BANNARI AMMAN INSTITUTE OF TECHNOLOGY
An Autonomous Institution Affiliated to Anna University Chennai - Approved by AICTE - Accredited by NAAC with "A+" Grade
SATHYAMANGALAM - 638 401 ERODE DISTRICT TAMIL NADU INDIA
Ph: 04295-226000 / 221289 Fax: 04295-226666 E-mail: stayahead@bitsathy.ac.in Web: www.bitsathy.ac.in
Stay Ahead

22CD405-DATABASE MANAGEMENT SYSTEMS

Unit 5&LP4-IN-MEMORY DATABASES AND CACHING,DATABASE SECURITY AND ENCRYPTION DATABASE PERFORMANCE TUNING

1.WHAT IS AN IN-MEMORY DATABASE?

An in-memory database is a data storage software that holds all of its data in the memory of the host. The main difference between a traditional database and an in-memory database relies upon where the data is stored. Even when compared with solid-state drives (SSD), random access memory (RAM) is orders of magnitude faster than disk access. Because an in-memory database uses the latter for storage, access to the data is much faster than with a traditional database using disk operations.

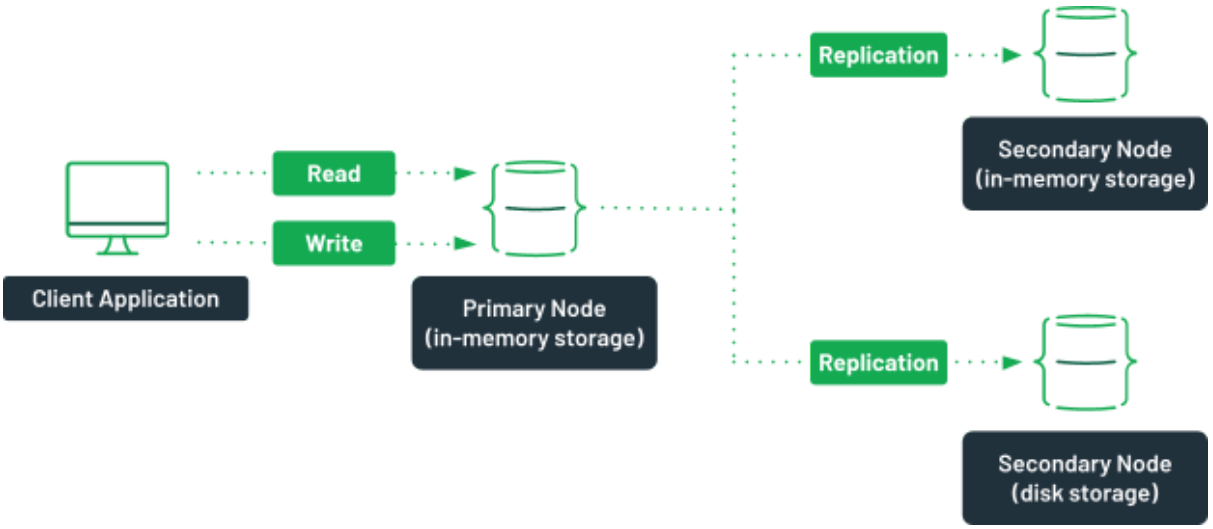
In-memory databases provide quick access to their content; on the downside, they are at high risk of losing data in case of a server failure, since the data is not persisted anywhere. If a server failure or shutdown should occur, everything currently in the memory of that computer would be lost due to the volatile nature of RAM. It is also worth noting that the cost of memory is much higher than the cost of hard disks. This is why there is typically much more hard disk space than memory on modern computers. This factor makes in-memory databases much more expensive. They are also more at risk of running out of space to store data.

Your decision to use an in-memory database would depend on your use case. In-memory databases are great for high-volume data access where a data loss would be acceptable. Think of a large e-commerce website. The information about the products is crucial and should be kept on a persisted storage, but the information in the shopping cart could potentially be kept in an in-memory database for quicker access.

1.1 How does an in-memory database work?

An in-memory database works in a very similar way as any other database, but the data is kept in RAM rather than on a traditional disk. Replacing the disk access with memory operations highly reduces the latency required to access data.

Using RAM as a storage medium comes with a price. If a server failure occurs, all data will be lost. As a way to prevent this, replica sets can be created in modern databases such as MongoDB with a mix of in-memory engines and traditional on-disk storage. This replica set ensures that some of the members of the cluster are persisting data.



