number of pages fetched/second by free space scans. These scans search for free space within pages
already allocated to an allocation unit, to address requests to insert or modify record fragments.

**FreeSpace Scans/sec**: The number of scans per second that were initiated to search for free space within pages already allocated to an allocation unit to insert or modify record fragment. Each scan may find multiple pages.

**Full Scans/sec**: The number of unrestricted full scans per second. These can be either base-table or full-index scans.

**Granted Workspace Memory (KB)**: Total amount of memory currently granted to executing processes such as hash, sort, bulk copy, and index creation operations.

**Guided Plan Executions/sec**: The number of plan executions per second in which the query plan has been generated by using a plan guide.

**HTTP Authenticated Requests**: The number of authenticated HTTP requests started per second. Index Searches/sec Number of index searches per second. These are used to start a range scan, reposition a range scan, revalidate a scan point, fetch a single index record, and search down the index to locate where to insert a new row.

**Index Searches/sec**: Number of index searches per second. These are used to start a range scan, reposition a range scan, revalidate a scan point, fetch a single index record, and search down the index to locate where to insert a new row.

**Latch Waits/sec**: The number of latch requests that could not be granted immediately.

**Lazy writes/sec**: The number of buffers written per second by the buffer manager's lazy writer. The lazy writer is a system process that flushes out batches of dirty, aged buffers (buffers that contain changes that must be written back to disk before the buffer can be reused for a different page) and makes them available to user processes. The lazy writer eliminates the need to perform frequent checkpoints in order to create available buffers.

**LobHandle Create Count**: Count of temporarylobs created.

**LobHandle Destroy Count**: Count of temporary lobs destroyed.

**LobSS Provider Create Count**: Count of LOB Storage Service Providers (LobSSP) created. One worktable created per LobSSP.

**LobSS Provider Destroy Count**: Count of LobSSP destroyed.

**LobSS Provider Truncation Count**: Count of LobSSP truncated.

**Lock Blocks**: Current number of lock blocks in use on the server (refreshed periodically). A lock block represents an individual locked resource, such as a table, page, or row.

**Lock Blocks Allocated**: Current number of allocated lock blocks. At server startup, the number of allocated lock blocks plus the number of allocated lock owner blocks depends on the SQL Server

**Lock Memory (KB)**: Total amount of dynamic memory the server is using for locks.

**Lock Owner Blocks**: The number of lock owner blocks currently in use on the server (refreshed periodically). A lock owner block represents the ownership of a lock on an object by an individual thread. Therefore, if three threads each have a shared (S) lock on a page, there will be three lock owner blocks.

**Lock Owner Blocks Allocated**: Current number of allocated lock owner blocks. At server startup, the number of allocated lock owner blocks and the number of allocated lock blocks depend on the SQL Server

**Lock Requests/sec**: The number of new locks and lock conversions per second requested from the lock manager.

**Lock Timeouts (timeout > 0)/sec**: The number of lock requests per second that timed out, but excluding requests for NOWAIT locks.

**Lock Timeouts/sec**: The number of lock requests per second that timed out, including requests for NOWAIT locks.

**Lock Wait Time (ms)**: Total wait time (in milliseconds) for locks in the last second.

**Lock Waits/sec**: The number of lock requests per second that required the caller to wait.

**Log Bytes Flushed/sec**: Total number of log bytes flushed.

**Log Cache Hit Ratio**: Percentage of log cache reads satisfied from the log cache.

**Log Cache Reads/sec**: Reads performed per second through the log manager cache.

**Log File(s) Size (KB)**: Cumulative size (in kilobytes) of all the transaction log files in the database.

**Log File(s) Used Size (KB)**: The cumulative used size of all the log files in the database.

**Log Flush Wait Time** Total wait time (in milliseconds) to flush the log.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Flush Waits/sec</td>
<td>The number of commits per second waiting for the log flush.</td>
</tr>
<tr>
<td>Log Flashes/sec</td>
<td>The number of log flushes per second.</td>
</tr>
<tr>
<td>Log Growths</td>
<td>Total number of times the transaction log for the database has been expanded.</td>
</tr>
<tr>
<td>Log Shrinks</td>
<td>Total number of times the transaction log for the database has been shrunk.</td>
</tr>
<tr>
<td>Log Truncations</td>
<td>Total number of times the transaction log for the database has been truncated.</td>
</tr>
<tr>
<td>Logical Connections</td>
<td>The number of logical connections to the system.</td>
</tr>
<tr>
<td>Logins/sec</td>
<td>Total number of logins started per second. This does not include pooled connections.</td>
</tr>
<tr>
<td>Logouts/sec</td>
<td>Total number of logout operations started per second.</td>
</tr>
<tr>
<td>Longest Transaction Running Time</td>
<td>The length of time (in seconds) since the start of the transaction that has been active longer than any other current transaction.</td>
</tr>
<tr>
<td>Mars Deadlocks</td>
<td>The number of Mars Deadlocks detected.</td>
</tr>
<tr>
<td>Maximum Workspace Memory (KB)</td>
<td>Maximum amount of memory available for executing processes such as hash, sort, bulk copy, and index creation operations.</td>
</tr>
<tr>
<td>Memory Grants Outstanding</td>
<td>Total number of processes that have successfully acquired a workspace memory grant.</td>
</tr>
<tr>
<td>Memory Grants Pending</td>
<td>Total number of processes waiting for a workspace memory grant.</td>
</tr>
<tr>
<td>Mixed page allocations/sec</td>
<td>The number of pages allocated per second from mixed extents. These could be used for storing the IAM pages and the first eight pages that are allocated to an allocation unit.</td>
</tr>
<tr>
<td>Non-atomic yield rate</td>
<td>The number of non-atomic yields per second.</td>
</tr>
<tr>
<td>NonSnapshot Version Transactions</td>
<td>The number of currently active transactions that are not using snapshot isolation level and have made data modifications that have generated row versions in the Number of active cursor plans.</td>
</tr>
<tr>
<td>Number of active cursor plans</td>
<td>Number of cursor plans.</td>
</tr>
<tr>
<td>Number of Deadlocks/sec</td>
<td>The number of lock requests per second that resulted in a deadlock.</td>
</tr>
<tr>
<td>Number of SuperLatches</td>
<td>The number of latches that are currently SuperLatches.</td>
</tr>
<tr>
<td>Optimizer Memory (KB)</td>
<td>Total amount of dynamic memory the server is using for query optimization.</td>
</tr>
<tr>
<td>Page compression attempts/sec</td>
<td>The number of pages evaluated for page-level compression. Includes pages that were not compressed because significant savings could be achieved. Includes all objects in the instance of SQL Server. For information about specific objects, see sys.dm_db_index_operational_stats (Transact-SQL).</td>
</tr>
<tr>
<td>Page Deallocations/sec</td>
<td>The number of pages deallocated per second in all databases in this instance of SQL Server. These include pages from mixed extents and uniform extents.</td>
</tr>
<tr>
<td>Page life expectancy</td>
<td>The number of seconds a page will stay in the buffer pool without references.</td>
</tr>
<tr>
<td>Page lookups/sec</td>
<td>The number of requests per second to find a page in the buffer pool.</td>
</tr>
<tr>
<td>Page reads/sec</td>
<td>The number of physical database page reads that are issued per second. This statistic displays the total number of physical page reads across all databases. Because physical I/O is expensive, you may be able to minimize the cost, either by using a larger data cache, intelligent indexes, and more efficient queries or by changing the database design.</td>
</tr>
<tr>
<td>Page Splits/sec</td>
<td>The number of page splits per second that occur as the result of overflowing index pages.</td>
</tr>
<tr>
<td>Page writes/sec</td>
<td>The number of physical database page writes issued per second.</td>
</tr>
<tr>
<td>Pages Allocated/sec</td>
<td>The number of pages allocated per second in all databases in this instance of SQL Server. These include pages allocations from both mixed extents and uniform extents.</td>
</tr>
<tr>
<td>Pages compressed/sec</td>
<td>The number of data pages that are compressed by using PAGE compression. Includes all objects in the instance of SQL Server. For information about specific objects, see sys.dm_db_index_operational_stats (Transact-SQL).</td>
</tr>
<tr>
<td>Percent Log Used</td>
<td>Percentage of space in the log that is in use.</td>
</tr>
<tr>
<td>Probe Scans/sec</td>
<td>The number of probe scans per second that are used to find at most one single qualified row in an index or base table directly.</td>
</tr>
<tr>
<td>Processes blocked</td>
<td>The number of currently blocked processes.</td>
</tr>
</tbody>
</table>
Range Scans/sec: The number of qualified range scans through indexes per second.

