GOA COLLEGE OF ENGINEERING

"Bhausaheb Bandodkar Technical Education Complex"

Assignment No: 2

1. List the types of Distributed Databases

A.

Distributed databases are used for horizontal scaling, and they are designed to meet the workload requirements without having to make changes in the database application or vertically scale a single machine.

Distributed databases resolve various issues, such as availability, fault tolerance, throughput, latency, scalability, and many other problems that can arise from using a single machine and a single database.

Distributed Database Types

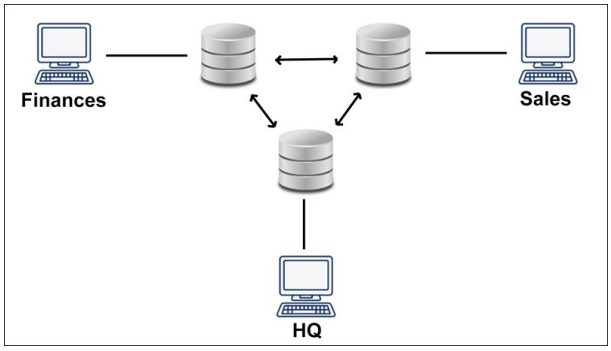
There are two types of distributed databases:

- Homogenous
- Heterogenous

Homogeneous

A homogenous distributed database is a network of identical databases stored on multiple sites. The sites have the same operating system, DDBMS, and data structure, making them easily manageable. Homogenous databases allow users to access data from each of the databases seamlessly.

The following diagram shows an example of a homogeneous database:

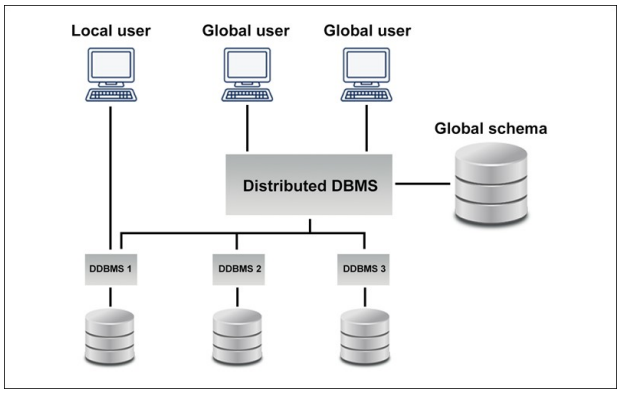


Heterogeneous

A heterogeneous distributed database uses different schemas, operating systems, DDBMS, and different data models.

In the case of a heterogeneous distributed database, a particular site can be completely unaware of other sites causing limited cooperation in processing user requests. The limitation is why translations are required to establish communication between sites.

The following diagram shows an example of a heterogeneous database:



2. Illustrate the architecture of a Distributed Database.

A.

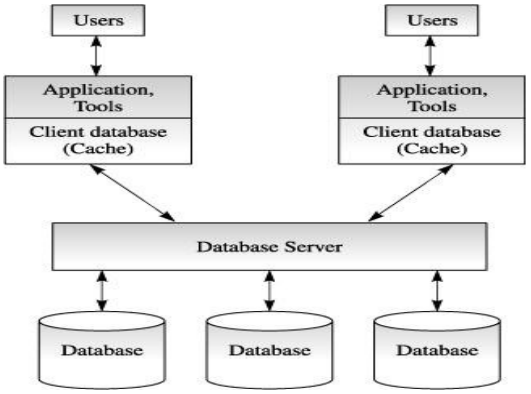
Following three architectures are used in distributed database systems:

- Client/server Architecture.
- Collaborating server system.
- Middleware system.

