IDERA® | Database Tools Whitepaper

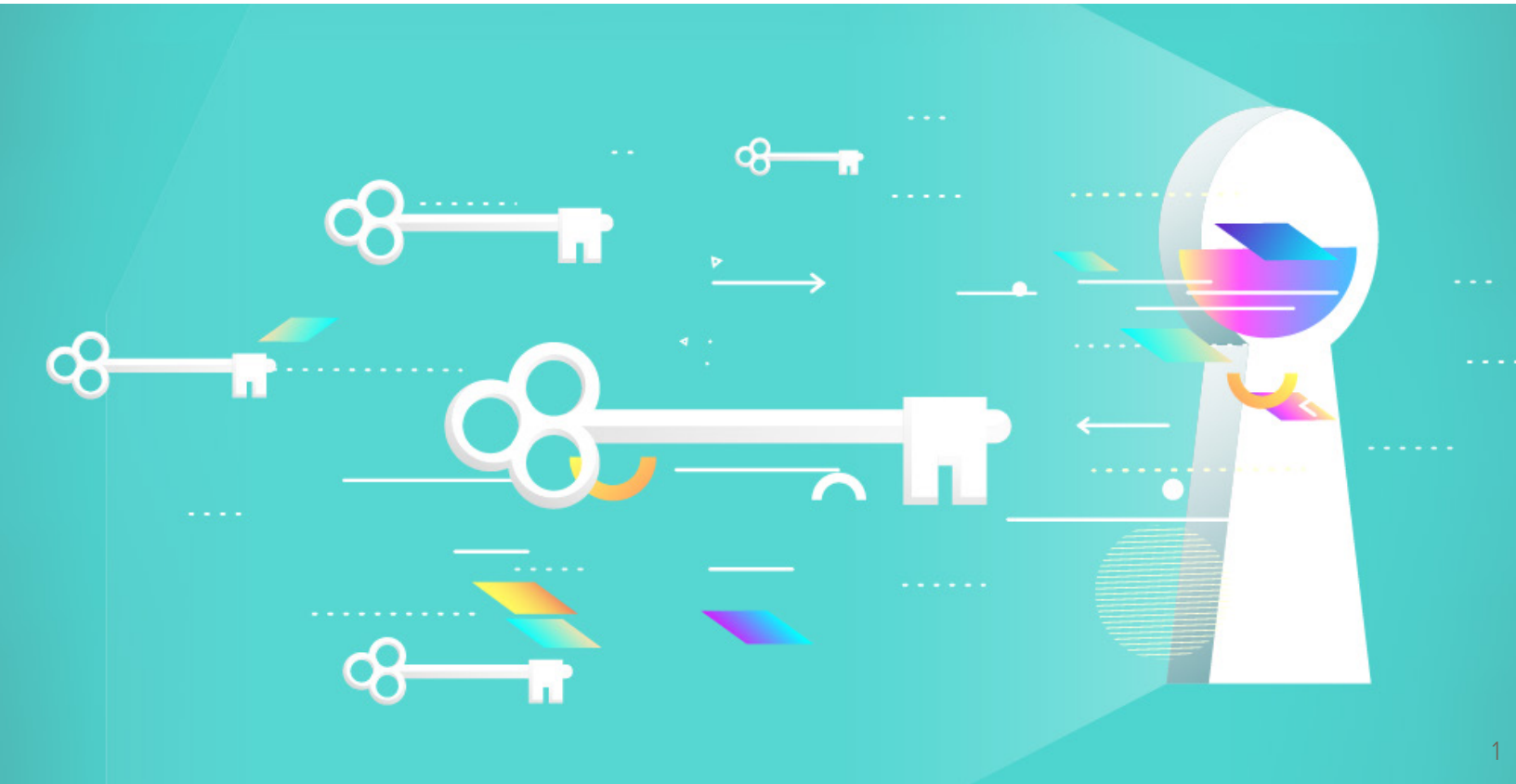
THE KEYS TO A HIGH-PERFORMING DATABASE

INTRODUCTION

Performance is one of the most critical characteristics of enterprise databases and their associated applications. The inability to meet performance expectations can doom an otherwise excellent solution to failure. Internal systems will suffer from misuse and poorly performing public database applications may drive current and prospective customers to search for alternatives.

It can be challenging to design, implement, and maintain a high-performing database. In this whitepaper, we will be discussing performance best practices used during system development, and after the database goes into production. In some cases, we will use examples of recommended guidelines for specific database platforms like SQL Server. The standards apply to other databases in the same class. Methods that are useful for one flavor of a relational database will usually work for others with some minor variations.

