22XX405 - DATABASE MANAGEMENT SYSTEM

UNIT 1 & LP 2 - MODERN DATABASES, DBMS ARCHITECTURE AND COMPONENTS

1. EVOLUTION OF DATABASES:

The history of databases begins in the 1960s with the computerization of databases. Computers emerged as a more cost-effective option for organizations. It also became easier to shift data storage and databases to computers. The chronological order of the development of databases is as follows:

- (1970s-1990s) - Flat files, hierarchical and network
- (1980s-present) - Relational
- (1990s- present) - Object-oriented, object-relational, web-enabled

1.1 FLAT FILES:

Flat files databases were used during the 1970s-1990s. This is a type of database system that stores data in a single file or table. They are basically text files, where every line contains one record and fields either have fixed lengths or are separated by commas, whitespaces and tabs. Such a file cannot contain multiple tables. Below is an example of what a flat file database looks like. This text file stores lines of data, where each line represents a record. The fields OrderID, CustomerID and OrderDate are separated by commas.

1.2 HIERARCHICAL DATABASE SYSTEMS:

Hierarchical database systems that were in use during the same era store data in a hierarchically arranged manner. Think about it this way: parents can have many children, but one child can only have one parent. In other words, the database represents a one-to-many relationship: all attributes of a specific record are listed under an entity type. Below is an example of how data is stored in a hierarchical database. In this case, it is data on college students who are taking different courses. A course can be assigned to only a single student, but a student can take as many courses as they want. Thus, there is a one-to-many relationship.
There are three students:John, Anil and Rohan
And there are four courses: C#, Perl, Python and Java.
Student and Course are the entity types. John takes C# and Anil takes both Python and Java. Rohan takes Perl.

1.3 NETWORK DATABASES:

Network databases were introduced by Charles Bachmann. Unlike the hierarchical database model, a network database allows multiple parent and child relationships. In other words, many-to many relationships. In network database terminology, a child record is known as a member. A member or child can be reached through more than one parent, which is called an owner. A network database has a graph-like structure, and it allows you to represent more complex relationships among data. Here's an example of a network database. A teacher can teach multiple courses and a course can have multiple teachers teaching it.

In this era, a language known as the SEQUEL query language was used to work with databases. Later on, with relational databases, this developed into SQL (Structured Query Language) which was made a standard query language to work with databases by the American National Standards Institute once relational database systems were introduced.

1.4 RELATIONAL DATABASE SYSTEM:

The relational database system that was introduced in the 1980s is still the most used database system. It was invented by E. F. Codd and it's the successor of hierarchical and network database systems. It was viewed as a major paradigm shift in database technology.

In a relational database system, data is stored in tables. The columns of the table hold attributes of the data. Each record usually has a value for each attribute, making it easy to establish the relationships between data points. In a relational database, each row in the table is a record with a unique ID attribute called the primary key. A relational database stores and provides access to data that are related to one another using an attribute known as a foreign key.

Here's an example of what a Relational Database would look like. Here, there are tables with attributes/ columns that store rows/records of data in them. The relationships between data in tables are established using key columns known as foreign keys that are themselves the primary key(s) of a given table. For example, the primary key of the PROFESSOR table is PROF_ID and in the CLASS table, it's there as a foreign key. It creates the relationship between the PROFESSOR table and the CLASS table. Another example, the COURSE_ID is the primary key of the COURSE table, and it is there in the CLASS table as a foreign key. It establishes the relationship between the COURSE table and the CLASS table.

1.5 OBJECT-ORIENTED DATABASES:

In the 1990s, object-oriented databases were introduced. This was when the object-oriented (OO) programming paradigm became popular and there was a need to represent data in a system as objects as well. Unlike relational databases, object-oriented databases work in the framework of real programming languages like Java and C++, for example. Below is what an object-oriented database looks like. Instead of tables, there are entities or classes like Author, Book and Customer with their attributes and behaviors. It's possible to represent data according to OO concepts like inheritance and parent-child relationships among data. For example, an Author and Customer are both descendants of Person. Thus, a person is a generic entity that can represent both an Author and a Customer.

