UNIT 5:

Backup, Recovery AND Database Security

Database Backup and Recovery

It is imperative to have a backup of the database in case the original is corrupted or lost because of any reason. Using this backup, the database can be recovered as it was before the failure.

Database backup basically means that a duplicate of the database information and data is created and stored in backup server just to be on the safe side. Transaction logs are also stored in the backup along with the database data because without them, the data would be useless.

Reasons of Failure in a Database

There can be multiple reasons of failure in a database because of which a database backup and recovery plan is required. Some of these reasons are:

- **User Error -** Normally, user error is the biggest reason of data destruction or corruption in a database. To rectify the error, the database needs to be restored to the point in time before the error occured.
- **Hardware Failure -** This can also lead to loss of data in a database. The database is stored on multiple hard drives across various locations. These hard drives may sometimes malfunction leading to database corruption. So, it is important to periodically change them.
- **Catastrophic Event -** A catastrophic event can be a natural calamity like a flood or earthquake or deliberate sabotage such as hacking of the database. Either way, the database data may be corrupted and backup may be required.

Methods of Backup

The different methods of backup in a database are:

- **Full Backup -** This method takes a lot of time as the full copy of the database is made including the data and the transaction records.
- **Transaction Log -** Only the transaction logs are saved as the backup in this method. To keep the backup file as small as possible, the previous transaction log details are deleted once a new backup record is made.
- **Differential Backup -** This is similar to full backup in that it stores both the data and the transaction records. However only that information is saved in the backup that has changed since the last full backup. Because of this, differential backup leads to smaller files.

Hardware Protection and Type of Hardware Protection

In this article, we are going to learn about hardware protection and its types. So first, lets take a look at the type of hardware which is used in a computer system. We know that a computer system consists of hardware components like processor, monitor, RAM and many more. The important thing is, that the operating system ensures that these devices are not directly accessible by the user.

Basically, hardware protection is divided into 3 categories: CPU protection, Memory Protection, and I/O protection. These are explained as follows:

1. CPU Protection:

CPU protection ensures that, a process does not monopolize the CPU indefinetely, as it would prevent other processes from being executed. Each process should get a limited time, so that every process gets time to execute it's instructions. To address this, a timer is used to limit the amount of time, which a process can occupy from the CPU. After the timer expires, a signal is sent to the process for relenquishing the CPU. Hence one process cannot hold the CPU forever.

2. Memory Protection:

In memory protection, we are talking about that situation when two or more processes are in memory and one process may access the other process memory. To prevent this situation we use two registers which are known as:

1. Base register
2. Limit register

So basically **Base register** store the starting address of program and limit register store the size of the process. This is done to ensure that whenver a process wants to access the memory, the OS can check that – Is the memory area which the process wants to access is previliged to be accessed by that

process or not.

3. I/O Protection:
With I/O protection, an OS ensures that following can be never done by a processes:

1. **Termination I/O of other process** – This means one process should not be able to terminate I/O operation of othe processes.
2. **View I/O of other process** – One process should not be able to access the data being read/written by other processes from/to the Disk(s).
3. **Giving priority to a particular process I/O** – No process must be able to priorotize itself or other processes which are doing I/O operations, over other processes.

Redundancy in DBMS

In this article, we will learn about redundancy in DBMS. First, let us understand data redundancy.

Data redundancy means the occurrence of duplicate copies of similar data. It is done intentionally to keep the same piece of data at different places, or it occurs accidentally.

What is Data redundancy in the database management system?

In DBMS, when the same data is stored in different tables, it causes data redundancy.

Sometimes, it is done on purpose for recovery or backup of data, faster access of data, or updating data easily. Redundant data costs extra money, demands higher storage capacity, and requires extra effort to keep all the files up to date.

Sometimes, unintentional duplicity of data causes a problem for the database to work properly, or it may become harder for the end user to access data. Redundant data unnecessarily occupy space in the database to save identical copies, which leads to space constraints, which is one of the major problems.

Let us understand redundancy in DBMS properly with the help of an example.

Student_id	Name	Course	Session	Fee	Department

101	Devi	B. Tech	2022	90,000	CS
102	Sona	B. Tech	2022	90,000	CS
103	Varun	B. Tech	2022	90,000	CS
104	Satish	B. Tech	2022	90,000	CS
105	Amisha	B. Tech	2022	90,000	CS

In the above example, there is a "Student" table that contains data such as "Student_id", "Name", "Course", "Session", "Fee", and "Department". As you can see, some data is repeated in the table, which causes redundancy.