The decisions made in the planning, design, and development of a database have a tremendous impact on the functionality of the finished product. Performance optimization can be performed after database applications have been implemented, but it is more productive to address as many issues as possible before the system goes live. Changes in usage patterns and the computing environment may necessitate performance tuning throughout the life of a database.



DATA STRUCTURE AND DATABASE SELECTION

An essential first step is to understand the data that the database and application will process. The database that is selected for a particular application can have a tremendous impact on the ability of the system to perform up to expectations. Well-structured data is a prime candidate for one of the many SQL-based relational databases on the market.

When processing documents or more variable structured data such as emails or social media feeds, a NoSQL database is a more appropriate platform choice. These databases are designed to make it easier to handle unstructured data. Choosing the wrong solution will make it impossible to achieve peak performance and may result in a failed application.

HARDWARE CHOICES ARE CRITICAL TO PERFORMANCE

Databases are a combination of physical and logical entities. The hardware components on which a database runs form its foundation. As with any complex structure, an inadequate foundation leads to problems that can bring down the whole building. What may seem like minor issues during system development and testing can grow exponentially to become major impediments to system functionality and performance. As a general rule, avoid making unnecessary concessions regarding the hardware used for a database.

Hardware is a broad category that can be broken down into individual elements that contribute to database performance. In some cases, tuning a single item can have a dramatic effect on the overall system. The recommended parameters are valid with either physical or virtual machines.

STORAGE

A discussion of storage encompasses the type of devices used to store database information as well as the space capacity requirements of a given system.

DEVICE TYPE

The disk subsystem used to support a database implementation needs to have the speed to address its input and output (I/O) requirements. Millions of disk I/O operations can be required to satisfy a single query. Seemingly minor differences in disk access speed can multiply to cause substantial changes in response time.

Enterprise-class solid-state drives (SSDs) and redundant arrays of independent disks (RAID) will provide better performance than traditional hard disk drives (HDDs). SSDs are faster and use less energy than HDDs but are slightly less reliable. Strictly from a performance perspective, SSDs will provide better database response time by reducing latency.

Taking the next step and using RAID implementations helps solve potential reliability issues while maintaining high I/O speed. RAID comes in multiple flavors that offer different features designed to improve reliability and speed. Mirroring is a technique that fully replicates data to guard against disk failure. Striping data across two or more disks improves performance when writing data by allowing more information to be written simultaneously. Using mirroring and striping together offers the best combination of performance and reliability in a disk subsystem.

Here are some recommendations for implementing RAID arrays with SQL Server databases.

- RAID 10 should be considered for databases with high I/O requirements. Mirroring will reduce the usable storage capacity by 50%.
- RAID 5 and RAID 6 provide redundancy with parity blocks that slow down the speed of write operations. They provide exceptional read performance and are suitable for databases that are primarily used to read data.
- Segregating log files on a RAID array separate from data files will improve overall database performance.

It is more expensive to implement RAID or purchase SSDs to replace the HDDs that may have been allocated for a database. Business decisions will influence whether the performance gains are worth the capital expense.

STORAGE CAPACITY

Regardless of the type of disk subsystem that is used for database storage, meeting the capacity sizing guidelines is a critical aspect of attaining optimum performance. We will look at the recommended amount of storage necessary for SQL Server instances. These are several distinct SQL Server components that need to be considered when allocating space resources. All of these storage elements scale linearly as the size of a database grows.

- **Database files** - A good rule is that every 1,000 nodes require a gigabyte of storage. Databases with 20,000 nodes need at least 20 GB while a 100,000 node system requires a minimum of 100 GB for its database files.
- **Transaction logs** - The storage capacity provisioned for transaction logs should be 20% of the size of the database files.
- **tempdb** - This important file needs to be sized at 10% of that used for database files.
- **Backup files** - The same amount of space allocated to database files needs to be made available for their backup.

Based on these guidelines, an SQL Server with 20,000 nodes requires a minimum of 20 GB for database files, 4 GB for the transaction log, 2 GB for tempdb, and another 20 GB for backups to provide good performance.

MEMORY AND CPU

The amount of random access memory (RAM) that a database requires depends on several factors, including:

- **The number of concurrent users** - When more than 15 simultaneous users will be accessing and retrieving data, an additional 1 to 2 GB of RAM should be allocated for every five users.
- **Execution packages** - Databases that are updated frequently benefit from additional RAM. In some cases, doubling the amount of available RAM can cut execution time by 50%**.**
- **Database size** - A database will exhibit better performance if it has at least the minimum recommended amount of RAM available. That may vary based on the chosen platform. These are guidelines that apply for an SQL Server implementation.
    - Small production databases < 250 GB - 8 GB of RAM
    - Medium production databases < 1 TB - 8 GB of RAM
    - Databases that are up to 2 TB in size - 32 GB of RAM
    - Databases that are between 2 and 5 TB in size - 64 GB of RAM
    - Databases over 5 TB in size will improve caching speed with more than 64 GB of RAM
- **Expected rate of growth** - When provisioning RAM, database teams need to consider future growth and allow for expanding memory locations if requirements demand it.

