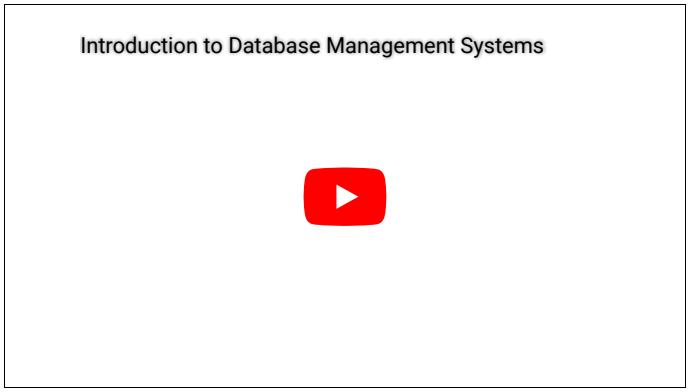
Introduction: Database and Database Management System

CONTENTS

Refer to

- What is a Database? Definition, Meaning, Types with Example,
- Database Management System Tutorial,
- What is Database? What is SQL?
- Relational vs. Non-Relational Databases

What is Database Management System (DBMS)?



Introduction to Database Management Systems

Data: Data can be defined as an assemblage of unprocessed facts, numerical figures, and statistical metrics, all of which can be manipulated and analyzed to derive meaningful and beneficial insights. It presents itself in diverse modalities including text, visuals, auditory signals, and video content.

Database refers to a systematic and organized assembly of interconnected data, while data itself constitutes a compilation of quantifiable facts and statistics, which upon analysis, yields meaningful information.

- **Databases support storage and manipulation of data.**
- **Databases make data management easy.**

Examples:

- **Online Retail Store:** Consider the use of databases in an online retail platform like Amazon.

Top

These databases store extensive data on products, including descriptions, prices, stock levels, and supplier information. Additionally, they manage customer data, such as order history, shipping addresses, and payment details. The database is essential for processing transactions, recommending products based on past purchases, and maintaining an efficient supply chain.

- **Hospital Management System:** Hospitals extensively use databases for managing patient information, staff records, appointment schedules, and medical histories. For example, a hospital's database might store information about patients' diagnoses, treatments, medication records, and upcoming appointments. These databases are vital for doctors to access patient history, for the administration to manage hospital resources, and for facilitating billing and insurance processes.

A **Database Management System (DBMS)** constitutes **an integrated suite of software applications designed to facilitate users in interacting with databases.** This interaction includes not only accessing data but also manipulating it according to specific needs. Furthermore, the DBMS offers the capability for comprehensive data reporting and visualization, enabling efficient and intuitive representation of information.

More videos:

Learn What is Database | Types of Database | DBMS

What is data management - and why do you need it in inter...

In the early days, database applications were built directly on top of file systems. A **Flat File Management System** refers to a simple way of storing data in which each database is contained in a single table or file. In this system, data is typically stored in **plain text files**, where each line of the file represents a single record and fields are separated by delimiters, such as commas or tabs. This format is often used in simple applications or for small amounts of data that don't require complex relationships or queries.

- Data redundancy and inconsistency: data is stored in multiple file formats resulting induplication of information in different files
- Difficulty in accessing data: Need to write a new program to carry out each new task
- Data isolation: Multiple files and formats
- Integrity problems:
  - Integrity constraints (e.g., account balance > 0) become “buried” in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones
- Atomicity of updates:
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not

Top

happen at all
- Concurrent access by multiple users
- Security problems: Hard to provide user access to some, but not all, data

Database systems offer solutions to all the above problems

DBMS vs. Flat File

DBMS	Flat File Management System
Multi-user access	It does not support multi-user access
Design to fulfill the need for small and large businesses	It is only limited to smaller DBMS system.
Remove redundancy and Integrity	Redundancy and Integrity issues
Expensive. But in the long term Total Cost of Ownership is cheap	It's cheaper
Easy to implement complicated transactions	No support for complicated transactions

Figure: DBMS vs. Flat File (from https://www.guru99.com/what-is-dbms.html#11)

Application of DBMS

Sector	Use of DBMS
Banking	For customer information, account activities, payments, depos loans, etc.
Airlines	For reservations and schedule information.
Universities	For student information, course registrations, colleges and grac
Telecommunication	It helps to keep call records, monthly bills, maintaining balance
Finance	For storing information about stock, sales, and purchases of fin instruments like stocks and bonds.
Sales	Use for storing customer, product & sales information.
Manufacturing	It is used for the management of supply chain and for tracking production of items. Inventories status in warehouses.
HR Management	For information about employees, salaries, payroll, deduction, generation of paychecks, etc.

Figure: Application of DBMS (from https://www.guru99.com/what-is-dbms.html#11)

Characteristics of Database Management System

DBMS Characteristics

The characteristics of a Database Management System (DBMS) define its core functionalities and attributes. These characteristics make it possible for a DBMS to efficiently store, organize, and manage large volumes of data. Here are some key characteristics of a DBMS:

1. **Data Abstraction**: A DBMS hides the physical storage details and presents data to users in a logical view. This allows users to interact with the data without needing to know how it is physically stored.