1.6 NOSQL DATABASES:

Relational databases that are widely used even at present only allows to store structured data. Later on, there was a need to work more and more with unstructured data. This was when NoSQL databases came about as a response to the Internet and the need for faster speed and the processing of unstructured data. NoSQL databases are preferred over relational databases because of their speed and flexibility in storing data. It does not store data in relations or tables that belong to a strict structure. Data can be stored in an ad-hoc manner and they allow to store and process high volumes of different kinds of data. NoSQL databases are capable of processing unstructured big data that's generated by social media, IoT and others. Therefore, social platforms like Twitter, LinkedIn, Facebook, and Google for example makes use of NoSQL databases.
These are some of the advantages of NoSQL databases:

- Higher scalability
- Distributed
- Lower costs
- A flexible schema
- Can process unstructured and semi-structured data
- Has no complex relationships
- Over time there were different types of NoSQL databases that were introduced:

Document databases store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects.

Key-value databases are a simpler type of database where each item contains keys and values.

Wide-column databases store data in tables, rows, and dynamic columns.

Graph databases store data in nodes and edges. Nodes typically store information about people, places, and things, while edges store information about the relationships between the nodes.

2. MODERN DATABASES - NOSQL & NEWSQL

2.1 SQL:

SQL, also known as Structured Query Language, is a flexible programming language specifically designed for the management and manipulation of relational databases. It acts as a standardized communication tool between databases and applications. SQL empowers users to define, access, and modify data within a database system. With a rich set of commands, SQL enables tasks like creating and modifying database structures (DDL), manipulating data (DML), and managing access and permissions (DCL). It excels in executing complex operations such as sorting, joining, aggregating, and grouping data. It also ensures an efficient data retrieval.

SQL finds extensive applications across various fields, serving as an essential tool for database administrators, developers, and data analysts. It is widely supported by popular database management systems like MySQL, Oracle, SQL Server, and PostgreSQL.

Advanced SQL allows individuals to proficiently handle and interact with databases, which helps in enabling critical tasks like data retrieval, manipulation, and database administration. Its user-friendly nature, adaptability, and widespread adoption have solidified SQL's position as an indispensable language in data management and analysis.

2.2 NO SQL:

NoSQL, also known as "Not Only SQL," represents a category of databases that offers an alternative approach to data management compared to traditional SQL databases. It provides a flexible and scalable solution for storing and handling diverse data types, such as unstructured and semi-structured data which is found in social media posts, sensor data, and real-time streams.

Unlike SQL databases, NoSQL databases do not adhere to a rigid schema and instead employ various data models, including key-value stores, document databases, column-family stores, and graph databases. SQL's versatility enables dynamic data modeling to cater to the specific requirements of various applications which make it adaptable and customizable.

NoSQL databases prioritize scalability, high performance, and fault tolerance. All these features make the databases well-suited for those situations which invlove large data volumes, rapid data processing, and distributed computing. They excel at handling the complexities of modern data environments.

Organizations adopting NoSQL databases can leverage their capabilities to overcome challenges posted by big data, real-time processing, and dynamic data structures. However, it is crucial to carefully assess the requirements before selecting a NoSQL database solution that best aligns with the specific application needs.

2.3 NEW SQL:

NewSQL represents a category of database systems that blend the advantages of traditional SQL (relational) databases with the scalability and performance benefits offered by NoSQL databases. NewSQL aims to address the limitations of SQL databases, particularly in distributed environments which require high scalability.

These databases retain the essential ACID (Atomicity, Consistency, Isolation, Durability) properties for data integrity while implementing innovative techniques for handling large datasets and achieving impressive scalability. Through methods such as sharding, replication, and distributed architectures, NewSQL databases efficiently manage extensive workloads and meet the demands of modern applications.

