Course 5 - Relational Database Administration (DBA)

Introduction to Database Management

Module 1: - Overview of Database Management Tasks

1] Day in the Life of Database Administrator

Starting the day: -

- Check database dashboards for activity alerts and resolve issues.
- Check overnight batch jobs

Supporting users: -

- Check email
- Review support tickets
- Triage user requests
- Optimize queries
- Clarify requests and schema changes

Working with stakeholders: -

- Meeting with developers, data engineers, and data architects.
- Stress test scenarios.
- Determine appropriate server resources needed.

Finishing the day: -

- Automate repeating tasks.
- Respond to user requests.
- Monitor database activities.

Summary: -

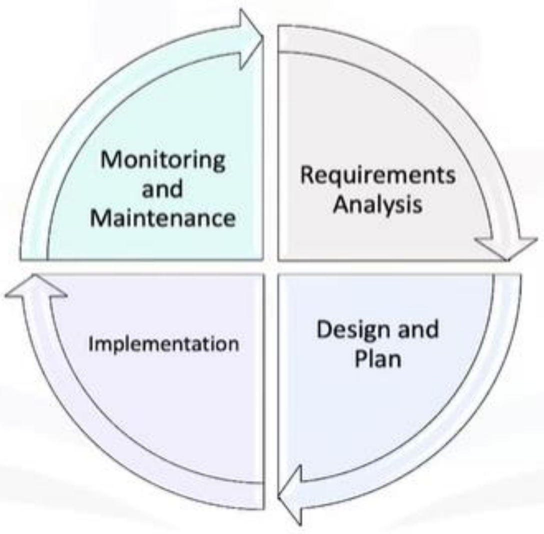
A typical day for a database administrator includes:

- Checking the state of the database

- Resolving issues
- Responding to support tickets
- Meeting with developers and other stakeholders
- Monitoring database activity

2] Database Management Lifecycle

The database life cycle –



Requirement Analysis: -

- DBAs like Rygel work with data engineers to understand the purpose and scope of the database. Rygel and the rest of the design team must establish what data is involved, talk to the users and producers of that data, and develop samples of how users will use the data, such as reports or dashboards.
- Rygel and the design team work with the stakeholders to determine their requirements. Stakeholders may include developers, data engineers, administrators or end-users of the applications supported by the database, technology managers, and other DBAs. A few of the tasks they might perform include:
  - Analyzing the need for the database.
  - Clarifying the goals, the database fulfils and
  - Identifying the users of the database.

Design and plan: -

- Rygel's job is to develop a plan for implementing the database. To do so, he works with database objects such as instances, databases, tables, and indexes.

- A database model represents the design of a database: which instance contains which databases and tables, how the tables relate to each other, how users access the data, and so on. Database Architects and DBAs model the databases and their objects with the help of Entity Relationship Diagrams or ERDs.
- In self-managed environments such as on-premise databases, DBAs also need to consider size, capacity, and growth. They determine appropriate server resources like storage space, memory, processing power, and log file size. They also need to plan for how database objects are physically stored. For example, DBAs can choose to store frequently used data on a high-speed disk array or to store indexes separately from the data for better performance.

Implementation: -

- DBAs roll out the carefully planned design.
- Rygel creates database objects like instances, databases, tables, views, and indexes.
- Another task Rygel performs is:
    - Configuring database security,
    - Granting access for database users, groups, and roles, so database objects are accessible only by the specific users and applications authorized to access them.
    - Automates repeating database tasks such as backups, restores, and deployments to improve efficiency.
- In populating the database, he might import data from other databases, export data based on a query from a different source, or migrate projects from one environment to another, such as moving a project from the Application Development environment to the Production environment.

Monitor and Maintain: -

- Rygel looks after the daily operations of the database.
- Monitor system for long-running queries and help end-users optimize them to run faster (performance issues) and non-overuse system resources.
- Review Reports
- Most Relational Database Management Systems have built-in reports to monitor activity, identify expensive queries, resource waits, and other relevant items. Often companies build custom reports on top of these, and DBAs help with that.
- To keep the databases working at top efficiency, Rygel may also apply upgrades and patches to database software. He needs to stay up to date on issues and advancements in the field so he can recommend and implement emerging database technologies.
- Automate deployments and routine tasks such as backups whenever possible to keep processes working efficiently.
- In every database, operational issues will sometimes arise. Rygel troubleshoots these issues, escalating problems where necessary.

- Security and Compliance (DBA) – Keep data safe
    - Ensure data is secure and only authorized users can access it.
    - Review logs, monitor failed logins and data access activity.
    - Maintain database permissions – grant/revoke access.

Summary: -

- The database life cycle stages are requirements analysis, design and plan, implementation, and monitor and maintain.
- In the requirements analysis stage, DBAs determine the purpose and scope of the database by interviewing data users and producers, examining the data, and creating samples.
- In the design and plan stage, DBAs work on logical and physical design.
- In the implementation stage, DBAs deploy the database.
- In the monitor and maintain stage, DBAs manage the daily operations of the database.

Reading: Data Security, Ethical and Compliance Considerations

Fundamental Ethics

A foundation of basic ethical concepts supports good data security practices. These should help guide the policies and workflows you create and the actions you take. Some important concepts are:

- **Transparency**: When you collect information, you should tell the owners of the information exactly what data you will collect and what you will do with it. Inform them about how you use the data, how you store it, who will have access to it, and how you will dispose of it when you have finished using it.
- **Consent**: You should get clear consent from data owners before you collect their data. This should detail what data you will be allowed to collect and how you will be allowed to use it.
- **Integrity**: Always be clear about your procedures and policies, and always follow them consistently. As far as you can, make sure that others in your organization also follow the correct procedures and policies.

Consider creating a code of ethics–a written statement of security-related standards and intentions. You can include priorities, best practices, who will be responsible, and whatever else is important to understand clearly. This will create shared expectations for yourself and others, which will help build trust and make it easier for everyone to follow correct procedures.

Secure System Design

The structure of your system is a powerful tool in keeping your data safe. If your system is built to maintain security, it's much easier to prevent breaches. To make sure your system works for you, consider these factors.

- **Protection from malicious access**: The front line of protection for your data is basic software security. Your firewall and other cybersecurity tools should actively prevent hacking and malware installation, and alert you to threats. Be sure you update this software frequently, to keep scanning lists up to date. Also, educate users about phishing and other ways that they can unwittingly enable malicious access.
- **Secure storage**: The storage you choose for your data must be secure not only from malicious access, but also from hardware failure and even natural disasters. Select your services carefully and make sure you understand their security practices and disaster preparedness plans. Back up your data regularly and reliably to minimize data loss in case of an emergency.
- **Accurate access**: Only those who need certain data should be able to access it. Establish a system of assigning and tracking privileges that assigns each user only the necessary privileges, and controls what they can do with the data. Ensure that your policy complies with any data usage agreements you have made.
- **Secure movement**: Data can be particularly vulnerable to interception when you move it into or out of storage. Be sure to consider safe transfer methods as carefully as you plan safety for the rest of your system.
- **Secure archiving**: At some point, you may want to move data from active storage to an archive. This can protect it from accidental access and make your system more efficient. Make sure your archiving system is as secure as the rest of your storage. Data agreements often specify how long you may use the data, so be sure the archived data is regularly weeded for expired rights and don't retain any more data than you will need for compliance with organization policy. Eliminate your discarded data securely and completely.

Compliance Issues

Maintaining compliance with all relevant laws and standards is a vital concern. Failure can result in data insecurity, professional censure for your organization, and even legal action. This list includes some of the most common types of standards, but it's not exhaustive; always find out which regulations and standards apply to your organization.

- **National/international regulations**: Many industries must be concerned with important legal standards on the national or international level. Some examples include HIPAA regulations for health-related information in the US, the GDPR in Europe, and the Information Technology Act, 2000 in India.
- **Industry standards**: Some data standards aren't enforced by law but can still carry repercussions for your organization's reputation and standing if they aren't followed. An example might be the Payment Card Industry Data Security Standard (PCI DSS), which applies to any organization that collects, stores, or transmits cardholder data.
- **Organization best practices**: Each organization will formulate standards for handling its internal data; as a DBA, you may work on that as part of your job. Employee confidentiality is often an important part of these policies, as is protecting intellectual property owned by the organization.

If you build your system and procedures thoughtfully and maintain them with consistency and vigilance, you can keep the data in your system safe and productive.

Summary & Highlights:-

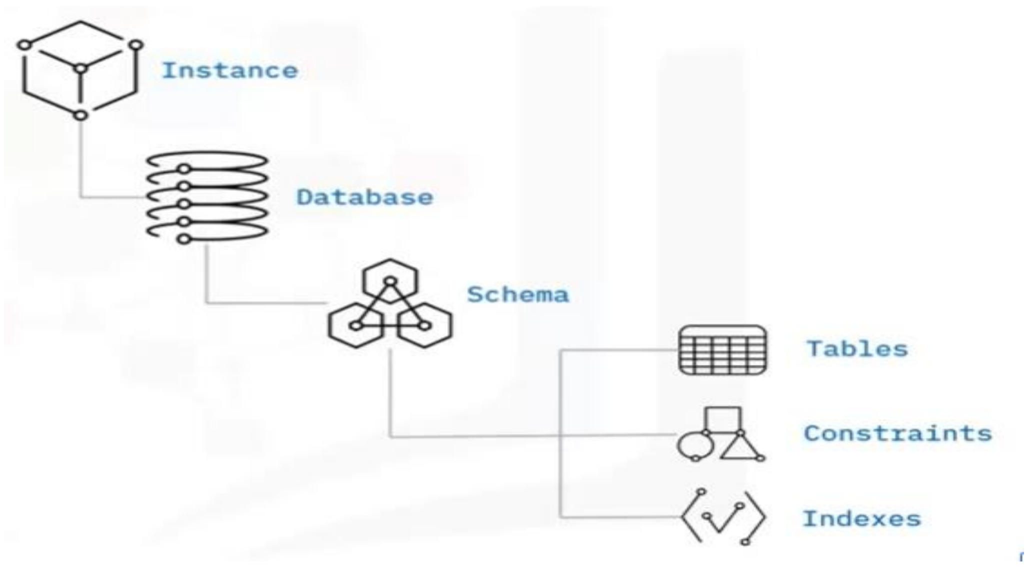
- A typical day for a database administrator includes checking the state of the database and resolving any issues, responding to support tickets, meeting with developers and other stakeholders, and monitoring database activity.
- The stages of the database life-cycle are requirements analysis, design and plan, implementation, and monitor and maintain.
- That DBAs and other major stakeholders are involved in making decisions during each database lifecycle stage.

Module 2: - Server Objects and Hierarchy

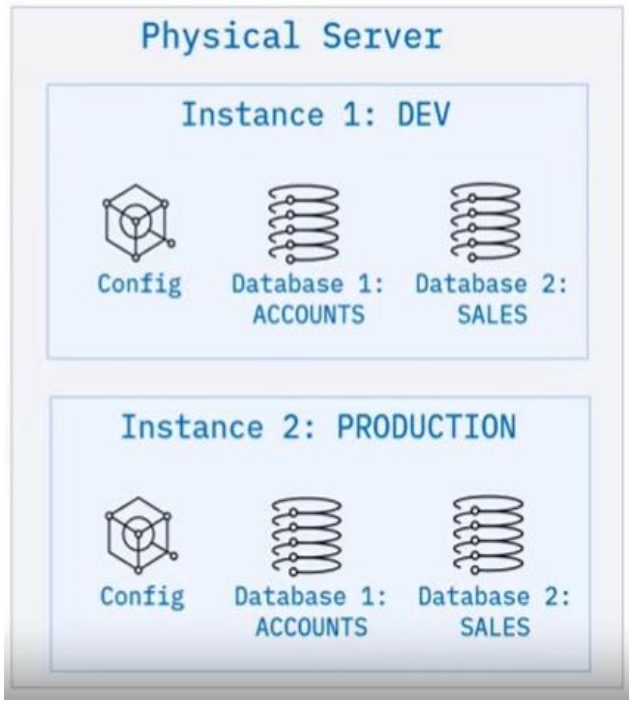
1] Database Objects

Database Hierarchy –

- Instance
- Database
- Schema
- Database objects
  - Tables
  - Constraints
  - Indexes
- Other database objects in a hierarchical structure allows database administrators to manage security, maintenance, and accessibility.

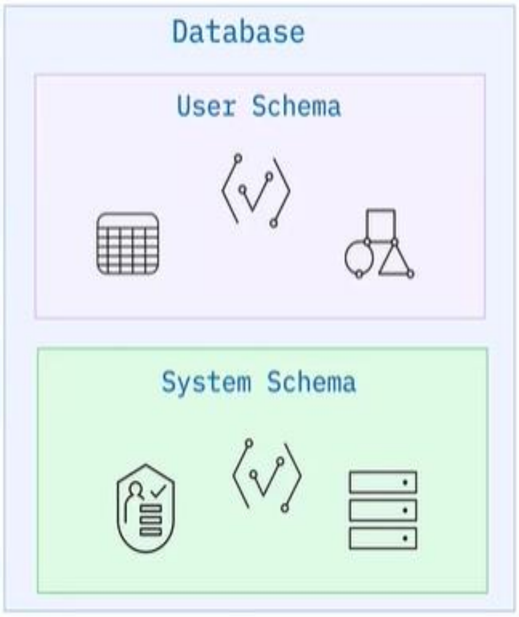


Instances –



- A logical boundary for a database or set of databases where you organize database objects and set configuration parameters.
- Every database within an instance is assigned a unique name, has its own set of system catalog tables (which keep track of the objects within the database) and has its own configuration files.
- You can create more than one instance on the same physical server
- provides a unique database server environment for each instance.
- The databases and other objects within one instance are isolated from those in any other instance.
- You can use multiple instances when you want to use one instance for a development environment and another instance for a production environment, restrict access to sensitive information, or control high-level administrative access.
- Not all RDBMSes use the concept of instances, often managing database configuration information in a special database instead.
- In Cloud-based RDBMSes, the term instance means a specific running copy of a service.

Schemas –



- Specialized database object that provides a way to group other database objects logically.
- Can contain tables, views, nicknames, triggers, functions, packages, and other objects.
- When you create a database object, you can assign it to a schema.
- If you want to assign the object to a specific schema, you can explicitly include the schema name.
- If you don't include the schema name, the object is implicitly assigned to the current schema.
- Default schema is the user schema for the currently logged-on user.
- Provides a naming context. Using the schema name as a name qualifier enables you to distinguish between objects with the same name in different schemas.
- For example, the schema names Internal and External make it easy to distinguish two different SALES tables: INTERNAL.SALES in the internal schema and EXTERNAL.SALES in the External schema. Thus, schemas enable multiple applications to store data in a single database without encountering namespace conflicts.
- Many RDBMSes use a specialized schema to hold configuration information and metadata about a particular database. For example, tables in a system schema can store lists of database users and their access permissions, information about the indexes on tables, details of any database partitions that exist, and user-defined data types.

Database objects –

- Physical database design consists of defining database objects.
- Create and manage using Graphical tools, scripting, APIs, SQL.
- If you use SQL to create or manage the object, you will use Data Definition Language statements like CREATE or ALTER.

Common database objects –

- **Tables –** Logical structures consisting of rows and columns which store data.
- **Constraints –**
    - Within any business, data is often subject to certain restrictions or rules.
    - For example, an employee number must be unique. Constraints provide a way to enforce such rules.
- **Indexes –** An index is a set of pointers used to improve performance and ensure the uniqueness of the data.
- **Keys –**
    - A key uniquely identifies a row in a table.
    - Keys enable DBAs to define the relationships between tables.
- **Views –**
    - A view provides a different way of representing the data in one or more tables.
    - A view is not an actual table and requires no permanent storage.
- **Aliases –**
    - An alias is an alternative name for an object such as a table.
    - DBAs use aliases to provide shorter, simpler names to reference objects.
- **Events –** An event is a Data Manipulation Language (DML) or Data Definition Language (DDL) action on a database object that can initiate a trigger.
- **Triggers –** A trigger defines a set of actions performed in response to an insert, update, or delete on a specified table.
- **Log files –** Log files store information about transactions in a database.

Summary: -

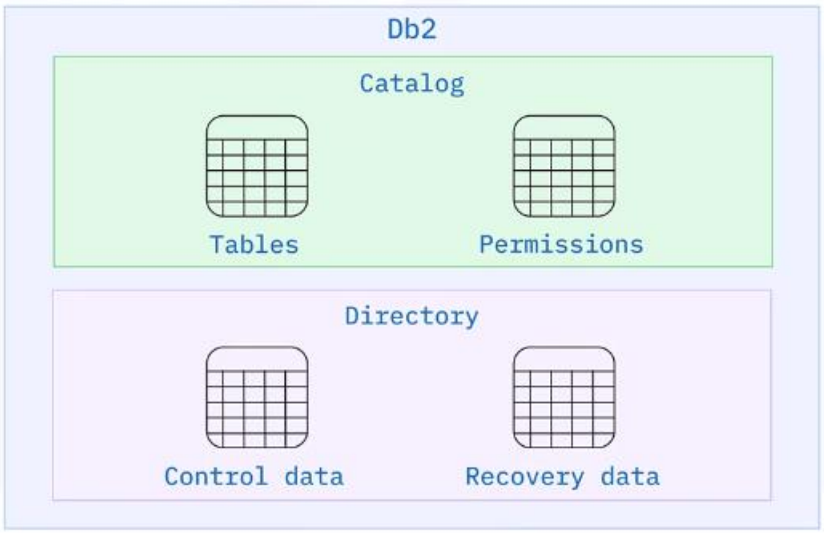
- An instance is a logical boundary for a database or set of databases where you organize database objects and set configuration parameters.
- A schema is a specialized database object that provides a way to group other database objects logically. Schemas also provide a naming context for database objects.
- Database objects are the items that exist within the database, such as tables, constraints, indexes, keys, views, aliases, triggers, events, and log files.

2] System Objects and Database Configuration

System Objects –

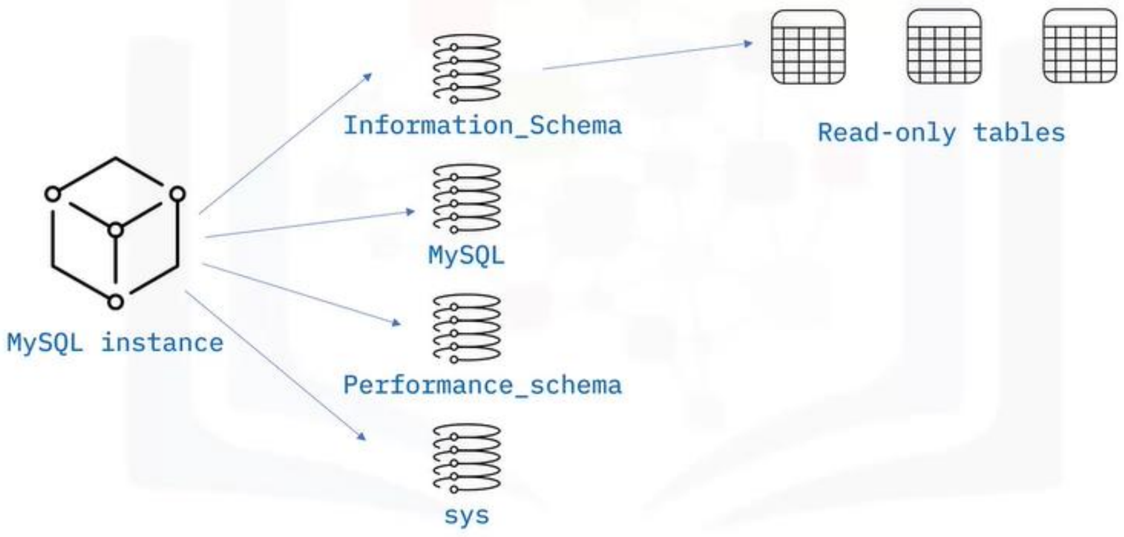
- Store database metadata in special objects.
- Known as system database, system schema, catalog or dictionary.
- Query to retrieve data.

Db2 system objects –



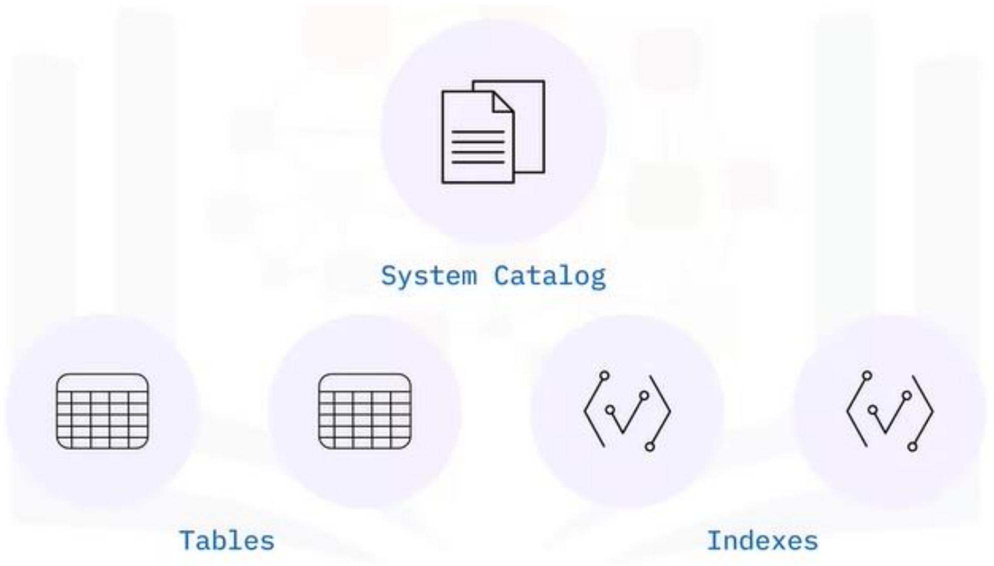
- Different RDBMSes use different names for their metadata stores: Db2 uses the catalog and the directory.
- The catalog consists of tables of data about everything defined to the Db2 system. When a user creates, alters, or drops any object, Db2 inserts, updates, or deletes rows of the catalog that describe that object and how it relates to other objects.
- The directory contains the Db2 internal control data used by Db2 during its normal operation. Among other things, the directory contains database descriptors, access paths for applications, and recovery and utility status information.