Top

2. **Data Independence**: A DBMS allows changes to the physical storage or access methods without affecting the application programs that use the data. This separation of physical and logical data is known as data independence.
3. **Efficient Data Access**: A DBMS uses sophisticated algorithms and indexing techniques to provide fast and efficient access to data.
4. **Data Integrity and Consistency**: A DBMS enforces rules and constraints to ensure that the data stored in the database is accurate, consistent, and valid.
5. **Security**: A DBMS provides robust security features, including user authentication, authorization, and access control, to prevent unauthorized access to the data.
6. **Concurrency Control**: A DBMS allows multiple users to access the database simultaneously while ensuring that their transactions do not interfere with each other and that the data remains consistent.
7. **Transaction Management**: A DBMS supports transactions, which are sequences of operations that are executed as a single unit. The DBMS ensures that transactions are executed in a way that preserves the integrity and consistency of the data.
8. **Backup and Recovery**: A DBMS provides features for backing up and restoring data in case of failures, ensuring data availability and durability.
9. **Data Redundancy and Duplication Control**: A DBMS minimizes data redundancy and duplication by storing data in a structured way and using normalization techniques.
10. **Multi-user and Multi-access Support**: A DBMS supports multiple users and provides different levels of access to the data based on user roles and privileges.
11. **Data Relationships**: A DBMS allows the definition of relationships between different data items, making it possible to represent complex data structures and associations.
12. **Scalability**: A DBMS can handle large volumes of data and can scale to accommodate growing data requirements.
13. **Data Dictionary**: A DBMS maintains a data dictionary, which is a catalog of metadata that describes the structure, data types, constraints, and other attributes of the database.
14. **Data Modeling and Design Tools**: A DBMS provides tools for designing and modeling the database schema, making it easier to create and manage the database.
15. **Query Language Support**: A DBMS supports a standard query language, such as SQL, for retrieving and manipulating data.

ACID

ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. It is a set of properties that ensure reliable processing of database transactions. These properties are essential for the correct functioning of a database system, especially in a multi-user environment where many transactions can occur simultaneously. Below is an introduction to each of the four ACID properties:

1. ***Atomicity**: This property ensures that a transaction is treated as a single, indivisible unit of work.* **Either all the operations in a transaction are executed successfully, or none of them are executed.** *If a transaction is interrupted (due to a system crash or other failure), any changes made during the transaction are rolled back to leave the database in its original state.*
2. ***Consistency**: This property ensures that a transaction brings the database from one consistent state to another consistent state.* **The database must satisfy a set of integrity constraints, such as unique key constraints and referential integrity.** *If a transaction would violate these constraints, it is rolled back, and the database remains unchanged.*
3. ***Isolation**: This property ensures that* **concurrent transactions are executed in such a way that the results are the same as if they had been executed one after the other.** *Isolation prevents the "dirty reads" problem, where one transaction reads uncommitted changes made by another transaction.*
4. ***Durability**: This property ensures that* **once a transaction has been committed, its effects are permanent and will survive any subsequent failures** *, such as a system crash or power outage. This is typically achieved by storing the transaction's changes in a durable storage medium, such as a hard disk, and maintaining logs that can be used to recover the database after a failure.*

Together, these ACID properties provide a strong guarantee of the reliability and integrity of database transactions. They are a fundamental concept in database management systems and are crucial for applications that require high levels of data integrity, such as financial systems, airline reservation systems, and e-commerce systems.

Top

View of Data

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data.

A major purpose of a database system is to provide users with an abstract view of the data.

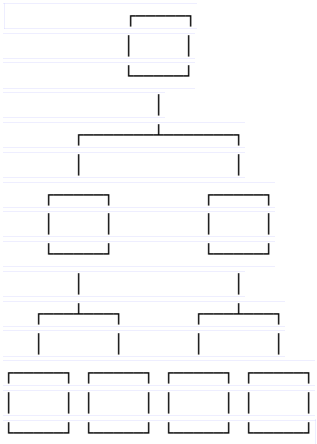
- **Data models**: A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
- **Data abstraction**: Hide the complexity of data structures to represent data in the database from users through several levels of data abstraction.

Data Models

Data models serve as a blueprint for the logical structure of a database, establishing how data is interconnected and processed within the system.

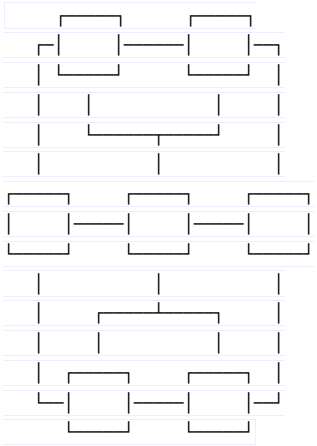
Figure: Evolution of DBMS categories (from https://www.guru99.com/introduction-to-database-sql.html)

- **Hierarchical**: This is one of the oldest types of data models. This system employs a "parent-child" data storage relationship, resembling a tree structure where nodes denote records and branches denote fields.
  - Despite its rare use today, an example of a hierarchical database is the Windows registry used in Windows XP, which stores configuration settings as tree structures with nodes.
  - IBM's Information Management System (IMS). IMS is one of the oldest database systems still in use and it primarily uses a hierarchical model. It's often used in banking, manufacturing, and other industries where high-volume transaction processing is critical.