A replica set with both in-memory and traditional storage.

In this scenario, possible with MongoDB Enterprise Advanced, the primary node of the replica set uses an in-memory storage engine. It has two other nodes, one of which uses an in-memory storage, the other one using the WiredTiger engine. The secondary node using the disk storage is configured as a hidden member.

In case of a failure, the secondary in-memory server would become the primary and still provide quick access to the data. Once the failing server comes back up, it would sync with the server using the WiredTiger engine, and no data would be lost.

Many in-memory database offerings nowadays offer in-memory performance with persistence. They typically use a configuration similar to this one.

Similar setups can be done with sharded clusters when using MongoDB Enterprise Advanced.

1.2 What are the advantages and disadvantages of in-memory databases?

The most obvious advantage of using an in-memory database is the speed to retrieve data from the database. Without the need of performing disk operations, the latency is reduced greatly and is more consistent. Because there are no more reasons to limit the number of reading operations on a disk, different algorithms can be used to search data, increasing the overall performance of an in-memory database.

It might seem like a great idea to use exclusively in-memory databases to benefit from the speed gains, but there are some drawbacks. First, the cost of RAM is about 80 times higher than the price of traditional disk space. This would significantly increase the operating costs of your infrastructure, whether the servers are hosted in the cloud or on-premises.

Secondly, the lack of data persistence in case of a failure can be an issue in some cases. While there are ways to mitigate the risks associated with these data losses, those risks might not be acceptable for your business case.

Finally, because servers typically have much less RAM than disk space, it is not uncommon to run out of space. In this case, write operations would fail, losing any new data stored.

Using an on-disk database with an NVMe SSD can prove to be a solid alternative to in-memory databases. These disks offer a data bandwidth similar to RAM, although the latency is slightly higher.

1.3 Why use an in-memory database?

The main use case for in-memory databases is when real-time data is needed. With its very low latency, RAM can provide near-instantaneous access to the needed data. Because of the potential data losses, in-memory databases without a persistence mechanism should not be used for mission-critical applications.

Any application where the need for speed is more important than the need for durability could benefit from an in-memory database over a traditional database.

In many cases, the in-memory database can be used only by a small portion of the total application, while the more critical data is stored in an on-disk database such as MongoDB Atlas.

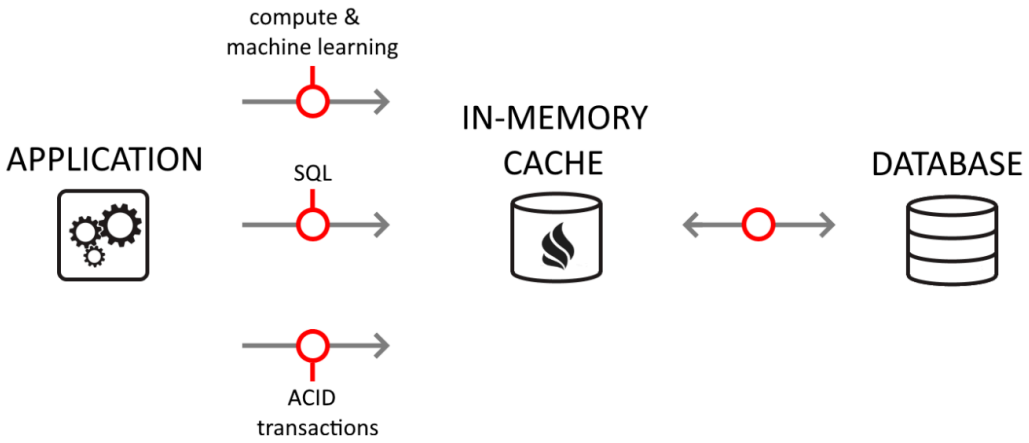
1.4 In-memory database examples

In-memory databases can find their place in many different scenarios. Some of the typical use cases could include:

- IoT data: IoT sensors can provide large amounts of data. An in-memory database could be used for storing and computing data to later be stored in a traditional database.
- E-commerce: Some parts of e-commerce applications, such as the shopping cart, can be stored in an in-memory database for faster retrieval on each page view, while the product catalog could be stored in a traditional database.
- Gaming: Leaderboards require quick updates and fast reads when millions of players are accessing a game at the same time. In-memory databases can help to sort the results more quickly than traditional databases.
- Session management: In stateful web applications, a session is created to keep track of a user identity and recent actions. Storing this information in an in-memory database avoids a round trip to the central database with each web request.