MySQL system objects –



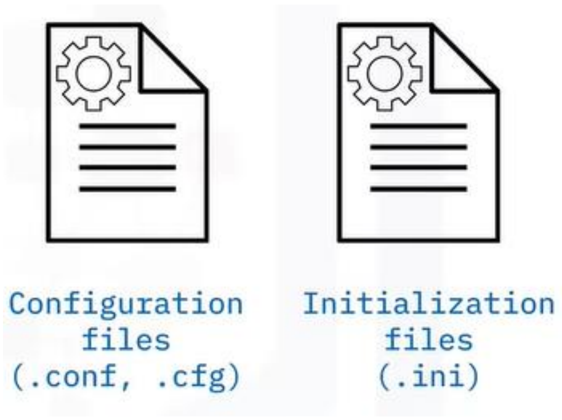
- MySQL uses a system schema to store database metadata.
- For example, each new MySQL instance has four system databases: information_schema, mysql, performance_schema, and sys.
- Each system database contains several read-only tables.

PostgreSQL system objects –



- PostgreSQL uses the system catalog, which is a schema with tables and views that contain metadata about all the other objects inside the database, and more.
- With it, you can discover when various operations happen, how tables or indexes are accessed, and whether the database system is reading information from memory or needing to fetch data from disk.

Configuration files –

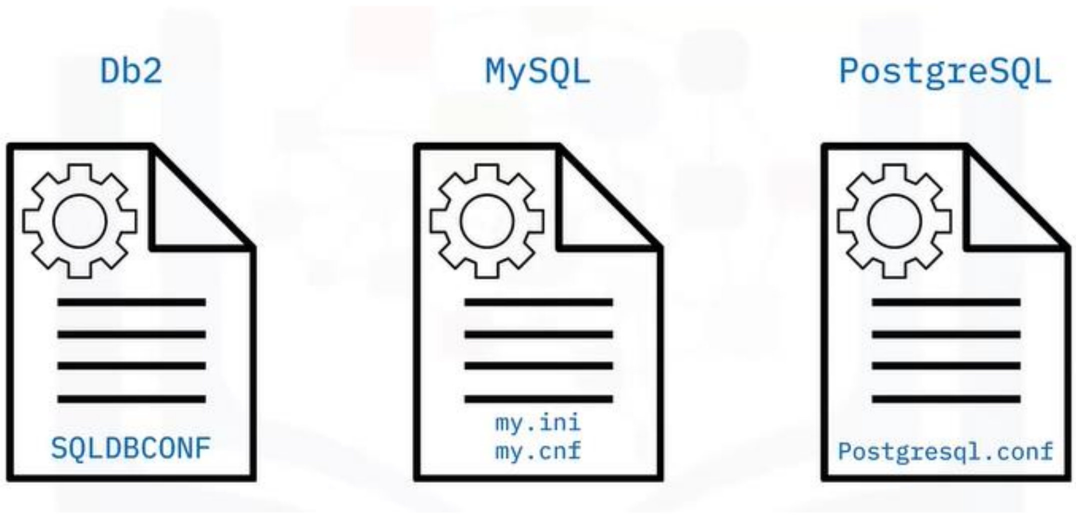

Configuration files (.conf, .cfg)
Initialization files (.ini)

- During any database installation, you supply parameters like the location of the data directory, the port number that the service listens on for connections, memory allocation, and much more.
- You can accept the default options or supply custom values to suit the environment.
- The database installation process saves this information into files, known as configuration or initialization files.
- The database uses the information in these files to set parameters as it starts up.

Common configuration settings –

- Initial configuration settings for a database can include:
    - Location of data files and log files
    - Port the server listens on
    - Memory allocation
    - Connection timeout
    - Maximum packet size

Configuration file examples –



my.ini – window-based config file

my.cnf – Linux-based config file

How to configure databases –

1. **On-premises:**
    - Stop the database service
    - Modify the configuration file
    - Restart the database service

2. **Cloud-based:**
    - Use graphical tools or APIs to modify the setting
    - Database scales dynamically.
    - Can scale many configuration options, such as storage size and compute power, through a graphical interface as the service is running. You don't have to edit configuration files.

Summary: -

- System objects store database metadata. You can query these objects to get information about the configuration and performance of your database.
- Different RDBMSes use different names for their system objects. Most use the terms system schema, system tables, catalog, or directory.
- Configuration files store the information that the database needs as it initializes. Again, different RDBMSes use different names for the configuration files, like SQLDBCONF, my.ini, or postgresql.conf.
- In an on-premises RDBMS, you edit the configuration file to change a setting.
- In a Cloud-based RDBMS, you can scale settings dynamically.

3] Database Storage

Plan database storage: –

As a DBA you will responsible for –

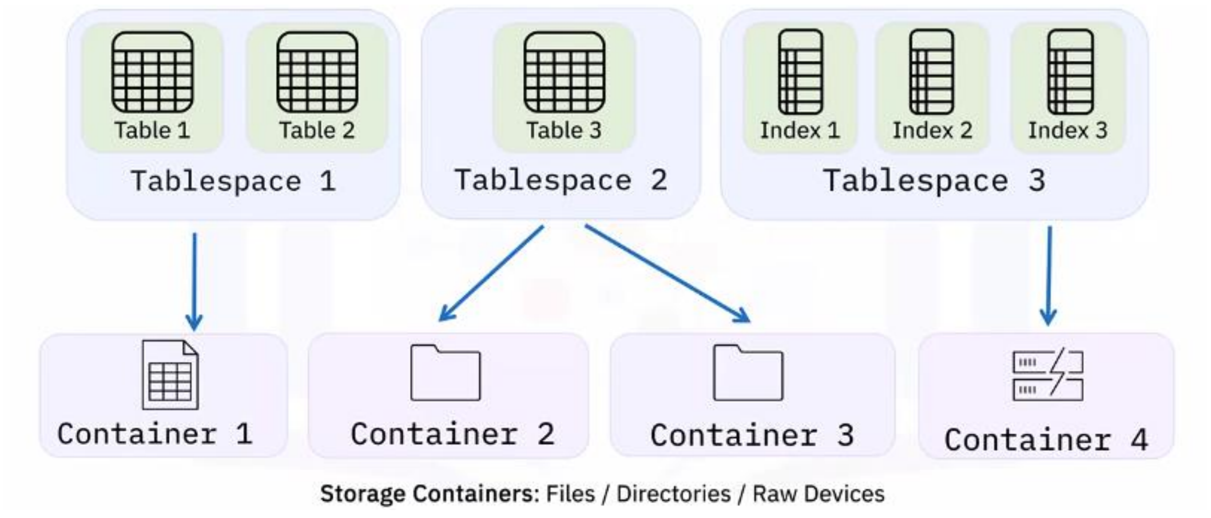
- Determine the capacity required for database plan and growth.
- In cloud-based databases expanding your storage space is performed through an API or a Graphical Console. One of the advantages of using a cloud-based database is that it is so easy to scale up and scale down in terms of storage space.
- In a self-managed or on-premises environment, you can also plan storage space for improved performance, storing competing resources on different disks.
- Manage data through a logical object without being concerned about the nature.
- For example, you can issue a command to back up a database without having to specify all the physical disks that store the database.

Tablespaces and Containers: –

- Tablespaces are structures that contain database objects such as tables, indexes, large objects, and long data.
- DBAs use tablespaces to logically organize database objects based on where their data is stored.
- Tablespaces define the mapping between logical database objects and the physical storage containers that house the data for these objects.
- A storage container can be a data file, a directory, or a raw device.
- Tablespaces can contain one or more database objects.

Example –

- In this example, Tablespace 1 contains multiple small tables, whereas Tablespace 2 only houses a single large table. Tablespace 3 contains frequently used indexes. DBAs configure one or more storage containers to store each tablespace.
- In this example, Container 1 stores Tablespace 1, Containers 2 and 3 store Tablespace 2, and Container 4 stores Tablespace 4.



Tablespaces benefits –

Tablespaces separate logical database storage separate from physical storage.

Manage the disk layout of a database and its objects.

Several benefits:

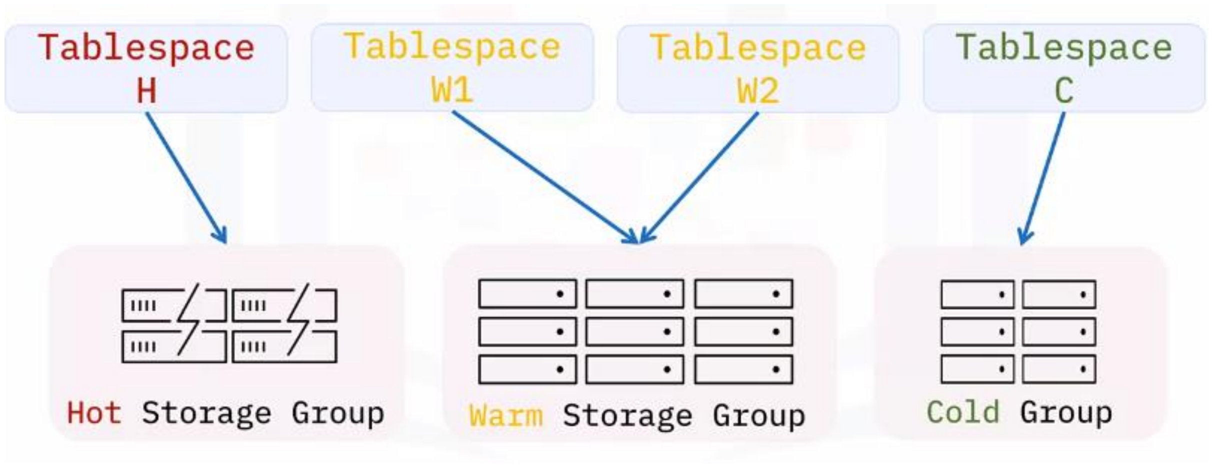
- **Performance:** You can use tablespaces to optimize performance. For example, you can place a heavily used index on a fast SSD. Alternatively, you can store tables containing rarely accessed or archived data on a less expensive but slower magnetic hard drive.
- **Recoverability:** Tablespaces make backup and restore operations more convenient. Using a single command, you can make a backup or restore all the database objects without worrying about which storage container each object or tablespace is stored on.
- **Storage Management:** The RDBMS creates and extends the datafiles or containers depending on the need. When necessary, you can also manually expand the storage space by adding another storage path or container to the tablespace.

Storage groups –

- Some RDBMSes provide Storage Groups.
- A storage group is a grouping of storage paths or containers based on similar performance characteristics.
- This allows you to perform Multi-Temperature Data Management more easily.
- Using storage groups helps with optimizing performance for frequently accessed data and reducing costs for storing infrequently accessed data.

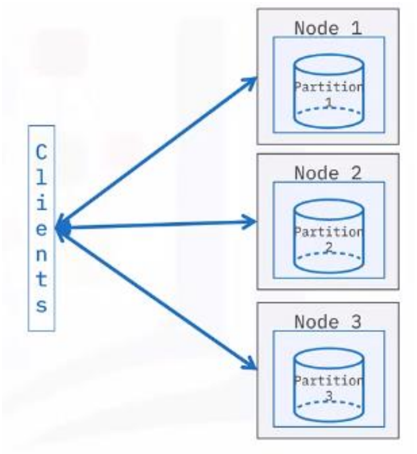
Example –

- In this context, temperature refers to the frequency of data access. Hot data is accessed very frequently, Warm data is accessed somewhat frequently, and Cold data is accessed infrequently.
- By using storage groups, you can organize your data and storage based on temperature.
- sIn this example, very frequently accessed hot tables can be placed in Tablespace H, which is distributed on a group of fast storage devices. Somewhat frequently accessed tables in Tablespaces W1 and W2 can be stored on a Warm Storage Group. And the least frequently accessed tables in Tablespace C can be stored on slower and less expensive storage devices in a Cold Group.



Database partitions –

- A partitioned relational database is a relational database whose data is managed across multiple database partitions.
- You can partition tables that need to contain very large quantities of data into multiple logical partitions, with each partition containing a subset of the overall data.
- Database partitioning is used in scenarios that involve very large volumes of data, such as data warehousing and data analysis for business intelligence.



Summary: -

- Database storage is managed through logical database objects and physical disk files.
- Tablespaces are structures that organize database objects based on where their data is stored.
- Storage groups are groupings of storage paths or containers based on similar performance characteristics.
- Partitions store subsets of data from a very large database to improve performance.

Reading: Storage Engines in MySQL

Storage engines in MySQL

A storage engine is a software component that handles the operations that store and manage information in a database. MySQL is unusual among relational databases because it supports multiple storage engines.

Each storage engine has a particular set of operational characteristics, including the types of locks to manage query contention and whether the storage engine supports transactions. These properties have implications for database performance, so choose your storage engine based on the type of data you want to store and the operations you want to perform.

Most applications require only one storage engine for the whole database, but you can specify the storage engine on a table-by-table basis if your data has different requirements.

Storage engines available in MySQL

Engines	Description
InnoDB	• Default storage engine for MySQL 5.5 and later. • Suitable for most data storage scenarios. • Provides ACID-compliant tables and FOREIGN KEY referential-integrity constraints. • Supports commit, rollback, and crash recovery capabilities to protect data. • Supports row-level locking. • Stores data in clustered indexes which reduces I/O for queries based on primary keys.
MyISAM	• Manages non-transactional tables. • Provides high-speed storage and retrieval. • Supports full-text searching.
MEMORY	• Provides in-memory tables, formerly known as HEAP. • Stores all data in RAM for faster access than storing data on disks. • Useful for quick lookups of reference and other identical data.

MERGE	• Treats groups of similar MyISAM tables as a single table. • Handles non-transactional tables.
EXAMPLE	• Allows developers to practice creating a new storage engine. • Allows developers to create tables. • Does not store or fetch data.
ARCHIVE	• Stores a large amount of data. • Does not support indexes.
CSV	• Stores data in Comma Separated Value format in a text file.
BLACKHOLE	• Accepts data to store but always returns empty.
FEDERATED	• Stores data in a remote database.

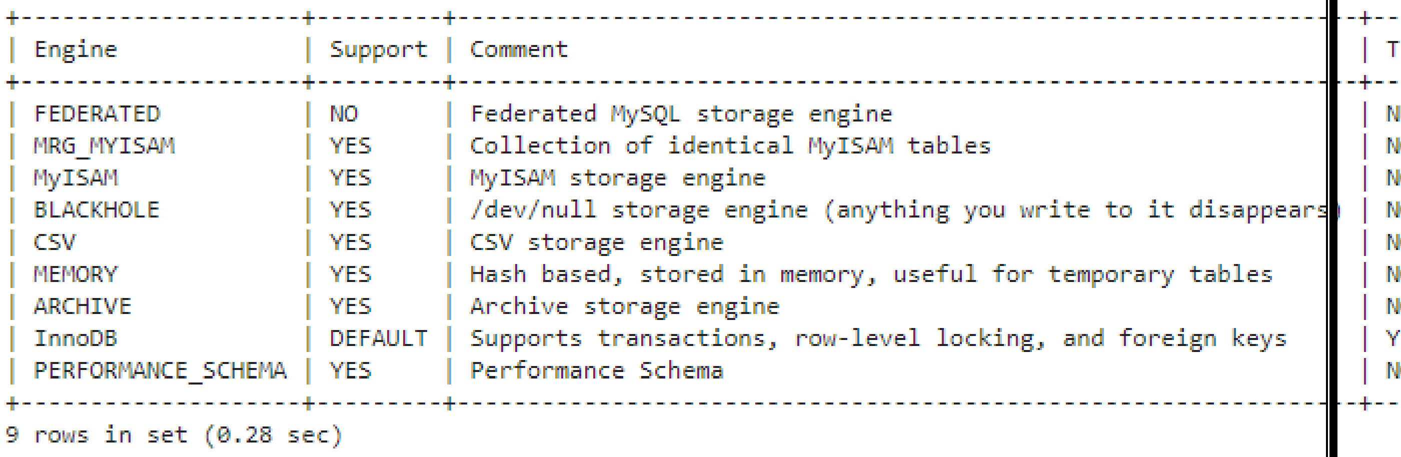
Commands for working with Storage Engines

If you're a DBA working with storage engines in MySQL, you should be familiar with the following common commands.

SHOW ENGINES

Displays status information about the server's storage engines. Useful for checking whether a storage engine is supported, or what the default engine is.

mysql> SHOW ENGINES;



```
+--------------------+---------+----------------------------------------------------------------+--
| Engine             | Support | Comment                                                        | T
+--------------------+---------+----------------------------------------------------------------+--
| FEDERATED          | NO      | Federated MySQL storage engine                                 | N
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | N
| MyISAM             | YES     | MyISAM storage engine                                          | N
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | N
| CSV                | YES     | CSV storage engine                                             | N
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | N
| ARCHIVE            | YES     | Archive storage engine                                         | N
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | Y
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | N
+--------------------+---------+----------------------------------------------------------------+--
9 rows in set (0.28 sec)
```

CREATE TABLE

Creates a table using the storage engine specified in the ENGINE clause, as shown in the following examples:

CREATE TABLE Products (i INT) ENGINE = INNODB;

CREATE TABLE Product_Codes (i INT) ENGINE = CSV;

```
CREATE TABLE History (i INT) ENGINE = MEMORY;
```

If you do not specify the ENGINE clause, the CREATE TABLE statement creates the table with the default storage engine, usually **InnoDB**.

SET

For databases with non-standard storage needs, you can specify a different default storage engine using the set command.

Set the default storage engine for the current session by setting the default_storage_engine variable using the SET command. For example, if you are creating a database to store archived data, you can use the following command:

```
SET default_storage_engine=ARCHIVE;
```

To set the default storage engine for all sessions, set the default-storage-engine option in the my.cnf configuration file.

ALTER TABLE

You can convert a table from one storage engine to another using an ALTER TABLE statement. For example, the following statement set the storage enigne for the Products table to Memory:

```
ALTER TABLE Products ENGINE = MEMORY;
```

InnoDB is suitable for most data storage needs but setting and working with different storage engines in **MYSQL** gives you more control over how your data is stored and accessed. Using the most appropriate storage engine for your data brings operational benefits like faster response times or efficient use of available storage.

Summary & Highlights:-

- An instance is a logical boundary for a database or set of databases where you organize database objects and set configuration parameters.
- Common database objects are items that exist within the database such as tables, constraints, indexes, keys, views, aliases, triggers, events, and log files.
- Different RDBMSs use different names for their system objects. Most use the terms system schema, system tables, catalog, or directory.
- Database storage is managed through logical database objects and physical storage.

Week 2

Managing Database

Module 1: - Backup and Restore Databases

1] Introduction to Backup and Restore

Backup and restore scenarios –

- Saving a copy of data for protection
- Recovering from data loss
    - After unplanned shutdown
    - Accidental deletion
    - Data corruption
- Move to a different database system
- Share data with business partners
- Use a copy of the data, e.g., dev or test

Physical vs. logical backups: –

Logical Backup –

- Contains DDL and DML commands to recreate database
- Can reclaim wasted space
- Slow and may impact performance
- Granular
- Backup/restore, import/export, dump & load utilities

Physical Backup –

- Copy of physical files, including logs, and configuration
- Smaller and quicker
- Less granular
- Can only restore to similar RDBMS
- Common for specialized storage and cloud.

What to back up? –

- You can back up a whole database
- The contents of a schema
- One or more tables from a database
- A subset of data from one or more tables in a database
- Collection of other objects in the database.

Key consideration –

- Check that your backup is valid
- Check that your restore plan works
- Ensure that your back up files are secure

<u>Backup options</u> –

Compression

- Reduces size for storage and transmission
- Increases time for backup and restore processes

Encryption

- Reduces the risk of data being compromised
- Increases time for backup and restore processes

<u>Summary: -</u>

- You can use backup and restore for data recovery and other purposes.
- Physical backups create a copy of the raw database storage files and directories whereas logical backups extract the data from a database and save it in a special format.
- You can backup whole databases or objects within them.
- You should always check that your backup is safe and usable and that your restore plan works.
- You can use backup options to compress or encrypt your files.

2] Types of Backup

Types of backup: –

1. **Full backup –**
   - Backs up all the specified data
   - Multiple copies of the backup means storing many instances of large files
   - Only storing one copy risks data loss if file is corrupt
   - Could be needlessly backing up unchanging data
   - Must secure backup files

2. **Point-in-time backup –**
   - Uses logged transactions to restore to an earlier point in time

3. **Differential backup –**
   - A copy of any data that has changed since the last full backup was taken.

4. **Incremental backup –**
   - A copy of any data that has changed since the last backup of any type was taken.

Summary: -

- Full backups are simple to create and restore, but can be slow to run and result in large files.
- Point-in-time recovery provides a more granular recovery model than just using full database backups.
- Differential backups are quicker to run than full backups, but the restore process can take longer.
- Incremental backups are even quicker to run, but the restore process can take even longer.

3] Backup Policies

Hot vs. cold backups: –

Hot backup – taken while data is in use

Cold backup – data is offline

Backup Policies: –

- Physical or logical
- Full, differential, or incremental
- Hot or cold
- Compression
- Encryption
- This is dependent on the impact that any data loss would have on your business operations and how frequently your data is changing.

Example –

- For example, a table containing product information is likely to change less frequently than a table containing customer orders, so you could back that up less regularly than the orders table. But if your orders table already contains a large amount of data, you won't want to be performing frequent full backups, so should consider using differential or incremental backups for it. You also need to consider when to perform your backups. If your data is mainly accessed during the working day in one time zone, you should schedule your backup outside of those hours. However, if it is accessed across the whole day, you could consider implementing full backups at a weekend and incremental or differential backups on a daily basis. Because backups will be a regular task, if your RDBMS provides the ability to schedule or automate your backups, you should consider using it.

  Frequency:

  - Is data regularly changing or being added?
  - Is the existing table large?

  Schedule:

  - Is the data accessed equally across 24-hour day?
  - Is it accessed at weekends?

