



Week 1 - Introduction to Database Management

▼ Tipo	Anotaciones
≡ Posición	
⋮ Etiquetas	
↗ Bibliografía	
🕒 Fecha de creación	@June 9, 2022 6:29 PM
≡ Property	

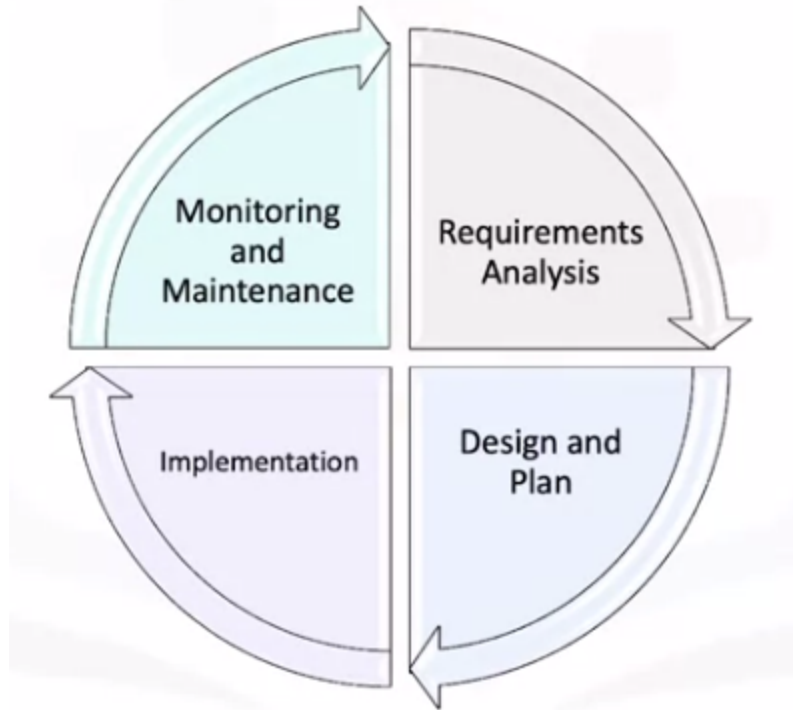
▼ Overview of Database Management Tasks

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
- Monitorindg database activity

Database Management Lifecycle

The database life cycle

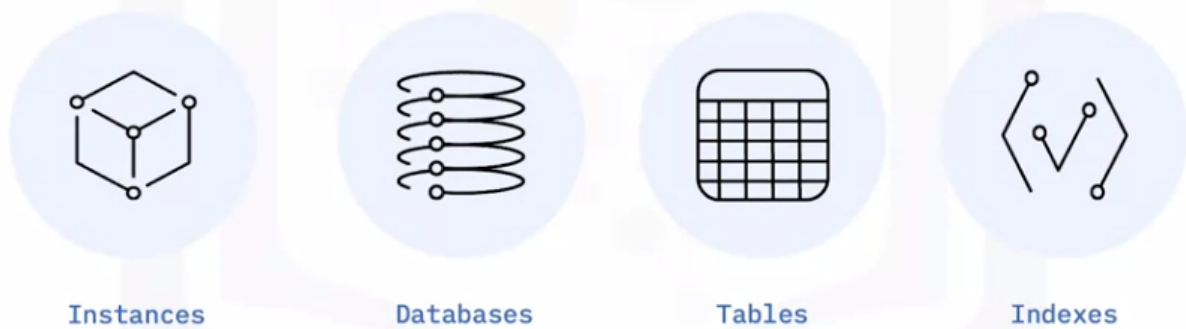


Requirements Analysis

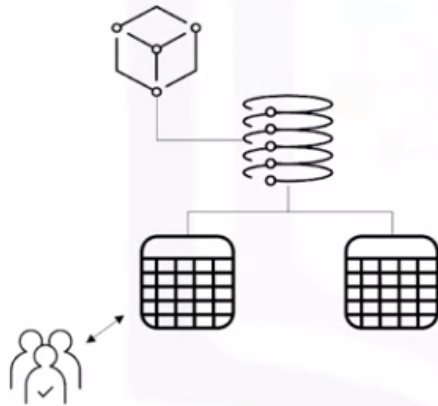
- Understand purpose and scope of the database
- Work with stakeholders
 - Analyze need for database
 - Clarify goals for database
 - Identify users

