DBMS - CONCURRENCY CONTROL

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories −

- Lock based protocols
- Time stamp based protocols

Lock-based Protocols

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds −

- **Binary Locks** − A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared/exclusive** − This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

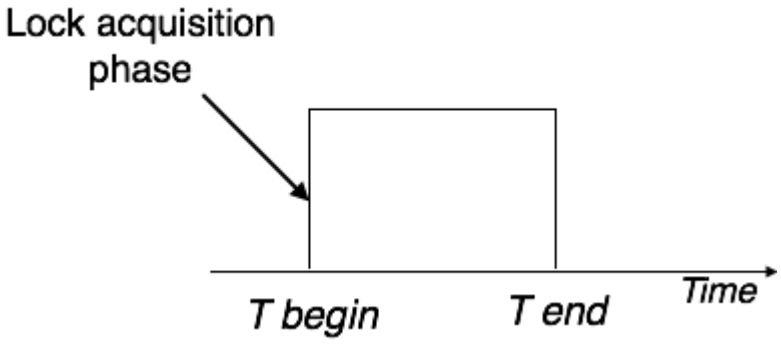
There are four types of lock protocols available −

Simplistic Lock Protocol

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the ‘write’ operation.

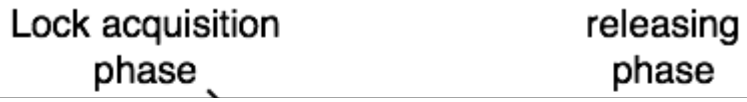
Pre-claiming Lock Protocol

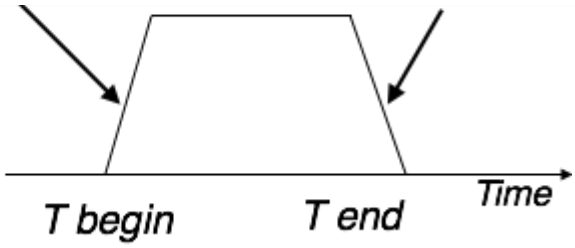
Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.



Two-Phase Locking 2PL

This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.



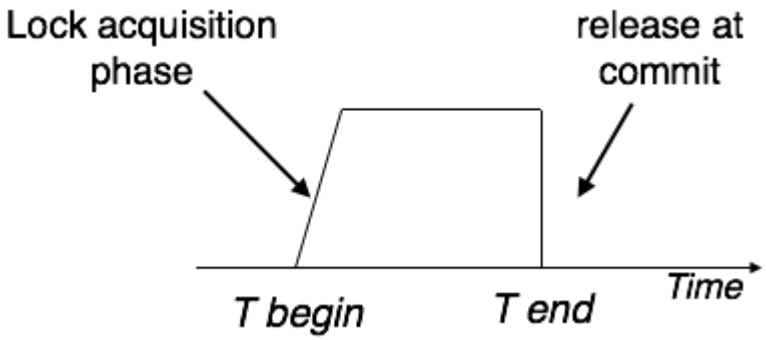


Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.

To claim an exclusive *write* lock, a transaction must first acquire a shared *read* lock and then upgrade it to an exclusive lock.

Strict Two-Phase Locking

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last ‘read and write’ operation was performed on the data item.

Timestamp Ordering Protocol

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

- The timestamp of transaction T_i is denoted as $TS(T_i)$.
- Read time-stamp of data-item X is denoted by R-timestamp$X$.
- Write time-stamp of data-item X is denoted by W-timestamp$X$.

Timestamp ordering protocol works as follows −

- **If a transaction Ti issues a read$X$ operation −**
  - If TSTi < W-timestamp$X$
    - Operation rejected.
  - If TSTi >= W-timestamp$X$
    - Operation executed.
  - All data-item timestamps updated.
- **If a transaction Ti issues a write$X$ operation −**
  - If TSTi < R-timestamp$X$
    - Operation rejected.
  - If TSTi < W-timestamp$X$
    - Operation rejected and Ti rolled back.
  - Otherwise, operation executed.