  Automated

Managed cloud backups: -

Options dependent upon RDBMS and cloud service provider include:

- Preconfigured automated backup
- Configurable automated backup
- Manual backup
- Third party tools
- Own strategy backup

Example –

- For example, the paid plans of Db2 on Cloud run an encrypted backup every day and with the Enterprise or Standard plans, you can also run manual backups and perform point-in-time restores. And when using Google Cloud SQL for MySQL, Postgres, and SQL Server, you can enable automated incremental backups or perform on-demand backups. If you find that your combination of RDBMS and cloud provider does not support any of these options, there are third party backup tools available to purchase.

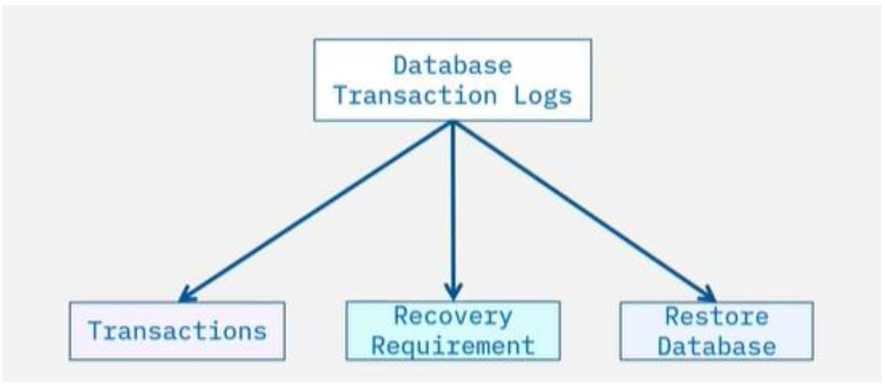
Summary: -

- Hot, or online, backups allow data to be backed up while the database is active
- Cold backups require the database to be offline.
- Your backup policy should be determined from your recovery and availability needs and your data usage patterns.
- Most managed cloud databases provide automated backup functionality with some configurable options.

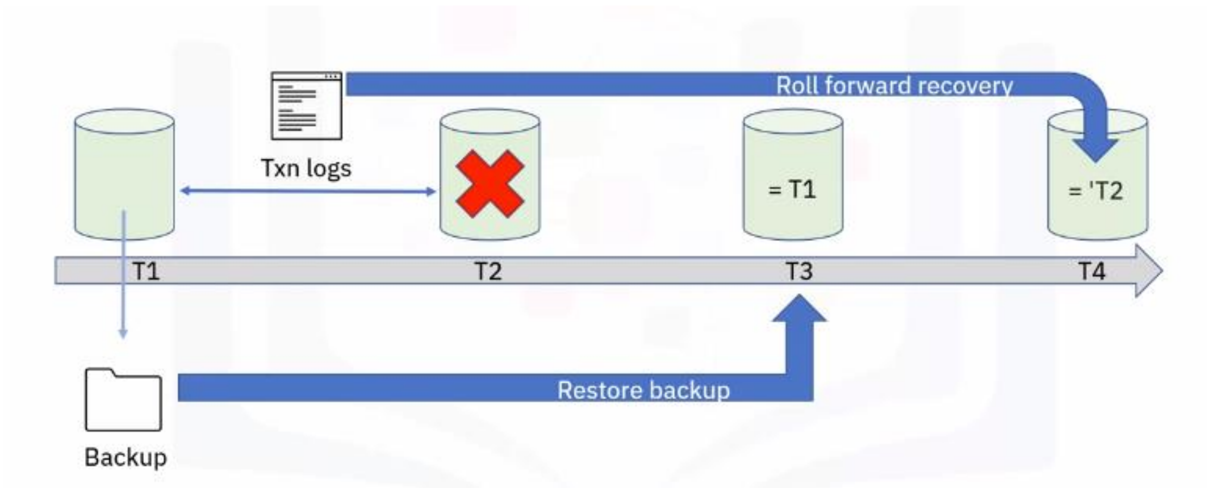
4] Using Database Transaction Logs for Recovery

Database transaction logs –

- Separate from the diagnostic log, a database management system (DBMS) uses transaction logs to keep track of all transactions that change or modify the database.
- The information stored in these transaction logs can be used for recovery purposes when data is accidentally deleted, or a system or hardware failure such as a disk crash occurs.
- In case of such events, the transaction log, in conjunction with a database backup, can help restore your database back to a consistent state at a specific point in time.



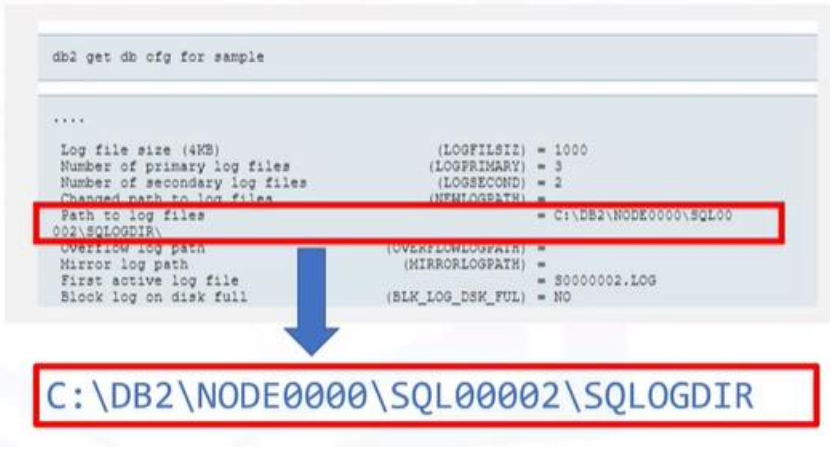
Transaction log usage example –



For example: In this timeline, at time T1 you issue the BACKUP command to make a full copy of your database. Users then continue working, and the information about their activities is kept in transaction logs. At time T2, there is a problem that damages the database. So, at time T3 you issue a RESTORE command using the backup image you took at time T1. This allows you to get the database back to the point in time that the backup was taken. Then, at time T4 you apply the transaction logs for the period of time between T1 and T2 to the restored database. This process of applying committed transactions from transaction logs is called roll forward recovery, and it returns your database to the point just before the crash happened.

Storing transaction log files –

- You typically specify where you want transaction log files to be stored using the database configuration.
- Here is an example location of a db2 transaction log file directory in a sqlogdir subdirectory on a Windows system.
- In PostgreSQL, the log files default to being stored in the pg_xlog sub-directory.
- In PostgreSQL, the transaction log file is referred to as a write-ahead log (WAL), because before the database manager writes the database changes or transactions to data files, the DBMS first writes them to the WAL.



```
db2 get db cfg for sample
```

```
....
 Log file size (4KB)                          (LOGFILSIZ) = 1000
 Number of primary log files                 (LOGPRIMARY) = 3
 Number of secondary log files                (LOGSECOND) = 2
 Changed path to log files                   (NEWLOGPATH) =
 Path to log files                                        = C:\DB2\NODE0000\SQL00
002\SQLOGDIR\
 Overflow log path                      (OVERFLOWLOGPATH) =
 Mirror log path                          (MIRRORLOGPATH) =
 First active log file                                    = S0000002.LOG
 Block log on disk full                (BLK_LOG_DSK_FUL) = NO
```

C:\DB2\NODE0000\SQL00002\SQLOGDIR

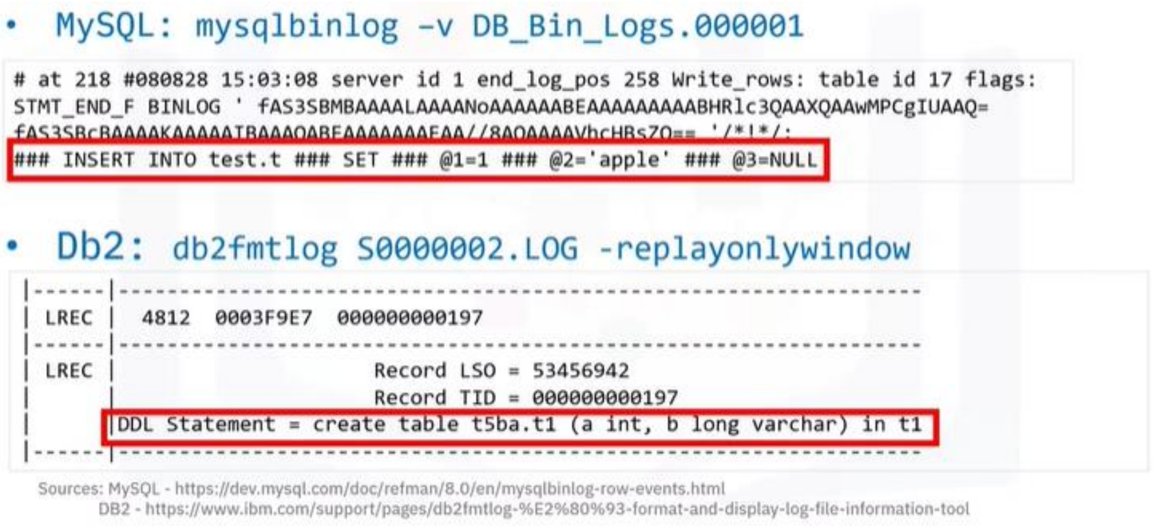
- Isolates logs on different volumes from data
- Performance – A recommended best practice, at least for production databases, is to place the transaction log files on storage volumes that are separate from where the database objects like tablespaces are stored.
- Recoverability – This not only helps with performance, so database writes are not competing with log writes, but also improves recoverability since it isolates logs from crashes or the corruption of data volumes.
- Log mirroring – Store second copy of log files in an alternative location.
- Log shipping – Copy and send logs ro replica or standby servers

Accessing transaction log files –

- Logging may or may not be enabled by default
    - Log setting are usually configurable
- MySQL – To view transaction log files:
  SHOW BINARY LOGS;
- Transaction logs:
    - Typically, in binary format
    - May be encrypted
    - View using special tools

Example –

- For example, in MySQL, you use the mysqlbinlog tool to view the contents of MySQL transaction log files, also known as binary logs. The sample output shows a log file entry for an INSERT statement. And for Db2, you use the db2fmtlog tool for formatting and displaying log file information. The sample output shows a record for a DDL statement to create a table.



- MySQL: `mysqlbinlog -v DB_Bin_Logs.000001`

```
# at 218 #080828 15:03:08 server id 1 end_log_pos 258 Write_rows: table id 17 flags:
STMT_END_F BINLOG ' fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAABHR1c3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAOABEAAAAAAAEAA//8AQAAAAVhcHBsZQ== '/*!*/;
### INSERT INTO test.t ### SET ### @1=1 ### @2='apple' ### @3=NULL
```

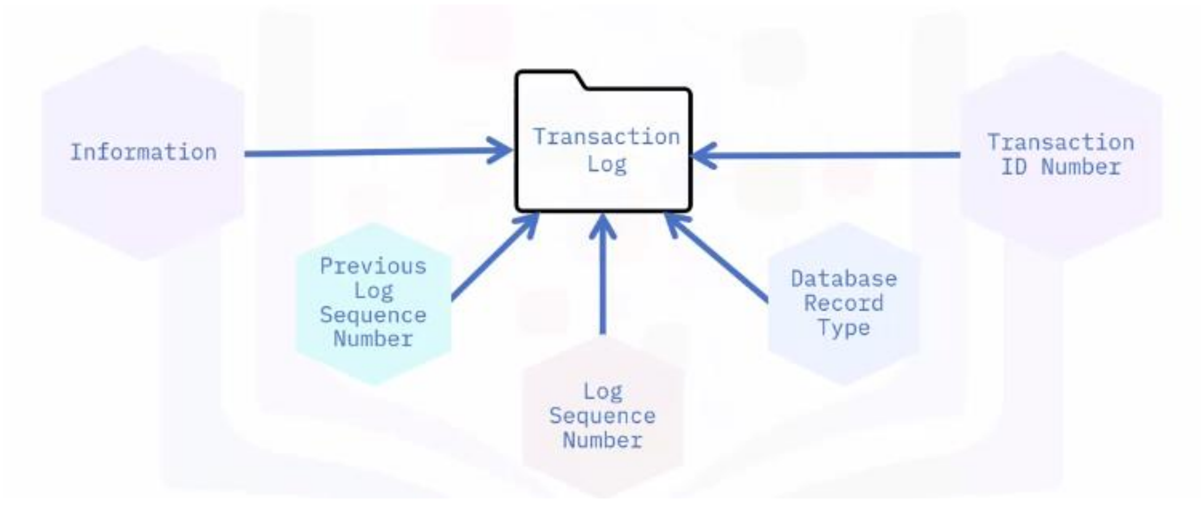
- Db2: `db2fmtlog S0000002.LOG -replayonlywindow`

```
|------|-----------------------------------------------------------------
| LREC |   4812   0003F9E7   000000000197
|------|-----------------------------------------------------------------
| LREC |                          Record LSO = 53456942
|      |                          Record TID = 000000000197
|      |DDL Statement = create table t5ba.t1 (a int, b long varchar) in t1
|------|-----------------------------------------------------------------
```

Sources: MySQL - https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog-row-events.html
DB2 - https://www.ibm.com/support/pages/db2fmtlog-%E2%80%93-format-and-display-log-file-information-tool

Basic anatomy of a database log –

- A log file contains multiple entries, and the information they contain may vary depending on the RDBMS.
- For example, a typical database log record is made up of: A log transaction ID number, which is a unique ID for the log record.
- The database record type, which describes the type of database log record. A log sequence number, which is a reference to the database transaction generating the log record. A previous log sequence number is a link to the last log record. This implies that database logs are constructed in linked list form. And information which details the actual changes that triggered the log record to be written.



<u>Summary: -</u>

- Database transaction logs keep track of all activities that change the database structure and record transactions that insert, update, or delete data in the database.
- Transaction logs assist with recovery of databases in case of a system failure, disk corruption, or accidental deletion. By applying or replaying transactions in the log the database can be recovered or rolled forward to a point in time before a data catastrophic event.
- A good practice is to store log files in a location separate from database volumes by changing default settings in the database configuration and can be accessed using command-line or graphical user interfaces.
- And the format of transaction logs varies by RDBMS but each log entry typically contains a log record ID and details about the database activity or transaction that modified the database.
- Log files contain transaction ID number, database record type, log sequence number, and other information.

Summary & Highlights:-

- The types of backups are full, point-in-time, differential, and incremental.
- The difference between physical backups and logical backups, and between hot backups and cold backups.
- Your backup policy should be determined from your recovery needs and your data usage.
- Database transaction logs keep track of all activities that change the database structure and record transactions that insert, update, or delete data in the database.

Module 2: - Security and User Management

1] Overview of Database Security

Levels of database security –

It involves identifying the potential risks to your data and mitigating those risks by applying security measures and controls at all levels of a system. Even though the actual implementation of the measures at some levels may not be your direct responsibility, it is imperative that you evaluate that they are adequate for the data you store or access there.

Server security –

On-premise servers:

- Who has access?
- How are they physically secured?

Managed cloud:

- Check provider documentation

Operating system configuration –

- Regular patching
- System hardening
- Access monitoring
- You should consider the operating system hosting your database. You should ensure that it is regularly patched with the latest tested updates, that it is hardened using a known configuration to reduce vulnerabilities, and that it is continuously monitored for any unauthorized access.

RDBMS configuration –

- Regular patching
- Review and use system-specific security features
- Reduce the number of administrations

Accessing databases and objects –

- Users and client applications need to be permitted to access a server or database and the objects in it.
- Firstly, users need to be authenticated on the server or database to enable them to access it.
- And then they need to be granted permissions on individual objects, or groups of objects, in the database to interact with them.

Authentication –

- Similar to:
    - PIN for cell phone
    - Password for computer
- Verifies that the user is who they claim to be:
    - For example, by validating username and password
- External authentication methods
    - PAM (Pluggable Authentication Module)
    - Windows login IDs
    - LDAP (Lightweight Directory Access Protocol)
    - Kerberos

Authorization –

- Authorized to access:
    - Objects
    - Data
- Grant privileges to:
    - Users
    - Groups
    - Roles

Privileges –

- Assigning privileges will vary depending on the object you are working with.
- For example, on a table you can allow users to select, insert, update, or delete data, or to alter the structure of the table.
- In some RDBMSs, you can narrow this down even further to assign privileges on a columnar basis. You should always use the principle of least privilege, that is, only allow users to access the least amount of data that they need to succeed in their tasks.

Auditing –

- **Monitor:**
  - Who accesses what objects
  - What actions they perform
- **Audit:**
  - Actual access against security plan

Encryption –

- Adds another layer of security:
  - Intruders need to decrypt
- Industry & regional regulations:
  - Algorithms
  - Key management
- Performance impact

Application security –

- The most secure combination of operating system and RDBMS can be compromised by lax application security such as SQL injection strings and insecure code.
- Therefore, it is important that you test and monitor the security of any applications interacting with your data and databases.

Summary: -

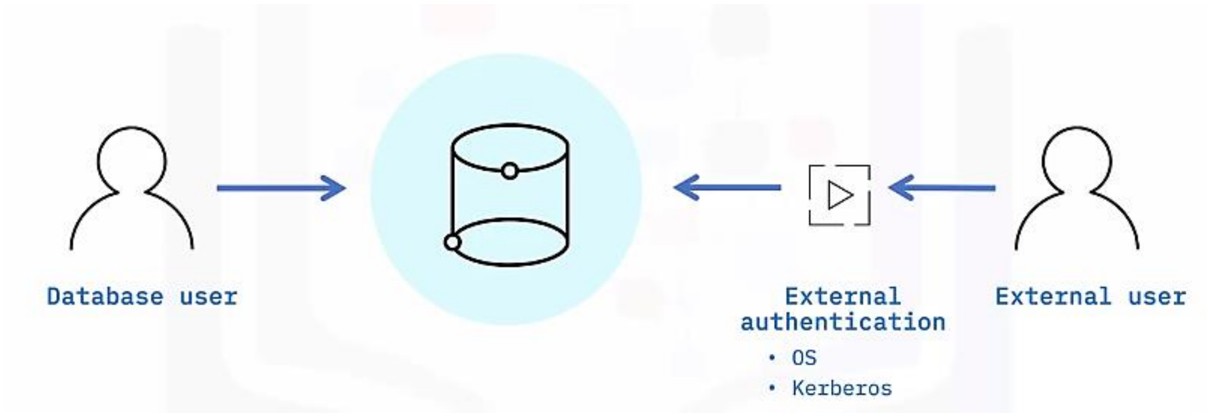
- When securing a database, you need to consider the security of the server and operating system, as well as the database and data.
- Users need to be authenticated on the server or database to access it.
- Users need to be authorized, that is given permissions or privileges, to access the objects and data in the database.
- You should use the principle of least privilege to reduce access to data.
- Monitoring and auditing database activity can alert you to threats and security gaps.

- Encryption can strengthen your data security by making it harder to read any compromised data.
- Data security includes securing your applications.

2] Users, Groups, and Roles

Database users –

- A database user is a user account that is allowed to access specified database objects.
- Depending on the DBMS and your security policies, a user might be explicitly created and authenticated within the database system, or may be created externally and authenticated using external authentication such as the operating system or external identity management services like Kerberos, LDAP, and Cloud IAM.



- In most systems, users need to be explicitly granted access to database objects.
- User names are stored in system tables or catalog tables which you should not try to edit directly.
- All RDBMSs will provide SQL commands and/or user interface tools that you can use to manage your users.

Database groups –

- In some cases, such as Postgres, you define groups in the database and they are logical groupings of users to simplify user management.
- In other systems, such as SQL Server and Db2, you can map a database group to an administrative group in the underlying operating system. This is particularly useful when you are using that operating system to authenticate your users.
- Similarly, if you are running your database on Amazon Relational Database Service (RDS) for example, you can use virtual private cloud (VPC) security groups and database (DB) security groups to manage your user access.

Database roles –

- Database roles are similar to database groups in that they confer privileges and access rights to all users of that role. A database role defines a set of permissions needed to undertake a specific role in the database. For example, a backup operator role would have permissions to access a database and to perform backup functions.
  - **Predefined roles:**
    - databaseowner
    - backupoperator
    - datareader
    - datawriter
  - **Custom roles:**
    - salesperson
    - accountsclerk
    - groupheads
    - developer
    - tester

Managing security objects –

- Assigning privileges to groups or roles, rather than individual users, greatly simplifies your security management tasks. If you know that a set of users all need access to the same functionality to fulfil their job role, you can put those users in one group and assign the relevant permissions to the group. If that job role changes in the future, it is quicker, easier, and less prone to mistakes to just add the new permissions to the group rather than individual users. Similarly, if a new member of staff joins the team, you can simply add them to the role or group, rather than having to assign all the separate permissions to them.
- One user can be a member of one or more roles. So the head of the sales group could be a member of the group heads role and the salesperson role.
- Principle of least privileges:
  - Only adding users to groups they need to be a member of,
  - And ensuring that your roles don't include any permissions that the majority of the users in them will not need.
  - In the latter scenario, you should reassess your role permissions and create multiple, more granular, roles or groups.

<u>Summary: -</u>

- Depending on the database you can create and authenticate users directly in the database system or use external authentication such as the operating system
- In some systems, you define groups in the database to manage users.

- And in others, you can map database groups to operating system groups.
- You can use predefined database roles to assign privileges to common sets of database users.
- You can define custom roles for your own requirements.
- Groups and roles simplify user management.

3] Managing Access to Databases and Their Objects

Authorization –

- After a user is authenticated on a database, they need permissions or privileges to access the objects and data in that database.
- Privileges are granted to users, and to groups or roles.
- The combination of a user's own personal permissions and those of the groups or roles they belong to defines their overall permissions.
- The RDBMS that you are using may use slightly different syntax to the examples shown here, and is also likely to provide a graphical interface option to perform these tasks.