Design and Plan

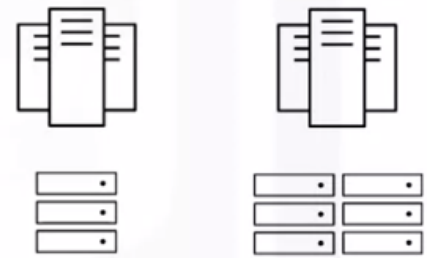
Work with database objects



Database Modelling



Capacity Planning



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

- Create and configure database objects:
 - instances
 - databases
 - tables
 - views
 - indexes
- Grant access for database users, groups
- Automate repeating tasks
- Deploy data movement

Monitoring and Maintenance

- Monitor system for performance issues
- Review reports
- Apply upgrades and patches to RDBMSes
- Automate deployments and routine tasks
- Troubleshoot issues
- Security and Compliance
 - 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

Data Security, Ethical and Compliance Considerations

Fundamental Ethics

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.



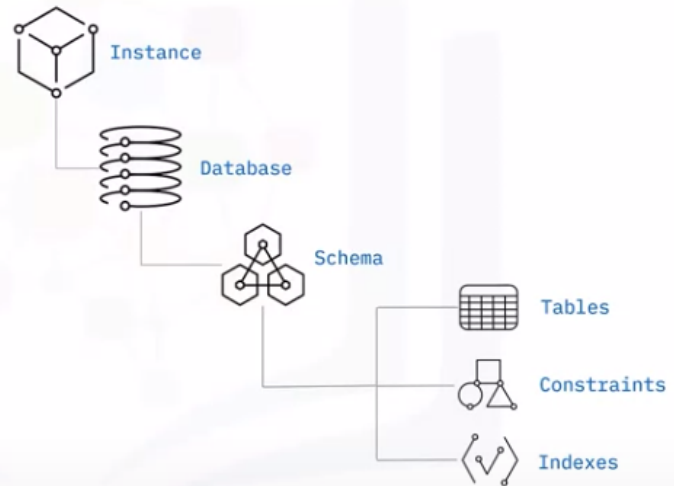
Ungraded External Tool: Obtain an IBM Cloud Feature Code
Hands-on Lab: Create Db2 service instance

▼ Server Objects and Hierarchy

Database Objects

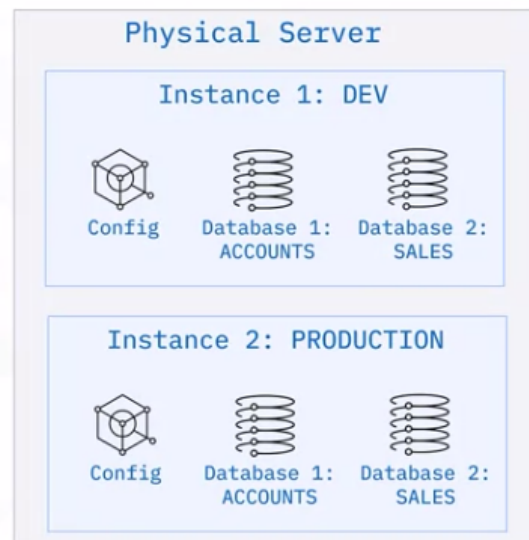
Database Hierarchy

- Instance
- Database
- Schema
- Database objects
 - Tables
 - Constraints
 - Indexes



