

University of Technology
الجامعة التكنولوجية

Computer Science Department
قسم علوم الحاسوب

Distributed Database
Lect. Ragheed Dawood

م.د. رغيد داود سالم



cs.uotechnology.edu.iq

Distributed DBMS - Concepts

For proper functioning of any organization, there's a need for a well-maintained database. In the recent past, databases used to be centralized in nature. However, with the increase in globalization, organizations tend to be diversified across the globe. They may choose to distribute data over local servers instead of a central database. Thus, arrived the concept of **Distributed Databases**.

This chapter gives an overview of databases and Database Management Systems (DBMS). A database is an ordered collection of related data. A DBMS is a software package to work upon a database. A detailed study of DBMS is available in our tutorial named "Learn DBMS". In this chapter, we revise the main concepts so that the study of DDBMS can be done with ease. The three topics covered are database schemas, types of databases and operations on databases.

Database and Database Management System

A **database** is an ordered collection of related data that is built for a specific purpose. A database may be organized as a collection of multiple tables, where a table represents a real world element or entity. Each table has several different fields that represent the characteristic features of the entity.

For example, a company database may include tables for projects, employees, departments, products and financial records. The fields in the Employee table may be Name, Company_Id, Date_of_Joining, and so forth.

A **database management system** is a collection of programs that enables creation and maintenance of a database. DBMS is available as a software package that facilitates definition, construction, manipulation and sharing of data in a database. Definition of a database includes description of the structure of a database. Construction of a database involves actual storing of the data in any storage medium. Manipulation refers to the retrieving information from the database, updating the database and generating reports. Sharing of data facilitates data to be accessed by different users or programs.

Examples of DBMS Application Areas

- Automatic Teller Machines
- Train Reservation System

- Employee Management System
- Student Information System

Examples of DBMS Packages

- MySQL
- Oracle
- SQL Server
- dBASE
- FoxPro
- PostgreSQL, etc.

Database Schemas

A database schema is a description of the database which is specified during database design and subject to infrequent alterations. It defines the organization of the data, the relationships among them, and the constraints associated with them.

Databases are often represented through the **three-schema architecture** or **ANSISPARC architecture**. The goal of this architecture is to separate the user application from the physical database. The three levels are –

- **Internal Level having Internal Schema** – It describes the physical structure, details of internal storage and access paths for the database.
- **Conceptual Level having Conceptual Schema** – It describes the structure of the whole database while hiding the details of physical storage of data. This illustrates the entities, attributes with their data types and constraints, user operations and relationships.
- **External or View Level having External Schemas or Views** – It describes the portion of a database relevant to a particular user or a group of users while hiding the rest of database.

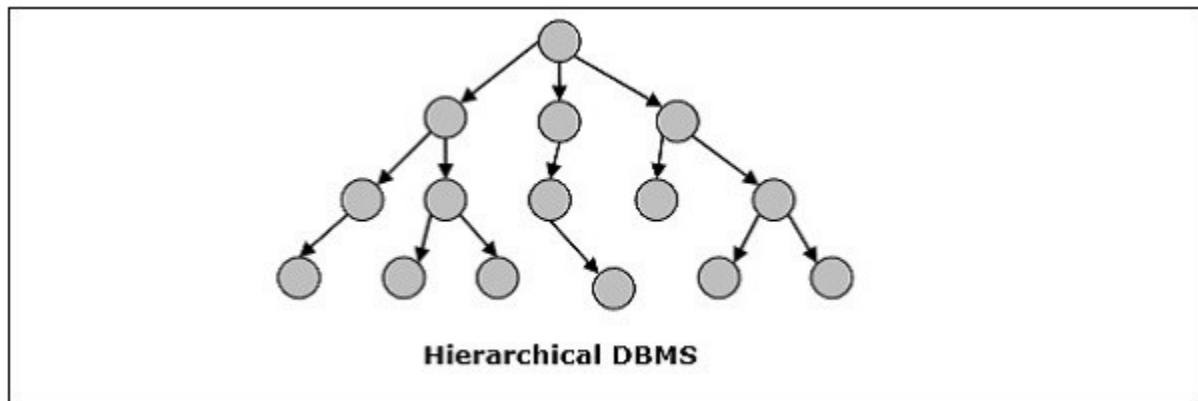
Types of DBMS

There are four types of DBMS.

Hierarchical DBMS

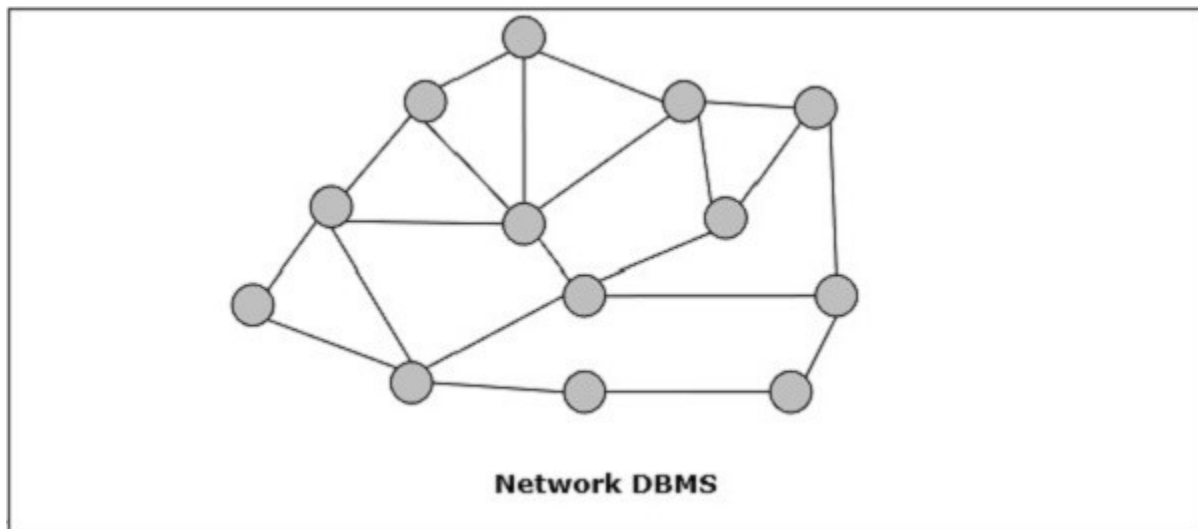
In hierarchical DBMS, the relationships among data in the database are established so that one data element exists as a subordinate of another. The data elements have

parent-child relationships and are modelled using the “tree” data structure. These are very fast and simple.



Network DBMS

Network DBMS is one where the relationships among data in the database are of type many-to-many in the form of a network. The structure is generally complicated due to the existence of numerous many-to-many relationships. Network DBMS is modelled using “graph” data structure.



Relational DBMS

In relational databases, the database is represented in the form of relations. Each relation models an entity and is represented as a table of values. In the relation or table, a row is called a tuple and denotes a single record. A column is called a field or an attribute and denotes a characteristic property of the entity. RDBMS is the most popular database management system.

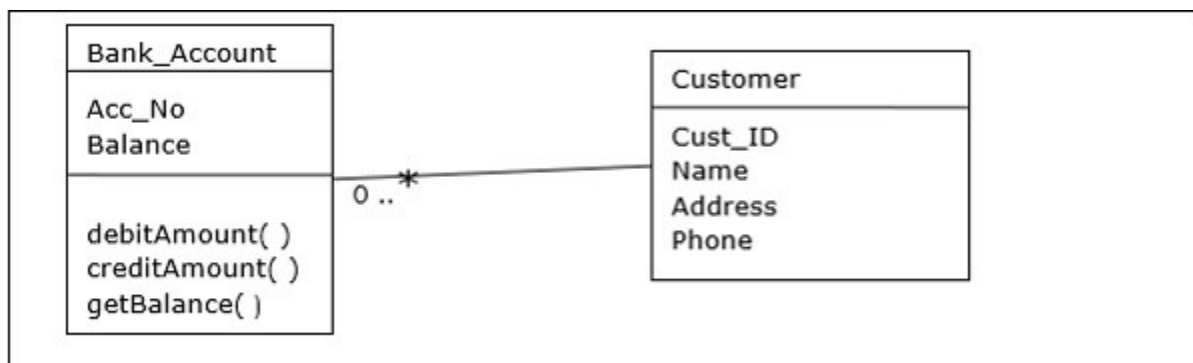
For example – A Student Relation –

	Field			
Tuple	S_Id	Name	Year	Stream
	1	Ankit Jha	1	Computer Science
	2	Pushpa Mishra	2	Electronics
	5	Ranjini Iyer	2	Computer Science

Object Oriented DBMS

Object-oriented DBMS is derived from the model of the object-oriented programming paradigm. They are helpful in representing both consistent data as stored in databases, as well as transient data, as found in executing programs. They use small, reusable elements called objects. Each object contains a data part and a set of operations which works upon the data. The object and its attributes are accessed through pointers instead of being stored in relational table models.

For example – A simplified Bank Account object-oriented database –



Distributed DBMS

A distributed database is a set of interconnected databases that is distributed over the computer network or internet. A Distributed Database Management System (DDBMS) manages the distributed database and provides mechanisms so as to make the databases transparent to the users. In these systems, data is intentionally distributed among multiple nodes so that all computing resources of the organization can be optimally used.

Operations on DBMS

The four basic operations on a database are Create, Retrieve, Update and Delete.

- **CREATE** database structure and populate it with data – Creation of a database relation involves specifying the data structures, data types and the constraints of the data to be stored.

Example – SQL command to create a student table –

```
CREATE TABLE STUDENT (  
  ROLL INTEGER PRIMARY KEY,  
  NAME VARCHAR2(25),  
  YEAR INTEGER,  
  STREAM VARCHAR2(10)  
);
```

- Once the data format is defined, the actual data is stored in accordance with the format in some storage medium.

Example SQL command to insert a single tuple into the student table –

```
INSERT INTO STUDENT ( ROLL, NAME, YEAR, STREAM)  
VALUES ( 1, 'ANKIT JHA', 1, 'COMPUTER SCIENCE');
```

- **RETRIEVE** information from the database – Retrieving information generally involves selecting a subset of a table or displaying data from the table after some computations have been done. It is done by querying upon the table.

Example – To retrieve the names of all students of the Computer Science stream, the following SQL query needs to be executed –

```
SELECT NAME FROM STUDENT  
WHERE STREAM = 'COMPUTER SCIENCE';
```

- **UPDATE** information stored and modify database structure – Updating a table involves changing old values in the existing table's rows with new values.

Example – SQL command to change stream from Electronics to Electronics and Communications –

```
UPDATE STUDENT  
SET STREAM = 'ELECTRONICS AND COMMUNICATIONS'  
WHERE STREAM = 'ELECTRONICS';
```

- Modifying database means to change the structure of the table. However, modification of the table is subject to a number of restrictions.

Example – To add a new field or column, say address to the Student table, we use the following SQL command –

```
ALTER TABLE STUDENT
```

```
ADD ( ADDRESS VARCHAR2(50) );
```

- **DELETE** information stored or delete a table as a whole – Deletion of specific information involves removal of selected rows from the table that satisfies certain conditions.

Example – To delete all students who are in 4th year currently when they are passing out, we use the SQL command –

```
DELETE FROM STUDENT  
WHERE YEAR = 4;
```

- Alternatively, the whole table may be removed from the database.

Example – To remove the student table completely, the SQL command used is –

```
DROP TABLE STUDENT;
```

Distributed DBMS - Distributed Databases

This chapter introduces the concept of DDBMS. In a distributed database, there are a number of databases that may be geographically distributed all over the world. A distributed DBMS manages the distributed database in a manner so that it appears as one single database to users. In the later part of the chapter, we go on to study the factors that lead to distributed databases, its advantages and disadvantages.

A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Features

- Databases in the collection are logically interrelated with each other. Often they represent a single logical database.
- Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites.
- The processors in the sites are connected via a network. They do not have any multiprocessor configuration.
- A distributed database is not a loosely connected file system.
- A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system.

Distributed Database Management System

A distributed database management system (DDBMS) is a centralized software system that manages a distributed database in a manner as if it were all stored in a single location.