Thomas' Write Rule

This rule states if TSTi < W-timestamp$X$, then the operation is rejected and T_i is rolled back.

Time-stamp ordering rules can be modified to make the schedule view serializable.

Instead of making T_i rolled back, the 'write' operation itself is ignored.

DBMS - DEADLOCK

In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

For example, assume a set of transactions $\{T_0, T_1, T_2, ...,T_n\}$. T_0 needs a resource X to complete its task. Resource X is held by T_1, and T_1 is waiting for a resource Y, which is held by T_2. T_2 is waiting for resource Z, which is held by T_0. Thus, all the processes wait for each other to release resources. In this situation, none of the processes can finish their task. This situation is known as a deadlock.

Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

Deadlock Prevention

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

Wait-Die Scheme

In this scheme, if a transaction requests to lock a resource *dataitem*, which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur −

- If $TS(T_i) < TS(T_j)$ − that is T_i, which is requesting a conflicting lock, is older than T_j − then T_i is allowed to wait until the data-item is available.
- If $TS(T_i) > TS(t_j)$ − that is T_i is younger than T_j − then T_i dies. T_i is restarted later with a random delay but with the same timestamp.

This scheme allows the older transaction to wait but kills the younger one.

Wound-Wait Scheme

In this scheme, if a transaction requests to lock a resource *dataitem*, which is already held with conflicting lock by some another transaction, one of the two possibilities may occur −

- If $TS(T_i) < TS(T_j)$, then T_i forces T_j to be rolled back − that is T_i wounds T_j. T_j is restarted later with a random delay but with the same timestamp.
- If $TS(T_i) > TS(T_j)$, then T_i is forced to wait until the resource is available.

This scheme, allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.

In both the cases, the transaction that enters the system at a later stage is aborted.

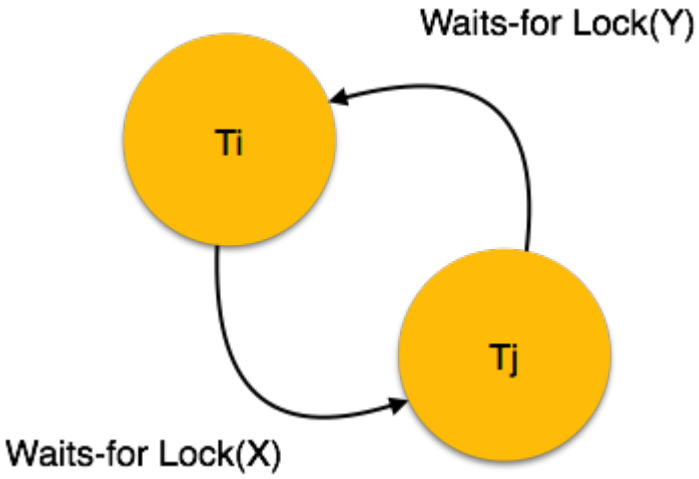
Deadlock Avoidance

Aborting a transaction is not always a practical approach. Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance. Methods like "wait-for graph" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource. In a bulky system, deadlock prevention techniques may work well.

Wait-for Graph

This is a simple method available to track if any deadlock situation may arise. For each transaction entering into the system, a node is created. When a transaction T_i requests for a lock on an item, say X, which is held by some other transaction T_j, a directed edge is created from T_i to T_j. If T_j releases item X, the edge between them is dropped and T_i locks the data item.

The system maintains this wait-for graph for every transaction waiting for some data items held by others. The system keeps checking if there's any cycle in the graph.



Here, we can use any of the two following approaches −

- First, do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.
- The second option is to roll back one of the transactions. It is not always feasible to roll back the younger transaction, as it may be important than the older one. With the help of some relative algorithm, a transaction is chosen, which is to be aborted. This transaction is known as the **victim** and the process is known as **victim selection**.

DBMS - DATA RECOVERY

Crash Recovery

DBMS is a highly complex system with hundreds of transactions being executed every second. The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

Failure Classification

To see where the problem has occurred, we generalize a failure into various categories, as follows −

Transaction failure

A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further. This is called transaction failure where only a few transactions or processes are hurt.