Client/Server Architecture

Client/server architectures are those in which a DBMS-related workload is split into two logical components namely client and server, each of which typically executes on different systems. Client is the user of the resource whereas the server is a provider of the resource. Client/server architecture has one or more client processes and one or more server processes. The applications and tools are put on one or more client platforms (generally, personal computers or workstations) and are connected to a database management system that resides on the server (typically a large workstation, midrange system, or a mainframe system). The applications and tools act as 'clients' of the DBMS, making requests for its services. The DBMS, in turn, services these requests and returns the results to the client(s). A client process can send a query to any one-server process. Clients are responsible for user-interface issues and servers manage data and execute transactions. In other words, the client/server architecture can be used to implement a DBMS in which the client is the transaction processor (TP) and the server is the data processor (DP). A client process could run on a personal computer and send queries to a server running on a mainframe computer. All modern information systems are based on client/server architecture of computing. Fig. 18.4 shows a schematic of client/server architecture.

Fig. 18.4. Client/server database architecture



Client/server architecture consists of the following main components:

- Clients in the form of intelligent workstations as the user's contact point.
- DBMS server as common resources performing specialised tasks for devices requesting their services.
- Communication networks connecting the clients and the servers.
- Software applications connecting clients, servers and networks to create a single logical architecture.

The client applications (which may be tools, vendor-written applications or user-written applications) issue SQL statements for data access, just as they do in a centralized computing environment. The networking interface enables client applications to connect to the server, send SQL statements and receive results or error return code after the server has processed the SQL statements. The applications themselves often make use of presentation services, such as graphic user interface, on the client.While writing client/server applications, it is important to remember the boundary between the client and the server and to keep the communication between them as set-oriented as possible. Application writing (programming) is most often done using a host language (for example, C, C++, COBOL and so on) with embedded data manipulation language (DML) statements (for example, SQL), which are communicated to the server.

Collaborating Server Systems

In collaborating server architecture, there are several database servers, each capable of running transactions against local data, which cooperatively execute transactions spanning multiple servers. When a server receives a query that requires access to data at other servers, it generates appropriate sub-queries to be executed by other servers and puts the results together to compute answers to the original query.

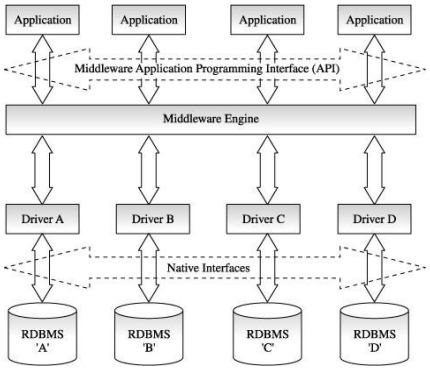
Middleware Systems

The middleware database architecture, also called data access middleware, is designed to allow a single query to span multiple servers, without requiring all database servers to be capable of managing such multicite execution strategies. Data access middleware provides users with a consistent interface to multiple DBMSs and file systems in a transparent manner. Data access middleware simplifies heterogeneous environments for programmers and provides users with an easier means of accessing live data in multiple sources. It eliminates the need for programmers to code many environment specific requests or calls in any application that needs access to current data rather than copies of such data. The direct request or calls for data movement to several DMSs are handled by the middleware, and hence a major rewrite of the application program is not required.

The middleware is basically a layer of software, which works as a special server and coordinates the execution of queries and transactions across one or more independent database servers. The middleware layer is capable of executing joins and other relational operations on data obtained from the other servers, but typically, does not itself maintain any data. Middleware provides an application with a consistent interface to some underlying services, shielding the application from different native interfaces and complexities required to execute the services. Middleware might be responsible for routing a local request to one or more remote servers, translating the request from one SQL dialect to another as needed, supporting various networking protocols, converting data from one for one format to another, coordinating work among various resource managers and performing other functions.