Unlike NoSQL databases, NewSQL databases maintain compatibility with SQL, providing users with familiar SQL-based interfaces, complex queries, and transaction support. This makes them a compelling choice for organizations seeking both the scalability of NoSQL and the reliability of SQL. NewSQL databases use the flexibility and scalability of NoSQL databases while preserving the integrity and consistency of SQL databases.

2.4 SQL vs NOSQL vs NEWSQL:

Below table illustrates the distinctions among SQL, NoSQL, and NewSQL databases:

S. No.	SQL	NO SQL	NEW SQL
1.	SQL databases organize data in structured tables.	NoSQL databases support various data models.	NewSQL databases maintain a relational data structure.
2.	SQL databases employ a rigid data schema that defines the structure and organization of data.	NoSQL databases have a flexible schema.	NewSQL databases also support a flexible schema.
3.	SQL is the standard query language used.	NoSQL databases have diverse query languages.	NewSQL databases also use SQL for queries.
4.	SQL databases have limited scalability.	NoSQL databases offer high scalability.	NewSQL databases provide enhanced scalability.
5.	SQL databases adhere to ACID properties	NoSQL databases have varying compliance.	NewSQL databases maintain ACID compliance.
6.	SQL databases generally do not have a distributed architecture as a characteristic feature.	NoSQL databases are specifically designed to operate in distributed environments.	NewSQL databases are specifically engineered with distributed architecture as a core principle.
7.	SQL databases support transactions.	NoSQL databases may have varied transaction support.	NewSQL databases offer transaction support.
8.	SQL databases are suitable for traditional applications with structured data.	NoSQL databases excel with unstructured and diverse data.	NewSQL databases cater to scalable applications with relational data and ACID compliance.

3. DATABASE ARCHITECTURE - TWO TIER & THREE TIER ARCHITECTURE

A Database stores a lot of critical information to access data quickly and securely. Hence it is important to select the correct architecture for efficient data management. DBMS Architecture helps users to get their requests done while connecting to the database. We choose database architecture depending on several factors like the size of the database, number of users, and relationships between the users. There are two types of database models that we generally use, logical model and physical model. Several types of architecture are there in the database which we will deal with in the next section.

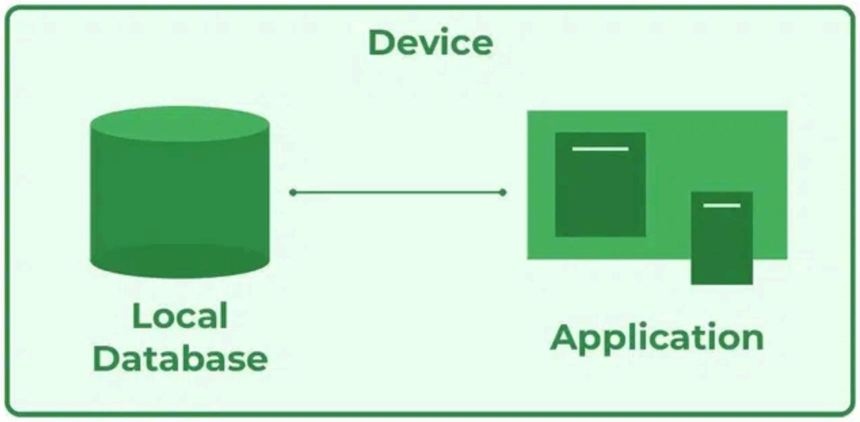
3.1 TYPES OF DBMS ARCHITECTURE:

There are several types of DBMS Architecture that we use according to the usage requirements. Types of DBMS Architecture are discussed here.