Features

- It is used to create, retrieve, update and delete distributed databases.
- It synchronizes the database periodically and provides access mechanisms by the virtue of which the distribution becomes transparent to the users.
- It ensures that the data modified at any site is universally updated.
- It is used in application areas where large volumes of data are processed and accessed by numerous users simultaneously.
- It is designed for heterogeneous database platforms.
- It maintains confidentiality and data integrity of the databases.

Factors Encouraging DDBMS

The following factors encourage moving over to DDBMS –

- **Distributed Nature of Organizational Units** – Most organizations in the current times are subdivided into multiple units that are physically distributed over the globe. Each unit requires its own set of local data. Thus, the overall database of the organization becomes distributed.
- **Need for Sharing of Data** – The multiple organizational units often need to communicate with each other and share their data and resources. This demands common databases or replicated databases that should be used in a synchronized manner.
- **Support for Both OLTP and OLAP** – Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) work upon diversified systems which may have common data. Distributed database systems aid both these processing by providing synchronized data.
- **Database Recovery** – One of the common techniques used in DDBMS is replication of data across different sites. Replication of data automatically helps in data recovery if database in any site is damaged. Users can access data from other sites while the damaged site is being reconstructed. Thus, database failure may become almost inconspicuous to users.

- **Support for Multiple Application Software** – Most organizations use a variety of application software each with its specific database support. DDBMS provides a uniform functionality for using the same data among different platforms.

Advantages of Distributed Databases

Following are the advantages of distributed databases over centralized databases.

Modular Development – If the system needs to be expanded to new locations or new units, in centralized database systems, the action requires substantial efforts and disruption in the existing functioning. However, in distributed databases, the work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system, with no interruption in current functions.

More Reliable – In case of database failures, the total system of centralized databases comes to a halt. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.

Better Response – If data is distributed in an efficient manner, then user requests can be met from local data itself, thus providing faster response. On the other hand, in centralized systems, all queries have to pass through the central computer for processing, which increases the response time.

Lower Communication Cost – In distributed database systems, if data is located locally where it is mostly used, then the communication costs for data manipulation can be minimized. This is not feasible in centralized systems.

Disadvantages of Distributed Databases

Following are some of the adversities associated with distributed databases.

- **Need for complex and expensive software** – DDBMS demands complex and often expensive software to provide data transparency and co-ordination across the several sites.
- **Processing overhead** – Even simple operations may require a large number of communications and additional calculations to provide uniformity in data across the sites.

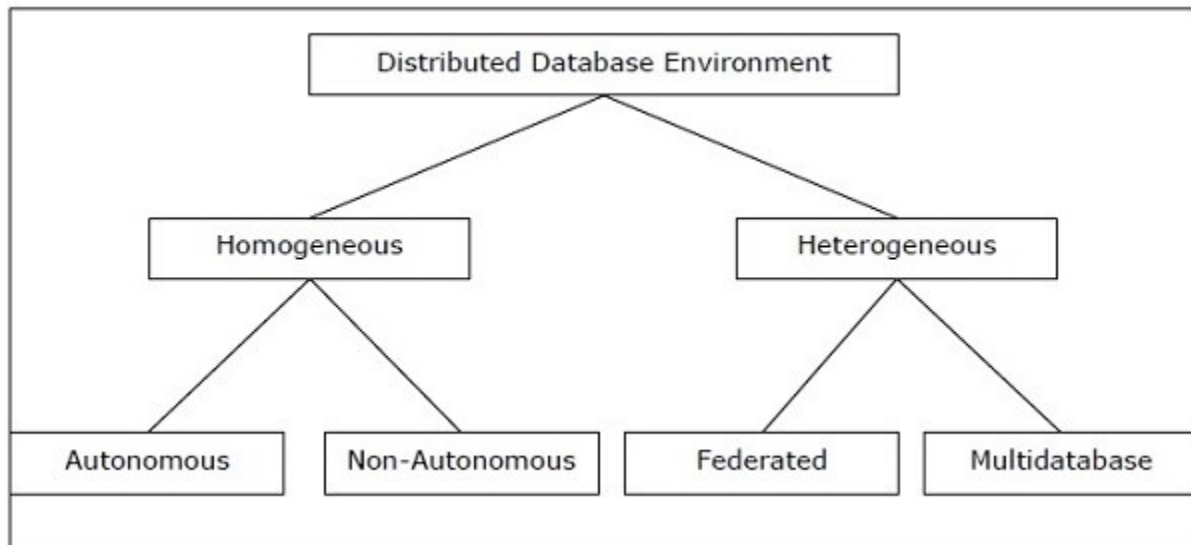
- **Data integrity** – The need for updating data in multiple sites pose problems of data integrity.
- **Overheads for improper data distribution** – Responsiveness of queries is largely dependent upon proper data distribution. Improper data distribution often leads to very slow response to user requests.

Distributed DBMS - Database Environments

In this part of the tutorial, we will study the different aspects that aid in designing distributed database environments. This chapter starts with the types of distributed databases. Distributed databases can be classified into homogeneous and heterogeneous databases having further divisions. The next section of this chapter discusses the distributed architectures namely client – server, peer – to – peer and multi – DBMS. Finally, the different design alternatives like replication and fragmentation are introduced.

Types of Distributed Databases

Distributed databases can be broadly classified into homogeneous and heterogeneous distributed database environments, each with further sub-divisions, as shown in the following illustration.



Homogeneous Distributed Databases

In a homogeneous distributed database, all the sites use identical DBMS and operating systems. Its properties are –

- The sites use very similar software.
- The sites use identical DBMS or DBMS from the same vendor.
- Each site is aware of all other sites and cooperates with other sites to process user requests.
- The database is accessed through a single interface as if it is a single database.

Types of Homogeneous Distributed Database

There are two types of homogeneous distributed database –

- **Autonomous** – Each database is independent that functions on its own. They are integrated by a controlling application and use message passing to share data updates.
- **Non-autonomous** – Data is distributed across the homogeneous nodes and a central or master DBMS co-ordinates data updates across the sites.

Heterogeneous Distributed Databases

In a heterogeneous distributed database, different sites have different operating systems, DBMS products and data models. Its properties are –

- Different sites use dissimilar schemas and software.
- The system may be composed of a variety of DBMSs like relational, network, hierarchical or object oriented.
- Query processing is complex due to dissimilar schemas.
- Transaction processing is complex due to dissimilar software.
- A site may not be aware of other sites and so there is limited co-operation in processing user requests.

Types of Heterogeneous Distributed Databases

- **Federated** – The heterogeneous database systems are independent in nature and integrated together so that they function as a single database system.
- **Un-federated** – The database systems employ a central coordinating module through which the databases are accessed.

Distributed DBMS Architectures

DDBMS architectures are generally developed depending on three parameters –

- **Distribution** – It states the physical distribution of data across the different sites.
- **Autonomy** – It indicates the distribution of control of the database system and the degree to which each constituent DBMS can operate independently.
- **Heterogeneity** – It refers to the uniformity or dissimilarity of the data models, system components and databases.

Architectural Models

Some of the common architectural models are –

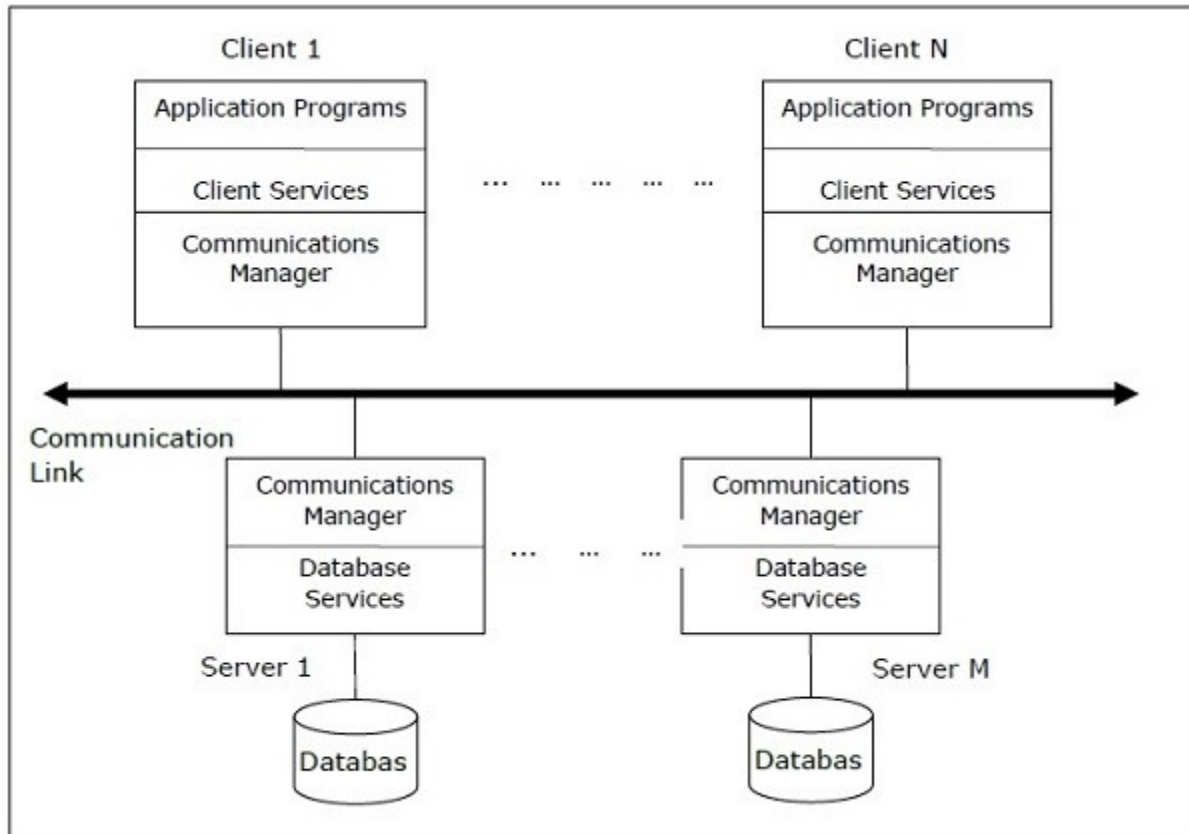
- Client - Server Architecture for DDBMS
- Peer - to - Peer Architecture for DDBMS
- Multi - DBMS Architecture

Client - Server Architecture for DDBMS

This is a two-level architecture where the functionality is divided into servers and clients. The server functions primarily encompass data management, query processing, optimization and transaction management. Client functions include mainly user interface. However, they have some functions like consistency checking and transaction management.

The two different client - server architecture are –

- Single Server Multiple Client
- Multiple Server Multiple Client (shown in the following diagram)

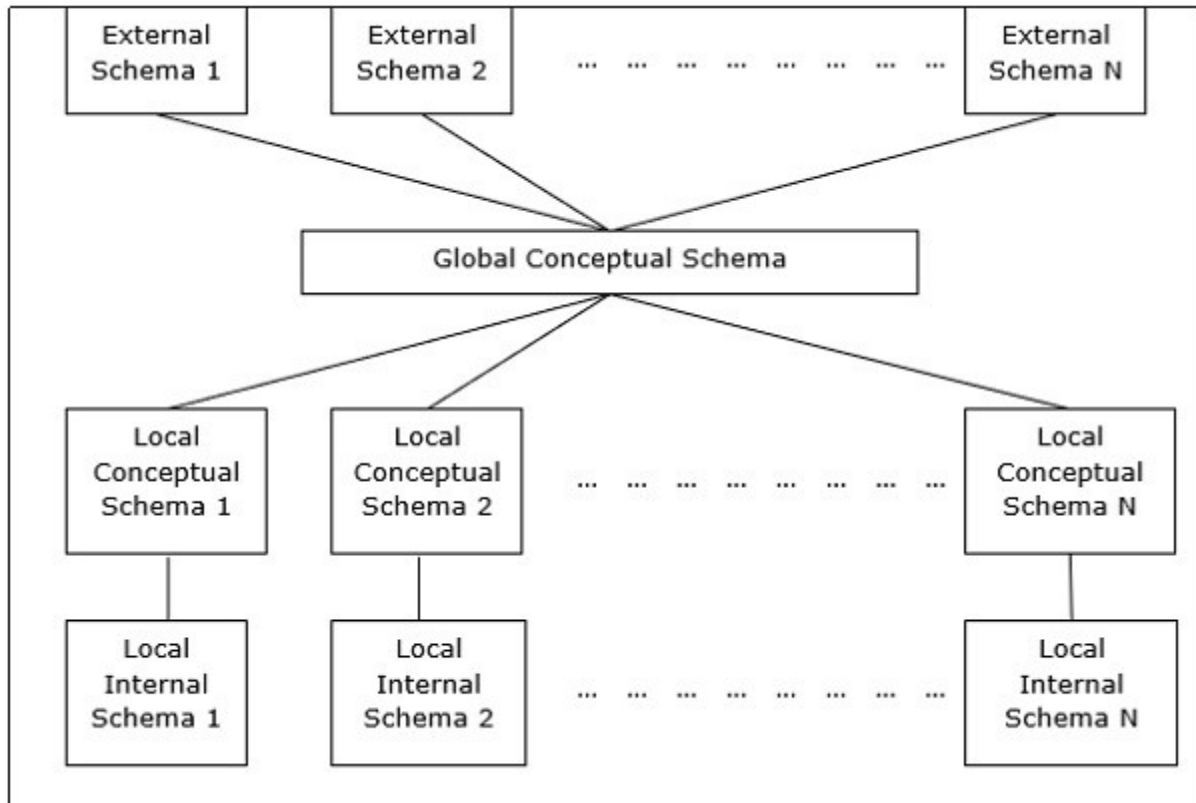


Peer- to-Peer Architecture for DDBMS

In these systems, each peer acts both as a client and a server for imparting database services. The peers share their resource with other peers and co-ordinate their activities.

