**Basic Object Oriented Data Model**

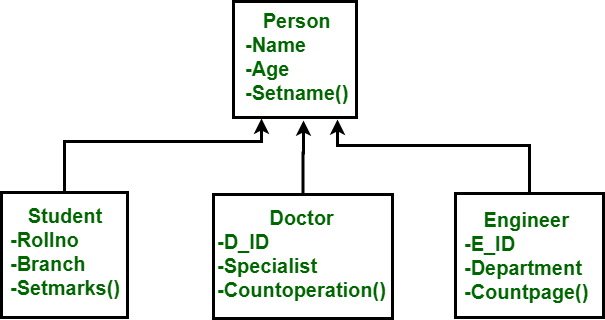
**eed of Object Oriented Data Model :**  
To represent the complex real world problems there was a need for a data model that is closely related to real world. Object Oriented Data Model represents the real world problems easily.

**Object Oriented Data Model :**  
In Object Oriented Data Model, data and their relationships are contained in a single structure which is referred as object in this data model. In this, real world problems are represented as objects with different attributes. All objects have multiple relationships between them. Basically, it is combination of Object Oriented programming and Relational Database Model as it is clear from the following figure :

Object Oriented Data Model

= Combination of Object Oriented Programming + Relational database model

**Components of Object Oriented Data Model :**



*Basic Object Oriented Data Model*

* **Objects –**  
  An object is an abstraction of a real world entity or we can say it is an instance of class. Objects encapsulates data and code into a single unit which provide data abstraction by hiding the implementation details from the user. For example: Instances of student, doctor, engineer in above figure.
* **Attribute –**  
  An attribute describes the properties of object. For example: Object is STUDENT and its attribute are Roll no, Branch, Semester in the Student class.
* **Methods –**  
  Method represents the behavior of an object. Basically, it represents the real-world action. For example: Finding a STUDENT marks in above figure as Setmarks().
* **Class –**  
  A class is a collection of similar objects with shared structure i.e. attributes and behavior i.e. methods. An object is an instance of class. For example: Person, Student, Doctor, Engineer in above figure.

class student

{

char Name[20];

int roll\_no;

--

--

public:

void search();

void update();

}

In this example, students refers to class and S1, S2 are the objects of class which can be created in main function.

* **Inheritance –**  
  By using inheritance, new class can inherit the attributes and methods of the old class i.e. base class. For example: as classes Student, Doctor and Engineer are inherited from the base class Person.

**Advantages of Object Oriented Data Model :**

* Codes can ne reused due to inheritance.
* Easily understandable.
* Cost of maintenance can reduced due to reusability of attributes and functions because of inheritance.

**Disadvantages of Object Oriented Data Model :**

* It is not properly developed so not accepted by users easily.

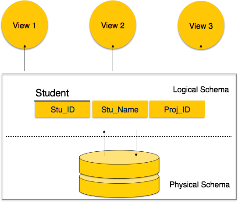
Attention reader! Don’t stop learning now. Get hold of all the important CS Theory concepts for SDE interviews with the [**CS Theory Course**](https://practice.geeksforgeeks.org/courses/SDE-theory?vC=1) at a student-friendly price and become industry ready.

|  |  |
| --- | --- |
| **Abstract Data Types or structure (ADT)** | **Concrete Data Types or structure (CDT)** |
| **1** | Abstract Data Types or structures describe the data and the operations to manipulate and change it. | Concrete data types or structures provide  how these operations are actually implemented. |
| **2** | Most of the program becomes independent of the abstract data types representation, so it can be improved without breaking the program. | Which is not possible in Concrete Data Types or structure (CDT) |
| **3** | It’s easier for each part of a program to use an implementation of its data types and that will be more efficient. | It is not so efficient compared to ADT. |
| **4** | Implementation of a high level concept | Implementation of a simple concept |
| **5** | It is usable beyond its original use. | It is rarely reusable beyond its original use. |
| **6** | It hides the internal details. | It doesn’t hide anything. |
| **7** | It uses class. | It uses structure. |
| **8** | Examples- lists, sets, stacks. | Examples-Arrays, linked lists, trees, graphs. |

Database Schema

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It’s the database designers who design the schema to help programmers understand the database and make it useful.



A database schema can be divided broadly into two categories −

* **Physical Database Schema** − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
* **Logical Database Schema** − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

Database Instance

It is important that we distinguish these two terms individually. Database schema is the skeleton of database. It is designed when the database doesn't exist at all. Once the database is operational, it is very difficult to make any changes to it. A database schema does not contain any data or information.

A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time. A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.

What is a Database Schema?

Simply put, a database schema is a formal description of the structure or organization of a particular database. The term “database schema” is most commonly used in reference to relational databases, i.e. databases that organize information in tables and use the SQL query language. Non-relational (i.e. “NoSQL”) databases come in several different formats and are not generally considered to have a “schema” in the same way that relational databases do (although they do have an underlying structure).

**Related Reading:**[SQL vs. NoSQL: 5 Critical Differences](https://www.xplenty.com/blog/the-sql-vs-nosql-difference/)

There are two fundamental components of any database schema:

* **Physical database schema:** The physical database schema describes how data will be stored physically on a storage system and the form of storage used (files, key-value pairs, indices, etc.).
* **Logical database schema:** The logical database schema describes the logical constraints applied to the data and defines fields, tables, relations, views, integrity constraints, etc. These requirements provide useful information that programmers can apply to the physical design of the database. The rules or constraints that are defined in this logical model help determine how the data in different tables relate to each other.

The definition of physical tables in the schema comes from the logical data model. Entities become tables, the entity’s attributes become table fields, etc.

6 Types of Database Schemas

What are your options when it comes to the various types of database schemas? In this section, we’ll give a brief overview of some of the most common database schema types.

* **Flat model:** A “flat model” database schema organizes data in a single, two-dimensional array—think of a Microsoft Excel spreadsheet or a [CSV file](https://www.xplenty.com/blog/what-is-a-csv-file-and-how-to-work-with-it/). This schema is best for simple tables and databases without complex relations between different entities.
* **Hierarchical model:** Database schemas in a hierarchical model have a “tree-like” structure, with child nodes branching out from a root data node. This schema is ideal for storing nested data—for example, family trees or biological taxonomies.
* **Network model:** The network model, like the hierarchical model, treats data as nodes connected to each other; however, it allows for more complex connections, such as many-to-many relationships and cycles. This schema can model the movement of goods and materials between locations, or the workflow required to accomplish a particular task.
* **Relational model:** As discussed above, this model organizes data in a series of tables, rows, and columns, with relationships between different entities. We’ll mainly be working with the relational model in the remainder of this article.
* **Star schema:** The star schema is an evolution of the relational model that organizes data into “facts” and “dimensions.” Fact data is numerical (e.g. the number of sales of a product), while dimensional data is descriptive (e.g. the product’s price, color, weight, etc.).
* **Snowflake schema:** The [snowflake schema](https://www.xplenty.com/blog/snowflake-schemas-vs-star-schemas-what-are-they-and-how-are-they-different/) is a further abstraction on top of the star schema. Fact tables point to dimensional tables, which can also have their own dimensional tables, expanding the descriptiveness possible within the database. (As you might have guessed, the “snowflake” schema is named after the intricate patterns of a snowflake, in which smaller structures radiate off of the central arms.)

**Related Reading:**[6 Database Schema Designs and How to Use Them](https://www.xplenty.com/blog/database-schema-examples/)

What is Database Schema Design?

Database schema design refers to the practices and strategies for constructing a database schema.

You can think of database schema design as a “blueprint” for how to store massive amounts of information in a database. The schema is an abstract structure or outline that represents the logical view of the database as a whole. By defining categories of data and relationships between those categories, database schema design makes data much easier to retrieve, consume, manipulate, and interpret.

Database schema design organizes the data into separate entities, determines how to create relationships between organized entities, and how to apply the constraints on the data. Designers create database schemas to give other database users, such as programmers and analysts, a logical understanding of the data.

Integrate Your Data Today!

Try Xplenty free for 14 days. No credit card required.

Top of Form





Get Started

Bottom of Form

**Why is Database Schema Design Important?**

Databases that are inefficiently organized suck up tons of energy and resources, tend to be confusing, and are hard to maintain and administer. That’s where database schema design comes into play.

Without a clean, efficient, consistent database schema, you’ll struggle to make the best use of your enterprise data. For example, the same data might be duplicated in multiple locations—or even worse, might be inconsistent between these locations.