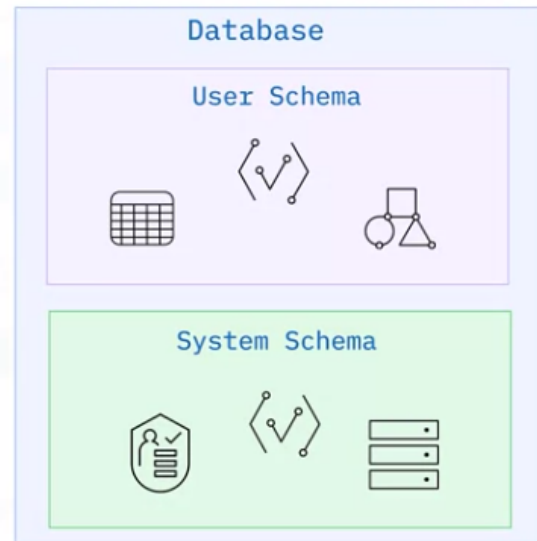
Instance

- Is a logical boundary for databases, objects, and configuration
- Provides unique database server environment
- Allows isolation between databases



Schema

- Organize database objects
- Default schema is the user schema
- System schemas contain database configuration information

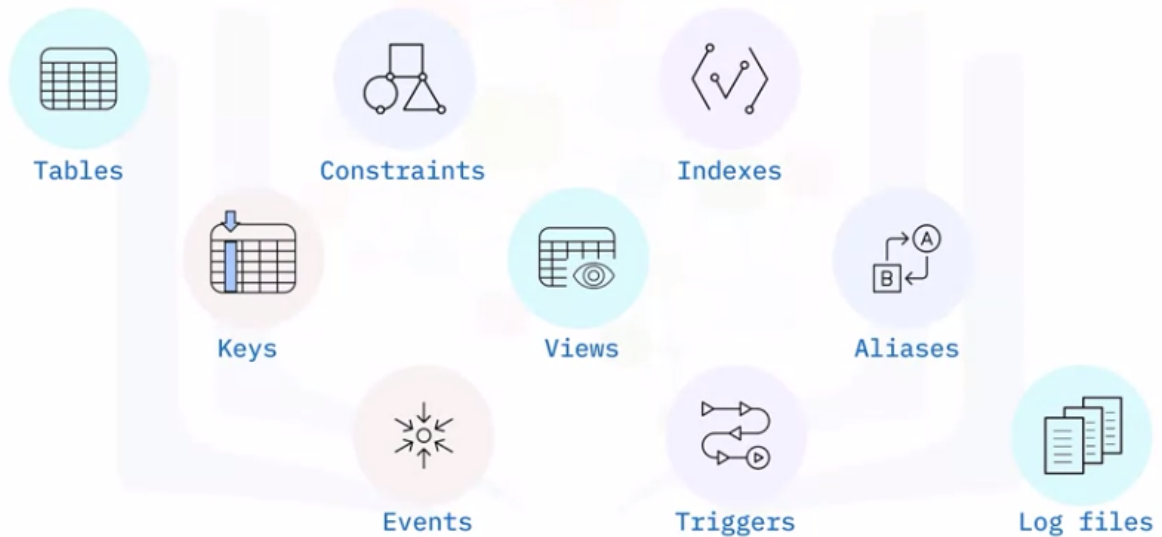


A schema can contain tables, indexes, constraints, and other objects. When you create a database object, you can assign it to a schema. In most RDBMSes, the default schema is the user schema for the currently logged-on user. Many RDBMSes use a specialized schema to hold configuration information and 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

You can create and manage database objects through graphical database management tools, scripting, or accessing the database through an API. 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.