Readahead pages/sec: The number of pages read per second in anticipation of use.

Repl. Trans. Rate: The number of transactions per second read out of the transaction log of the publication database and delivered to the distribution database.

Safe Auto-Params/sec: The number of safe auto-parameterization attempts per second. Safe refers to a determination that a cached execution plan can be shared between different similar-looking Transact-SQL statements. SQL Server makes many auto-parameterization attempts some of which turn out to be safe and others fail. Note that auto-parameterizations are also known as simple parameterizations in later versions of SQL Server. This does not include forced parameterizations.

Scan Point Revalidations/sec: The number of times per second that the scan point had to be revalidated to continue the scan.

Skipped Ghosted Records/sec: The number of ghosted records per second skipped during scans.

Snapshot Transactions: The number of currently active transactions using the snapshot isolation level. N.B. The SOAP Empty Requests Number of empty SOAP requests started per second.

SOAP Empty Requests: The number of SOAP method invocations passed to the stored procedure (or template) with an empty string as its value (not a NULL value) in order to provide an included input parameter with no value assigned to it.

SOAP Method Invocations: The number of SOAP method invocations started per second.

SOAP Session Initiate Requests: The number of SOAP Session initiate requests started per second.

SOAP Session Terminate Requests: The number of SOAP Session terminate requests started per second.

SOAP SQL Requests: The number of SOAP SQL requests started per second.

SOAP WSDL Requests: The number of SOAP Web Service Description Language requests started per second.

SQL Attention rate: The number of attentions per second. An attention is a request by the client to end the currently running request.

SQL Cache Memory (KB): Total amount of dynamic memory the server is using for the dynamic SQL cache.

SQL Compilations/sec: The number of SQL compilations per second. Indicates the number of times the compile code path is entered. Includes compiles caused by statement-level recompilations in SQL Server. After SQL Server user activity is stable, this value reaches a steady state.

SQL Re-Compilations/sec: The number of statement recompiles per second. Counts the number of times statement recompiles are triggered. Generally, you want the recompiles to be low. In later versions of SQL Server, recompiles are statement-scoped instead of batch-scoped recompilations in Microsoft SQL Server 2000. Therefore, direct comparison of values of this counter between SQL Server and earlier versions is not possible.

Stored Procedures Invoked/sec: This counter reports the total number of activation stored procedures invoked by all queue monitors in the instance per second.

SuperLatch Demotions/sec: The number of SuperLatches that have been demoted to regular latches in the last second.

SuperLatch Promotions/sec: The number of latches that have been promoted to SuperLatches in the last second.

Table Lock Escalations/sec: The number of times locks on a table were escalated to the TABLE or HoBT granularity. Target pages Ideal number of pages in the buffer pool.

Target pages: Ideal number of pages in the buffer pool.

Target Server Memory (KB): Total amount of dynamic memory the server can consume.

Task Limit Reached: The number of times that a queue monitor would have started a new task, but did not because the maximum number of tasks for the queue is already running.

Task Limit Reached/sec: The number of times per second that a queue monitor would have started a new task, but did not because the maximum number of tasks for the queue is already running.

Tasks Aborted/sec: The number of activation stored procedure tasks that end with an error or are aborted by a queue monitor for failing to receive messages.

Tasks Running: The number of activation stored procedures that are currently running.

Tasks Started/sec: The number of activation stored procedures started per second by all queue monitors in the instance.

Temp Tables Creation Rate: The number of temporary tables/table variables created per second.
Temp Tables For Destruction: The number of temporary tables/table variables waiting to be destroyed by the cleanup system thread.

Total Latch Wait Time (ms): Total latch wait time (in milliseconds) for latch requests in the last second.

Total Server Memory (KB): The committed memory from the buffer pool (in kilobytes).

Transactions/sec: The number of transactions started for the database per second.

Unsafe Auto-Params/sec: The number of unsafe auto-parameterization attempts per second. For example, the query has some characteristics that prevent the cached plan from being shared. These are designated as unsafe. This does not count the number of forced parameterizations.

Update conflict ratio: The percentage of those transactions using the snapshot isolation level that have encountered update conflicts within the last second. An update conflict occurs when a snapshot isolation level transaction attempts to modify a row that last was modified by another transaction that was not committed when the snapshot isolation level transaction started.

Update Snapshot Transactions: The number of currently active transactions using the snapshot isolation level and have modified data.

Used leaf page cookie: The number of times a leaf page cookie is used successfully during an index search since no change happened on the leaf page. The cookie is used to speed up the index search.

Used tree page cookie: The number of times a tree page cookie is used successfully during an index search since no change happened on the parent page of the tree page. The cookie is used to speed up the index search.

User Connections: The number of users currently connected to SQL Server.

Version Cleanup rate (KB/s): The rate (in kilobytes per second) at which row versions are removed from the snapshot isolation version store.