This architecture generally has four levels of schemas –

- **Global Conceptual Schema** – Depicts the global logical view of data.
- **Local Conceptual Schema** – Depicts logical data organization at each site.
- **Local Internal Schema** – Depicts physical data organization at each site.
- **External Schema** – Depicts user view of data.



Multi - DBMS Architectures

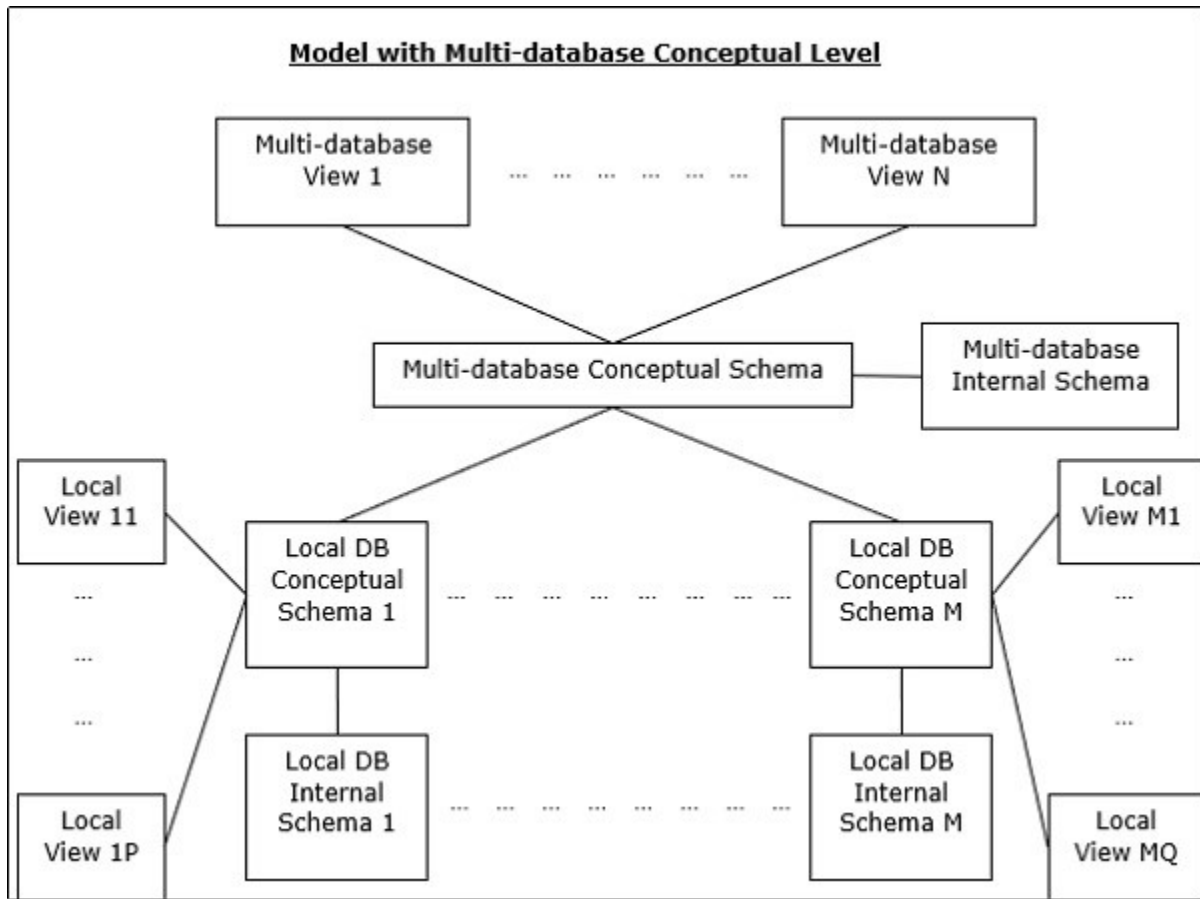
This is an integrated database system formed by a collection of two or more autonomous database systems.

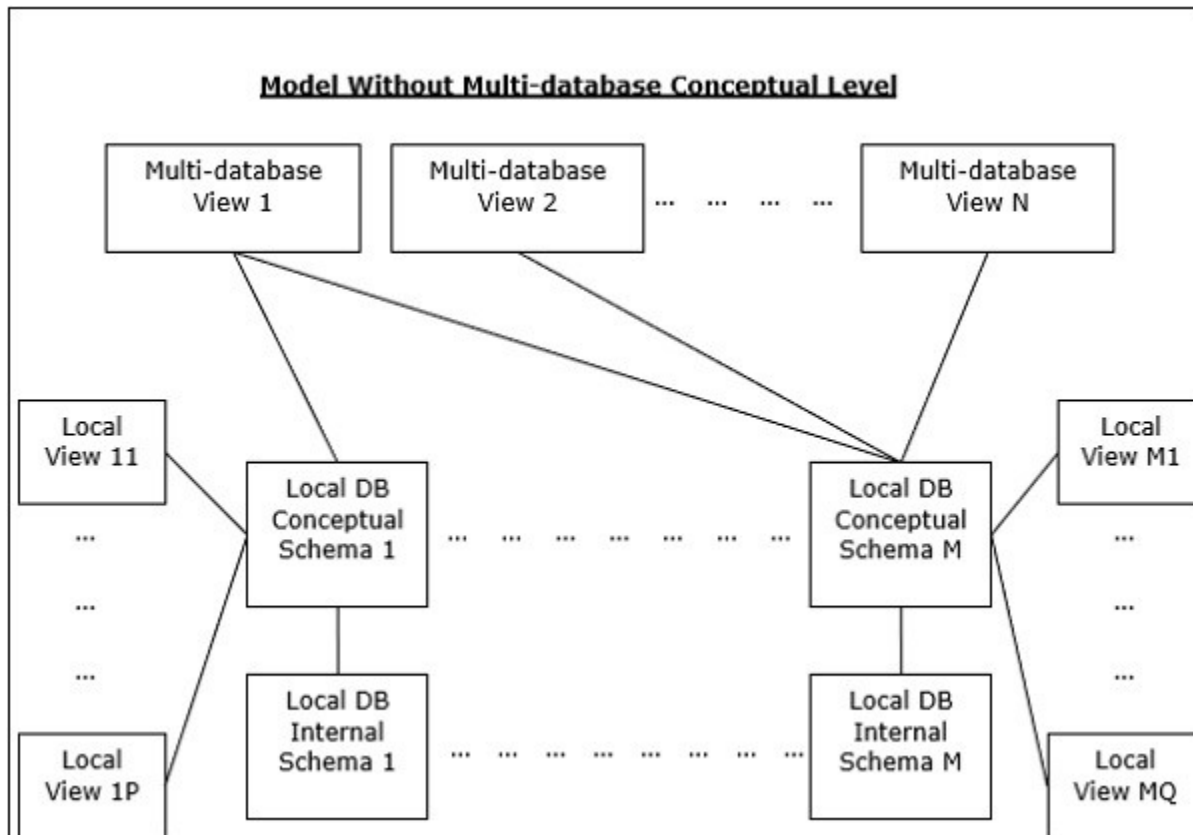
Multi-DBMS can be expressed through six levels of schemas –

- **Multi-database View Level** – Depicts multiple user views comprising of subsets of the integrated distributed database.
- **Multi-database Conceptual Level** – Depicts integrated multi-database that comprises of global logical multi-database structure definitions.
- **Multi-database Internal Level** – Depicts the data distribution across different sites and multi-database to local data mapping.
- **Local database View Level** – Depicts public view of local data.
- **Local database Conceptual Level** – Depicts local data organization at each site.
- **Local database Internal Level** – Depicts physical data organization at each site.

There are two design alternatives for multi-DBMS –

- Model with multi-database conceptual level.
- Model without multi-database conceptual level.





Design Alternatives

The distribution design alternatives for the tables in a DDBMS are as follows –

- Non-replicated and non-fragmented
- Fully replicated
- Partially replicated
- Fragmented
- Mixed

Non-replicated & Non-fragmented

In this design alternative, different tables are placed at different sites. Data is placed so that it is at a close proximity to the site where it is used most. It is most suitable for database systems where the percentage of queries needed to join information in tables placed at different sites is low. If an appropriate distribution strategy is adopted, then this design alternative helps to reduce the communication cost during data processing.

Fully Replicated

In this design alternative, at each site, one copy of all the database tables is stored. Since, each site has its own copy of the entire database, queries are very fast requiring negligible communication cost. On the contrary, the massive redundancy in data requires huge cost during update operations. Hence, this is suitable for systems where a large number of queries is required to be handled whereas the number of database updates is low.

Partially Replicated

Copies of tables or portions of tables are stored at different sites. The distribution of the tables is done in accordance to the frequency of access. This takes into consideration the fact that the frequency of accessing the tables vary considerably from site to site. The number of copies of the tables (or portions) depends on how frequently the access queries execute and the site which generate the access queries.

Fragmented

In this design, a table is divided into two or more pieces referred to as fragments or partitions, and each fragment can be stored at different sites. This considers the fact that it seldom happens that all data stored in a table is required at a given site. Moreover, fragmentation increases parallelism and provides better disaster recovery. Here, there is only one copy of each fragment in the system, i.e. no redundant data.

The three fragmentation techniques are –

- Vertical fragmentation
- Horizontal fragmentation
- Hybrid fragmentation

Mixed Distribution

This is a combination of fragmentation and partial replications. Here, the tables are initially fragmented in any form (horizontal or vertical), and then these fragments are partially replicated across the different sites according to the frequency of accessing the fragments.

Distributed DBMS - Design Strategies

In the last chapter, we had introduced different design alternatives. In this chapter, we will study the strategies that aid in adopting the designs. The strategies can be

broadly divided into replication and fragmentation. However, in most cases, a combination of the two is used.

Data Replication

Data replication is the process of storing separate copies of the database at two or more sites. It is a popular fault tolerance technique of distributed databases.

Advantages of Data Replication

- **Reliability** – In case of failure of any site, the database system continues to work since a copy is available at another site(s).
- **Reduction in Network Load** – Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.
- **Quicker Response** – Availability of local copies of data ensures quick query processing and consequently quick response time.
- **Simpler Transactions** – Transactions require less number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.

Disadvantages of Data Replication

- **Increased Storage Requirements** – Maintaining multiple copies of data is associated with increased storage costs. The storage space required is in multiples of the storage required for a centralized system.
- **Increased Cost and Complexity of Data Updating** – Each time a data item is updated, the update needs to be reflected in all the copies of the data at the different sites. This requires complex synchronization techniques and protocols.
- **Undesirable Application – Database coupling** – If complex update mechanisms are not used, removing data inconsistency requires complex coordination at application level. This results in undesirable application – database coupling.

