IDERA® | SQL Server Whitepaper

ALERTING: FINDING THE SIGNAL IN THE NOISE

BY SCOTT STONE

INTRODUCTION

A key ingredient to managing security or performance for database administrators is having well configured alerting. It is unfortunate that any software which generates alerts can have both false positives and false negatives.

We define a false positive as a test result which is incorrect when it shows that a particular condition or attribute is present. For example, a false positive refers to alerting on an exceeded default threshold, but which the system did not account for using baselining. Refer to the section “Baseline Alerts”, below. The dangers of false positives will cause a database administrator to investigate alerts which are not actual issues leading to wasted effort.

We define a false negative as a test result which is incorrect when it shows that a particular condition or attribute is absent. For example, a false negative refers to not catching a database intrusion attempt because the number of failed logins did not exceed a certain amount over a defined period. For example, a threshold was set for ten failed logins in a five-minute window, but the attacker only tried nine times. False negatives are trickier to handle because they may mislead a database administrator into thinking they do not have a problem when they do. This could lead to unanticipated downtime, application impacts, or worse, a security breach.

False positives and false negatives can lead to a loss of confidence in the monitoring software being used. In such cases, the problem may only be the lack of a tuned alerting system.

STRATEGIES TO REDUCE FALSE POSITIVES AND FALSE NEGATIVES WITHOUT UNDUE RISK

The first step in a strategy to reduce false positives and false negatives is to realize that ignoring certain alerts is not the answer. In such cases, we need better alert configuration to automate accurate monitoring of alerts. Here are several basic strategies for reducing false positives and false negatives:

- Do not exclude or disable too many alerts, otherwise you may miss something important.
- Set good default thresholds.
- Account for alert-dependent exceptions.
- Filter for alert noise.
- Use baselining and then adjust alert thresholds in accordance.

SET GOOD DEFAULT THRESHOLDS

Starting off with good default alert thresholds is always a good idea. Database management software should always come with a suitable set of pre-set defaults. However, the database administrator needs to always review those defaults since they have more intimate knowledge of operating their databases and the applications that use them. Here are some general guidelines:

- Set alert thresholds according to the version of the database vendor and edition being monitored.Set alert thresholds depending on the criticality of the application using the database.
- Set alert thresholds for specify monitoring categories such as backup, databases, logs, and resources.

ACCOUNT FOR ALERT-DEPENDENT EXCEPTIONS

Database administrators need to know that some alerts have dependent exceptions. For every raised alert, there may be other related alerts which get raised. These dependencies tend to imply there may be a root cause. The database administrator needs to use their experience and knowledge to help determine this root cause. Next, the database administrator needs to focus on the alert or alerts associated with the root cause rather than chase the dependent exceptions.

FILTER FOR NOISE

Once a database administrator better understands a good alert configuration, they can eliminate excessive alerts. Some general rules for filtering noise are:

- Finer precision monitoring will return more natural fluctuations and therefore the system must be average the data over several observations to remove spikes and dips.
- Exclude or disable alerts for certain events or performance conditions.
- Create consolidation rules for alerts which occur a lot during a defined period, so instead of getting multiple alerts, you only get one.

BASELINE ALERTS

Baseline metrics are time-lagged calculations using an averaging over different times of the day or days of the week. These baseline metrics provide a basis for making comparisons of past performance to current performance. Baseline alerts would use these calculated metrics to set thresholds when monitoring database performance based on those baselines. As a last step, always give your baselining more observation time to avoid false positives.

The database administrator may use baselines at regular intervals to tune static thresholds more in a more appropriate way for specific environments or use them in a dynamic way to change thresholds in an automated fashion with changing conditions. Dynamic baseline alerting should always account for the phenomenon of slow changes being undetected as the system adjusts the baselines by itself.

For more information on using baselines, refer to the whitepaper “Detect baseline anomalies”.