Version Generation rate (KB/s): The rate (in kilobytes per second) at which new row versions are added to the snapshot isolation version store.

Version Store Size (KB): The amount of space (in kilobytes) in Version Store unit count. The number of active allocation units in the snapshot isolation version store.

Version Store unit creation: The number of allocation units that have been created in the snapshot isolation store since the instance of the Database Engine was started.

Version Store unit truncation: The number of allocation units that have been removed from the snapshot isolation store since the instance of the Database Engine was started.

Workfiles Created/sec: The number of work files created per second. For example, work files could be used to store temporary results for hash joins and hash aggregates.

Worktables Created/sec: The number of work tables created per second. For example, work tables could be used to store temporary results for query spool, lob variables, XML variables, and cursors.

Worktables From Cache Ratio: Percentage of work tables created where the initial two pages of the work table were not allocated but were immediately available from the work table cache. (When a work table is dropped, two pages may remain allocated and they are returned to the work table cache. This increases performance.)

Write Transactions/sec: The number of transactions that wrote to the database and committed, in the last second.

Microsoft SQL Azure Metrics

avg_cpu_percent: Average compute utilization in the percentage of the limit of the service tier.

avg_data_io_percent: Average data I/O utilization in percentage based on the limit of the service tier.

avg_log_write_percent: Average write resource utilization in the percentage of the limit of the service tier.

avg_memory_percent: Average memory utilization in the percentage of the limit of the service tier.

MySQL Server Metrics

Aborted_clients: The number of clients that were aborted (because they did not properly close the connection to the MySQL server). For some applications, this can be OK, but for some other applications you might want to track the value, as aborted connects may indicate some sort of application failure.
Aborted_connects: The number of failed attempts to connect to the MySQL server.

Bytes_received: The number of bytes received from all clients.

Bytes_sent: The number of bytes sent to all clients.

Com_alter_table: The number of times each ALTERTABLE statement has been executed.

Com_create_index: The number of times each CREATE INDEX statement has been executed.

Com_create_table: The number of times each CREATE TABLE statement has been executed.

Com_delete: The number of times each DELETE statement has been executed.

Com_insert: The number of times each INSERT statement has been executed.

Com_optimize: The number of times each OPTIMIZE statement has been executed.

Com_select: The number of times each SELECT statement has been executed.

Com_update: The number of times each UPDATE statement has been executed.

Connections: The number of connection attempts (successful or not) to the MySQL server.

Created_tmp_disk_tables: The number of temporary tables on disk created automatically by the server while executing statements.

Created_tmp_files: How many temporary files mysqld has created.

Created_tmp_tables: The number of in-memory temporary tables created automatically by the server while executing statements. If Created_tmp_disk_tables is large, you may want to increase the tmp_table_size value to cause temporary tables to be memory-based instead of disk-based.

Handler_delete: The number of times that rows have been deleted from tables.

Innodb_buffer_pool_pages_data: The number of pages containing data (dirty or clean).

Innodb_buffer_pool_pages_dirty: The number of pages currently dirty.

Innodb_buffer_pool_pages_flushed: The number of buffer pool page-flush requests.

Innodb_buffer_pool_pages_free: The number of free pages.

Innodb_buffer_pool_pages_misc: The number of pages that are busy because they have been allocated for administrative overhead such as row locks or the adaptive hash index. This value can also be calculated as Innodb_buffer_pool_pages_total.

Innodb_buffer_pool_pages_total: The total size of the buffer pool, in pages.

Innodb_buffer_pool_read_ahead_rnd: The number of random read-aheads initiated by InnoDB. This happens when a query scans a large portion of a table but in random order.

Innodb_buffer_pool_read_requests: The number of logical read requests InnoDB has done.

Innodb_buffer_pool_reads: The number of logical reads that InnoDB could not satisfy from the buffer pool and had to do a single-page read.

Innodb_buffer_pool_wait_free: Normally, writes to the InnoDB buffer pool happen in the background. However, if it is necessary to read or create a page and no clean pages are available, it is also necessary to wait for pages to be flushed first. This counter counts instances of these waits. If the buffer pool size has been set properly, this value should be small.

Innodb_buffer_pool_write_requests: The number writes done to the InnoDB buffer pool.

Innodb_data_fsyncs: The number of fsync() operations so far.

Innodb_data_pending_fsyncs: The current number of pending fsync() operations.

Innodb_data_pending_reads: The current number of pending reads.

Innodb_data_pending_writes: The current number of pending writes.

Innodb_data_read: The amount of data read so far, in bytes.

Innodb_data_reads: The total number of data reads.

Innodb_data_writes: The total number of data writes.
Innodb_data_written: The amount of data written so far, in bytes.
Innodb_dblwr_pages_written: The number of doublewrite operations that have been performed.
Innodb_dblwr_writes: The number of pages that have been written for doublewrite operations.
Innodb_log_wait: The number of times that the log buffer was too small and a wait was required for it to be flushed before continuing.
Innodb_log_write_requests: The number of log write requests.
Innodb_log_writes: The number of physical writes to the log file.
Innodb_pages_created: The number of pages created.
Innodb_pages_read: The number of pages read.
Innodb_pages_written: The number of pages written.
Innodb_row_lock_current_waits: The number of row locks currently being waited for.
Innodb_row_lock_time: The total time spent in acquiring row locks, in milliseconds.
Innodb_row_lock_time_avg: The average time to acquire a row lock, in milliseconds.
Innodb_row_lock_time_max: The maximum time to acquire a row lock, in milliseconds.
Innodb_row_lock_waits: The number of times a row lock had to be waited for.
Innodb_rows_deleted: The number of rows deleted from InnoDB tables.
Innodb_rows_inserted: The number of rows inserted into InnoDB tables.
Innodb_rows_read: The number of rows read from InnoDB tables.
Innodb_rows_updated: The number of rows updated in InnoDB tables.
Key_blocks_used: The number of used blocks in the key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.
Key_read_requests: The number of requests to read a key block from the cache. Key_writes: The number of physical writes of a key block to disk.
Key_reads: The number of physical reads of a key block from disk. If Key_reads is large, then your key_buffer_size value is probably too small. The cache miss rate can be calculated as Key_reads/Key_read_requests.
Key_write_requests: The number of requests to write a key block to the cache.
Key_writes: The number of physical writes of a key block from the MyISAM key cache to the physical disk.
Open_files: The number of files that are open. Open_streams: The number of streams that are open (used mainly for logging).
Open_tables: The number of table cache misses. If the value is large, you probably need to increase table_cache. Typically you would want this to be less than 1 or 2 opened tables per second.
Opened_tables: The number of tables that have been opened. The number of tables that have been opened. If Opened_tables is big, your table_cache value is probably too small.
Qcache_free_blocks: The number of free memory blocks in the query cache.
Qcache_free_memory: The amount of free memory for the query cache.
Qcache_hits: The number of query cache hits.
Qcache_inserts: The number of queries added to the query cache.
Qcache_lowmem_prunes: The number of queries that were deleted from the query cache because of low memory.
Qcache_not_cached: The number of non-cached queries (not cacheable, or not cached due to the query_cache_type setting).
Qcache_queries_in_cache: The number of queries registered in the query cache.
Qcache_total_blocks: The total number of blocks in the query cache.
Questions: The number of statements that clients have sent to the server.
Seconds_Behind_Master: The number of seconds that the replica SQL thread is behind processing the primary binary log.