Reasons for a transaction failure could be −

- **Logical errors** − Where a transaction cannot complete because it has some code error or any internal error condition.
- **System errors** − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash

There are problems − external to the system − that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.

Examples may include operating system errors.

Disk Failure

In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

Storage Structure

We have already described the storage system. In brief, the storage structure can be divided into two categories −

- **Volatile storage** − As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
- **Non-volatile storage** − These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile *batterybackedup* RAM.

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened

for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following −

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Log-based Recovery

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows −

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.

```
<T_n, Start>
```

- When the transaction modifies an item X, it write logs as follows −

```
<T_n, X, V_1, V_2>
```

It reads T_n has changed the value of X, from V_1 to V_2.

- When the transaction finishes, it logs −

```
<T_n, commit>
```

The database can be modified using two approaches −

- **Deferred database modification** − All logs are written on to the stable storage and the database is updated when a transaction commits.
- **Immediate database modification** − Each log follows an actual database modification. That is, the database is modified immediately after every operation.

Recovery with Concurrent Transactions

When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

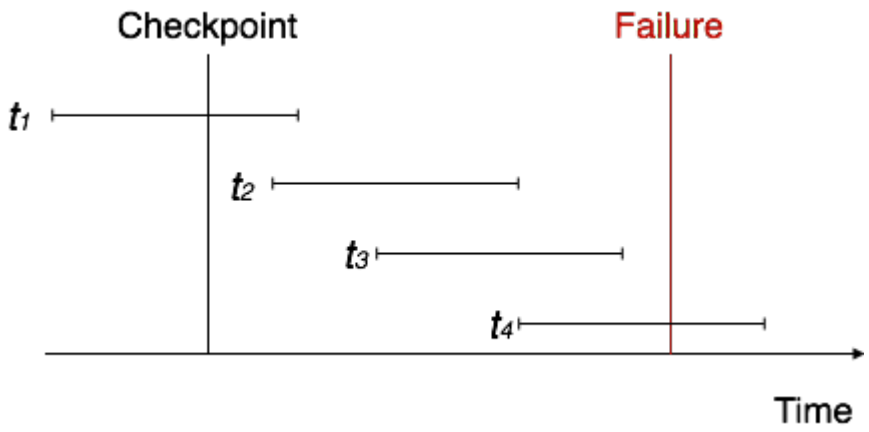
Checkpoint

Keeping and maintaining logs in real time and in real environment may fill out all the memory

space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner −



- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with $<T_n$, Start> and $<T_n$, Commit> or just $<T_n$, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with $<T_n$, Start> but no commit or abort log found, it puts the transaction in undo-list.

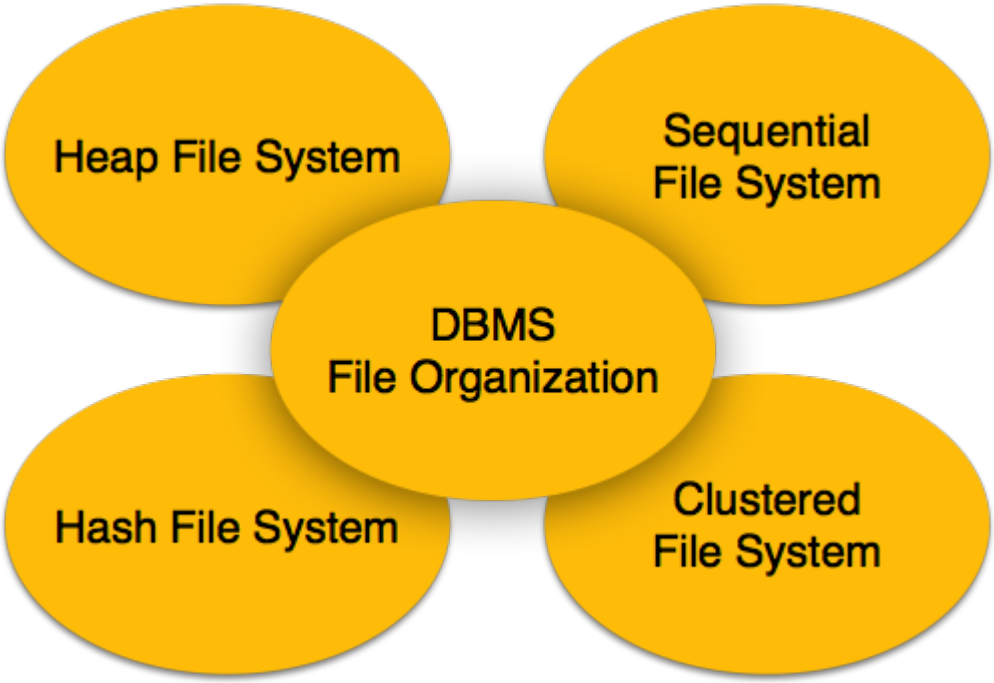
All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