- 1-Tier Architecture
- 2-Tier Architecture
- 3-Tier Architecture

3.1.1. 1-TIER ARCHITECTURE:

In 1-Tier Architecture the database is directly available to the user, the user can directly sit on the DBMS and use it, that is, the client, server, and Database are all present on the same machine. For Example: to learn SQL we set up an SQL server and the database on the local system. This enables us to directly interact with the relational database and execute operations. The industry won't use this architecture; they logically go for 2-tier and 3-tier Architecture.



DBMS 1-Tier Architecture

Advantages of 1-Tier Architecture:

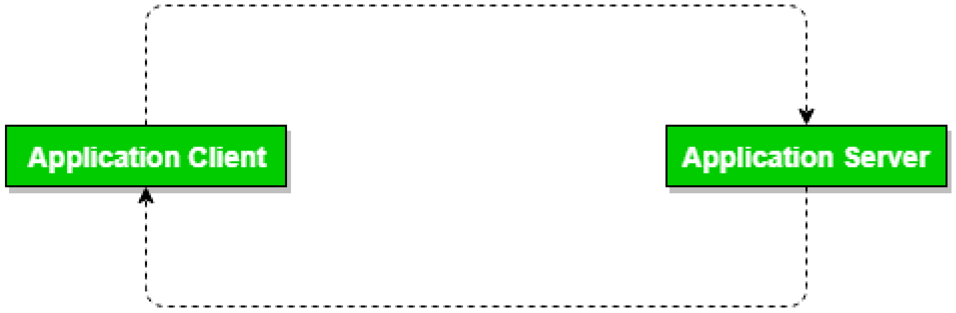
- Simple Architecture: 1-Tier Architecture is the most simple architecture to set up, as only a single machine is required to maintain it.
- Cost-Effective: No additional hardware is required for implementing 1-Tier Architecture, which makes it cost-effective.

- Easy to Implement: 1-Tier Architecture can be easily deployed, and hence it is mostly used in small projects.

3.1.2. 2-TIER ARCHITECTURE:

The 2-tier architecture is similar to a basic client-server model. The application at the client end directly communicates with the database on the server side. APIs like ODBC and JDBC are used for this interaction. The server side is responsible for providing query processing and transaction management functionalities. On the client side, the user interfaces and application programs are run. The application on the client side establishes a connection with the server side to communicate with the DBMS.

An advantage of this type is that maintenance and understanding are easier, and compatible with existing systems. However, this model gives poor performance when there are a large number of users.