2. What is an In-Memory Cache?

An in-memory cache is a data storage layer that sits between applications and databases to deliver responses with high speeds by storing data from earlier requests or copied directly from databases. An in-memory cache removes the performance delays when an application built on a disk-based database must retrieve data from a disk before processing.



Reading data from memory is faster than from the disk. In-memory caching avoids latency and improves online application performance. Background jobs that took hours or minutes can now execute in minutes or seconds.

Since cache data is kept separate from the application, the in-memory cache requires data movement from the cache system to the application and back to the cache for processing. This often occurs across networks.

3. TOOLS FOR SUPPORTING IN MEMORY DATABASES AND CACHE

In-memory databases and caches play a crucial role in improving the performance of applications by storing and retrieving data in the system's main memory rather than relying on slower disk-based storage. Several tools and technologies support the implementation and management of in-memory databases and caches. Here are some commonly used tools:

Redis:

Description: Redis is an open-source, in-memory data structure store. It supports various data structures such as strings, hashes, lists, sets, and more.

Use Cases: Caching, real-time analytics, messaging, leaderboards, and other scenarios where low-latency data access is critical.

Key Features: Persistence options, built-in replication, support for transactions, and a variety of data types.

Memcached:

Description: Memcached is a distributed, in-memory caching system that is simple yet powerful. It stores key-value pairs in memory and is often used to speed up dynamic web applications.

Use Cases: Caching frequently accessed data, session storage, and database query result caching.

Key Features: Distributed architecture, simple API, and support for multiple programming languages.

Hazelcast:

Description: Hazelcast is an open-source, in-memory data grid and computing platform. It provides distributed data structures and supports caching and computation across a cluster.

Use Cases: Distributed caching, distributed computing, and real-time analytics.

Key Features: Distributed data structures (maps, queues, etc.), distributed computing capabilities, and support for multiple programming languages.

Apache Ignite:

Description: Apache Ignite is an open-source, distributed in-memory computing platform. It provides an in-memory data grid, caching, and computation capabilities.

Use Cases: Real-time analytics, distributed caching, and high-performance computing.

Key Features: Distributed in-memory storage, SQL queries, distributed computing, and integration with various data sources.

Microsoft Azure Redis Cache:

Description: Azure Redis Cache is a fully managed, in-memory caching service provided by Microsoft Azure. It is built on the open-source Redis.

Use Cases: Caching, session storage, and improving the performance of Azure applications.

Key Features: Fully managed service, support for Redis commands, and integration with Azure services.

Oracle TimesTen:

Description: Oracle TimesTen is an in-memory relational database management system (RDBMS). It is designed for high-performance transaction processing.

Use Cases: Real-time data processing, low-latency applications, and caching.

Key Features: Full ACID compliance, in-memory storage, and seamless integration with Oracle Database.

SAP HANA:

Description: SAP HANA is an in-memory, column-oriented, relational database management system. It is designed for both analytical and transactional processing.

Use Cases: Real-time analytics, data warehousing, and high-performance transactional applications.

Key Features: In-memory storage, advanced analytics, and integration with SAP applications.

When choosing a tool for in-memory databases or caches, consider factors such as the specific use case, scalability requirements, ease of integration, and the programming languages and frameworks supported. Each tool has its strengths and may be better suited for particular scenarios.

4. WHAT IS DATABASE SECURITY?

Database security refers to the range of tools, controls, and measures designed to establish and preserve database confidentiality, integrity, and availability. This article will focus primarily on confidentiality since it's the element that's compromised in most data breaches.

Database security must address and protect the following:

- The data in the database
- The database management system (DBMS)
- Any associated applications
- The physical database server and/or the virtual database server and the underlying hardware
- The computing and/or network infrastructure used to access the database

Database security is a complex and challenging endeavor that involves all aspects of information security technologies and practices. It's also naturally at odds with database usability. The more accessible and usable the database, the more vulnerable it is to security threats; the more invulnerable the database is to threats, the more difficult it is to access and use. (This paradox is sometimes referred to as Anderson's Rule. (link resides outside IBM) Why is it important?

By definition, a data breach is a failure to maintain the confidentiality of data in a database. How much harm a data breach inflicts on your enterprise depends on a number of consequences or factors:

- **Compromised intellectual property:** Your intellectual property—trade secrets, inventions, proprietary practices—may be critical to your ability to maintain a competitive advantage in your market. If that intellectual property is stolen or exposed, your competitive advantage may be difficult or impossible to maintain or recover.