DBMS - FILE STRUCTURE

Relative data and information is stored collectively in file formats. A file is a sequence of records stored in binary format. A disk drive is formatted into several blocks that can store records. File records are mapped onto those disk blocks.

File Organization

File Organization defines how file records are mapped onto disk blocks. We have four types of File Organization to organize file records −



Heap File Organization

When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records. Heap File does not support any ordering, sequencing, or indexing on its own.

Sequential File Organization

Every file record contains a data field *attribute* to uniquely identify that record. In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. Practically, it is not possible to store all the records sequentially in physical form.

Hash File Organization

Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.

Clustered File Organization

Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

File Operations

Operations on database files can be broadly classified into two categories −

- **Update Operations**

- **Retrieval Operations**

Update operations change the data values by insertion, deletion, or update. Retrieval operations, on the other hand, do not alter the data but retrieve them after optional conditional filtering. In both types of operations, selection plays a significant role. Other than creation and deletion of a file, there could be several operations, which can be done on files.

- **Open** − A file can be opened in one of the two modes, **read mode** or **write mode**. In read mode, the operating system does not allow anyone to alter data. In other words, data is read only. Files opened in read mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.

- **Locate** − Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find *seek* operation, it can be moved forward or backward.

- **Read** − By default, when files are opened in read mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.

- **Write** − User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows to do so.

- **Close** − This is the most important operation from the operating system's point of view. When a request to close a file is generated, the operating system

    - removes all the locks *ifinsharedmode*,
    - saves the data *ifaltered* to the secondary storage media, and
    - releases all the buffers and file handlers associated with the file.

The organization of data inside a file plays a major role here. The process to locate the file pointer to a desired record inside a file various based on whether the records are arranged sequentially or clustered.

Introduction to Parallel Databases

Companies need to handle huge amount of data with high data transfer rate. The client server and centralized system is not much efficient. The need to improve the efficiency gave birth to the concept of Parallel Databases.

A **parallel database** system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries. Although data may be stored in a distributed fashion, the distribution is governed solely by performance considerations. Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel. Centralized and client–server database systems are not powerful enough to handle such applications. In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially. Parallel databases can be roughly divided into two groups, the first group of architecture is the multiprocessor architecture, the alternatives of which are the following:

Shared memory architecture

Where multiple processors share the main memory (RAM) space but each processor has its own disk (HDD). If many processes run simultaneously, the speed is reduced, the same as a computer when many parallel tasks run and the computer slows down.

Shared disk architecture

Where each node has its own main memory,but all nodes share mass storage, usually a storage area network. In practice, each node usually also has multiple processors.

Shared nothing architecture

Where each node has its own mass storage as well as main memory.