System Objects and Database Configuration

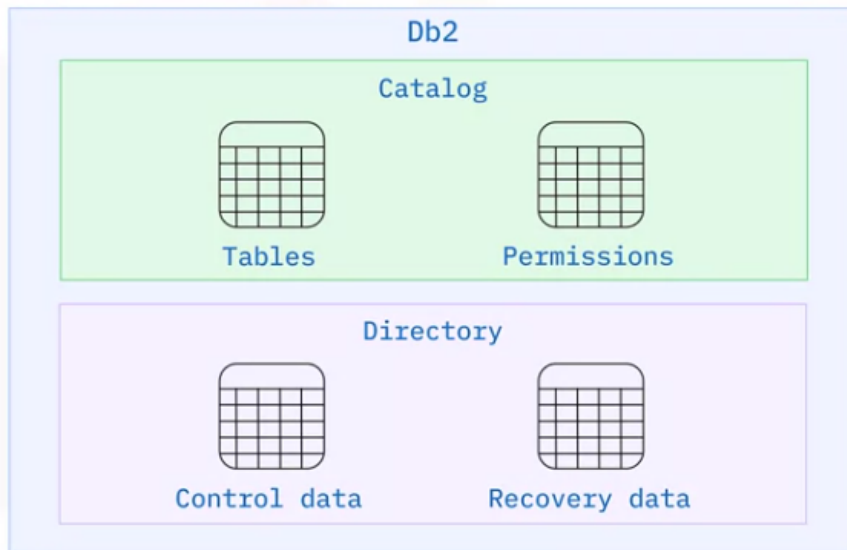
RDBMSes store information about their databases, known as metadata, in special databases, schemas, or catalogs. They store specific types of information about the database, such as the name of a database or table, the data type of a column, or access privileges, known as metadata. Other terms used for this information store are data dictionary and system catalog.

- Store database metadata in special objects
- Known as system database, system schema, catalog, or dictionary

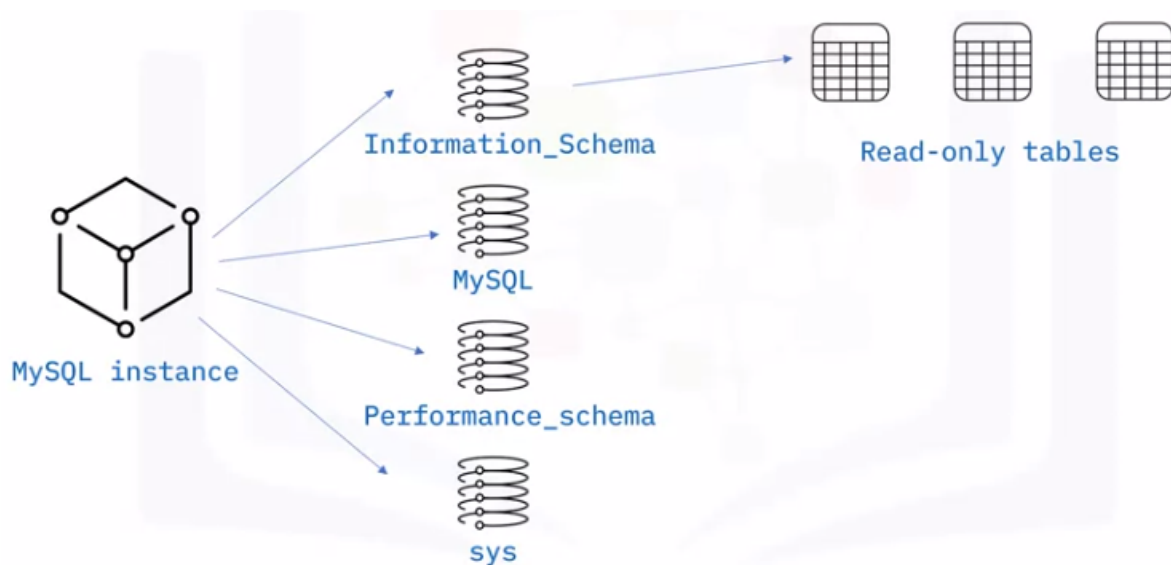


The RDBMS controls and updates the system objects, but you can query the metadata tables to discover information about the objects in the database.