PRIORITIZE ALERTS

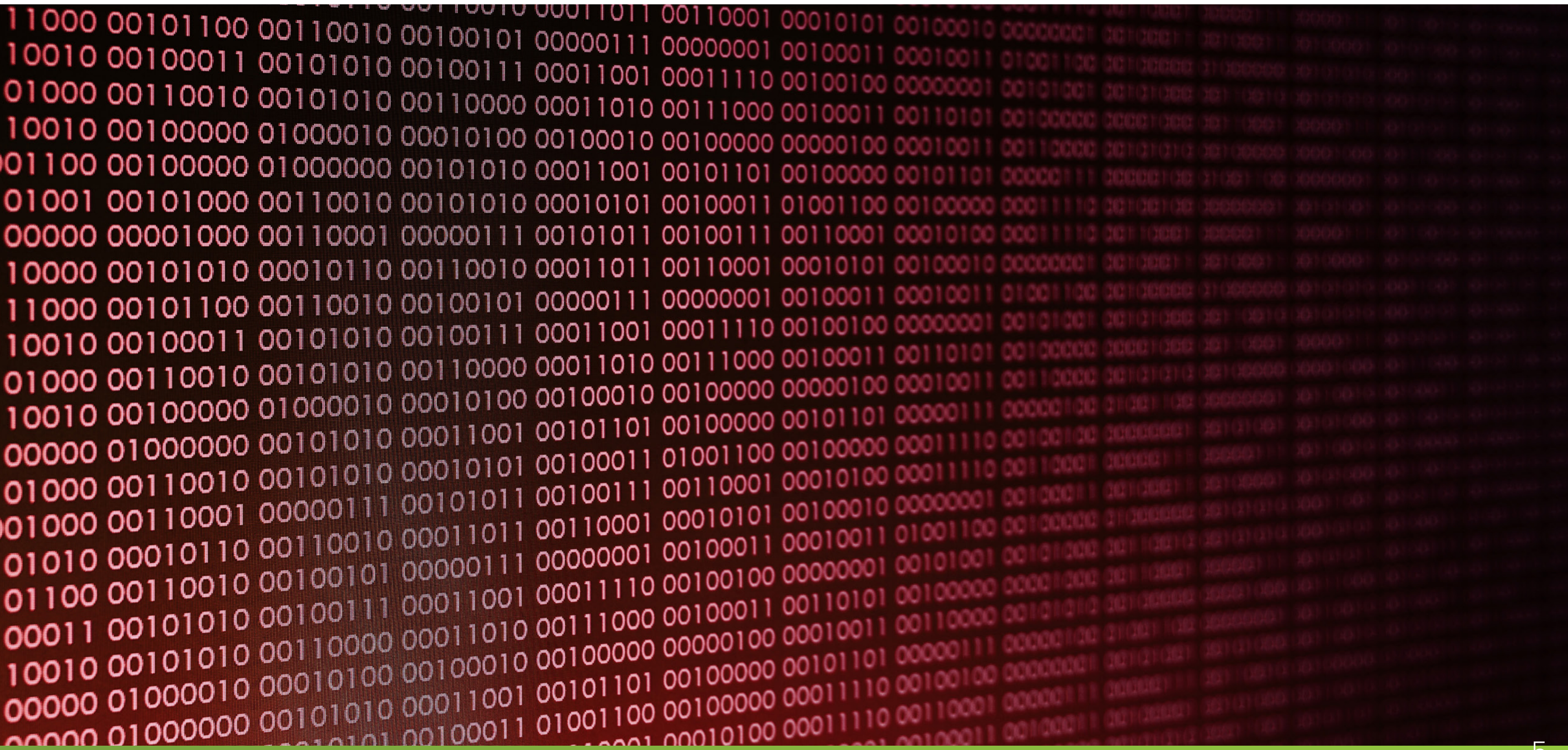
The key to effective database monitoring is to prioritize alerts which database monitoring tools generate. General classification alerts may include high, medium, or low-priority or some other finer grained ranking system. Start with broad priority classifications and next move to prioritize within those broad classifications as needed.

High-priority alerts should include anything which is critical. For example, running out of disk space on a database server or failed backups which would cause the inability to perform recovery.

Medium-priority alerts are difficult to define since they differ for every organization. It is rare when these alerts require immediate attention, but you should address them in a reasonable timeframe. For example, under or over utilized processors which may affect performance may show a need to re-provision the hardware.

Low-priority alerts may be noteworthy to review once we address all the high and medium alerts, but it is rare when these alerts affect security or performance.

Alert prioritizations require proper notification planning, so the right people get notified depending on the problem, the severity, and the priority of the alert. Always use some default alert prioritization to begin and next adjust as needed.