Goals of Parallel Databases

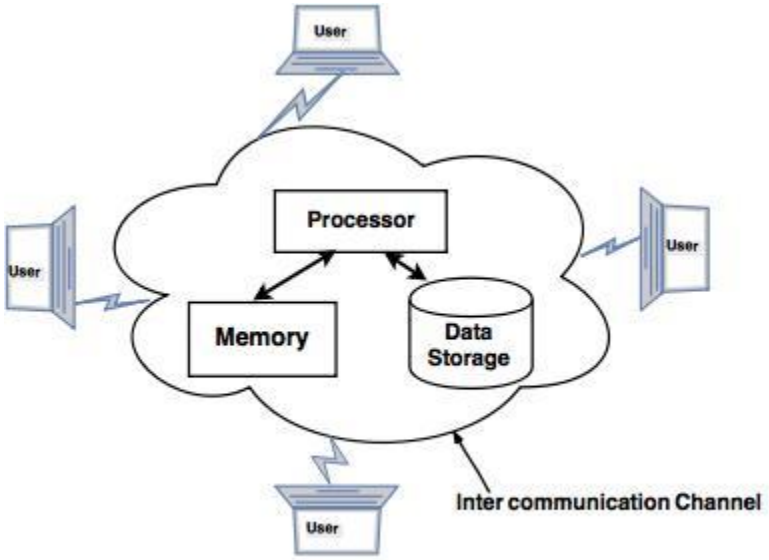
The concept of Parallel Database was built with a goal to:

Improve performance:
The performance of the system can be improved by connecting multiple CPU and disks in parallel. Many small processors can also be connected in parallel.

Improve availability of data:
Data can be copied to multiple locations to improve the availability of data.
For example: if a module contains a relation (table in database) which is unavailable then it is important to make it available from another module.

Improve reliability:
Reliability of system is improved with completeness, accuracy and availability of data.

Provide distributed access of data:
Companies having many branches in multiple cities can access data with the help of parallel database system.



Parallel database system

CLOUD DATABASE

A **cloud database** is a database that typically runs on a cloud computing platform, and access to the database is provided as-a-service.

Database services take care of scalability and high availability of the database. Database services make the underlying software-stack transparent to the user

There are two primary methods to run a database in a cloud:

Virtual machine image

Cloud platforms allow users to purchase virtual-machine instances for a limited time, and one can run a database on such virtual machines. Users can either upload their own machine image with a database installed on it, or use ready-made machine images that already include an optimized installation of a database.

Database-as-a-service (DBaaS)

With a database as a service model, application owners do not have to install and maintain the database themselves. Instead, the database service provider takes responsibility for installing and maintaining the database, and application owners are charged according to their usage of the service. This is a type of Saas - Software as a Service

Architecture and common characteristics

- Most database services offer web-based consoles, which the end user can use to provision and configure database instances.
- Database services consist of a database-manager component, which controls the underlying database instances using a service API. The service API is exposed to the end user, and permits users to perform maintenance and scaling operations on their database instances.
- Underlying software-stack stack typically includes the operating system, the database and third-party software used to manage the database. The service provider is responsible for installing, patching and updating the underlying software stack and ensuring the overall health and performance of the database.
- Scalability features differ between vendors – some offer auto-scaling, others enable the user to scale up using an API, but do not scale automatically.
- There is typically a commitment for a certain level of high availability (e.g. 99.9% or 99.99%). This is achieved by replicating data and failing instances over to other database instances.

Data model

The design and development of typical systems utilize data management and relational databases as their key building blocks. Advanced queries expressed in SQL work well with the strict relationships that are imposed on information by relational databases. However, relational database technology was not initially designed or developed for use over distributed systems. This issue has been addressed with the addition of clustering enhancements to the relational databases, although some basic tasks require complex and expensive protocols, such as with data synchronization.

Modern relational databases have shown poor performance on data-intensive systems, therefore, the idea of NoSQL has been utilized within database management systems for cloud based systems. Within NoSQL implemented storage, there are no requirements for fixed table schemas, and the use of join operations is avoided. "The NoSQL databases have proven to provide efficient horizontal scalability, good performance, and ease of assembly into cloud applications. Data models relying on simplified relay algorithms have also been employed in data-intensive cloud mapping applications unique to virtual frameworks.

It is also important to differentiate between cloud databases which are relational as opposed to non-relational or NoSQL.