Some commonly used replication techniques are –

- Snapshot replication
- Near-real-time replication
- Pull replication

Fragmentation

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called **fragments**. Fragmentation can be of three types: horizontal, vertical, and hybrid (combination of horizontal and vertical). Horizontal fragmentation can further be classified into two techniques: primary horizontal fragmentation and derived horizontal fragmentation.

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called “reconstructiveness.”

Advantages of Fragmentation

- Since data is stored close to the site of usage, efficiency of the database system is increased.
- Local query optimization techniques are sufficient for most queries since data is locally available.
- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

- When data from different fragments are required, the access speeds may be very low.
- In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
- Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

Vertical Fragmentation

In vertical fragmentation, the fields or columns of a table are grouped into fragments. In order to maintain reconstructiveness, each fragment should contain the primary key field(s) of the table. Vertical fragmentation can be used to enforce privacy of data.

For example, let us consider that a University database keeps records of all registered students in a Student table having the following schema.

STUDENT

Regd_No	Name	Course	Address	Semester	Fees	Marks
---------	------	--------	---------	----------	------	-------

Now, the fees details are maintained in the accounts section. In this case, the designer will fragment the database as follows –

```
CREATE TABLE STD_FEES AS  
SELECT Regd_No, Fees  
FROM STUDENT;
```

Horizontal Fragmentation

Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also conform to the rule of reconstructiveness. Each horizontal fragment must have all columns of the original base table.

For example, in the student schema, if the details of all students of Computer Science Course needs to be maintained at the School of Computer Science, then the designer will horizontally fragment the database as follows –

```
CREATE COMP_STD AS  
SELECT * FROM STUDENT  
WHERE COURSE = "Computer Science";
```

Hybrid Fragmentation

In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task.

Hybrid fragmentation can be done in two alternative ways –

- At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
- At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

DDBMS - Distribution Transparency

Distribution transparency is the property of distributed databases by the virtue of which the internal details of the distribution are hidden from the users. The DDBMS designer may choose to fragment tables, replicate the fragments and store them at different sites. However, since users are oblivious of these details, they find the distributed database easy to use like any centralized database.

The three dimensions of distribution transparency are –

- Location transparency
- Fragmentation transparency
- Replication transparency

Location Transparency

Location transparency ensures that the user can query on any table(s) or fragment(s) of a table as if they were stored locally in the user's site. The fact that the table or its fragments are stored at remote site in the distributed database system, should be completely oblivious to the end user. The address of the remote site(s) and the access mechanisms are completely hidden.

In order to incorporate location transparency, DDBMS should have access to updated and accurate data dictionary and DDBMS directory which contains the details of locations of data.

Fragmentation Transparency

Fragmentation transparency enables users to query upon any table as if it were unfragmented. Thus, it hides the fact that the table the user is querying on is actually a fragment or union of some fragments. It also conceals the fact that the fragments are located at diverse sites.

This is somewhat similar to users of SQL views, where the user may not know that they are using a view of a table instead of the table itself.

Replication Transparency

Replication transparency ensures that replication of databases are hidden from the users. It enables users to query upon a table as if only a single copy of the table exists.

Replication transparency is associated with concurrency transparency and failure transparency. Whenever a user updates a data item, the update is reflected in all the copies of the table. However, this operation should not be known to the user. This is concurrency transparency. Also, in case of failure of a site, the user can still proceed with his queries using replicated copies without any knowledge of failure. This is failure transparency.

Combination of Transparencies

In any distributed database system, the designer should ensure that all the stated transparencies are maintained to a considerable extent. The designer may choose to fragment tables, replicate them and store them at different sites; all oblivious to the end user. However, complete distribution transparency is a tough task and requires considerable design efforts.

Distributed DBMS - Database Control

Database control refers to the task of enforcing regulations so as to provide correct data to authentic users and applications of a database. In order that correct data is available to users, all data should conform to the integrity constraints defined in the database. Besides, data should be screened away from unauthorized users so as to maintain security and privacy of the database. Database control is one of the primary tasks of the database administrator (DBA).

The three dimensions of database control are –

- Authentication
- Access rights
- Integrity constraints

Authentication

In a distributed database system, authentication is the process through which only legitimate users can gain access to the data resources.

Authentication can be enforced in two levels –

- **Controlling Access to Client Computer** – At this level, user access is restricted while login to the client computer that provides user-interface to the database server. The most common method is a username/password combination. However, more sophisticated methods like biometric authentication may be used for high security data.
- **Controlling Access to the Database Software** – At this level, the database software/administrator assigns some credentials to the user. The user gains access to the database using these credentials. One of the methods is to create a login account within the database server.

Access Rights

A user's access rights refers to the privileges that the user is given regarding DBMS operations such as the rights to create a table, drop a table, add/delete/update tuples in a table or query upon the table.

In distributed environments, since there are large number of tables and yet larger number of users, it is not feasible to assign individual access rights to users. So, DDBMS defines certain roles. A role is a construct with certain privileges within a database system. Once the different roles are defined, the individual users are assigned one of these roles. Often a hierarchy of roles are defined according to the organization's hierarchy of authority and responsibility.

For example, the following SQL statements create a role "Accountant" and then assigns this role to user "ABC".

```
CREATE ROLE ACCOUNTANT;  
GRANT SELECT, INSERT, UPDATE ON EMP_SAL TO ACCOUNTANT;  
GRANT INSERT, UPDATE, DELETE ON TENDER TO ACCOUNTANT;  
GRANT INSERT, SELECT ON EXPENSE TO ACCOUNTANT;  
COMMIT;  
GRANT ACCOUNTANT TO ABC;  
COMMIT;
```

Semantic Integrity Control

Semantic integrity control defines and enforces the integrity constraints of the database system.

The integrity constraints are as follows –

- Data type integrity constraint
- Entity integrity constraint
- Referential integrity constraint

Data Type Integrity Constraint

A data type constraint restricts the range of values and the type of operations that can be applied to the field with the specified data type.

For example, let us consider that a table "HOSTEL" has three fields - the hostel number, hostel name and capacity. The hostel number should start with capital letter "H" and cannot be NULL, and the capacity should not be more than 150. The following SQL command can be used for data definition –

```
CREATE TABLE HOSTEL (  
  H_NO VARCHAR2(5) NOT NULL,  
  H_NAME VARCHAR2(15),  
  CAPACITY INTEGER,  
  CHECK ( H_NO LIKE 'H%'),  
  CHECK ( CAPACITY <= 150)  
);
```

Entity Integrity Control

Entity integrity control enforces the rules so that each tuple can be uniquely identified from other tuples. For this a primary key is defined. A primary key is a set of minimal fields that can uniquely identify a tuple. Entity integrity constraint states that no two tuples in a table can have identical values for primary keys and that no field which is a part of the primary key can have NULL value.

For example, in the above hostel table, the hostel number can be assigned as the primary key through the following SQL statement (ignoring the checks) –

```
CREATE TABLE HOSTEL (  
  H_NO VARCHAR2(5) PRIMARY KEY,  
  H_NAME VARCHAR2(15),
```



```
CAPACITY INTEGER
```

```
);
```

Referential Integrity Constraint

Referential integrity constraint lays down the rules of foreign keys. A foreign key is a field in a data table that is the primary key of a related table. The referential integrity constraint lays down the rule that the value of the foreign key field should either be among the values of the primary key of the referenced table or be entirely NULL.

For example, let us consider a student table where a student may opt to live in a hostel. To include this, the primary key of hostel table should be included as a foreign key in the student table. The following SQL statement incorporates this –

```
CREATE TABLE STUDENT (  
  S_ROLL INTEGER PRIMARY KEY,  
  S_NAME VARCHAR2(25) NOT NULL,  
  S_COURSE VARCHAR2(10),  
  S_HOSTEL VARCHAR2(5) REFERENCES HOSTEL  
);
```

Relational Algebra for Query Optimization

When a query is placed, it is at first scanned, parsed and validated. An internal representation of the query is then created such as a query tree or a query graph. Then alternative execution strategies are devised for retrieving results from the database tables. The process of choosing the most appropriate execution strategy for query processing is called query optimization.

Query Optimization Issues in DDBMS

In DDBMS, query optimization is a crucial task. The complexity is high since number of alternative strategies may increase exponentially due to the following factors –

- The presence of a number of fragments.

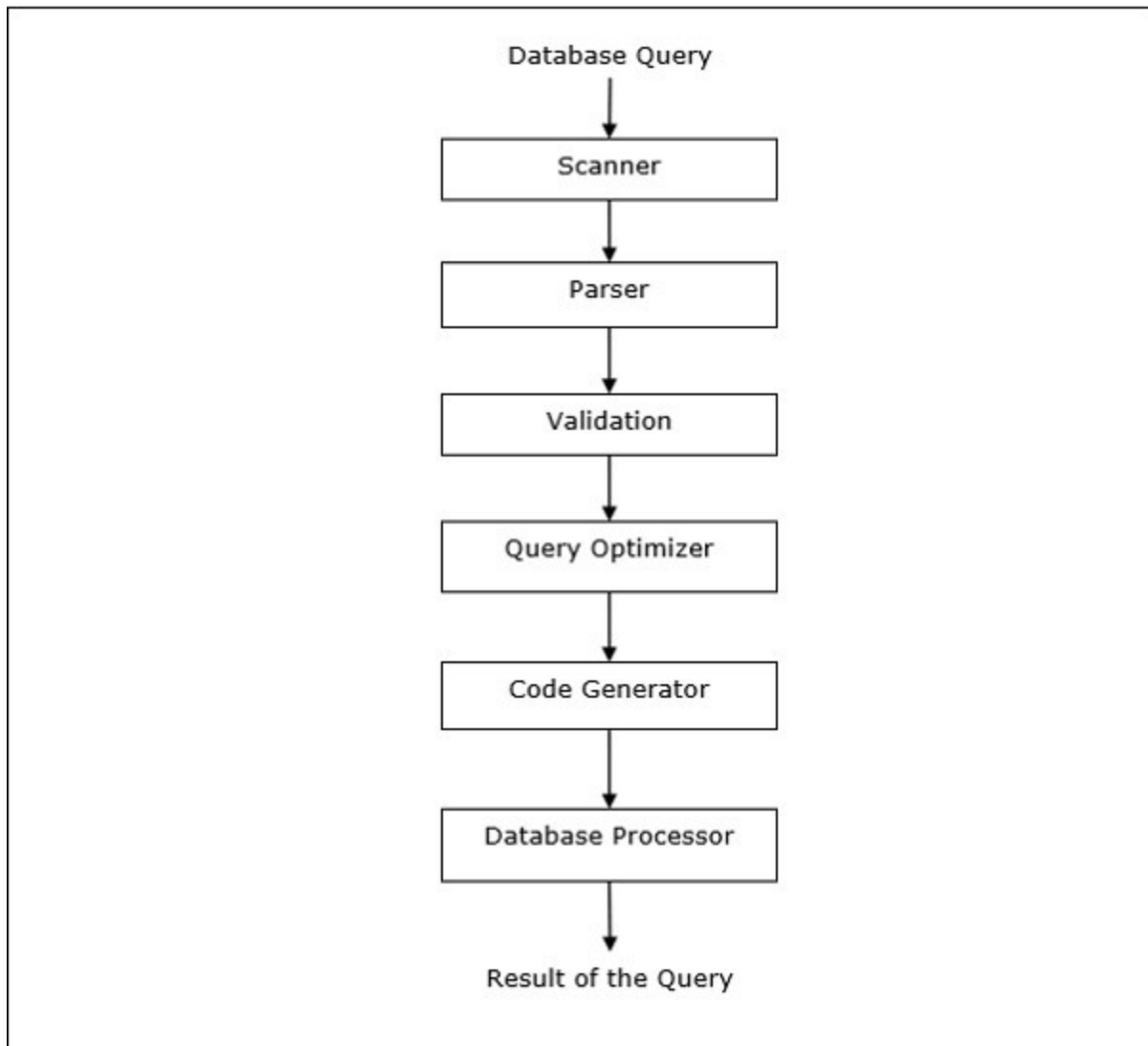
- Distribution of the fragments or tables across various sites.
- The speed of communication links.
- Disparity in local processing capabilities.