Select_full_join: Joins performed without keys. This should be zero. This is a good way to catch development errors, as just a few such queries can decrease the system's performance.

Select_full_range_join: The number of joins that used a range search on a reference table.

Select_range: The number of joins that used ranges on the first table. This is normally not a critical issue even if the value is quite large.

Select_range_check: The number of joins without keys that check for key usage after each row. If this is not 0, you should carefully check the indexes of your tables.

Select_scan: Number of queries that performed a full table scan. In some cases these are OK but their ratio to all queries should be constant. If you have the value growing it can be a problem with the optimizer, lack of indexes or some other problem.

Slave_IO_Running: Whether the I/O thread for reading the primary's binary log is running.

Slave_open_temp_tables: The number of temporary tables that the replica SQL thread currently has open.

Slave_SQL_Running: Whether the SQL thread for executing events in the relay log is running.

Slow_launch_threads: The number of threads that have taken more than slow_launch_time seconds to create.

Slow_queries: The number of queries longer than --long-query-time or that are not using indexes. These should be a small fraction of all queries. If it grows, the system will have performance problems.

Sort_merge_passes: The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the sort_buffer_size system variable.

Sort_range: The number of sorts that were done using ranges.

SQL_Delay: The number of seconds that the replica lags behind the primary

Threads_cached: The number of threads in the thread cache.

Threads_connected: The number of currently open connections.

Threads_created: This should be low. Higher values may mean that you need to increase the value of thread_cache or you have the number of connections increasing, which also indicates a potential problem.

Threads_running: The number of threads that are not sleeping.

Oracle Server Metrics

bytes received via SQL*Net from client: Total number of bytes received from the client over Oracle Net Services

bytes received via SQL*Net from dblink: Total number of bytes received from a database link over Oracle Net Services

bytes sent via SQL*Net to client: Total number of bytes sent to the client from the foreground processes

bytes sent via SQL*Net to dblink: Total number of bytes sent over a database link

cluster key scan block gets: The number of blocks obtained in a cluster scan

cluster key scans: The number of cluster scans that were started

commit cleanouts: Total number of times the cleanout block at commit function was performed

commit cleanouts successfully completed: The number of times the cleanout block at commit function completed successfully

Commit SCN cached: The number of times the system change number of a commit operation was cached

consistent changes: The number of times a user process has applied rollback entries to perform a consistent read on the blockWork loads that produce a great deal of consistent changes can consume a great deal of resources. The value of this statistic should be small in relation to the consistent gets statistic.

consistent gets: The number of times a consistent read was requested for a block. See Also: consistent changes and session logical reads statistics

consistent gets direct: The number of times a consistent read was requested for a block bypassing the buffer cache (for example, direct load operation). This is a subset of consistent gets statistics value.
consistent gets from cache: The number of times a consistent read was requested for a block from buffer cache. This is a subset of consistent gets statistics value.

CPU used by this session: Amount of CPU time (in 10s of milliseconds) used by a session from the time a user call starts until it ends. If a user call completes within 10 milliseconds, the start and end user-call time are the same for purposes of this statistics, and 0 milliseconds are added.

CPU used when call started: The CPU time used when the call is started. See Also: CPU used by this session

CR blocks created: The number of CURRENT blocks cloned to create CR (consistent read) blocks. The most common reason for cloning is that the buffer is held in an incompatible mode.

current blocks converted for CR: The number CURRENT blocks converted to CR state

cursor authentications: The number of privilege checks conducted during execution of an operation

data blocks consistent reads - undo records applied: The number of undo records applied to data blocks that have been rolled back for consistent read purposes

db block changes: Closely related to consistent changes, this statistic counts the total number of changes that were part of an update or delete operation that were made to all blocks in the SGA. Such changes generate redo log entries and hence become permanent changes to the database if the transaction is committed. This approximates total database work. It statistic indicates the rate at which buffers are being dirtied (on a per-transaction or per-second basis, for example).

db block gets: The number of times a CURRENT block was requested. See Also: consistent gets

db block gets direct: The number of times a CURRENT block was requested bypassing the buffer cache (for example, a direct load operation). This is a subset of db block gets statistics value.

db block gets from cache: The number of times a CURRENT block was requested from the buffer cache. This is a subset of db block gets statistics value.

DBWR checkpoint buffers written: The number of buffers that were written for checkpoints

DBWR checkpoints: The number of times the DBWR was asked to scan the cache and write all blocks marked for a checkpoint or the end of recovery. This statistic is always larger than background checkpoints completed.

DBWR lrus scans: The number of times that DBWR scans the LRU queue looking for buffers to write. This count includes scans to fill a batch being written for another purpose (such as a checkpoint). This statistic is always greater than or equal to DBWR make free requests.

DBWR revisited being-written buffer: The number of times that DBWR tried to save a buffer for writing and found that it was already in the write batch. This statistic measures the amount of useless work that DBWR had to do in trying to fill the batch. Many sources contribute to a write batch. If the same buffer from different sources is considered for adding to the write batch, then all but the first attempt will be useless because the buffer is already marked as being written.

DBWR transaction table writes: The number of rollback segment headers written by DBWR. This statistic indicates how many hot buffers were written, causing a user process to wait while the write completed.

DBWR undo block writes: The number of rollback segment blocks written by DBWR.

DDL statements parallelized: The number of DDL statements that were executed in parallel.

deffered (CURRENT) block cleanout applications: The number of times cleanout records are deferred, piggyback with changes, always current get.

DFO trees parallelized: The number of times a serial execution plan was converted to a parallel plan.

dirty buffers inspected: The number of dirty buffers found by the user process while it is looking for a buffer to reuse.