ESCALATION STRATEGY

A good escalation strategy defines how organizations handle alerts. The strategy should outline who the system should notify:

- when an alert occurs
- when escalation occurs
- when an alert remains stagnant for a defined period
- based on the on-call schedule
- based on the severity level, duration, and scope

As part of the alert escalation strategy, consider defining a process for when to bump up the priority for unresolved alerts. As a result, these alerts get addressed well. However, do not to raise the priority of non-critical alerts.

The size of database organizations and the complexity of ecosystems determine how much detail you need.

TUNE THRESHOLDS

Database monitoring tools tend to come with pre-defined thresholds and priorities, so alerting configuration follows best practices. However, database administrators should always strive to tune alert thresholds and priorities based on their knowledge of the database technology being used and the applications which use these databases.

The two basic methods of tuning thresholds should be:

1: Review pre-set thresholds to determine whether they seem right.

2: Establish reliable baselines and next adjust alert thresholds based on those results.

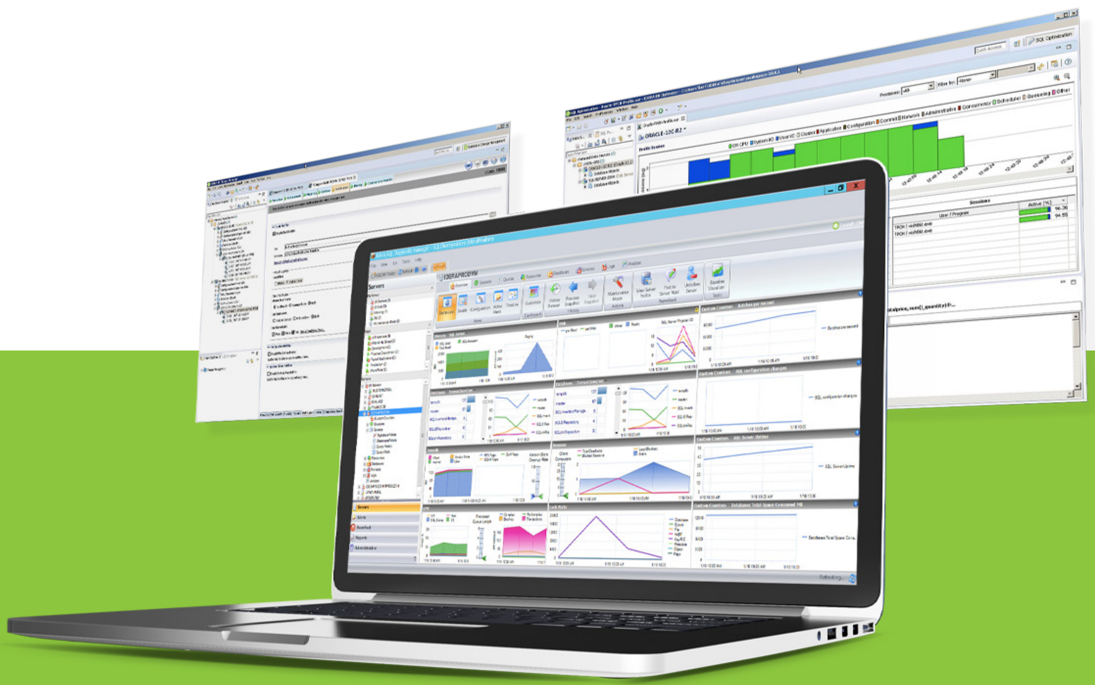
The goal of tuning alert thresholds is to reduce the number of alerts which database administrators need to deal with daily. Well managed alerts result in database administrators being able to use their time in a more productive way to handle more of the top line organizational business goals and not fighting fires.

IDERA'S SOLUTION FOR MONITORING MICROSOFT SQL SERVER

Quickly find and fix performance problems for SQL Server, Azure SQL Database, and Amazon RDS for SQL Server with SQL Diagnostic Manager:

- Monitor physical, virtual, and cloud environments.
- Track queries and plans to fix blocks and locks.
- Alert predictively and avoid false alerts.
- View expert advice with executable scripts.

START FOR FREE



ABOUT THE AUTHOR

The author, Scott Stone, manages IDERA'S database performance management products and has over twenty years of experience in product management and product marketing in the software and technology industry from small start-ups to Fortune 500 companies. For the past fifteen years, Scott has focused on development of database performance and security products at various companies. Earlier in his career, Scott was a software engineer in the space and defense industry. Scott holds an MBA from Rice University as well as bachelor's and master's degrees in electrical engineering from the Georgia Institute of Technology.