Relational database systems heavily depend on having a solid database schema in place. The goals

**Storage structure is the memory structure in the system. It is mainly divided into two categories :**

**Volatile Memory**

These are the primary memory devices in the system, and are placed along with the CPU.  These memories can store only small amount of data, but they are very fast. E.g.:- main memory, cache memory etc. these memories cannot endure system crashes- data in these memories will be lost on failure.

**Non-Volatile memory**

These are secondary memories and are huge in size, but slow in processing. E.g.:- Flash memory, hard disk, magnetic tapes etc. these memories are designed to withstand system crashes.

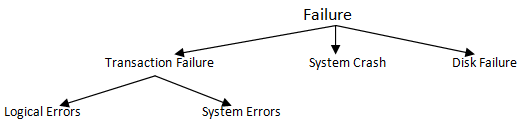
**Stable Memory**

This is said to be third form of memory structure but it is same as non volatile memory. In this case, copies of same non volatile memories are stored at different places. This is because, in case of any crash and data loss, data can be recovered from other copies. This is even helpful if there one of non-volatile memory is lost due to fire or flood. It can be recovered from other network location. But there can be failure while taking the backup of DB into different stable storage devices. Even it may fail to transfer all the data successfully; either it will partially transfer the data to remote devices or completely fail to store the data in stable memory.  Hence extra caution has to be taken while taking the backup of data from one stable memory to other. There are different methods followed to copy the data. One of them is to copy the data in two phases – copy the data blocks to first storage device, if it is successful copy to second storage device. The copying is complete only when second copy is executed successfully. But second copy of data blocks may fail to copy whole blocks. In such case, each data blocks in first copy and second copy needs to be compared for its inconsistency. But verifying each blocks would be very costly task as we may have huge number of data block. One of the better way to identify the failed block is to identify the block which was in progress during the failure. Take only this block, compare the data and correct the mismatches.

**Failure Classification**

When a transaction is being executed in the system, it may fail to execute due to various reasons. The failure can be because of system program, bug in a program, user, or system crash. These failures can be broadly classified into three categories.

* **Transaction Failure :**This type of failure affects only few [tables](https://www.tutorialcup.com/dbms/tables.htm) or processes. This is the condition in the transaction where a transaction cannot execute it further. This failure can be because of user or executing program/ transaction. The user may cancel the transaction when the transaction is executing by pressing the cancel button or abort using the DB commands. The transaction may fail because of the constraints on the tables – violation of constraints. It can even fail if there is concurrent processing of multiple transactions and there is lack of resources for all of them or deadlock situation. All these will cause the transaction to stop processing in the middle of its execution. When a transaction fails / stops in the middle, it would have partially changed DB and it needs to be rolled back to previous consistent state. In ATM withdrawal example, if the user cancels his transaction after step (i), the system should be able to stop further processing of the transaction, or if he cancels the transaction after step (ii), the system should be strong enough to update his balance in his account. Here system may cancel the transaction due to insufficient balance. The failure can be because of errors in the code – logical errors or because of system errors like deadlock or unavailability of system resources to execute the transactions.
* **System Crash :** This can be because of hardware or software failure or because of external factors like power failure. This is the failure of the system because of the bug in the software or the failure of system processor. This crash mainly affects the data in the primary memory. If it affects only the primary memory, the actual data will not be really affected and recovery from this failure is easy. This is because primary memories are temporary storages and it would not have updated the actual database. Hence the system will be in a consistent state before to the transaction. But when secondary memory crashes, there would be a loss of data and need to take serious actions to recover lost data. Because secondary memories contain actual DB data. Recovering them from crash is little tedious and requires more effort. DB Recovery system provides strong mechanisms to recovery the system from crash and maintains the atomicity of the transactions. In most of the cases data in the secondary memory are not affected because of this crash. This is because; the database has lots of integrity checkpoints to prevent the data loss from secondary memory.
* **Disk Failure :** These are the issues with hard disks like formation of bad sectors, disk head crash, unavailability of disk etc. Data can even be lost because of fire, flood, theft etc. This is mainly affects the secondary memory where the actual data lies. In these cases, we need to have alternative ways of storing DB. We can create backups of DB at regular basis and store them separately from the memory where DB is stored or maintain multiple copies of DB at different network locations to recover them from failure.

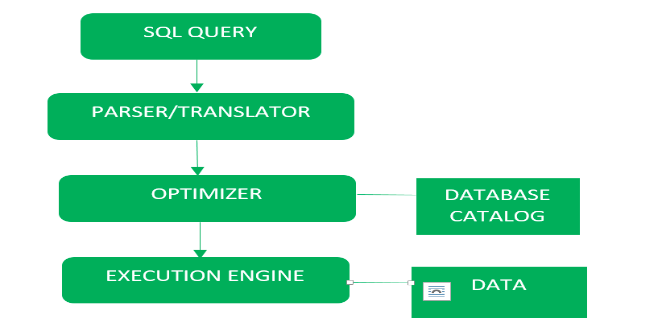


**SQL | Query Processing**

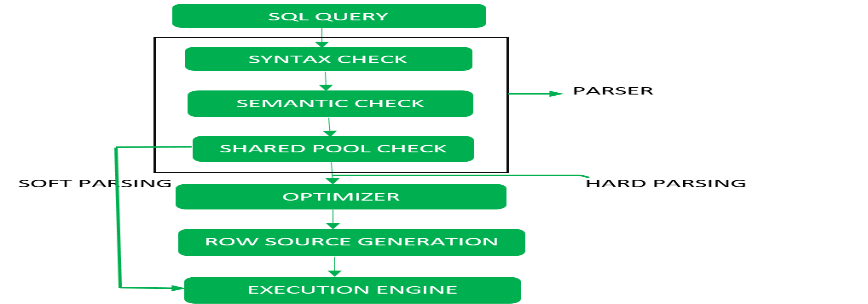
* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 14 Aug, 2018

**Query Processing** includes translations on high level Queries into low level expressions that can be used at physical level of file system, query optimization and actual execution of query to get the actual result.

Block Diagram of Query Processing is as:



Detailed Diagram is drawn as:



1. Query Processing and Optimization
2. [2.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-2-638.jpg?cb=1488898281)Introduction • In this chapter we shall discuss the techniques used by a DBMS to process, optimize and execute high-level queries. • The techniques used to split complex queries into multiple simple operations and methods of implementing these low-level operations. • The query optimization techniques are used to chose an efficient execution plan that will minimize the runtime as well as many other types of resources such as number of disk I/O, CPU time and so on.
3. [3.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-3-638.jpg?cb=1488898281)Query Processing • Query Processing is a procedure of transforming a high-level query (such as SQL) into a correct and efficient execution plan expressed in low-level language. • When a database system receives a query for update or retrieval of information, it goes through a series of compilation steps, called execution plan. • Query processing goes through various phases: • first phase is called syntax checking phase, the system parses the query and checks that it follows the syntax rules or not. • It then matches the objects in the query syntax with the view tables and columns listed in the system table.
4. [4.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-4-638.jpg?cb=1488898281)• In second phase the SQL query is translated in to an algebraic expression using various rules. • So that the process of transforming a high-level SQL query into a relational algebraic form is called Query Decomposition. • The relational algebraic expression now passes to the query optimizer. • In third phase optimization is performed by substituting equivalent expression depends on the factors such that the existence of certain database structures, whether or not a given file is stored, the presence of different indexes & so on. • Query optimization module work in tandem with the join manager module to improve the order in which joins are performed.
5. [5.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-5-638.jpg?cb=1488898281)• At this stage the cost model and several other estimation formulas are used to rewrite the query. • The modified query is written to utilize system resources so as to bring the optimal performance. • The query optimizer then generates an action plan also called a execution plan. • This action plans are converted into a query codes that are finally executed by a run time database processor. • The run time database processor estimate the cost of each action plan and chose the optimal one for the execution.
6. [6.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-6-638.jpg?cb=1488898281)Query Analyzer • The syntax analyzer takes the query from the users, parses it into tokens and analyses the tokens and their order to make sure they follow the rules of the language grammar. • Is an error is found in the query submitted by the user, it is rejected and an error code together with an explanation of why the query was rejected is return to the user.
7. [7.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-7-638.jpg?cb=1488898281)Query Decomposition • In query decomposition the query processing aims are to transfer the high-level query into a relational algebra query and to check whether that query is syntactically and semantically correct. • Thus the query decomposition is start with a high-level query and transform into query graph of low-level operations, which satisfy the query. • The SQL query is decomposed into query blocks (low-level operations), which form the basic unit. • Hence nested queries within a query are identified as separate query blocks. • The query decomposer goes through five stages of processing for decomposition into low-level operation and translation into algebraic expressions.
8. [8.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-8-638.jpg?cb=1488898281)Query Analysis • During the query analysis phase, the query is syntactically analyzed using the programming language compiler (parser). • A syntactically legal query is then validated, using the system catalog, to ensure that all data objects (relations and attributes) referred to by the query are defined in the database. • The type specification of the query qualifiers and result is also checked at this stage.
9. [13.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-13-638.jpg?cb=1488898281)Query Normalization • The primary phase of the normalization is to avoid redundancy. • The normalization phase converts the query into a normalized form that can be more easily manipulated. • In the normalization phase, a set of equivalency rules are applied so that the projection and selection operations included on the query are simplified to avoid redundancy. • The projection operation corresponds to the SELECT clause of SQL query and the selection operation correspond to the predicate found in WHERE clause. • The equivalency transformation rules that are applied.
10. [17.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-17-638.jpg?cb=1488898281)Query Optimization • The primary goal of query optimization is of choosing an efficient execution strategy for processing a query. • The query optimizer attempts to minimize the use of certain resources (mainly the number of I/O and CPU time) by selecting a best execution plan (access plan). • A query optimization start during the validation phase by the system to validate the user has appropriate privileges. • Now an action plan is generate to perform the query.
11. [18.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-18-638.jpg?cb=1488898281)Block Diagram of Query Optimization
12. [19.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-19-638.jpg?cb=1488898281)• Relational algebra query tree generated by the query simplifier module of query decomposer. • Estimation formulas used to determine the cardinality of the intermediate result table. • A cost Model. • Statistical data from the database catalogue. ϖThe output of the query optimizer is the execution plan in form of optimized relational algebra query. ϖA query typically has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as Query Optimization.
13. [20.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-20-638.jpg?cb=1488898281)The basic issues in Query Optimization • How to use available indexes? • How to use memory to accumulate information and perform immediate steps such as sorting? • How to determine the order in which joins should be performed?
14. [21.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-21-638.jpg?cb=1488898281)Objective of query optimization • The term query optimization does not mean giving always an optimal (best) strategy as the execution plan. • It is just a responsibly efficient strategy for execution of the query. • The decomposed query block of SQL is translating into an equivalent extended relational algebra expression and then optimized.
15. [22.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-22-638.jpg?cb=1488898281)Techniques for Query Optimization • The first technique is based on Heuristic Rules for ordering the operations in a query execution strategy. • The second technique involves the systematic estimation of the cost of the different execution strategies and choosing the execution plan with the lowest cost.
16. [23.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-23-638.jpg?cb=1488898281)• Semantic query optimization is used with the combination with the heuristic query transformation rules. • It uses constraints specified on the database schema such as unique attributes and other more complex constraints, in order to modify one query into another query that is more efficient to execute.
17. [24.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-24-638.jpg?cb=1488898281)Heuristic Rules • The heuristic rules are used as an optimization technique to modify the internal representation of query. • Usually, heuristic rules are used in the form of query tree of query graph data structure, to improve its performance. • One of the main heuristic rule is to apply SELECT operation before applying the JOIN or other BINARY operations. • This is because the size of the file resulting from a binary operation such as JOIN is usually a multi- value function of the sizes of the input files.
18. [25.](https://image.slidesharecdn.com/queryprocessingandoptimizationnew-170307144940/95/query-processing-and-optimization-updated-25-638.jpg?cb=1488898281)Heuristic Rules • The SELECT and PROJECT reduced the size of the file and hence, should be applied before the JOIN or other binary operation. • Heuristic query optimizer transforms the initial (canonical) query tree into final query tree using equivalence transformation rules. • This final query tree is efficient to execute.

Introduction to Query Processing

* In databases that provide low-level access routines such as IMS or flat file databases, the programmer must write code to perform the queries.
* With higher level database query languages such as SQL and QUEL, a special component of the DBMS called the Query Processor takes care of arranging the underlying access routines to satisfy a given query.
* Thus queries can be specified in terms of the required results rather than in terms of how to achieve those results.

A query is processed in four general steps:

1. Scanning and Parsing
2. Query Optimization or planning the execution strategy
3. Query Code Generator (interpreted or compiled)
4. Execution in the runtime database processor

1. Scanning and Parsing

* When a query is first submitted (via an applications program), it must be scanned and parsed to determine if the query consists of appropriate syntax.
* **Scanning** is the process of converting the query text into a tokenized representation.
* The tokenized representation is more compact and is suitable for processing by the parser.
* This representation may be in a tree form.
* The **Parser** checks the tokenized representation for correct syntax.
* In this stage, checks are made to determine if columns and tables identified in the query exist in the database and if the query has been formed correctly with the appropriate keywords and structure.
* If the query passes the parsing checks, then it is passed on to the Query Optimizer.

2. Query Optimization or Planning the Execution Strategy

* For any given query, there may be a number of different ways to execute it.
* Each operation in the query (SELECT, JOIN, etc.) can be implemented using one or more different *Access Routines*.
* For example, an access routine that employs an index to retrieve some rows would be more efficient that an access routine that performs a full table scan.
* The goal of the **query optimizer** is to find a *reasonably efficient* strategy for executing the query (not quite what the name implies) using the access routines.
* Optimization typically takes one of two forms: *Heuristic Optimization* or *Cost Based Optimization*
* In **Heuristic Optimization**, the query execution is refined based on *heuristic rules* for reordering the individual operations.
* With **Cost Based Optimization**, the overall cost of executing the query is systematically reduced by estimating the costs of executing several different execution plans.

3. Query Code Generator (interpreted or compiled)

* Once the query optimizer has determined the execution plan (the specific ordering of access routines), the code generator writes out the actual access routines to be executed.
* With an interactive session, the query code is interpreted and passed directly to the runtime database processor for execution.
* It is also possible to *compile* the access routines and store them for later execution.

4. Execution in the runtime database processor

* At this point, the query has been scanned, parsed, planned and (possibly) compiled.
* The runtime database processor then executes the access routines against the database.
* The results are returned to the application that made the query in the first place.
* Any runtime errors are also returned.

**Distributed Database Design**

|  |  |
| --- | --- |
|  | [Data Fragmentation](https://www.dlsweb.rmit.edu.au/Toolbox/knowmang/content/distributed_sys/ddms_design.htm#Data%20Fragmentation) |
|  | [Data Replication](https://www.dlsweb.rmit.edu.au/Toolbox/knowmang/content/distributed_sys/ddms_design.htm#Data%20Replication) |
|  | [Data Allocation](https://www.dlsweb.rmit.edu.au/Toolbox/knowmang/content/distributed_sys/ddms_design.htm#Data%20Allocation) |
|  | [Client/Server Architecture](https://www.dlsweb.rmit.edu.au/Toolbox/knowmang/content/distributed_sys/ddms_design.htm#Client%20Server%20Architecture) |

In this section, you will be introduced to distributed database design issues.  These include data fragmentation, data replication and data allocation.  We will also look briefly at how distributed database capabilities are implemented within a client/server architecture.

**Data Fragmentation**[top of page](https://www.dlsweb.rmit.edu.au/Toolbox/knowmang/content/distributed_sys/ddms_design.htm#top)

Data fragmentation is a technique used to break up objects.  In designing a distributed database, you must decide which portion of the database is to be stored where.  One technique used to break up the database into logical units called fragments.  Fragmentation information is stored in a distributed data catalogue which the processing computer uses to process a user's request.

As a point of discussion, we can look at data fragmentation in terms of relations or tables. The following matrix describes the different types of fragmentation that can be used.

|  |  |
| --- | --- |
| **Horizontal fragmentation** | This type of fragmentation refers division of a relation into fragments of rows. Each fragment is stored at a different computer or node, and each fragment contains unique rows. Each horizontal fragment may have a different number of rows, but each fragment must have the same attributes. |
| **Vertical fragmentation** | This type of fragmentation refers to the division of a relation into fragments that comprise a collection of attributes. Each vertical fragment must have the same number of rows, but can have different attributes depending on the key. |
| **Mixed fragmentation** | This type of fragmentation is a two-step process. First, horizontal fragmentation is done to obtain the necessary rows, then vertical fragmentation is done to divide the attributes among the rows. |

**Data Replication**[top of page](https://www.dlsweb.rmit.edu.au/Toolbox/knowmang/content/distributed_sys/ddms_design.htm#top)

Data replication is the storage of data copies at multiple sites on the network. Fragment copies can be stored at several site, thus enhancing data availability and response time. Replicated data is subject to a **mutual consistency rule**. This rule requires that all copies of the data fragments must be identical and to ensure data consistency among all of the replications.