Hence, in a distributed system, the target is often to find a good execution strategy for query processing rather than the best one. The time to execute a query is the sum of the following –

- Time to communicate queries to databases.
- Time to execute local query fragments.
- Time to assemble data from different sites.
- Time to display results to the application.

Query Processing

Query processing is a set of all activities starting from query placement to displaying the results of the query. The steps are as shown in the following diagram –



Relational Algebra

Relational algebra defines the basic set of operations of relational database model. A sequence of relational algebra operations forms a relational algebra expression. The result of this expression represents the result of a database query.

The basic operations are –

- Projection
- Selection
- Union
- Intersection
- Minus

- Join

Projection

Projection operation displays a subset of fields of a table. This gives a vertical partition of the table.

Syntax in Relational Algebra

$$\pi\langle\text{AttributeList}\rangle(\langle\text{TableName}\rangle)\pi\langle\text{AttributeList}\rangle(\langle\text{TableName}\rangle)$$

For example, let us consider the following Student database –

STUDENT				
Roll_No	Name	Course	Semester	Gender
2	Amit Prasad	BCA	1	Male
4	Varsha Tiwari	BCA	1	Female
5	Asif Ali	MCA	2	Male
6	Joe Wallace	MCA	1	Male
8	Shivani Iyengar	BCA	1	Female

If we want to display the names and courses of all students, we will use the following relational algebra expression –

$$\pi\text{Name,Course}(\text{STUDENT})\pi\text{Name,Course}(\text{STUDENT})$$

Selection

Selection operation displays a subset of tuples of a table that satisfies certain conditions. This gives a horizontal partition of the table.

Syntax in Relational Algebra

$$\sigma\langle\text{Conditions}\rangle(\langle\text{TableName}\rangle)\sigma\langle\text{Conditions}\rangle(\langle\text{TableName}\rangle)$$

For example, in the Student table, if we want to display the details of all students who have opted for MCA course, we will use the following relational algebra expression –

$\sigma_{\text{Course}=\text{"BCA"}}(\text{STUDENT})$

Combination of Projection and Selection Operations

For most queries, we need a combination of projection and selection operations. There are two ways to write these expressions –

- Using sequence of projection and selection operations.
- Using rename operation to generate intermediate results.

For example, to display names of all female students of the BCA course –

- Relational algebra expression using sequence of projection and selection operations

$\pi_{\text{Name}}(\sigma_{\text{Gender}=\text{"Female"} \wedge \text{Course}=\text{"BCA"}}(\text{STUDENT}))$
 $\pi_{\text{Name}}(\sigma_{\text{Gender}=\text{"Female"} \wedge \text{Course}=\text{"BCA"}}(\text{STUDENT}))$

- Relational algebra expression using rename operation to generate intermediate results

$\text{FemaleBCAStudent} \leftarrow \sigma_{\text{Gender}=\text{"Female"} \wedge \text{Course}=\text{"BCA"}}(\text{STUDENT})$
 $\text{FemaleBCAStudent} \leftarrow \sigma_{\text{Gender}=\text{"Female"} \wedge \text{Course}=\text{"BCA"}}(\text{STUDENT})$
 $\text{Result} \leftarrow \pi_{\text{Name}}(\text{FemaleBCAStudent})$
 $\text{Result} \leftarrow \pi_{\text{Name}}(\text{FemaleBCAStudent})$

Union

If P is a result of an operation and Q is a result of another operation, the union of P and Q ($P \cup Q$) is the set of all tuples that is either in P or in Q or in both without duplicates.

For example, to display all students who are either in Semester 1 or are in BCA course –

$\text{Sem1Student} \leftarrow \sigma_{\text{Semester}=1}(\text{STUDENT})$
 $\text{Sem1Student} \leftarrow \sigma_{\text{Semester}=1}(\text{STUDENT})$
 $\text{BCAStudent} \leftarrow \sigma_{\text{Course}=\text{"BCA"}}(\text{STUDENT})$
 $\text{BCAStudent} \leftarrow \sigma_{\text{Course}=\text{"BCA"}}(\text{STUDENT})$
 $\text{Result} \leftarrow \text{Sem1Student} \cup \text{BCAStudent}$
 $\text{Result} \leftarrow \text{Sem1Student} \cup \text{BCAStudent}$

Intersection

If P is a result of an operation and Q is a result of another operation, the intersection of P and Q ($P \cap Q$) is the set of all tuples that are in P and Q both.

For example, given the following two schemas –

EMPLOYEE

EmpID	Name	City	Department	Salary
-------	------	------	------------	--------

PROJECT

PIId	City	Department	Status
------	------	------------	--------

To display the names of all cities where a project is located and also an employee resides –

$CityEmp \leftarrow \pi_{City}(EMPLOYEE)$

$CityProject \leftarrow \pi_{City}(PROJECT)$

$Result \leftarrow CityEmp \cap CityProject$

Minus

If P is a result of an operation and Q is a result of another operation, $P - Q$ is the set of all tuples that are in P and not in Q.

For example, to list all the departments which do not have an ongoing project (projects with status = ongoing) –

$AllDept \leftarrow \pi_{Department}(EMPLOYEE)$

$ProjectDept \leftarrow \pi_{Department}(\sigma_{Status="ongoing"}(PROJECT))$

$Result \leftarrow AllDept - ProjectDept$

Join

Join operation combines related tuples of two different tables (results of queries) into a single table.

For example, consider two schemas, Customer and Branch in a Bank database as follows –

CUSTOMER

CustID	AccNo	TypeOfAc	BranchID	DateOfOpening
--------	-------	----------	----------	---------------

BRANCH

BranchID	BranchName	IFSCcode	Address
----------	------------	----------	---------

To list the employee details along with branch details –

```
Result←CUSTOMER⋈Customer.BranchID=Branch.BranchIDBRANCHResult←  
CUSTOMER⋈Customer.BranchID=Branch.BranchIDBRANCH
```

Translating SQL Queries into Relational Algebra

SQL queries are translated into equivalent relational algebra expressions before optimization. A query is at first decomposed into smaller query blocks. These blocks are translated to equivalent relational algebra expressions. Optimization includes optimization of each block and then optimization of the query as a whole.

Examples

Let us consider the following schemas –

EMPLOYEE

EmpID	Name	City	Department	Salary
-------	------	------	------------	--------

PROJECT

PIId	City	Department	Status
------	------	------------	--------

WORKS

EmpID	PID	Hours
-------	-----	-------

Example 1

To display the details of all employees who earn a salary LESS than the average salary, we write the SQL query –

```
SELECT * FROM EMPLOYEE
```

```
WHERE SALARY < ( SELECT AVERAGE(SALARY) FROM EMPLOYEE );
```

This query contains one nested sub-query. So, this can be broken down into two blocks.

The inner block is –

```
SELECT AVERAGE(SALARY)FROM EMPLOYEE ;
```

If the result of this query is AvgSal, then outer block is –

```
SELECT * FROM EMPLOYEE WHERE SALARY < AvgSal;
```

Relational algebra expression for inner block –

```
AvgSal ←  $\pi$  AVERAGE(Salary)EMPLOYEE  
AvgSal ←  $\exists$  AVERAGE(Salary)EMPL  
OYEE
```

Relational algebra expression for outer block –

```
 $\sigma$ Salary < AvgSal > EMPLOYEE  $\sigma$ Salary < AvgSal > EMPLOYEE
```

Example 2

To display the project ID and status of all projects of employee 'Arun Kumar', we write the SQL query –

```
SELECT PID, STATUS FROM PROJECT  
WHERE PID = ( SELECT FROM WORKS WHERE EMPID = ( SELECT  
EMPID FROM EMPLOYEE  
WHERE NAME = 'ARUN KUMAR'));
```

This query contains two nested sub-queries. Thus, can be broken down into three blocks, as follows –

```
SELECT EMPID FROM EMPLOYEE WHERE NAME = 'ARUN KUMAR';  
SELECT PID FROM WORKS WHERE EMPID = ArunEmpID;  
SELECT PID, STATUS FROM PROJECT WHERE PID = ArunPID;
```

(Here ArunEmpID and ArunPID are the results of inner queries)

Relational algebra expressions for the three blocks are –


```

ArunEmpID←πEmpID(σName="ArunKumar"(EMPLOYEE))ArunEmpID←πEmpID(σName="ArunKumar"(EMPLOYEE))
ArunPID←πPID(σEmpID="ArunEmpID"(WORKS))ArunPID←πPID(σEmpID="ArunEmpID"(WORKS))
Result←πPID,Status(σPID="ArunPID"(PROJECT))Result←πPID,Status(σPID="ArunPID"(PROJECT))

```

Computation of Relational Algebra Operators

The computation of relational algebra operators can be done in many different ways, and each alternative is called an **access path**.

The computation alternative depends upon three main factors –

- Operator type
- Available memory
- Disk structures

The time to perform execution of a relational algebra operation is the sum of –

- Time to process the tuples.
- Time to fetch the tuples of the table from disk to memory.

Since the time to process a tuple is very much smaller than the time to fetch the tuple from the storage, particularly in a distributed system, disk access is very often considered as the metric for calculating cost of relational expression.

Computation of Selection

Computation of selection operation depends upon the complexity of the selection condition and the availability of indexes on the attributes of the table.

Following are the computation alternatives depending upon the indexes –

- **No Index** – If the table is unsorted and has no indexes, then the selection process involves scanning all the disk blocks of the table. Each block is brought into the memory and each tuple in the block is examined to see whether it satisfies the selection condition. If the condition is satisfied, it is displayed as output. This is the costliest approach since each tuple is brought into memory and each tuple is processed.

- **B+ Tree Index** – Most database systems are built upon the B+ Tree index. If the selection condition is based upon the field, which is the key of this B+ Tree index, then this index is used for retrieving results. However, processing selection statements with complex conditions may involve a larger number of disk block accesses and in some cases complete scanning of the table.
- **Hash Index** – If hash indexes are used and its key field is used in the selection condition, then retrieving tuples using the hash index becomes a simple process. A hash index uses a hash function to find the address of a bucket where the key value corresponding to the hash value is stored. In order to find a key value in the index, the hash function is executed and the bucket address is found. The key values in the bucket are searched. If a match is found, the actual tuple is fetched from the disk block into the memory.

Computation of Joins

When we want to join two tables, say P and Q, each tuple in P has to be compared with each tuple in Q to test if the join condition is satisfied. If the condition is satisfied, the corresponding tuples are concatenated, eliminating duplicate fields and appended to the result relation. Consequently, this is the most expensive operation.

The common approaches for computing joins are –

Nested-loop Approach

This is the conventional join approach. It can be illustrated through the following pseudocode (Tables P and Q, with tuples tuple_p and tuple_q and joining attribute a) –

```

For each tuple_p in P
  For each tuple_q in Q
    If tuple_p.a = tuple_q.a Then
      Concatenate tuple_p and tuple_q and append to Result
    End If
  Next tuple_q
Next tuple_p

```

Sort-merge Approach

In this approach, the two tables are individually sorted based upon the joining attribute and then the sorted tables are merged. External sorting techniques are adopted since the number of records is very high and cannot be accommodated in the memory. Once the individual tables are sorted, one page each of the sorted tables are brought to the memory, merged based upon the joining attribute and the joined tuples are written out.

Hash-join Approach

This approach comprises of two phases: partitioning phase and probing phase. In partitioning phase, the tables P and Q are broken into two sets of disjoint partitions. A common hash function is decided upon. This hash function is used to assign tuples to partitions. In the probing phase, tuples in a partition of P are compared with the tuples of corresponding partition of Q. If they match, then they are written out.

Query Optimization in Centralized Systems

Once the alternative access paths for computation of a relational algebra expression are derived, the optimal access path is determined. In this chapter, we will look into query optimization in centralized system while in the next chapter we will study query optimization in a distributed system.

In a centralized system, query processing is done with the following aim –

- Minimization of response time of query (time taken to produce the results to user's query).
- Maximize system throughput (the number of requests that are processed in a given amount of time).
- Reduce the amount of memory and storage required for processing.
- Increase parallelism.

Query Parsing and Translation

Initially, the SQL query is scanned. Then it is parsed to look for syntactical errors and correctness of data types. If the query passes this step, the query is decomposed into smaller query blocks. Each block is then translated to equivalent relational algebra expression.