DML statements parallelized: The number of DML statements that were executed in parallel.

enqueue conversions: Total number of conversions of the state of table or row lock.

enqueue deadlocks: Total number of deadlocks between table or row locks in different sessions.

enqueue releases: Total number of table or row locks released.

enqueue requests: Total number of table or row locks acquired.

enqueue timeouts: Total number of table and row locks (acquired and converted) that timed out before they could complete.

enqueue waits: Total number of waits that occurred during an enqueue convert or get because the enqueue get was deferred
exchange deadlocks: The number of times that a process detected a potential deadlock when exchanging two buffers and raised an internal, restartable error. Index scans are the only operations that perform exchanges.

execute count: Total number of calls (user and recursive) that executed SQL statements.

free buffer inspected: The number of buffers skipped over from the end of an LRU queue in order to find a reusable buffer. The difference between this statistic and dirty buffers inspected is the number of buffers that could not be used because they had a user, a waiter, or were being read or written, or because they were busy or needed to be written after rapid aging out.

free buffer requested: The number of times a reusable buffer or a free buffer was requested to create or load a block.

gc current block receive time: The total time required for consistent read requests to complete. It records the round-trip time for all requests for consistent read blocks.

gc current block send time: The time it takes to send the current block to the requesting instance over the interconnect.

gcs messages sent: Global Cache Services (GCS) Messages Sent.

ges messages sent: Global Enqueue Services (GES) Messages Sent.

hot buffers moved to head of LRU: When a hot buffer reaches the tail of its replacement list, Oracle moves it back to the head of the list to keep it from being reused. This statistic counts such moves.

immediate (CR) block cleanout applications: The number of times cleanout records are applied immediately during consistent-read requests.

immediate (CURRENT) block cleanout applications: The number of times cleanout records are applied immediately during current gets. Compare this statistic with deferred (CURRENT) block cleanout applications.

index fast full scans (direct read): The number of fast full scans initiated using direct read.

index fast full scans (full): The number of fast full scans initiated for full segments.

index fast full scans (rowid ranges): The number of fast full scans initiated with rowid endpoints specified.

leaf node splits: The number of times an index leaf node was split because of the insertion of an additional value.

lob reads: The number of LOB API read operations performed in the session/system. A single LOB API read may correspond to multiple physical/logical disk block reads.

lob writes: The number of LOB API write operations performed in the session/system. A single LOB API write may correspond to multiple physical/logical disk block writes.

lob writes unaligned: The number of LOB API write operations whose start offset or buffer size is not aligned to the internal chunk size of the LOB. Writes aligned to chunk boundaries are the most efficient write operations. The internal chunk size of a LOB is available through the LOB API (for example, DBMS_LOB.GETCHUNKSIZE()).

logons cumulative: Total number of logons since the instance started. Useful only in V$SYSSTAT. It gives an instance overview of all processes that logged on.

logons current: Total number of current logons. Useful only in V$SYSSTAT.

messages received: The number of messages sent and received between background processes.

messages sent: The number of messages sent and received between background processes native hash arithmetic execute Number of hash operations performed using native arithmetic rather than Oracle NUMBERs.

no buffer to keep pinned count: The number of times a visit to a buffer attempted, but the buffer was not found where expected. Like buffer is not pinned count and buffer is pinned count, this statistic is useful only for internal debugging purposes.

no work - consistent read gets: The number consistent gets that require neither block cleanouts nor rollbacks. See Also: consistent gets.

Number of Small Reads: The total number of physical reads less the number of physical read total multi block requests.

Number of Small Writes: The total number of physical writes less the number of physical write total multi block requests.

opened cursors cumulative: Total number of cursors opened since the instance started. In V$SESSTAT: Total number of cursors opened since the start of the session.

opened cursors current: Total number of current open cursors.

Parallel operations downgraded 1 to 25 pct: The number of times parallel execution was requested and the degree of parallelism
was reduced because of insufficient parallel execution servers.

**Parallel operations downgraded 25 to 50 pct**: The number of times parallel execution was requested and the degree of parallelism was reduced because of insufficient parallel execution servers.

**Parallel operations downgraded 50 to 75 pct**: The number of times parallel execution was requested and the degree of parallelism was reduced because of insufficient parallel execution servers.

**Parallel operations downgraded 75 to 99 pct**: The number of times parallel execution was requested and the degree of parallelism was reduced because of insufficient parallel execution servers.

**Parallel operations downgraded to serial**: The number of times parallel execution was requested but execution was serial because of insufficient parallel execution servers.

**Parallel operations not downgraded**: The number of times parallel execution was executed at the requested degree of parallelism.

**parse count (hard)**: Total number of parse calls (real parses). A hard parse is a very expensive operation in terms of memory use, because it requires Oracle to allocate a workheap and other memory structures and then build a parse tree.

**parse count (total)**: Total number of parse calls (hard and soft). A soft parse is a check on an object already in the shared pool, to verify that the permissions on the underlying object have not changed.

**parse time cpu**: Total CPU time used for parsing (hard and soft) in 10s of milliseconds.

**parse time elapsed**: Total elapsed time for parsing, in 10s of milliseconds. Subtract parse time cpu from this statistic to determine the total waiting time for parse resources.

**physical read bytes**: Total size in bytes of all disk reads by application activity (and not other instance activity) only.

**physical read IO requests**: The number of read requests for application activity (mainly buffer cache and direct load operation) which read one or more database blocks per request. This is a subset of physical read total IO requests statistic.

**physical read total bytes**: Total size in bytes of disk reads by all database instance activity including application reads, backup and recovery, and other utilities. The difference between this value and physical read bytes gives the total read size in bytes by non-application workload.

**physical read total IO requests**: The number of read requests which read one or more database blocks for all instance activity including application, backup and recovery, and other utilities. The difference between this value and physical read total multi block requests gives the total number of single block read requests.

**physical read total multi block requests**: Total number of Oracle instance read requests which read in two or more database blocks per request for all instance activity including application, backup and recovery, and other utilities.

**physical reads**: Total number of data blocks read from disk. This value can be greater than the value of physical reads direct plus physical reads cache as reads into process private buffers also included in this statistic.

**physical reads cache**: Total number of data blocks read from disk into the buffer cache. This is a subset of physical reads statistic.

**physical reads direct**: The number of reads directly from disk, bypassing the buffer cache. For example, in high bandwidth, data-intensive operations such as parallel query, reads of disk blocks bypass the buffer cache to maximize transfer rates and to prevent the premature aging of shared data blocks resident in the buffer cache.

**physical reads prefetch warmup**: The number of data blocks that were read from the disk during the automatic prewarming of the buffer cache.

**physical write bytes**: Total size in bytes of all disk writes from the database application activity (and not other kinds of instance activity).

**physical write IO requests**: The number of write requests for application activity (mainly buffer cache and direct load operation) which wrote one or more database blocks per request.

**physical write total bytes**: Total size in bytes of all disk writes for the database instance including application activity, backup and recovery, and other utilities. The difference between this value and physical write bytes gives the total write size in bytes by non-application workload.