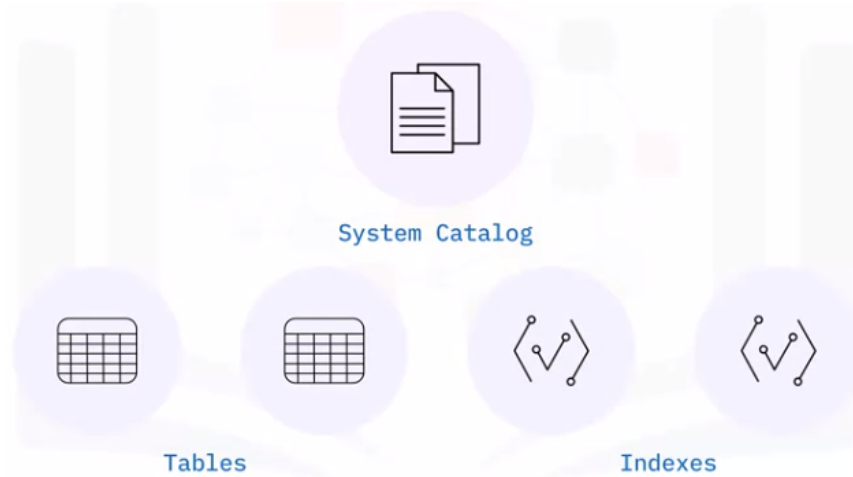
Db2 uses the catalog and the directory. The catalog consists of tables of data about everything defined to the Db2 system. When a user creates, alters, or drops any object, Db2 inserts, updates, or deletes rows of the catalog that describe that object and how it relates to other objects. The directory 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 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.

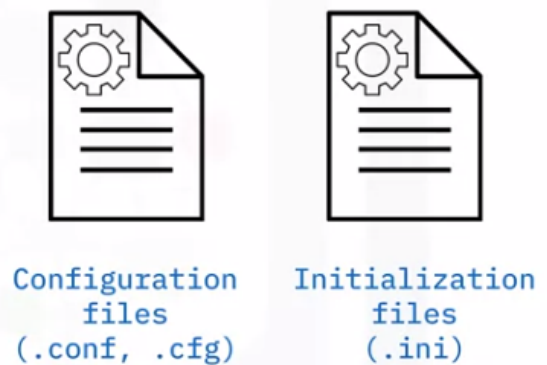


PostgreSQL uses the system catalog, which is a schema with tables and views that contain metadata about all the other objects inside the database, and more. With it, you can discover when various operations happen, how tables or indexes are accessed, and whether the database system is reading information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



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.

- Set configuration parameters during installation
- Save into files
 - Configuration
 - Initialization



Again, different RDBMSes use different files, file locations, and settings. Still, they all serve the same purpose: to provide the initial configuration information for the database as it starts up and operates. Configuration information can include general settings like the location of data and log files, and the port the server listens on for requests. Or configuration information can be more focused, like settings that affect performance, including memory allocation, a connection timeout, and the maximum packet size.

To store configuration information: **Db2** uses the SQLDBCONF file. **MySQL** uses my.ini files on Windows-based systems and my.cnf files on Linux-based systems. **PostgreSQL** uses the postgresql.conf file.

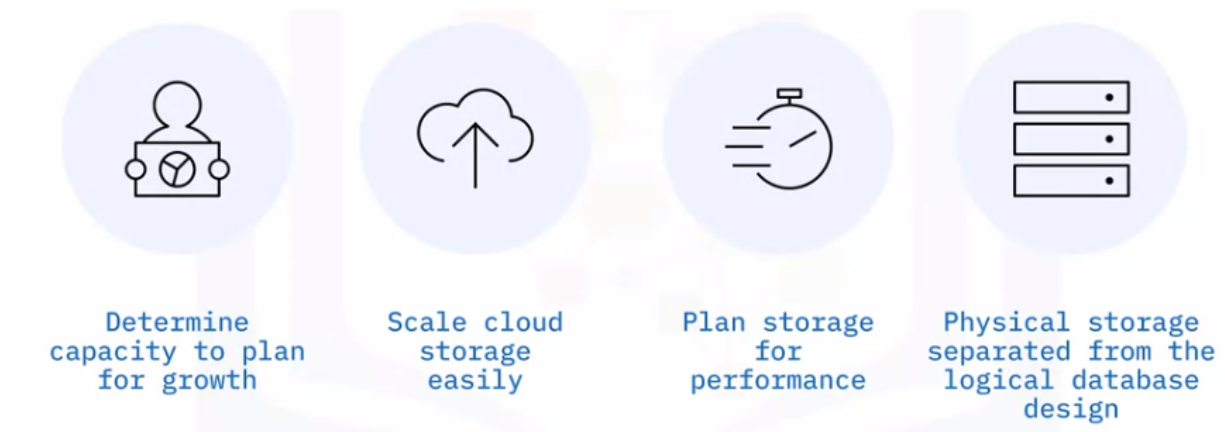
Modify parameters after the database has started