Steps for Query Optimization

Query optimization involves three steps, namely query tree generation, plan generation, and query plan code generation.

Step 1 – Query Tree Generation

A query tree is a tree data structure representing a relational algebra expression. The tables of the query are represented as leaf nodes. The relational algebra operations are represented as the internal nodes. The root represents the query as a whole.

During execution, an internal node is executed whenever its operand tables are available. The node is then replaced by the result table. This process continues for all internal nodes until the root node is executed and replaced by the result table.

For example, let us consider the following schemas –

EMPLOYEE

EmpID	EName	Salary	DeptNo	DateOfJoining
-------	-------	--------	--------	---------------

DEPARTMENT

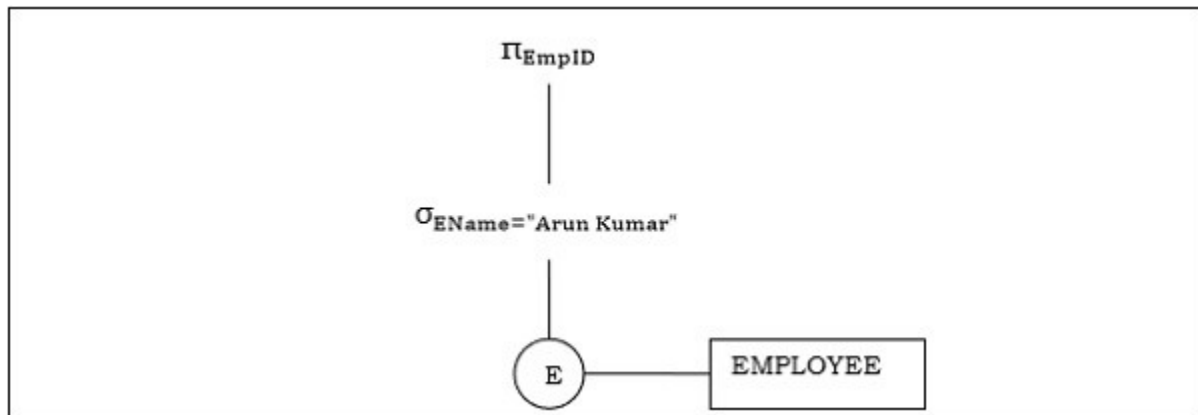
DNo	DName	Location
-----	-------	----------

Example 1

Let us consider the query as the following.

$\pi_{EmpID}(\sigma_{EName="ArunKumar"}(EMPLOYEE)) \pi_{EmpID}(\sigma_{EName="ArunKumar"}(EMPLOYEE))$

The corresponding query tree will be –

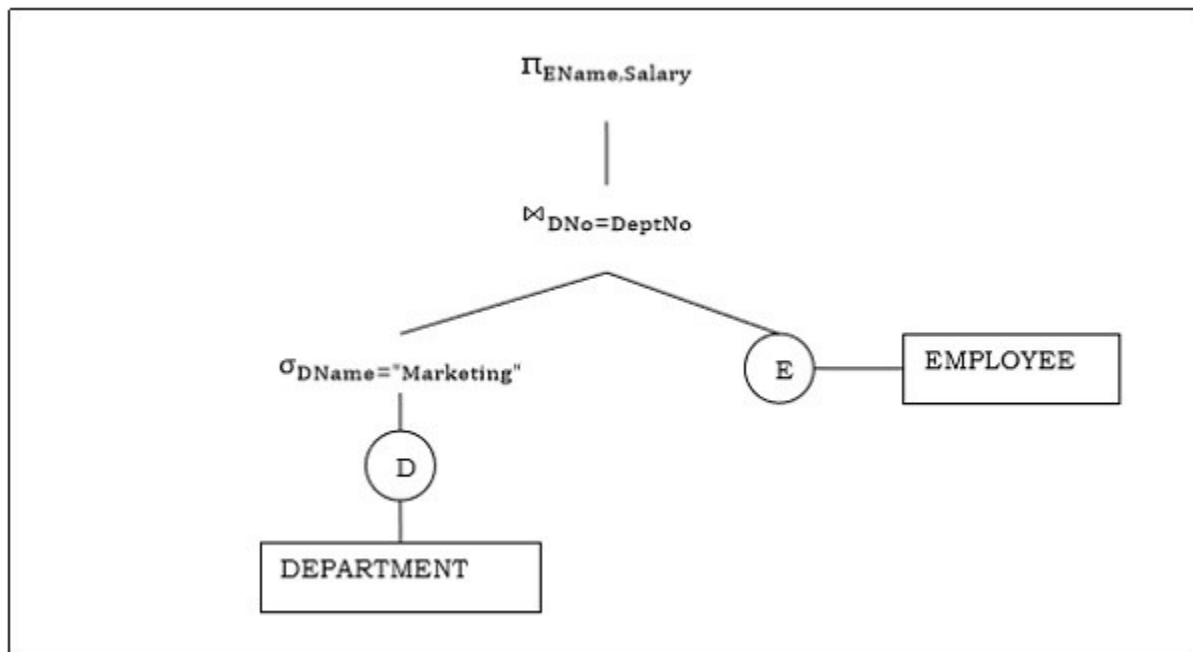


Example 2

Let us consider another query involving a join.

$\pi_{ENAME,Salary}(\sigma_{DName="Marketing"}(DEPARTMENT)) \bowtie_{DNo=DeptNo} (\pi_{ENAME,Salary}(\sigma_{DName="Marketing"}(DEPARTMENT)) \bowtie_{DNo=DeptNo} (EMPLOYEE))$

Following is the query tree for the above query.



Step 2 – Query Plan Generation

After the query tree is generated, a query plan is made. A query plan is an extended query tree that includes access paths for all operations in the query tree. Access paths specify how the relational operations in the tree should be performed. For example, a selection operation can have an access path that gives details about the use of B+ tree index for selection.

Besides, a query plan also states how the intermediate tables should be passed from one operator to the next, how temporary tables should be used and how operations should be pipelined/combined.

Step 3– Code Generation

Code generation is the final step in query optimization. It is the executable form of the query, whose form depends upon the type of the underlying operating system.

Once the query code is generated, the Execution Manager runs it and produces the results.

Approaches to Query Optimization

Among the approaches for query optimization, exhaustive search and heuristics-based algorithms are mostly used.

Exhaustive Search Optimization

In these techniques, for a query, all possible query plans are initially generated and then the best plan is selected. Though these techniques provide the best solution, it has an exponential time and space complexity owing to the large solution space. For example, dynamic programming technique.

Heuristic Based Optimization

Heuristic based optimization uses rule-based optimization approaches for query optimization. These algorithms have polynomial time and space complexity, which is lower than the exponential complexity of exhaustive search-based algorithms. However, these algorithms do not necessarily produce the best query plan.

Some of the common heuristic rules are –

- Perform select and project operations before join operations. This is done by moving the select and project operations down the query tree. This reduces the number of tuples available for join.
- Perform the most restrictive select/project operations at first before the other operations.
- Avoid cross-product operation since they result in very large-sized intermediate tables.

Query Optimization in Distributed Systems

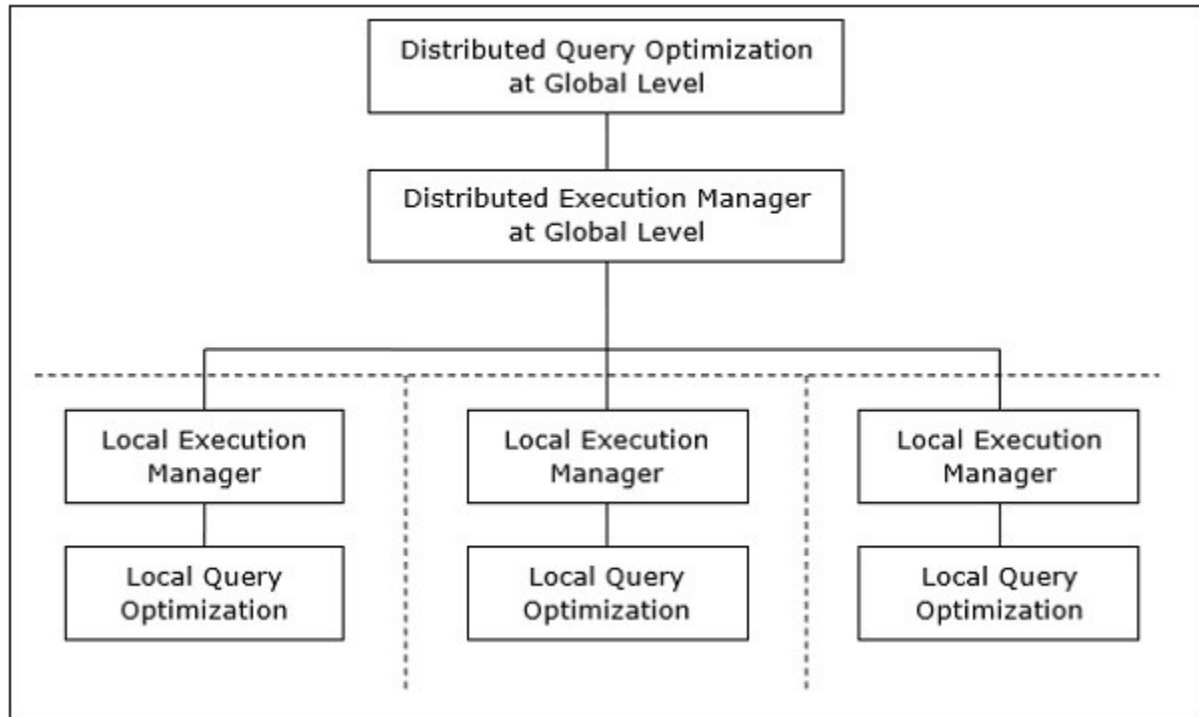
This chapter discusses query optimization in distributed database system.

Distributed Query Processing Architecture

In a distributed database system, processing a query comprises of optimization at both the global and the local level. The query enters the database system at the client

or controlling site. Here, the user is validated, the query is checked, translated, and optimized at a global level.

The architecture can be represented as –



Mapping Global Queries into Local Queries

The process of mapping global queries to local ones can be realized as follows –

- The tables required in a global query have fragments distributed across multiple sites. The local databases have information only about local data. The controlling site uses the global data dictionary to gather information about the distribution and reconstructs the global view from the fragments.
- If there is no replication, the global optimizer runs local queries at the sites where the fragments are stored. If there is replication, the global optimizer selects the site based upon communication cost, workload, and server speed.
- The global optimizer generates a distributed execution plan so that least amount of data transfer occurs across the sites. The plan states the location of the fragments, order in which query steps needs to be executed and the processes involved in transferring intermediate results.

- The local queries are optimized by the local database servers. Finally, the local query results are merged together through union operation in case of horizontal fragments and join operation for vertical fragments.

For example, let us consider that the following Project schema is horizontally fragmented according to City, the cities being New Delhi, Kolkata and Hyderabad.

PROJECT

PIId	City	Department	Status
------	------	------------	--------

Suppose there is a query to retrieve details of all projects whose status is “Ongoing”.

The global query will be &inus;

$\sigma_{\text{status}=\text{"ongoing"}}(\text{PROJECT})$

Query in New Delhi’s server will be –

$\sigma_{\text{status}=\text{"ongoing"}}(\text{NewD-PROJECT})$

Query in Kolkata’s server will be –

$\sigma_{\text{status}=\text{"ongoing"}}(\text{Kol-PROJECT})$

Query in Hyderabad’s server will be –

$\sigma_{\text{status}=\text{"ongoing"}}(\text{Hyd-PROJECT})$

In order to get the overall result, we need to union the results of the three queries as follows –

$\sigma_{\text{status}=\text{"ongoing"}}(\text{NewD-PROJECT}) \cup \sigma_{\text{status}=\text{"ongoing"}}(\text{kol-PROJECT}) \cup \sigma_{\text{status}=\text{"ongoing"}}(\text{Hyd-PROJECT})$

Distributed Query Optimization

Distributed query optimization requires evaluation of a large number of query trees each of which produce the required results of a query. This is primarily due to the presence of large amount of replicated and fragmented data. Hence, the target is to find an optimal solution instead of the best solution.