**physical write total IO requests**: The number of write requests which wrote one or more database blocks from all instance activity including application activity, backup and recovery, and other utilities. The difference between this statistic and physical write total multi block requests gives the number of single block write requests.

**physical write total multi block requests**: Total number of Oracle instance write requests which wrote two or more blocks per request to the disk for all instance activity including application activity, recovery and backup, and other utilities.

**physical writes**: Total number of data blocks written to disk. This statistics value equals the sum of physical writes direct and physical writes.
writes from cache values.

**physical writes direct**: The number of writes directly to disk, bypassing the buffer cache (as in a direct load operation).

**physical writes from cache**: Total number of data blocks written to disk from the buffer cache. This is a subset of physical writes statistic.

**physical writes non checkpoint**: The number of times a buffer is written for reasons other than the advancement of the checkpoint. Used as a metric for determining the I/O overhead imposed by setting the FAST_START_IO_TARGET parameter to limit recovery I/Os. (Note that FAST_START_IO_TARGET is a deprecated parameter.) Essentially this statistic measures the number of writes that would have occurred had there been no checkpointing. Subtracting this value from physical writes gives the extra I/O for checkpointing.

**pinned buffers inspected**: The number of times a user process, when scanning the tail of the replacement list looking for a buffer to reuse, encountered a cold buffer that was pinned or had a waiter that was about to pin it. This occurrence is uncommon, because a cold buffer should not be pinned very often.

**prefetched blocks aged out before use**: The number of contiguous and noncontiguous blocks that were prefetched but aged out before use.

**process last non-idle time**: The last time this process executed.

**PX local messages recv’d**: The number of local messages received for parallel execution within the instance local to the current session.

**PX local messages sent**: The number of local messages sent for parallel execution within the instance local to the current session.

**PX remote messages recv’d**: The number of remote messages received for parallel execution within the instance local to the current session.

**PX remote messages sent**: The number of remote messages sent for parallel execution within the instance local to the current session.

**queries parallelized**: The number of SELECT statements executed in parallel.

Read Percent:

**recovery array read time**: Elapsed time of I/O during recovery.

**recovery array reads**: The number of reads performed during recovery.

**recovery blocks read**: The number of blocks read during recovery.

**recursive calls**: The number of recursive calls generated at both the user and system level. Oracle maintains tables used for internal processing. When Oracle needs to make a change to these tables, it internally generates an internal SQL statement, which in turn generates a recursive call.

**recursive cpu usage**: Total CPU time used by non-user calls (recursive calls). Subtract this value from CPU used by this session to determine how much CPU time was used by the user calls.

**redo blocks checksummed by FG (exclusive)**: The number of exclusive redo blocks that were checksummed by the generating foreground processes. An exclusive redo block is the one whose entire redo content belongs to a single redo entry.

**redo blocks checksummed by LGWR**: The number of redo blocks that were checksummed by the LGWR. redo blocks written Total number of redo blocks written. This statistic divided by redo writes equals the number of blocks per write.

**redo blocks written**: This is the total number of redo blocks written. This statistic divided by “redo writes” equals the number of blocks per write.

**redo buffer allocation retries**: Total number of retries necessary to allocate space in the redo buffer. Retries are needed either because the redo writer has fallen behind or because an event such as a log switch is occurring.

**redo entries**: The number of times a redo entry is copied into the redo log buffer redo entries for lost write detection Number of times a Block Read Record is copied into the log buffer.

**redo entries for lost write detections**: Number of times a Block Read Record is copied into the log buffer.

**redo log space requests**: The number of times the active log file is full and Oracle must wait for disk space to be allocated for the redo log entries. Such space is created by performing a log switch.

**redo log space wait time**: Total elapsed waiting time for redo log space requests in 10s of milliseconds.

**redo ordering marks**: The number of times that a system change number was allocated to force a redo record to have a higher SCN than a record generated in another thread using the same block.
redo size: Total amount of redo generated in bytes.

redo size for lost write detection: Total amount of Block Read Records generated in bytes.

redo synch time: Elapsed time of all redo synch writes calls in 10s of milliseconds

redo synch writes: The number of times a change being applied to the log buffer must be written out to disk due to a commit. The log buffer is a circular buffer that LGWR periodically flushes. Usually, redo that is generated and copied into the log buffer need not be flushed out to disk immediately.

redo wastage: The number of bytes wasted because redo blocks needed to be written before they are completely full. Early writing may be needed to commit transactions, to be able to write a database buffer, or to switch logs.

redo write time: Total elapsed time of the write from the redo log buffer to the current redo log file in microseconds.

redo writes: Total number of writes by LGWR to the redo log files. redo blocks written divided by this statistic equals the number of blocks per write.

rollback changes - undo records applied: The number of undo records applied to user-requested.

rollbacks only - consistent read gets: The number of consistent gets that require only block rollbacks, no block cleanouts. See Also: consistent gets.

rows fetched via callback: Rows fetched via callback. Useful primarily for internal debugging purposes.

serializable aborts: The number of times a SQL statement in a serializable isolation level had to abort.

session cursor cache count: Total number of cursors cached. This statistic is incremented only if SESSION_CACHED_CURSORS > 0. This statistic is the most useful in V$SESSTAT. If the value for this statistic in V$SESSTAT is close to the setting of the SESSION_CACHED_CURSORS parameter, the value of the parameter should be increased.

session cursor cache hits: The number of hits in the session cursor cache. A hit means that the SQL statement did not have to be reparsed. Subtract this statistic from parse count (total) to determine the real number of parses that occurred.

session logical reads: The sum of db block gets plus consistent gets. This includes logical reads of database blocks from either the buffer cache or process private memory.

session stored procedure space: Amount of memory this session is using for stored procedures.

Small IO Percent:

sorts (disk): The number of sort operations that required at least one disk write.

sorts (memory): The number of sort operations that were performed completely in memory and did not require any disk writes.

sorts (rows): Total number of rows sorted.

SQL*Net roundtrips to/from client: Total number of Oracle Net Services messages sent to and received from the client.