Processor speed and size can affect database performance. Individual software vendors have recommendations that usually provide a list of specific hardware platforms on which to run their databases. Using 4 to 8 cores is appropriate for smaller databases, with larger ones benefiting from 8 to 24 cores.

NETWORK CONSIDERATIONS

Slow network connections can be at the root of database performance issues. The speed of the network hardware may be beyond the scope of issues that can be directly addressed by a database team. They can, however, implement more efficient use of network resources, which can improve performance.

One area that presents optimization possibilities is how the database handles connections. Creating connections is an expensive operation. It takes time and requires system resources that could be better used elsewhere. Connection and thread pools can help minimize the expense of continually recreating these items.

Connection pools establish several client/server connections which are dynamically allocated on demand when needed by the client application. In a thread pool, threads are reused for requests and responses rather than creating and destroying them as required. Both techniques can be used to improve database performance.

CODING AND QUERY OPTIMIZATION

Provisioning or upgrading the hardware of a database server should result in improvements in performance. Further gains can be found by optimizing the code and queries of a database. There are many aspects of queries that can be investigated for possible optimization. Here are some of the most productive elements of database queries that can be addressed to increase performance.

PRIORITIZE THE QUERIES FOR OPTIMIZATION

Before embarking on a quest to optimize queries, it is essential to know which queries get used most often and which ones cost the most in system resources. It may be that addressing a small subset of queries can dramatically improve performance. Less critical queries can be optimized when time permits.

OPTIMIZE INDEXES

Indexes are crucial logical constructs in both relational and NoSQL databases. The way they are designed and used can have a tremendous impact on the speed with which the database can carry out a query and return its results. Having too few or too many indexes leads to different types of performance degradation.

Two techniques to consider are the creation of composite and clustered indexes. A composite index contains more than one field and is useful for queries that contain multiple fields in its WHERE clause. Clustered indexes determine the physical order of data in a table and are very efficient on columns that are searched for a range of values. Some database platforms use different terminology to refer to these indexing techniques.

MINIMIZE SQL STATEMENT PARSING

Application code should be optimized to reduce the amount of parsing required. Parsing SQL statements is CPU-intensive activity. One way to minimize parsing is to use stored procedures that are kept in a parsed form, which reduces runtime processing.

PROCESS MULTIPLE ROWS

Whenever possible, it is advisable to work with multiple rows simultaneously. Optimized performance is achieved when a single SQL statement is used to process all rows and perform the required operations on them. Only one network round-trip is necessary for this scenario, reducing the occurrence of bandwidth and latency issues.

KEEP DATABASE FILES ON SEPARATE DISKS

A relatively painless performance fix can be obtained by the effective use of multiple disks for database files. Storing all files on the same disk will negatively affect performance as the subsystem struggles to satisfy read and write requests. At a minimum, tempdb should be located on a different drive than the rest of the database components. The best practice is also to keep data, log, and backup files on discrete disks.

This structure should be implemented during the planning and development stages of a database. Moving the files can result in some immediate and substantial performance gains in cases where this was not done.

USING THE MOST RECENT DATABASE VERSION

Upgrading systems to the most current version of database software can contribute to improved performance. Code optimization and additional features performed by the vendor may result in better response time and happier users. While performing an upgrade may not be a quick fix, it can help keep systems performing efficiently.

USE NATIVE DATABASE FEATURES

Making use of the native database features is preferred over re-implementing them in a homegrown manner. Use the replication, auditing, database integrity checks, and other features that were put in place by the software development team of a database vendor.

There are many aspects of a database application that need to work in concert to provide peak performance. This fact presents multiple opportunities for database teams to address the bottlenecks affecting the response time of a system. A systematic review of the outlined areas can be instrumental in discovering modifications that can be made to provide the type of performance envisioned when the database was designed and implemented.

IDERA'S SOLUTION

Try IDERA'S SQL Diagnostic Manager

24X7 SQL PERFORMANCE MONITORING, ALERTING AND DIAGNOSTICS

- Performance monitoring for physical, virtual, and cloud SQL Servers
- Deep query analysis to identify excessive waits and resource consumption
- History browsing to find and troubleshoot past issues
- Adaptive & automated alerting with 100+ pre-defined and configurable alerts . Capacity planning to see database growth trends and minimize server sprawl . SCOM management pack for integration with System Center

Start a free, fully-functional, 14-day trial today!

Start for FREE