- **Damage to brand reputation:** Customers or partners may be unwilling to buy your products or services (or do business with your company) if they don't feel they can trust you to protect your data or theirs.

- **Business continuity (or lack thereof):** Some business cannot continue to operate until a breach is resolved.

- **Fines or penalties for non-compliance:** The financial impact for failing to comply with global regulations such as the Sarbannes-Oxley Act (SAO) or Payment Card Industry Data Security Standard (PCI DSS), industry-specific data privacy regulations such as HIPAA, or regional data privacy regulations, such as Europe's General Data Protection Regulation (GDPR) can be devastating, with fines in the worst cases exceeding several million dollars *per violation*.

- **Costs of repairing breaches and notifying customers:** In addition to the cost of communicating a breach to customer, a breached organization must pay for forensic and investigative activities, crisis management, triage, repair of the affected systems, and more.

4.1 Common Threats and Challenges

Many software misconfigurations, vulnerabilities, or patterns of carelessness or misuse can result in breaches. The following are among the most common types or causes of database security attacks and their causes.

Insider threats
An insider threat is a security threat from any one of three sources with privileged access to the database:

- A malicious insider who intends to do harm

- A negligent insider who makes errors that make the database vulnerable to attack

- An infiltrator—an outsider who somehow obtains credentials via a scheme such as phishing or by gaining access to the credential database itself

Insider threats are among the most common causes of database security breaches and are often the result of allowing too many employees to hold privileged user access credentials.

Human error

Accidents, weak passwords, password sharing, and other unwise or uninformed user behaviors continue to be the cause of nearly half (49%) of all reported data breaches.

Exploitation of database software vulnerabilities

Hackers make their living by finding and targeting vulnerabilities in all kinds of software, including database management software. All major commercial database software vendors and open source database management platforms issue regular security patches to address

these vulnerabilities, but failure to apply these patches in a timely fashion can increase your exposure.

SQL/NoSQL injection attacks

A database-specific threat, these involve the insertion of arbitrary SQL or non-SQL attack strings into database queries served by web applications or HTTP headers. Organizations that don't follow secure web application coding practices and perform regular vulnerability testing are open to these attacks.

Buffer overflow exploitations

Buffer overflow occurs when a process attempts to write more data to a fixed-length block of memory than it is allowed to hold. Attackers may use the excess data, stored in adjacent memory addresses, as a foundation from which to launch attacks.

Malware

Malware is software written specifically to exploit vulnerabilities or otherwise cause damage to the database. Malware may arrive via any endpoint device connecting to the database's network.

Attacks on backups
Organizations that fail to protect backup data with the same stringent controls used to protect the database itself can be vulnerable to attacks on backups.

These threats are exacerbated by the following:

- **Growing data volumes:** Data capture, storage, and processing continues to grow exponentially across nearly all organizations. Any data security tools or practices need to be highly scalable to meet near and distant future needs.

- **Infrastructure sprawl**: Network environments are becoming increasingly complex, particularly as businesses move workloads to multicloud or hybrid cloud architectures, making the choice, deployment, and management of security solutions ever more challenging.

- **Increasingly stringent regulatory requirements:** The worldwide regulatory compliance landscape continues to grow in complexity, making adhering to all mandates more difficult.

- **Cybersecurity skills shortage:** Experts predict there may be as many as 8 million unfilled cybersecurity positions by 2022.

Denial of service (DoS/DDoS) attacks

In a denial of service (DoS) attack, the attacker deluges the target server—in this case the database server—with so many requests that the server can no longer fulfill legitimate requests from actual users, and, in many cases, the server becomes unstable or crashes.

In a distributed denial of service attack (DDoS), the deluge comes from multiple servers, making it more difficult to stop the attack. See our video “What is a DDoS Attack”(3:51) for more information:

Best practices

Because databases are nearly always network-accessible, any security threat to any component within or portion of the network infrastructure is also a threat to the database, and any attack impacting a user’s device or workstation can threaten the database. Thus, database security must extend far beyond the confines of the database alone.

When evaluating database security in your environment to decide on your team’s top priorities, consider each of the following areas:

- **Physical security:** Whether your database server is on-premise or in a cloud data center, it must be located within a secure, climate-controlled environment. (If your database server is in a cloud data center, your cloud provider will take care of this for you.)