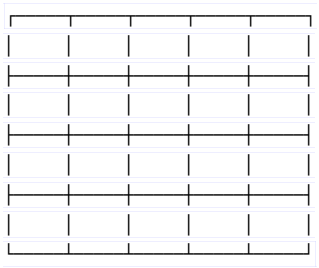


Top

- **Network DBMS**: This system allows for many-to-many relationships, often leading to more complex database structures.
  - The Raima Database Manager (RDM) Server exemplifies a database management system implementing the network model.
  - Integrated Data Store (IDS) is known as the first database system that used the network model.



- **Relational DBMS**: This system characterizes database relationships in the form of tables, or **relations**. Each row is a record, and each column is an attribute. Tables can be related to each other using keys (primary keys and foreign keys). This model is known for its simplicity and efficiency in handling large amounts of data.
  - This type of DBMS enjoys widespread use, with systems like **MySQL**, **Oracle**, and **Microsoft SQL Server database** serving as examples.



- **Object Oriented Relation DBMS**: This type supports the storage of new data types in the form of objects, which have attributes (like gender, age) and methods that dictate data operations.
  - **ObjectDB** stands as an example of an object-oriented relational DBMS.
  - Another example is db4o (database for objects), which is designed for .NET and Java.
- **Semi-structured data model**: a type of data model that falls between the fully structured and unstructured data models. In the context of data modeling, XML (Extensible Markup Language) is one of the most popular semi-structured data formats.

The **relational model** is the most commonly used DBMS, as it saves data in table formats and uses SQL as its standard query language.

Data Abstraction

Levels of abstraction in database management systems refer to the different layers at which data is managed, each providing a different perspective of the data. The three primary levels of abstraction are the **physical level**, the **logical level**, and the **view level**.

- **Physical level**: This is the lowest level of abstraction. **It deals with the physical storage of the data, such as how the data is stored on disk or in memory.** It describes the actual format of the data on the storage medium, the data structures used (like B-trees or hash tables), and the methods for accessing them. This level is concerned with topics like disk blocks, track sectors, data compression, and data encryption.
  - For example, at this level, we might describe how a record, such as information about an

Top

instructor, is physically stored in terms of bytes and blocks on a disk drive.

- **Logical level**: The logical level is a higher level of abstraction, and **it describes what data is stored in the database and the relationships among that data.** At this level, we talk about tables, columns, rows, and indexes, but not about the physical details of how the data is stored. It's about the schema of the database.
  - For example, a database could have an "instructors" table, with columns for "ID," "Name," "Department," and "Salary." This level of abstraction describes what the data is, but not how it's organized or stored on disk.

```
type instructor = record
    ID : string;
    name : string;
    dept_name : string;
    salary : integer;
end;
```

- **View level**: The highest level of abstraction is the view level. **It involves how the data is presented to end users and application programs.** A view is a tailored presentation of the data contained in one or more tables, which can be used to hide complexity or to secure certain parts of the database. A view can show a subset of the data from a table, combine data from multiple tables, or even hide columns (like salary information). Views are often used to provide a specific perspective on the data that is most relevant to a particular group of users or to hide sensitive data from unauthorized users.

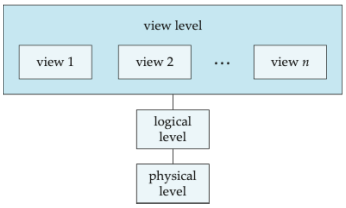


Figure: An architecture for a database system (from Database System Concepts 7th Ed)

The levels of abstraction in database management systems help in the organization, management, and retrieval of data. They allow the physical implementation details to be **separated** from the logical design and presentation of the data, making it easier for different users and applications to interact with the database in a meaningful way.

Instances and Schemas

In database management systems, the terms **Instances** and **Schemas** refer to different aspects of a database.

Schemas

A schema is similar to a blueprint for the database. It defines the logical structure of the database - what data is stored, how it is organized, and the relationships among the data. There are two main types of schemas:

1. **Logical Schema:** This describes the logical structure of the entire database. It defines what data will be stored and how it will be organized in terms of tables, columns, and relationships. For example, a logical schema could describe a database for a bank, where there are tables for "customers" and "accounts," with columns like "name," "address," and "balance," and a relationship indicating which accounts belong to which customers. This is analogous to type information in a programming language.
2. **Physical Schema:** This describes how the data is physically stored and accessed on the computer's storage medium (e.g., hard disk or solid-state drive). It deals with issues like data block size, clustering, indexing, and more. Users typically do not interact with the physical schema directly. Instead, the database management system translates operations on the logical schema into operations on the physical schema.

Instances

An instance of a database refers to the actual content of the database at a particular point in time. It is a snapshot of the data in the database. For example, the instance could be the specific records of customers and their account balances in a bank at 10 AM on a specific day. The instance

Top

changes over time as data is added, modified, or removed. This is analogous to the value of a variable in a programming language. While the schema remains relatively constant over time (it changes infrequently and typically requires a redesign of the database), the instance can change rapidly as the data in the database is updated.

In conclusion, schemas and instances are essential concepts in database management systems. The schema is a static aspect that defines the structure of the database, while the instance is a dynamic aspect that represents the actual data in the database at a specific moment in time.

Physical Data Independence

Physical data independence is an important concept in database management systems (DBMS). **It refers to the ability to change the physical schema of a database without affecting the logical schema or application programs.**

In other words, you can modify how data is stored and accessed on the physical storage medium (e.g., hard disk or solid-state drive) without having to change the overall logical structure of the data or the way that applications interact with the data.