The main issues for distributed query optimization are –

- Optimal utilization of resources in the distributed system.
- Query trading.
- Reduction of solution space of the query.

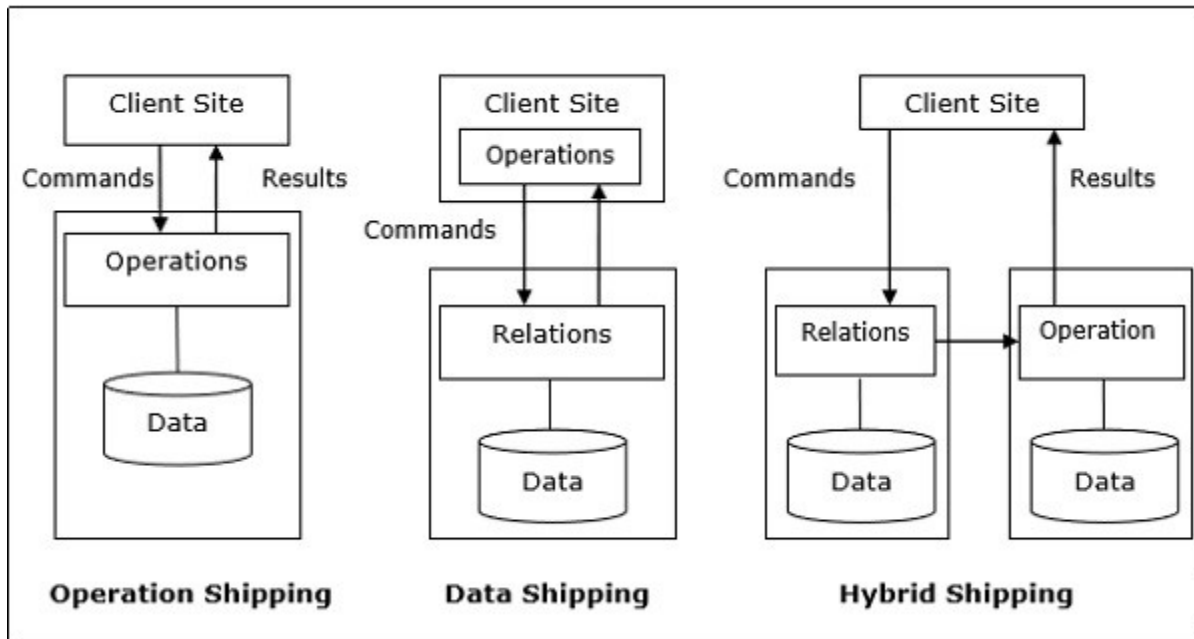
Optimal Utilization of Resources in the Distributed System

A distributed system has a number of database servers in the various sites to perform the operations pertaining to a query. Following are the approaches for optimal resource utilization –

Operation Shipping – In operation shipping, the operation is run at the site where the data is stored and not at the client site. The results are then transferred to the client site. This is appropriate for operations where the operands are available at the same site. Example: Select and Project operations.

Data Shipping – In data shipping, the data fragments are transferred to the database server, where the operations are executed. This is used in operations where the operands are distributed at different sites. This is also appropriate in systems where the communication costs are low, and local processors are much slower than the client server.

Hybrid Shipping – This is a combination of data and operation shipping. Here, data fragments are transferred to the high-speed processors, where the operation runs. The results are then sent to the client site.



Query Trading

In query trading algorithm for distributed database systems, the controlling/client site for a distributed query is called the buyer and the sites where the local queries execute are called sellers. The buyer formulates a number of alternatives for choosing sellers and for reconstructing the global results. The target of the buyer is to achieve the optimal cost.

The algorithm starts with the buyer assigning sub-queries to the seller sites. The optimal plan is created from local optimized query plans proposed by the sellers combined with the communication cost for reconstructing the final result. Once the global optimal plan is formulated, the query is executed.

Reduction of Solution Space of the Query

Optimal solution generally involves reduction of solution space so that the cost of query and data transfer is reduced. This can be achieved through a set of heuristic rules, just as heuristics in centralized systems.

Following are some of the rules –

- Perform selection and projection operations as early as possible. This reduces the data flow over communication network.
- Simplify operations on horizontal fragments by eliminating selection conditions which are not relevant to a particular site.

- In case of join and union operations comprising of fragments located in multiple sites, transfer fragmented data to the site where most of the data is present and perform operation there.
- Use semi-join operation to qualify tuples that are to be joined. This reduces the amount of data transfer which in turn reduces communication cost.
- Merge the common leaves and sub-trees in a distributed query tree.

DDBMS - Transaction Processing Systems

This chapter discusses the various aspects of transaction processing. We'll also study the low level tasks included in a transaction, the transaction states and properties of a transaction. In the last portion, we will look over schedules and serializability of schedules.

Transactions

A transaction is a program including a collection of database operations, executed as a logical unit of data processing. The operations performed in a transaction include one or more of database operations like insert, delete, update or retrieve data. It is an atomic process that is either performed into completion entirely or is not performed at all. A transaction involving only data retrieval without any data update is called read-only transaction.

Each high level operation can be divided into a number of low level tasks or operations. For example, a data update operation can be divided into three tasks –

- **read_item()** – reads data item from storage to main memory.
- **modify_item()** – change value of item in the main memory.
- **write_item()** – write the modified value from main memory to storage.

Database access is restricted to `read_item()` and `write_item()` operations. Likewise, for all transactions, read and write forms the basic database operations.

Transaction Operations

The low level operations performed in a transaction are –

- **begin_transaction** – A marker that specifies start of transaction execution.
- **read_item or write_item** – Database operations that may be interleaved with main memory operations as a part of transaction.

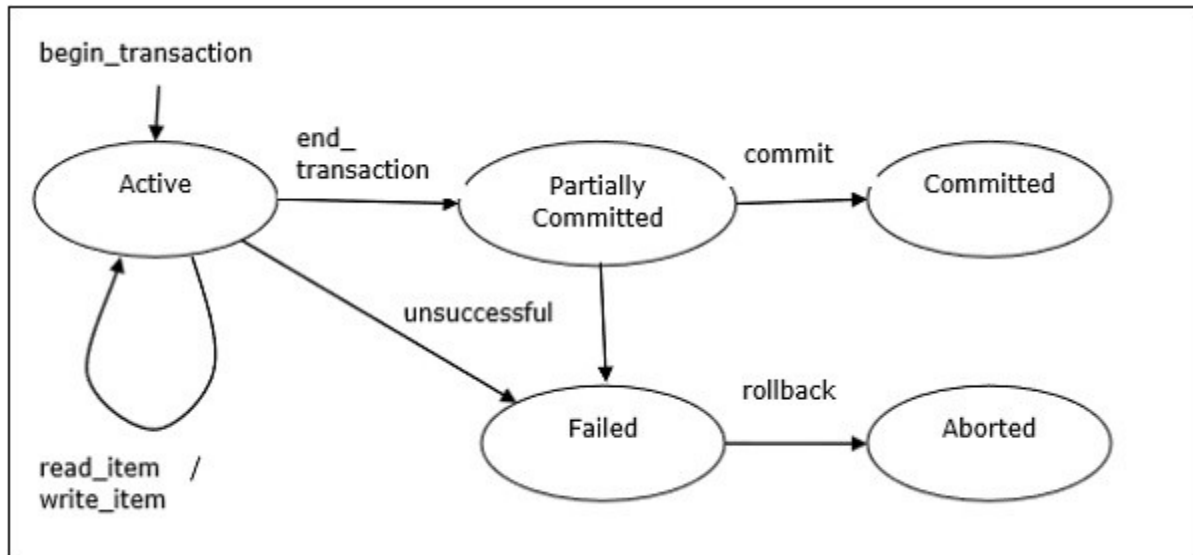
- **end_transaction** – A marker that specifies end of transaction.
- **commit** – A signal to specify that the transaction has been successfully completed in its entirety and will not be undone.
- **rollback** – A signal to specify that the transaction has been unsuccessful and so all temporary changes in the database are undone. A committed transaction cannot be rolled back.

Transaction States

A transaction may go through a subset of five states, active, partially committed, committed, failed and aborted.

- **Active** – The initial state where the transaction enters is the active state. The transaction remains in this state while it is executing read, write or other operations.
- **Partially Committed** – The transaction enters this state after the last statement of the transaction has been executed.
- **Committed** – The transaction enters this state after successful completion of the transaction and system checks have issued commit signal.
- **Failed** – The transaction goes from partially committed state or active state to failed state when it is discovered that normal execution can no longer proceed or system checks fail.
- **Aborted** – This is the state after the transaction has been rolled back after failure and the database has been restored to its state that was before the transaction began.

The following state transition diagram depicts the states in the transaction and the low level transaction operations that causes change in states.



Desirable Properties of Transactions

Any transaction must maintain the ACID properties, viz. Atomicity, Consistency, Isolation, and Durability.

- **Atomicity** – This property states that a transaction is an atomic unit of processing, that is, either it is performed in its entirety or not performed at all. No partial update should exist.
- **Consistency** – A transaction should take the database from one consistent state to another consistent state. It should not adversely affect any data item in the database.
- **Isolation** – A transaction should be executed as if it is the only one in the system. There should not be any interference from the other concurrent transactions that are simultaneously running.
- **Durability** – If a committed transaction brings about a change, that change should be durable in the database and not lost in case of any failure.

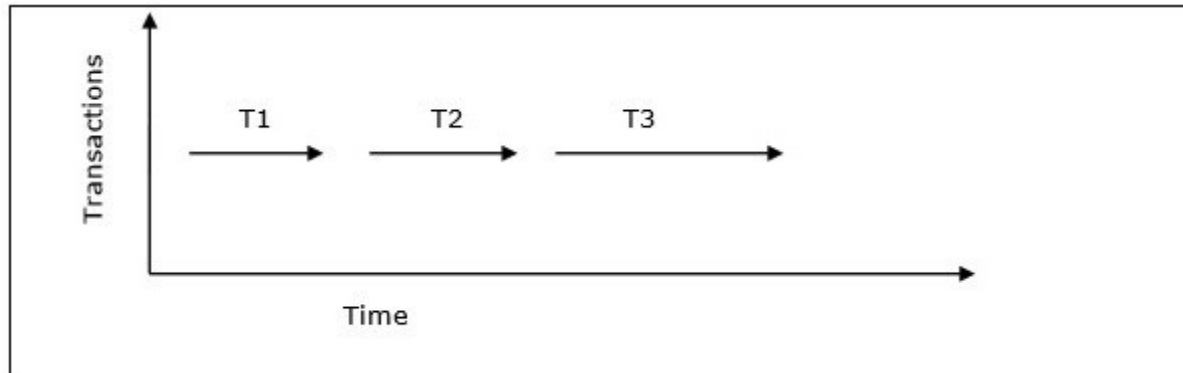
Schedules and Conflicts

In a system with a number of simultaneous transactions, a **schedule** is the total order of execution of operations. Given a schedule S comprising of n transactions, say T1, T2, T3.....Tn; for any transaction Ti, the operations in Ti must execute as laid down in the schedule S.