Although data replication is beneficial in terms of availability and response times, the maintenance of the replications can become complex. For example, if data is replicated over multiple sites, the DDBMS must decide which copy to access. For a query operation, the nearest copy is all that is required to satisfy a transaction. However, if the operation is an update, then all copies must be selected and updated to satisfy the mutual consistency rule.

A database can be either **fully replicated, partially replicated**or**unreplicated.**

|  |  |
| --- | --- |
| **Full replication** | Stores multiple copies of each database fragment at multiple sites. Fully replicated databases can be impractical because of the amount of overhead imposed on the system. |
| **Partial replication** | Stores multiple copies of some database fragments at multiple sites. Most DDBMS can handle this type of replication very well. |
| **No replication** | Stores each database fragment at a single site. No duplication occurs. |

Data replication is particularly useful if usage frequency of remote data is high and the database is fairly large. Another benefit of data replication is the possibility of restoring lost data at a particular site.

**Data Allocation**[top of page](https://www.dlsweb.rmit.edu.au/Toolbox/knowmang/content/distributed_sys/ddms_design.htm#top)

Data allocation is a process of deciding where to store the data. It also involves a decision as to which data is stored at what location. Data allocation can be **centralised, partitioned**or **replicated.**

|  |  |
| --- | --- |
| **Centralised** | The entire database is stored at one site. No distribution occurs. |
| **Partitioned** | The database is divided into several fragments that are stored at several sites. |
| **Replicated** | Copies of one or more database fragments are stored at several sites. |

**Client Server Architecture**[top of page](https://www.dlsweb.rmit.edu.au/Toolbox/knowmang/content/distributed_sys/ddms_design.htm#top)

Implementation of a distributed database system must be carefully managed within a client server architecture. Typically, the server provides the resources for the client to use. The client receives the request from the user and the request is passed to the server. The server receives, schedules and executes the requests, selecting only what the client requires. The request is sent only when the client requests it.

There are advantages to implementing a distributed database system using client server architecture:

|  |  |
| --- | --- |
| **Cost** | Client server systems are less expensive that mainframes. There is also a considerable cost savings in off-loading applications development from the mainframe to PCs |
| **PC Functionality and Use** | Many users are more familiar and more skilled with PC technology than they are with mainframe technology. The use of PC technology is more widespread in the workplace. |
| **Data Analysis and Query Tools** | These tools are readily available in the marketplace and can be used with many database systems. |

There are some disadvantages however, in that the client server environment is more complex and thus requires more management resources. Security is also another issue because of the number of users and sites.

[top of page](https://www.dlsweb.rmit.edu.au/Toolbox/knowmang/content/distributed_sys/ddms_design.htm#top)

**Concepts of Distributed databases**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 03 Jun, 2021

A Distributed database is defined as a logically related collection of data that is shared which is physically distributed over a computer network on different sites.

**Distributed DBMS :**  
The Distributed DBMS is defined as, the software that allows for the management of the distributed database and make the distributed data available for the users.  
A distributed DBMS consist of a single logical database that is divided into a number of pieces called the fragments. In DDBMS, Each site is capable of independently processing the users request.

Users can access the DDBMS via applications classified:

1. **Local Applications –**  
   Those applications that doesn’t require data from the other sites are classified under the category of Local applications.
2. **Global Applications –**  
   Those applications that require data from the other sites are classified under the category of Global applications.

**Characteristics of Distributed DDBMS :**  
A DDBMS has the following characteristics-

1. A collection of logically related shared data.
2. The data is split into a number of fragments.
3. Fragments may be duplicate.
4. Fragments are allocated to sites.
5. The data at each site is under the control of DBMS and managed by DBMS.

**Distributed Processing :**  
The Distributed processing is centralized database that can be accessed over a computer network by different sites. The data is centralized even though other users may be accessing the data from the other sites, we do not consider this to be DDBMS, simply distributed processing.

**Parallel DBMS :**  
A parallel DBMS is a DBMS that run across multiple processor and is designed to execute operations in parallel, whenever possible. The parallel DBMS link a number of smaller machines to achieve same throughput as expected from a single large machine.

There are three main architectures for Parallel DBMS-

1. **Shared Memory –**  
   Shared memory is a highly coupled architecture in which a number of processors within a single system who share system memory. It is also known as symmetric multiprocessing (SMP). This approach is more popular on platforms like personal workstations that support a few microprocessor in parallel.
2. **Shared Disk –**  
   Shared disk is a loosely coupled architecture used for application that are centralized and require a high availability and performance.Each processor is able to access all disks directly, but each has it’s own private memory.It is also called Clusters.
3. **Shared Nothing –**  
   Shared nothing is a multiple processor architecture in which every processor is a part of a complete system, which has its own memory and disk storage( has it’s own resources). It is also called Massively Parallel Processing (MPP).

# Web-interface

**(1) The interaction between a user and software running on a Web server. The user interface is the Web browser and the Web page it downloaded and rendered. See**[**Web application**](https://www.yourdictionary.com/web-application)**and**[**Web server**](https://www.yourdictionary.com/web-server)**.**

**What is Web Interface?**

The simplest type of web management is your web user interface or interface web. This is due to the point-and-select capabilities enabling you to jumpstart your firewall management. To define web interface, it is a straightforward interface with configuration options located on your browser’s left-hand side. Your menu is either based on the DHTML or dynamic hypertext markup language default or is Java-based. There is no difference in functionality but there are slight variations to the look and feel.

To define a web interface, you must understand it is configured by default to only work over your HTTP or hypertext transfer protocol. You can also configure your web interface for working over your HTTPS or hypertext transfer protocol secure. This offers you the mechanism required for securing your web management traffic. You can use the majority of popular web browsers with your website interface including:

* Firefox
* Internet Explorer
* Chrome

(Firefox interface, ghacks.net)

The question of what is a web interface can be answered by a mechanism enabling you to interact with the software or content you are running through your web browser on a remote server. Your web server downloads your web page content enabling you to use your browser to interact with the content. Your browser performs the function of a client. you can store your content on a remote server due to the distributed nature of your web browser in addition to receiving convenient content access.

There is a tremendous amount of data currently in existence due to these kinds of interfaces. The most frequently used web applications include:

* Webmail
* Instant messaging
* Online shopping
* Online document sharing
* Social media

**How to Login to Your Router’s Web Interface**

Once you understand what a website interface is, you need to learn how to login to your router’s website interface. This is essential for configuring your advanced settings. You can perform several different configurations including:

* Manually setting up your router
* Forwarding ports for your video camera and gaming console
* Setting up wireless-security for your network

Using a hardwired connection between your computer and router is recommended to access the web interface of your router. In addition to ensuring you have a stable connection, you will eliminate any potential access to other routers throughout your area. Some routers have an auto-sensing capability not requiring you to use straight-through or crossover cables. If you are accessing the web interface through a wireless device, make certain you have connected your router to your device before proceeding.

The steps required will vary slightly depending on the version and model of your router and the type of operating system for your computer. Before following the steps detailed below, the power for your router must be turned on.

(Web interface login, tp-link.com)

**XML**

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions −

* **XML is extensible** − XML allows you to create your own self-descriptive tags, or language, that suits your application.
* **XML carries the data, does not present it** − XML allows you to store the data irrespective of how it will be presented.
* **XML is a public standard** − XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

XML Usage

A short list of XML usage says it all −

* XML can work behind the scene to simplify the creation of HTML documents for large web sites.
* XML can be used to exchange the information between organizations and systems.
* XML can be used for offloading and reloading of databases.
* XML can be used to store and arrange the data, which can customize your data handling needs.
* XML can easily be merged with style sheets to create almost any desired output.
* Virtually, any type of data can be expressed as an XML document.

**What is Markup?**

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable. So *what exactly is a markup language?* Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

Following example shows how XML markup looks, when embedded in a piece of text −

<message>

<text>Hello, world!</text>

</message>

This snippet includes the markup symbols, or the tags such as <message>...</message> and <text>... </text>. The tags <message> and </message> mark the start and the end of the XML code fragment. The tags <text> and </text> surround the text Hello, world!.

Is XML a Programming Language?