Database Languages

Database languages are specialized languages used to create, retrieve, update, and manage data in a database management system (DBMS). They play a crucial role in interacting with databases, allowing users and applications to perform a variety of tasks related to data handling.

- **Data Definition Language (DDL)**: Used to define and modify the structure of database objects like tables, indexes, and relationships. It deals with the schema (the logical structure) of the database, not the actual data within it.
- **Data Manipulation Language (DML)**: Used for managing data within database objects like tables. It allows for insertion, updating, and deletion of data.
- **Data Control Language (DCL)**: Used to control access to data in the database. It deals with permissions and security.
- **Transaction Control Language (TCL)**: Used to manage transactions in the database, ensuring data integrity. It controls transactional processes like beginning, committing, and rolling back transactions.
- **Data Query Language (DQL)**: Used specifically for querying data from the database. This is where data is retrieved and presented in a structured format.

There are basically two types of data-manipulation language

- Procedural DQL: require a user to specify what data are needed and how to get those data.
- Declarative DQL (also referred to as non-procedural DQLs): require a user to specify what data are needed without specifying how to get those data.

SQL Query Language is nonprocedural.

Example to find all instructors in Comp. Sci. dept

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

SQL is NOT a Turing machine equivalent language. In other words, it is not as powerful as a universal Turing machine. For example, SQL does not support actions such as input from users, output to displays, or communication over the network.

Such computations and actions must be written in a **host language**, such as C/C++, Java or Python, with embedded SQL queries that access the data in the database.

Application programs generally access databases through one of

- Language extensions to allow embedded SQL
- Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

Top

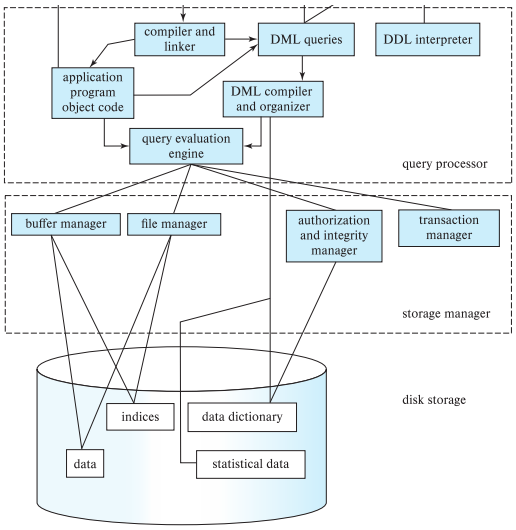
Database Design

The functional components of a database system can be divided into

- The storage manager,
- The query processor component,
- The transaction management component.

Storage Manager

The Storage Manager is an essential component of a database management system (DBMS). It acts as an interface between the low-level data stored in the database and the application programs and queries submitted to the system. Essentially, the Storage Manager handles the interaction with the physical storage of data and ensures that it's done efficiently.



(from Database System Concepts 7th Ed)

Key Responsibilities

- Interaction with the OS File Manager: The Storage Manager communicates with the operating system's file manager to handle the actual reading and writing of data on the storage medium (e.g., hard disk, SSD).
- Efficient Storing, Retrieving, and Updating of Data: It's responsible for ensuring that data is stored, retrieved, and updated efficiently. It determines how data is organized on the disk, manages free space, and handles the physical aspects of data access.

Main Components

- Authorization and Integrity Manager: Ensures that only authorized users can access the data and that the data remains accurate and consistent.
- Transaction Manager: Coordinates database transactions, ensures their atomicity, consistency, isolation, and durability (ACID properties), and manages concurrency control.
- File Manager: Manages the allocation of space on disk and the data structures used to represent information stored on disk.
- Buffer Manager: Manages the buffer pool, a memory area that holds pages fetched from disk and determines which pages to fetch into memory and which to write back to disk.

Data Structures Implemented

- Data Files: These store the actual database itself.
- Data Dictionary: Also known as system catalog or metadata, it stores information about the structure of the database, including the schema of the database, information about tables, columns, keys, and indexes.
- Indices: These are data structures that can provide fast access to data items. A database index provides pointers to those data items that hold a particular value, allowing for faster query performance.

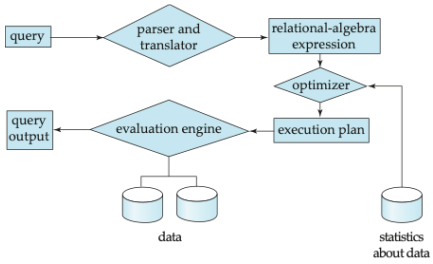
Top

Query Processor

The query processor components include:

- DDL interpreter: interprets DDL statements and records the definitions in the data dictionary.
- DML compiler: translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
    - The DML compiler performs query optimization; that is, it picks the lowest cost evaluation plan from among the various alternatives.
- Query evaluation engine -- executes low-level instructions generated by the DML compiler.

The query processor ensures that user queries are executed accurately and efficiently, returning the desired results in the shortest possible time.



(from Database System Concepts 7th Ed)

Transaction Management

The transaction management component ensures that all transactions adhere to the ACID properties (Atomicity, Consistency, Isolation, Durability). It ensures that all database transactions are processed reliably and system integrity is maintained.