Users
Groups or roles

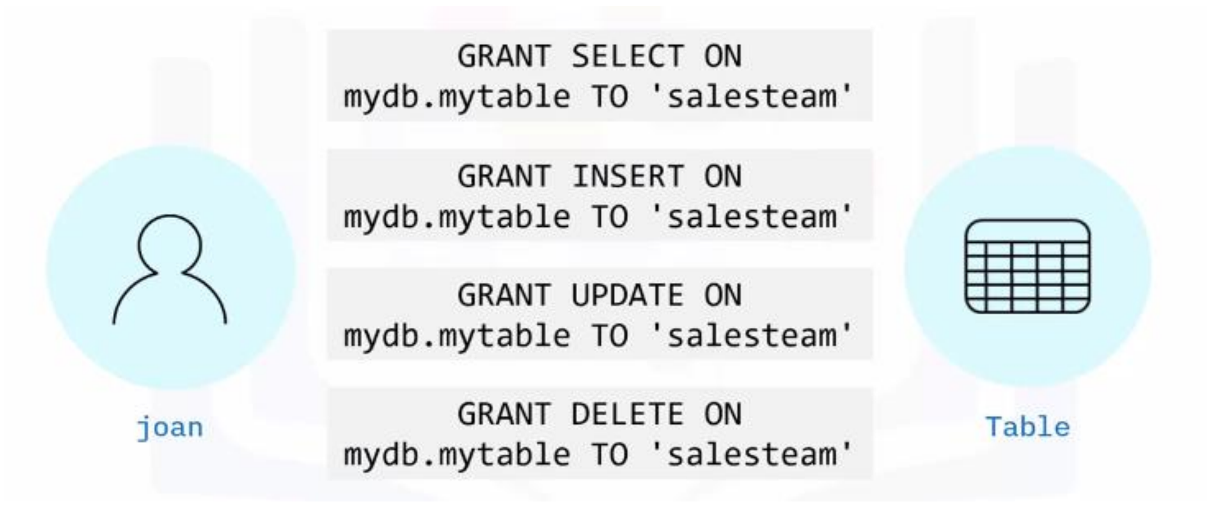
Database access –

- In systems where users are authenticated outside the database (such as on the operating system or through an external plugin), you may also need to grant them access to the databases they will be working with.
- You can use the SQL **GRANT CONNECT** command to grant connection access to a particular database to a particular user, group, or role.
- To grant connection access to an individual user rather than a group or role, you simply specify the user name in place of the group name.
- If your RDBMS provides a guest or public account, you may find that by default it has connect privileges to all databases. In this scenario it is a good practice to revoke

that permission so that you ensure only users given explicit permission can access your data.

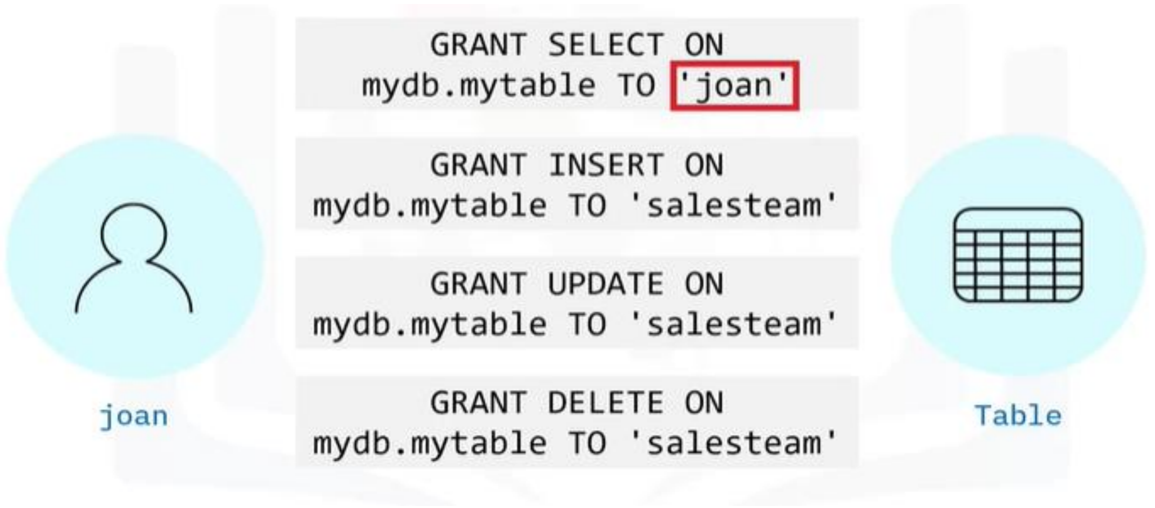
Table access –

- You use the SQL GRANT command to grant privileges on tables in a database.
- In this example, the salesteam group is granted select privileges on the mytable table in the mydb database. You can use similar statements to enable users to insert, update, and delete data by granting them the relevant privileges on a table or tables.



```
GRANT SELECT ON
mydb.mytable TO 'salesteam'

GRANT INSERT ON
mydb.mytable TO 'salesteam'

GRANT UPDATE ON
mydb.mytable TO 'salesteam'

GRANT DELETE ON
mydb.mytable TO 'salesteam'
```

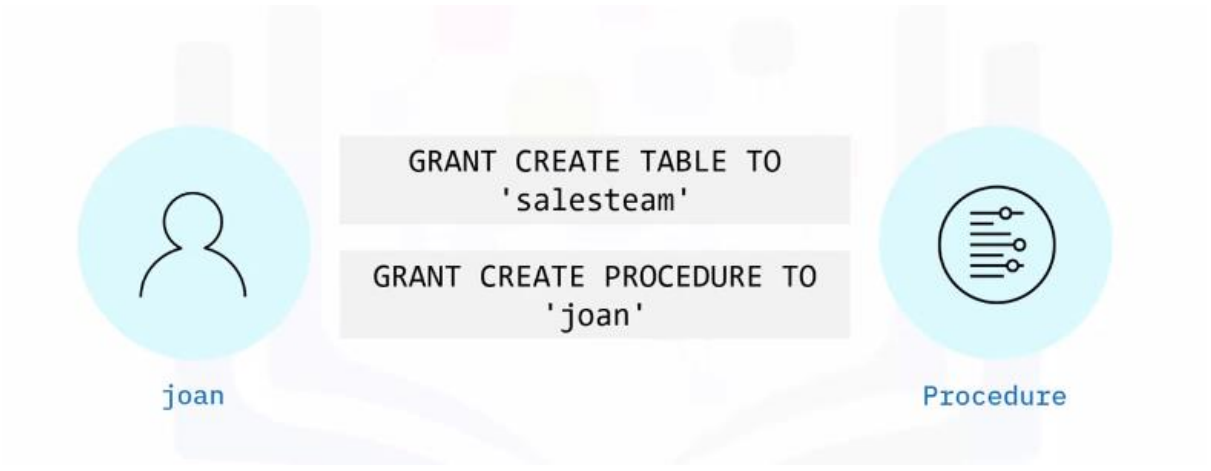
- To provide these privileges to an individual user, just replace the salesteam group name with the name of that user.



```
GRANT SELECT ON
mydb.mytable TO 'joan'

GRANT INSERT ON
mydb.mytable TO 'salesteam'

GRANT UPDATE ON
mydb.mytable TO 'salesteam'

GRANT DELETE ON
mydb.mytable TO 'salesteam'
```

Object definition access –

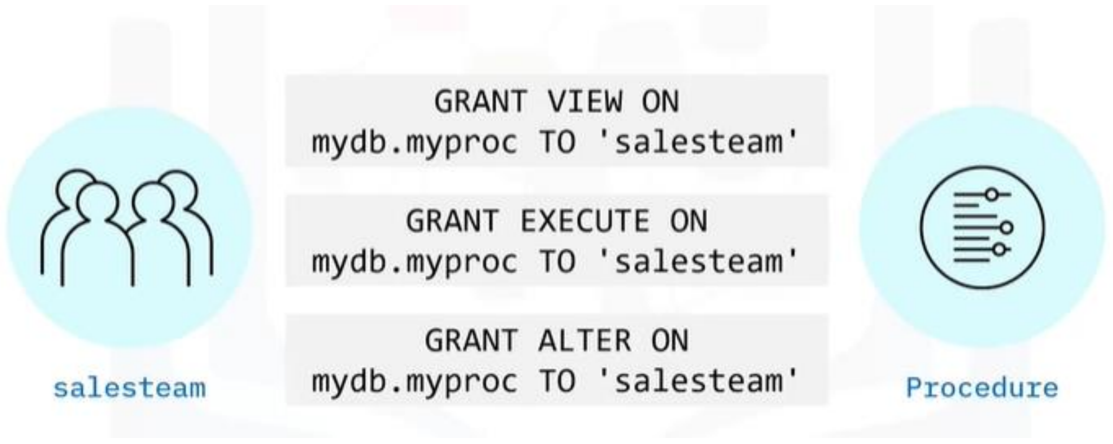
- And you can also use the GRANT statement to grant privileges to create objects in a database. This example shows how to enable a group or role to create a table. And again, you can use a similar statement to enable a user to create an object such as a table or stored procedure.



```
GRANT CREATE TABLE TO
'salesteam'

GRANT CREATE PROCEDURE TO
'joan'
```

Object access –

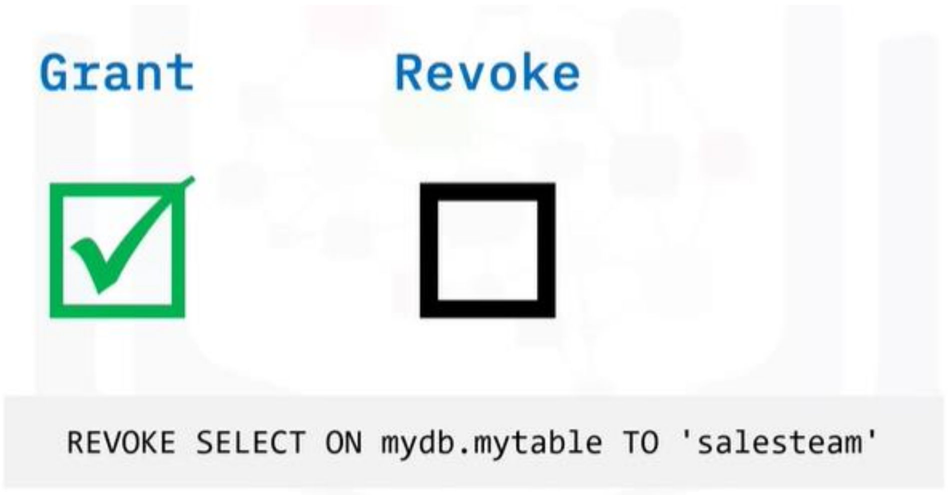
- On a procedure or function, you can permit a user or group to view the code. This means that they can view the definition of that function or procedure, but they will not be able to run it or change it. To run the code, a user needs the execute permission. And to change the definition of a procedure, a user needs the alter permission.



```
GRANT VIEW ON
mydb.myproc TO 'salesteam'

GRANT EXECUTE ON
mydb.myproc TO 'salesteam'

GRANT ALTER ON
mydb.myproc TO 'salesteam'
```

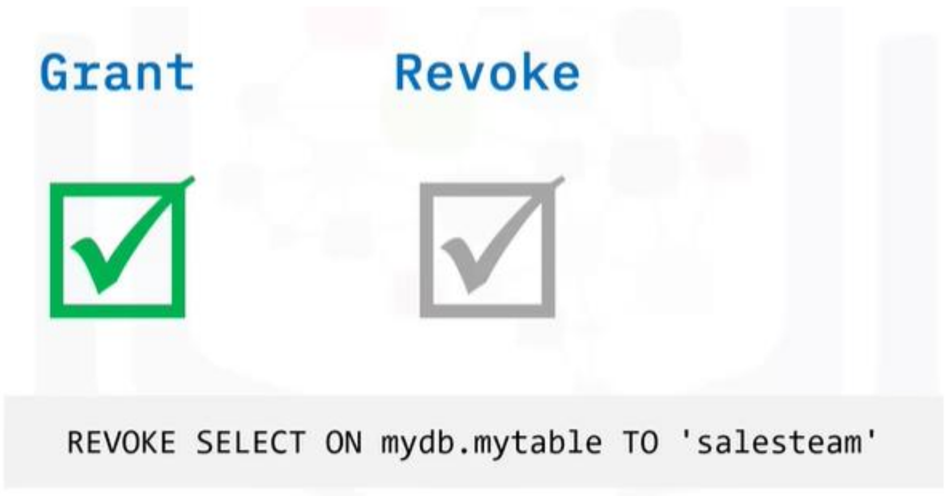
Revoke and deny access –

- As well as granting privileges, at times you may want to remove them. You can use the revoke statement to remove granted privileges, and most RDBMSs will also provide revoke functionality in the user interface.



```
REVOKE SELECT ON mydb.mytable TO 'salesteam'
```

- However, because an individual's privileges are a combination of those granted to all the groups or roles they belong to, a member of the sales team role may still be able to access this table through their membership of another role.



```
REVOKE SELECT ON mydb.mytable TO 'salesteam'
```

- If you want to ensure that users do not have permission for a certain object or action, you can use the deny statement to override any previous grant of that permission.

```
DENY SELECT ON mydb.mytable TO 'salesteam'
```

Summary: -

- You grant permissions to users, groups, or roles.
- Permissions control access to databases and the objects in them.
- The range of permissions (from select to insert, update, delete, create, alter, and drop for different types of objects) allow for fine tuning of access for users, groups, and roles.
- You can revoke a previously granted permission.
- You deny a permission to override an existing granted permission.

4] Auditing Database Activity

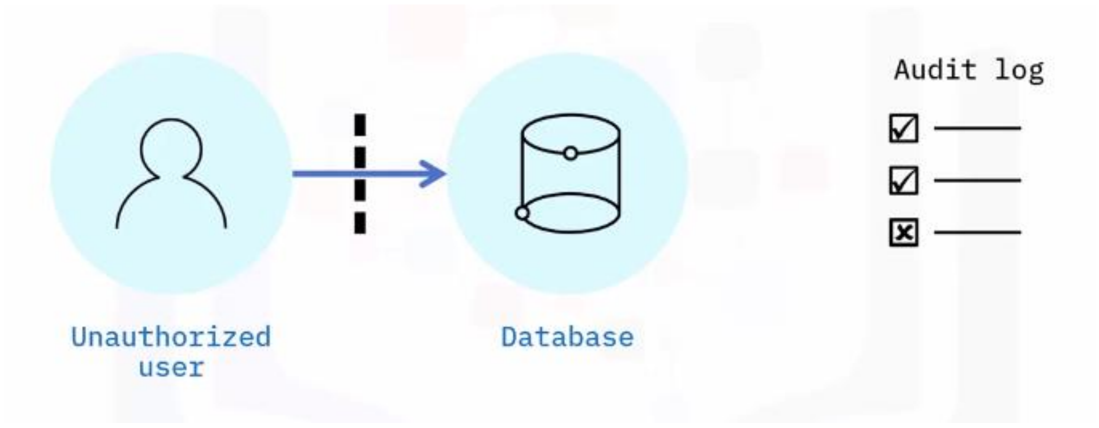
Why audit your database?

- Does not directly offer protection
- But it does help you to:
    - Identify gaps in security system
    - Track errors in privileges administration
- Compliance
- Implement on premise and in the cloud.

Auditing database access –

- Auditing a database involves recording the access and the activity of database users on your database objects. By reviewing such records, also known as logs, you can identify suspicious activity and quickly respond to any security threats you find.
- If an unauthorized user accesses your database, they have already overcome one or more of the levels of security in your system. Therefore, it is imperative that you track

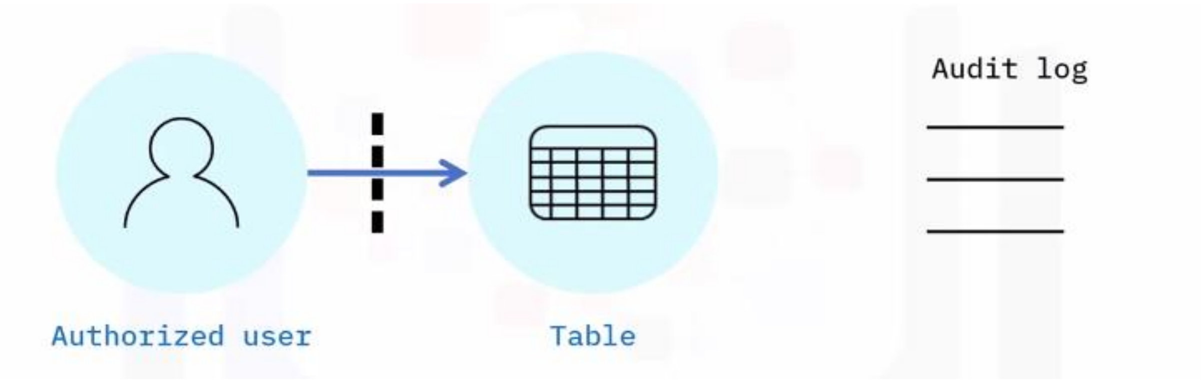
who accesses your database and review the information to identify any unauthorized users. You should also track failed attempts to access the database, as these can help you to identify potential attacks, such as brute force attempts, on your system.



- Most database systems provide functionality that you can use to audit database access. For example, in Db2 on Cloud you can use the user validation category of the built-in audit facility to log user authentication events. In MySQL the audit log plugin tracks connect and disconnect events.

Auditing database activities –

- Sometimes authorized users may find that they can view and edit data that they should not have permissions on.
- If you track all user activity, you can then use the output reports to review which users are accessing which tables and check whether those actions match your security plans.
- To audit database activity, some RDBMSs use triggers. These are special stored procedures that automatically log the activity after a DML statement event, such as an insert, occurs.
- Other systems enable you to attach actions to the events that occur in the database. In Db2 on Cloud you can use the change history event monitor to enable auditing on specific tables or objects. And in Postgres you can use the downloadable pgAudit tool to log individual action types or all actions in a database.



- You should ensure that whatever auditing implementation you use meets the compliance requirements of your customer or region.

Summary: -

- Auditing does not directly protect your database, but does identify gaps in your security.
- Some industries, customers, or regions may require that audit logs are created and retained.
- You should audit both successful and failed attempts to access your database.
- You should audit and review all activity in your database.

5] Encrypting Data

Why encrypts your data?

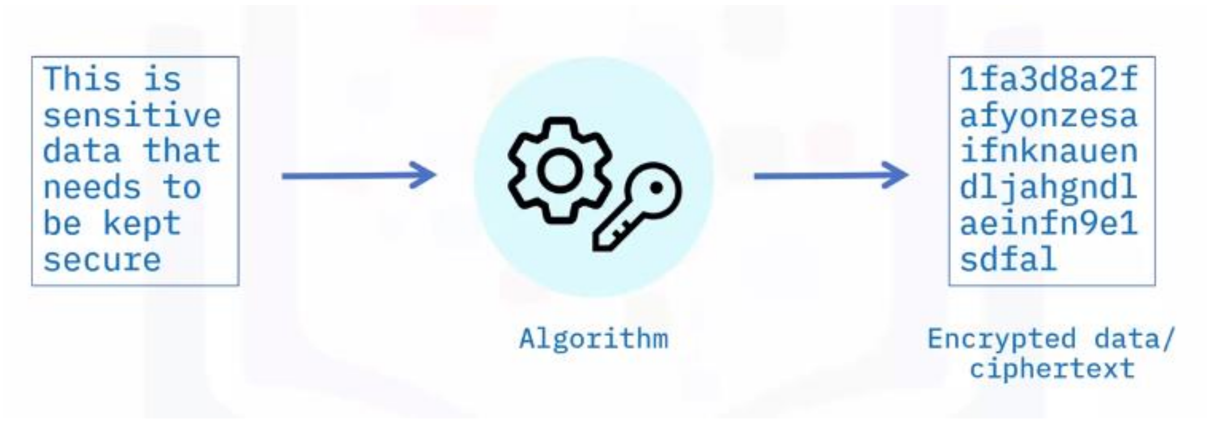
- Another layer in security system
- Often last line of defense
- Can protect data:
  - At rest
  - During transmission
- May be required by:
  - Industry
  - Region
  - Customer

Protecting data at rest: -

- "At rest data" refers to your data when it is stored, or at rest, in your RDBMS.
- It is important to encrypt data at rest to ensure that malicious users cannot directly access the data files, bypassing your database security measures.
- Some RDBMSs provide database-level encryption for the entire database. This is often termed transparent data encryption, or TDE.
- Some provide table or tablespace encryption for individual tables.
- And some provide column level for maximum granularity and flexibility.
- Do note though, that the flexibility that column level encryption provides is counteracted by performance degradation and complexity of set up.

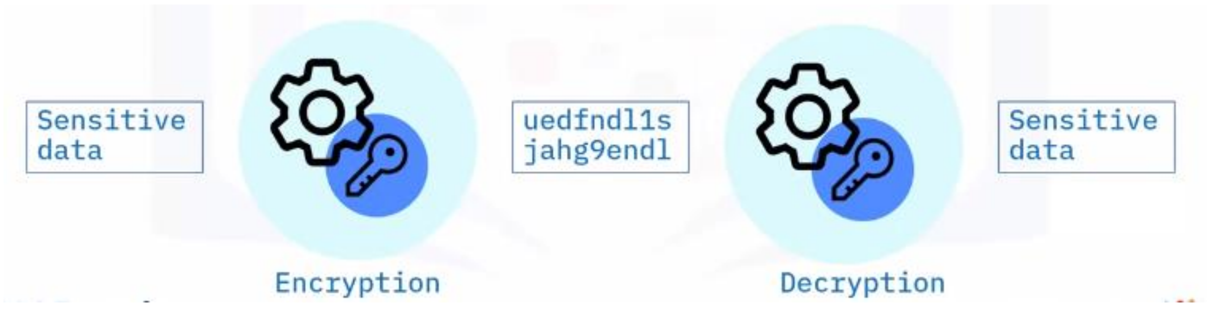
Algorithms and keys: -

- You encrypt data by using an algorithm to change the data into an unreadable string, commonly known as ciphertext. These algorithms use a key to ensure that anyone trying to decipher the text cannot do so without the key.