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instruct the computer to perform specific tasks. XML does not qualify to be a programming language as it does not perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

# Spatial Databases

[Database](https://www.tutorialspoint.com/questions/category/Database)[MCA](https://www.tutorialspoint.com/questions/category/MCA)

Spatial data is associated with geographic locations such as cities,towns etc. A spatial database is optimized to store and query data representing objects. These are the objects which are defined in a geometric space.

## Characteristics of Spatial Database

A spatial database system has the following characteristics

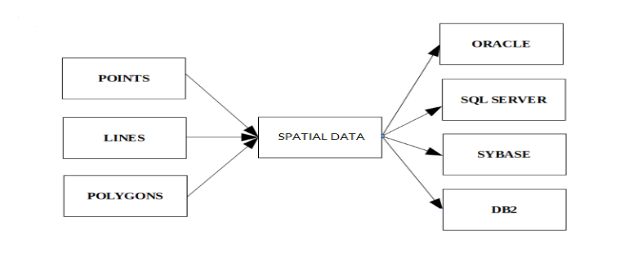
* It is a database system
* It offers spatial data types (SDTs) in its data model and query language.
* It supports spatial data types in its implementation, providing at least spatial indexing and efficient algorithms for spatial join.

## Example

A road map is a visualization of geographic information. A road map is a 2-dimensional object which contains points, lines, and polygons that can represent cities, roads, and political boundaries such as states or provinces.

In general, spatial data can be of two types −

* Vector data: This data is represented as discrete points, lines and polygons
* Rastor data: This data is represented as a matrix of square cells.



The spatial data in the form of points, lines, polygons etc. is used by many different databases as shown above. What is data mining?

**Data mining**, also known as knowledge discovery in data (KDD), is the process of uncovering patterns and other valuable information from large data sets. Given the evolution of [data warehousing](https://www.ibm.com/cloud/learn/data-warehouse) technology and the growth of big data, adoption of data mining techniques has rapidly accelerated over the last couple of decades, assisting companies by transforming their raw data into useful knowledge. However, despite the fact that that technology continuously evolves to handle data at a large-scale, leaders still face challenges with scalability and automation.

Data mining has improved organizational decision-making through insightful data analyses. The data mining techniques that underpin these analyses can be divided into two main purposes; they can either describe the target dataset or they can predict outcomes through the use of [machine learning](https://www.ibm.com/cloud/learn/machine-learning) algorithms. These methods are used to organize and filter data, surfacing the most interesting information, from fraud detection to user behaviors, bottlenecks, and even security breaches.

When combined with data analytics and visualization tools, like [Apache Spark](https://www.ibm.com/cloud/learn/apache-spark), delving into the world of data mining has never been easier and extracting relevant insights has never been faster. Advances within [artificial intelligence](https://www.ibm.com/cloud/learn/what-is-artificial-intelligence) only continue to expedite adoption across industries.

Data mining process

The data mining process involves a number of steps from data collection to visualization to extract valuable information from large data sets. As mentioned above, data mining techniques are used to generate descriptions and predictions about a target data set. Data scientists describe data through their observations of patterns, associations, and correlations. They also classify and cluster data through classification and regression methods, and identify outliers for use cases, like spam detection.

Data mining usually consists of four main steps: setting objectives, data gathering and preparation, applying data mining algorithms, and evaluating results.

**1. Set the business objectives:** This can be the hardest part of the data mining process, and many organizations spend too little time on this important step. Data scientists and business stakeholders need to work together to define the business problem, which helps inform the data questions and parameters for a given project. Analysts may also need to do additional research to understand the business context appropriately.

**2. Data preparation:** Once the scope of the problem is defined, it is easier for data scientists to identify which set of data will help answer the pertinent questions to the business. Once they collect the relevant data, the data will be cleaned, removing any noise, such as duplicates, missing values, and outliers. Depending on the dataset, an additional step may be taken to reduce the number of dimensions as too many features can slow down any subsequent computation. Data scientists will look to retain the most important predictors to ensure optimal accuracy within any models.

**3. Model building and pattern mining:** Depending on the type of analysis, data scientists may investigate any interesting data relationships, such as sequential patterns, association rules, or correlations. While high frequency patterns have broader applications, sometimes the deviations in the data can be more interesting, highlighting areas of potential fraud.

[Deep learning](https://www.ibm.com/cloud/learn/deep-learning) algorithms may also be applied to classify or cluster a data set depending on the available data. If the input data is labelled (i.e. [supervised learning](https://www.ibm.com/cloud/learn/supervised-learning)), a classification model may be used to categorize data, or alternatively, a regression may be applied to predict the likelihood of a particular assignment. If the dataset isn’t labelled (i.e. [unsupervised learning](https://www.ibm.com/cloud/learn/unsupervised-learning)), the individual data points in the training set are compared with one another to discover underlying similarities, clustering them based on those characteristics.

**4. Evaluation of results and implementation of knowledge:** Once the data is aggregated, the results need to be evaluated and interpreted. When finalizing results, they should be valid, novel, useful, and understandable. When this criteria is met, organizations can use this knowledge to implement new strategies, achieving their intended objectives.

Data mining techniques

Data mining works by using various algorithms and techniques to turn large volumes of data into useful information. Here are some of the most common ones:

**Association rules:** An association rule is a rule-based method for finding relationships between variables in a given dataset. These methods are frequently used for market basket analysis, allowing companies to better understand relationships between different products. Understanding consumption habits of customers enables businesses to develop better cross-selling strategies and recommendation engines.

**Neural networks:** Primarily leveraged for deep learning algorithms, [neural networks](https://www.ibm.com/cloud/learn/neural-networks) process training data by mimicking the interconnectivity of the human brain through layers of nodes. Each node is made up of inputs, weights, a bias (or threshold), and an output. If that output value exceeds a given threshold, it “fires” or activates the node, passing data to the next layer in the network. Neural networks learn this mapping function through supervised learning, adjusting based on the loss function through the process of gradient descent. When the cost function is at or near zero, we can be confident in the model’s accuracy to yield the correct answer

**Decision tree:** This data mining technique uses classification or regression methods to classify or predict potential outcomes based on a set of decisions. As the name suggests, it uses a tree-like visualization to represent the potential outcomes of these decisions.

**K- nearest neighbor (KNN):** K-nearest neighbor, also known as the KNN algorithm, is a non-parametric algorithm that classifies data points based on their proximity and association to other available data. This algorithm assumes that similar data points can be found near each other. As a result, it seeks to calculate the distance between data points, usually through Euclidean distance, and then it assigns a category based on the most frequent category or average.

Data mining applications

Data mining techniques are widely adopted among business intelligence and data analytics teams, helping them extract knowledge for their organization and industry. Some data mining use cases include:

Sales and marketing

Companies collect a massive amount of data about their customers and prospects. By observing consumer demographics and online user behavior, companies can use data to optimize their marketing campaigns, improving segmentation, cross-sell offers, and customer loyalty programs, yielding higher ROI on marketing efforts. Predictive analyses can also help teams to set expectations with their stakeholders, providing yield estimates from any increases or decreases in marketing investment.

Education

Educational institutions have started to collect data to understand their student populations as well as which environments are conducive to success. As courses continue to transfer to online platforms, they can use a variety of dimensions and metrics to observe and evaluate performance, such as keystroke, student profiles, classes, universities, time spent, etc.

Operational optimization

[Process mining](https://www.ibm.com/cloud/learn/process-mining) leverages data mining techniques to reduce costs across operational functions, enabling organizations to run more efficiently. This practice has helped to identify costly bottlenecks and improve decision-making among business leaders.

Fraud detection

While frequently occurring patterns in data can provide teams with valuable insight, observing data anomalies is also beneficial, assisting companies in detecting fraud. While this is a well-known use case within banking and other financial institutions, SaaS-based companies have also started to adopt these practices to eliminate fake user accounts from their datasets.

**What Is Data Warehousing?**

Data warehousing is the secure electronic storage of information by a business or other organization. The goal of data warehousing is to create a trove of historical data that can be retrieved and analyzed to provide useful insight into the organization's operations.

Data warehousing is a vital component of [business intelligence](https://www.investopedia.com/terms/b/business-intelligence-bi.asp). That wider term encompasses the information infrastructure that modern businesses use to track their past successes and failures and inform their decisions for the future.

* Data warehousing is the storage of information over time by a business or other organization.
* New data is periodically added by people in various key departments such as marketing and sales.
* The warehouse becomes a library of historical data that can be retrieved and analyzed in order to inform decision-making in the business.
* The key factors in building an effective data warehouse include defining the information that is critical to the organization and identifying the sources of the information.
* A database is designed to supply real-time information. A data warehouse is designed as an archive of historical information.