- Concurrency Control Manager: Ensures that multiple transactions can occur simultaneously without interfering with each other, preserving data consistency.
- Recovery Manager: Handles system failures by restoring the database to a consistent state using logs and backups.

Database Architecture

1 tier Architecture: This architecture is the simplest form of database structure where the Client, Server, and Database all operate on the same device. For example, when you install a database system on your local machine and use it to practice SQL queries, you are utilizing a single-tier architecture. However, this architecture is seldom used in production environments.

Figure: 1-tier Architecture Diagram (from https://www.guru99.com/dbms-architecture.html)

2-tier Architecture: A two-tier architecture is a more complex structure where:

- The presentation layer is executed on a client device, such as a PC, mobile device, or tablet.
- The data is stored on a separate server.

Top

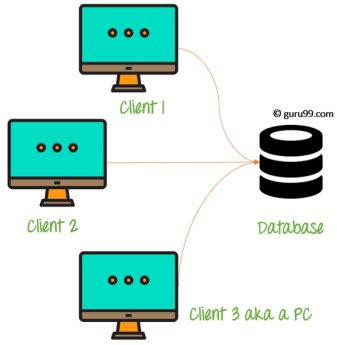


Figure: 2-tier Architecture Diagram (from https://www.guru99.com/dbms-architecture.html)

3-tier Architecture: This architecture is an advanced version of the two-tier structure. The three-tier architecture is divided into the following layers:

- Presentation layer, which runs on your personal devices like a PC, tablet, or mobile.
- Application layer, which is executed on a server.
- Database Server, where all the data is stored.

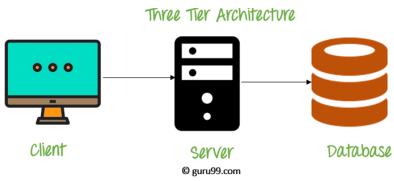
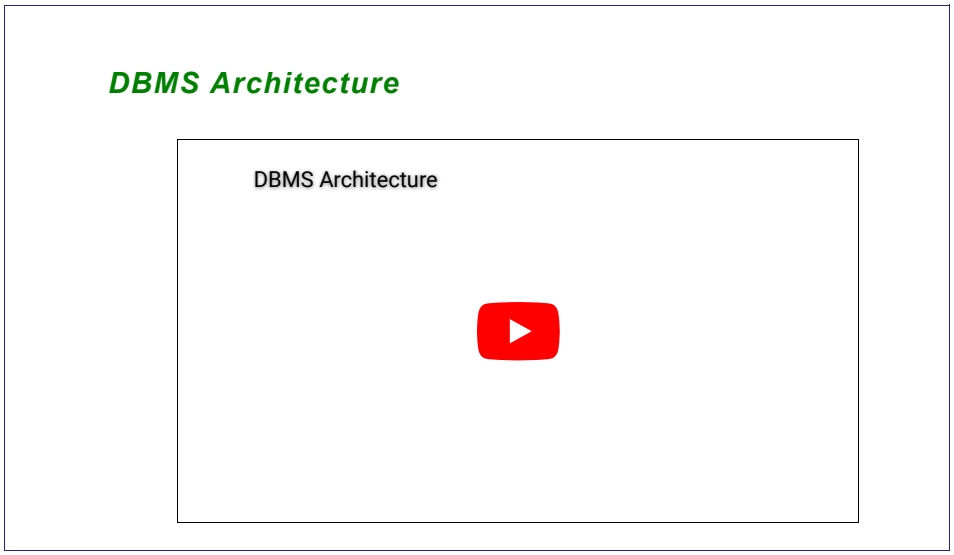


Figure: 3-tier Architecture Diagram (from https://www.guru99.com/dbms-architecture.html)



History of Database Systems

The history of database systems spans several decades and reflects the evolving needs of businesses and organizations for organized and efficient data management. Below is an overview of the major milestones in the history of database systems:

- **1960s – Hierarchical and Network Database Models:**

  In the early 1960s, the hierarchical and network database models emerged. The hierarchical model organized data in a tree-like structure, while the network model allowed for more complex relationships. IBM's Information Management System (IMS) is an example of a hierarchical database system.

- **1970s – Relational Database Model:**

  In 1970, Edgar F. Codd, a researcher at IBM, introduced the relational database model in his paper

Top

"A Relational Model of Data for Large Shared Data Banks." This model organized data in tables with rows and columns and allowed for more flexible and efficient querying of data. The relational model became the dominant approach to database management.

- **1980s – SQL and Commercial RDBMS:**

  The Structured Query Language (SQL) became the standard language for querying and managing relational databases. Commercial relational database management systems (RDBMS) such as Oracle, IBM DB2, and Microsoft SQL Server were introduced.

- **1990s – Object-Oriented Databases and OLAP:**

  Object-oriented databases emerged, offering a new approach to data modeling that incorporated object-oriented programming concepts. Online Analytical Processing (OLAP) systems were developed for multidimensional analysis of data, enabling more complex business intelligence queries.

- **2000s – NoSQL Databases:**

  As the volume and variety of data grew, there was a need for more scalable and flexible database systems. NoSQL databases, including key-value, document, columnar, and graph databases, became popular for managing large-scale, unstructured, and semi-structured data.