SQL databases

> are one type of database which can run in the cloud, either in a virtual machine or as a service, depending on the vendor. While SQL databases are easily vertically scalable, horizontal scalability poses a challenge,that cloud database services based on SQL have started to address.

NoSQL databases

Are another type of database which can run in the cloud. NoSQL databases are built to service heavy read/write loads and can scale up and down easily, and therefore they are more natively suited to running in the cloud.: However, most contemporary applications are built around an SQL data model, so working with NoSQL databases often requires a complete rewrite of application code.

Some SQL databases have developed NoSQL capabilities including JSON, binary JSON (e.g. BSON or similar variants), and key-value store data types.

A multi-model database with relational and non-relational capabilities provides a standard SQL interface to users and applications and thus facilitates the usage of such databases for contemporary applications built around an SQL data model. Native multi-model databases support multiple data models with one core and a unified query language to access all data models.

MOBILE DATABASE

Mobile computing devices (e.g., smartphones and PDAs) store and share data over a mobile network, or a database which is actually stored by the mobile device. This could be a list of contacts, price information, distance travelled, or any other information.

Many applications require the ability to download information from an information repository and operate on this information even when out of range or disconnected. An example of this is your contacts and calendar on the phone. In this scenario, a user would require access to update information from files in the home directories on a server or customer records from a database. This type of access and work load generated by such users is different from the traditional workloads seen in client–server systems of today.

Mobile databases are not used solely for the revision of company contacts and calendars, but used in a number of industries.

Considerations

- Mobile users must be able to work without a network connection due to poor or even non-existent connections. A cache could be maintained to hold recently accessed data and transactions so that they are not lost due to connection failure. Users might not require access to truly live data, only recently modified data, and uploading of changing might be deferred until reconnected.
- Bandwidth must be conserved (a common requirement on wireless networks that charge per megabyte or data transferred).
- Mobile computing devices tend to have slower CPUs and limited battery life.
- Users with multiple devices (e.g. smartphone and tablet) need to synchronize their devices to a centralized data store. This may require application-specific automation features.

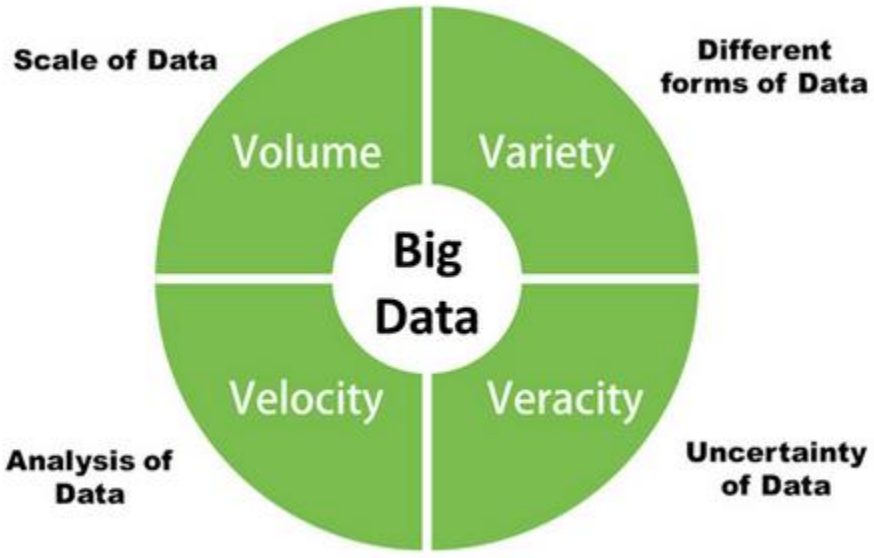
This is in database theory known as "replication", and good mobile database system should provide tools for automatic replication that takes into account that others may have modified the same data as you while you were away, and not just the last update is kept, but also supports "merge" of variants.

- Users may change location geographically and on the network. Usually dealing with this is left to the operating system, which is responsible for maintaining the wireless network connection.