How Data Warehousing Works

The need to warehouse data evolved as businesses began relying on computer systems to create, file, and retrieve important business documents. The concept of data warehousing was introduced in 1988 by IBM researchers Barry Devlin and Paul Murphy.1

Data warehousing is designed to enable the analysis of historical data. Comparing data consolidated from multiple heterogeneous sources can provide insight into the performance of a company. A data warehouse is designed to allow its users to run queries and analyses on historical data derived from transactional sources.

Data added to the warehouse do not change and cannot be altered. The warehouse is the source that is used to run [analytics](https://www.investopedia.com/terms/d/data-analytics.asp) on past events, with a focus on changes over time. Warehoused data must be stored in a manner that is secure, reliable, easy to retrieve, and easy to manage.

Maintaining the Data Warehouse

There are certain steps that are taken to maintain a data warehouse. One step is data extraction, which involves gathering large amounts of data from multiple source points. After a set of data has been compiled, it goes through data cleaning, the process of combing through it for errors and correcting or excluding any that are found.

The cleaned-up data are then converted from a database format to a warehouse format. Once stored in the warehouse, the data goes through sorting, consolidating, and summarizing, so that it will be easier to use. Over time, more data are added to the warehouse as the various data sources are updated.

A key book on data warehousing is W. H. Inmon's "Building the Data Warehouse," a practical guide that was first published in 1990 and has been reprinted several times.Today, businesses can invest in cloud-based data warehouse software services from companies including Microsoft, Google, Amazon, and Oracle, among others.2

**What is a Decision Support System (DSS)?**

A decision support system (DSS) is an information system that aids a business in decision-making activities that require judgment, determination, and a sequence of actions. The information system assists the mid- and high-level management of an organization by analyzing huge volumes of unstructured data and accumulating information that can help to solve problems and help in decision-making. A DSS is either human-powered, automated, or a combination of both.

**Purpose of a Decision Support System**

A decision support system produces detailed information reports by gathering and analyzing data. Hence, a DSS is different from a normal operations application, whose goal is to collect data and not analyze it.

In an organization, a DSS is used by the planning departments – such as the operations department – which collects data and creates a report that can be used by managers for decision-making. Mainly, a DSS is used in sales projection, for [inventory](https://corporatefinanceinstitute.com/resources/knowledge/accounting/inventory/) and operations-related data, and to present information to customers in an easy-to-understand manner.

Theoretically, a DSS can be employed in various knowledge domains from an organization to forest management and the medical field. One of the main applications of a DSS in an organization is real-time reporting. It can be very helpful for organizations that take part in [just-in-time (JIT)](https://corporatefinanceinstitute.com/resources/knowledge/accounting/just-in-time-jit-method/) inventory management.

In a JIT inventory system, the organization requires real-time data of their inventory levels to place orders “just in time” to prevent delays in production and cause a negative domino effect. Therefore, a DSS is more tailored to the individual or organization making the decision than a traditional system.

**Components of a Decision Support System**

The three main components of a DSS framework are:

**1. Model Management System**

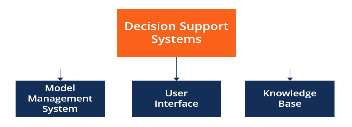
The model management system S=stores models that managers can use in their decision-making. The models are used in decision-making regarding the financial health of the organization and forecasting demand for a good or service.

**2. User Interface**

The user interface includes tools that help the end-user of a DSS to navigate through the system.

**3. Knowledge Base**

The knowledge base includes information from internal sources (information collected in a transaction process system) and external sources (newspapers and online databases).



**Types of Decision Support Systems**

* **Communication-driven**: Allows companies to support tasks that require more than one person to work on the task. It includes integrated tools such as Microsoft SharePoint Workspace and Google Docs.
* **Model-driven**: Allows access to and the management of financial, organizational, and statistical models. Data is collected, and parameters are determined using the information provided by users. The information is created into a decision-making model to analyze situations. An example of a model-driven DSS is Dicodess – an open-source model-driven DSS.
* **Knowledge-driven**: Provides factual and specialized solutions to situations using stored facts, procedures, rules, or interactive decision-making structures like [flowcharts](https://corporatefinanceinstitute.com/resources/templates/presentations/flowchart-templates/).
* **Document-driven**: Manages unstructured information in different electronic formats.
* **Data-driven**: Helps companies to store and analyze internal and external data.

**Advantages of a Decision Support System**

* A decision support system increases the speed and efficiency of decision-making activities. It is possible, as a DSS can collect and analyze real-time data.
* It promotes training within the organization, as specific skills must be developed to implement and run a DSS within an organization.
* It automates monotonous managerial processes, which means more of the manager’s time can be spent on decision-making.
* It improves [interpersonal communication](https://corporatefinanceinstitute.com/resources/careers/soft-skills/communication/) within the organization.

**Disadvantages of a Decision Support System**

* The cost to develop and implement a DSS is a huge capital investment, which makes it less accessible to smaller organizations.
* A company can develop a dependence on a DSS, as it is integrated into daily decision-making processes to improve efficiency and speed. However, managers tend to rely on the system too much, which takes away the subjectivity aspect of decision-making.
* A DSS may lead to [information overload](https://www.forbes.com/sites/laurashin/2014/11/14/10-steps-to-conquering-information-overload/) because an information system tends to consider all aspects of a problem. It creates a dilemma for end-users, as they are left with multiple choices.
* Implementation of a DSS can cause fear and backlash from lower-level employees. Many of them are not comfortable with new technology and are afraid of losing their jobs to technology.

RDBMS and OODBMS are database management systems. RDBMS uses tables to represent data and their relationships whereas OODBMS represents data in form of objects similar to Object Oriented Programming.

| **Sr. No.** | **Key** | **RDBMS/traditional database** | **OODBMS** |
| --- | --- | --- | --- |
| 1 | Definition | RDBMS stands for Relational DataBase Management System. | OODBMS stands for Object Oriented DataBase Management System. |
| 2 | Data Management | Data is stored as entities defined in tabular format. | Data is stored as objects. |
| 3 | Data Complexity | RDBMS handles simple data. | OODBMS handles large and complex data. |
| 4 | Term | An entity refers to collection of similar items having same definition. | An class refers to group of objects having common relationships, behaviors and properties. |
| 5 | Data Handling | RDBMS handles only data. | OODBMS handles both data and functions operating on that data. |
| 6 | Objective | To keep data independent from application program. | To implement data encapsulation. |
| 7 | Key | A primary key identifies in object in a table uniquely. | Object Id, OID represents an object uniquely in group of objects. |

**RDBMS:**  
RDBMS stands for Relational Database Management System. It is a database management system based on the relational model i.e. the data and relationships are represented by a collection of inter-related tables. It is a DBMS that enables the user to create, update, administer and interact with a relational database. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

**OODBMS:**  
OODBMS stands for Object-Oriented Database Management System. It is a DBMS where data is represented in the form of objects, as used in object-oriented programming. OODB implements object-oriented concepts such as classes of objects, object identity, polymorphism, encapsulation, and inheritance. An object-oriented database stores complex data as compared to relational database. Some examples of OODBMS are Versant Object Database, Objectivity/DB, ObjectStore, Caché and ZODB.

There is no standard definition of what capabilities or attributes define either an Extended Relational Database Management System or an Object-Relational Database Management System (ORDBMS). Here, both are referred to as “Extended Relational.”

**ERDBMS**ERDBMSs have characteristics of both an RDBMS and an ODBMS (thus, the loose application of the ORDBMS label).  ERDBMS products provide a relational data model and query language that have been extended to include many of the features that are typical of ODBMSs.

**Features** Extending the query language allows for the integration of well-understood query optimization techniques.  Typically, programming capabilities are embedded in the query language.  This capability is not to be confused with stored procedures that are provided by a number of relational vendors.  With the ERDBMSs, programmers are able to write functions in conventional languages as well as in SQL.  These functions can then be embedded in standard SQL statements in exactly the same manner as a DBMS vendor function (e.g., the Sybase getdate() function).Most significantly, these DBMSs have been extended to handle:  complex data types, which include user defined abstract data types,

non-tabular structure

automatically generated, logical object identity

tables within tables

, a type hierarchy

·          multiple inheritance,

·          compound objects,

·          schema evolution,

·          transitive closure operations.