Problems that are caused due to redundancy in the database

Redundancy in DBMS gives rise to anomalies, and we will study it further. In a database management system, the problems that occur while working on data include inserting, deleting, and updating data in the database.

We will understand these anomalies with the help of the following student table:

student_id	student_name	student_age	dept_id	dept_name	dept_head
1	Shiva	19	104	Information Technology	Jaspreet Kaur
2	Khushi	18	102	Electronics	Avni Singh
3	Harsh	19	104	Information Technology	Jaspreet Kaur

1. Insertion Anomaly:

Insertion anomaly arises when you are trying to insert some data into the database, but you are not able to insert it.

Example: If you want to add the details of the student in the above table, then you must know the details of the department; otherwise, you will not be able to add the details because student details are dependent on department details.

2. Deletion Anomaly:

Deletion anomaly arises when you delete some data from the database, but some unrelated data is also deleted; that is, there will be a loss of data due to deletion anomaly.

Example: If we want to delete the student detail, which has student_id 2, we will also lose the unrelated data, i.e., department_id 102, from the above table.

3. Updating Anomaly:

An update anomaly arises when you update some data in the database, but the data is partially updated, which causes data inconsistency.

Example: If we want to update the details of dept_head from Jaspreet Kaur to Ankit Goyal for Dept_id 104, then we have to update it everywhere else; otherwise, the data will get partially updated, which causes data inconsistency.

Advantages of data redundancy in DBMS

- **Provides Data Security:** Data redundancy can enhance data security as it is difficult for cyber attackers to attack data that are in different locations.
- **Provides Data Reliability:** Reliable data improves accuracy because organizations can check and confirm whether data is correct.
- **Create Data Backup:** Data redundancy helps in backing up the data.

Disadvantages of data redundancy in DBMS

- **Data corruption:** Redundant data leads to high chances of data corruption.
- **Wastage of storage:** Redundant data requires more space, leading to a need for more storage space.

- **High cost:** Large storage is required to store and maintain redundant data, which is costly.

How to reduce data redundancy in DBMS

We can reduce data redundancy using the following methods:

- **Database Normalization:** We can normalize the data using the normalization method. In this method, the data is broken down into pieces, which means a large table is divided into two or more small tables to remove redundancy. Normalization removes insert anomaly, update anomaly, and delete anomaly.
- **Deleting Unused Data:** It is important to remove redundant data from the database as it generates data redundancy in the DBMS. It is a good practice to remove unwanted data to reduce redundancy.
- **Master Data:** The data administrator shares master data across multiple systems. Although it does not remove data redundancy, but it updates the redundant data whenever the data is changed.

Log-Based Recovery

- The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- If any operation is performed on the database, then it will be recorded in the log.
- But the process of storing the logs should be done before the actual transaction is applied in the database.

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.

1. <Tn, Start>

- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

1. <Tn, City, 'Noida', 'Bangalore' >

- When the transaction is finished, then it writes another log to indicate the end of the transaction.

1. <Tn, Commit>

There are two approaches to modify the database:

1. Deferred database modification:

- The deferred modification technique occurs if the transaction does not modify the database until it has committed.
- In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. Immediate database modification:

- The Immediate modification technique occurs if database modification occurs while the transaction is still active.
- In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery using Log records

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

1. If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.
2. If log contains record$<T_n$, Start> but does not contain the record either <Ti, commit> or <Ti, abort>, then the Transaction Ti needs to be undone.

Database Recovery

Database recovery techniques are used in database management systems (DBMS) to restore a database to a consistent state after a failure or error has occurred. The main goal of recovery techniques is to ensure data integrity and consistency and prevent data loss. There are mainly two types of recovery techniques used in DBMS:

Rollback/Undo Recovery Technique: The rollback/undo recovery technique is based on the principle of backing out or undoing the effects of a transaction that has not completed successfully due to a system failure or error. This technique is accomplished by undoing the changes made by the transaction using the log records stored in the transaction log. The transaction log contains a record of all the

transactions that have been performed on the database. The system uses the log records to undo the changes made by the failed transaction and restore the database to its previous state.

Commit/Redo Recovery Technique: The commit/redo recovery technique is based on the principle of reapplying the changes made by a transaction that has been completed successfully to the database. This technique is accomplished by using the log records stored in the transaction log to redo the changes made by the transaction that was in progress at the time of the failure or error. The system uses the log records to reapply the changes made by the transaction and restore the database to its most recent consistent state.