This is sensitive data that needs to be kept secure
Algorithm
1fa3d8a2f afyonzesa ifnknauen dljahgndl aeinfn9e1 sdfal
Encrypted data/ ciphertext

Symmetric Encryption: -

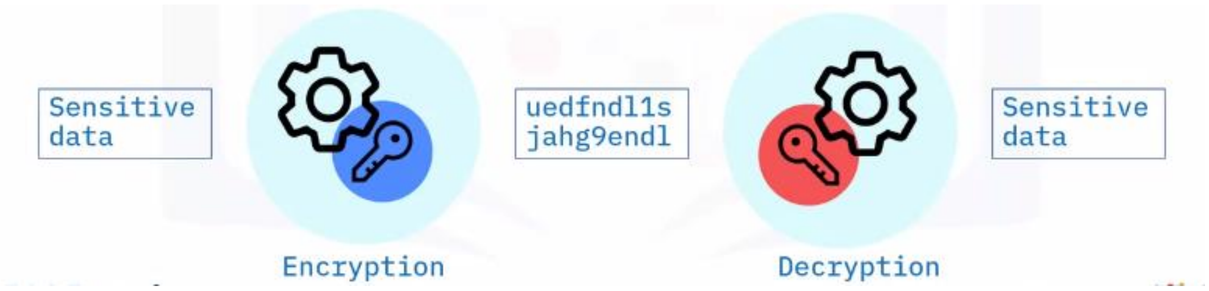
- The simplest form of encryption is symmetric encryption, where the same key is used to encrypt and decrypt the data.
- Mechanisms such as data encryption standard, or DES, and advanced encryption standard, or AES, use symmetric keys.
- When using only one key, it must be shared between all parties wanting to access that data so that they can all decrypt it.
- Sharing a key increases the likelihood of it being compromised and if the key to the encryption is compromised, your data is effectively compromised.



Asymmetric Encryption: -

- Some databases use asymmetric encryption, also known as public key encryption. This uses two keys: a public key and a private key, known as a key pair.
- You encrypt the data with the public key and valid users have a unique matching private key to decrypt it.

- Rivest-Shamir-Adleman (RSA) and elliptic curve cryptography (ECC) are both examples of asymmetric encryption.
- Because asymmetric encryption involves the use of multiple private keys, it is essential that those keys are held securely.

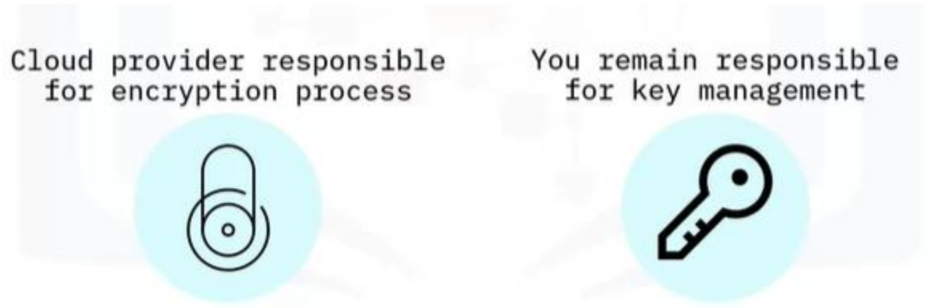


Transparent Data Encryption: -

- Encryption and key management
- Not visible to users
- Database engine encrypts and decrypts data
- Also encrypts backups

Customer managed keys: -

- Provide the data owner with more control over their data stored in the cloud
- Bring Your Own Key (BYOK)
- Benefits:
  - Cloud provider cannot access your confidential data
  - Security admins manage keys; database admins manage data
  - Complete control over keys and their lifecycle


Cloud provider responsible for encryption process
You remain responsible for key management

Encryption vs. performance: -

- Choice of symmetric or asymmetric encryption may be configurable.
- All encryption takes time and effort
- Asymmetric algorithms generally use longer keys, therefor have greater overheads
- Symmetric algorithms are often sufficient.

Protecting data at transit: -

- Often provided by the RDBMS
- Protocols:
  - TLS
  - SSL
- May encrypt by default, may be configurable
- Performance impact

Summary: -

- Data encryption helps make any data that is compromised unusable.
- You can encrypt data at rest or in transit.
- You use algorithms to encrypt and decrypt data.
- Symmetric encryption is easier to use but less secure than asymmetric encryption.
- TDE can simplify encryption setup.
- Encryption decreases database and transport performance.
- You can use TLS and SSL to protect data in transit.

Reading: User Management with DB2

Manage Db2 user accounts & roles

User management is the process of controlling which users are allowed to connect to the Db2 server and what permissions they have on each database. In Db2, users are created and assigned a specific built-in user role with a predefined set of privileges on the system. The built-in user roles are simple to use and easy to manage.

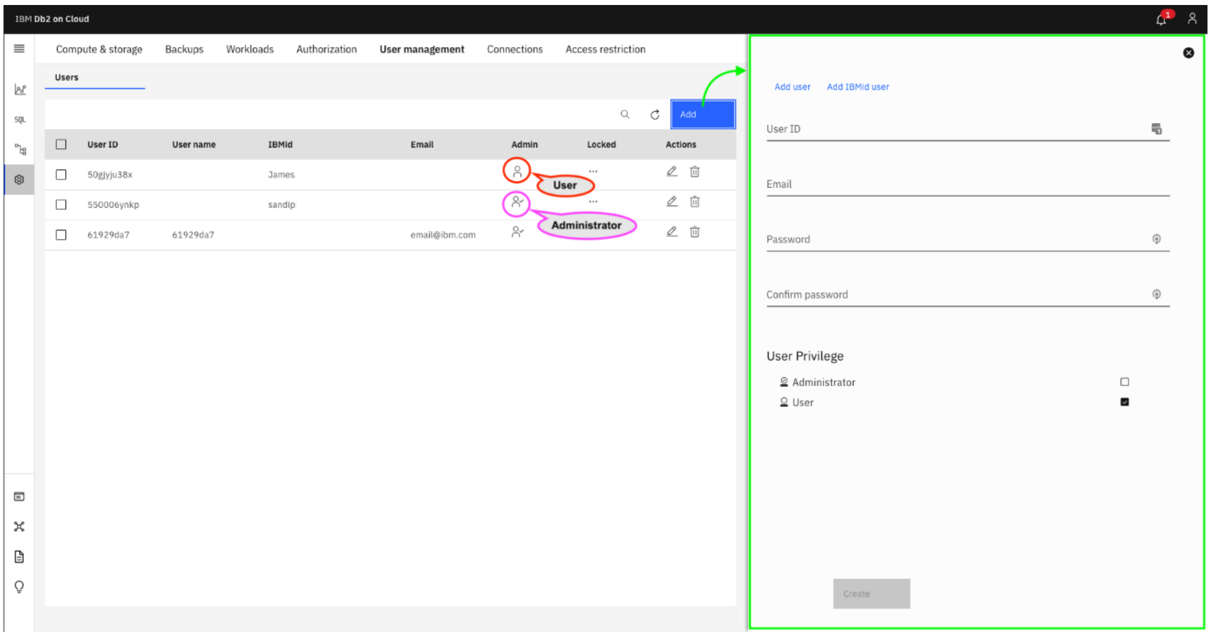
In Db2 on Cloud Standard plan, a user is created by the administrator and typically assigned one of the following built-in user roles: Administrator or User. These built-in user roles provide different levels of access commonly needed in a database system. The following general summary describes what each of the built-in user roles can do:

Administrator

- Has access to all of the features in the console, and has the privileges to manage other users.

User

- Has access to many of the features in the console, and has the privileges to manage their own user profile. Users with this role also have full access to their own tables and can use their tables with all permissions, including giving other users permission to access and use their tables.



	User ID	User name	IBMid	Email	Admin	Locked	Actions
☐	50gjyju38x		James.			...	
☐	550006ynkp		sandip			...	
☐	61929da7	61929da7		email@ibm.com			

User ID

Email

Password

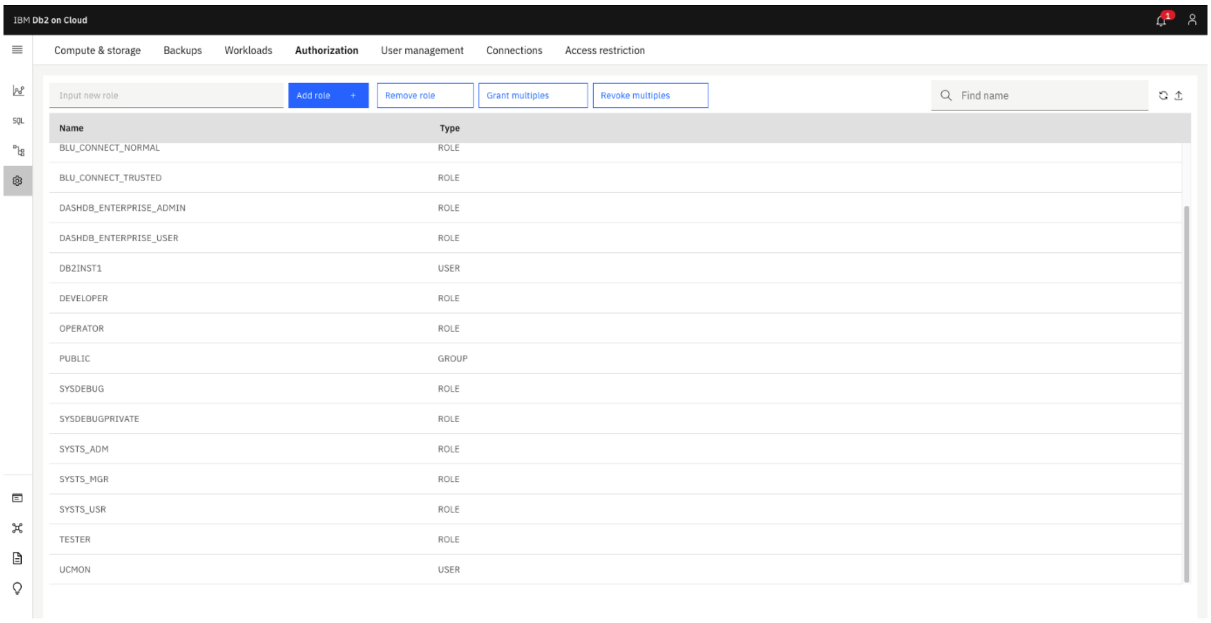
Confirm password

User Privilege

Administrator	☐
User	☑

Under **Administration** > **User management** of Db2 on Cloud Standard plan, the **Add** option allows you to create/add new user with either of the built-in user roles. You can see some created users shown in the above image, for example **James** with user privilege and **sandip** with administrator privilege.

The built-in set of user roles is simple to use and easy to manage, but if it does not provide you with the desired set of fine-grained permissions, you can also create your own user-defined user roles to satisfy your special requirements. A role is a database object that groups together one or more privileges and can be assigned to users. A user that is assigned a role receives all of the privileges of that role. A user can have multiple roles. A role hierarchy is also supported. This allows one of the roles to inherit all of the privileges and authorities of the other role or roles that they have been granted. A user-defined user role simplifies the administration and management of privileges by allowing the administrator to group authorities and privileges into a single role and then grant this role to the users that need those authorities and privileges to perform their jobs.



Name	Type
BLU_CONNECT_NORMAL	ROLE
BLU_CONNECT_TRUSTED	ROLE
DASHDB_ENTERPRISE_ADMIN	ROLE
DASHDB_ENTERPRISE_USER	ROLE
DB2INST1	USER
DEVELOPER	ROLE
OPERATOR	ROLE
PUBLIC	GROUP
SYSDEBUG	ROLE
SYSDEBUGPRIVATE	ROLE
SYSTS_ADM	ROLE
SYSTS_MGR	ROLE
SYSTS_USR	ROLE
TESTER	ROLE
UCMON	USER

You can see some created user defined roles shown in the above image, for example **DEVELOPER**, **TESTER**, **OPERATOR**, etc.

Summary & Highlights:-

- Authentication is the process of verifying that the user is who they claim to be. Authorization is the process of giving users permissions or privileges to access the objects and data in the database.
- When securing a database, you need to consider the security of the server and operating system, as well as the database and data.
- A database user is a user account that is allowed to access specified database objects. Groups are logical groupings of users to simplify user management. A database role defines a set of permissions needed to undertake a specific role in the database.
- When implementing role or group membership, use the principle of least privilege.
- Auditing does not directly protect your database but does identify gaps in your security.
- Customer-managed keys provide the data owner with more control over their data stored in the cloud.

Week 3

Monitoring and Optimization

Module 1: - Monitoring and Optimization

1] Overview of Database Monitoring

What is database monitoring?

- Critical part of database management.
- Scrutinization of day-to-day operational database status.
- Crucial to maintain RDBMS health and performance.

Why monitor your databases?

- Regular database monitoring helps identify issues in a timely manner.
- If you do not monitor, database problems might go undetected.
- RDBMSs offer tools to observe database state and track performance.

Database monitoring tasks –

- Forecasting your future hardware requirements based on database usage patterns.
- Analyzing the performance of individual applications or database queries.
- Tracking the usage of indexes and tables.
- Determining the root cause of any system performance degradation.
- Optimizing database elements to provide the best possible performance.
- And assessing the impact of any optimization activities.

Reactive monitoring –

- Done after issue occurs – in direct response to the issue.
- Typical scenarios include security breaches and critical performance levels.

Proactive monitoring –

- Prevents reactive panic by identifying issues before they grow larger.
- Observes specific database metrics and sends alerts if values reach abnormal levels.
- Uses automated processes.
- Best strategy and preferred by most database admins.

Establish a performance baseline –

- Determines whether your database system is performing at its most optimal
- Record key performance metrics at regular intervals over a given time period.
- Compare baseline statistics with database performance at any given time.
- If measurements are significantly above or below baseline = analyze and investigate further.
- Use your performance baseline to determine operational norms:
- Peak and off-peak hours of operation
- Typical response times for running queries and processing batch commands
- Time taken to perform database backup and restore operations

Baseline data –

The following areas typically have the greatest effect on the performance of your database system:

- System hardware resources
- Network architecture
- Operating system
- Database applications
- Client applications

Database monitoring options –

Point-in-time (manual) –

- Monitoring table functions
- Examine monitor elements and metrics
- Lightweight, high-speed monitoring infrastructure

Historical (Automated) –

- Event monitors
- Capture info on database operations
- Generate output in different formats

<u>Summary: -</u>

- Database monitoring is crucial to maintain the health and performance of your relational database management system.

- Proactive monitoring seeks to prevent a reactive panic by identifying issues before they grow into larger problems.
- You need to establish a baseline for your database system's performance, to determine whether your database system is performing at its most optimal.
- To monitor operations in your database, you can either view the state of various elements of your database at a specific point in time, or you can set up event monitors to capture historical information as specific types of database events occur over time.

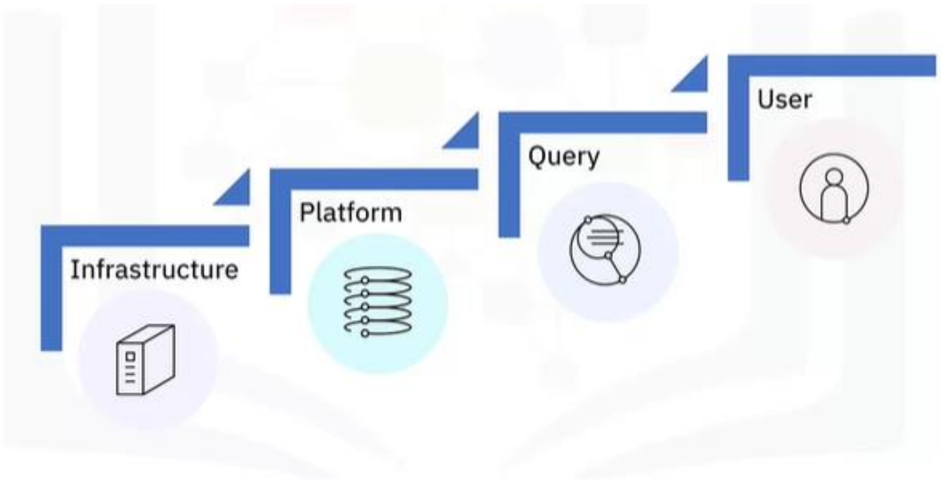
2] Monitoring Usage and Performance – Part 1

Monitoring usage and performance –

- Need performance indicators (KPIs) to measure database usage and performance.
- More commonly referred to as 'metrics'
- Metrics enable DBAs to optimize organizations' database for best performance.
- Regular monitoring also useful for operations, availability, and security.

Monitoring at multiple levels –

- Ultimate goal of monitoring is to identify and prevent issues from adversely affecting database performance.
- Issues might be caused by:
  - Hardware
  - Software
  - Network connections
  - Queries
  - Something else
- Database monitoring should be multilevel.



- Underlying infrastructure components:
    - OS
    - Servers
    - Storage hardware
    - Network components
- Managing Db2, PostgreSQL, MySQL, or any other RDBMS
- Offers holistic insight into all elements necessary for consistent database performance.
- LOB apps repeatedly run queries against database.
- Most bottlenecks due to inefficient query statements:
    - Cause latency
    - Mishandle errors
    - Diminish query throughput and concurrency
- Successful monitoring happens continually and proactively:
    - Monitoring *nirvana* is achieved when you identify issues before users are aware of them.
- Monitoring at all levels is crucial to maintaining SLAs:
    - High availability
    - High uptime
    - Low Latency

Summary: -

- Database performance is measured by using key performance indicators, known as metrics.
- Metrics enable DBAs to optimize organizations' databases for best performance.
- Successful database performance monitoring means monitoring at multiple levels.
- You should monitor at the infrastructure, platform, query, and user levels.

3] Monitoring Usage and Performance – Part 2

Key database metrics –

- **Database throughput** is one of the most significant metrics of database performance. It indicates how much total work is being taken on by your database and is typically measured by the number of queries executed per second.
- **Database resource usage** monitors the database resource usage by measuring the CPU, memory, log space, and storage usage. This summary metric represents the database resource usage by two aspects: average/max/latest number and time series number.

- **Database availability** signals whether the database is up or down, that is, available or unavailable. It is typically a summary metric that represents the historical data on available time as a percentage.
- **Database responsiveness** shows how well the system is responding to inbound requests and is another of the more commonly used database performance metrics. It provides DBAs with information on the average response time per query for their database servers, indicating how quickly they respond with query results.
- **Database contention** indicates whether there is any contention between connections, by measuring lock-waits and concurrent database connections. Database contention is the term used to describe what happens when multiple processes are competing to access the same database resource at the same time.
- **Units of work** tracks what transactions (units of work) are consuming the most resources on the database server.
- **Connections** can display all kinds of network connection information to a database management console and can indicate whether a database server might fail due to long-running queries or having too many open connections. Database connections are network connections that enable communication between clients and database software. Open connections are used for sending commands and receiving responses in the form of result sets.
- **Most-frequent queries** tracks the most frequent queries received by your database servers, including their frequency and average latency, that is, how long they take to be processed. It can help DBAs optimize these queries to gain substantial performance improvements.
- **Locked objects** shows detailed information about any locked processes and the process that blocked them. Locks and blocks stop several concurrent transactions from accessing an object at the same time. They put contending processes on hold until that object has been released and is accessible again.
- **Stored procedures** displays the aggregated execution metrics for procedures, external procedures, compiled functions, external functions, compiled triggers, and anonymous blocks invoked since database activation.
- **Buffer pools** tracks the usage of buffer pools and table spaces. A directory server uses buffer pools to store cached data and to improve database performance. When the buffer pool fills up, it removes older and less-used data to make room for newer data.
- **Top consumers** shows the top consumers of a system's resources and can help DBAs with capacity planning and resource placement.

Monitoring tools: –

Db2 –

- Db2 Data Management Console
- Workload manager
- Snapshot monitors

PostgreSQL –

- pgAdmin dashboard (downloadable open-source tool)

MySQL –

- MySQL Workbench: Performance Dashboard
- MySQL Workbench: Performance Reports
- MySQL Workbench: Query Statistics
- MySQL Query Profiler

Third-party Monitoring tools: –

- pganalyze (PostgreSQL)
- PRIG Network Monitor (PostgreSQL, MySQL, SQL Server, Oracle)
- Available for multiple database systems:
- Solarwinds Database Performance Analyzer
- Quest Foglight for Databases
- Datadog (database, system, and application monitoring)

Summary: -

- There are many different metrics you can use to monitor the usage and performance of your databases.
- There are several different tools available to monitor your databases and queries.

4] Optimizing Databases

Database optimization –

Why do you need to optimize your databases?

- Identify bottlenecks
- Fine-tune queries
- Reduce response times

RDBMS have their own optimization commands

- MySQL OPTIMIZE TABLE command
- PostgreSQL VACCUM and REINDEX commands
- Db2 RUNSTATS and REORG commands

MySQL OPTIMIZE TABLE command –

- After significant amount of insert, update, or delete operations, databases can get fragmented.
- OPTIMIZE TABLE reorganizes physical storage of table data and associated index to reduce storage space and improve efficiency
- Requires SELECT and INSERT privileges
  OPTIMIZE TABLE accounts, employees, sales;
- Optimizes three table in one operation
- You can also use phpMyAdmin graphical tool

PostgreSQL VACUUM command –

- Garbage collection for PostgreSQL databases
- Can also analyze (optional parameter)
- Reclaims lost storage space consumed by 'dead' tuples.
- Regular use can help database optimization and performance
- **Autovaccum** does this for you (if enabled)

Examples:

VACUUM

- Frees up space on all tables

VACUUM tablename

- Frees up space on specific table

VACUUM FULL tablename

- Reclaims more space, locks database table, takes longer to run

PostgreSQL REINDEX command –

- Rebuild an index using the data stored in the index's table and replace the old version
- Must be owner of index, table, or database
- Reindexing has similar effect as dropping and recreating an index
  REINDEX INDEX myindex;

- Rebuilds a single index
  REINDEX TABLE mytable;
- Rebuilds all indexes on a table

Db2 RUNSTATS command –

- Updates statistics in system catalog about characteristics of tables, associated indexes, or statistical views
  - Characteristics include # of records, # of pages, avg record length
- For a table, call RUNSTATS when table has had many updates, or after table is reorganized.
- For a statistical view, call RUNSTATS when changes to underlying tables substantially affect rows returned by the view
  - View must be previously enabled for use in query optimization by using the ALTER VIEW statement

Examples:

RUNSTATS ON TABLE employee

WITH DISTRIBUTION ON COLUMNS (empid, empname)

- Collects stats on table only, with distribution stats on 'empid' and 'empname' columns.