**Risks** The mapping of the object model can, in some cases, be greatly simplified (e.g., indentured parts list) with ERDBMS products.  But, mapping is still required at both the model and the language level.  This raises risk and development cost considerations similar to those of an RDBMS.The cost of having to actually implement the mapping software can be mitigated through the ODBC interface, which most of these products support.  However, some of the extended features of the ERDBMS are not visible through an ODBC interface.  Thus, custom interface code is frequently required to fully utilize the services provided.  Again, risk and development costs must be considered.Third party or vendor tools are available to handle the processing of some of the more common complex data types (e.g., time series, spatial, text, image, sound).  But if such software is not available for a required data type, the good news is that it can be written.  The bad news is that this custom code is frequently non-trivial to write. Writing complex custom code can add significant risk and development costs to the project.  Also consider the question of the instability that can be introduced to the ERDBMS through these custom software components.

| **BASIS FOR COMPARISON** | **OLTP** | **OLAP** |
| --- | --- | --- |
| Basic | It is an online transactional system and manages database modification. | It is an online data retrieving and data analysis system. |
| Focus | Insert, Update, Delete information from the database. | Extract data for analyzing that helps in decision making. |
| Data | OLTP and its transactions are the original source of data. | Different OLTPs database becomes the source of data for OLAP. |
| Transaction | OLTP has short transactions. | OLAP has long transactions. |
| Time | The processing time of a transaction is comparatively less in OLTP. | The processing time of a transaction is comparatively more in OLAP. |
| Queries | Simpler queries. | Complex queries. |
| Normalization | Tables in OLTP database are normalized (3NF). | Tables in OLAP database are not normalized |

**Temporal Database Concepts**

 Temporal databases, in the broadest sense, encompass all database applications that require some aspect of time when organizing their information. Hence, they provide a good example to illustrate the need for developing a set of unifying concepts for application developers to use. Temporal database applications have been

developed since the early days of database usage. However, in creating these applications, it is mainly left to the application designers and developers to discover, design, program, and implement the temporal concepts they need. There are many examples of applications where some aspect of time is needed to maintain the information in a database. These include *healthcare,* where patient histories need to be maintained; *insurance,* where claims and accident histories are required as well as information about the times when insurance policies are in effect; *reservation systems*in general (hotel, airline, car rental, train, and so on), where information on thedates and times when reservations are in effect are required; *scientific databases,* where data collected from experiments includes the time when each data is measured; Temporal databases: The complexities of time

A temporal database is generally understood as a database capable of supporting storage and reasoning of time-based data. For example, medical applications may be able to benefit from temporal database support â€” a record of a patient's medical history has little value unless the test results, e.g. the temperatures, are associated to the times at which they are valid, since we may wish to do reasoning about the periods in time in which the patientâ€™s temperature changed.

Temporal databases store temporal data, i.e. data that is time dependent (timevarying). Typical temporal database scenarios and applications include time-dependent/time-varying economic data, such as:

* Share prices
* Exchange rate
* Interest rates
* Company profits

The desire to model such data means that we need to store not only the respective value but also an associated date or a time period for which the value is valid. Typical queries expressed informally might include:

* Give me last month's history of the Dollar-Pound Sterling exchange rate.
* Give me the share prices of the NYSE on October 17, 1996.

Many companies offer products whose prices vary over time. Daytime telephone calls, for example, are usually more expensive than evening or weekend calls. Travel agents, airlines or ferry companies distinguish between high and low seasons. Sports centres offer squash or tennis courts at cheaper rate during the day. Hence, prices are time dependent in these examples. They are typically summarised in tables with prices associated with a time period.

As well as the user of a database recording when certain data values are valid, we may wish to store (for backup, or analysis reasons) historical records of changes made to the database. So each time a change to a database is made the system may automatically store a transaction timestamp. Therefore a temporal database may be storing two different pieces of time data for a tuple â€” the user-defined period of time for which the data is valid (e.g. October to April [winter season] rental of tennis courts are 1 US dollar per hour), and the system-generated transaction timestamp for when the tuple (or part of a tuple) was changed (e.g. 14:55 on 03/01/1999). A temporal database ability to store these different kinds of data makes possible many different kinds of temporal-based queries, as long as its query language and data model are sophisticated enough to formally model and allow reasoning about temporal data. It is the possible gains from such temporal querying facilities that has provided the motivation for research and development into extending the Relational database model for temporal data (and suggesting alternatives to the Relational modelâ€¦).

Our everyday life is very often influenced by timetables for buses, trains, flights, university lectures, laboratory access and even cinema, theatre or TV programmes. As one consequence, many people plan their daily activities by using diaries, which themselves are a kind of timetable. Timetables or diaries can be regarded as temporal relations in terms of a Relational temporal data model. Medical diagnosis often draws conclusions from a patient's history, i.e. from the evolution of his/her illness. The latter is described by a series of values, such as the body temperature, cholesterol concentration in the blood, blood pressure, etc. As in the first example, each of these values is only valid during a certain period of time (e.g. a certain day). Typically a doctor would retrieve a patient's values' history, analyse trends and base diagnosis on such observations. Similar examples can be found in many areas that rely on the observation of evolutionary processes, such as environmental studies, economics and many natural sciences.

Concepts of time

Continuous or discrete

From a philosophical point of view we might argue either that time passes continuously, flowing as if a stream or river of water, or we can think of time passing in discrete units of time, each with equal duration, as we think of time when listening to the ticking of a clock. For the purposes of recording and reasoning about time, many people prefer to work with a conceptual model of time as being discrete and passing in small, measurable units; however, there are some occasions and applications where a continuous model of time is most appropriate.

Granularity

When we think about time as passing in discrete units, depending on the purpose or application, different-sized units may be appropriate. So the age of volcanoes may be measured in years, or decades, or hundreds of years. The age of motor cars may be measured in years, or perhaps months for â€˜youngâ€™ cars. The age of babies may be measured in years, or months, or weeks, or days. The age of bacteria in seconds or milliseconds. The size of the units of time used to refer to a particular scenario is referred to as the granularity of the temporal units â€” small temporal grains refer to short units of time (days, hours, seconds, milliseconds, etc), and large temporal grains refer to longer units of time (months, years, decades, etc).

Time quanta

For a particular situation we may wish to define the smallest unit of time which can be recorded or reasoned about. One way to refer to the chosen, indivisible unit of time is as 'time quanta'. Sometimes the term â€˜chrononâ€™ or â€˜time granuleâ€™ is used to refer to the indivisible units of time quanta for a situation or system. In this chapter we shall use the term time quanta.

Timelines, points, duration and intervals

When attempting to represent and reason about time, four important concepts are:

* **Points:** Formally a point in time has no duration; it simply refers to a particular position in the timeline under discussion. We can talk of the point in time at which some event begins or ends.
* **Duration:** A duration refers to a number of time quanta; for example, a week, two months, three years and 10 seconds are all durations. A duration refers to a particular magnitude (size) of a part of a timeline, but not the direction (so whether we talk of a week ago or a week beginning in three days' time, we are still referring to a length of time of a week).
* **Interval:** An interval has a start time point and an end time point. Using more formal notation, we can refer to an interval I(s, e) with start point â€˜sâ€™ and end point â€˜eâ€™, and for which all points referring to time from s to e (inclusive) make up the interval. Note that there is an assumption (constraint) that the timepoint â€˜sâ€™ does not occur after the timepoint â€˜eâ€™ (an interval of zero time quanta would have a start point and end point that were equal).
* **Timeline:** Conceptually we can often imagine time as moving along a line in one direction. When graphically representing time, it is usual to draw a line (often with an arrow to show time direction), where events shown towards the end of the timeline have occurred later than those shown towards the beginning of the line. Often a graphical timeline is draw like an axis on a graph (by convention, the X-axis represents time) and the granularity of the time units is marked (and perhaps labelled) along the X-axis.

An XML *document* is a basic unit of XML information composed of elements and other markup in an orderly package. An XML *document* can contains wide variety of data. For example, database of numbers, numbers representing molecular structure or a mathematical equation.

## **XML Document Example**

A simple document is shown in the following example −

<?xml version = "1.0"?>

<contact-info>

<name>Tanmay Patil</name>

<company>TutorialsPoint</company>

<phone>(011) 123-4567</phone>

</contact-info>

The following image depicts the parts of XML document.



## **Document Prolog Section**

**Document Prolog** comes at the top of the document, before the root element. This section contains −

* XML declaration
* Document type declaration

You can learn more about XML declaration in this chapter − [XML Declaration](https://www.tutorialspoint.com/xml/xml_declaration.htm)

## **Document Elements Section**

**Document Elements** are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose. You can separate a document into multiple sections so that they can be rendered differently, or used by a search engine. The elements can be containers, with a combination of text and other elements.