In addition to these two techniques, there is also a third technique called checkpoint recovery. Checkpoint recovery is a technique used to reduce the recovery time by periodically saving the state of the database in a checkpoint file. In the event of a failure, the system can use the checkpoint file to restore the database to the most recent consistent state before the failure occurred, rather than going through the entire log to recover the database.

Overall, recovery techniques are essential to ensure data consistency and availability in DBMS, and each technique has its own advantages and limitations that must be considered in the design of a recovery system

Database systems, like any other computer system, are subject to failures but the data stored in them must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transaction are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database. There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations. The techniques used to recover the lost data due to system crashes, transaction errors, viruses, catastrophic failure, incorrect commands execution, etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred update and immediate update or backing up data can be used. Recovery techniques are heavily dependent upon the existence of a special file known as a **system log**. It contains information about the start and end of each transaction and any updates which occur during the **transaction**. The log keeps track of all transaction operations that affect the values of database items. This information is needed to recover from transaction failure.

- The log is kept on disk start_transaction(T): This log entry records that transaction T starts the execution.
- read_item(T, X): This log entry records that transaction T reads the value of database item X.

- write_item(T, X, old_value, new_value): This log entry records that transaction T changes the value of the database item X from old_value to new_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.
- commit(T): This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.
- abort(T): This records that transaction T has been aborted.
- checkpoint: Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in a consistent state, and all the transactions were committed.

A transaction T reaches its **commit** point when all its operations that access the database have been executed successfully i.e. the transaction has reached the point at which it will not **abort** (terminate without completing). Once committed, the transaction is permanently recorded in the database. Commitment always involves writing a commit entry to the log and writing the log to disk. At the time of a system crash, item is searched back in the log for all transactions T that have written a start_transaction(T) entry into the log but have not written a commit(T) entry yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process.

- **Undoing –** If a transaction crashes, then the recovery manager may undo transactions i.e. reverse the operations of a transaction. This involves examining a transaction for the log entry write_item(T, x, old_value, new_value) and set the value of item x in the database to old-value. There are two major techniques for recovery from non-catastrophic transaction failures: deferred updates and immediate updates.
- **Deferred update –** This technique does not physically update the database on disk until a transaction has reached its commit point. Before reaching commit, all transaction updates are recorded in the local transaction workspace. If a transaction fails before reaching its commit point, it will not have changed the database in any way so UNDO is not needed. It may be necessary to REDO the effect of the operations that are recorded in the local transaction workspace, because their effect may not yet have been written in the database. Hence, a deferred update is also known as the **No-undo/redo algorithm**
- **Immediate update –** In the immediate update, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations are recorded in a log on disk before they are applied to the database, making recovery still possible. If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must

be rolled back hence we require both undo and redo. This technique is known as **undo/redo algorithm.**

- **Caching/Buffering –** In this one or more disk pages that include data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk. A collection of in-memory buffers called the DBMS cache is kept under the control of DBMS for holding these buffers. A directory is used to keep track of which database items are in the buffer. A dirty bit is associated with each buffer, which is 0 if the buffer is not modified else 1 if modified.
- **Shadow paging –** It provides atomicity and durability. A directory with n entries is constructed, where the ith entry points to the ith database page on the link. When a transaction began executing the current directory is copied into a shadow directory. When a page is to be modified, a shadow page is allocated in which changes are made and when it is ready to become durable, all pages that refer to the original are updated to refer new replacement page.
- **Backward Recovery** – The term “Rollback ” and “UNDO” can also refer to backward recovery. When a backup of the data is not available and previous modifications need to be undone, this technique can be helpful. With the backward recovery method, unused modifications are removed and the database is returned to its prior condition. All adjustments made during the previous traction are reversed during the backward recovery. In another word, it reprocesses valid transactions and undoes the erroneous database updates.
- **Forward Recovery** – “Roll forward “and “REDO” refers to forwarding recovery. When a database needs to be updated with all changes verified, this forward recovery technique is helpful.
  Some failed transactions in this database are applied to the database to roll those modifications forward. In another word, the database is restored using preserved data and valid transactions counted by their past saves.