RUNSTATS ON TABLE employee

for indexes empl1, empl2

- Collects stats on a set of indexes.

Db2 REORG TABLE command –

- Reorganizes a table by reconstructing rows to eliminate fragmented data and by compacting.
- On a partitioned table, you can reorganize a single partition
- Affects all database partitions in database partition group
  db2 REORG TABLE employee USE mytemp1
  Reorganizes a table to reclaim space and uses the temporary table space 'mytemp1'.

Db2 REORG INDEX command –

- Reorganize all indexes that are defined on a table by rebuilding the index data into unframed, physically contiguous pages
- Use CLEANUP option to perform cleanup without rebuilding indexes

- Affects all database partitions in database partition group

Summary: -

- Optimizing databases can improve response times and increase overall performance.
- Relational database management systems provide their own commands for optimizing databases.
- In MySQL, you can use the OPTIMIZE TABLE command.
- In PostgreSQL, you can use the VACUUM and REINDEX commands.
- In Db2, you can use the RUNSTATS and REORG commands.

5] Using Indexes

What is a database index?

Similar to a book index

- Helps you quickly find information
- No need to search every page (table)

Searching entire large database can be slow.

A database index can significantly improve database search performance.

Ordered copy of selected columns of data

Enables efficient searches without searching every row

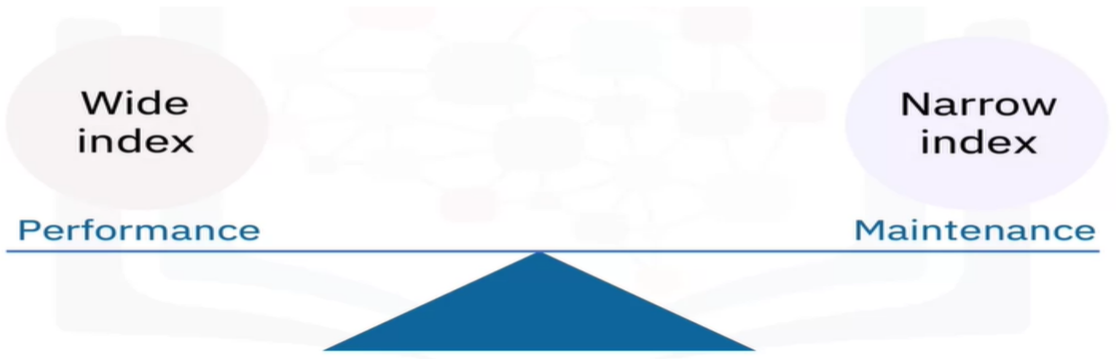
Data columns defined by admins based on factors

- Frequently searched terms, customer ID

'Lookup table' points to original rows in table

Can include one or more columns

Creating effective indexes –



- However, while indexes can improve database performance, they also require additional storage space and maintenance because they must be continually updated.
- For any database, you will likely have to experiment with different types of indexes and indexing options.
- This will help you find an optimal balance of query performance and index maintenance—such as how much disk space and activity are required to keep the index up to date.
- For example, narrow indexes, or indexes with relatively few columns in the index key, take up less disk space and require less overhead to maintain, whereas wide indexes, while they may cover a greater number of queries, also require more space and maintenance.

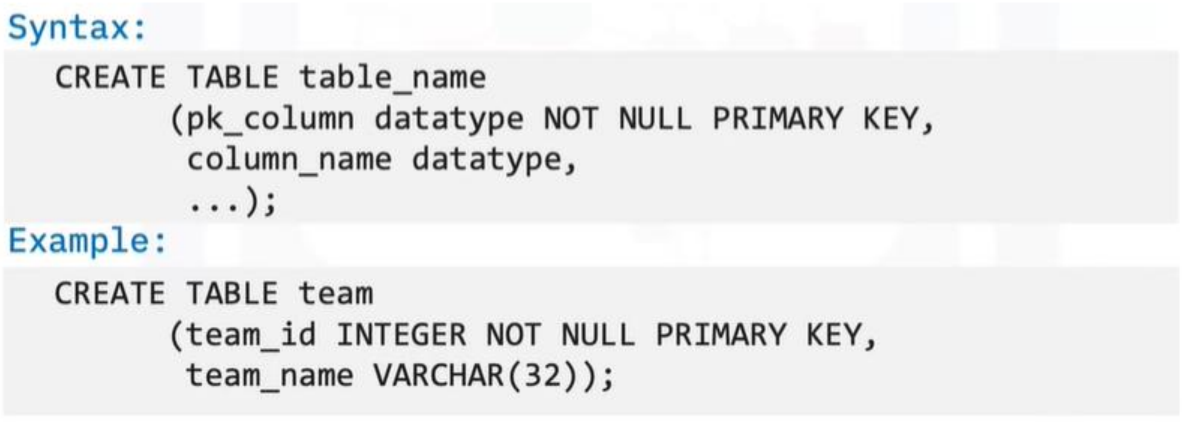
Types of database indexes –

- **Primary Key**
  - Always unique, non-nullable, one per table
  - Clustered – data stored in table in order by primary key
- **Indexes**
  - Non-clustered, single or multiple columns
  - Unique or non-unique
- **Column ordering is important**
  - Ascending (default) or descending
  - First column first, then next, and so on.

Creating primary keys –

- Uniquely identifies each row in a table
- Good practice to create when creating the table

Syntax & Example –


Syntax:

```
CREATE TABLE table_name
      (pk_column datatype NOT NULL PRIMARY KEY,
       column_name datatype,
       ...);
```

Example:

```
CREATE TABLE team
      (team_id INTEGER NOT NULL PRIMARY KEY,
       team_name VARCHAR(32));
```

Primary key with multiple columns

Syntax –

```
CREATE TABLE table_name
      (column_1_name datatype NOT NULL,
       column_2_name datatype NOT NULL,
       ...
       PRIMARY KEY(column_1_name, column_2_name));
```

Ensure uniqueness with auto-incrementing values.

Db2: IDENTITY column

```
CREATE TABLE team
       (team_id INT GENERATED BY DEFAULT AS IDENTITY
                    PRIMARY KEY,
        team_name VARCHAR(32));
```

MySQL: AUTO_INCREMENT column

```
CREATE TABLE player
       ( player_id SMALLINT NOT NULL AUTO_INCREMENT,
         player name CHAR(30) NOT NULL,
         PRIMARY KEY (player_id) );
```

Creating indexes –

Syntax –

```
CREATE INDEX index_name
      ON table_name (column_1_name, column_2_name, ...);
```

Example –

```
CREATE UNIQUE INDEX unique_nam
      ON project (projname);

CREATE INDEX job_by_dpt
      ON employee (workdept, job);
```

Dropping indexes –

Syntax –

```
DROP INDEX index_name;
```

Example –

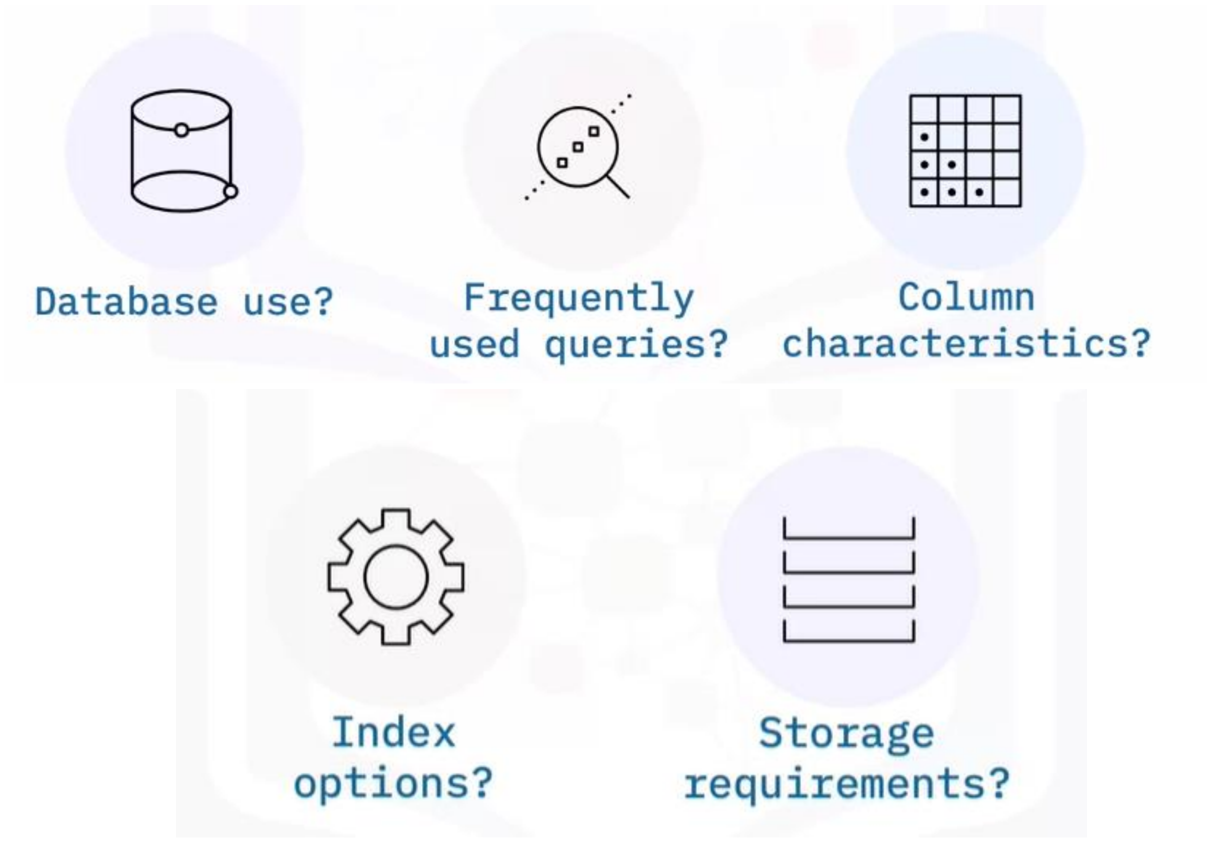
```
DROP INDEX job_by_dpt;
```

Primary key/unique key indexes cannot be explicitly dropped using this method

- Use ALTER TABLE statement instead.

Considerations when creating indexes –

Major source of database bottlenecks is poorly designed or insufficient indexed



Summary: -

- Database indexes can significantly improve search performance for a database.
- Database indexes can be either primary key indexes or non-primary indexes, that is, standard indexes.
- Primary keys can be defined when creating tables.
- The CREATE INDEX statement is used to define an index on a database table, and the DROP statement is used to drop, or delete, an index.
- A primary key, or unique key, index cannot be explicitly dropped using the DROP statement, it must be dropped by using the ALTER TABLE statement first.
- One of the major sources of database application bottlenecks is poorly designed indexes, or a lack of sufficient, appropriate indexes.
- Designing efficient indexes is vital for getting optimal database application performance.

Reading: Improving Performance of Slow Queries in MySQL

Common Causes of Slow Queries

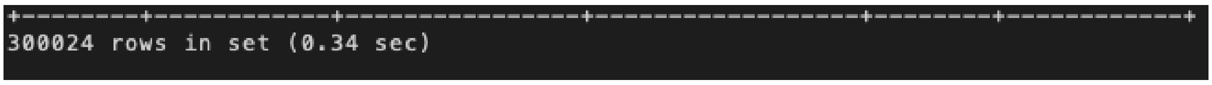
Sometimes when you run a query, you might notice that the output appears much slower than you expect it to, taking a few extra seconds, minutes or even hours to load. Why might that be happening?

There are many reasons for a slow query, but a few common ones include:

1. The size of the database, which is composed of the number of tables and the size of each table. The larger the table, the longer a query will take, particularly if you're performing scans of the entire table each time.

2. Unoptimized queries can lead to slower performance. For example, if you haven't properly indexed your database, the results of your queries will load much slower.

Each time you run a query, you'll see output similar to the following:

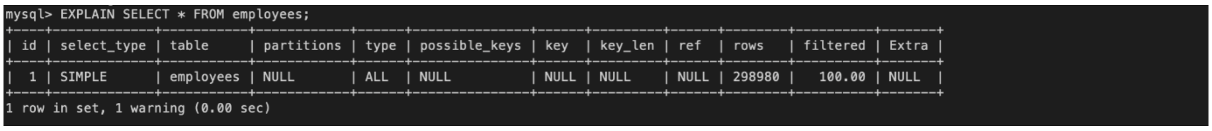


```
+--------+------------+---------------+------------------+--------+------------+
300024 rows in set (0.34 sec)
```

As can be seen, the output includes the number of rows outputted and how long it took to execute, given in the format of `0.00` seconds.
One built-in tool that can be used to determine why your query might be taking a longer time to run is the `EXPLAIN` statement.

EXPLAIN Your Query's Performance

The EXPLAIN statement provides information about how MySQL executes your statement—that is, how MySQL plans on running your query. With EXPLAIN, you can check if your query is pulling more information than it needs to, resulting in a slower performance due to handling large amounts of data.
This statement works with SELECT, DELETE, INSERT, REPLACE and UPDATE. When run, it outputs a table that looks like the following:



```
mysql> EXPLAIN SELECT * FROM employees;
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
|  1 | SIMPLE      | employees | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 298980 |   100.00 | NULL  |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

As shown in the outputted table, with SELECT, the EXPLAIN statement tells you what type of select you performed, the table that select is being performed on, the number of rows examined, and any additional information.
In this case, the EXPLAIN statement showed us that the query performed a simple select (rather than, for example, a subquery or union select) and that 298,980 rows were examined (out of a total of about 300,024 rows).
The number of rows examined can be helpful when it comes to determining why a query is slow. For example, if you notice that your output is only 13 rows, but the query is examining about 300,000 rows—almost the entire table!—then that could be a reason for your query's slow performance.

In the earlier example, loading about 300,000 rows took less than a second to process, so that may not be a big concern with this database. However, that may not be the case with larger databases that can have up to a million rows in them.

One method of making these queries faster is by adding indexes to your table.

Indexing a Column

Think of indexes like bookmarks. Indexes point to specific rows, helping the query determine which rows match its conditions and quickly retrieves those results. With this process, the query avoids searching through the entire table and improves the performance of your query, particularly when you're using SELECT and WHERE clauses.

There are many types of indexes that you can add to your databases, with popular ones being regular indexes, primary indexes, unique indexes, full-text indexes and prefix indexes.

Type of Index	**Description**
Regular Index	An index where values do not have to be unique and can be NULL.
Primary Index	Primary indexes are automatically created for primary keys. All column values are unique and NULL values are not allowed.
Unique Index	An index where all column values are unique. Unlike the primary index, unique indexes can contain a NULL value.
Full-Text Index	An index used for searching through large amounts of text and can only be created for **char**, **varchar** and/or **text** datatype columns.
Prefix Index	An index that uses only the first N characters of a text value, which can improve performance as only those characters would need to be searched.

Now, you might be wondering: if indexes are so great, why don't we add them to each column?

Generally, it's best practice to avoid adding indexes to all your columns, only adding them to the ones that it may be helpful for, such as a column that is frequently accessed. While indexing can improve the performance of some queries, it can also slow down your inserts, updates and deletes because each index will need to be updated every time. Therefore, it's important to find the balance between the number of indexes and the speed of your queries.

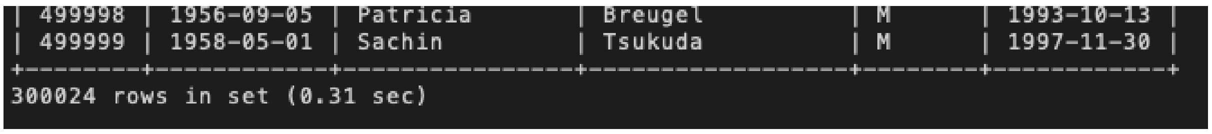
In addition, indexes are less helpful for querying small tables or large tables where almost all the rows need to be examined. In the case where most rows need to be examined, it would be faster to read all those rows rather than using an index. As such, adding an index is dependent on your needs.

Be SELECTive With Columns

When possible, avoid selecting all columns from your table. With larger datasets, selecting all columns and displaying them can take much longer than selecting the one or two columns that you need.

For example, with a dataset of about 300,000 employee entries, the following query takes about 0.31 seconds to load:

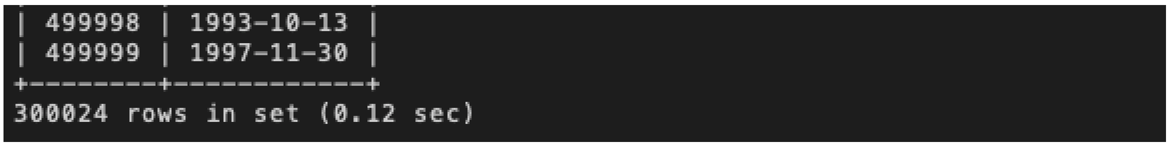
1. SELECT * FROM employee;



```
| 499998 | 1956-09-05 | Patricia       | Breugel          | M      | 1993-10-13 |
| 499999 | 1958-05-01 | Sachin         | Tsukuda          | M      | 1997-11-30 |
+--------+------------+----------------+------------------+--------+------------+
300024 rows in set (0.31 sec)
```

But if we only wanted to see the employee numbers and their hire dates (2 out of the 6 columns) we could easily do so with this query that takes 0.12 seconds to load:

1. SELECT employee_number, hire_date FROM employee;