BIG DATA

The term Big Data refers to all the data that is being generated across the globe at an unprecedented rate. This data could be either structured or unstructured. Today's business enterprises owe a huge part of their success to an economy that is firmly knowledge-oriented. Data drives the modern organizations of the world and hence making sense of this data and unravelling the various patterns and revealing unseen connections within the vast sea of data becomes critical and a hugely rewarding endeavour indeed. There is a need to convert Big Data into Business Intelligence that enterprises can readily deploy. Better data leads to better decision making and an improved way to strategize for organizations regardless of their size, geography, market share, customer segmentation and such other categorizations. Hadoop is the platform of choice for working with extremely large volumes of data. The most successful enterprises of tomorrow will be the ones that can make sense of all that data at extremely high volumes and speeds in order to capture newer markets and customer base.

Big Data has certain characteristics and hence is defined using 4Vs namely:



Volume: the amount of data that businesses can collect is really enormous and hence the volume of the data becomes a critical factor in Big Data analytics.

Velocity: the rate at which new data is being generated all thanks to our dependence on the internet, sensors, and machine-to-machine data is also important to parse Big Data in a timely manner.

Variety: the data that is generated is completely heterogeneous in the sense that it could be in various formats like video, text, database, numeric, and sensor data and so on and hence understanding the type of Big Data is a key factor to unlocking its value.

Veracity: knowing whether the data that is available is coming from a credible source is of utmost importance before deciphering and implementing Big Data for business needs.

SEMI-STRUCTURED DATA

Extensible Markup Language (XML) has its roots in document management, and is derived from a language for structuring large documents known as the Standard Generalized Markup Language (SGML). However, unlike SGML and HTML, XML is designed to represent data. It is particularly useful as a data format when an application must communicate with another application, or integrate information from several other applications.
The term markup refers to anything in a document that is not intended to be part of the printed output. Markup languages evolved from specifying instructions for how to print parts of the document to specifying the function of the content. Such functional markup allows:

1. The document to be formatted differently in different situations.
2. Different parts of a large document, or different pages in a large Web site, to be formatted in a uniform manner.
3. To record what each part of the text represents semantically, and correspondingly helps automate extraction of key parts of documents.

Unlike HTML, XML does not prescribe the set of tags allowed, and the set may be chosen as needed by each application. This feature is the key to XML's major role in data representation and exchange, whereas HTML is used primarily for document formatting.

XML representation of university information:

```
<university>
    <department>
```

```
        <dept name> Comp. Sci. </dept name>
        <building> Taylor </building>
        <budget> 100000 </budget>
    </department>
    <course>
        <course id> CS-101 </course id>
        <title> Intro. to Computer Science </title>
        <dept name> Comp. Sci </dept name>
        <credits> 4 </credits>
    </course>
    <instructor>
        <IID> 10101 </IID>
        <name> Srinivasan </name>
        <dept name> Comp. Sci. </dept name>
        <salary> 65000 </salary>
    </instructor>
    <teaches>
        <IID> 10101 </IID>
        <course id> CS-101 </course id>
    </teaches>
</university>
```

Compared to storage of data in a relational database, the XML representation may be inefficient, since tag names are repeated throughout the document. However, inspite of this disadvantage, an XML representation has significant advantages when it is used to exchange data between organizations, and for storing complex structured information in files:

I. Self-documenting, that is, a schema need not be consulted to understand the meaning of the text
II. Ability to recognize and ignore unexpected tags that allows the format of the data to evolve over time, without invalidating existing applications. Similarly, the ability to

have multiple occurrences of the same tag makes it easy to represent multivalued attributes.

III. Allows nested structures gathering all information related to an entity into a single nested structure, even at the cost of redundancy, is attractive when information has to be exchanged with external parties.

IV. Being widely accepted, a wide variety of tools are available to assist in its processing, including programming language APIs to create and to read XML data, browser software, and database tools.

Schema in the XML documents is optional, while such freedom may occasionally be acceptable given the self-describing nature of the data format, it is not generally useful when XML documents must be processed automatically as part of an application, or even when large amounts of related data are to be formatted in XML.