**Lock Based Concurrency Control Protocol in DBMS.**

Now, we all know the four properties a transaction must follow. Yes, you got that right, I mean the [**ACID** properties](https://www.geeksforgeeks.org/acid-properties-in-dbms/). Concurrency control techniques are used to ensure that the *Isolation* (or non-interference) property of concurrently executing transactions is maintained.

*A trivial question I would like to pose in front of you, (I know you must know this but still) why do you think that we should have interleaving execution of transactions if it may lead to problems such as Irrecoverable Schedule, Inconsistency and many more threats.  
Why not just let it be Serial schedules and we may live peacefully, no complications at all.*

Yes, the performance effects the efficiency too much which is not acceptable.  
Hence a Database may provide a mechanism that ensures that the schedules are either conflict or view serializable and recoverable (also preferably cascadeless). Testing for a schedule for Serializability after it has executed is obviously *too late!*  
So we need Concurrency Control Protocols that ensures Serializability .

**Concurrency-control protocols :** allow concurrent schedules, but ensure that the schedules are conflict/view serializable, and are recoverable and maybe even cascadeless.  
These protocols do not examine the precedence graph as it is being created, instead a protocol imposes a discipline that avoids non-seralizable schedules.  
Different concurrency control protocols provide different advantages between the amount of concurrency they allow and the amount of overhead that they impose.  
We’ll be learning some protocols which are important for GATE CS. Questions from this topic is frequently asked and it’s recommended to learn this concept. (At the end of this series of articles I’ll try to list all theoretical aspects of this concept for students to revise quickly and they may find the material in one place.) Now, let’s get going:

Different categories of protocols:

* **Lock Based Protocol**
  + Basic 2-PL
  + Conservative 2-PL
  + Strict 2-PL
  + Rigorous 2-PL
* **Graph Based Protocol**
* **Time-Stamp Ordering Protocol**
* **Multiple Granularity Protocol**
* **Multi-version Protocol**

For GATE we’ll be focusing on the First three protocols.

**Lock Based Protocols –**  
A lock is a variable associated with a data item that describes a status of data item with respect to possible operation that can be applied to it. They synchronize the access by concurrent transactions to the database items. It is required in this protocol that all the data items must be accessed in a mutually exclusive manner. Let me introduce you to two common locks which are used and some terminology followed in this protocol.

**Shared Lock (S):** also known as Read-only lock. As the name suggests it can be shared between transactions because while holding this lock the transaction does not have the permission to update data on the data item. S-lock is requested using lock-S instruction.

1. **Exclusive Lock (X):** Data item can be both read as well as written.This is Exclusive and cannot be held simultaneously on the same data item. X-lock is requested using lock-X instruction.

**Lock Compatibility Matrix –**  


* A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other  
  transactions.
* Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive(X) on the item no other transaction may hold any lock on the item.
* If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. Then the lock is granted.

**Upgrade / Downgrade locks :** A transaction that holds a lock on an item **A** is allowed under certain condition to change the lock state from one state to another.  
Upgrade: A S(A) can be upgraded to X(A) if Ti is the only transaction holding the S-lock on element A.  
Downgrade: We may downgrade X(A) to S(A) when we feel that we no longer want to write on data-item A. As we were holding X-lock on A, we need not check any conditions.  
  
So, by now we are introduced with the types of locks and how to apply them. But wait, just by applying locks if our problems could’ve been avoided then life would’ve been so simple! If you have done Process Synchronization under OS you must be familiar with one consistent problem, starvation and Deadlock! We’ll be discussing them shortly, but just so you know we have to apply Locks but they must follow a set of protocols to avoid such undesirable problems. Shortly we’ll use 2-Phase Locking (2-PL) which will use the concept of Locks to avoid deadlock. So, applying simple locking, we may not always produce Serializable results, it may lead to Deadlock Inconsistency.

**Problem With Simple Locking…**

Consider the Partial Schedule:

|  |  |  |
| --- | --- | --- |
|  | **T1** | **T2** |
| 1 | lock-X(B) |  |
| 2 | read(B) |  |
| 3 | B:=B-50 |  |
| 4 | write(B) |  |
| 5 |  | lock-S(A) |
| 6 |  | read(A) |
| 7 |  | lock-S(B) |
| 8 | lock-X(A) |  |
| 9 | …… | …… |

**Deadlock –** consider the above execution phase. Now, **T1** holds an Exclusive lock over B, and **T2** holds a Shared lock over A. Consider Statement 7, **T2** requests for lock on B, while in Statement 8 **T1** requests lock on A. This as you may notice imposes a **Deadlock** as none can proceed with their execution.

**Starvation –** is also possible if concurrency control manager is badly designed. For example: A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item. This may be avoided if the concurrency control manager is properly designed.

**Fragmentation in Distributed DBMS**

* Difficulty Level : [Expert](https://www.geeksforgeeks.org/expert/)
* Last Updated : 25 Jun, 2021

**Fragmentation** is a process of dividing the whole or full database into various subtables or sub relations so that data can be stored in different systems. The small pieces of sub relations or subtables are called *fragments*. These fragments are called logical data units and are stored at various sites. It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn’t any loss of data).

In the fragmentation process, let’s say, If a table T is fragmented and is divided into a number of fragments say T1, T2, T3….TN. The fragments contain sufficient information to allow the restoration of the original table T. This restoration can be done by the use of UNION or JOIN operation on various fragments. This process is called ***data fragmentation***. All of these fragments are independent which means these fragments can not be derived from others. The users needn’t be logically concerned about fragmentation which means they should not concerned that the data is fragmented and this is called *fragmentation Independence* or we can say *fragmentation transparency*.

**Advantages :**

* As the data is stored close to the usage site, the efficiency of the database system will increase
* Local query optimization methods are sufficient for some queries as the data is available locally
* In order to maintain the security and privacy of the database system, fragmentation is advantageous

**Disadvantages :**Access speeds may be very high if data from different fragments are needed

* If we are using recursive fragmentation, then it will be very expensive

We have 3 methods for data fragmenting of a table:

* **Horizontal fragmentation**
* **Vertical fragmentation**
* **Mixed or Hybrid fragmentation**

Let’s discuss them one by one.

1. **Horizontal fragmentation –**Horizontal fragmentation refers to the process of dividing a table horizontally by assigning each row or (a group of rows) of relation to one or more fragments. These fragments are then be assigned to different sides in the distributed system. Some of the rows or tuples of the table are placed in one system and the rest are placed in other systems. The rows that belong to the horizontal fragments are specified by a condition on one or more attributes of the relation. In relational algebra horizontal fragmentation on table T, can be represented as follows:

**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.

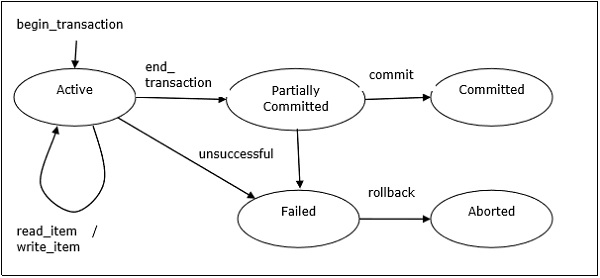
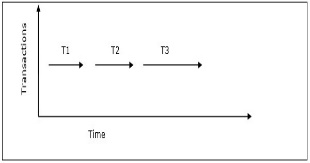
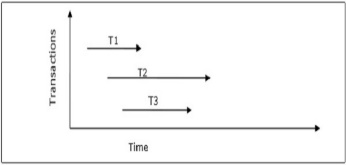
**More Reliable** − In case of database failures, the total system of centralized databases comes to a halt. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.

**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.

Adversities 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.DDBMS - Transaction Processing Systems
* Advertisements
* [Previous Page](https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_query_optimization_distributed_systems.htm)
* [Next Page](https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_controlling_concurrency.htm)
* 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.

**METHODS**

**In**[**object-oriented programming**](https://searchapparchitecture.techtarget.com/definition/object-oriented-programming-OOP)**, a method is a programmed procedure that is defined as part of a**[**class**](https://whatis.techtarget.com/definition/class)**and included in any**[**object**](https://searchapparchitecture.techtarget.com/definition/object)**of that class. A class (and thus an object) can have more than one method. A method in an object can only have access to the data known to that object, which ensures data integrity among the set of objects in an application. A method can be re-used in multiple objects.**