On-premises	Cloud-based
<ol style="list-style-type: none">1. Stop the database service2. Modify the configuration file3. Restart the database service	<ol style="list-style-type: none">1. Use graphical tools or APIs to modify the setting2. Database scales dynamically

One advantage of Cloud-based systems is that you 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.

Database Storage

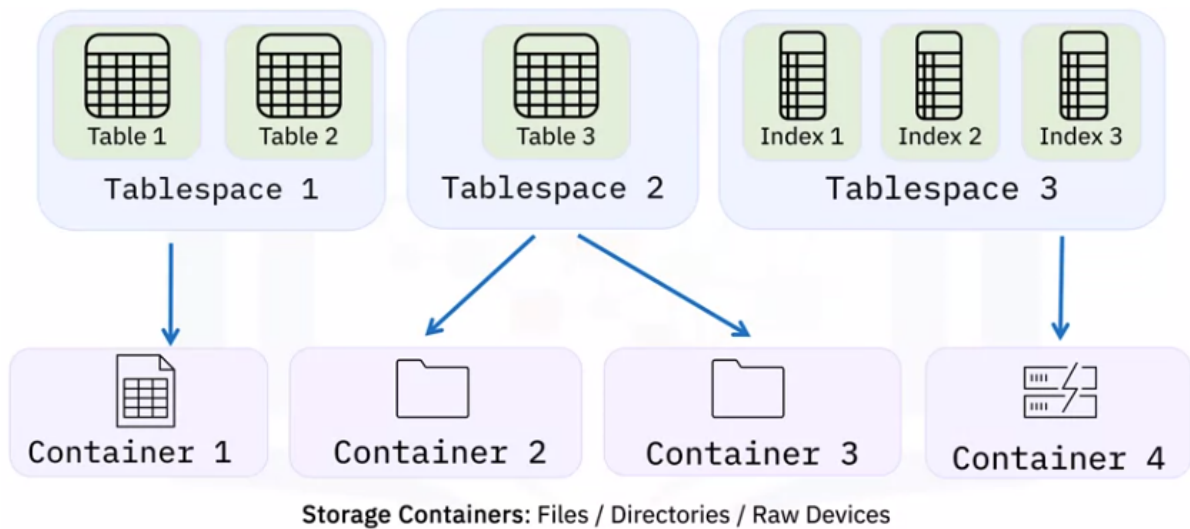
Physical and Logical Storage



You must determine the capacity required for the database and plan for 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. RDBMSes separate the physical storage of database files on disk from the logical design of the database, allowing more flexibility in managing the data files. You can manage the data through a logical object without being concerned about the nature of the physical storage. 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. 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