Some of the backup techniques are as follows :

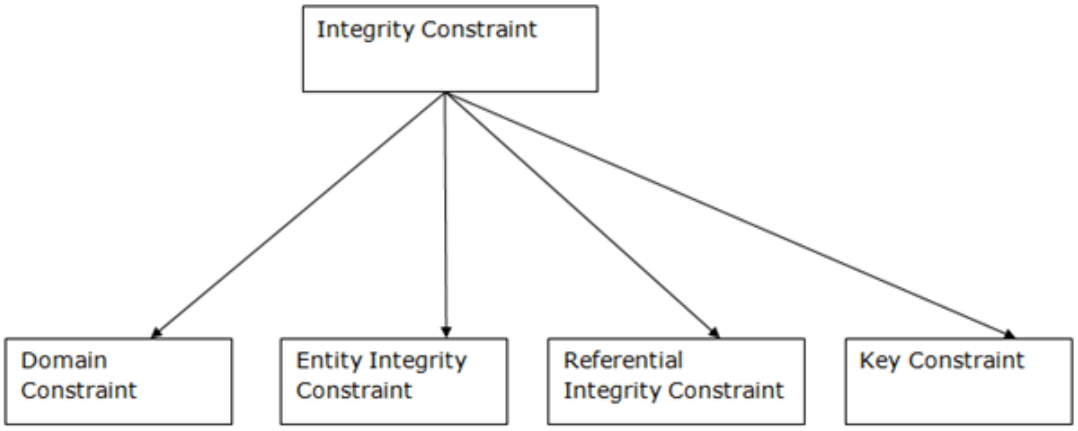
- **Full database backup –** In this full database including data and database, Meta information needed to restore the whole database, including full-text catalogs are backed up in a predefined time series.
- **Differential backup –** It stores only the data changes that have occurred since the last full database backup. When some data has changed many times since last full database backup, a differential backup stores the most recent version of the changed data. For this first, we need to restore a full database backup.
- **Transaction log backup –** In this, all events that have occurred in the database, like a record of every single statement executed is backed up. It is the backup of transaction log entries and contains all transactions that had happened to the database. Through this, the database can be recovered to a specific point in time. It

is even possible to perform a backup from a transaction log if the data files are destroyed and not even a single committed transaction is lost.

Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

Types of Integrity Constraint



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example:

EMPLOYEE

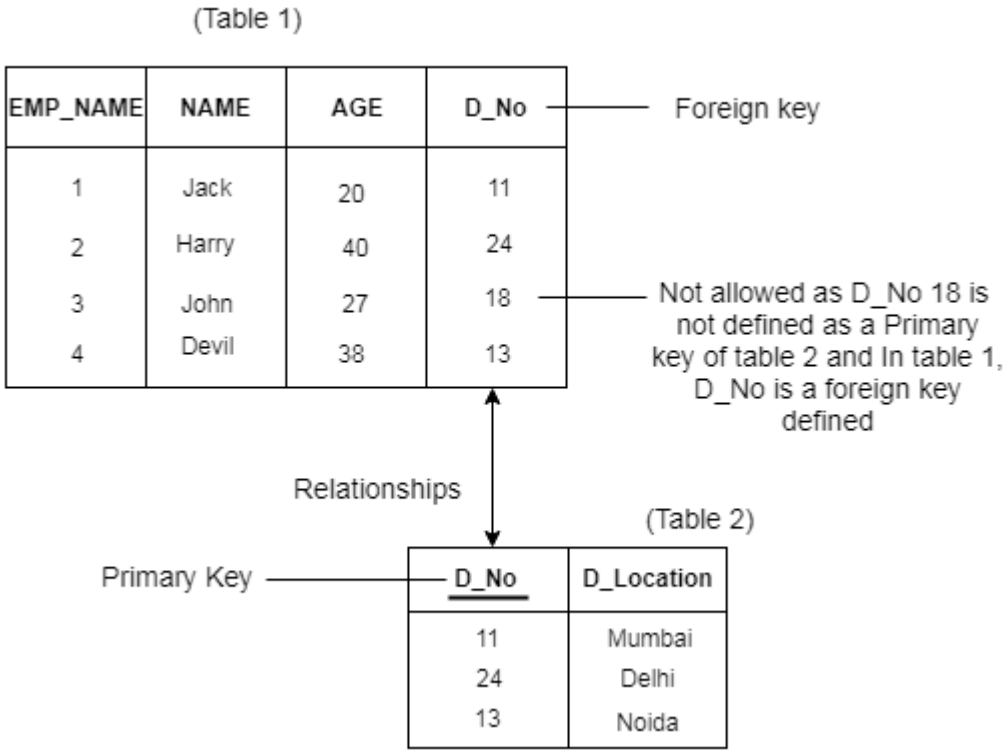
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:



4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

Database Security

Security of databases refers to the array of controls, tools, and procedures designed to ensure and safeguard confidentiality, integrity, and accessibility. This tutorial will concentrate on confidentiality because it's a component that is most at risk in data security breaches.