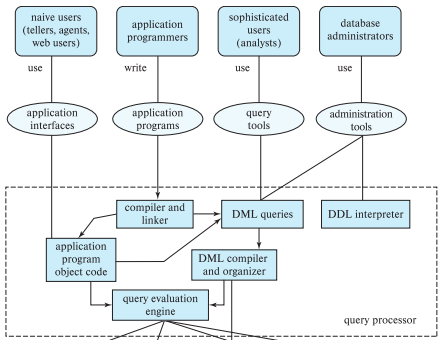
- **2010s – Big Data and Cloud Databases:**

  Big Data technologies like Hadoop and Spark emerged to handle massive datasets. Cloud databases became increasingly popular, offering scalable, cost-effective, and flexible data management solutions. Services like Amazon RDS, Google Cloud SQL, and Microsoft Azure SQL Database provided managed relational database services in the cloud.

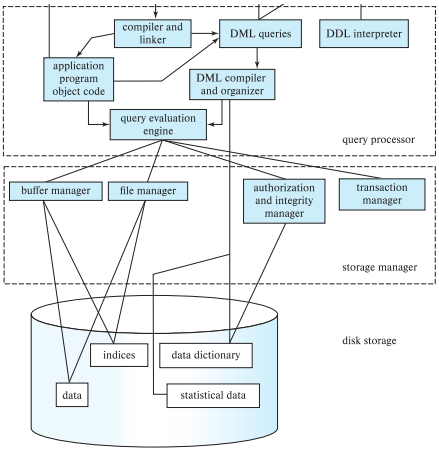
- **2020s and Beyond – AI, Machine Learning, and Edge Computing:**

  Database systems continue to evolve with advancements in AI and machine learning, enabling more intelligent data analysis and decision-making. Edge computing is also influencing database systems, as data is increasingly processed closer to its source.

Database Users

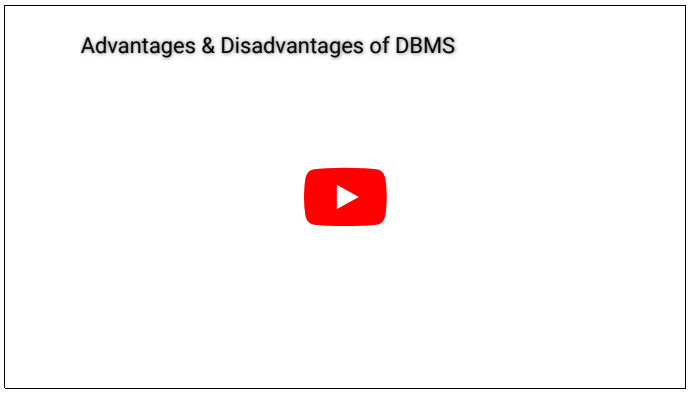


Top



(from Database System Concepts 7th Ed)

Advantages & Disadvantages of DBMS



History of Database Applications

Relational Database vs Non-relational Database

Relational Database

In the digital era, data can be broadly classified into two categories: operational data and analytical data.

Top

- **Operational Data** pertains to the data used for daily transactions which necessitates freshness,

for instance, the real-time data of product inventory or bank balance. This type of data is captured using Online Transaction Processing (OLTP) systems.

- **Analytical Data** is employed by enterprises to derive insights concerning customer behavior, product performance, and predictive analysis. It encapsulates data accumulated over a certain duration and is typically stored in Online Analytical Processing (OLAP) systems, data warehouses, or data lakes.

Databases offer the most effective method to permanently save and retrieve operational and analytical data in a digital format.

Companies, according to their specific project needs, should opt for a database that can:

- Accommodate all sorts of data.
- Provide swift access to the required data.
- Deliver real-time insights to facilitate strategic business decision-making.

Most enterprises require both OLTP (operational) and OLAP (analytical) systems for their data storage. They can use either a relational database, a non-relational database, or a combination of both to cater to their business requirements.

A **relational database**, or Relational Database Management System (RDBMS), organizes information within tables. Often, these tables have shared information between them, resulting in the formation of relationships among them. This characteristic is what gives the relational database its name.

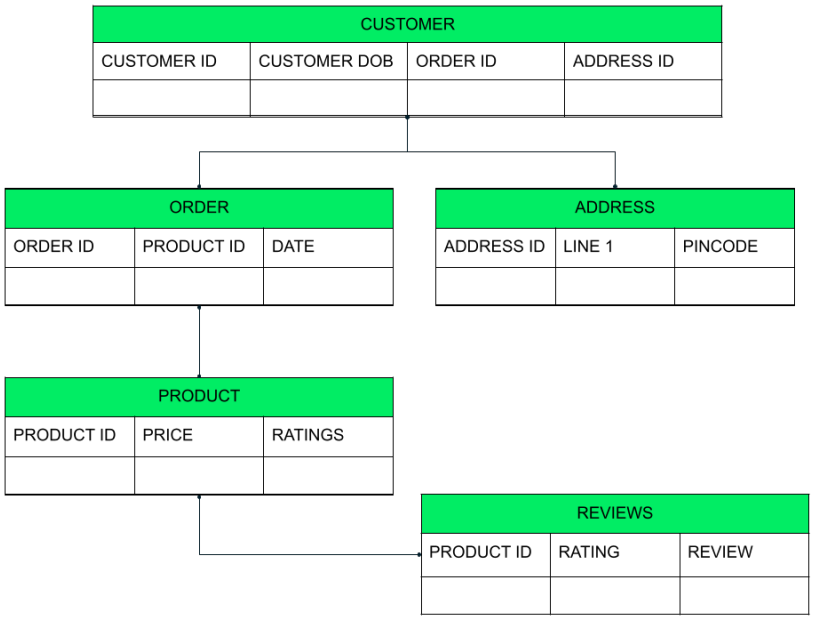


Figure: Data is stored as tables with rows and columns in a relational database (from https://www.mongodb.com/compare/relational-vs-non-relational-databases)

Structured Query Language (SQL) is a prevalent method for interaction with relational database systems. It's pronounced as "S-Q-L" or alternatively as "See-Quel".