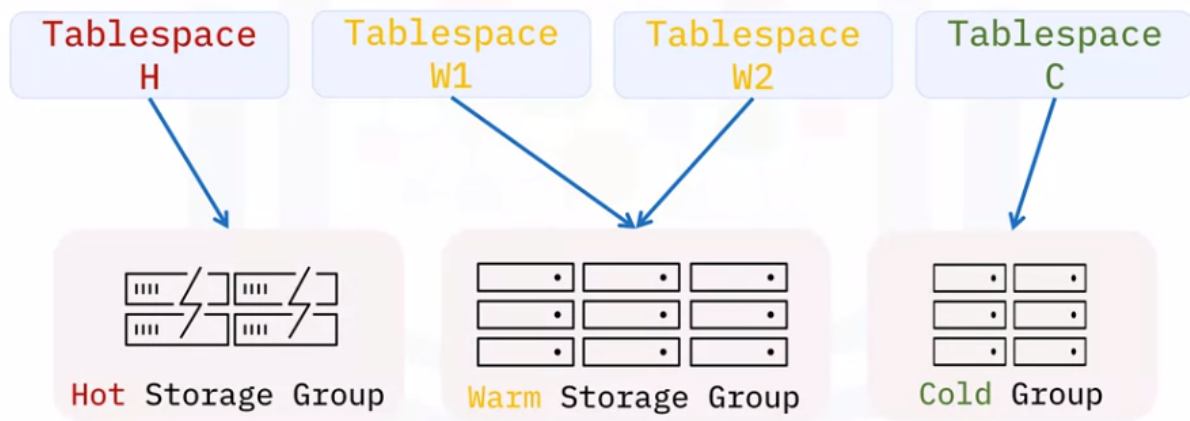
Tablespaces separate logical database storage
separate from physical storage



- **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:** 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



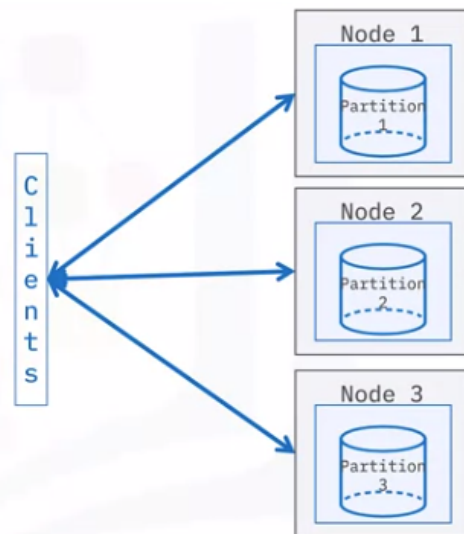
A storage group is a grouping of storage paths or containers based on similar performance characteristics. This allows you to perform Multi-Temperature Data Management more easily. In this context, temperature refers to the frequency of data access. Hot data is accessed very frequently, Warm data is accessed somewhat frequently, and Cold data is accessed infrequently. By using storage groups, you can organize your data and storage based on temperature. In 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. Using storage groups helps with optimizing performance for frequently accessed data and reducing costs for storing infrequently accessed data.

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.

- Data is managed across multiple partitions
- Split tables that contain very large quantities of data
- Partitions hold a subset of the data
- Common in data warehousing



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

<u>Aa</u> Engines	Description
-------------------	-------------

 Engines	 Description
<u>InnoDB</u>	<ul style="list-style-type: none"> • 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.
<u>MyISAM</u>	<ul style="list-style-type: none"> • Manages non-transactional tables. • Provides high-speed storage and retrieval. • Supports full-text searching.
<u>MEMORY</u>	<ul style="list-style-type: none"> • 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.
<u>MERGE</u>	<ul style="list-style-type: none"> • Treats groups of similar MyISAM tables as a single table. • Handles non-transactional tables.
<u>EXAMPLE</u>	<ul style="list-style-type: none"> • Allows developers to practice creating a new storage engine. • Allows developers to create tables. • Does not store or fetch data.
<u>ARCHIVE</u>	<ul style="list-style-type: none"> • Stores a large amount of data. • Does not support indexes.
<u>CSV</u>	<ul style="list-style-type: none"> • Stores data in Comma Separated Value format in a text file.
<u>BLACKHOLE</u>	<ul style="list-style-type: none"> • Accepts data to store but always returns empty.
<u>FEDERATED</u>	<ul style="list-style-type: none"> • Stores data in a remote database.

Commands for working with Storage 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.

```
SHOW ENGINES;
```

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.
```

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.
```

You can convert a table from one storage engine to another using an `ALTER TABLE` statement. For example, the following statement set the storage engine 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.



Hands-on Lab: Db2 System Tables

Ungraded External Tool: Hands-on Lab: MySQL Configuration, Storage Engines, and System Tables

Ungraded External Tool: Hands-on Lab: PostgreSQL Instance Configuration and System Catalog