Security for databases must cover and safeguard the following aspects:

- The database containing data.
- Database management systems (DBMS)
- Any applications that are associated with it.
- Physical database servers or the database server virtual, and the hardware that runs it.
- The infrastructure for computing or network that is used to connect to the database.

Security of databases is a complicated and challenging task that requires all aspects of security practices and technologies. This is inherently at odds with the accessibility of databases. The more usable and accessible the database is, the more susceptible we are to threats from security. The more vulnerable it is to attacks and threats, the more difficult it is to access and utilize.

Why Database Security is Important?

According to the definition, a data breach refers to a breach of data integrity in databases. The amount of damage an incident like a data breach can cause our business is contingent on various consequences or elements.

- **Intellectual property that is compromised:** Our intellectual property--trade secrets, inventions, or proprietary methods -- could be vital for our ability to maintain an advantage in our industry. If our intellectual property has been stolen or disclosed and our competitive advantage is lost, it could be difficult to keep or recover.
- **The damage to our brand's reputation:** Customers or partners may not want to purchase goods or services from us (or deal with our business) If they do not feel they can trust our company to protect their data or their own.
- **The concept of business continuity (or lack of it):** Some businesses cannot continue to function until a breach has been resolved.

- **Penalties or fines to be paid for not complying:** The cost of not complying with international regulations like the Sarbanes-Oxley Act (SAO) or Payment Card Industry Data Security Standard (PCI DSS) specific to industry regulations on data privacy, like HIPAA or regional privacy laws like the European Union's General Data Protection Regulation (GDPR) could be a major problem with fines in worst cases in excess of many million dollars for each violation.
- **Costs for repairing breaches and notifying consumers about them:** Alongside notifying customers of a breach, the company that has been breached is required to cover the investigation and forensic services such as crisis management, triage repairs to the affected systems, and much more.

Common Threats and Challenges

Numerous software configurations that are not correct, weaknesses, or patterns of carelessness or abuse can lead to a breach of security. Here are some of the most prevalent kinds of reasons for security attacks and the reasons.

Insider Dangers

An insider threat can be an attack on security from any three sources having an access privilege to the database.

- A malicious insider who wants to cause harm
- An insider who is negligent and makes mistakes that expose the database to attack. vulnerable to attacks
- An infiltrator is an outsider who acquires credentials by using a method like phishing or accessing the database of credential information in the database itself.

Insider dangers are among the most frequent sources of security breaches to databases. They often occur as a consequence of the inability of employees to have access to privileged user credentials.

Human Error

The unintentional mistakes, weak passwords or sharing passwords, and other negligent or uninformed behaviours of users remain the root causes of almost half (49 percent) of all data security breaches.

Database Software Vulnerabilities can be Exploited

Hackers earn their money by identifying and exploiting vulnerabilities in software such as databases management software. The major database software companies and open-source databases management platforms release regular security patches to fix these weaknesses. However, failing to implement the patches on time could increase the risk of being hacked.

SQL/NoSQL Injection Attacks

A specific threat to databases is the infusing of untrue SQL as well as other non-SQL string attacks in queries for databases delivered by web-based apps and HTTP headers. Companies that do not follow the safe coding practices for web applications and conduct regular vulnerability tests are susceptible to attacks using these.

Buffer Overflow is a way to Exploit Buffers

Buffer overflow happens when a program seeks to copy more data into the memory block with a certain length than it can accommodate. The attackers may make use of the extra data, which is stored in adjacent memory addresses, to establish a basis for they can begin attacks.

DDoS (DoS/DDoS) Attacks

In a denial-of-service (DoS) attack in which the attacker overwhelms the targeted server -- in this case, the database server with such a large volume of requests that the server is unable to meet no longer legitimate requests made by actual users. In most cases, the server is unstable or even fails to function.

Malware

Malware is software designed to exploit vulnerabilities or cause harm to databases. Malware can be accessed via any device that connects to the databases network.

Attacks on Backups

Companies that do not protect backup data using the same rigorous controls employed to protect databases themselves are at risk of cyberattacks on backups.