Fig. 18.5 illustrates sample data access middleware architecture. Data access middleware architecture consists of middleware application programming interface (API), middleware engine, drivers and native interfaces. The application programming interface (API) usually consists of a series of available function calls as well as a series of data access statements (dynamic SQL, QBE and so on). The middleware engine is basically an application programming interface for routing of requests to various drivers and performing other functions. It handles data access requests that have been issued. Drivers are used to connect various back-end data sources and they translate requests issued through the middleware API to a format intelligible to the target data source. Translation service may include SQL translation, data type translation and error messages and return code translation.

Fig. 18.5. Data access middleware architecture



Many data access middleware products have client/server architecture and access data residing on multiple remote systems. Therefore, networking interfaces may be provided between the client and the middleware, as well as between the middleware and data sources. Specific configurations of middleware vary from product to product. Some are largely client centric with the middleware engine and drivers residing on a client workstation or PC. Others are largely server centric with a small layer of software on the client provided to connect it into the remainder of the middleware solution, which resides primarily on a LAN server or host system.

3. Write a short note on Distributed database management system

A.

A distributed database management system (DDBMS) manages the database as if it were not all stored on the same computer. The DDBMS synchronizes all the data periodically and, in cases where multiple users must access the same data, ensures that updates and deletes performed on the data at one location will be automatically reflected in the data stored elsewhere. The users and administrators of a distributed system, should, with proper implementation, interact with the system as if the system was centralized. This transparency allows for the functionality desired in such a structured system without special programming requirements, allowing for any number of local and/or remote tables to be accessed at a given time across the network.

The different types of transparency sought after in a DDBMS are data distribution transparency, heterogeneity transparency, transaction transparency, and performance transparency.

- **Data distribution transparency** requires that the user of the database should not have to know how the data is fragmented (fragmentation transparency), know where the data they access is actually located (location transparency), or be aware of whether multiple copies of the data exist (replication transparency).

- **Heterogeneity transparency** requires that the user should not be aware of the fact that they are using a different DBMS if they access data from a remote site. The user should be able to use the same language that they would normally use at their regular access point and the DDBMS should handle query language translation if needed.

- **Transaction transparency** requires that the DDBMS guarantee that concurrent transactions do not interfere with each other (concurrency transparency) and that it must also handle database recovery (recovery transparency).

- **Performance transparency** mandates that the DDBMS should have a comparable level of performance to a centralized DBMS. Query optimizers can be used to speed up response time.

4. List advantages and disadvantages of DDBMS over centalised DBMS

A.

DDBMS Advantages and Disadvantages

Distributed database management systems deliver several advantages over traditional systems. That being said, they are subject to some problems.

Advantages of DDBMS's

- Reflects organizational structure
- Improved share ability
- Improved availability
- Improved reliability
- Improved performance
- Data are located nearest the greatest demand site and are dispersed to match business requirements.
- Faster Data Access because users only work with a locally stored subset of the data.
- Faster data processing because the data is processed at several different sites.
- Growth Facilitation: New sites can be added without compromising the operations of other sites.
- Improved communications because local sites are smaller and closer to customers.
- Reduced operating costs: It is more cost-effective to add workstations to a network rather than update a mainframe system.
- User Friendly interface equipped with an easy-to-use GUI.
- Less instances of single-point failure because data and workload are distributed among other workstations.

- Processor independence: The end user is able to access any available copy of data.

Disadvantages of DDBMS's

- Increased Cost
- Integrity control more difficult,
- Lack of standards,
- Database design is more complex.
- Complexity of management and control. Applications must recognize data location and they must be able to stitch together data from various sites.
- Technologically difficult: Data integrity, transaction management, concurrency control, security, backup, recovery, query optimization, access path selection are all issues that must be addressed and resolved
- Security lapses have increased instances when data are in multiple locations.
- Lack of standards due to the absence of communication protocols can make the processing and distribution of data difficult.
- Increased storage and infrastructure requirements because multiple copies of data are required at various separate locations which would require more disk space.
- Increased costs due to the higher complexity of training.
- Requires duplicate infrastructure (personnel, software and licensing, physical location/environment) and these can sometimes offset any operational savings.