Databases like MySQL, Oracle, Ms SQL server, Sybase, and others employ SQL. Developers utilize SQL queries for executing CRUD (Create, Read, Update, Delete) operations.

The syntaxes used across these SQL-based databases share similarities, albeit with some variations in syntax and occasionally proprietary SQL syntaxes. An example of a basic query is:

```
SELECT PRODUCT_NAME, PRICE FROM PRODUCT WHERE PRODUCT _ID = 23;
```

Advantages of relational databases

- **ACID Compliance**: ACID, standing for Atomicity, Consistency, Isolation, and Durability, is a set of properties that ensure reliable processing of database transactions. It stipulates that if any part of a transaction fails, the entire transaction is aborted, leaving the database in its original state before the transaction began.
- **Data Accuracy**: The use of primary and foreign keys can prevent duplication of data, thereby improving data accuracy, as it prevents repeated information.

Top

- **Normalization**: Normalization is a process that organizes data to minimize redundancy and anomalies, which in turn reduces storage costs.
- **Ease of Use**: Relational Database Management Systems (RDMS), or SQL databases, have been prevalent for a significant period, prompting the development of a vast array of tools and resources to assist users. The English-like syntax of SQL makes it accessible to non-developers for generating reports and queries.

Disadvantages of relational databases

- **Scalability**: RDMSs were traditionally designed to operate on a single machine, implying that in case of increased data size or access frequency, the solution would be to upgrade the machine's hardware - a process known as vertical scaling. This can be a costly endeavor with limitations, especially when the expense exceeds the benefits, or the hardware can no longer accommodate the database. Upgrading to a more capable machine is an alternative, but it comes with a high cost.
- **Flexibility**: The schema in relational databases is inflexible. Once you define columns, data types, and any constraints such as length or format, it becomes challenging to alter the data structure. If changes are necessary later, the entire data set must be modified, which may require temporary database downtime.
- **Performance**: The performance of a relational database is directly proportional to the complexity of the tables in terms of their number and the volume of data they contain. As these factors increase, the time required to execute queries also rises.

Non-relational Database

A **non-relational database**, alternatively referred to as **NoSQL** (Not Only SQL), represents a class of database that does not adhere to the conventional structures of tables, fields, and columns embodied by relational databases.

When a relational database manages colossal data volumes, response time can become sluggish. One approach to mitigate this is to "**scale up**" or "**vertically scale**" the systems by enhancing the existing hardware. However, an alternate solution is to distribute the database load across multiple hosts as the load intensifies, a process known as "**scaling out**" or "**horizontal scaling**".

Non-relational databases are engineered with cloud technology in mind, hence they excel at horizontal scaling.

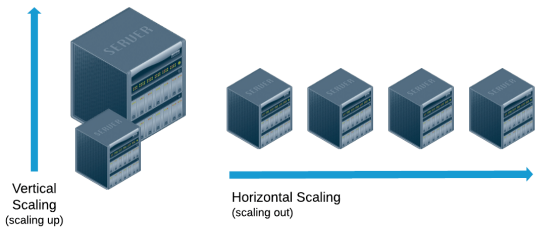
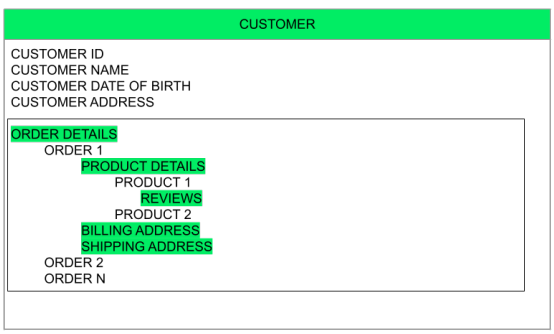


Figure: Horizontal Scaling (scaling out) vs Vertical Scaling (scaling up) (from https://www.section.io/blog/scaling-horizontally-vs-vertically/#:~:text=What's%20the%20main%20difference%3F,as%20%E2%80%9Cscaling%20up%E2%80%9D).)

- **Document databases**: This type of database stores data in documents, typically in a JSON-like structure that supports diverse data types, such as strings, integers, floats, longs, dates, objects, arrays, and even nested documents. Data is stored in pairs, akin to key/value pairs.



Top

Figure: How data is stored in a document non-relational database (from https://www.mongodb.com/compare/relational-vs-non-relational-databases)

- **Key-value database**: This basic type of database saves information in two components: a key and a value. The key is subsequently used to fetch the information from the database.