The following factors amplify the threats:

- **Data volumes are growing:** Data capture, storage, and processing continue to increase exponentially in almost all organizations. Any tools or methods must be highly flexible to meet current as well as far-off needs.
- **The infrastructure is sprawling:** Network environments are becoming more complicated, especially as companies shift their workloads into multiple clouds and hybrid cloud architectures and make the selection of deployment, management, and administration of security solutions more difficult.
- **More stringent requirements for regulatory compliance:** The worldwide regulatory compliance landscape continues to increase by complexity. This makes the compliance of every mandate more challenging.

Best use of Database Security

As databases are almost always accessible via the network, any security risk to any component or part of the infrastructure can threaten the database. Likewise, any security attack that impacts a device or workstation could endanger the database. Therefore, security for databases must go beyond the limits of the database.

In evaluating the security of databases in our workplace to determine our organization's top priorities, look at each of these areas.

- **Security for physical security:** If the database servers are on-premises or the cloud data centre, they should be placed in a secure, controlled climate. (If our server for database is located in a cloud-based data centre, the cloud provider will handle the security on our behalf.)
- **Access to the network and administrative restrictions:** The practical minimum number of users granted access to the database and their access rights should be restricted to the minimum level required to fulfil their tasks. Additionally, access to the network is limited to the minimum permissions needed.
- **End security of the user account or device**: Be aware of who has access to the database and when and how data is used. Monitoring tools for data can notify you of data-related activities that are uncommon or seem to be dangerous. Any device that connects to the network hosting the database must be physically secured (in the sole control of the appropriate person) and be subject to security checks throughout the day.

- **Security:** ALL data--including data stored in databases, as well as credential information should be secured using the highest-quality encryption when in storage and while in transport. All encryption keys must be used in accordance with the best practices guidelines.
- **Security of databases using software:** Always use the most current version of our software to manage databases and apply any patches immediately after they're released.
- **Security for web server applications and websites:** Any application or web server that connects to the database could be a target and should be subjected to periodic security testing and best practices management.
- **Security of backups:** All backups, images, or copies of the database should have the identical (or equally rigorous) security procedures as the database itself.
- **Auditing:** Audits of security standards for databases should be conducted every few months. Record all the logins on the server as well as the operating system. Also, record any operations that are made on sensitive data, too.

Data protection tools and platforms

Today, a variety of companies provide data protection platforms and tools. A comprehensive solution should have all of the following features:

- **Discovery:** The ability to discover is often needed to meet regulatory compliance requirements**.** Look for a tool that can detect and categorize weaknesses across our databases, whether they're hosted in the cloud or on-premises. It will also provide recommendations to address any vulnerabilities that are discovered.
- **Monitoring of Data Activity:** The solution should be capable of monitoring and analysing the entire data activity in all databases, whether our application is on-premises, in the cloud, or inside a container. It will alert us to suspicious activity in real-time to allow us to respond more quickly to threats. It also provides visibility into the state of our information through an integrated and comprehensive user interface. It is also important to choose a system that enforces rules that govern policies, procedures, and the separation of duties. Be sure that the solution we select is able to generate the reports we need to comply with the regulations.
- **The ability to Tokenize and Encrypt Data:** In case of an incident, encryption is an additional line of protection against any compromise. Any software we choose to use must

have the flexibility to protect data cloud, on-premises hybrid, or multi-cloud environments. Find a tool with volume, file, and application encryption features that meet our company's regulations for compliance. This could require tokenization (data concealing) or advanced key management of security keys.

- **Optimization of Data Security and Risk Analysis:** An application that will provide contextual insights through the combination of security data with advanced analytics will allow users to perform optimizing, risk assessment, and reporting in a breeze. Select a tool that is able to keep and combine large amounts of recent and historical data about the security and state of your databases. Also, choose a solution that provides data exploration, auditing, and reporting capabilities via an extensive but user-friendly self-service dashboard.

Database Authentication

Database authentication is the process or act of confirming that a user who is attempting to log in to a database is authorized to do so, and is only accorded the rights to perform activities that he or she has been authorized to do.

The concept of authentication is familiar to almost everyone. For example, a mobile phone performs authentication by asking for a PIN. Similarly, a computer authenticates a username by asking for the corresponding password.

In the context of databases, however, authentication acquires one more dimension because it may happen at different levels. It may be performed by the database itself, or the setup may be changed to allow either the operating system, or some other external method, to authenticate users.

For example, while creating a database in Microsoft’s SQL Server, a user is required to define whether to to use database authentication, operating system authentication, or both (the so-called mixed-mode authentication). Other databases in which security is paramount employ near-foolproof authentication modes like fingerprint recognition and retinal scans.