- **Administrative and network access controls:** The practical minimum number of users should have access to the database, and their permissions should be restricted to the minimum levels necessary for them to do their jobs. Likewise, network access should be limited to the minimum level of permissions necessary.

- **End user account/device security:** Always be aware of who is accessing the database and when and how the data is being used. Data monitoring solutions can alert you if data activities are unusual or appear risky. All user devices connecting to the network housing the database should be physically secure (in the hands of the right user only) and subject to security controls at all times.

- **Encryption:** ALL data—including data in the database, and credential data—should be protected with best-in-class encryption while at rest and in transit. All encryption keys should be handled in accordance with best-practice guidelines.

- **Database software security:** Always use the latest version of your database management software, and apply all patches as soon as they are issued.

- **Application/web server security:** Any application or web server that interacts with the database can be a channel for attack and should be subject to ongoing security testing and best practice management.

- **Backup security:** All backups, copies, or images of the database must be subject to the same (or equally stringent) security controls as the database itself.

- **Auditing:** Record all logins to the database server and operating system, and log all operations performed on sensitive data as well. Database security standard audits should be performed regularly.

Controls and policies

In addition to implementing layered security controls across your entire network environment, database security requires you to establish the correct controls and policies for access to the database itself. These include:

- **Administrative controls** to govern installation, change, and configuration management for the database.

- **Preventative controls** to govern access, encryption, tokenization, and masking.

- **Detective controls** to monitor database activity monitoring and data loss prevention tools. These solutions make it possible to identify and alert on anomalous or suspicious activities.

Database security policies should be integrated with and support your overall business goals, such as protection of critical intellectual property and your cybersecurity policies and cloud security policies. Ensure you have designated responsibility for maintaining and auditing security controls within your organization and that your policies complement those of your cloud provider in shared responsibility agreements. Security controls, security awareness training and education programs, and penetration testing and vulnerability assessment strategies should all be established in support of your formal security policies.
Data protection tools and platforms

Today, a wide array of vendors offer data protection tools and platforms. A full-scale solution should include all of the following capabilities:

- **Discovery:** Look for a tool that can scan for and classify vulnerabilities across all your databases—whether they're hosted in the cloud or on-premise—and offer recommendations for remediating any vulnerabilities identified. Discovery capabilities are often required to conform to regulatory compliance mandates.

- **Data activity monitoring:** The solution should be able to monitor and audit all data activities across all databases, regardless of whether your deployment is on-premise,

in the cloud, or in a container. It should alert you to suspicious activities in real-time so that you can respond to threats more quickly. You'll also want a solution that can enforce rules, policies, and separation of duties and that offers visibility into the status of your data through a comprehensive and unified user interface. Make sure that any solution you choose can generate the reports you'll need to meet compliance requirements.

- **Encryption and tokenization capabilities:** In case of a breach, encryption offers a final line of defense against compromise. Any tool you choose should include flexible encryption capabilities that can safeguard data in on-premise, cloud, hybrid, or multicloud environments. Look for a tool with file, volume, and application encryption capabilities that conform to your industry's compliance requirements, which may demand tokenization (data masking) or advanced security key management capabilities.

- **Data security optimization and risk analysis:** A tool that can generate contextual insights by combining data security information with advanced analytics will enable you to accomplish optimization, risk analysis, and reporting with ease. Choose a solution that can retain and synthesize large quantities of historical and recent data about the status and security of your databases, and look for one that offers data exploration, auditing, and reporting capabilities through a comprehensive but user-friendly self-service dashboard.

5.WHAT IS DATABASE PERFORMANCE TUNING?

Database Performance Tuning refers to the process of optimizing database systems for improved performance and efficiency. It involves making adjustments in the database configuration, query design, indexing, and other system parameters to achieve maximum performance. By fine-tuning databases, businesses can ensure a faster response time for data retrieval and analytics, contributing to better decision making and enhanced user experiences.