SQL*Net roundtrips to/from dblink: Total number of Oracle Net Services messages sent over and received from a database link.

summed dirty queue length: The sum of the dirty LRU queue length after every write request. Divide by write requests to get the average queue length after write completion.

switch current to new buffer: Number of times the CURRENT block moved to a different buffer, leaving a CR block in the original buffer table fetch by rowid Number of rows that are fetched using a ROWID (usually recovered from an index).

table fetch by rowid: Number of rows fetched using a ROWID (usually recovered from an index). Table scans by rowid usually indicate either non-optimal queries or tables without indexes. Therefore, this statistic should increase as you optimize queries and provide indexes in the application.

table fetch continued row: Number of times a chained or migrated row is encountered during a fetch.

table scan blocks gotten: During scanning operations, each row is retrieved sequentially by Oracle. This statistic counts the number of blocks encountered during the scan.

table scan rows gotten: The number of rows that are processed during scanning operations.

table scans (cache partitions): The number of range scans performed on tables that have the CACHE option enabled.

table scans (direct read): The number of table scans performed with direct read (bypassing the buffer cache).
table scans (long tables): Long (or conversely short) tables can be defined as tables that do not meet the short table criteria as described in table scans (short tables).

table scans (rowid ranges): During a parallel query, the number of table scans conducted with specified ROWID ranges.

table scans (short tables): Long (or conversely short) tables can be defined by optimizer hints coming down into the row source access layer of Oracle. The table must have the CACHE option set.

transaction lock background get time: Useful only for internal debugging purposes.

transaction lock background gets: Useful only for internal debugging purposes.

transaction lock foreground requests: Useful only for internal debugging purposes.

transaction lock foreground wait time: Useful only for internal debugging purposes.

transaction rollbacks: The number of transactions being successfully rolled back.

transaction tables consistent read rollbacks: The number of times rollback segment headers are rolled back to create consistent read blocks.

transaction tables consistent reads - undo records applied: The number of undo records applied to transaction tables that have been rolled back for consistent read purposes.

user calls: The number of user calls such as login, parse, fetch, or execute

user commits: The number of user commits. When a user commits a transaction, the redo generated that reflects the changes made to database blocks must be written to disk. Commits often represent the closest thing to a user transaction rate.

user rollbacks: The number of times users manually issue the ROLLBACK statement or an error occurs during a user's transactions.

write clones created in background: The number of times a background or foreground process clones a CURRENT buffer that is being written. The clone becomes the new, accessible CURRENT buffer, leaving the original buffer (now the clone) to complete writing.

write clones created in foreground: The number of times a background or foreground process clones a CURRENT buffer that is being written. The clone becomes the new, accessible CURRENT buffer, leaving the original buffer (now the clone) to complete writing.

PostgreSQL Server Metrics

blks_hit: Number of times disk blocks were found already in the buffer cache, so that a physical disk read was not necessary. This only includes hits in the PostgreSQL buffer cache, does not include the operating system’s file system .cache).

blks_read: Number of disk blocks read from the database.

confl_bufferpin: Number of queries in the database that were canceled because of pinned buffers.

confl_deadlock: Number of queries in the database that were canceled because of deadlocks

confl_lock: Number of queries in the database that were canceled due because of timeouts.

confl_snapshot: Number of queries in the database that were canceled because of old snapshots.

confl_tablespace: Number of queries in the database that were canceled because of dropped tablespaces.

numbackends: Number of backends currently connected to the database.

size_mb:

tup_deleted: Number of rows deleted by queries in the database.

tup_fetched: Number of rows fetched by queries in the database.

tup_inserted: Number of rows inserted by queries in the database.

tup_returned: Number of rows returned by queries in the database.

transaction rollbacks: The number of transactions being successfully rolled back.

transaction tables consistent read rollbacks: The number of times rollback segment headers are rolled back to create consistent read blocks.

transaction tables consistent reads - undo records applied: The number of undo records applied to transaction tables that have been rolled back for consistent read purposes.

user calls: The number of user calls such as login, parse, fetch, or execute

user commits: The number of user commits. When a user commits a transaction, the redo generated that reflects the changes made to database blocks must be written to disk. Commits often represent the closest thing to a user transaction rate.

user rollbacks: The number of times users manually issue the ROLLBACK statement or an error occurs during a user's transactions.

write clones created in background: The number of times a background or foreground process clones a CURRENT buffer that is being written. The clone becomes the new, accessible CURRENT buffer, leaving the original buffer (now the clone) to complete writing.

write clones created in foreground: The number of times a background or foreground process clones a CURRENT buffer that is being written. The clone becomes the new, accessible CURRENT buffer, leaving the original buffer (now the clone) to complete writing.
**xact_commit**: Number of transactions in the database that have been committed.

**xact_rollback**: Number of transactions in the database that have been rolled back.

### Sybase ASE and IQ Server Metrics

**active_connections**: The number of users currently connected to the database

**ActiveReq**: Returns the number of server threads that are currently handling a request.

**ActiveVersionsCount**: Count of Active Transaction Versions.

**ActiveVersionsCreateMB**: Size in Mb of versions for active transaction.

**ActiveVersionsDeleteMB**: Size in Mb of versions for active transactions.

**AIOs_delayed_due_to_engine_limit**:

**AIOs_delayed_due_to_os_limit**:

**AIOs_delayed_due_to_server_limit**:

**BytesReceived**: Returns the number of bytes received during client/server communications. This value is updated for HTTP and HTTPS connections.

**BytesReceivedUncomp**: Returns the number of bytes that would have been received during client/server communications if compression was disabled. (This value is the same as the value for BytesReceived if compression is disabled.)

**BytesSent**: Returns the number of bytes sent during client/server communications. This value is updated for HTTP and HTTPS connections.

**BytesSentUncomp**: Returns the number of bytes that would have been sent during client/server communications if compression was disabled. (This value is the same as the value for BytesSent if compression is disabled.)

**CacheFileDirty**: Returns the number of cache pages that are dirty (needing a write).

**CacheFree**: Returns the number of cache pages not being used.

**CacheHits**: Returns the number of database page lookups.

**CachePanics**: Returns the number of times the cache manager has failed to find a page to allocate.

**CachePinned**: Returns the number of pinned cache pages.

**CacheRead**: Returns the number of cache reads.

**CacheReplacements**: Returns the number of pages in the cache that have been replaced.

**CacheScavenges**: Returns the number of times the cache manager has scavenged for a page to allocate.

**DiskRead**: Returns the number of disk reads.

**mempages_allocated**: The number of memory pages that have been allocated.

**mempages_freed**: The number of memory pages that have been de-allocated.

**OtherVersionsCount**: Shows count of other db versions. These versions will eventually be dropped when they are no longer referenced or referencable by active transactions.

**OtherVersionsMB**: Shows space usage in MB of other db versions. These versions will eventually be dropped when they are no longer referenced or referencable by active transactions.

**ProcessCPU**: Returns CPU usage for the database server process. Values are in seconds. This property is supported on Windows and Unix. This property is not supported on Windows Mobile.

**ProcessCPUSystem**: Returns system CPU usage for the database server process CPU. This is the amount of CPU time that the database server spent inside the operating system kernel. Values are in seconds. This property is supported on Windows and Unix.
This property is not supported on Windows Mobile.

**ProcessCPUUser**: Returns user CPU usage for the database server process. Values are in seconds. This excludes the amount of CPU time that the database server spent inside the operating system kernel. This property is supported on Windows and Unix. This property is not supported on Windows Mobile.