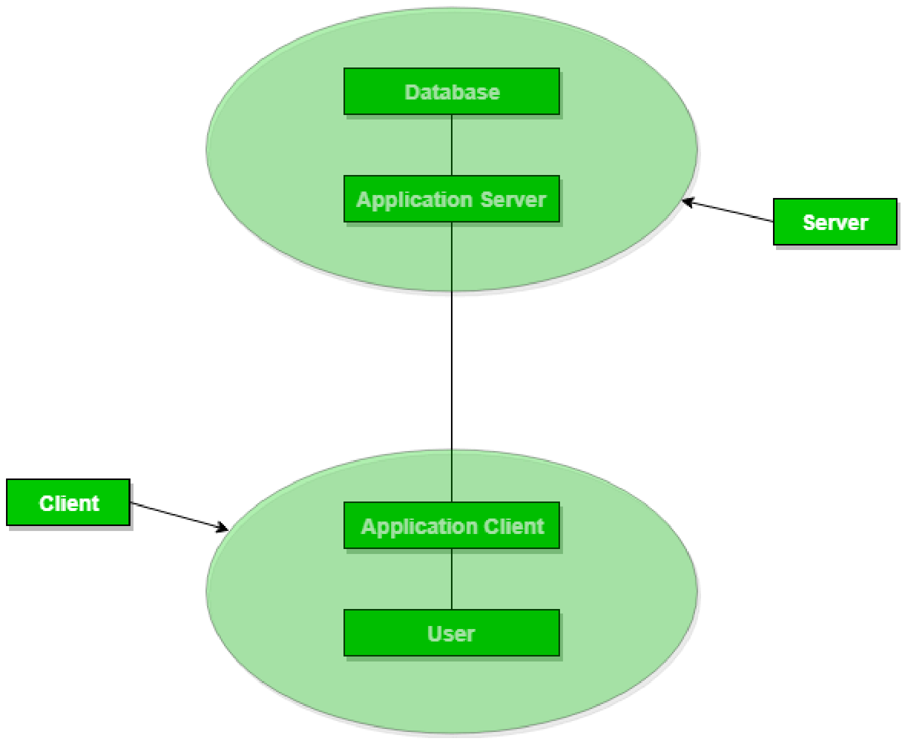
DBMS 2-Tier Architecture

Advantages of 2-Tier Architecture:

- Easy to Access: 2-Tier Architecture makes easy access to the database, which makes fast retrieval.
- Scalable: We can scale the database easily, by adding clients or upgrading hardware.
- Low Cost: 2-Tier Architecture is cheaper than 3-Tier Architecture and Multi-Tier Architecture.
- Easy Deployment: 2-Tier Architecture is easier to deploy than 3-Tier Architecture.
- Simple: 2-Tier Architecture is easily understandable as well as simple because of only two components.

3.1.3 3-TIER ARCHITECTURE:

In 3-Tier Architecture, there is another layer between the client and the server. The client does not directly communicate with the server. Instead, it interacts with an application server which further communicates with the database system and then the query processing and transaction management takes place. This intermediate layer acts as a medium for the exchange of partially processed data between the server and the client. This type of architecture is used in the case of large web applications.



DBMS 3-Tier Architecture

Advantages of 3-Tier Architecture:

- Enhanced scalability: Scalability is enhanced due to the distributed deployment of application servers. Now, individual connections need not be made between the client and server.
- Data Integrity: 3-Tier Architecture maintains Data Integrity. Since there is a middle layer between the client and the server, data corruption can be avoided/removed.
- Security: 3-Tier Architecture Improves Security. This type of model prevents direct interaction of the client with the server thereby reducing access to unauthorized data.

Disadvantages of 3-Tier Architecture:

- More Complex: 3-Tier Architecture is more complex in comparison to 2-Tier Architecture. Communication Points are also doubled in 3-Tier Architecture.
- Difficult to Interact: It becomes difficult for this sort of interaction to take place due to the presence of middle layers.

4. DATABASE USERS:

In simple terms, if we understand, any person who uses a database and avails benefits from the database is known as a database user in DBMS. Database users in DBMS can access the database and retrieve the data from the database using applications and interfaces provided by the Database Management System (DBMS). We can always provide security to our database from being accessed by unauthorized users. Different database users with different login IDs and passwords can only have unrestricted access to that part of the database with which they are associated as per the requirements.

Database users in DBMS can be categorized based on their interaction with the databases. According to the tasks performed by the database users on the databases, we can categorize them into seven categories as follows:

- Database Administrators (DBA)
- Database Designers
- System Analysts
- Application Programmers / Back-End Developers
- Naive Users / Parametric Users
- Sophisticated Users
- Casual Users / Temporary Users

4.1 DATABASE ADMINISTRATORS (DBA):

Database Administrators (DBA) are the most important type of database users in DBMS. Database Administrator is an individual or a team of users who define the database schema and takes charge of controlling various levels of the database within the organization. Database Administrators (DBAs) have full control of the database and they are sometimes known as the super-users of the database. They work alongside developers in order to discuss and design the overall structure of the database including layouts, functionalities, workflow, etc.

Database Administrators (DBAs) can grant or revoke authorization permission to all other users at any point of time. In order to access the database, DBAs have to provide login credentials (account ID and password) to all other users when required. Database Administrators (DBAs) are solely responsible for providing security to the database by restricting unauthorized users from accessing the database. Database Administrators (DBAs) have all the privileges allowed by the DBMS. They have to update the database timely in terms of technology, functionality, and workflow in order to meet future requirements and in making the database ready for future scope.