5.1 Functionality and Features

Database Performance Tuning involves various techniques and methodologies aimed at improving database performance, such as:

- **Query optimization**: Rewriting SQL queries for better execution plans
- Index management: Creating, modifying, and deleting indexes to optimize **data access**
- Resource allocation: Assigning memory, CPU, and disk resources for optimal performance
- Database design: Designing **database schemas** that allow efficient data processing

- **Data partitioning**: Dividing large tables into smaller, more manageable pieces for improved query performance
- Caching: Storing frequently accessed data in memory for faster retrieval

5.2 Benefits and Use Cases

Database Performance Tuning offers several advantages to businesses, including:

- Increased query efficiency: Faster data retrieval allows users to run complex queries without experiencing delays
- Reduced resource consumption: Optimized databases require fewer computing resources, leading to cost savings
- Improved scalability: Tuned databases can handle more concurrent users and larger data volumes
- Better user experience: Prompt data access enhances user satisfaction and productivity
- Higher return on investment: Efficient databases make the best use of hardware and software investments

5.3 Challenges and Limitations

Despite the benefits, Database Performance Tuning does have limitations:

- Time-consuming: Rigorous tuning efforts can require significant time and expertise
- Diminishing returns: Ongoing tuning might not yield substantial performance improvements
- Complexity: Adjusting multiple configurations can create complications in maintaining the database system

5.4 Database Performance Tuning Techniques:

1. Make minor adjustments to queries

There are plenty of ways a query can be tuned. However, one of the most vital is to take your time with the process. You can start with basic arrangements and then work your way up. You can adjust indexes or place a simple skip list around the query.

You can also run a database command and look for high-selectivity queries. Then, make changes to these queries by placing indexes and changing your query plan. Making these simple changes can massively affect performance. Check out the practical ways to maximize your website for lead generation.

2. Statistics should be up to date

- Statistics can be used effectively to generate the right execution plans. This is for performance tuning.

- There are plenty of great performance tuning tools. However, if you're using outdated statistics, these plans couldn't be optimized to meet present issues.

3. Don't use the leading wildcards

- Wildcard parameters can force a full table scan even if indexed fields are inside given tables.
- If database engines scan every row in a table to look for a specific entry, then the delivery speed can significantly decrease. Parallel queries may also suffer from this scanning of the whole data in the memory.
- This may result in optimum CPU utilization and not letting other queries run in the memory.

4. Use Constraints

- Constraints are one of the most effective ways to speed up queries since it allows SQL optimizer to develop a better execution plan. However, the enhanced performance also comes at the cost of the data since it needs more memory.
- Depending on your objectives, the enhanced query speed will be well worth it, but being aware of the price is vital.

5. Increase memory

- In data from Statista, 32% of B2B businesses expect to boost their spending on database management.
- One way that DBAs might handle SQL performance tuning issues is to enhance the memory allocation of the current databases. SQL databases have plenty of memory, improve efficiency, and performance often follows.

6. Overhaul CPU

- If you're regularly experiencing database and operation strain, you might want to invest in a more robust CPU.
- Businesses that often use obsolete hardware and other systems experience database performance issues that can affect their ROI.

7. Improve indexes

- Aside from queries, another essential element of a database is the index. If done right, indexing enhances database performance and optimizes the query execution duration.
- In the same way, it helps build a data structure that lets you optimize your data and make it easy to find information. Because it's a lot easier to find data, indexing boosts the efficiency of data retrieval and speeds up the whole process. This saves you and the system a lot of time and effort.

8. Defragment data

- One of the best approaches to improving database performance is data defragmentation. Over time, so much data is constantly being written and deleted in the database.
- As a result, data can become defragmented. It slows down the data retrieval process since it affects the query's ability to find the information it's searching for.
- When you defragment data, you allow that relevant data to be grouped together, and then you erase index page issues. As a result, I/O-related operations will run so much quicker.

9. Review the actual execution plan, not just the estimated plan

- The estimated execution plan can be helpful if you're writing queries since it previews how the plan will run. However, it could be blind to parameter data types which would be wrong.
- If you want to get the best results in performance tuning, it helps that you review the actual execution plan first. This is because the existing plan will use accurate statistics.

10. Adjust queries, make one small change at a time

- Many too many changes at once can be detrimental in the long run. A more efficient approach is to make changes with the most expensive operations and then work from there.

11. Review access

- Once you know that your database hardware works well, the next thing that you need to do is to review your database access. It includes which applications are accessing your database.
- Suppose one of the services and applications suffers from poor database performance. In that case, you mustn't immediately jump to conclusions about the service or application responsible for it.
- It's also possible that a single client might be experiencing poor performance. However, there's also a possibility that the entire database is having issues. Thus, you need to dig into who has access to the database and whether or not it's just a single service having a problem.
- That's where the right database management support comes in. You can then drill down into its metrics so that you know the root cause.