**RequestsReceived**: Requests received by the server.

**TempBufferCapacityCount**: Capacity count of Temporary Buffers.

**TempBufferCapacityMB**: Capacity in Mb of Temporary Buffers.

**TempBufferLockedCount**: The number of Temporary Buffers that are locked.

**TempBufferUsedCount**: The number of Temporary Buffers in use.

**total_bytes_received**: The number of bytes received during client/server communications. This value is updated for HTTP and HTTPS connections.

**total_bytes_sent**: Returns the number of bytes sent during client/server communications. This value is updated for HTTP and HTTPS connections.

**xacts**: The number of transactions.

**yields**: Returns the number of times the Adaptive Server engine(s) yielded to the operating system.
Database Agent Events Reference

**On this page:**
- Agent Diagnostic Events
- JDBC Driver-related Agent Diagnostic Events

**Related pages:**
- Monitor Events

You can see Database Agent events on the **Database Monitoring > Events** window and in the `<agent_home>/logs/db-agent#.log` files. Database Agent events are of type Agent Diagnostic Event, with case-specific messages attached.

### Agent Diagnostic Events

**Initialize/re-initialize the DB collector**

**Summary:** Initialize/re-initialize the DB collector

**Description:** Sent when database collector has successfully started. This happens when you configure a new collector, start/restart the agent process (in which case you'll see this event once for each collector), or when the collector recovers from a failure to communicate with the monitored database (for example, when the database goes down for a while, then comes back).

**Severity:** Info

**Initialize/re-initialize the hardware metric collector**

**Summary:** Initialize/re-initialize the hardware metric collector

**Description:** Same as the above, but refers to a hardware collector, not a database collector. This is sent when hardware collector has successfully started. This happens when you configure a new collector, start/restart the agent process (in which case you'll see this event once for each collector), or when the collector recovers from a failure to communicate with the monitored hardware (for example, when the system goes down for a while, then comes back).

**Severity:** Info

**Server parameter [?] has been changed from [?] to [?]**

**Summary:** Server parameter [?] has been changed from [?] to [?]

**Description:** The Database Agent constantly monitors database configuration parameters. When a parameter changes, which generally means a database administrator made a deliberate change, the agent sends this event. AppDynamics Database Visibility users can use these events to track configuration changes in their databases.

**Severity:** Info

### JDBC Driver-related Agent Diagnostic Events

**Summary:** The text of the message is the error from the vendor's JDBC driver.

**Description:** When the agent fails to collect data from a monitored database, it sends this event, and goes into sleep mode for one minute. It will then retry contacting the database every minute until it succeeds. Once it succeeds, it will send an Agent Event "initialize/re-initialize" message, as described above. This happens when the database goes down, or when you add a collector with a wrong password, or wrong hostname, etc.

When the Collector is in sleep mode because of an error, you'll see an indicator next to the database on the Infrastructure and Infrastructure > Databases, on Configure > Collectors window. You will also see the full text of the error in the Infrastructure > Events window.
Severity: Error
Wait State Filtering

Database wait states are pauses or delays in database activity. While Database Visibility collects data on all wait states, you may be more interested in some wait states than others. You can specify which wait states that you want to exclude from being reported by the agent so that you can focus on the critical wait states.

To specify wait states to exclude, navigate to Configuration > Wait State Filtering. Check the boxes of the wait states that you want to exclude. The wait states you exclude are no longer displayed in the Top 10 Query Wait States metric. In addition, any future queries that experience the wait states you exclude are not reported by the agent.
Configuring Custom Metrics

With Database Visibility, you can extend monitoring by specifying SQL queries that you want to run on the monitored database in addition to the queries run during normal database activity. You can schedule these custom queries to run at regular intervals and collect the results in a custom metric.

To specify custom queries to run:

1. Navigate to **Configuration > Custom Metrics > New.**
2. Complete the following fields:
   - **Name**: The name you want to name your custom metric. Once you create a custom metric/event, you cannot change its name.
   - **Custom Query Metric Type**: The type of result that you want from the query.
     - Select the **Metric** type to generate metrics for which health rules can be defined. Metrics of this type are displayed in the metric browser or in the **Custom Metrics** tab of each collector.
     - Select the **Event Data** type to generate custom events whenever the custom query output is non-empty. If you want to be notified when a custom event occurs, you can create a policy that is triggered by that custom event. Custom events are displayed in the **Events** tab. If the custom query outputs multiple rows, the first 40 rows will be included in the event details. Each row displays a maximum of 5000 characters, including the column separators (|).
   - **Database Type**: The database platform that you want to run the metric on.
   - **Databases**: The database instances that you want to run the metric on. You can run the metric on all database instances of the specified database type, or on specific database instances that you specify.
   - **Schedule**: The time interval at which you want to run the query. For metric type custom queries, the metric value reported in between the specified intervals is the value observed in the previous query execution.
   - **Query Text**: The query that you want to execute. If you are creating a custom metric of Metric type, its query must have one of the following return types:
     - Positive integer. For example, the query below returns a positive integer.
       ```sql
       SELECT COUNT(*) FROM employees
       ```
     - String and positive integer. For example, the query below returns a string and a positive integer.
       ```sql
       SELECT name, salary FROM employees
       ```

When you create a custom query, it appears for all collectors, but its data only reflects the collector that you created it for.
You can test the semantics of the query and also validate the results by clicking the Test Query button.

**Note**
The results are validated only if the database agent version is greater than 4.5.5.

The following limits apply when configuring custom metrics:

- The total number of custom metrics or events that you can create per account is the product of total number of Collectors and \( \text{dbmon.config.max.custommetric} \). By default, the value of \( \text{dbmon.config.max.custommetric} \) is 20. You can adjust the limit up to 50 using the \( \text{dbmon.config.max.custommetric} \) property in the Controller settings.
- Up to 40 custom metrics can be reported per collector per minute.

Custom metrics are supported for all relational databases: MySQL, Microsoft SQL Server, Oracle, PostgreSQL, DB2, and Sybase.

**Viewing Custom Metrics**

You can view your custom metrics in the following locations:

- **Navigate to Databases > Metric Browser** and expand the database that you want to view custom metrics for.

- In Databases, open a database instance, then click the **Custom Metrics** tab.
Configure Query Literals Security

In SQL, literals are the string or integer values representing data retrieved from the database. By default, literals are removed from SQL queries, as they can contain sensitive user data such as social security numbers, CC numbers, etc.

If you want to show literals in the queries, you can disable the Remove literals from the queries option.

1. Click Databases.
2. Click Configuration.
3. In the Security section, uncheck Remove literals from the queries.

Once you uncheck the option, literals will always be displayed in the queries. You can revert to hiding the literals by checking the option.