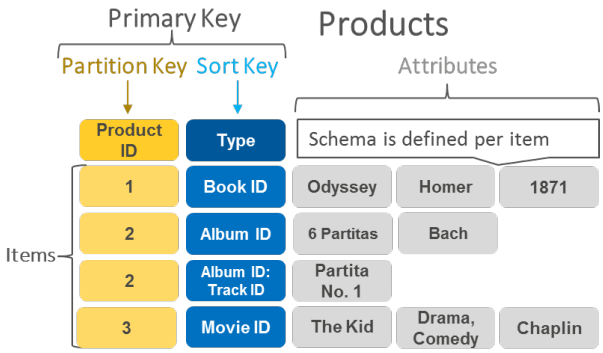


Figure: data stored as key-value pairs in DynamoDB. (from https://aws.amazon.com/nosql/key-value/)

- **Graph databases**: Graph databases are the most specialized types of non-relational databases. They employ a structure composed of elements called nodes that store data. Edges between these nodes contain attributes detailing their relationship.

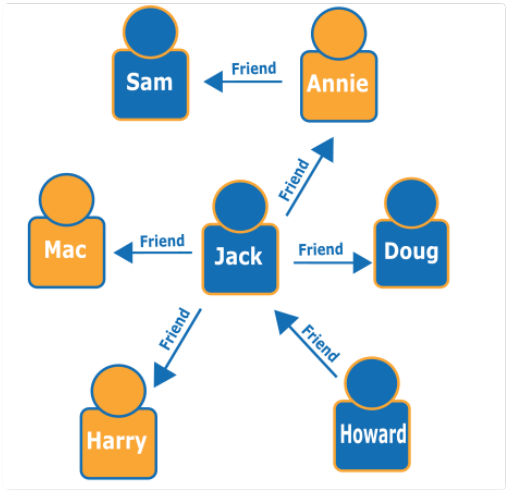
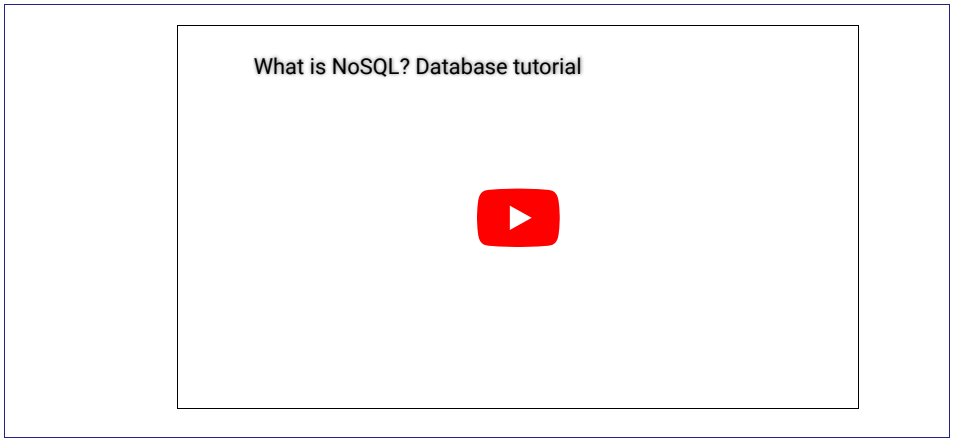


Figure: a social network graph. (from https://aws.amazon.com/nosql/graph/)

- **Wide-column databases**: Wide-column databases store data in tables, columns, and rows, much like relational databases. Nevertheless, the column names and formats can vary in each row and can even be stored on different servers. These databases are considered two-dimensional key-value stores due to their use of multi-dimensional mapping to reference data by row and column.

	Document Database	Column Store Database	Key-Value Store Database	Graph Database
Performance	High	High	High	Moderate
Availability	High	High	High	High
Flexibility	High	Moderate	High	High
Scalability	High	High	High	Moderate
Complexity	Low	Low	Moderate	High

Top



Relational Database vs Non-relational Database

Features	Non-Relational	Relational
Availability	High	High
Horizontal Scaling	High	Low
Vertical Scaling	High	High
Data Storage	Optimized for huge data volumes	Medium to large data
Performance	High	Low To Medium
Reliability	Medium	High (Acid)
Complexity	Low	Medium (Joins)
Flexibility	High	Low (Strict-Schema)
Suitability	Suitable For OLAP and OLTP	Suitable For OLTP

Considerations for using a relational database:

- Relational databases are ideal for projects where data is predictable in structure, size, and access frequency.
- Through normalization, duplication and anomalies are minimized, reducing the data size on disk and decreasing the future need for vertical scaling.
- If relationships between entities are critical, relational databases are the preferable choice.
- While non-relational databases can nest documents within documents, keeping co-accessed data together, relational databases may be a better option for large datasets with intricate structures and relationships where embedding might not create sufficient clarity.
- Relational Database Management Systems (RDMS) have a long-standing presence, offering extensive support, tools, and integration capabilities with data from other systems.

When to consider a non-relational database:

- Non-relational databases are diverse, each with its own set of strengths and weaknesses.
- However, they share some common advantages. If your data is expected to be flexible in shape or size or likely to change in the future, a non-relational database could be a suitable choice.
- Designed with cloud technologies in mind, modern NoSQL databases naturally support horizontal scaling, allowing numerous smaller servers to be activated to handle increased loads.

Top