Database Administrators (Database admins) are also responsible to keep a check on data integrity, data consistency, data redundancy, hardware and software installation requirements, and routine maintenance of the databases. They are also responsible for handling data loss, which can be caused due to any error or even due to system failures. Also, Database Administrators (DBAs) monitor the backup & recovery of the database records and provide technical support as well. If the technical team raises any concern, DBAs have to resolve it.

4.2 DATABASE DESIGNERS:

As the name suggests, Database Designers are the users in DBMS who design and create the structure of the database including triggers, indexes, schemas, entity relationships, tables, constraints, etc. which complete the database. Database designers try to gather information depending upon the requirements related to the database like the layout, looks, database functioning, costing, technologies to be used & implementation techniques, and finally, they design the final layout of the database for programmers to code its logic.

Database Designers are the type of database users in DBMS who are responsible for implementing the overall design of the database. They decide which form of data needs to be stored, what kind of relations exist among different entities of the database, what will be the type of attributes etc.

4.3 NAIVE USERS / PARAMETRIC USERS:

Naive users also known as Parametric End users, don't have any knowledge of DBMS but still frequently use the database applications to get the desired results. With the help of the interface provided by the DBMS applications, Naive users mostly use the database to fill in or retrieve the information (view level of the database). Naive users don't need to be aware of the presence of the database system as they can interact with the database with the help of menu-driven application interface. In simple terms, Naive / Parametric End users work directly on developed applications to access the database indirectly for getting the desired results.

For example: Railway ticket booking users in general are naive/parametric end users as they don't have much knowledge about DBMS and directly use railway booking application to book their tickets.

4.4 SYSTEM ANALYST:

System Analysts are the type of database users in DBMS who analyze the requirements of Naive / Parametric End users. It is their responsibility to check whether all the requirements of end users are satisfied or not. Analyzing feasibility, economic and technical aspects are some of the major responsibilities for a system analyst in DBMS. Sometimes, they are also responsible for the design, structure & functioning of the database. They usually check and gather all the necessary information related to the database, and if needed, they can change or update the final layout of the database as per requirements. System Analysts always make sure that the final product should meet all the requirements.

4.5 APPLICATION PROGRAMMERS / BACK-END DEVELOPERS:

Application Programmers also known as Back-End Developers, are computer professional users who are responsible for developing the application programs (C, C++, Java, PHP, Python, etc.) or the user interface so that other users can use these applications to interact with the database. Application Programmers have deep knowledge of DBMS & databases and know everything in detail. They interact with the database using DML (Data Manipulation Language) queries to store data inside the database and when needed, they can also fetch the data from it. When needed, Application Programmers also specify the modifications needed in the database structure for an application. They are efficient enough in designing or developing their database in any language they know.

4.6 SOPHISTICATED USERS:

Sophisticated users are the type of database users in DBMS who know DBMS (DDL & DML commands) and are familiar with the database. Sophisticated users can be business analysts, engineers, scientists, system analysts, etc. Sophisticated users can develop or access their database applications according to the requirements, without actually writing the program code for it. These users are also known as SQL programmers as they can interact with the database directly using SQL queries using query processors. Using SQL queries, they can fetch the data from the database.

They can also delete, update or insert new data into the database.

For example: Data Engineers & Developers who are familiar with the database, directly access the database using SQL queries rather than writing programming code for accessing the database again. Hence, they are termed as sophisticated users.

4.7 CASUAL USERS / TEMPORARY USERS:

Casual users, also known as temporary users, are the type of database users in DBMS who frequently or occasionally use the database services. Whenever these users try to access the database, they want all the information sorted in place. Casual/Temporary users have little knowledge about DBMS and each time they try to access the database, they require new information. For example: High-level management people are casual users who have little knowledge about DBMS and hence, they can access the database to either fill in new information or retrieve existing results.