```
| 499998 | 1993-10-13 |
| 499999 | 1997-11-30 |
+--------+------------+
300024 rows in set (0.12 sec)
```

Notice how the execution time of the query is much faster compared to the when we selected them all. This method can be helpful when dealing with large datasets that you only need select specific columns from.

Avoid Leading Wildcards

Leading wildcards, which are wildcards ("%abc") that find values that end with specific characters, result in full table scans, even with indexes in place.
If your query uses a leading wildcard and performs poorly, consider using a full-text index instead. This will improve the speed of your query while avoiding the need to search through every row.

Use the UNION ALL Clause

When using the OR operator with LIKE statements, a UNION ALL clause can improve the speed of your query, especially if the columns on both sides of the operator are indexed.
This improvement is due to the OR operator sometimes scanning the entire table and overlooking indexes, whereas the UNION ALL operator will apply them to the separate SELECT statements.

Summary & Highlights: -

- Database performance is measured by using key performance indicators, known as metrics, that enable DBAs to optimize organizations' databases.
- You should monitor at the infrastructure, platform, query, and user levels.
- A database diagnostic log file, also known as an error log or troubleshooting log, contains timestamped messages for various types of informational messages, events, warnings, and error details.
- Database optimization commands include OPTIMIZE TABLE in MySQL, VACUUM, and REINDEX in PostgreSQL, and RUNSTATS and REORG in Db2.
- Query execution plans show details of the steps used to access data when running query statements.

Week 4

Troubleshooting and Automation

Module 1: - Troubleshooting and Automation

1] Troubleshooting Common Issues

Troubleshooting basics: -

- Troubleshooting is a process of identifying and then solving a problem
- It begins by answering the following questions:
  - What are the symptoms?
  - Where is the problem happening?
  - When does the problem happen?
  - Under which conditions does the problem happen?
  - Is the problem reproducible?

Common problems: -

- **Poor Performance –**

  Poor performance can result in slow response to user queries or applications accessing the database.

- **Improper Configuration –**

  Improperly configured clients, servers, or databases can cause a wide range of problems, including poor performance, crashes, errors, or even database corruption.

- **Poor Connectivity –**

  Poor connectivity can cause poor performance, time outs, or a variety of errors when interacting with the database. Poor performance is usually caused by high latency for disk reads and writes, slow processing time by the server, or a poor connection between the server and client.

Common performance issues: -

Poor performance is often caused by:

- **Inadequate server hardware or configuration –**
  For example, the server could be running out of disk space, memory, or processing power. Increasing these resources on the database server is called vertical scaling. Adding additional database partitions and shards, to improve performance is called horizontal scaling.

- **Improper configuration –**
  A poorly configured database may be operational but still unable to meet demand. For example, it may need to allow more connections, or it may need changes to its buffering and indexing settings to keep up with queries and return results quickly.

- **Connectivity –**
  A slow or poor network connection, or limited bandwidth between the client and database can cause high latency and processing times.

- **Queries and Application Logic –**
  A poorly written database query or improper application logic (such as unnecessary locking of database objects) can also result in performance issues.

Common configuration issues: -

- **Client configuration –**
  An improperly or sub-optimally configured client, server, or database can cause any number of problems that can manifest in many ways. For example: A user with an incorrectly configured client or incorrect driver might be unable to connect to the database.

- **Sever configuration –**
  A poorly or incorrectly configured server can reduce performance, cause time outs or any number of possible errors.

- **Database configuration –**
  The database may need to be configured to allow more connections or other settings, like caching or indexing, and may need to be adjusted to correct problems or improve overall performance.

Client configuration issues: -

- Caused by incorrect login credentials, an incorrect host name or IP address, or even a corrupt or outdated connection driver.
- To fix these problems, check the client's driver configuration and verify the following:
    - The username and password specified in the connection settings are correct.
    - Be sure to verify that the client is also configured to use the correct type of authentication, such as Windows or SQL authentication, for example.
    - The connection configuration settings, such as IP address, host name, and server name, are correct.
    - The driver version for the database application is up to date and correct.

Server configuration issues: -

- Significantly impacts performance and operation.
- Some examples of things you might change or configure to improve performance or correct a problem include:
    - Add more physical RAM or increase the memory assigned to the server. (Out of memory)
    - Add more physical disk space or assign additional disk space to the server. (Out of disk space)
    - Consider upgrading the CPU or assigning more processing power to the server. (Inadequate processing power)
    - Consider defragmenting the hard disk. Fragmented data degrades overall performance. (Disk Fragmentation)
    - Sometimes configuring the storage system appropriately can alleviate performance issues, for example, placing frequently accessed tables on a faster disk. (Improper storage configuration)
    - Bugs in operating systems or in RDBMS software can result in errors and server crashes, so ensure you regularly apply software patches and security updates to guard against this.

Database configuration issues: -

- The configuration of the database is something you need to monitor and continually evaluate to ensure it meets demand.
- Some examples of configuration settings you might need to change or correct are:
    - Increase the number of allowed connections to the database to meet increasing demand. (Insufficient database connections)
    - Change database buffering to improve performance. (Insufficient buffering)
    - Change database indexing to improve performance. (Indexing)

Connectivity issues: -

- Poor connectivity between the client and the database server can cause a wide range of problems, including poor performance, error messages, or loss of function.

Examples of connectivity issues –

- The database server cannot be reached or is not running properly.
- The database instance on the server is not running properly.
- The client login credentials are incorrect, or missing security settings such as for SSL connections.
- The client configuration is incorrect.

Common connectivity solutions –

- Verify that the database server is running properly. The exact procedure depends on your configuration and environment. For example, you may need to physically check an on-premises server. Or you may need to verify that a virtual machine in a cloud service is running.
- Next, verify that the database instance on the server is running. This process varies depending on operating system and database. For example, on a Windows-based system you could use the Task Manager to verify that the instance is running. On a Db2 configuration, you could run db2cmd.exe and then issue commands in the command line.
- Verify that the database can be reached from the client. A common method is to use the PING command from the client to communicate with the server's IP address or host name.
- Finally, verify that the client connection driver is configured correctly. For example, make sure the user name and password for the connection are correct, and that other settings like IP address or host name, or security and encryption protocols are also correct.

Troubleshooting tools –

- **Monitoring Tools –**
  Performance monitoring, reports, and server and database logs help identify performance bottlenecks and determine the best way to correct them. Performance monitoring helps identify potential network, server, and database issues before they occur and helps determine where improvements can be made.

- **Dashboards and reports –**
  Dashboards can monitor databases in real time and provide an early warning system for problems before they affect users, in addition to tracking historical performance and other metrics.

- **Server and database logs –**
  Server and database logs help identify a problem and when it occurred.

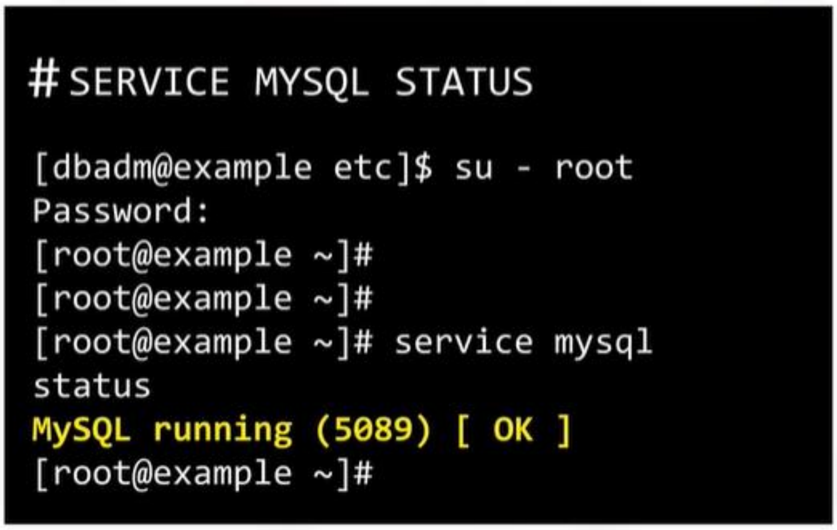
Summary: -

- The most common problems encountered with databases are caused by poor performance, improper configuration, or poor connectivity.
- Poor performance is typically caused by high latency for disk reads and writes, slow processing time by the server, or a poor connection between the client and server.
- Server configuration issues, such as inadequate hardware resources or misconfigured settings, can significantly impact performance.
- Some of the most common connectivity problems are not being able to connect to the database server, the database server or instance not running properly, and client login credentials being incorrect.
- Performance monitoring, reports, and server and database logs can help identify performance bottlenecks and determine the best way to correct them.

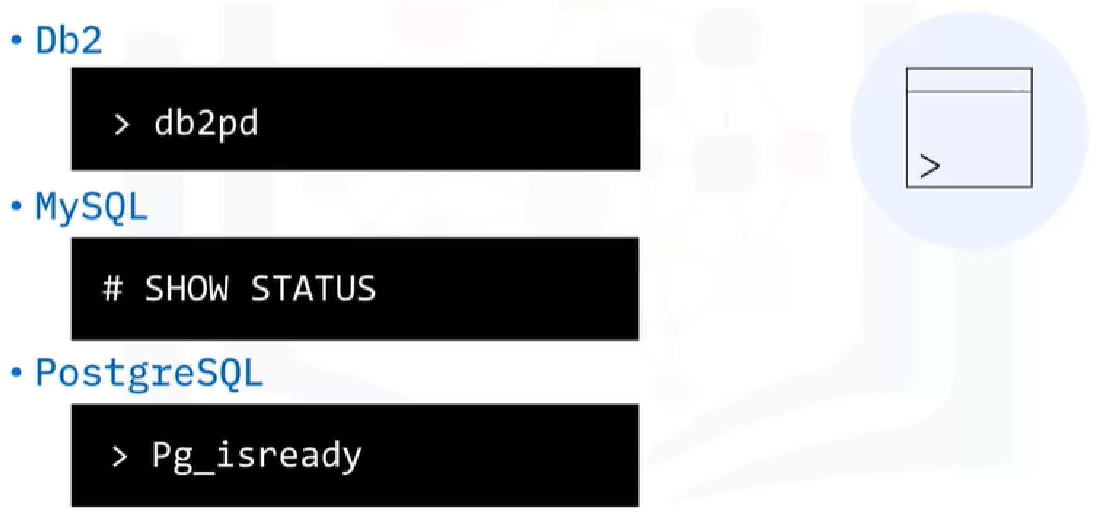
2] *Using Status Variables, Error Codes, and Documentation*

Getting server status: -

- Databases have utilities to check health and operational status
  - Command line
  - GUI



```
#SERVICE MYSQL STATUS

[dbadm@example etc]$ su - root
Password:
[root@example ~]#
[root@example ~]#
[root@example ~]# service mysql
status
MySQL running (5089) [ OK ]
[root@example ~]#
```



- Db2

```
> db2pd
```

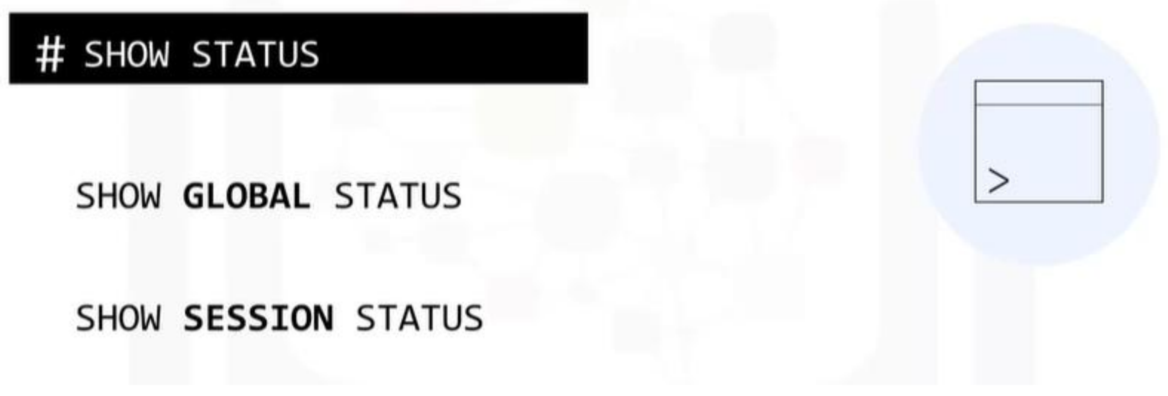
- MySQL

```
# SHOW STATUS
```

- PostgreSQL

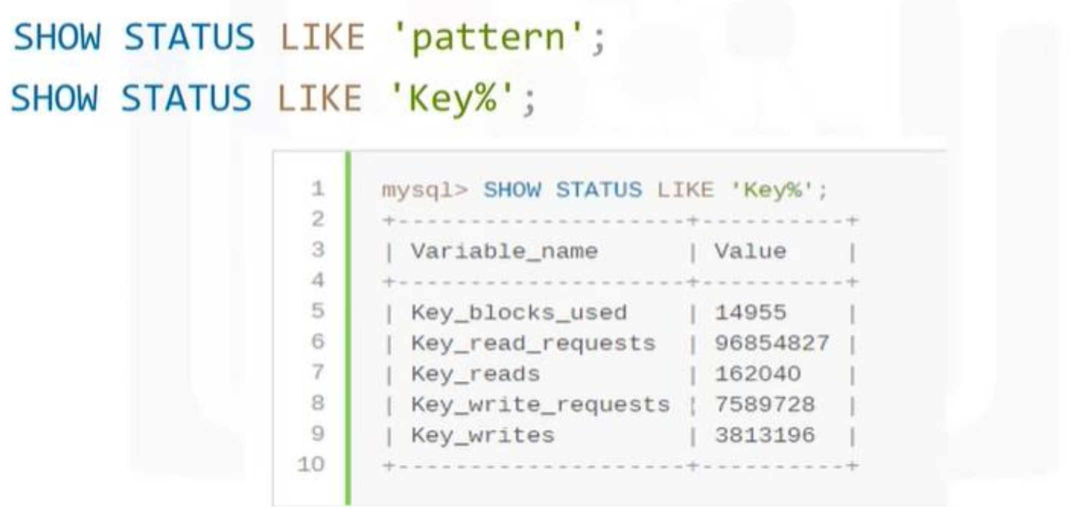
```
> Pg_isready
```

Using status variables: -



```
# SHOW STATUS
```

```
SHOW GLOBAL STATUS

SHOW SESSION STATUS
```



```
SHOW STATUS LIKE 'pattern';
SHOW STATUS LIKE 'Key%';
```

```
mysql> SHOW STATUS LIKE 'Key%';
+--------------------+----------+
| Variable_name      | Value    |
+--------------------+----------+
| Key_blocks_used    | 14955    |
| Key_read_requests  | 96854827 |
| Key_reads          | 162040   |
| Key_write_requests | 7589728  |
| Key_writes         | 3813196  |
+--------------------+----------+
```

Using GUIs to get server status: -

Windows SQL Server GUI Tools –

- Activity Monitor
- System Monitor

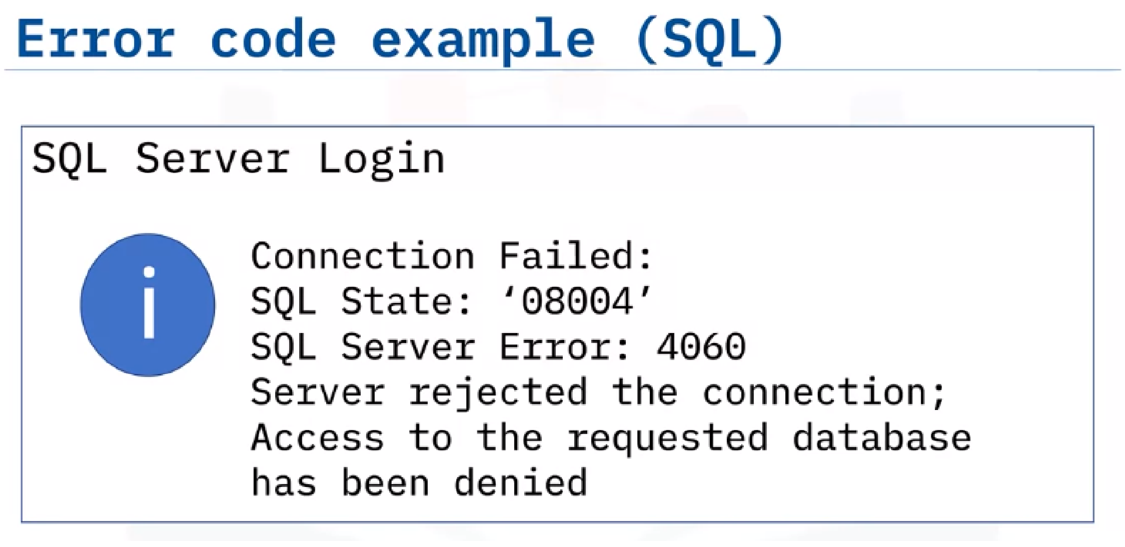
Reviewing logs: -

Server and OS Logs – which log general server activity, connectivity, and other aspects of the server or servers running the database

Database Error Logs – which log information and errors specific to the database being operated, such as a MYSQL or PostgreSQL system.

Log file examples (SQL): -

- Log files are important tools for discovering when an error occurred and the description of the error.
- Some of most accessed logs are:
- **Error log** – The Error Log file, which is created every time SQL server is started
- **Event log** – The Event Log file, which shows informational and error events
- **Trace log** – The Default Trace Log, which is an optional log file that tracks all database configuration changes.

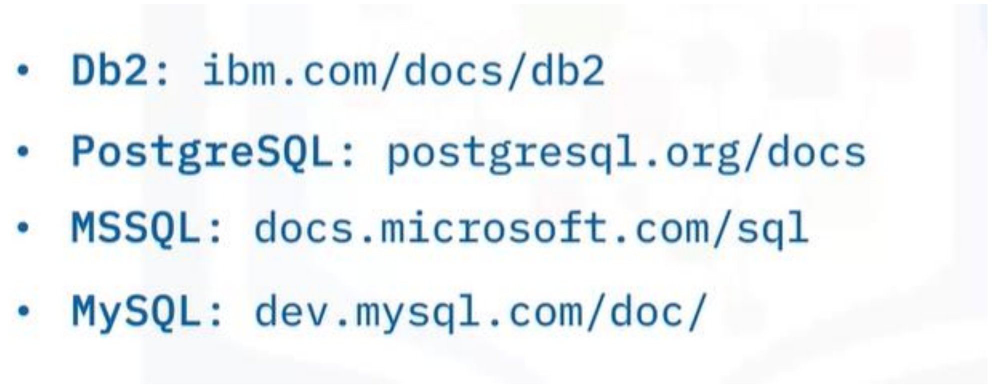


Error code example (SQL)

```
SQL Server Login

Connection Failed:
SQL State: '08004'
SQL Server Error: 4060
Server rejected the connection;
Access to the requested database
has been denied
```

Decoding errors: -

Where to find documentation and troubleshooting help:



- Db2: ibm.com/docs/db2
- PostgreSQL: postgresql.org/docs
- MSSQL: docs.microsoft.com/sql
- MySQL: dev.mysql.com/doc/

Summary: -

- Databases have utilities to check health and operational status.
- The available commands and their syntax vary with the flavor of database you are using.
- Databases use many status variables to provide information about their operation, and status variables can be either GLOBAL or SESSION based.
- Databases may also have a graphical user interface with dashboards and reports for monitoring database status and information in real time.
- There are several log files you can use to help identify when and where an error occurred.
- There are many documentation and help sites available on the Internet with error code tables that can help you decode and correct errors.

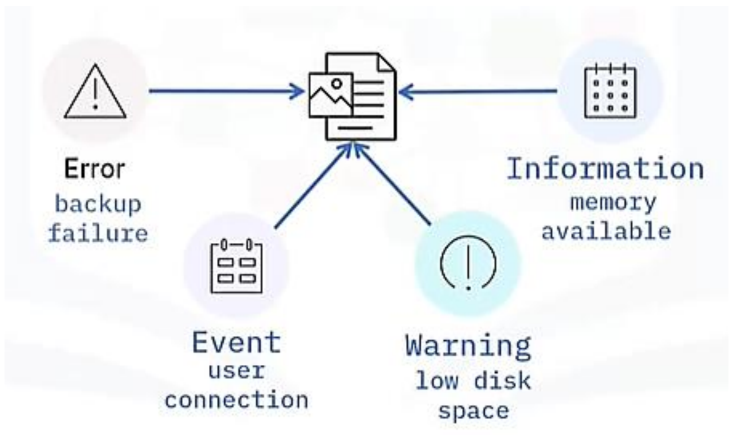
3] Using Logs for Troubleshooting

Troubleshooting with logs: -

- Systematic process used to locate faults or errors and provide detail on how to correct hardware and software issues.
- Approaching troubleshooting logically and methodically is essential to successful resolution.
- Troubleshooting is the main reason to create diagnostic logs.

What are diagnostic logs?

Diagnostic logs provide chronological records of events and errors in a particular component or application and can be used for troubleshooting problems.



Types of diagnostic logs: -

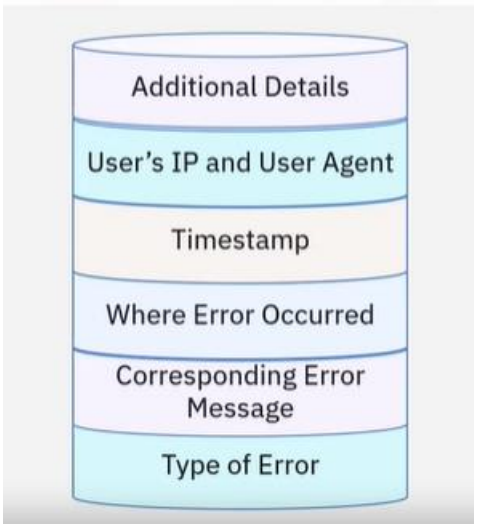
- Device logs
- Server logs
- Operating system logs
- Network logs
- Database logs
- Application logs

While the systems administrator may monitor the other logs, the DBA needs to be intimately familiar with the database diagnostic logs, and review them frequently. These database diagnostic logs are generally separate from the database transaction logs or query logs

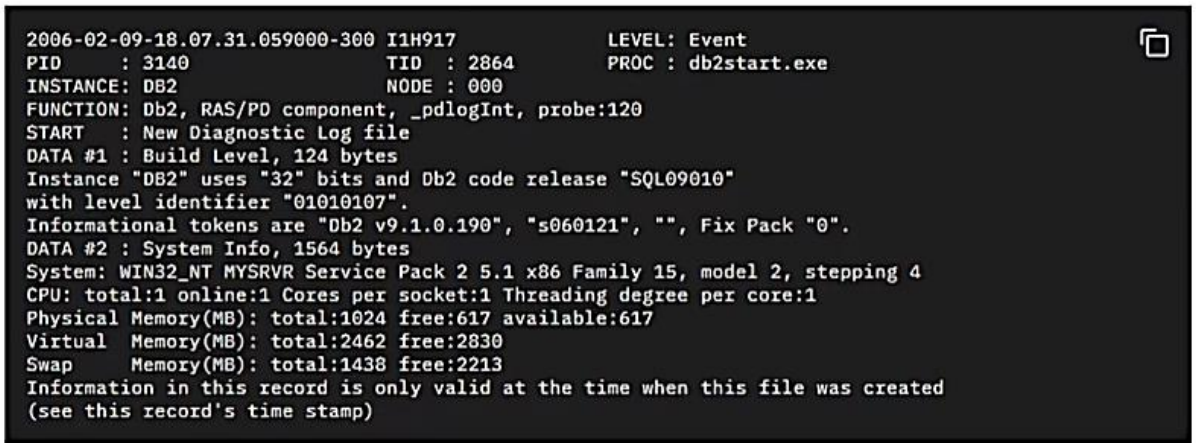
Working with log files: -

- May be possible to configure location of log
- Many log files are in plain text format
- Some logs may require special tools to view or filter contents

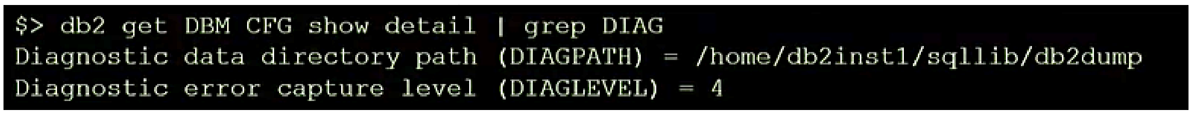
Diagnostic log contents: -



Db2 diagnostic log: -



```
2006-02-09-18.07.31.059000-300 I1H917           LEVEL: Event
PID     : 3140                 TID  : 2864      PROC : db2start.exe
INSTANCE: DB2                  NODE : 000
FUNCTION: Db2, RAS/PD component, _pdlogInt, probe:120
START   : New Diagnostic Log file
DATA #1 : Build Level, 124 bytes
Instance "DB2" uses "32" bits and Db2 code release "SQL09010"
with level identifier "01010107".
Informational tokens are "Db2 v9.1.0.190", "s060121", "", Fix Pack "0".
DATA #2 : System Info, 1564 bytes
System: WIN32_NT MYSRVR Service Pack 2 5.1 x86 Family 15, model 2, stepping 4
CPU: total:1 online:1 Cores per socket:1 Threading degree per core:1
Physical Memory(MB): total:1024 free:617 available:617
Virtual  Memory(MB): total:2462 free:2830
Swap     Memory(MB): total:1438 free:2213
Information in this record is only valid at the time when this file was created
(see this record's time stamp)
```



```
$> db2 get DBM CFG show detail | grep DIAG
Diagnostic data directory path (DIAGPATH) = /home/db2inst1/sqllib/db2dump
Diagnostic error capture level (DIAGLEVEL) = 4
```

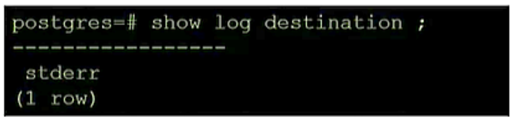
PostgreSQL server logs: -

log_destination syslog (Linux / Unix)

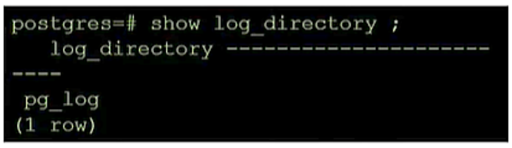
- event log (Windows)
- csvlog
- stderr (default)

log_directory

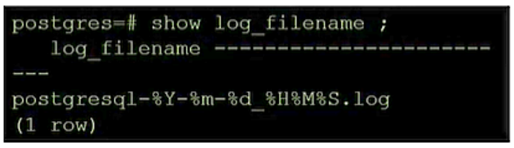
log_filename