DATA MINING

Data mining is the process of extracting knowledge from data

Alternative name:

- Knowledge discovery (mining) in databases (KDD), knowledge extraction, data/pattern analysis, data archeology, data dredging, information harvesting, business intelligence, etc.

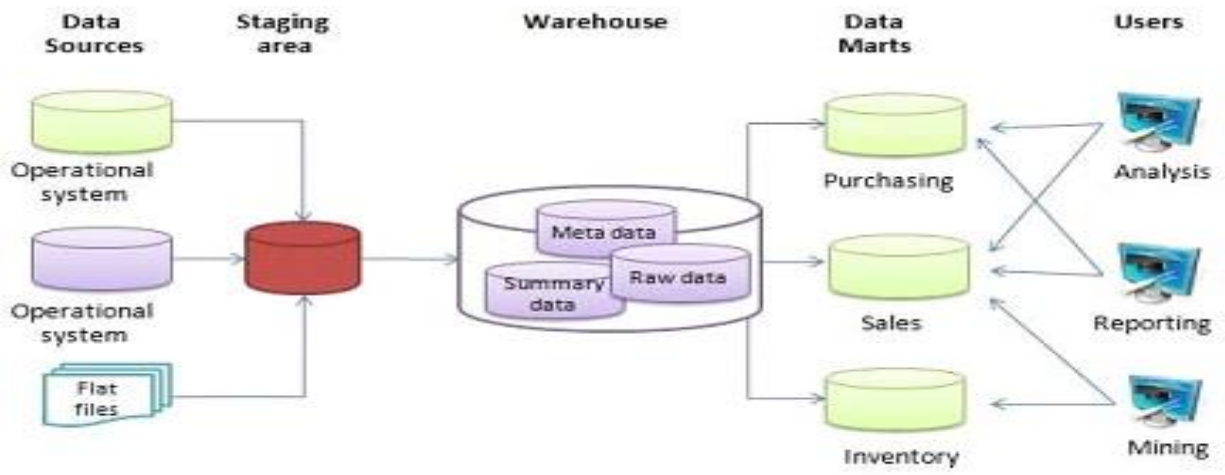
Data mining is a sub process of KDD process. Data mining plays an essential role in the knowledge discovery process

Data mining is the process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems. Data mining is an interdisciplinary subfield of computer science and statistics with an overall goal to extract information (with intelligent methods) from a data set and transform the information into a comprehensible structure for further use.[1][2][3][4] Data mining is the analysis step of the "knowledge discovery in databases" process or KDD. Aside from the raw analysis step, it also involves database and data management aspects, data pre-processing, model and inference considerations, interestingness metrics, complexity considerations, post-processing of discovered structures, visualization, and online updating.[1]

The term "data mining" is a misnomer, because the goal is the extraction of patterns and knowledge from large amounts of data, not the extraction (mining) of data itself.[6] It also is a buzzword[7] and is frequently applied to any form of large-scale data or information processing (collection, extraction, warehousing, analysis, and statistics) as well as any application of computer decision support system, including artificial intelligence (e.g., machine learning) and business intelligence. The book Data mining: Practical machine learning tools and techniques with Java[8] (which covers mostly machine learning material) was originally to be named just

Practical machine learning, and the term data mining was only added for marketing reasons. Often the more general terms (large scale) data analysis and analytics – or, when referring to actual methods, artificial intelligence and machine learning – are more appropriate.

DATA WAREHOUSING



The basic architecture of a data warehouse

In computing, a **data warehouse** (**DW** or **DWH**), also known as an **enterprise data warehouse** (**EDW**), is a system used for reporting and data analysis, and is considered a core component of business intelligence.[1] DWs are central repositories of integrated data from one or more disparate sources. They store current and historical data in one single place[2] that are used for creating analytical reports for workers throughout the enterprise.[3]

The data stored in the warehouse is uploaded from the operational systems (such as marketing or sales). The data may pass through an operational data store and may require data cleansing[2] for additional operations to ensure data quality before it is used in the DW for reporting.

Extract, transform, load (ETL) and extract, load, transform (E-LT) are the two main approaches used to build a data warehouse system.