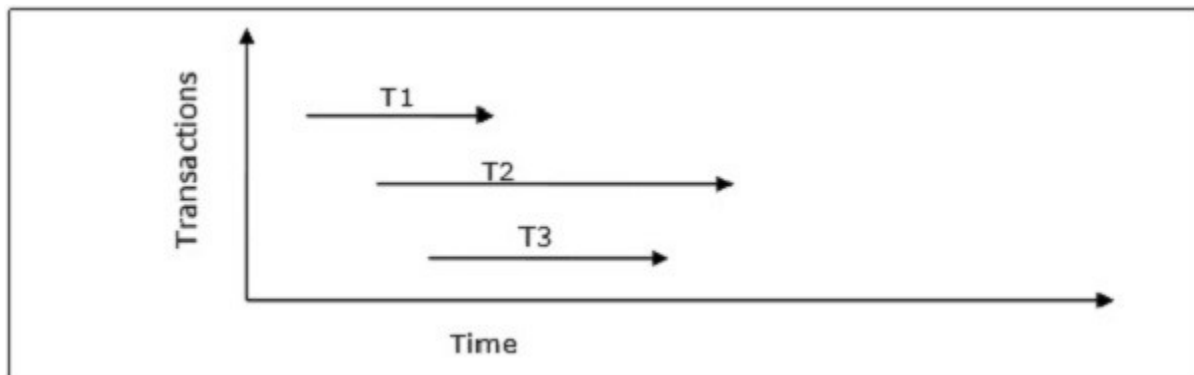
Types of Schedules

There are two types of schedules –

- **Serial Schedules** – In a serial schedule, at any point of time, only one transaction is active, i.e. there is no overlapping of transactions. This is depicted in the following graph –



- **Parallel Schedules** – In parallel schedules, more than one transactions are active simultaneously, i.e. the transactions contain operations that overlap at time. This is depicted in the following graph –



Conflicts in Schedules

In a schedule comprising of multiple transactions, a **conflict** occurs when two active transactions perform non-compatible operations. Two operations are said to be in conflict, when all of the following three conditions exists simultaneously –

- The two operations are parts of different transactions.
- Both the operations access the same data item.
- At least one of the operations is a write_item() operation, i.e. it tries to modify the data item.

Serializability

A **serializable schedule** of 'n' transactions is a parallel schedule which is equivalent to a serial schedule comprising of the same 'n' transactions. A serializable schedule contains the correctness of serial schedule while ascertaining better CPU utilization of parallel schedule.

Equivalence of Schedules

Equivalence of two schedules can be of the following types –

- **Result equivalence** – Two schedules producing identical results are said to be result equivalent.
- **View equivalence** – Two schedules that perform similar action in a similar manner are said to be view equivalent.
- **Conflict equivalence** – Two schedules are said to be conflict equivalent if both contain the same set of transactions and has the same order of conflicting pairs of operations.
- **Relations as unit of distribution:**

– If the relation is not replicated, we get a high volume of remote data accesses.

– If the relation is replicated, we get unnecessary replications, which cause problems in executing updates and waste disk space

– Might be an Ok solution, if queries need all the data in the relation and data stays at the only sites that uses the data.

- **Fragments of relations as unit of distribution:**

– Application views are usually subsets of relations.

– Thus, locality of accesses of applications is defined on subsets of relations.

– Permits a number of transactions to execute concurrently, since they will access different portions of a relation.

– Parallel execution of a single query (intra-query concurrency).

– However, semantic data control (especially integrity enforcement) is more difficult.

Fragments of relations are (usually) the appropriate unit of distribution.

Fragmentation aims to improve:

– Reliability.

– Performance.

– Balanced storage capacity and costs.

– Communication costs.

– Security.

• The following information is used to decide fragmentation:

– Quantitative information: frequency of queries, site, where query is run, selectivity of the queries, etc.

– Qualitative information: types of access of data, read/write, etc.

Types of Fragmentation

– Horizontal: partitions a relation along its tuples

– Vertical: partitions a relation along its attributes

– Mixed/hybrid: a combination of horizontal and vertical fragmentation

Example:

<u>Branch-name</u>	<u>account-number</u>	<u>customer-name</u>	<u>balance</u>
Baghdad	305	Salem	500
Baghdad	226	Ahmed	336

Mousel	177	Ahmed	205
Mousel	402	Hassan	1000
Baghdad	155	Hassan	62
Mousel	408	Hassan	1123
Mousel	639	Ali	750

Table (1)

Horizontal fragmentations

Consists of partitioning the tuples of a global relation r into subsets $r_1, r_2 \dots r_n$ each subset can contain data with common properties. The reconstruction of relation r can be obtained by taking the union of all fragments, that is: $r = r_1 \cup r_2 \cup \dots \cup r_n$ For example, suppose that the relation r is the deposit relation of table (1) this relation has only two branches, Baghdad and Mosul, and if we choose the attribute branch-name for horizontal fragmentation the relation, then the result are two different fragment shows in Table (2).

<u>Branch-name</u>	<u>account-number</u>	<u>customer-name</u>	<u>balance</u>
Baghdad	305	Salem	500
Baghdad	226	Ahmad	336
Baghdad	155	Hassan	62

deposit1

<u>Branch-name</u>	<u>customer-number</u>	<u>account-name</u>	<u>balance</u>
Mosul	117	Ahmad	205
Mosul	402	Hassan	1000
Mosul	408	Hassan	1123
Mosul	639	Ali	750

deposit2

Table (2)

Vertical Fragmentations

Vertical Fragmentation for global relation is the subdivision of its attributes into groups; subdivision is accomplished by adding a special attribute called a tuple-id to the scheme R. A tuple-id is a physical or logical address for a tuple since each tuple in r must have a unique address; the tuple-id attribute is a key for the scheme.

Deposit-scheme3 = (branch-name, customer-name, tuple-id)

Deposit-scheme4 = (account-number, balance, tuple-id)

<u>Branch-name</u>	<u>account-number</u>	<u>customer-name</u>	<u>balance</u>	<u>tuple-id</u>
Baghdad	305	Salem	500	1
Baghdad	226	Ahmad	336	2
Mosul	177	Ahmad	205	3
Mosul	402	Hassan	1000	4
Baghdad	155	Hassan	62	5
Mosul	408	Hassan	1123	6
Mosul	639	Ali	750	7

Table (3)

<u>Branch-name</u>	<u>customer-name</u>	<u>tuple-id</u>
Baghdad	Salem	1
Baghdad	Ahmad	2
Mosul	Ahmad	3
Mosul	Hassan	4

Baghdad	Hassan	5
Mosul	Hassan	6
Mosul	Ali	7

Deposit3

<u>Account-number</u>	<u>blanance</u>	<u>tuple-id</u>
305	500	1
226	336	2
177	205	3
402	1000	4
155	62	5
408	1123	6
639	75	7

Deposit 4

Table (4)

3- Distribution Design Issues

In the preceding section we indicated that the relations in a database schema are usually decomposed into smaller fragments, but we did not offer any justification or details for this process. The objective of this section is to fill in these details. The following set of interrelated questions covers the entire issue. We will therefore seek to answer them in the remainder of this section.

1. Why fragment at all?
2. How should we fragment?
3. How much should we fragment?
4. Is there any way to test the correctness of decomposition?

5. How should we allocate?

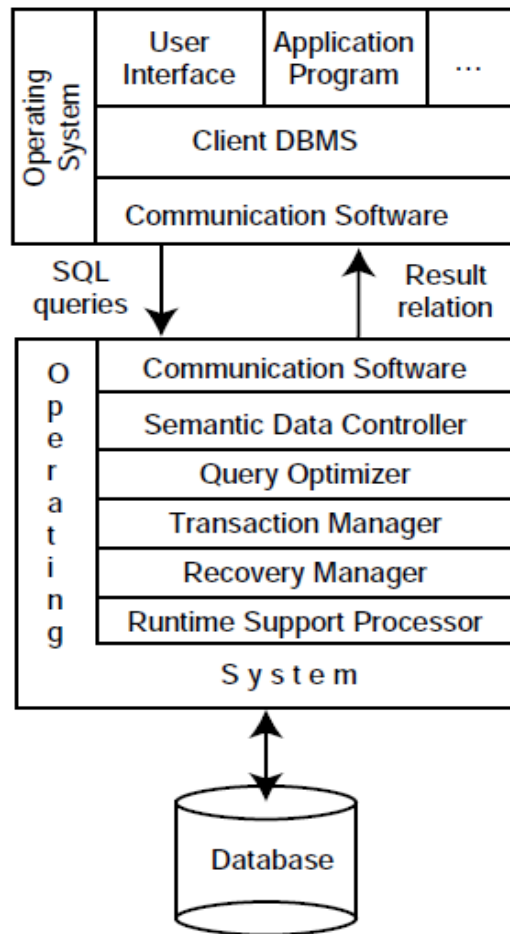
6. What is the necessary information for fragmentation and allocation?

1.8. Architectural Alternatives

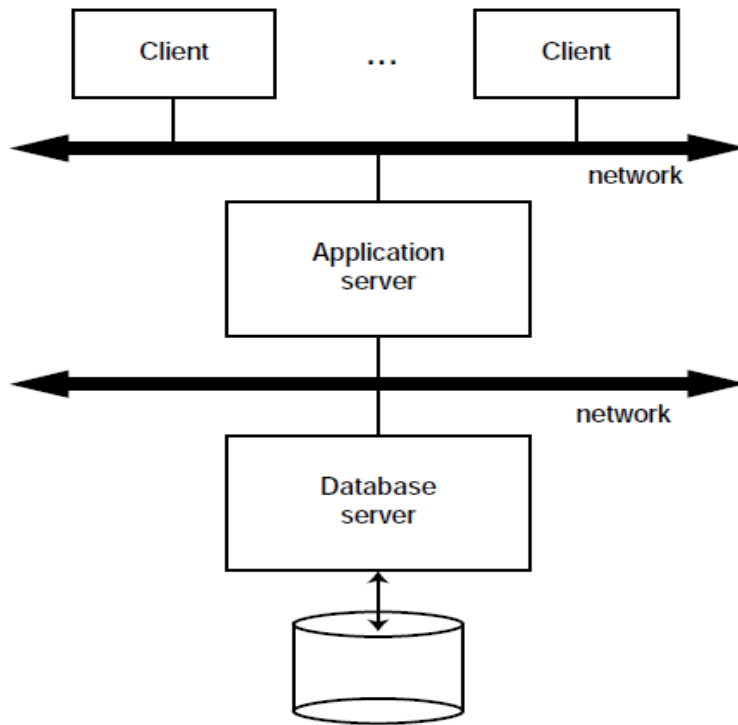
Architecturally, a distributed database system consists of a (possibly empty) set of query sites and a non-empty set of data sites. The data sites have data storage capability while the query sites do not. The latter only run the user interface (in addition to applications) in order to facilitate data access at data sites.

Client/Server Systems

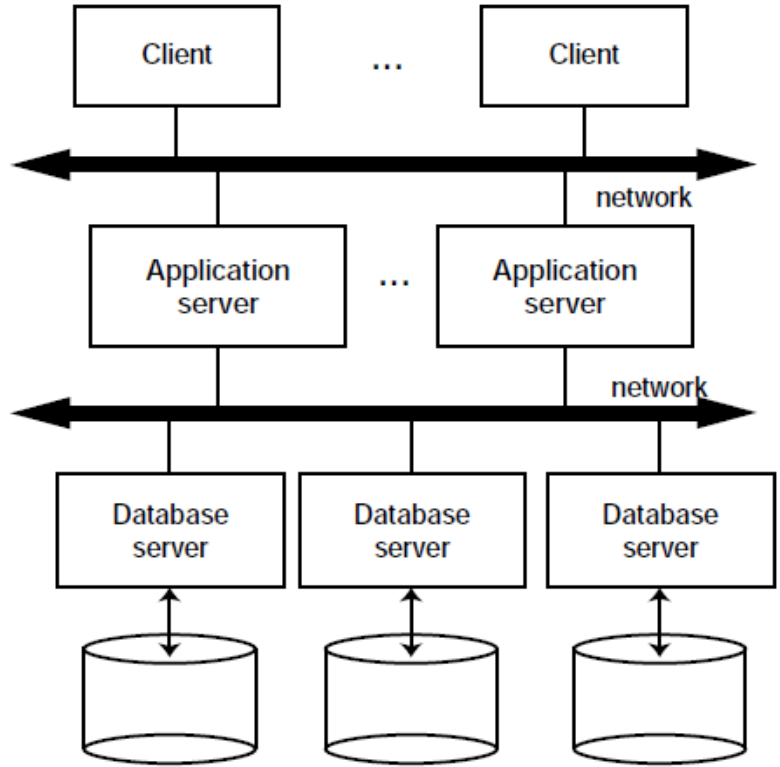
Client/server DBMSs entered the computing scene at the beginning of 1990's and have made a significant impact on both the DBMS technology and the way we do computing. The general idea is very simple and elegant: distinguish the functionality that needs to be provided and divide these functions into two classes: server functions and client functions. This provides a two-level architecture which makes it easier to manage the complexity of modern DBMSs and the complexity of distribution. As with any highly popular term, client/server has been much abused and has come to mean different things



Client/Server Reference Architecture



Data base server approach



Distributed Database Servers