```
postgres=# show log destination ;
-----------------
 stderr
(1 row)
```



```
postgres=# show log_directory ;
   log_directory ---------------------
----
 pg_log
(1 row)
```



```
postgres=# show log_filename ;
   log_filename ----------------------
---
postgresql-%Y-%m-%d_%H%M%S.log
(1 row)
```

MySQL server logs: -

MySQL has several logs for troubleshooting:

- **General Query log**

  Connections and queries received from clients

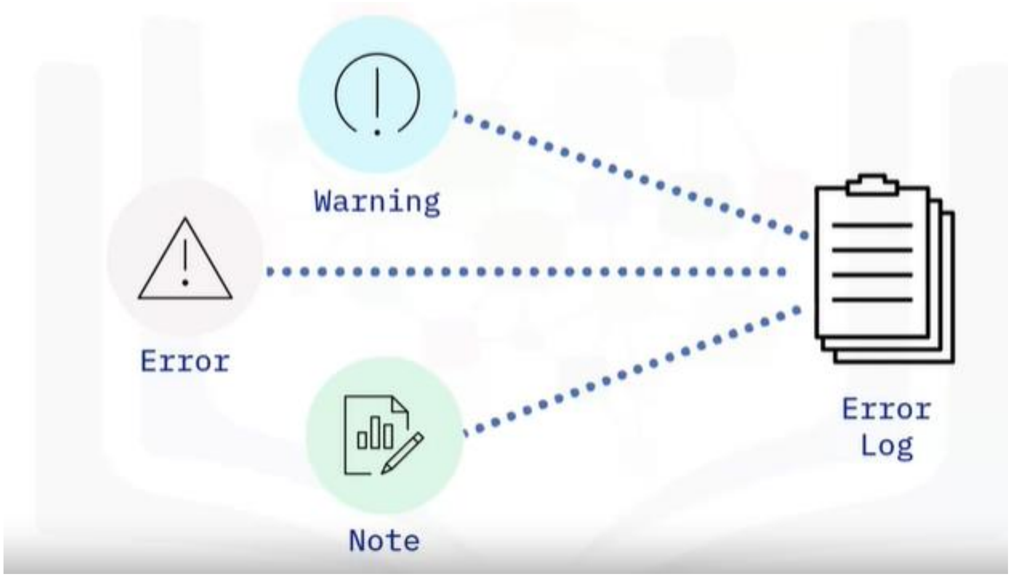
- **Slow Query log**

  Queries that take longer than log_query_time

- **Error log**

  Diagnostic messages for mysqld

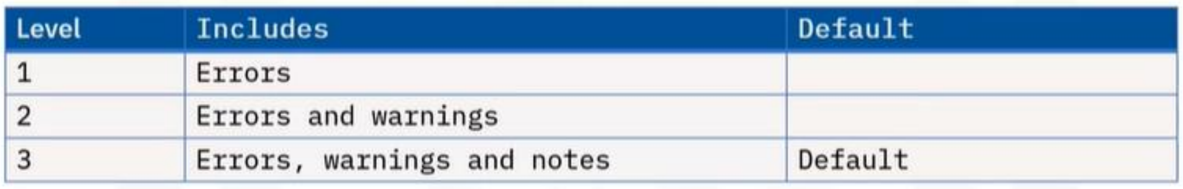
MySQL error logs: -



- Can be configured to write to:
  - **event log** – enabled by default in Windows
  - **syslog** – used on Linux or Unix
  - **stderr** – enabled by default on Linux or Unix
  - specific file
- Specify using log_error

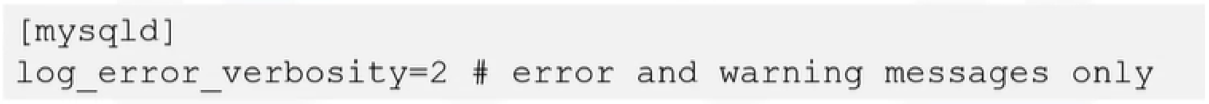
```
[mysqld]
log_error = /var/log/mysql/error.log
```

Configure how much is stored in the log:



Level	Includes	Default
1	Errors	
2	Errors and warnings	
3	Errors, warnings and notes	Default

- Use log_err_verbosity:



```
[mysqld]
log_error_verbosity=2 # error and warning messages only
```

Summary: -

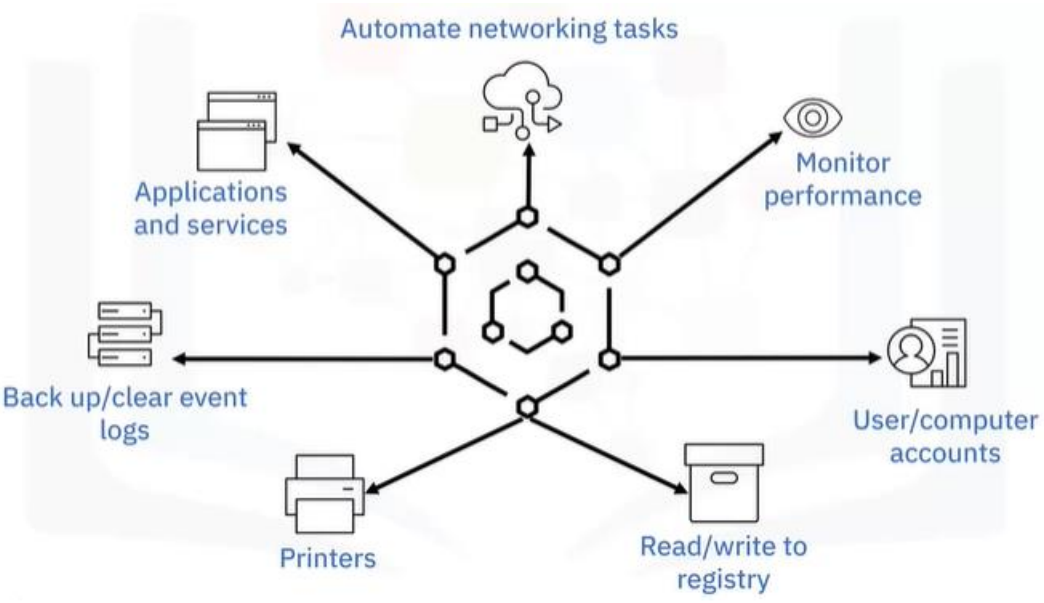
- A Database Administrator needs to be familiar with many types of logs, especially the database diagnostic logs.
- A database diagnostic log file, also known as an error log or troubleshooting log, contains timestamped messages for various types of informational, events, warnings, and error details.
- Most diagnostic logs are text-based and can be viewed on the console or in a text editor, but some may require special tools or viewers to view and filter their contents.
- Most databases provide settings for enabling and configuring diagnostic and error logging, such as for setting log filenames and locations and verbosity of messages.
- Some databases, like MySQL, offer more than one diagnostic log, for example, the error log and slow query log.

4] Overview of Automating Database Tasks

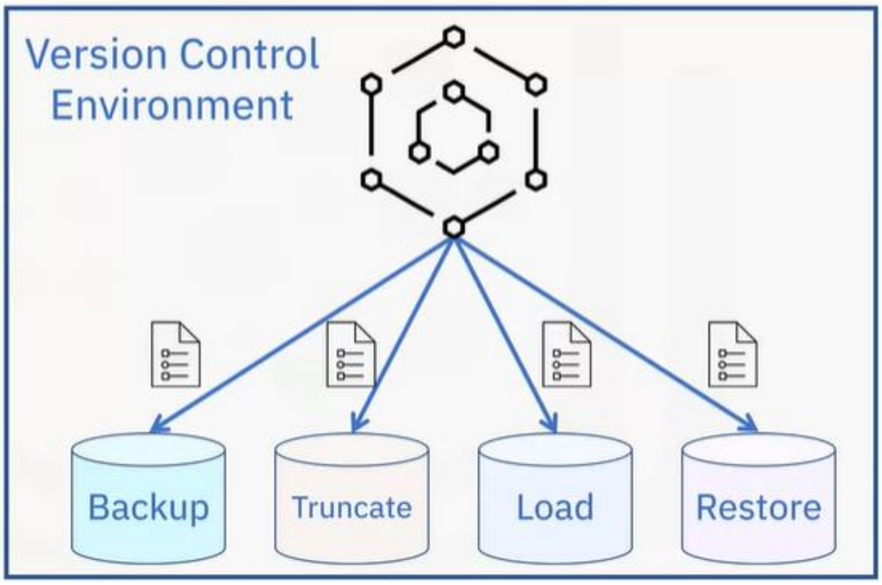
Identifying automated tasks: -

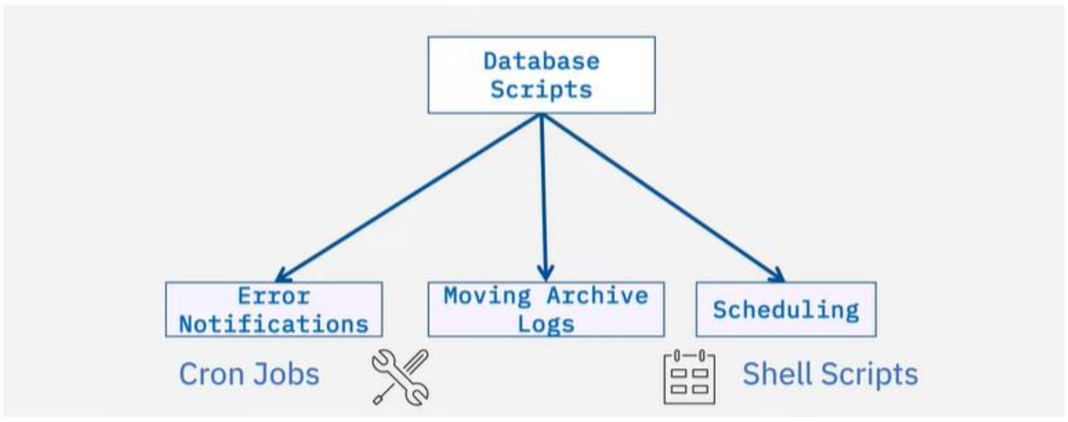
- Leverages unattended processes and self-updating procedures
- Fewer deployment errors/higher reliability and speed
- Enables staff to focus on more important tasks and coding
- Ideal tasks to automate are time-consuming and repetitive ones

Using script to automate tasks: -



Keep database in sync with code

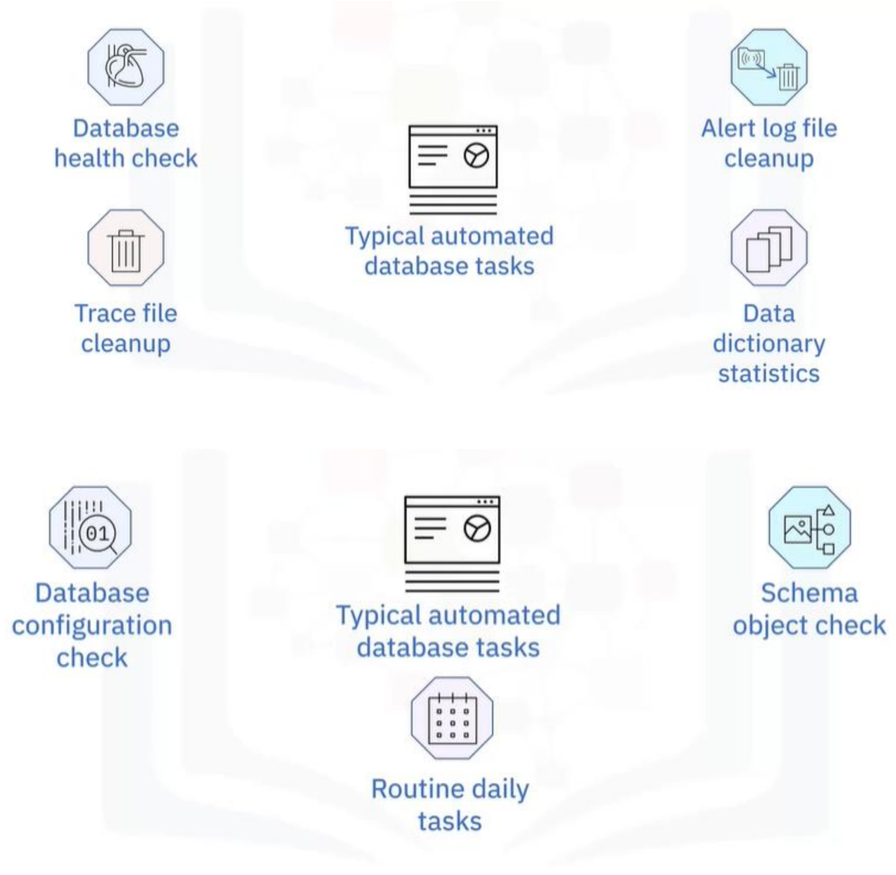




Advantages of automating tasks: -

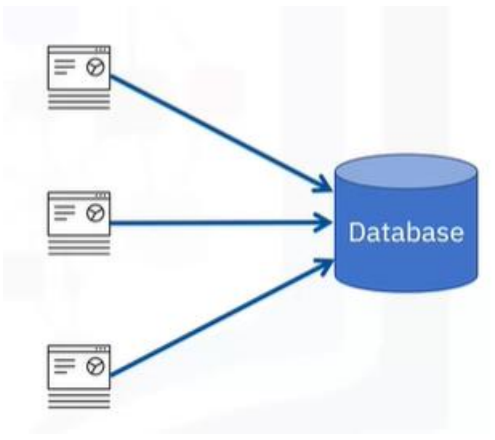
- Increase throughput and productivity
- Improve quality or increase predictability of quality
- Improve consistency of process or product
- Increase consistency of outputs or results
- Free up staff to perform other activities
- Provide higher level jobs in automated processes

Automated database task examples:


Database health check
Alert log file cleanup
Typical automated database tasks
Trace file cleanup
Data dictionary statistics
Database configuration check
Typical automated database tasks
Schema object check
Routine daily tasks

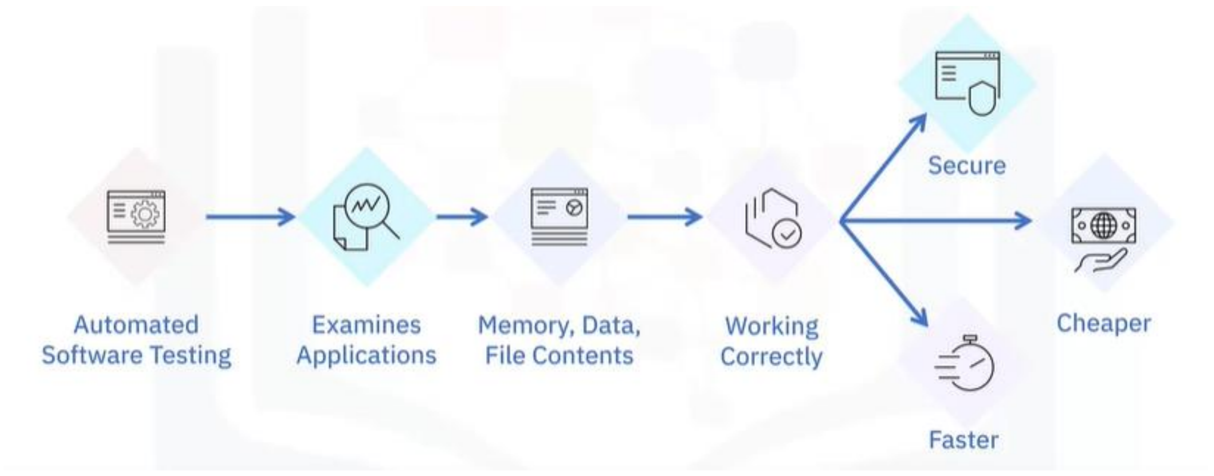
Automating database testing:

- Information in database is correct and running properly within controlled testing environment
- Schema
- Tables
- Triggers
- Prevent data loss
- Saves aborted transaction data
- Prohibits unauthorized access
- Checks data integrity and consistency



Automating database testing benefits:

- Automated software testing can look inside an application and see memory contents, data tables, file contents, and internal program states to determine if the product is behaving as expected.
- Once created, automated tests can be run repeatedly at no additional cost and are much faster and more secure than manual tests.
- Automated tests can increase the depth and scope of tests to help improve quality and security of automation database testing.



<u>**Summary: -**</u>

- Database automation is the use of unattended processes and self-updating procedures
- For administrative tasks in a database, scripting languages easily automate processes at the application level,
- Some automation processes include backing up, truncating, and restoring databases

- Some of the advantages of automating database tasks include increased throughput or productivity, improved quality or increased predictability of quality and better consistency of processes or product.
- Automated database testing has many benefits including being more secure, cheaper, and faster

5] Automating Reports and Alerts

Reports: -

- Health status of databases
- Address issues/problems
- Keep track of trends over time
- Predict future needs
- Regular schedule: daily, weekly, or monthly

Automated Reports: -

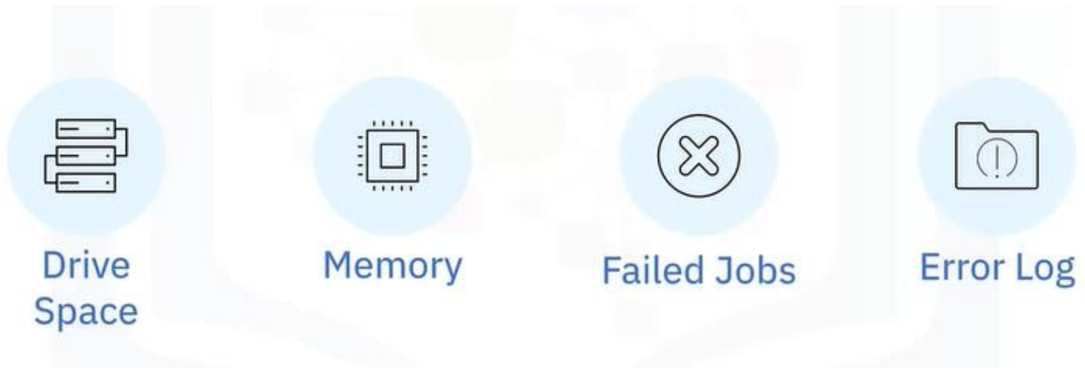
In larger companies, an entire department may manage automated reports, while in smaller companies, a few DBAs, or in some cases only one DBA, might perform that task.

Notifications: -

- Bring an event to the DBA's attention
- Raise awareness of specific events
- Display on dashboards or sent through email

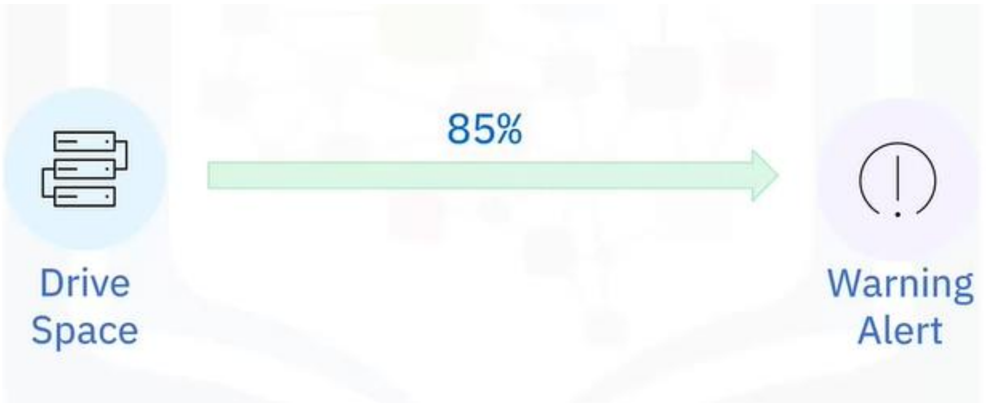
Alerts: -

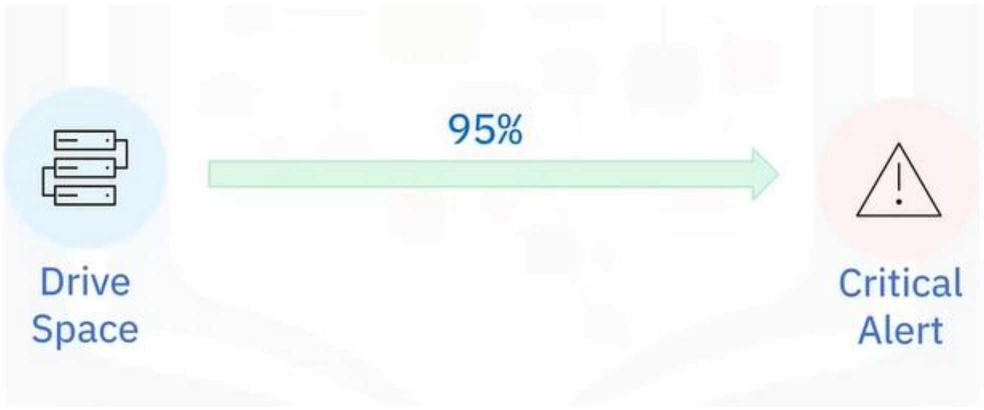
Alerts bring urgent issues to your attention.



Configurable severity thresholds

Warning Threshold
Critical Threshold





Automated reports: -

- Configure content and frequency
- Use a sample report
- Configure your own

Automation: -

- Configure through:
    - Graphical interface

    - Command line
    - Script
- Varies by RDBMS

Summary: -

- Reports give a regular overview of database health.
- Notifications give a forewarning of a situation that could become troublesome if not addressed.
- Alerts bring awareness to an issue that needs immediate attention.
- DBAs automate reports, notifications, and alerts to suit the needs of their environment.

Summary & Highlights: -

- Performance monitoring, dashboards and reports, and server/database logs help identify bottlenecks.
- Database automation is the use of unattended processes and self-updating procedures for basic database tasks.
- Some automation processes include backing up, truncating, and restoring databases.
- Reports give a regular overview of database health, notifications give a forewarning of a situation that could become troublesome if not addressed, and alerts bring awareness to an issue that needs immediate attention.